FRIEDRICH-SCHILLER-
UNIVERSITÄT
JENA

Building High-Quality Merged Ontologies from Multiple Sources with Requirements Customization

Dissertation
zur Erlangung des akademischen Grades
Doktor-Ingenieur (Dr.-Ing.)

vorgelegt dem Rat der Fakultät für Mathematik und Informatik
der Friedrich-Schiller-Universität Jena

von M.Sc. Samira Babalou
geboren am 14.10.1987 in Karaj, Iran

Printed and/or published with the support of the German Academic Exchange Service.

Gutachter
Prof. Dr. Birgitta König-Ries
Friedrich-Schiller-Universität Jena, 07743 Jena, Thüringen, Deutschland

Prof. Dr. Adrian Paschke
Freie Universität Berlin, 10589 Berlin, Deutschland

Tag der öffentlichen Verteidigung: 19.02.2021

Declaration of Authorship

Hiermit erkläre ich,

- dass mir die Promotionsordnung der Fakultät bekannt ist,
- dass ich die Dissertation selbst angefertigt habe, keine Textabschnitte oder Ergebnisse eines Dritten oder eigenen Prüfungsarbeiten ohne Kennzeichnung übernommen und alle von mir benutzten Hilfsmittel, persönliche Mitteilungen und Quellen in meiner Arbeit angegeben habe,
- dass ich die Hilfe eines Promotionsberaters nicht in Anspruch genommen habe und dass Dritte weder unmittelbar noch mittelbar geldwerte Leistungen von mir für Arbeiten erhalten haben, die im Zusammenhang mit dem Inhalt der vorgelegten Dissertation stehen,
- dass ich die Dissertation noch nicht als Prüfungsarbeit für eine staatliche oder andere wissenschaftliche Prüfung eingereicht habe.

Bei der Auswahl und Auswertung des Materials sowie bei der Herstellung des Manuskripts haben mich folgende Personen unterstützt:

- Prof. Dr. Birgitta König-Ries

Ich habe die gleiche, eine in wesentlichen Teilen ähnliche bzw. eine andere Abhandlung bereits bei einer anderen Hochschule als Dissertation eingereicht: Ja / Nein.

Jena, den 10. August 2020

[Samira Babalou]

To my Family

Deutsche Zusammenfassung

Ontologien sind das semantische Modell von Daten im Web. In vielen Fällen decken einzelne Ontologien nur einen Teil der interessierenden Domäne ab, oder es existieren unterschiedliche Ontologien, die die Domäne unter verschiedenen Gesichtspunkten modellieren. In beiden Fällen kann durch die Vereinigung (engl. Merging) zu einem integrierten Wissensgraphen ihre Komplementarität genutzt und einheitliches Wissen über eine bestimmte Domäne erworben werden. Die Vereinigung ist wichtig, aber es gibt mehrere Herausforderungen, die mit bestehenden Ansätzen nicht zufriedenstellend gelöst werden können. Dazu gehören mangelnde Skalierbarkeit, Anpassbarkeit und Qualitätskontrolle. Diese Dissertation trägt zur Verbesesser aller drei Aspekte bei .

Bestehende Ansätze aufgrund der Verwendung einer binären Vereinigungsstrategie eher schlecht auf das Vereinigen mehrerer Ontologien skalieren. Eine Reihe von binären Vereinigungen kann schrittweise auf mehr als zwei Ontologien angewendet werden. Dieser Ansatz ist jedoch für eine große Anzahl von Ontologien nicht ausreichend skalierbar und praktikabel. Das Vereinigen mehrerer Ontologien in einem einzigen Schritt unter Verwendung einer sogenannten n-fachen Strategie wurde bisher nicht ausführlich untersucht. Es ist notwendig, eine effiziente n-fache-Technik zu entwickeln, die sich auf die Vereinigung mehrerer Ontologien skalieren lässt. Daher wollen wir untersuchen, inwieweit die n-ary-Strategie das Skalierbarkeitsproblem lösen kann. Um eine große Anzahl von Quellontologien verarbeiten zu können, haben wir uns zum Ziel gesetzt, die Zeit und die Ablaufkomplexität im Vergleich zu binären Vereinigungen zu reduzieren und gleichzeitig mindestens die gleiche Qualität des Endergebnisses zu erzielen oder sogar zu verbessern.

Insgesamt trägt diese Dissertation zu folgenden wichtigen Aspekten bei:

1. Unsere n-ary Merge-Strategie verwendet eine Reihe von Quell-Ontologien und deren Abbildungen als Eingabe und generiert eine zusammengefügte Ontologie. Anstatt vollständige Ontologien paarweise nacheinander zusammenzufügen, gruppieren wir für eine effiziente Bearbeitung Konzepte über Ontologien hinweg in Partitionen und fügen diese zuerst innerhalb und dann über die Partitionen

hinweg zusammen. Die experimentellen Tests an bekannten Datensätzen bestätigen die Machbarkeit unseres Ansatzes und zeigen seine Überlegenheit gegenüber binären Strategien hinsichtlich Ablaufkomplexität und Laufzeit.

2. Wir machen einen Schritt in Richtung parametrierbarer Zusammenführungsmethoden. Wir haben eine Reihe von GMR (Generic Merge Requirements) identifiziert, deren Erfüllung von zusammengeführten möglicherweise erwartet wird. Wir haben die Kompatibilität der GMRs mit einer graphbasierten Methode untersucht. Anhand von Anwendungsfallstudien für die vom Benutzer ausgewählten GMRs zeigen wir mögliche Obermengen kompatibler GMRs auf, die gleichzeitig erfüllt werden können.

3. Wenn mehrere Ontologien zusammengeführt werden, können Inkonsistenzen aufgrund unterschiedlicher Weltanschauungen auftreten, die in den Quellontologien codiert sind. Zu diesem Zweck schlagen wir eine neuartige, auf Subjektiver Logik basierende Methode vor, um die beim Zusammenführen von Ontologien auftretenden Inkonsistenzen zu behandeln. Wir wenden diese Logik an, um die Vertrauenswürdigkeit widersprüchlicher Axiome, die Inkonsistenzen innerhalb einer zusammengeführten Ontologie verursachen, einzustufen und abzuschätzen. In den experimentellen Tests analysieren wir die Eigenschaften der inkonsistenten zusammengeführten Ontologien und zeigen, dass die inkonsistenten zusammengeführten Ontologien, die durch unseren Ansatz repariert werden, mit konsistenten zusammengeführten Ontologien konkurrenzfähig sind, die durch menschliches Eingreifen erreicht werden.

4. Angesichts der zentralen Rolle, die die zusammengeführten Ontologien bei der Realisierung realer Anwendungen spielen, besteht ein starker Bedarf an der Entwicklung von Bewertungsmethoden, mit denen ihre Qualität gemessen werden kann. Um die Qualität der zusammengeführten Ontologien systematisch bewerten zu können, bieten wir einen umfassenden Kriterienkatalog in einem Bewertungsrahmen. Die vorgeschlagenen Kriterien decken eine Vielzahl von Merkmalen jedes einzelnen Aspekts der zusammengeführten Ontologie in strukturellen, funktionalen und benutzerfreundlichen Dimensionen ab. Wir bewerten die strukturellen Maßnahmen zusammen mit GMRs. Wir führen eine systematische Formulierung ein, um die funktionalen Maßnahmen anhand des Verwendungszwecks und der Semantik der zusammengeführten Ontologie zu bewerten und Kriterien für die Dimension der Usability bereitzustellen.

5. Der letzte Beitrag dieser Forschung ist die Entwicklung des *CoMerger*-Tools, das alle oben genannten Aspekte implementiert, auf die über eine einheitliche Schnittstelle zugegriffen werden kann.

Acknowledgements

Firstly, I would like to express my sincere gratitude to my advisor Prof. Dr. Birgitta König-Ries for her continuous support of my Ph.D. study, for her patience, and immense knowledge. She gave me a great opportunity to learn a lot from her and to believe and trust in my abilities. I would like to give special thanks to Prof. Dr. Adrian Paschke for the time and effort he devoted to review and evaluate this thesis.

My sincere thanks also go to my colleges Dr. Alsayed Algergawy, who supported and advised me in the first year of my Ph.D., and Sirko Schindler, who provided feedback on my methodology, and to all other in Heinz-Nixdorf Chair for Distributed Information Systems, which always support me in my work and my personal life. Without their precious support, it would not be possible to conduct this research.

I would like also to thank my scholarship organization, German Academic Exchange Service (DAAD), which has supported me financially for four years.

My deepest gratitude thanks goes to my family for their big love, generous support and huge encourage. Many thanks to my friend through all years in Jena for showing me the beauty of life making my mind relax during the hard. This dissertation would not have been possible without their warm love, continued patience, and endless support.

Finally, I would like to thank myself for my hard-working time and persistent to accomplish my Ph.D.

Abstract

Ontologies represent the semantic model of data on the web. For many usecases, individual ontologies cover just a part of the domain of interest or different ontologies exist that model the domain from different viewpoints. In both cases, by merging them into an integrated knowledge graph, their complementarity can be leveraged and unified knowledge of a given domain can be acquired. Merging is important, but there are several challenges associated with it that existing approaches do not solve satisfactory. These include lack of scalability, customizability, and quality control. This thesis contributes to all three.

Existing approaches scale rather poorly to the merging of multiple ontologies due to using a binary merge strategy. A series of binary merges can be applied incrementally to more than two ontologies. However, this approach is not sufficiently scalable and viable for a large number of ontologies. Merging multiple ontologies in a single step, employing what is called an *n-ary* strategy, has not been extensively studied so far. It is a necessity to develop an efficient n-ary technique that scales to merging multiple ontologies. Thus, we aim to investigate the extent to which the n-ary strategy can solve the scalability problem. To handle a large number of source ontologies, we aimed to reduce the time and operational complexity compared to the binary merges while achieving at least the same quality of the final result or even improve upon it.

As a whole, this thesis contributes to the following important aspects:

1. Our n-ary merge strategy takes as input a set of source ontologies and their mappings and generates a merged ontology. For efficient processing, rather than successively merging complete ontologies pairwise, we group related concepts across ontologies into partitions and merge first within and then across those partitions. The experimental tests on well-known datasets confirm the feasibility of our approach and demonstrate its superiority over binary strategies in terms of operational complexity and time performance.

2. We take a step towards parameterizable merge methods. We have identified a set of Generic Merge Requirements (GMRs) that merged ontologies might be expected

to meet. We have investigated and developed compatibilities of the GMRs by a graph-based method. Through use case studies, for the given user-selected GMRs, we show possible supersets of compatible GMRs that can be fulfilled simultaneously.

3. When multiple ontologies are merged, inconsistencies can occur due to different world views encoded in the source ontologies. To this end, we propose a novel Subjective Logic-based method to handling the inconsistency occurring while merging ontologies. We apply this logic to rank and estimate the trustworthiness of conflicting axioms that cause inconsistencies within a merged ontology. In the experimental tests, we analyze the characteristics of the inconsistent merged ontologies and show the inconsistent merged ontologies repaired by our approach is competitive with consistent merged ontologies achieved by human intervention.

4. Given the central role the merged ontologies play in realising real world applications, there is a strong need to establish evaluation methods that can measure their quality. Thus, to assess the quality of the merged ontologies systematically, we provide a comprehensive set of criteria in an evaluation framework. The proposed criteria cover a variety of characteristics of each individual aspect of the merged ontology in structural, functional, and usability dimensions. We evaluate the structural measures along with GMRs. We define the systematic formulation to evaluate the functional measures against the intended use and semantics of the merged ontology and provide criteria for the usability dimension.

5. The final contribution of this research is the development of the *CoMerger* tool that implements all aforementioned aspects accessible via a unified interface.

Contents

I Problem Definition **1**

1 Introduction **3**
1.1 Motivation & Problem Statements 3
1.2 Research Solution 7
1.3 Research Hypothesis 9
1.4 Research Contributions 11
1.5 Outline of the Thesis 12

2 Literature Review **15**
2.1 Existing Ontology Merging Approaches 15
2.2 Overview of Existing Data Model Merging 21
2.3 Classification of the Ontology Merge Approaches 21
2.3.1 Binary vs. N-ary Merge 21
2.3.2 One-level vs. Two-levels Merge 22
2.3.3 Symmetric vs. Asymmetric Merge 22
2.4 Survey on Generic Merge Requirements 23
2.5 Literature on Ontology Inconsistency Handling 26
2.5.1 Ontology Inconsistency Handling in the Single Development Environment 26
2.5.2 Inconsistency Handling in the Ontology Merging Domain 26
2.6 Survey on the Ontology Quality Assessment 27
2.6.1 Ontology Quality Assessment in the Single Deployment Scenario . 28
2.6.2 Quality Assessment in the Ontology Merging Context 29
2.7 Summary 30

II Approach **33**

3 Solution Overview **35**

4 Multiple Ontology Merging Method **37**
4.1 Introduction . . . 39
4.2 Preliminaries . . . 40
4.3 The Workflow of N-Ary Merge Method in *CoMerger* . . . 43
4.4 Initialization Phase . . . 43
4.5 Partitioning Phase . . . 45
4.5.1 Partitioner Goals . . . 46
4.5.2 Finding Pivot Classes $\mathcal{P}$ . . . 46
4.5.3 Partitioner: a Structure Driven Strategy . . . 47
4.5.4 Partitioning Phase Characteristics - Summary . . . 48
4.6 Combining Phase . . . 49
4.6.1 Intra-combination: Independent Merge . . . 49
4.6.2 Inter-combination: Dependent Merge . . . 50
4.6.3 Combining Phase Characteristics- Summary . . . 51
4.7 N-ary Merge Algorithm . . . 51
4.8 Example . . . 52
4.9 Summary . . . 54

5 Generic Merge Requirements **59**
5.1 Introduction . . . 61
5.2 GMRs Classification . . . 61
5.3 GMR Overview . . . 62
5.4 Proposed Approach for Checking GMR Compatibility . . . 67
5.4.1 Building GMRs Interactions Graph $\mathcal{G}$ . . . 69
5.4.2 Clique Finder . . . 75
5.5 Ranking the Compatible Sets . . . 77
5.6 Conflict Resolution . . . 79
5.6.1 Conflicts Occurring by One type Restriction . . . 80
5.6.2 Conflicts Occurring by Property Value's Constraint . . . 81
5.7 Summary . . . 83

6 Handling Inconsistencies **85**
6.1 Introduction . . . 87
6.2 Preliminaries . . . 87
6.2.1 Example: an Inconsistent Ontology . . . 90
6.2.2 Subjective Logic Theory . . . 90
6.3 Proposed Method for Inconsistency Handling by Subjective Logic . . . 94
6.3.1 Negative Observation . . . 94
6.3.2 Positive Observation . . . 95
6.3.3 Atomicity . . . 96
6.3.4 Combining Opinions . . . 97
6.3.5 Applying Conditional Opinions to Reflect the Dependencies . . . 97
6.4 Inconsistency Handling Workflow . . . 99
6.5 Algorithm . . . 100
6.6 Repair plan . . . 100
6.7 Example: Applying Our Method on an Inconsistent Ontology . . . 101

6.8 Summary . . . 103

7 Quality Assessment for the Merged Ontology 105
7.1 Introduction . . . 107
7.2 Preliminaries & Background . . . 107
7.2.1 Ontology Ranking vs. Ontology Quality Evaluation . . . 108
7.2.2 Insufficiency of Global Scoring . . . 108
7.2.3 Evaluation Domains . . . 109
7.2.4 A Customizable Evaluation . . . 109
7.3 Evaluation Standards . . . 110
7.4 Proposed Quality Indicators for the Evaluation of Merged Ontology . . . 114
7.4.1 Extending Ontology Evaluation Frameworks . . . 114
7.4.2 Quality Evaluation Function . . . 117
7.4.3 Associated Quality Indicators of Evaluation Dimensions . . . 119
7.5 Ontology Merging Quality Assessment Workflow . . . 125
7.6 Summary . . . 126

III Evaluation 129

8 Experimental Evaluation 131
8.1 Datasets . . . 131
8.2 Implementation . . . 133
8.3 Overview of Experimental Tests . . . 133

9 *CoMerger*: Proposed Tool 135
9.1 *CoMerger* Overview . . . 135
9.2 *CoMerger* Architecture . . . 136
9.3 *CoMerger* Component . . . 137
9.3.1 GMRs Compatibility Checker . . . 137
9.3.2 Multiple Ontologies Merger . . . 137
9.3.3 Merged Ontology Evaluator . . . 138
9.3.4 Consistency Checker . . . 138
9.4 *CoMerger* GUI . . . 138
9.5 Summary . . . 139

10 Experimental Tests on the N-ary Merge Method 145
10.1 Test Setting . . . 147
10.1.1 Adjusting Parameters . . . 147
10.1.2 Adjusting Binary Methods . . . 148
10.1.3 Building Different Versions of Merge . . . 148
10.1.4 Adjusting Refinements . . . 148
10.2 Experimental Results . . . 149
10.2.1 Characteristics of the N-Ary Merged Result . . . 149
10.2.2 Answering Competency Questions . . . 153
10.2.3 Binary versus N-ary . . . 156
10.3 Summary . . . 158

11 Experimental Tests on GMRs **165**
11.1 Use Case Study on Compatibility Checker . . . 167
11.1.1 First Use Case . . . 167
11.1.2 Second Use Case . . . 167
11.1.3 Third Use Case . . . 168
11.2 Use Case Study on Conflict Resolution . . . 169
11.3 Summary . . . 170

12 Experimental Tests on Inconsistency Handling of Merged Ontologies **173**
12.1 Characteristics of Inconsistent Ontologies . . . 175
12.2 Answering Competency Questions . . . 178
12.3 Scalability . . . 180
12.4 Summary . . . 181

13 Experimental Tests on the Quality Assessment of the Merged Ontology **183**
13.1 Quality Evaluation of the Structural Dimension . . . 186
13.2 Quality Evaluation of the Functional Dimension . . . 187
13.2.1 Quality Assessment of Intended Use with Competency Questions Testing . . . 187
13.2.2 Quality Assessment of Intended Semantics with Query Testing . . . 189
13.3 Quality Evaluation of the Usability Dimension . . . 191
13.4 Time Performance . . . 191
13.5 Overall Result Demonstrations . . . 193
13.6 Total analyzing . . . 193
13.7 Analyzing the Fulfilment of the Principles of Evaluation Standards . . . 196
13.8 Summary . . . 197

IV Conclusion **199**

14 Summary **201**

15 Future Work **205**

V Appendix **207**

A Competency Questions on Conference domain **209**

B GMR Implementation **213**

C User Help **225**
C.1 Merging Ontologies . . . 226
C.2 Quality Assessment . . . 228
C.3 Consistency Checker . . . 231
C.4 Compatibility Checker . . . 232
C.5 SPARQL Query Endpoint . . . 234

Bibliography **237**

List of Figures

1.1 An example query on the merged ontology 5
1.2 Five sample source ontologies. 6
1.3 The merged ontologies in each step of the binary merge. 6
1.4 Two sample consistent ontologies and an inconsistent merged ontology. . 8
1.5 The merged ontology for the given source ontologies. 9

3.1 Solution overview. 36

4.1 Example of overlapping bio-ontologies from BioPortal. 40
4.2 *CoMerger* architecture. 43
4.3 Taxonomy and non-taxonomy relations for a sample class. 47
4.4 Three sample source ontologies. 55
4.5 The initial merge model $\mathcal{I}_M$ for the given source ontologies. 56
4.6 Generating two blocks for the given source ontologies. 57
4.7 Augmenting the blocks with non-taxonomy relations. 57

5.1 Generic Merge Requirements (GMRs) classification. 62
5.2 Two sample source ontologies. 63
5.3 Two different merged ontologies for the given source ontologies. 64
5.4 The workflow of GMRs' compatibility checker. 68
5.5 Compatibility along with four cases. 73
5.6 Sorted GMRs based on their compatibility degree. 76
5.7 GMRs interaction graph with samples cliques. 77
5.8 All maximum compatible sets for the user-selected GMRs {*R7*, *R9*, *R10*, *R16*}. 80
5.9 A conflicting merged ontology and the repaired one. 80
5.10 The attributed Restriction Graph $\mathcal{RG}$ for six OWL Restriction types. 82

6.1 Examples of various inconsistency and incoherence [FHP+06]. 89
6.2 Example of an inconsistent ontology with its statistics. 91
6.3 Opinion triangle with example opinion [Jøs16]. 93

6.4 Our workflow to handle inconsistency of the merged ontology. 100
6.5 Applying Subjective Logic on the inconsistent merged ontology. 103

7.1 The relationship between an ontology and a conceptualization. 123
7.2 Workflow of ontology merging quality assessment. 126

9.1 *CoMerger* architecture. 136
9.2 *CoMerger* components. 137
9.3 GUI of adjusting the desired GMRs. 140
9.4 Ontology merging GUI. 141
9.5 The GUI for customizable evaluation of the merged ontology. 142
9.6 The result of the consistency test. 143

10.1 Effect of taxonomic weight and non-taxonomic weight. 147
10.2 Class, property, and instance coverage with the number of unpreserved structure coverage of n-ary merge. 151
10.3 Number of local and global refinements, oneness, unconnected classes and cycle of the n-ary merge. 151
10.4 Comparing translated axioms with corresponding entities. 152
10.5 Number of blocks versus class overlap, max cardinality, and distributed axioms. 152
10.6 Comparing distributed axioms. 152
10.7 Runtime performance of the merge method. 156

12.1 Competency Question-based experimental tests. 179
12.2 The scalability test of processing the unsustainable root concepts. 179
12.3 The scalability test of processing all unsatisfiable concepts. 179

13.1 The GUI for structural quality evaluation. 188
13.2 Functional quality's evaluation via Competency Questions. 190
13.3 Functional measure's evaluation via queries. 191
13.4 Runtime performance within the evaluation framework 192
13.5 Overall result's view within the evaluation framework. 192
13.6 Total Analyzing of the merged ontologies within the evaluation framework. 194
13.7 Query endpoint GUI. 195

B.1 *R1*- Repair solution. 214
B.2 *R2*- Repair solution. 215
B.3 *R3*- Repair solution. 215
B.4 *R4*- Repair solution. 216
B.5 *R5*- Repair solution. 216
B.6 *R6*- Repair solution. 217
B.7 *R7*- Repair solution. 217
B.8 *R8*- Repair solution. 217
B.9 *R9*- Repair solution. 218
B.10 *R10*- Repair solution. 218
B.11 *R11*- Repair solution. 218
B.12 *R12*- Repair solution. 219

B.13 *R13*- Repair solution. 219
B.14 *R14*- Repair solution. 220
B.15 *R15*- Repair solution. 220
B.16 *R16*- Repair solution. 221
B.17 *R17*- Repair solution. 221
B.18 *R18*- Repair solution. 222
B.19 *R19*- Repair solution. 222
B.20 *R20*- Repair solution. 223

C.1 Merger GUI. 226
C.2 Merging Result GUI. 227
C.3 Setting of the evaluation criteria. 228
C.4 Evaluation results. 229
C.5 Evaluator GUI. 230
C.6 Parameter setting of consistency test. 231
C.7 Consistency test result. 231
C.8 Compatibility checker GUI. 232
C.9 Result of compatibility checker. 232
C.10 Generic Merge Requirements (GMR)s information page 233
C.11 Running a single query. 234
C.12 Running different queries. 235

List of Tables

1.1 Number of operations for merging sample source ontologies. 6

2.1 Generic Merge Requirements (GMRs). 25
2.2 Summary of existing ontology merging methods: Merge strategy (binary or n-ary); Merge type (one-level or two-levels); Merge nature (symmetric or asymmetric); Fulfilled GMRs; Inconsistency handling (✓) or not (×), $\times^*$ shows the approach handles other types of inconsistencies rather than classical ones; Evaluation technique of the merged result. 32

4.1 The used notations, symbols, and nomenclatures in Chapter 4. 38
4.2 Corresponding pairs between the source ontologies given in Figure 4.4. . . 54
4.3 A map model $\mathbb{M}$ and its elements with their reputation degree. 54

5.1 The used notations, symbols, and nomenclature in Chapter 5. 60
5.2 Corresponding pairs between the source ontologies given in Figure 5.2. . . 62
5.3 Scope of changes by applying GMRs in the merged ontology. 70
5.4 The scopes and operations of each GMR. The symbol * indicates an alternative solution. 72
5.5 Compatibility interaction between GMRs with separated sub-scopes. . . . 74
5.6 Compatibility interaction between GMRs with the intersection of sub-scopes. 75
5.7 Conflicts between six OWL restrictions. 84

6.1 The used notations, symbols, and nomenclature in Chapter 6. 86
6.2 Example opinions for x_1 from four source ontologies $\mathcal{O}_1$-$\mathcal{O}_4$. 102

7.1 The used notations, symbols, and nomenclature in Chapter 7. 106
7.2 Variables of the quality evaluation function for each quality dimension. . . 118
7.3 Dimension, aspect, indicator, and type of the ontology merging evaluation task. 127

8.1 Dataset statistics. 132

10.1 The used notations, symbols, and nomenclatures in Chapter 10. 146
10.2 Number of taxonomic and non-taxonomic relations for the datasets. 147
10.3 The settings for generating twelve variants of the merged ontologies. . . . 149
10.4 Answering CQs on the different versions of merged ontologies. 154
10.5 Comparing n-ary (N), balanced (B), and ladder (L) merge strategies. . . . 156
10.6 Characteristics of the n-ary merged result- OAEI dataset. 160
10.7 Characteristics of the n-ary merged result- BioPortal dataset. 161
10.8 Comparing n-ary (V_4), balanced (V_7), and ladder(V_{10}) merge strategies. . . 162
10.9 Comparing n-ary(V_5), balanced (V_8), and ladder (V_{11}) merge strategies. . . 163
10.10Comparing n-ary(V_6), balanced (V_9), and ladder (V_{12}) merge strategies. . . 164

11.1 The used notations, symbols, and nomenclature in Chapter 11. 166
11.2 All possible compatible maximum sets for user-selected GMRs $\mathcal{U} = \{R2, R3, R8, R16\}$. 168
11.3 All possible compatible maximum sets for user-selected GMRs $\mathcal{U} = \{R3, R6, R13\}$. 169
11.4 All possible compatible maximum sets for user-selected GMRs $\mathcal{U} = \{R3, R6, R13\}$. 170
11.5 Three sample merged ontologies with their OWL restriction. 170
11.6 CQs answers on the conflicted and revised merged ontology. 171

12.1 The used notations, symbols, and nomenclature in Chapter 12. 174
12.2 The characterization of the consistency test. 176

13.1 The used notations, symbols, and nomenclature in Chapter 13. 185
13.2 Occurrence of unsatisfied GMRs. 186
13.3 True or false positive and negative responses. 189
13.4 Evaluating the usability quality of the merged ontologies. 192

A.1 The Competency Questions for the conference domain of the OAEI benchmark. 210
A.2 The Competency Questions (CQs) used in conflict resolution test. 211

B.1 The used notations and symbols in Appendix B. 213

List of Symbols and Nomenclature

GMR	General Merge Requirement
R1-R20	individual GMRs
CQ	Competency Questions
N	n-ary merge strategy
B	balanced merge strategy
L	ladder merge strategy
$\mathcal{O}$	an ontology
e	an entity of an ontology
$\mathcal{E}$	the entities of an ontology
C	a set of classes in an ontology
c	a class in an ontology
P	a set of properties of an ontology
p	a property of an ontology
I	an individual of an ontology
c'	an integrated class
p_i	a property of an ontology
I_i	an individual of an ontology
$Sig(\mathcal{O})$	a signature of an ontology
$\mathcal{O}_S$	a set of source ontologies
$\mathcal{O}_i$	one of the source ontologies
$\mathcal{O}_M$	a merged ontology
$\mathcal{O}'_M$	an inconsistent merged ontology
$\mathcal{M}$	a mapping set between the source ontologies
$\mathcal{M}'$	an imperfect mapping set between the source ontologies
n	number of source ontologies
rel	a relation between two entities
cs	a corresponding set between the source ontologies
CS	a set of corresponding between the source ontologies
cs_j^C	a corresponding class

cs_j^P	a corresponding property
$\mathbb{M}$	a map model including a group of correspondences over multiple ontologies
$\equiv$	a correspond operator between entities
$Card(cs)$	a cardinality value of a corresponding set
c_t	a class in the merged ontology
$Conn(c_t)$	number of connections of a class
$\mathcal{L}$	a block
$\mathcal{CL}$	a set of blocks
k	number of blocks
$\mathcal{I}_M$	the initial merged model
$\mathcal{P}$	a set of pivot classes
$reputation(c_t)$	a reputation degree of a class
$taxo\_rel(c_t)$	taxonomy relation of a class
$non\_taxo\_rel(c_t)$	non-taxonomy relation of a class
w_t	weight degree of taxonomy relation
w_{nt}	weight degree of non-taxonomy relation
$inter\_rel(\mathcal{L}_i, \mathcal{L}_j)$	inter-relatedness degree of two blocks
$dist_{axiom}$	a set of distributed axioms between two blocks
str	degree of the structure preservation
on	oneness- properties that have multiple domains or ranges
C_u	unconnected classes in an ontology
cyc	cycles in the class hierarchy
R_G	Global refinements on the merged ontology
R_L	local refinements on the blocks
Cor	all corresponding entities
$Card$	the max cardinality of a corresponding set
n	number of source ontologies
k	number of blocks
ds	distributed axioms between two blocks
tr	translated axioms
ov	amount of overlap between source ontologies
$\|Mer.\|$	number of merge operations
V_1-V_{12}	twelve versions of the merged ontologies
d_1-d_{12}	twelve tested datasets
$\sqsubseteq$	subClassOf relation between two classes
$\mathcal{U}$	a set of user-selected GMRs
$\mathcal{U}^C$	a compatible subset of $\mathcal{U}$
$\mathcal{U}^{EC}$	an extra compatible set of GMRs related to $\mathcal{U}$
$\mathcal{G}$	GMRs interactions graph
V	a set of vertices in the graph
E	a set of edges of in the graph
$\mathcal{K}$	number of vertices in the clique
$\mathcal{K}^C$-*Clique*	compatible clique with $\mathcal{K}$ vertices

$\mathcal{K}^C$-*max-Clique*	compatible clique with maximum $\mathcal{K}$ vertices
$\mathcal{RS}$	a set of compatible sets with $\mathcal{U}$
rs	a compatible set with $\mathcal{U}$
l	number of rs in $\mathcal{RS}$
S_i	scope of changes by a GMR on $\mathcal{O}_M$
$\mu(R_j)$	a set of axioms getting effect by applying $R_j \in$ GMRs on $\mathcal{O}_M$
$\parallel$	compatibility between two GMRs
$\nparallel$	incompatibility between two GMRs
f_d	compatibility degree of a GMR
$\lvert rs_z \rvert$	the number of GMRs in the compatible set rs_z
$\lvert \mathcal{U} \rvert$	the number of GMRs in $\mathcal{U}$
$\lvert \mathcal{U} \cap rs_z \rvert$	the number of GMRs that contain in both rs_z and $\mathcal{U}$
$Score_i(rs_z)$	scoring rs_z
$Total\_Score(rs_z)$	total scoring of ranking the rs_z
$\Psi(\mathcal{U})$	the number of GMRs' aspect in $\mathcal{U}$
$\Psi(rs_z)$	the number of GMRs' aspect in rs_z
$\Psi(\mathcal{U} \cap rs_z)$	the number of common aspects in both rs_z and $\mathcal{U}$
$\lvert GMRs_{Aspect} \rvert$	the total number of aspects of GMRs
$\mathcal{SH}$	Subsumption Hierarchy
$depth(v_i)$	deep of datatype v_i on $\mathcal{SH}$
$\mathcal{RG}$	attributed Restriction Graph for detecting and solving OWL restriction
cases $\mathbb{A}$-$\mathbb{N}$	different solution for restriction conflicts
$\mathcal{P}$	a proposition
x	an axiom
X	a set of axioms
$\overline{X}$	the trustworthiness of axiom sets
w	an opinion
$w_x^{\mathcal{O}_i}$	an opinion of axiom x from $\mathcal{O}_i$
$b_x^{\mathcal{O}_i}$	a belief of axiom x from $\mathcal{O}_i$
$d_x^{\mathcal{O}_i}$	disbelief of axiom x from $\mathcal{O}_i$
$u_x^{\mathcal{O}_i}$	an uncertainty of axiom x from $\mathcal{O}_i$
$a_x^{\mathcal{O}_i}$	the atomicity of axiom x from $\mathcal{O}_i$
r	number of positive observations
$r_x^{\mathcal{O}_i}$	number of positive observations about x by $\mathcal{O}_i$
s	number of negative observations
$s_x^{\mathcal{O}_i}$	number of negative observations about x by $\mathcal{O}_i$
A	an agent
$\mathcal{C}_{un}$	a set of unsatisfiable concept in $\mathcal{O}'_M$
$Root_{\mathcal{C}_{un}}$	a set of root unsatisfiable concepts in $\mathcal{O}'_M$
$\mathcal{J}$	a set of justifications
$\mathcal{J}_d$	a justification
$\models$	entail

$\nvDash$	not entail
$\subseteq$	subset relationship
$\subsetneq$	proper (or strict) subset relationship
$\Psi_{x_j}(\mathcal{J})$	axiom frequency for x_j in $\mathcal{J}$
$\Psi_{\mathcal{O}_j}(\mathcal{J})$	number of conflicting axiom sets belong to $\mathcal{O}_i$
$\Gamma(\mathcal{O}_i)$	total number of axioms in $\mathcal{O}_i$
$\Gamma_{x_j}(\mathcal{O}_i)$	number of axioms in $\mathcal{O}_i$ containing elements of x_j
$f_{x_j}(\mathcal{O}_i)$	the fraction of axioms in $\mathcal{O}_i$ containing elements of x_j
α	the provenance of the axioms
β	the provenance of the elements of the axioms
$\mathcal{O}_k(L)$	ontology using logical language L under commitment k
$E_{\mathcal{O}_M}$	quality evaluation function
D	an evaluation dimension
S	a set of inputs for the evaluation function
$\dot{c}$	coefficient of measurement error
mp	a measurement procedure of evaluating quality indicators
m	the output (numerical value) of $E_{\mathcal{O}_M}$
L	logical language
$\mathbb{C}$	conceptualization
$I_k(L)$	intended model
k	commitment to certain I for L
$\triangle$	a set of relevant entities
W	a set of possible worlds
R	a set of intensional relations
TP	True Positive
FP	False Positive
FN	False Negative
TN	True Negative
$n_{TP}^{\mathcal{O}_k(L)}$	number of TP of $\mathcal{O}_k(L)$
$n_{FP}^{\mathcal{O}_k(L)}$	number of FP of $\mathcal{O}_k(L)$
$n_{FN}^{\mathcal{O}_k(L)}$	number of FN of $\mathcal{O}_k(L)$
$\mathbb{P}$	Precision
$\mathbb{R}$	Recall
$\Sigma\, R$	total number of unsatisfied GMRs
$\Sigma\, e$	total number of anomalous entities
$P1$-$P22$	principles of the evaluation standards

Part I

Problem Definition

1 Introduction

This chapter starts with the motivation and problem statements of the thesis in Section 1.1. Section 1.2 briefly discusses our proposed solution. Section 1.3 presents the research hypothesis. The research contributions are explained in Section 1.4. The chapter is concluded with an outline of the structure of the thesis in Section 1.5.

1.1 Motivation & Problem Statements

Ontologies are formal, explicit descriptions of a given domain [Gru+93]. They represent the semantic model of data on the Semantic Web and allow capturing domain knowledge. Oftentimes, more than one ontology is created for concepts of the same domain. These ontologies either cover a part of the given domain or model the domain from different points of view. In this fashion, multiple heterogeneous ontologies are independently developed, where each one covers particular aspects of a domain of discourse but overlaps to a certain degree with others. To support the interoperability between these ontologies, they themselves need to be integrated. By merging them into an integrated knowledge model, their complementarity can be leveraged, and unified, more comprehensive knowledge of the domain can be acquired.

To merge the ontologies, first, the correspondences between them should be recognized via the ontology matching process. Then, each group of corresponding entities is combined into one single entity, and their relations are reconstructed to generate the merged ontology. There have been considerable works on ontology matching as an independent problem (see [RB01; KS03] for some surveys). Due to the successful development of ontology matching systems (cf. the result of OAEI[1] (Ontology Alignment Evaluation Initiative) [AFF+19]), we assume that the corresponding entities between the ontologies are given. In this view, merging the ontologies is a process of creating a unified merged ontology from a set of source ontologies with a set of correspondence pairs extracted from a given mapping [PB03].

[1] http://oaei.ontologymatching.org/

The merge process plays an important role in multiple different aspects of the Semantic Web, such as ontology reusing [CR16], knowledge discovery [FFKJ19], and query processing [LBBH15] to reduce development efforts, cost, and time. Moreover, merging ontologies is becoming increasingly important for applications from a wide variety of domains ranging from biomedicine [FFKJ19] and food production [Doo+18] to social networks [PC19] and cultural heritage [ZPVOS18], to name just a few.

Motivation of Merging Ontologies. To demonstrate an application of merging ontologies, we recall a real world use-case taken from [FFKJ19] in the domain of medical imaging, where the associated knowledge is expressed and shared across different ontologies. Clinical imaging plays a central role in medical diagnosis and treatment. To improve the ability to automate reasoning about diseases and their manifestations in medical imaging examinations, the authors sought to map Radiology Gamuts Ontology (RGO) [BLKJ14] concepts to ontologies that organize and characterize human diseases (Disease Ontology (DO) [SAN+11]) and the manifestations of those diseases (Human Phenotype Ontology (HPO) [RKB+08]). Integration of RGO's causal knowledge expressed direct causal relationships between DO diseases and HPO phenotypic abnormalities and allowed to formulate queries about causal relations using the abstraction properties in those ontologies. Thus, one can formulate queries of varying levels of abstraction within their hierarchies like "Which musculoskeletal system disease may cause an abnormality of the digestive system?" (see Figure 1.1). As a result, RGO concepts are categorized within the hierarchies of disease and phenotypic abnormalities in order to enable to posit axioms that link DO and HPO entities at multiple levels in their hierarchies. This information can support clinical diagnosis, data mining, and knowledge discovery.

Most existing ontology merging approaches such as [PK19; PC19; Fah17; ZRL17; MTFH14; RR14; JERS+11; JRGHB09] are limited to merging two ontologies at a time, due to using a *binary* merge strategy. However, there is an increasing demand to interoperate with more than two ontologies toward acquiring the desired knowledge for researchers in real-world applications of the Semantic Web. In principle, a series of binary merges can be applied incrementally to more than two ontologies, thus merging more than two ontologies. However, this approach is not sufficiently scalable and viable when the number of ontologies to merge grows [Rah16]. Nevertheless, merging n ontologies ($n > 2$) in a single step, employing what is called an *n-ary* strategy, has not been extensively studied so far. This may be due to the much more complex search space, and it still remains one of the key challenges in the future research agenda. It is a necessity to investigate whether the n-ary strategy can solve the scalability problem. Thus, we aim to develop an efficient n-ary technique that scales to merging multiple ontologies.

Motivation of performing the n-ary merge. Let us demonstrate our motivation for performing the n-ary merge with an example and highlight the difference between the binary and n-ary approaches. Consider the five source ontologies in Figure 1.2. To estimate the merge effort, we measure three operations during merging: combining the corresponding entities into an integrated entity $|combine|$; reconstructing the relationship $|reconst|$; and output generation $|output|$. These measurements are not exact efforts of the computational complexity of the merge process. However, we believe they can provide good ground for comparing binary and n-ary strategies. In

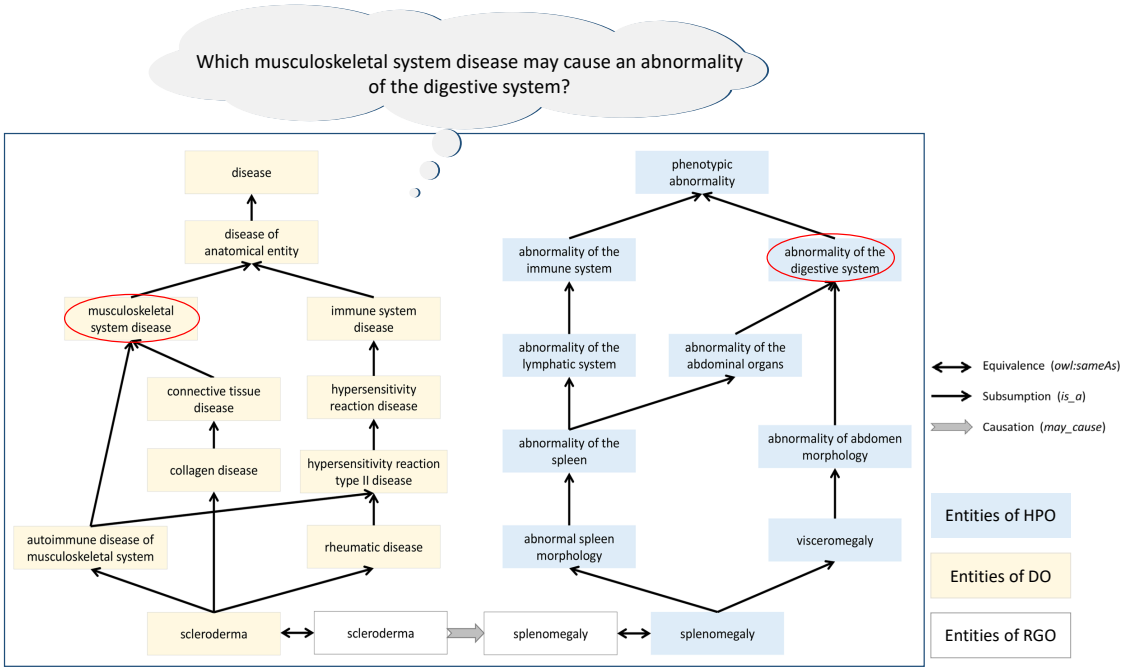


FIGURE 1.1: An example query on the merged ontology of RGO, DO, and HPO from [FFKJ19].

the n-ary approach, 6 combination operations (for 6 sets of corresponding entities) and 28 reconstruction operations are carried out. This approach needs only one output generation (see Table 1.1, second row). In the binary-ladder strategy [BLN86], in the first step, $\mathcal{O}_1$ and $\mathcal{O}_2$ are combined into an intermediate merged ontology $\mathcal{O}_{12}$. Then, $\mathcal{O}_{12}$ is merged with $\mathcal{O}_3$ and so on. All intermediate ontologies and the final merged ontology are shown in Figure 1.3, and the required operations are presented in Table 1.1. The n-ary merged ontology has the same structure as the final merged ontology of the binary-ladder merge for the given source ontologies. As a total, the binary merge approach needs 10 combinations, while the n-ary method needs 6, which shows 40% improvement in our example. The number of reconstruction operations in binary is 28, while n-ary needs 32 operations, which indicates 12.5% improvement. The n-ary method needs one time output generation, while the binary approach needs 4 times. While these numbers are specific to our example, the general pattern will be the same for other examples. This example demonstrates the n-ary approach promises to outscale binary methods. The achieved improvements are significant compared to binary approaches, especially when dealing with a large number of ontologies and processing large-scale ontologies.

In the literature, processing multiple ontologies is considered a challenging task. For instance, in the multiple ontologies matching scenario in [HCZQ11], to match 4000 web-extracted ontologies on six computers using a pairwise strategy took about one year, which indicates the insufficient scalability of pairwise strategies in practice. As a further example of multiple ontologies merging, the integration process in the biomedical ontology UMLS Metathesaurus [Bod04] was highly complex and involved

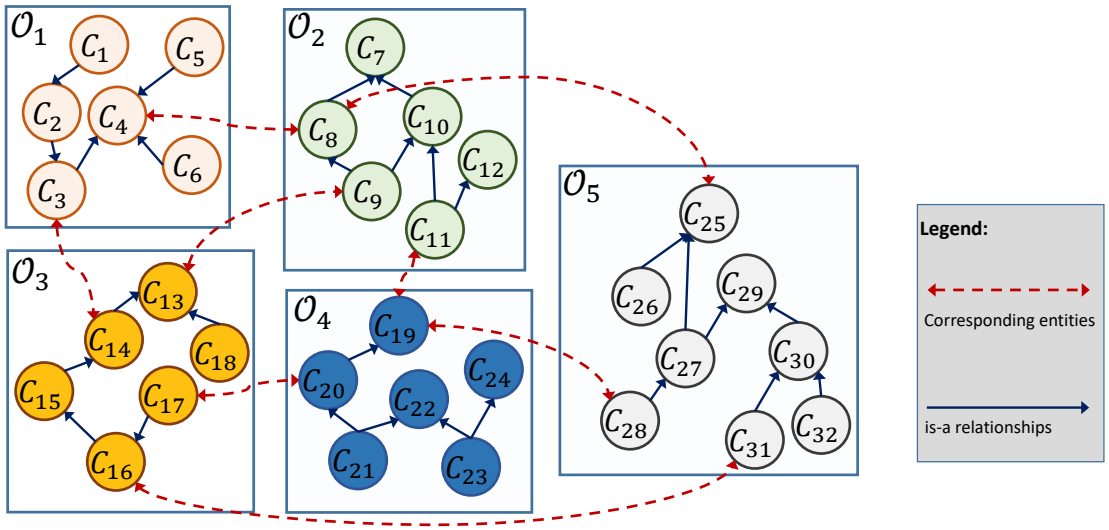


FIGURE 1.2: Five sample source ontologies with their correspondences depicted by dashed lines.

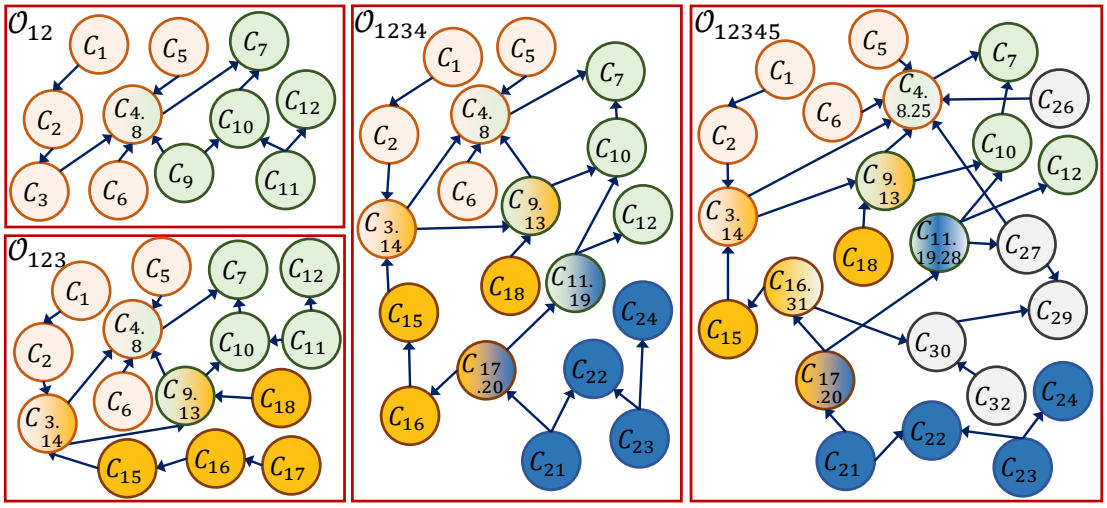


FIGURE 1.3: The merged ontologies in each step of the binary merge for the source ontologies from Figure 1.2.

TABLE 1.1: Number of operations for merging five sample source ontologies of Figure 1.2.

	Step	Source ontologies	$\lvert combine \rvert$	$\lvert reconst \rvert$	$\lvert output \rvert$
N-ary	1	$\mathcal{O}_1$ & $\mathcal{O}_2$ & $\mathcal{O}_3$ & $\mathcal{O}_4$ & $\mathcal{O}_5$	6	28	1
Binary	1	$\mathcal{O}_1$ & $\mathcal{O}_2$	1	5	1
	2	$\mathcal{O}_{12}$ & $\mathcal{O}_3$	2	8	1
	3	$\mathcal{O}_{123}$ & $\mathcal{O}_4$	4	6	1
	4	$\mathcal{O}_{1234}$ & $\mathcal{O}_5$	3	13	1
	Total | 4	$\mathcal{O}_{12345}$	10	32	4

a significant effort by domain experts.

1.2 Research Solution

To handle a large number of source ontologies, we aim to reduce the time and operational complexity while achieving at least the same quality of the final result or even improve upon it. For efficiently applying the n-ary method on merging multiple ontologies, we utilize a partitioning-based method inspired by ontology matching systems [HQC08; ADMR05; JRASC18]. The idea is to perform partitioning in such a way that every partition of one schema has to be matched with only a subset of the partitions (ideally, only with one partition) of the other schema. This results in a significant reduction of the search space and thus improved efficiency. Furthermore, matching smaller partitions reduces the memory requirements compared to matching the full schemas. Following that, in our n-ary method, *CoMerger*, we develop an efficient merging technique that scales to many ontologies. We show that by using a partitioning-based method, we can reduce the complexity of the search space[2]. Our method takes as input a set of source ontologies alongside the respective mappings and generates a merged ontology. At first, the n source ontologies are partitioned into k blocks ($k << n$). After that, the blocks are individually merged and refined. Finally, they are combined to produce the merged ontology followed by a global refinement. We provide experimental tests for merging a variety of ontologies showing the effectiveness of our approach over binary approaches.

With developing *CoMerger*, we focus on three more important aspects:

Parameterizable Merge Method. One aspect that various approaches to ontology merging differ in is the set of criteria they aim to fulfill, i.e., the requirements that they expect the merged ontology to meet, such as preserving all classes contained from the source ontologies in the result (cf. [MFRW00]) or avoiding class redundancy (cf. [TBL08]). Thus, given a comprehensive set of merge criteria, here called Generic Merge Requirements (GMR)s, a flexible and parameterizable merge approach can be achieved. Users can actively choose which requirements are important to them, instead of allowing only a very indirect choice by picking a merge system that uses their preferred set of criteria. Unfortunately, not all GMRs are compatible. For instance, one may want to preserve all properties contained in the original ontology in the merged ontology. On the other hand, one could wish to avoid cycles. Likely, these goals conflict. Thus, towards the customization of the merge process, once a user has chosen a set of important GMRs, a system is needed to check their compatibility and suggest a maximum subset of requirements that can be met simultaneously.

Inconsistency Handling of Merged Ontology. Given consistent source ontologies, the resulting merged ontology may be inconsistent due to differing world views encoded into the source ontologies. An inconsistent ontology by virtue of what has been stated in the ontology, cannot have any models and entails everything [BCM+03]. The inconsistencies need to be repaired if one wants to make use of the merged ontology.

[2]In our context, the search space is the set of entities and their relations that have to be processed for a specific merge step.

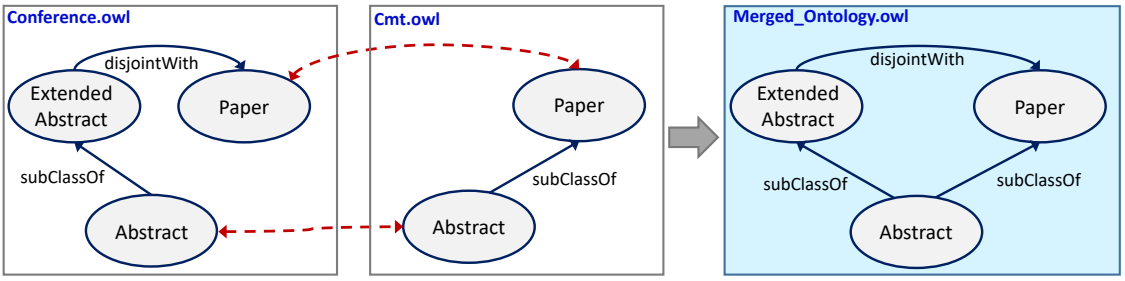


FIGURE 1.4: Excerpt of two sample ontologies that are consistent themselves, but the merged ontology, including all of these axioms, is inconsistent.

This resolution is, however, a challenging problem. Let us illustrate this issue with an example of how the merged ontology can become inconsistent relatively easily. Consider two fragments of sample ontologies of the OAEI conference benchmark[3] in Figure 1.4. The source ontologies themselves are consistent. However, when the corresponding classes are combined in the merged ontology, this results in an ontology that is inconsistent. The merged ontology, including simultaneously the axioms $\{(\texttt{Abstract} \sqsubseteq \texttt{Paper}), (\texttt{Abstract} \sqsubseteq \texttt{ExtendedAbstract}), (\texttt{ExtendedAbstract} \sqsubseteq \neg\texttt{Paper})\}$, is inconsistent. In order to turn the merged ontology into a consistent one, one or several of these axioms need to be removed or altered. Thus, a method is required to suggest some axioms to be revised. This solution desirably should take into account the knowledge of the source ontologies.

Quality Assessment of the Merged Ontology. Given the central role the merged ontologies are supposed to play in realizing real-world applications, there is a strong need to establish evaluation methods that can measure their quality. Assume the merged ontology in Figure 1.5 is used in an application. The cycle between $C_{5.10}$, C_9, and $C_{6.8}$ will cause a reasoner to consider all these entities to be the same [PB03]. Given the structure of the source ontologies, quite likely, that is not the case. However, if the cycle is broken, the structure of the source ontology either $\mathcal{O}_1$ or $\mathcal{O}_2$ is not preserved in the merged ontology $\mathcal{O}_M$. Moreover, users will want to know whether the corresponding entities are combined in $\mathcal{O}_M$. Similar evaluations are useful before using the created $\mathcal{O}_M$ in an application. However, the required analysis is not always straight forward, especially in large-scale ontologies or with a large number of source ontologies. Here, a set of evaluation indicators can significantly help users to gain an analytical perspective on the merged ontology. These evaluations should not only focus on the correctness in terms of structure, but they should also emphasize the correctness concerning its respective source ontologies. Most studies in ontology merging do not include experimental tests of the merged ontology (cf. [JERS+11]), or evaluate the accuracy of their generated alignment on the merge scenarios (cf. [MTFH14]). Other ontology merging systems (cf. [RR14; JERS+11]) that use a set of criteria to evaluate their methods are usually limited to a few measures and do not fully cover quality aspects. Thus, there is a gap in a comprehensive quality assessment of the merged ontology.

[3] http://oaei.ontologymatching.org/2019/conference/

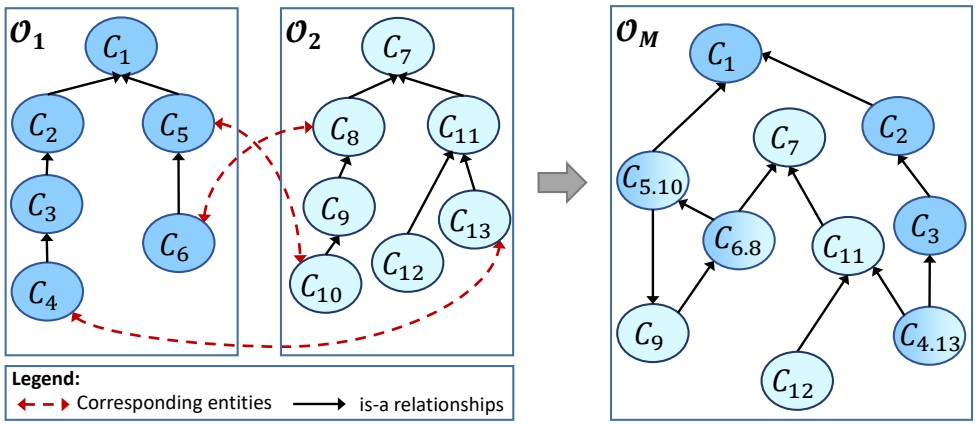


FIGURE 1.5: The merged ontology $\mathcal{O}_M$ for the given source ontologies $\mathcal{O}_1$ and $\mathcal{O}_2$ based on the give correspondences.

In the remainder of this chapter, we present the research hypothesis in Section 1.3, followed by research contribution in Section 1.4, and the outline of the thesis in Section 1.4.

1.3 Research Hypothesis

Based on the problem statements in Section 1.1, we define our research hypotheses. The prime hypothesis of this thesis is the processing of merging multiple ontologies with an n-ary strategy to be more scalable rather than binary merge:

> ***H1.*** *To scale to many sources, merging multiple ontologies using an n-ary strategy can generate the same or better results compared to a series of pairwise binary merges, which results in a performance improvement regarding time and computational complexity.*

Merging multiple ontologies can have a complex search space, especially when the number of source ontologies is increasing. To perform an efficient n-ary merge on multiple ontologies, the complexity of the search space needs to be minimized. This requires a methodology to reduce at least two types of complexities: (1) reducing the time complexity, (2) reducing operational complexity. Partitioning the problem into smaller subtasks can achieve both. Thus, we present the next hypothesis as:

> ***H2.*** *Given a set of corresponding sets extracted from mappings between source ontologies, the entities from n source ontologies can be efficiently partitioned into k blocks in order to facilitate the merge process by providing smaller and more tractable merging subtasks.*

The merged ontology is expected to meet a set of Generic Merge Requirements (GMRs). Customizing the GMRs within an ontology merging system provides a flexible merging

approach. Thus, if a system allows users to select desired GMRs, a system is needed to determine which user-selected GMRs can be met simultaneously.

> ***H3.*** *Given a set of user-selected GMRs, there is a subset of compatible GMRs that can be fulfilled simultaneously in binary or n-ary merge method.*

The entities inside each block should be merged to create a local sub-ontology where the user selected GMRs can be applied in a local refinement. Adjusting the GMRs enables a flexible and customizable merged method. The next step is to construct the final merged ontology based on the sub-ontologies, which also has to ensure adherence to the GMRs globally.

> ***H4.*** *The k blocks can be effectively combined to generate a merged ontology in which a given set of Generic Merge Requirements is compatible fulfilled.*

The final merged ontology should be free of inconsistencies. However, since the encoded knowledge of source ontologies may model different world views, it can easily happen that the merged ontology is inconsistent. These inconsistencies need to be resolved if one wants to make use of the merged ontology in some applications. Thus, a capable method is required to resolve potential issues. This method should make changes such as deleting or rewriting a part of conflicting axioms to turn the inconsistent merged ontology into a consistent one. The rewrite process necessitates a preceding step to detect which axioms among all existing conflicting axioms should be rewritten. Thus a prior function is required to rank the axioms, ideally by considering the source ontologies knowledge. Because the inconsistencies in merged ontologies may stem from differing source ontologies' perspectives on the domain at hand. The whole of this process can be accomplished automatically, or a user can review the system's suggestions and make necessary changes before applying them.

> ***H5.*** *When the merged ontology is inconsistent, it is possible to effectively repair it into a consistent merged ontology by changes to the ranked axioms automatically or with user intervention.*

The created merged ontology plays a central role in a variety of Semantic Web applications. Thus, prior to its usage, the quality and correctness of the merged ontology should be assessed. This requires a comprehensive set of evaluation criteria that systematically cover a variety of characteristics of individual aspects of the merged ontology. These evaluation criteria should also provide an analytic view on how well the created merged ontology reflects the given source ontologies.

> ***H6.*** *A comprehensive evaluation framework can be used to systematically assess diverse aspects of the quality of the merged ontology concerning the respective source ontologies.*

1.4 Research Contributions

This thesis exposes a novel approach for merging multiple ontologies in the Semantic Web domain. We propose an efficient, n-ary ontology merging technique that scales to many ontologies and addresses the mentioned research hypothesizes. We outline our main contributions as follows:

1. **Merging Multiple Ontologies with an N-ary Strategy**:
   We present a scalable, multi-ontology merging method. For efficient processing, rather than successively merging complete ontologies pairwise, we group related concepts across ontologies into blocks and merge first within and then across those blocks. The merged ontology built by our approach supports a set of user-selected Generic Merge Requirements, toward assuring its quality.

2. **Utilizing User-Driven Generic Merge Requirements**:
   We analyze the literature and compile a comprehensive list of Generic Merge Requirements (GMRs) that are used by different approaches. Users can adjust the list towards generating the customized merged ontology. We provide first insights into the compatibility of user-selected and describe a graph-based method for determining maximal sets of compatible GMRs for the user entry.

3. **Inconsistency Handling of the Merged Ontology**:
   We propose a novel Subjective Logic-based approach [Jøs16] to handling the inconsistency problem occurring while merging ontologies. Subjective Logic theory can capture opinions about the world in belief models and provides a set of operations for combining opinions. We apply this logic to rank and estimate the trustworthiness of conflicting axioms that cause inconsistencies within a merged ontology.

4. **Assessing the Quality of the Merged Ontology**:
   We provide a comprehensive set of evaluation criteria to cover a variety of characteristics of individual aspects of the merged ontology in three dimensions: (1) structural criteria via the evaluation of the General Merge Requirement (GMR)s, (2) functional measurements by the intended use and semantics of the merged ontology, and (3) usability evaluation on ontology and entity annotation. Evaluating the merged ontology can be performed even independently of the merge method.

5. ***CoMerger* Tool**:
   The final contribution of this research is the development of the *CoMerger* tool that implements all aforementioned aspects accessible via a unified interface. Our web-based application is freely accessible via `http://comerger.uni-jena.de/`, and the source code is publicly available[4] and distributed under an open-source license.

[4] `https://github.com/fusion-jena/CoMerger`

1.5 Outline of the Thesis

The thesis is subsequently organized as follows:

- **Chapter 2** surveys the literature in four subdomains (i) ontology merge approaches, (ii) Generic Merge Requirements, (iii) ontology inconsistency handling, and (iv) ontology quality assessment.

This concludes *Part I*, the introduction of this thesis. In *Part II*, the proposed approaches will be discussed. This part includes five chapters:

- **Chapter 3** presents an overview of the proposed methods that we will present in the next chapters.
- **Chapter 4** introduces the n-ary merge method applied to multiple ontologies. The method includes three main steps, initialization, partitioning, and combining. The workflow, algorithm, and an example of the proposed merge method will be presented in this chapter, too.
- **Chapter 5** gives a classification of the Generic Merge Requirements (GMR)s. Each GMR is discussed in detail, followed by the compatibility checking between them.
- **Chapter 6** presents the proposed method to handle the inconsistencies arising in the merged ontology. The approach is based on the Subjective Logic theory. We present, first, a brief introduction of Subjective Logic theory and then discuss how this logic can be applied to debug merged ontologies. An example, algorithm, and possible repair plan will be present at the end.
- **Chapter 7** introduces the quality assessment framework adapted for the evaluation of merged ontologies in three dimensions. The structural dimension is measured by General Merge Requirement (GMR)s. The functional dimension is formulated against the intended use and semantics of merged ontologies, and the usability dimension is evaluated by considering the ontology and entity annotation.

Part III is dedicated to the evaluation of the proposed methods in this thesis. It includes six chapters:

- **Chapter 8** gives an overview of the experimental tests carried on the proposed methods in this thesis. The detail of implementation and the used datasets are introduced in this Chapter, too.
- **Chapter 9** presents an overview of the *CoMerger* tool and its components. Moreover, the architecture of the tool in the user and system levels is presented, followed by presenting the GUI of the tool.
- **Chapter 10** shows the experimental tests on the proposed n-ary merge method. The tests include (i) presenting the characteristics of the merged results with different settings, (ii) analyzing by the Competency Questions, and (iii) comparing the binary approaches versus the n-ary merge.
- **Chapter 11** presents the experimental tests by use case studies on the compatibility checker and the conflict resolution.

- **Chapter 12** presents experimental tests on the proposed inconsistency handling method. The tests present characteristics of the inconsistent ontologies along with analyzing the Competency Questions on the consistent and inconsistent merged ontologies. Moreover, we show the extent to which the method is scalable.
- **Chapter 13** shows the empirical analysis of the proposed quality assessment dimensions. Moreover, we will present the time performance of the evaluation metrics, along with the overall result demonstration.

Part IV presents the concluding parts of the thesis. It consists of two chapters:

- **Chapter 14** briefly reiterate our assumption, the proposed approaches, and the discussion of the results. Moreover, we will state the extent to which the research hypothesizes are satisfied.
- **Chapter 15** exposes a set of directives of future work that will enrich and complement this research.

2 Literature Review

In this chapter, first, we present a classification of the ontology merge systems in Section 2.3. We then survey the literature on Generic Merge Requirements (GMRs) in Sections 2.4, followed by reviewing inconsistency handling approaches in Section 2.5. Section 2.6 discusses existing approaches on ontology quality assessment. At the end of each sections, we briefly describe our method concerning that aspect. This chapter is continuing by presenting existing model and ontology merging techniques in Sections 2.2 and 2.1, respectively. We conclude the chapter with a summary of existing approaches in Section 2.7.

2.1 Existing Ontology Merging Approaches

Ontology merging is the process of creating a merged ontology from a set of source ontologies based on given mappings [PB03]. There have been proposed a variety of merge approaches over the last decade. In this section, we give a detailed explanation of each system. We will classify and analyze the ontology merging systems with respect to different aspects in Section 2.3.

- **ATOM** [RR14] stands for Automatic Target-driven Ontology Merging. It merges a source ontology into the target ontology and aims to preserve the preference of one target ontology among the source ontologies. The approach at first creates an intermediate merged result, then refines it based on some of Generic Merge Requirements (GMRs) to produce the final merged ontology. The target-based merge has been evaluated with a full merge concerning the number of concepts and leaf paths.
- **Chimaera** [MFRW00] is an interactive merging tool. It suggests a set of potential merging candidates to the user. Chimaera is able to merge the semantically identical concepts from source ontologies in the merged ontology. Moreover, the system can identify concepts that should be related via the is-a, disjointness, or instance relationships. However, it does not support merging properties and axioms. Chimaera provides about seventy commands in the user interface to

generally edit the ontology and leaves the decision of what to do entirely to the user. Some of these commands are related to ontology merging, such as "merge classes" and "move class x to become a subclass of class y". There are also related commands for diagnosing tasks such as check for incompleteness, cycles, and value-type mismatches. Evaluation of this tool has been demonstrated through a user-study in terms of user actions and the required time for performing the merge process.

- **Chiticariu et al.** [CKP08] proposed a method to enumerate multiple integrated schemas from a set of source schemas. Their method supports refining the enumerated schemas via user interactions. The integrated schemas are built by considering all possible choices of merging concepts. Users can specify constraints on the merging process itself, such as enforcing a pair (or a group) of corresponding concepts to be always merged or never merged. The evaluation of the method is narrowed on analyzing the run time performance and the effect of the user interactions on the number of created integrated schemas.
- **CleanTax** [TBL08] investigates the merging problem when the relationships between concepts of different taxonomies can be expressed as algebraic (RCC-5) constraints [RCC92]. The merge process is performed by combining corresponding entities and creating a new taxonomy based on the given and inferred alignments. The consistency of the result is evaluated under global taxonomic constraints such as parent coverage. The evaluation has been carried with other existing systems in the view of the characteristics of the merge systems such as disjunctive relation support and type of supported reasoning.
- **CODE** [FRP14] stands for Common Ontology DEvelopment. It can be used to merge more than two ontologies at the same time. CODE merges the entities based on four different scenarios with respect to the relations between the corresponding entities between the source ontologies. The inconsistency handling in the view of classical inconsistency definition in [FHP+06; HPS09] is not presented. However, the authors deal with a certain type of disjoint conflict. CODE is evaluated through SPARQL in order to show the knowledge preservation in the merged ontology. However, in this evaluation, only the number of retrieved entities is compared.
- **ContentMap** [JRGHB09] stands for a logiC-based ONtology inTEgratioN Tool using MAPings. It aims at helping users to understand and evaluate the consequences of merging two ontologies as well as identifying and handling the possible errors. ContentMap includes computing and selecting the mapping, computing new entailments, detect and refine of unintended entailments. This includes any unintended entailments, not especially on the inconsistency entailments. Thus, inconsistency handling is narrowed to the refining mappings to prevent the inconsistency in the merged ontology. The approach has been evaluated only on two pairs of ontologies against the human-created merged ontology in the view of entailment satisfaction.
- **CreaDo** [JERS+11] is a parameter-based ontology merging technique that selects a subset of mapping based on the merge purpose. It allows the creation of a merged ontology only with relevant information for the specific purpose. To conduct this,

CreaDo uses the ontology modularization technique in [DTI07] to extract ontology modules (relevant information) from each source ontology. A merge parameter of CreaDo is a concept of the domain that should be represented in the merged ontology. Once the merged ontology is created, the detected errors will be reported to the user and no refinements take place for it. For the evaluation of the method, the authors reported basic statistics about a few common pitfalls related to the general design of the ontology taken from [PVSFGP10].

- **DKP-AOM** [Fah17] imports the first source ontology as the merged ontology and then performs several operations to build the combined definitions for each concept from the second source ontologies. Indeed, axioms from source ontologies are matched together, merging is performed on them, and the combined axioms are added in the merged ontology. DKP-AOM tries to prevent the inconsistency in the merged ontology by refining the initial mappings. For the evaluation of the merged method, the author presented how the method can produce the merged ontology on a pair of source ontologies by representing which entities can be merged (based on the detected corresponding entities). The created merged ontology did not evaluate furthermore, but the accuracy of the created alignment has been compared with other approaches.
- **FCA-Merge** [SM01], at first, extracts instances from a given set of domain-specific text documents by applying natural language processing techniques. Then, based on this information, it uses the Formal Concept Analysis (FCA) [GW12; Wil09] technique to produce a lattice of concepts that relates concepts from the source ontologies. Finally, it derives the merged ontology from the concept lattice with human interaction. The algorithm suggests equivalence and subclass–superclass relations. An ontology engineer can then analyze the result and use it as a guide for creating a merged ontology. No evaluation and consistency checker has been studied in FCA-Merge.
- **GCBOM** [PK19] performs the merging ontologies by applying the granular computing processes, including association, isolation, purification, and reduction upon the similarity values of corresponding entities. These operations modify the granularity level in order to produce the final merged ontology. The evaluation has been performed on two pairs of small ontologies in terms of the size of created merged ontologies.
- **GROM** [MTFH14] proposes a formal approach for merging ontologies using typed graph grammars with algebraic graph transformations. GROM replaces the equivalent entities from the second source ontology in the first source ontology to build a common ontology graph. After that, the properties are assigned to create the global merged ontology, if no conflict (such as maxCardinality conflict) in the final result occurs through those properties. No experimental test and evaluation result has been reported in [MTFH14], only the coverage metric (cf. [RR12]) has been mentioned as one quality metric.
- **HCONE-merge** [KVS06] uses linguistic and structural knowledge about ontologies to formalize the Latent Semantics Indexing (LSI) method [DDF+90]. Then, it makes use of the LSI mechanisms for computing all the possible

correspondences by mapping the intended informal meaning of concepts onto WordNet senses. Based on the detected mappings, the algorithm decides whether to merge two concepts. For the evaluation of the merged result, the authors used two small source ontologies, which an expert has created a gold-standard merged ontology of them. Then, the tool created ontology is compared with the human-created one in terms of the suggested corresponding entities to be merged. Thus, this evaluation is not on the evaluation of the merged result, but on the alignment accuracy.

- **HSSM** [PC19] generates the formal context for the source ontologies and converts them to the concept trees. Based on the similarity values (highly relevant, moderate relevant, and least relevant) within the concept trees, HSSM performs two different merge process. The dependent merging method merges the highly relevant nodes with their children nodes. Independent merging technique either merges moderate relevant nodes and all of the least relevant nodes or merges only the moderate relevant nodes and drops the least relevant nodes. The evaluation has been performed on two small ontologies from the social network domain in terms of the merged ontology's compactness.

- **iPrompt** [NM03] is an interactive ontology-merging tool. This system leads users through the ontology merging process by suggesting what should be merged, identifying inconsistencies and potential problems, and suggesting strategies to resolve them. The assumption about inconsistency in iPrompt is different from the classical definition of the inconsistencies. The consistency is defined as the conflicting names (i.e., duplicated names), dangling reference (referring a concept to another one which does not exist in the merged ontology), redundancy on the class hierarchy (existence of more than one path from a class to a superclass), and value restriction conflict. The evaluation is carried through a user-study on the quality of iPrompt's suggestion by analyzing whether the users follow the tool's suggestions.

- **Makwana & Ganatra** [MG18b] used the Jaccard Similarity Index (JSI) measure to create groups of ontologies by the k-means algorithm [Mac+67] based on a global similarity measure. Then, the authors merged the ontologies from a given specific group using the adapted GROM [MTFH14] tool. Although the authors deal with merging multiple ontologies, the approach indeed is narrowed to merging two ontologies per time due using GROM, a pairwise ontology merger. In addition to the SPARQL queries analyzing, the created merged ontologies have been evaluated with compactness, coverage, and redundancy (introduced criteria in [RR12]). The authors extend the explanation of the approach in [MG18a].

- **MeMo** [ALL10] is a system that calculates the similarity among the source ontologies to define an order in which the ontologies should be merged. MeMo uses a clustering technique in order to group the most similar ontologies. Although the system deals with merging multiple ontologies, it uses a set of the binary merge to achieve the final merge result. In each step of the running binary merge, MeMo takes the most two highest similar ontologies and combines them based on the ideas of Vanilla [PB03] in order to define the merge function. After each binary

merge, MeMO requires to calculate the similarity between the newly merged ontologies with the remaining source ontologies. The tool created merged ontology has been compared with the gold standard ontologies in which they were built by domain experts using iPrompt tool [NM03]. This evaluation emphasized on difference between the characteristic on the tool created merged ontology with the human-created one, such as the number of entities, class hierarchy, and label of entities. The ontologies used in this study are small, where the biggest one has 42 classes, only.

- **MoA** [KJH+05] finds the alignment between two source ontologies and based on the type of mappings between two concepts, decides whether to merge them. The method has been evaluated on three small pairs of ontologies by analyzing the expert's positive responses on the MoA's suggestions.
- **OIM-SM** [ZRL17] stands for Ontology Integration Method based on Semantic Mapping. OIM-SM first generates a network-based knowledge model for two source ontologies by merging the corresponding concepts. Then this model is split into several blocks. For each block, OIM-SM creates the mapping between its concepts, and re-align these concepts into a branch of the tree-based model. Finally, the network-based model is reconstructed to generate a tree-based model (cycle free) of the final merged ontology. Besides evaluating the accuracy of the generated mappings, the authors evaluated one pair of source ontologies against the human-created one in terms of the number of merged concepts and the time of processing.
- **OM** [GAC10] merges two ontologies A and B to produce a third ontology C (the merged result) in order to accumulate enough knowledge about a certain topic. OM imports the ontology A in the merged result (C) completely, then incrementally adds and merges the concepts from ontology B on the A's entities if they do not contradict knowledge from A. In this view, this method considers the restriction from both ontologies. The merged ontology achieved by OM has been evaluated against the human-created merged ontology in terms of result compactness.
- **OMerSec** [MFBB10] can merge more than two ontologies at the same time. It performs the merging process by clustering different entities belonging to source ontologies and making inferences on initial axioms. Then it uses the information in source ontologies to validate the axioms and build a global ontology. The classical inconsistency has not been investigated in OMerSec. However, synonymy and homonymy conflicts have been handled in the global set of concept pairs. The accuracy of the created correspondences has been compared with the other approaches, only.
- **Onto-Integrator** [EGED09] is an interactive tool to merge the ontologies axioms by providing a set of suggestions to the user. Based on the similarity values between source ontologies' entities, Onto-Integrator decides whether to merge them, create a common superconcept for them, or make one of the concepts as a subconcpet of the other one. This method creates at first an initial integrated ontology, then constructs a multiple-lattice to detect additional subsumption cases and discover new higher-level concept abstractions. Based on the refined mappings,

Onto-Integrator prevents of inconsistency in the merged ontology. The evaluation has been narrowed to a user-study, in which the users' discussions on the tool's suggestions have been analyzed.

- **PORSCHE** [SBH08] stands for Performance ORiented SCHEma mediation. It semi-automatically merges several tree-structured XML schemas and holistically clusters all matching elements in the nodes of the merged schema. Initially, it creates an intermediate schema based on one of the source schemas, and then it incrementally merges the other source schemas with this schema. During the merge process, if a node from a source schema has no correspondence in the intermediate schema, a new node is created to accommodate this node. The classical inconsistency has not been studied in PORSCHE, and only the data type conflict has been analyzed. The integrity of the merged schema has been evaluated regarding completeness and minimality measures.

- **Radwan et al.** [RPSY09] proposed a top-k ranking algorithm for the automatic generation of the best candidate schemas. The algorithm gives more weight to schemas that combine the concepts with higher similarity or coverage. The top-k generated integrated schema is presented to the user, in which the user can select the final desired merged schema. To carry the merge operation, the authors used the proposed method of Chiticariu et al. [CKP08]. The evaluation of the proposed tool has been carried via a user-based study by analyzing the number of actions that the users perform to achieve the final merged result.

- **SAMBO** [LT06] is a general framework for aligning and merging ontologies in interaction with the user to decide on the suggested alignments. SAMBO uses a simple technique to merge the corresponding entities. However, the underlying assumption about the merge method is not made explicit. The users can ask the SAMBO system to check the consistency of the merged result using an existing reasoner, but to the best of our knowledge, there is no explicit inconsistency handling in this system. The evaluation is mostly related to the alignment accuracy in order to compare the quality of the corresponding entities' suggestions with the human suggested ones.

- **SASMINT** [UA10] stands for Semi-Automatic Schema Matching and INTegration system. The input of the system is a pair of schemas of relational databases, and XML is chosen as the output in SASMINT for representing the integrated schema. The user can modify and approve the result of the created alignment. After the user validation process, schema integration is performed by applying several schema derivation rules. These rules are customized in the relational schemas such as table renaming or columns union' rules. The approach follows an asymmetric strategy, which in the case of naming conflict, the preferred source ontology' naming is applied. The authors checked certain types of conflicts in the area of the database schema, such as databases' key conflict. For the evaluation, the authors used the completeness and minimality criteria.

2.2 Overview of Existing Data Model Merging

There are quite different studies on the merging data models such as [BDK92; PB03; BDK92; QKL07; MRB03]. Since our contribution is related to ontologies merging not model merging, we briefly survey the literature in the model merging studies in this section. However, we emphasized to a greater extent on the details of ontologies merging systems in the previous section.

A theoretical aspect of model merging has been started in the early work in [BDK92]. The authors give a general formalism of merge operation and show how to resolve a certain kind of conflict on the database schema level to produce a unique result. Later, in the Vanilla system [PB03], the authors provided a more practical solution by expanding the work in [BDK92]. Vanilla is a system that proposed several operations for the model merging with instantiating on ontologies merging. The first notation of Generic Merge Requirements (GMRs) has been introduced in Vanilla to present a set of requirements for model management. These works were later extended by GeRoMe [QKL07] and Rondo [MRB03] systems. GeRoMe presents a merge algorithm that is based on the intensional relationships between models. This enables the integration of models represented in different modeling languages. Rondo is introduced as a model management system prototype in which high-level operators are used to manipulating models and mappings between models. A set of generic operators, including merge operation, has been implemented in Rondo. In general, these merging model prototypes have been used as the base for the several ontology merging systems such as [RR14; CKP08; ALL10].

2.3 Classification of the Ontology Merge Approaches

Merge approaches can be outlined from three different perspectives: (1) *binary* vs. *n-ary*, (2) *one-level* vs. *two-levels*, and (3) *symmetric* vs. *asymmetric*.

2.3.1 Binary vs. N-ary Merge

Merging strategies basically have been divided into two main categories [BLN86]: *"binary"* and *"n-ary"*. The *binary* approach allows the merging of two ontologies at a time, while *n-ary* strategy merges n ontologies ($n > 2$) in a single step. To deal with merging more than two ontologies, the *binary* strategy requires a quadratic complexity of the merging process in terms of involved ontologies. However, the number of merging steps is minimized in the *n-ary* strategy. Most methodologies in the literature, such as [RR14; NM03; JRGHB09] agree on adopting a *binary* strategy due to the simplicity of the search space. Applying a series of binary merges to more than two ontologies is not sufficiently scalable and viable for a large number of ontologies [Rah16]. The existing n-ary approaches [SBH08; FRP14; CKP08; MFBB10] deal with merging multiple ontologies in a single step, however, each of these systems suffers certain drawbacks. In [SBH08], the final merge result depends on the order in which the source tree-structured XML schemas are matched and merged. The scalability of the approach in [FRP14; MFBB10; CKP08] is not clear since the experimental tests carried on a few

small source ontologies. For instance, in [CKP08], only three source ontologies have been merged. Despite the efforts of these research studies, developing an efficient, scalable *n-ary* method has not been practically applied and still is one of the crucial challenges in this field.

In our proposed method, we develop a scalable n-ary strategy to merge multiple ontologies at the same time by utilizing a divide and conquer approach.

2.3.2 One-level vs. Two-levels Merge

Merging ontologies either in a binary or n-ary strategy can be categorized into two different types: *"one-level merge"* and *"two-levels merge"*. The *one-level merge* approaches tend to create the merge result in one processing step without creating an intermediate result (cf. [JERS+11; MFRW00; SM01]). In contrast, in the *two-levels* merge type, an intermediate merge result is produced at the first level. Then, in the second level, the intermediate result is processed to generate a final merge result. In this merge type, a set of refinements is carried on the intermediated result. For instance, applying a set of GMRs in ATOM [RR14], utilizing granular processing in GCBOM [PK19], and considering source ontologies' restrictions in OM [GAC10] have been performed. The output of the second level in literature identically is called *merged ontology*. Whereas, the outcome of the first level comes under different names, such as an *integrated concept graph* in ATOM [RR14], *network-based knowledge model* in OIM-SM [ZRL17], *common ontology graph* in GROM [MTFH14], or *intermediate schema* in PORSCHE [SBH08].

The proposed method of this thesis, *CoMerger*, is built upon the *two-levels* technique. *CoMerger* creates at first an initial merged model. Then, through local and global refinements, the intermediate result is refined to generate the final merged ontology.

2.3.3 Symmetric vs. Asymmetric Merge

The merge process can be performed based on the two natures [RR14]: *"symmetric merge"* and *"asymmetric merge"*. The newly created merged ontology can keep safe the structure of one of the preferred source ontologies. This case calls an *asymmetric merge*, where the source ontologies have a different priority and are not considered equally important (cf. [RR14; MTFH14; Fah17]). In contrast, the *symmetric merge* considers all of the source ontologies with equal priority, where the merged ontology may or may not resemble any of the source ontologies' structure (cf. [CKP08; MFRW00; JERS+11]). The *symmetric merge* aims at maintaining all information of the source ontologies to achieve complete coverage. On the other side, the purpose of *asymmetric* approaches is to discard part of the information from the non-preferred source ontology to reduce redundancy in the merged ontology.

In *CoMerger*, the user can decide to perform the symmetric or asymmetric merge. In the last case, the user can adjust the preferred ontology.

2.4 Survey on Generic Merge Requirements

The Generic Merge Requirements (GMRs) have been first introduced in the Vanilla system [PB03]. Later other merge approaches implicitly or explicitly took them into consideration [RR14; MFRW00; CKP08; TBL08; FRP14; JRGHB09; JERS+11; SM01; PK19; MTFH14; PC19; NM03; ALL10; ZRL17; GAC10; EGED09; SBH08; RPSY09; LT06; UA10]. GMRs are a set of Generic Merge Requirements that the merged ontology is expected to achieve. To provide customizable GMRs through *CoMerger*, as our contribution, we surveyed the literature to compile a list of GMRs. This investigation leads to extracting twenty GMRs, summarized in Table 2.1, which will be explained in more detail in Chapter 5. We extracted GMRs by studying three different research fields: (1) ontology and model merging methods, (2) ontology merging benchmarks, and (3) ontology engineering domain:

1. *Ontology and model merging methods*: The GMRs *R1-R6*, *R8-R16*, *R19* have been extracted from existing ontology and model merging methods such as [RR14; JRGHB09; JERS+11]. These approaches aim to implicitly or explicitly meet at least one of the GMRs on their created merged ontology.

2. *Ontology merging benchmarks*: The existing benchmarks [MFH16; RR12] on the ontology merging domain introduced general desiderata and essential requirements that the merged ontology should meet. The criteria stated in these benchmarks are based on earlier research in [DB10], a study of the quality measurement of the merged ontology. In this respect, *R1*, *R4*, *R7- R9* have been extracted from these research studies.

3. *Ontology engineering*: Researchers of the ontology engineering domain [NM+01; PVSFGP12; RDH+04] came up with a set of criteria to present the correctness of an ontology, which is developed in a single environment. It is worthwhile to consider these criteria also on the merged ontology because the newly created merged ontology may be viewed the same as the developed ontologies in this category. Not all of these criteria can be extended in the ontology merging scenario since some relate to the problem of the source ontologies modeling, in which the merge process can not affect them. In this regards, we recast *R15 - R20* from this category. By recast, we mean customizing them in the ontology merging scenario in which, if the source ontologies do not fulfill those criteria, the merged ontology does not obligate to follow them. For example, *R20* emphasizes that the classes should not be unconnected in the merged ontology. However, if the source ontologies include unconnected classes, then this GMR is hard to be achieved in the merged ontology. To make GMRs' definitions free of the source ontologies modeling errors, we recast them in the context of the ontology merging domain. In this follow, the recast definition of *R20* is that all connected classes from the source ontologies should not be unconnected in the merged ontology.

There is a slight overlap between the criteria of the categories above. Some of GMRs such as *R16* and *R19* belong to both merge approaches and ontology engineering categories. Some others, e.g., *R1*, and *R8* are shared between the ontology merging systems and benchmarks. This is due to the fact that the mentioned categories are not significantly

separated because all are covering particular aspects of ontology modeling but overlap to a certain degree of ontology process.

TABLE 2.1: Generic Merge Requirements (GMRs).

***R1*- Class preservation**
All classes of (all/target) source ontologies should be preserved in the merged ontology [RR14; MFRW00; CKP08; TBL08; FRP14; JERS+11; MTFH14] [NM03; MG18b; SBH08; RPSY09; DB10; PB03; MFH16; RR12; UA10; ALL10].
***R2*- Property preservation**
All properties of (all/target) source ontologies should be preserved in the merged ontology [RR14; CKP08; TBL08; FRP14; PB03; EGED09].
***R3*- Instance preservation**
All instances of (all/target) source ontologies should be preserved in the merged ontology [RR14; CKP08; FRP14; SM01; PB03].
***R4*- Correspondence preservation**
The corresponding entities from source ontologies should be mapped to the same merged entity [RR14; NM03; PB03; RR12].
***R5*- Correspondences' property preservation**
The merged entity should have the same property of its corresponding entities [NM03; PB03].
***R6*- Property's value preservation**
Properties' values from the (all/target) source ontologies should be preserved in the merged ontology [NM03; PB03].
***R7*- Structure preservation**
The hierarchical structure of source ontologies' entities should be preserved in the merged ontology [DB10; ALL10; SJRG14].
***R8*- Class redundancy prohibition**
No redundant classes should exist in the merged ontology [TBL08] [PK19; PC19; GAC10; SBH08; LT06; DB10; MFH16; PB03; RR14; UA10].
***R9*- Property redundancy prohibition**
No redundant properties should exist in the merged ontology [CKP08; MFH16; EGED09].
***R10*- Instance redundancy prohibition**
No redundant instances should exist in the merged ontology [SM01; MTFH14].
***R11*- Extraneous entity prohibition**
No additional entities other than the source ontologies' entities should be added in the merged result [PB03].
***R12*- Entailment deduction satisfaction**
All entailments of the (all/target) source ontologies should be satisfied in the merged ontology [JRGHB09; TBL08].
***R13*- One type restriction**
Any merged entity should have one data type [PB03].
***R14*- Property value's constraint**
Restriction on property's values from source ontologies should be applied without conflict in the merged ontology [JRGHB09; PB03].
***R15*- Property's domain and range oneness**
The merge process should not result in multiple domains or ranges defined for a single property [PVSFGP12].
***R16*- Acyclicity in the class hierarchy**
The merge process should not produce a cycle in the class hierarchy [CKP08] [ZRL17; RPSY09; LT06; DB10; JERS+11; NM+01; PB03; PVSFGP12; RR14].
***R17*- Acyclicity in the property hierarchy**
The merge process should not produce a cycle between properties with respect to the is-subproperty-of relationship [PVSFGP12; FMB12].
***R18*- Prohibition of properties being inverses of themselves**
The merge process should not cause an inverse recursive definition on the properties [PVSFGP12].
***R19*- Unconnected class prohibition**
Each connected class from source ontologies should not be unconnected in the merged ontology [JERS+11; PB03; PVSFGP12].
***R20*- Unconnected property prohibition**
Each connected property from the source ontologies should not be unconnected in the merged ontology [NM03; PVSFGP12].

2.5 Literature on Ontology Inconsistency Handling

The classical inconsistency definition has been stated in [FHP+06; HPS09]. According to that, an *inconsistent ontology* is an ontology, that by virtue of what has been stated in the ontology, cannot have any models and entails everything. In this section, we survey the inconsistency handling concerning this definition.

Mostly, ontologies' inconsistency handling in the literature (cf. [HVHH+05; KPSCG06; LPSV06; SPF+12]) has been contemplated in the context of ontology development in the single environment. However, the possibility of inconsistencies handling in the ontology merging domain has not sufficiently been examined in the literature. The main difference between the two is that inconsistencies in a single ontology are typically the result of modeling errors and thus relatively easy to correct. In contrast, inconsistencies in merged ontologies may stem from differing perspectives on the domain at hand, each of them correct in their own right. Resolving these inconsistencies is more problematic and can be facilitated by determining the degree of trust of the respective source ontologies. This can be a considerable challenge even for experts in the face of modestly sized ontologies. In this regard, we distinguish between two different inconsistency handling scenarios: (1) ontology inconsistency handling in a single development environment, and (2) inconsistency handling in the ontology merging domain.

2.5.1 Ontology Inconsistency Handling in the Single Development Environment

Handling inconsistencies developed in the single environment has attracted considerable attention within the research community. In [FHP+06], the authors provided a well-accepted notion of inconsistency. Upon that, in other studies [HPS09; KPSH05; SC+03; PSK05], the authors provided a systematic way to detect the unsatisfiable concepts and their justifications within an inconsistent ontology. Unsatisfiable concepts are those who cannot be true of any possible individual, that is, they are equivalent to the empty set. Justification is a minimal subset of an ontology that causes it to be inconsistent. The detected justification sets have been further processed with a variety of research studies such as [KPSCG06; LPSV06] to be ranked or analyzed. In [KPSCG06], a single metric is used to rank the justifications. The approach in [LPSV06] ranks the axioms based on the history of the ontology's editing process, which might not be realistic in practice. Similarly, there is a group of studies such as [HVHH+05; PDT06] on handling inconsistency in the ontology evolution process. These approaches commonly keep different versions of the ontology to handle inconsistencies.

2.5.2 Inconsistency Handling in the Ontology Merging Domain

In the ontology merging domain, the merged result is built basically based on the knowledge provided by the source ontologies. The impact of the source ontologies' knowledge on the merged results is a key point that must be taken into account when we deal with inconsistencies in the merged ontology. This consideration has been investigated in [JHQ+09; PTP+18] within the multiple ontologies domain. The approach in [JHQ+09] considers multiple ontologies that are networked via mappings

for distributed and networked environments, only. For the repair process, the approach focuses on repairing the mappings, and only axioms in the mappings could be removed to resolve the inconsistency. Moreover, their ranking function does not depend on the assumption of the source ontologies. The proposed method in [PTP+18] builds a consistent ontology in the ontology aggregation setting, which mostly depends on using a reference ontology result, the submitted preference profile by the users.

Techniques used in alignment debugging can also be reviewed in this scenario. For instance, research studies in [EGED09; FMB12; JRGHB09; JRGH12] examined the inconsistency problem in the merged ontology based on the alignment of the respective source ontologies. In principle, this type of study deals with the initial stages of the ontology merging process to prevent the inconsistency in the merged ontology by analyzing the list of mappings. They process the given mapping before using them in the merging entities. If the specific mapping makes the merged result inconsistent, this mapping is dropped, i.e., a set of corresponding entities does not consider as correspond and does not merge. The removal of these alignments would achieve consistency in the merged ontology. The approach in [FMB12] is quite different from the works above. The assumption about inconsistency definition in this method is different from the well-known inconsistency definition in [FHP+06; HPS09]. The authors considered other types of modeling errors, such as redundancy and cycle on class or property hierarchies with the notation of inconsistency. However, following [FHP+06; HPS09], we do not consider these types of modeling errors as the inconsistency. Moreover, our method is not narrowed to debugging alignment since we aim to generalize it on handling the inconsistent merged ontology, independent of the merge method that it builds upon.

In this thesis, we consider the knowledge of source ontologies in the handling of the inconsistencies within the merged ontology by using the Subjective Logic theory [Jøs16]. This provides the necessary mechanisms to capture the subjective opinion of different communities represented by the source ontologies on the trustworthiness of each axiom in the merged ontology and identifies the least trustworthy axioms. To the best of our knowledge, the only study which considers Subjective Logic in this context is the approach in [SPF+12]. However, the authors only utilized the atomicity value and omitted the belief, disbelief, and uncertainty degree from this theory. Moreover, no agent's opinion combination has been considered. Lastly, we apply Subjective Logic to handle inconsistencies within the ontology merging domain, not in a single development environment, where the source ontologies assumptions play a central role.

2.6 Survey on the Ontology Quality Assessment

We distinguish between two different ontology evaluation scenarios: (i) ontology evaluation in a single deployment scenario, and (ii) ontology evaluation in the context of ontology merging. The first one focuses on how well (in terms of structure) the single ontology is created, independent of any other ontology. While the evaluation in the second scenario emphasizes the correctness and comparisons concerning to its respective source ontologies. The evaluation methods developed for the first scenario are not sufficient to address all relevant aspects in the second scenario, therefore, warrants

its own evaluation methods. Although the first scenario has been widely covered, the evaluation of merged ontologies has received far less attention.

2.6.1 Ontology Quality Assessment in the Single Deployment Scenario

Many approaches aim to assess the quality of one given ontology independent of its relatedness to other ontologies or the ones it was built upon. These approaches are analyzed and classified in a few research studies [HS14; BGM05; RC15]. In this section, we present a summary of the introduced classifications.

- *Gold-standard comparison* [HS14; RC15]: This typically compares an ontology against a gold-standard, which is suitably designed for the domain of interest. The gold-standard evaluation methods require a well-constructed ontology to serve as a reference in that domain. According to [HS14], a major limitation of this approach is that the gold standard itself needs to be evaluated, and it is crucial to establish the quality of the gold standard. Creating a suitable gold ontology is a challenging task [RC15] since it should be one that was created under the same conditions with similar goals to the ontology that want to be evaluated.
- *Application or task-based evaluation* [HS14; BGM05]: This analyzes how effective an ontology is in the context of an application or a use-case. This type of evaluation considers that a given ontology is intended for a particular task, and is only evaluated according to its performance in this task. There exist two main issues with the task-based approach to ontology evaluation [HS14]. First, it is hard to generalize the results of a task-based evaluation because what is applicable in one application context may not be applicable in another. Second, this is highly suitable for a small set of ontologies and would certainly become unmanageable in an automated setting with a variable number of ontologies.
- *User-based evaluation* [HS14]: This generally involves evaluating the ontology through users' experiences. User study evaluation concerns two problems [HS14]: it is difficult to establish objective standards on the criteria for evaluation, and it is also hard to establish who the right users are.
- *Data-driven evaluation* [HS14; BGM05; RC15]: Corpus-based approach [RC15] or namely data-driven evaluation [HS14; BGM05] compares the ontologies with existing data about the respective domain. They are used to evaluate how far an ontology sufficiently covers a given domain. Based on [HS14], the major limitation of data-driven ontology evaluation is that domain knowledge is implicitly considered to be constant. This is contradictory to the literature's assertions about the nature of domain knowledge.
- *Criteria-based evaluation* [RC15; BGM05]: These approaches mainly measures how far an ontology adheres to certain desirable criteria. In [RC15], criteria-based evaluation approaches are considered as the most efficient technique in evaluating the clarity of an ontology. The clarity could be evaluated using simple structure-based measures, or more complex metrics. In addition, this type of approach is capable of measuring the ability of the used tools to work with the ontology by evaluating the ontology properties. Moreover, criteria-based measures

are efficient in detecting the presence of contradictions by evaluating the axioms in an ontology. The general problem of criteria based evaluation in [BGM05] has been relegated to the question of how to evaluate the ontology concerning the individual evaluation criteria. On the positive side, these approaches allow us to combine criteria from different aspects. In this regard, we evaluate mainly the *CoMerger* by criteria-based evaluation technique.

2.6.2 Quality Assessment in the Ontology Merging Context

In this section, we present the evaluation strategy of the existing ontology merging systems categorized in the introduced classification from Section 2.6.1.

- *Gold-standard comparison*: The tool-created merged ontology has been compared against a human-created one in a couple of ontology merging systems such as [JRGHB09; ZRL17; GAC10; ALL10] and in the ontology merging benchmarks [MFH16; RR12]. In the proposed ontology merging systems, the comparison has been performed in terms of the size of merged result [ZRL17; GAC10], class hierarchy and size [ALL10], the time of processing [ZRL17], and the entailment satisfaction [JRGHB09]. The evaluation in these approaches encounters with two main drawbacks. First, the ontologies in these tests are small in size. For instance, in [ALL10], the biggest ontology includes 42 classes, only. So, the assessment of the quality on the medium or large scale ontologies is not intelligible. Second, only a few pairs of ontologies have been evaluated. Thus, we cannot generalize that the underlying evaluation is accurate on a variety of ontologies. For instance, in [JRGHB09; ZRL17; GAC10], one or two pairs of source ontologies have been tested. Two benchmarks for the ontology merging domain are introduced in [MFH16; RR12]. Benchmark [RR12] includes simple taxonomies, and only the number of paths and concepts of the tool-created merged ontology with the human-created ones are evaluated. Other criteria on the properties are not considered. Thus, it could not be extended for non-taxonomy ontologies. To our knowledge, this benchmark is not publicly available, so others are unable to use it. Benchmark [MFH16] included a few small ontologies. The authors presented criteria, achieved by their tool [MTFH14] without any comparison with human results.
- *Application or task-based evaluation*: To the best of our knowledge, the evaluation of the merged ontology in the context of an application or a use-case scenario has not been covered in the literature.
- *User-based evaluation*: Some researchers in [MFRW00; RPSY09; EGED09; NM03; KJH+05] provided a platform for user-based evaluation in the ontology merging scenario. Mostly, the authors analyzed the number of actions that a user performs to achieve the merged ontology (cf. [MFRW00; RPSY09]), or on the required time for the merge process (cf. [MFRW00]). In [NM03; EGED09; KJH+05], the authors analyzed to what extent users agree with the tool's suggestions. Thus, it is mainly related to the merge method's evaluation, not on the merge result. Moreover, the ontologies in this evaluation are small, and mostly only a few pairs of ontologies are evaluated by the users (cf. [KJH+05]). The task of user-based evaluation

generally is a labor-intensive task for humans, and insufficient, especially for large-scale ontologies or a large number of source ontologies.

- *Data-driven evaluation*: To the best of our knowledge, this approach is also barely used. Some semi-related attempts, such as [LBBH15] have been proposed, where a set of corpora and ontologies are merged to build a merged ontology. Their evaluation focuses only on query processing analysis. However, data-driven evaluation requires analyzing how well the created merged ontology covers a topic of the domain-corpus or how well it fits the domain knowledge by comparing ontology concepts. These issues are not covered in [LBBH15].
- *Criteria-based evaluation*: The authors in [RR14; PK19; PC19; FRP14] considered the evaluation of the merged ontology's size or compactness (introduced metric in [RR12]). In addition, in [RR14], the number of leaf path is examined. In other researches [MG18b; SBH08; UA10] two more criteria, namely coverage, and redundancy (introduced criteria in [RR12; DB10]) have been considered. Moreover, in CreaDo [JERS+11], the authors reported basic statistics about a few common pitfalls related to the general design of the ontology taken from [PVSFGP10].

To summarize, most approaches lack sufficient experimental evaluation on the merged result (cf. [SM01]). In others, such as GROM [MTFH14], the experimental evaluations have been narrowed down to the detected mappings, not the merged result. Moreover, user-based evaluation is a complex, time-consuming, error-prone task, and hard to achieve for large-scale ontologies. State of the art is far from an adequate benchmark. The ontology merging systems that evaluate their methods with criteria-based techniques are usually limited to a few measures and do not fully cover the most required aspects of the merged ontology's quality. This concludes a need for a comprehensive evaluation and demonstrates a gap in analyzing how well the merged result is.

In this thesis, we propose a new quality assessment for the merged ontology based on a set of criteria. In [GCCL05], the authors provided a theoretical framework for modeling ontology evaluation. We extend our evaluation criteria on top of the categories proposed there classified into structural, functional, and usability measures. We provide the formulation for them and analyze how these dimensions can be evaluated on the merged ontology in practice. We formulate the structural measures via General Merge Requirement (GMR)s, define the systematic formulation to evaluate the functional measures against the intended use and semantics of the merged ontology, and provide criteria for the usability dimension. As our novelty, we bring these dimensions with the systematic formulations rather than the theoretical aspects in [GCCL05] within the context of the merged ontology's evaluation. Moreover, we consider desirable characteristics towards an ideal evaluation into our formulation.

2.7 Summary

Table 2.2 shows a summary of existing ontology merge approaches in six main features introduced in the previous sections. The merge strategy (binary or n-ary) and the type of merge (one-level or two-levels) for each approach are specified in *columns 3-4*. Whether

the nature of the merge process is symmetric or asymmetric, is presented in *column 5*. *Column 6* indicates the GMRs that are fulfilled implicitly or explicitly by each ontology merge approach.

Column 7 shows to what extent the approach deals with inconsistency on the merged result in three categories: (i) the symbol ✓ demonstrates that the system deals with inconsistency based on the assumption of classical inconsistency defined in [FHP+06; HPS09]; (ii) the notation × indicates that no inconsistency handling is carried; and (iii) the symbol $\times^*$ shows that the approach deals with other types of inconsistencies rather than classical inconsistency.

The *last column* illustrates the evaluation technique used in each system in order to assess the quality of the merged result. If the evaluation technique belongs to none of the categories in Section 2.6, we mark it by "others". However, the exact evaluation technique of each method is already explained in Section 2.1. In case that the ontology merge approach provided no evaluation on the merged result, we mark it by × in Table 2.2. Those approaches which are narrowed only on the evaluation of the generated alignments and did not perform further evaluations on the merged ontology's quality are also shown by ×. The last row of Table 2.2 reports the characteristics of *CoMerger*, the proposed method of this thesis.

TABLE 2.2: Summary of existing ontology merging methods: Merge strategy (binary or n-ary); Merge type (one-level or two-levels); Merge nature (symmetric or asymmetric); Fulfilled GMRs; Inconsistency handling (✓) or not (×), ×* shows the approach handles other types of inconsistencies rather than classical ones; Evaluation technique of the merged result.

No.	Approach	Merge Strategy	Merge Type	Merge Nature	Fulfilled GMRs	Inc. Han.	Evaluation Technique
1	ATOM [RR14]	binary	two-levels	asymmetric	*R1-R4, R16*	×	criteria-based
2	Chimaera [MFRW00]	binary	one-level	symmetric	*R1, R16*	×	user-based
3	Chiticariu et al.[CKP08]	n-ary	one-level	symmetric	*R1-R3, R9, R16*	×	others
4	CleanTax [TBL08]	binary	two-levels	symmetric	*R1, R2, R8, R12*	×*	others
5	CODE [FRP14]	n-ary	one-level	symmetric	*R1-R3*	×*	criteria-based
6	ContentMap [JRGHB09]	binary	two-levels	symmetric	*R12, R14*	✓	gold-standard
7	CreaDo [JERS+11]	binary	one-level	symmetric	*R1, R16, R19*	×	criteria-based
8	DKP-AOM [Fah17]	binary	two-levels	asymmetric	-	✓	×
9	FCA-Merge [SM01]	binary	one-level	symmetric	*R3, R10*	×	×
10	GCBOM [PK19]	binary	two-levels	asymmetric	*R8*	×	criteria-based
11	GROM [MTFH14]	binary	two-levels	asymmetric	*R1, R10*	×	×
12	HCONE-merge [KVS06]	binary	tow-levels	symmetric	-	×	×
13	HSSM [PC19]	binary	one-level	asymmetric	*R8*	×	criteria-based
14	iPrompt [NM03]	binary	one-level	symmetric/ asymmetric	*R1, R4-R6, R20*	×*	user-based
15	Makwana & Ganatra [MG18b]	binary	two-levels	asymmetric	-	×	criteria-based
16	MeMo [ALL10]	binary	one-level	symmetric	*R1, R7*	×	gold-standard
17	MoA [KJH+05]	binary	one-level	symmetric	-	×	user-based
18	OIM-SM [ZRL17]	binary	two-levels	symmetric	*R16*	×	gold-standard
19	OM [GAC10]	binary	two-levels	asymmetric	*R8*	×*	gold-standard
20	OMerSec [MFBB10]	n-ary	two-levels	symmetric	-	×*	×
21	Onto-Integrator [EGED09]	binary	two-levels	symmetric	*R2, R9*	✓	user-based
22	PORSCHE [SBH08]	n-ary	two-levels	symmetric	*R1, R8*	×*	criteria-based
23	Radwan et al. [RPSY09]	binary	one-level	symmetric	*R1, R16*	×	user-based
24	SAMBO [LT06]	binary	one-level	symmetric	*R8, R16*	×	×
25	SASMINT [UA10]	binary	one-level	asymmetric	*R1, R8*	×	criteria-based
26	This thesis	n-ary	two-levels	symmetric/ asymmetric	*R1-R20*	✓	criteria-based

Part II

Approach

3 Solution Overview

Our proposed solution covers in four main directions:

1. **N-ary Merge Method**:
   Our solution is a partitioning-based approach to enable the merging multiple ontologies. It takes as input a set of source ontologies alongside the respective mappings and automatically generates a merged ontology. At first, the n source ontologies are divided into k blocks, and a local refinement is applied to them. After that, the blocks are combined to produce the merged ontology followed by a global refinement.

2. **Merge Requirements**:
   As our solution, we compile a list of Generic Merge Requirements (GMR)s which a merged ontology expects to achieve them. We provided a compatibility checker between the user-selected GMRs by a graph-based theory.

3. **Inconsistency Handling**:
   We propose a Subjective Logic-based solution to handling the inconsistency problem occurring while merging ontologies.

4. **Quality assessment technique**:
   We formulate a set of quality assessment criteria into structural, functional, and usability dimensions in the context of merged ontology.

Figure 3.1 shows a high-level view of these four main components of our solution. Each of these four main components will be discussed in the following chapters.

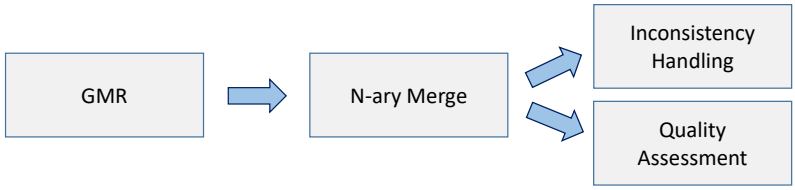


FIGURE 3.1: Solution overview.

4

Multiple Ontology Merging Method

A Partitioning-based Method for N-Ary Strategy Merge

This chapter starts with a general overview of the proposed method in Section 4.1. The preliminaries and basic definitions used in the n-ary method are described in Section 4.2. The architecture of the n-ary method in *CoMerger* is depicted in Section 4.3. This is mainly subdivided into (i) initialization, (ii) partitioning, and (iii) combining phases, described in Section 4.4, Section 4.5, and Section 4.6, respectively. The algorithm of this method is presented in Section 4.7. The list of used notations, symbols, and nomenclatures is in Table 4.1. The contents of this chapter have been previously published in [BKR20a; BKR20b; BALKR17].

TABLE 4.1: The used notations, symbols, and nomenclatures in Chapter 4.

Notation	Description
$\mathcal{O}$	an ontology
e	an entity of an ontology
$\mathcal{E}$	the entities of an ontology
C	a set of classes in an ontology
c	a class in an ontology
P	a set of properties of an ontology
p	a property of an ontology
I	an individual of an ontology
$Sig(\mathcal{O})$	a signature of an ontology
$\mathcal{O}_S$	a set of source ontologies
$\mathcal{O}_M$	a merged ontology
$\mathcal{M}$	a mapping set between the source ontologies
rel	a relation between two entities
cs	a corresponding set between the source ontologies
CS	a set of corresponding between the source ontologies
cs_j^C	a corresponding class
cs_j^P	a corresponding property
$\mathbb{M}$	a map model including a group of correspondences over multiple ontologies
$\equiv$	a correspond operator between entities
$Card(cs)$	a cardinality value of a corresponding set
c_t	a class in the merged ontology
$Conn(c_t)$	number of connections of a class
$\mathcal{L}$	a block
$\mathcal{CL}$	a set of blocks
n	number of source ontologies
k	number of blocks
$\mathcal{I}_M$	the initial merged model
$\mathcal{P}$	a set of pivot classes
$reputation(c_t)$	a reputation degree of a class
$taxo\_rel(c_t)$	taxonomy relation of a class
$non\_taxo\_rel(c_t)$	non-taxonomy relation of a class
w_t	weight degree of taxonomy relation
w_{nt}	weight degree of non-taxonomy relation
$inter\_rel(\mathcal{L}_i, \mathcal{L}_j)$	inter-relatedness degree of two blocks
$dist_{axiom}$	a set of distributed axioms between two blocks
GMR	General Merge Requirement
R1-R20	individual GMRs

4.1 Introduction

Ontologies represent the semantic model of data on the web and are widely developed and reused in different domains. For any given usecase, oftentimes, individual ontologies cover just a part of the domain of interest, or different ontologies exist that model the domain from different viewpoints. In both cases, by merging them into an integrated knowledge graph, their complementarity can be leveraged.

Merging and reusing ontologies becomes increasingly important for several usecases and applications from a wide variety of domains from biomedicine [FFKJ19] and food production [Doo+18] to social networks [PC19] and cultural heritage [ZPVOS18], to name just a few.

Nowadays, the applications of the Semantic Web demand to interoperate with more than two ontologies towards acquiring unified knowledge for researchers, because a large number of ontologies have been developed for a given domain. For instance, there are 188 ontologies available in the *Health* domain, 68 ontologies in the *Anatomy* domain, and 50 ontologies in the *Biological Process* domain in BioPortal[1]. The different ontologies in each category cover particular aspects of the domain of discourse but overlap to a certain degree. For instance, Figure 4.1 demonstrates small views of three biomedical ontologies from BioPortal: *Protein Ontology (PR)*[2], *Adverse Event Reporting Ontology (AERO)*[3], and *Alzheimer's disease ontology (ADO)*[4]. Corresponding classes, extracted by the mapping tool[5] of BioPortal, are represented by dashed lines. Merging these disparate ontologies into a coherent one is a necessary and demanding process in many applications, such as ontology building, extension, and evolution. Using merging methods to create a new ontology is more cost-efficient than building it from scratch [UKMZ98]; and also saves a lot of development effort in ontology reusing [CR16]. Additionally, maintaining the ontology versioning [KF01] can be achieved by an integration process. Another objective of ontology integration can be seen in query answering. For instance, KaBOB [LBBH15] provides a platform for integrated biological data by using ontologies, where biologists have real-time query possibilities without having to know internal structures of data.

As a whole, with the continuously increasing amount of data being produced, developing solutions to deal with the simultaneous merging of multiple ontologies is becoming necessary. Existing ontology merging approaches [GAC10; JERS+11; PC19; PK19; RR14; ZRL17] are limited to merging two ontologies at a time, partly due to using a *binary* merge strategy. In contrast, merging n ontologies ($n > 2$) in a single step, employing what is called an *n-ary* strategy, has not been extensively studied so far. In principle, to merge more than two ontologies, a series of binary merges can be applied incrementally. However, this approach is not sufficiently scalable and viable for a large number of ontologies [Rah16].

[1] https://bioportal.bioontology.org;AccessonMarch2020
[2] https://bioportal.bioontology.org/ontologies/PR
[3] https://bioportal.bioontology.org/ontologies/AERO
[4] https://bioportal.bioontology.org/ontologies/ADO
[5] http://bioportal.bioontology.org/mappings

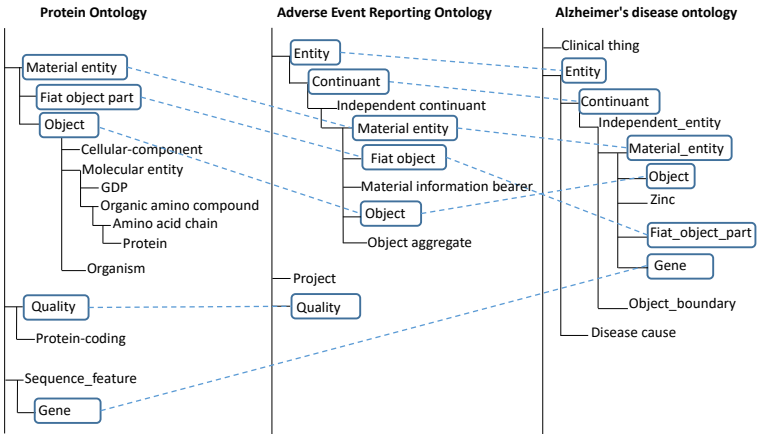


FIGURE 4.1: Example of overlapping bio-ontologies from BioPortal. Dashed lines show corresponding entities.

Thus, the *n-ary* strategy has been introduced as a feasible and efficient method in [Rah16] to overcome the limitations of binary merge. However, due to the much more complex search space, the n-ary strategy remains one of the key challenges in the future research agenda. Mostly, the *n-ary* strategy needs several optimization techniques, especially in two directions: (1) reducing the time needed to complete the task and (2) reducing the computational complexity. The optimization techniques can facilitate the *n-ary* merge process, but the achieved result is expected to have at least the same quality with the binary merge approaches or even improve upon it.

For efficiently applying the *n-ary* method on merging multiple ontologies, we utilize a partitioning-based method. In our *n-ary* method, *CoMerger*, we develop an efficient merging technique that scales to many ontologies. We show that, by using a partitioning-based method, we can reduce the complexity of the search space. In our context, the search space is the set of entities and their relations that have to be evaluated for a specific merge step. Our method takes as input a set of source ontologies alongside the respective mappings and generates a merged ontology. At first, we create $k << n$ blocks populated by partitions of the source ontologies. After that, the blocks are individually merged and refined. Finally, they are combined to produce the merged ontology followed by a global refinement. We provide experimental results in Chapter 10 for merging a variety of ontologies, showing the effectiveness of our approach over pure binary approaches.

4.2 Preliminaries

Before presenting our method, we outline our assumptions about ontologies, ontology matching, and ontology merging.

An ontology is a formal, explicit description of a given domain [Gru+93]. It consists of a finite list of fundamental terms and the relationships between these terms to create the conceptual model of the domain. In the OWL language [MVH+04], ontologies are

presented by a set of axioms, or statements, that are used to describe concepts and the relationships between them. In particular, each axiom makes a statement about the domain of interest. The building blocks of axioms are entity expressions. Entities correspond to the important terms or concepts in the domain and include classes, properties, individuals (instances), and data types. Properties are subdivided into object, data, and annotation properties. Also, the set of entities that appear in the ontology is called the signature of the ontology. In the formal context, we consider the ontology as:

Definition 4.1. *An **ontology** $\mathcal{O} = (\mathcal{T}, \mathcal{A})$ consists of a Tbox $\mathcal{T}$, a finite set of axioms describing constraints on the conceptual schema and an Abox $\mathcal{A}$, which contains assertions about individuals. It contains a set of classes C, properties P, and individuals I, $\mathcal{O} = (C, P, I)$. The signature of the ontology constituting of the entity axioms is indicated by $Sig(\mathcal{O})$.*

We use the term entities $\mathcal{E}$ to refer to the union of the classes, properties, and individuals in the ontology.

The ontology matching process or ontology alignment [RB01] takes a pair of source ontologies as input and produces a group of mappings (matches) between the elements of the source ontologies. The mapping between ontologies includes a set of *corresponding* entities. Formally, we present the mapping (also called match) between ontologies, adapted from [RB01] as follows:

Definition 4.2. *Mapping $\mathcal{M}$ of two ontologies $\mathcal{O}_1$ and $\mathcal{O}_2$ consists of a set of tuples (e_1, e_2, rel, c), where $e_1 \in \mathcal{O}_1$ and $e_2 \in \mathcal{O}_2$, rel describing the relationship between e_1 and e_2, c is a confidence value, usually, a real number within the interval (0, 1].*

The mapping relationship can be one of equality, similarity, or subset (is-a) types. In *CoMerger*, we consider the similarity type with at least a given confidence value.

Separation of Match and Merge. In this thesis, we explicitly separate the ontology matching and merging problems. There have been considerable works on ontology matching as an independent problem (see [RB01; KS03] for some surveys). The successful development of ontology matching systems increases the potential possibility that not only this research but also future studies on ontology merging use the results of these matching algorithms as input. Thus, ontology merging and matching will become more, rather than less, distinct over time. In the literature, some existing ontology merging systems, such as [ZRL17; JERS+11], generate the mapping between the source ontologies themselves. Some others [PB03; RR14] assume the mapping is given. We follow the second group. Since the accuracy of the existing ontology matching tools is relatively high (see the result of OAEI[6] (Ontology Alignment Evaluation Initiative) [AFF+19]), we use the achievements of those systems. In *CoMerger*, we assume that corresponding sets between two ontologies are known. They can be obtained from curated mappings or ontology matching tools.

Since existing mappings are usually generated for pairs of ontologies only, we maintain a map model $\mathbb{M}$ to combine the information across a group of correspondences over multiple ontologies, as follows:

[6] http://oaei.ontologymatching.org/

Definition 4.3. *A **model of mappings** $\mathbb{M}$ over multiple ontologies is built based on the corresponding sets $\mathcal{CS} = \{cs_1, ..., cs_t\}$. Each element of $\mathbb{M}$ holds a set of related corresponding entities between the source ontologies, i.e., $\mathbb{M}_i = \{e_1^{\mathcal{O}_j}, ..., e_d^{\mathcal{O}_h}\}, d \geq 2$, $\mathbb{M}_i \in \mathbb{M}$, $\mathcal{O}_j, \mathcal{O}_h \in \mathcal{O}_S$.*

We use $\equiv$ to indicate that two entities are corresponding to each other. Assuming that the underlying mappings found $e_1^{\mathcal{O}_1} \equiv e_2^{\mathcal{O}_2}$ and $e_1^{\mathcal{O}_1} \equiv e_3^{\mathcal{O}_3}$, we create one entry in $\mathbb{M}$ to combine this information into one corresponding set $cs_j = \{e_1^{\mathcal{O}_1}, e_2^{\mathcal{O}_2}, e_3^{\mathcal{O}_3}\}$ containing all three entities. The cardinality of the corresponding set is given by $Card(cs_j)$. It indicates the number of involved entities in a given $cs_j \in \mathcal{CS}$. In our given example, that is $Card(cs_j) = 3$.

We formulate the ontology merging process, extracted from [PB03], as follows:

Definition 4.4. ***Ontology merging*** *$Merge = (\mathcal{O}_S, \mathcal{M}, \mathcal{O}_M)$ is the process of creating a merged ontology $\mathcal{O}_M$ from a set of source ontologies $\mathcal{O}_S = \{\mathcal{O}_1, ..., \mathcal{O}_n\}$ given a set of corresponding sets $\mathcal{CS}$ extracted from their mappings $\mathcal{M}$, fulfilling a certain set of requirements.*

In an informal way, the ontology merging task is defined as follows: given a set of source ontologies $\mathcal{O}_S$, it creates a new ontology $\mathcal{O}_M$, which is called a merged ontology. $\mathcal{O}_M$ includes some, but not necessarily all entities from $\mathcal{O}_S$, where the corresponding entities are merged into new integrated entities. The merged ontology is expected to fulfill a set of requirements and criteria, assuring its quality. We assume that users will have quality requirements towards the merged ontology as specified in Chapter 5 and that the merging process strives to meet them.

The type of merging process can be symmetric or asymmetric [RR14]. In the symmetric merge, all source ontologies have equal priority. However, in the asymmetric merge, the source ontologies have a different priority and are not considered equally important. In the last case, one of the source ontologies is the target ontology, and the structure of the preferred (target) source ontology should be kept safe in the merged ontology. In this chapter, we assume a symmetric merge strategy.

The partitioning-based technique has been successfully applied by the several ontology matching systems [ADMR05; HQC08; JRASC18]. These systems first partition the source ontologies and then perform a partition-wise matching. Every partition of the first ontology has to be matched with only a subset of the partitions (ideally, with only one partition) of the second ontology. These systems showed a significant reduction of the search space and thus improved efficiency. Moreover, matching smaller partitions reduces the memory requirements compared to matching the full ontologies.

Ontology partitioning vs. module extraction: Our method is based on the ontology partitioning. It differs from the module extraction paradigm. The task of partitioning an ontology is the process of splitting up the set of ontology's axioms into a set of modules so that the union of all modules is semantically equivalent to the original ontology. In contrast, the task of module extraction consists of reducing an ontology to the sub-part, the module, that covers a particular sub-vocabulary. To be precise, for a given ontology and a set of terms from the ontology, a module extraction mechanism returns a module, supposed to be the relevant part of the original ontology that covers the given terms (see [dSSS07] for more comparison).

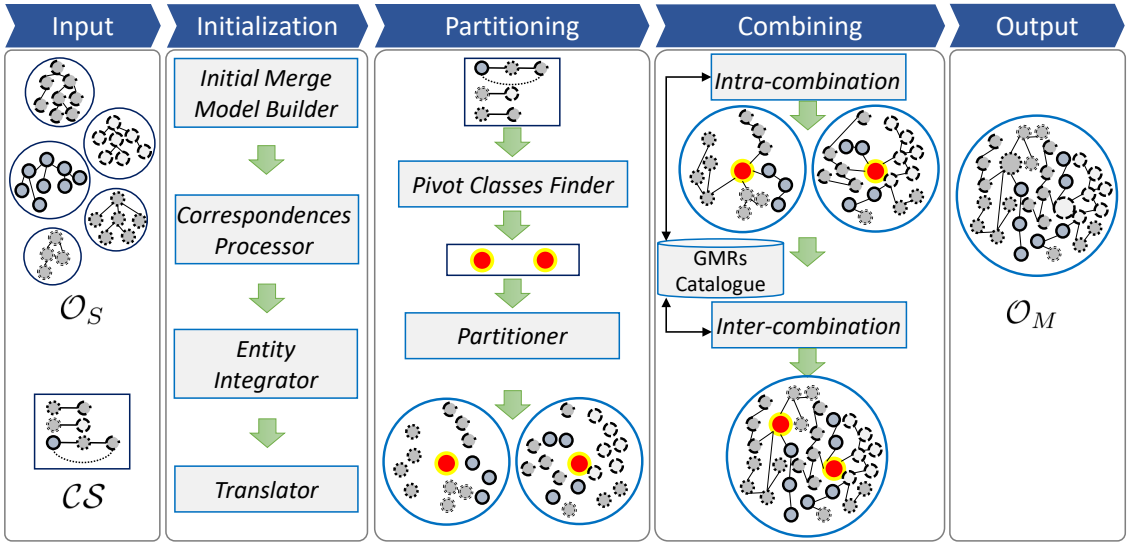


FIGURE 4.2: *CoMerger* architecture.

4.3 The Workflow of N-Ary Merge Method in *CoMerger*

Fig. 4.2 shows the workflow of our proposed n-ary method, which mainly includes:

- The *Input* of this model is a set of source ontologies $\mathcal{O}_S$ alongside the respective mappings $\mathcal{M}$.
- In the *Initialization* phase, the source ontologies $\mathcal{O}_S$ and the corresponding sets $\mathcal{CS}$ extracted from the given mappings $\mathcal{M}$ are processed to construct an initial merge model $\mathcal{I}_M$.
- In the *Partitioning* phase, the initial merge model, constructed upon the n source ontologies, is divided into k blocks based on structural similarity.
- In the *Combining* phase, first, the created blocks are merged and individually refined. Then, they are combined into the merged ontology $\mathcal{O}_M$.
- The merged ontology $\mathcal{O}_M$ is returned as the *Output* of this process.

In the following, we describe each phase in detail.

4.4 Initialization Phase

The *initializer* function takes as input a set of source ontologies with their corresponding sets and generates an initial merge model $\mathcal{I}_M$ (see Equation 4.1).

$$\mathcal{I}_M \Leftarrow initializer(\mathcal{O}_S, \mathcal{CS}) \tag{4.1}$$

Definition 4.5. *An **initial merge model** $\mathcal{I}_M$ contains the intermediate result of the merge process, built upon the source ontologies and their correspondences.*

Our initialization phase is partially similar to the preliminary process in [RR14] with an extension for processing multiple ontologies. The initial merge model (see Definition 4.5) is built through the following processes:

1. **Initial merge model builder**: We build an initial merge model $\mathcal{I}_M$ and parse the source ontologies by extracting corresponding and non-corresponding entities into $\mathcal{I}_M$.
2. **Correspondences processor**: The list of corresponding sets $\mathcal{CS}$ from the given mappings is processed to build the model of mappings $\mathbb{M}$ over multiple ontologies. If several entities from multiple source ontologies correspond to each other, one joined entry for all of them is created in this model.
3. **Entity integrator**: For each entry of $\mathbb{M}$, a new integrated entity in $\mathcal{I}_M$ is created. This means the corresponding entities are combined into a new integrated one and do not individually exist in $\mathcal{I}_M$. If the entities within a single set of correspondences have different labels, the newly generated integrated entity will have multiple labels. For instance, for two corresponding classes c_1 and c_2, the new integrated class, namely $c_{(1)(2)}$, is created in $\mathcal{I}_M$.
4. **Translator**: To construct the initial relations between the entities, we process axioms in $\mathcal{I}_M$. If an axiom's entity has corresponding entities in $\mathbb{M}$, the respective integrated entities will be replaced with the original entity in each axiom. In iPrompt [NM03], different steps have been done on merging classes and properties. Following this strategy, our translator process includes:

   - Translating the class expression axioms based on corresponding classes cs^c_j to establish the relationships between classes: For instance, for merging the corresponding classes c_1 and c_2, whose already combined entity has been integrated in step 3 under the name $c_{(1)(2)}$, all sub and superclasses of c_1 and c_2 are added to the integrated entity $c_{(1)(2)}$. In the same process, all other references of the property axioms from c_1 and c_2 become reference to the integrated entity $c_{(1)(2)}$ and the original references to c_1 and c_2 are deleted.

     Unlike the approach in [NM03] for merging the individuals, we consider the individuals' migration, only, because we assumed that the given correspondences do not include the correspondences between individuals. Thus, this step includes the processing of class assertions axioms in order to migrate the individuals to their respective (integrated) classes. Note that this process also included the translation of the constraint axioms. However, in this step, in the case of any conflicts between the source ontologies' constraint axioms, no conflict solution takes place. It will be done in the refinements step.

   - Translating the property expressions axioms based on corresponding properties cs^p_j to characterize and establish relationships between properties: For instance, to translate the axioms of two corresponding properties p_1 and p_2, whose integrated property $p_{(1)(2)}$ has already been created in step 3, (integrated) domains and ranges of p_1 and p_2 are added to the integrated

property $p_{(1)(2)}$. Similarly, all other (integrated) references from p_1 and p_2 are added to the integrated property $p_{(1)(2)}$ and the original references are deleted.

For two corresponding properties, iPrompt [NM03] suggests merging their respective classes. In this case, iPrompt infers these classes as the new corresponding set. We differ from this approach, in which the classes of the corresponding properties will not merge if they are not included in the given corresponding sets. We narrow our assumption only on the given mappings and do not infer new corresponding sets.

The proposed four steps in this phase generate the initial merge model $\mathcal{I}_M$. This model can be used to derive a merge result in a straightforward manner. *CoMerger* differs from using $\mathcal{I}_M$ alone, by its focus on applying a set of local and global refinements, including, e.g., structural preservation, acyclicity, or constraint and entailments satisfaction to achieve a quality-assured merged ontology. Thus, a set of modeling errors, such as violations of cardinality restrictions or cycles on the class hierarchy, can be handled on the $\mathcal{O}_M$, which shows the trustable result rather than $\mathcal{I}_M$.

4.5 Partitioning Phase

To partition the source ontologies, we use a set of *pivot* classes $\mathcal{P}$. This is inspired by work in [DA07], where a set of predetermined points has been successfully used in the partitioning method. The partitioning process generates ontologies' blocks, with the following definition:

Definition 4.6. *A block $\mathcal{L}$ is a subset (or whole) of one (or more) source ontologies $Sig(\mathcal{L}) \subseteq \bigcup_{i=1}^{n} Sig(\mathcal{O}_i), \mathcal{O}_i \in \mathcal{O}_S$ with no overlap with other blocks.*

We assume:

- Blocks are restricted to do not have overlap and are disjoint, i.e., $Sig(\mathcal{L}_j) \cap Sig(\mathcal{L}_t) = \emptyset, \forall j, t \in \mathcal{CL}, j \neq t$.
- $\mathcal{CL} = \{\mathcal{L}_1, ..., \mathcal{L}_k\}$ is the set of all blocks.
- k represents the number of blocks.

In this regard, the ontology merging task $Merge(\mathcal{O}_1, ..., \mathcal{O}_n)$ is decomposed into a block merging task $Merge(\mathcal{L}_1, ..., \mathcal{L}_k)$. The number of blocks is noticeably smaller than the number of source ontologies, i.e. $k << n$:

$$Merge(\mathcal{O}_1, ..., \mathcal{O}_n) \Rightarrow Merge(\mathcal{L}_1, ..., \mathcal{L}_k) \tag{4.2}$$

Thus, the *Partitioner component* in Figure 4.2 is accelerating the merge process by dividing the entities of the initial merge model $\mathcal{I}_M$ into the k blocks.

4.5.1 Partitioner Goals

Each partitioning method has an objective function or objective goals and criteria to act based on that. In software engineering, the notions of cohesion and coupling have been associated with different aspects of software quality (see [PHZY17] for a survey). The cohesion represents the high relevance of the elements within the same module, while the coupling denotes little relevance of elements across different modules. Similar to software module metrics, ontology module metrics are designed to quantify ontology modules' properties. Thus, the objective of the partitioning phase in *CoMerger* is to maximize intra-block similarity (cohesion) and minimize inter-block similarity (coupling). This indicates that entities within one block are close to each other in terms of structure, while the entities of different blocks are distant from each other. For each block, a sub-ontology will be created in the intra-combination phase in Section 4.6. Then, intra- and inter-similarity can be measured on the sub-ontologies' level. We design our partitioning objective function according to this general goal.

Next, we will discuss our approach to finding pivot classes and the divide method.

4.5.2 Finding Pivot Classes $\mathcal{P}$

Function $PivotFinder$, as shown in Equation 4.3, takes as input the corresponding sets $\mathcal{CS}$ and the source ontologies (to find connectivity degree). It generates a set of pivot classes $\mathcal{P}$ by measuring a value, called $reputation$ degree, for each class of $\mathcal{I}_M$.

$$\mathcal{P} \Leftarrow PivotFinder(\mathcal{CS}, \mathcal{O}_S) \tag{4.3}$$

The $reputation$ degree is built upon two factors:

- **Cardinality degree**: For a given class $c_t \in \mathcal{CS}$, the cardinality degree shows how much the class has overlap with other source ontologies' classes. Classes with high $Card(c_t)$ values in $\mathcal{CS}$ show the best overlap within $\mathcal{O}_S$. Putting a class that has a high number of related corresponding classes in one block can cause more corresponding (overlap) classes to be located in the block. Thus, it increases intra-block similarity and can achieve the objective of our partitioner.
- **Connectivity degree**: A class with high connected entities show the importance of that class in terms of its centrality [ABKD15]. Thus, to calculate the *reputation* degree of the classes, we consider the number of connections of each class, too. Assigning a high connected class in a block can cause more respective connected classes to be located on that block. The reason to consider the connectivity degree alongside with cardinality measure is to avoid creating blocks with small sizes. Since contemplating only the cardinality metric tends to choose isolated classes as the pivot classes. Thus, taking into account the connectivity degree helps to overcome this drawback.

The connectivity degree of a class is indicated by the number of associated taxonomic (subClassOf) and non-taxonomic (semantic) relations for the class in its respective source ontology. For instance, Figure 4.3 shows the taxonomic and non-taxonomic relations for

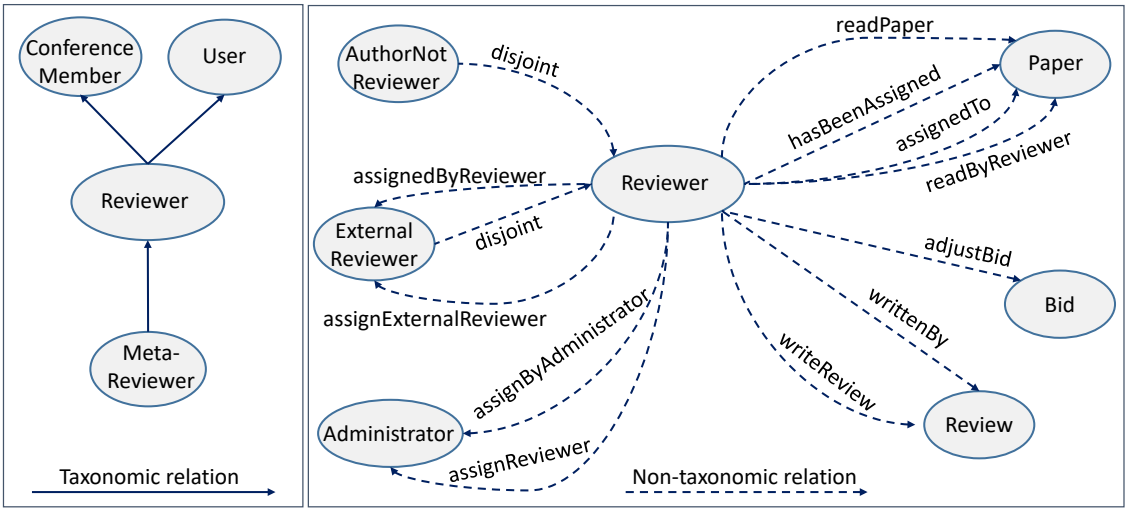


FIGURE 4.3: Taxonomy and non-taxonomy relations for the `Reviewer` class of *cmt* ontology.

the `Reviewer` class of the *cmt* ontology from the *Conference* track[7]. The `Reviewer` class has 3 taxonomic and 13 non-taxonomic relations. This example demonstrates how a class can be augmented with several relationship types. One can assign different weights for taxonomic or non-taxonomic relations based on the source ontologies' nature. Thus, we calculate the connectivity degree of a class $Conn(c_t)$ with user pre-determined weights (w_t and w_{nt}) on taxonomy $taxo\_rel$ and non-taxonomic non_taxo_rel relations, as given by Eq. 4.4.

$$Conn(c_t) = w_t \times |taxo\_rel(c_t)| + w_{nt} \times |non\_taxo\_rel(c_t)| \tag{4.4}$$

Thus, we calculate a *reputation* degree of each class based on the connectivity degree $Conn(c_t)$ and cardinality of corresponding classes $Card(c_t)$ as given in Eq. 4.5.

$$reputation(c_t) = Conn(c_t) \times Card(c_t),\ \forall c_t \in \mathcal{CS} \tag{4.5}$$

The largest sets of corresponding classes in $\mathcal{CS}$ that also have a high number of connections are very promising to be considered as $\mathcal{P}$. In this regard, $\mathcal{P}$ is achieved by a sorted list of $\mathcal{CS}$'s elements based on their reputation degrees.

4.5.3 Partitioner: a Structure Driven Strategy

Function $Partitioner$ (Equation 4.6) takes as input all classes from $\mathcal{I}_M$, divide them based on a structural-based similarity, and returns a set of blocks $\mathcal{CL} = \{\mathcal{L}_1, ..., \mathcal{L}_k\}$ as output.

$$\mathcal{CL} \Leftarrow Partitioner(\mathcal{I}_M, \mathcal{P}) \tag{4.6}$$

[7] `http://oaei.ontologymatching.org/2019/conference/index.html`

Our partitioning method is based on structural-based similarity. Thus, it considers classes close in the hierarchy as strongly related and places them in the same block. Thus, once a class is assigned to a block, all its adjacent classes (on the hierarchy levels of the respective source ontology) consequently will be added. In this regard, the first block $\mathcal{L}_1$ is created by the element of $\mathcal{P}$, which has the highest reputation degree. For each corresponding class $cs_j^c \in \mathcal{P}$, where $cs_j^c = \{c_1^{\mathcal{O}_i}, ..., c_d^{\mathcal{O}_h}\}$, all classes of cs_j^c, i.e., c_1 until c_d with all their adjacent classes on their respective ontologies are added to the block. Then, the next element of $\mathcal{P}$ is selected to create a new block if at least one of its classes has not been assigned to the previous blocks. This process is continued until all the elements of $\mathcal{P}$ are processed. Following this process, the overall number of blocks is automatically determined based on the ratio of the number of $\mathcal{P}$'s elements and the amount of overlap (shared classes) between $\mathcal{P}$'s elements.

The partitioning process is restricted by two assumptions:

- If the taxonomy relation of $\mathcal{P}$'s element is null, no block will be created for it. This condition helps to prevent the creation of very small blocks. Indeed, no other classes will be assigned to this block because the partitioning phase assigns the classes based on taxonomic relations.
- If some unconnected classes are left that do not belong to any block, they will not be added to any block, because they do not require any refinement in the block. However, the unconnected classes will be added directly to $\mathcal{O}_M$.

4.5.4 Partitioning Phase Characteristics - Summary

In this section, we aim to emphasis on the features and characteristics of our proposed partitioning strategy, as follows:

- Merging the n source ontologies is decomposed into the merging k blocks, where $k << n$. This helps to improve scalability and efficiency.
- The partitioning process divides the source ontologies into blocks.
- The partitioning process has low computational complexity since it does not need to run a similarity membership function. Thus, it scales well into a large number of ontologies with many classes.
- The partitioning process utilizes the structural similarity between classes by considering the adjacent relationship between them. Thus, it increases the intra-block similarity of blocks (in terms of hierarchical structure), and decreases the inter-block similarity.
- Using pivot classes accelerates the partitioning process.
- By applying two restrictions, more high-quality blocks can be achieved.

4.6 Combining Phase

In this phase, the created blocks are combined to generate the final merged ontology. To achieve that, we split the combining process into two steps:

4.6.1 Intra-combination: Independent Merge

In this step, all blocks are processed to be merged and refined. Thus, the $intraCombination$ function in Equation 4.7 takes as input the created blocks, and generate k sub-ontologies. Each created sub-ontology is a refined ontology model and can be used separately.

$$subOntologies \Leftarrow intraCombination(\mathcal{CL}) \tag{4.7}$$

Merging a smaller number of blocks reduces the memory requirements compared to merging all source ontologies. This results in a significant reduction of the search space and thus improved efficiency. To further improve performance, the block merging may be performed in parallel. Intra-combination parallelization enables the parallel execution of independently executable mergers to utilize multiple processors for a faster merging process. Thus, in the *intra-combination* step, the entities inside the blocks are combined to create local sub-ontologies. This step requires:

1. Assigning properties to the blocks
2. Applying a set of refinements to the blocks

In the next subsections, we will describe them in detail.

4.6.1.1 Assigning Properties

In the previous subsection, all classes are divided into disjoint blocks. However, these blocks cannot be directly used because the property axioms, which connect these classes, are missing. Thus, we need to add the properties of the classes to their respective blocks and construct the relationships between classes. We retrieve all axioms from $\mathcal{I}_M$, which in this model, the corresponding properties being already translated. Thus, each class is augmented by the original or translated properties axioms, including all taxonomy and non-taxonomy relations. So, in this follow, the taxonomy relations between classes as well as non-taxonomic relations are built for each block. A straightforward and yet effective approach is to assign each axiom to a block in which at least all its entities are contained. To keep the blocks disjoint, each axiom should belong to one and only one block. If the classes of each axiom are distributed across multiple blocks, they are not added to any block and are marked as *distributed axioms* $dist_{axiom}$ (see Definition 4.7).

Definition 4.7. *A **distributed axiom** is an axiom whose entities are distributed among multiple blocks.*

For instance, if $A \in \mathcal{L}_1$ and $B \in \mathcal{L}_2$, then, $A \sqsubseteq B$ will not be added to either $\mathcal{L}_1$ or $\mathcal{L}_2$ and it is marked as a $dist_{axiom}$. Their inclusion is delayed until the next step.

4.6.1.2 Applying Local Refinements

Up to now, the blocks are created based on the classes. They are augmented by the properties. However, each block should be checked for possible errors or conflicts. Thus, in this step, a set of local refinements is applied to the blocks. As a result, each block can achieve higher quality. Through our tool, in addition to the final merged ontology, users access the k created local sub-ontologies separately. Utilizing sub-ontologies rather than source ontologies has the advantage that the created sub-ontologies concisely contain knowledge about a sub-domain (w.r.t. the knowledge provided by the source ontologies) as they include all similar entities. An additional advantage is that maintaining the source ontologies while keeping the existing mapping between them requires much more effort than keeping the k local sub-ontologies (which the existing mappings gathered in one place with limited numbers of mappings between them). So, when local refinements are applied to the blocks, their quality is higher.

4.6.2 Inter-combination: Dependent Merge

In the *inter-combination* step, the global merged ontology $\mathcal{O}_M$ is constructed based on the k created local sub-ontologies. Thus, the $interCombination$ function in Equation 4.8 takes as input the k created sub-ontologies and generates the merged ontology $\mathcal{O}_M$ as the output.

$$\mathcal{O}_M \Leftarrow interCombination(subOntologies) \tag{4.8}$$

To achieve this, we follow a sequential merging process in this step based on calculating an inter-block relatedness degree, which is explained next.

4.6.2.1 Applying a sequential merging

Inter-block relatedness degree represents how much two blocks differ from each other. Thus, we calculate the number of shared distributed axioms $dist_{axiom}$ between two blocks $\mathcal{L}_i$ and $\mathcal{L}_j$ of $\mathcal{L}$ to indicate the inter-block relatedness, as shown in Eq. 4.9.

$$inter\_rel(\mathcal{L}_i, \mathcal{L}_j) = |dist_{axiom}(\mathcal{L}_i) \cap dist_{axiom}(\mathcal{L}_j)| \tag{4.9}$$

First, the two blocks $\mathcal{L}_i$ and $\mathcal{L}_j$ with the highest inter-block relatedness value (most similar blocks) are merged into $\mathcal{L}_{ij}$. This includes adding all distributed axioms to them. Then, the next block, which has the highest inter-block relatedness value with the recently merged block $\mathcal{L}_{ij}$ will be merged. After each block combination, the number of distributed axioms between the recent merged one and the remaining blocks will be updated. This process supports that the highest similar blocks can be executed earlier, and more disjoint blocks are only processed at later steps. Note that, the approach in [ADMR05] also matched only similar blocks and delayed the processing of dissimilar blocks. This sequential execution of the merging processes will be continued until all blocks are merged. If the inter-block relatedness between two blocks is zero, they will

be entirely disjoined and will not need any merge process. Thus, they will be imported directly to the $\mathcal{O}_M$.

The reason to merge the most similar blocks earlier than the lower ones is that the most similar blocks have much more distributed axioms. Combining these blocks in the earlier steps can be more efficient when the blocks are small. Note that, in each sequential merge, the intermediate merged blocks will get larger. So, it is more efficient in the number of processes to have less processing when the blocks get more massive.

4.6.2.2 Applying global refinements

A set of global refinements will be applied to the last combined block. Upon that, in the last step, the merged ontology $\mathcal{O}_M$ is built.

To apply local and global refinements within *CoMerger*, we include a list of General Merge Requirements (GMR)s (see Chapter 5). From GMRs, users can select a subset to be applied (see Appendix B for details of applying GMRs) according to their requirements. Fulfilling each GMR makes sure that the merged ontology meets the chosen GMRs if needed by adapting it, contributing to its refinement. This leads to a generic, flexible, parameterizable merge method. Moreover, the user can easily adjust this framework by performing different refinements in intra- and inter-combination steps.

CoMerger provides a flexible merging approach, where users can actively choose which requirements are essential to them, instead of allowing only a very indirect choice by picking the right merging method, something that is not transparent to the user.

4.6.3 Combining Phase Characteristics- Summary

We summarize and emphasize the characteristics of the combining phase as follows:

- The combining phase merges the created blocks to generate the merged ontology, taking into account the fulfilling the selected refinements and goals.
- The combining phase includes two steps: Intra- and inter-combining processes. The intra-combining process can be run in parallel, to accelerate the speed.
- A two-step refinement process, namely local refinements to the created blocks and global refinements to the merged ontology, is applied. Thus, the quality of the final result will be guaranteed.
- The created sub-ontologies based on the blocks' entities can be used separately. The sub-ontologies hold the most similar entities in one place. Thus, they are representing a richer domain knowledge; they require less maintenance of the inter-mappings.

4.7 N-ary Merge Algorithm

Algorithm 4.1 describes the proposed n-ary merge method. The algorithm accepts a set of source ontologies $\mathcal{O}_S$ with the respective corresponding sets $\mathcal{CL}$ and generates a merged ontology $\mathcal{O}_M$.

Algorithm 4.1: The n-ary merge algorithm for multiple ontologies in *CoMerger*.

Input: a set of source ontologies $\mathcal{O}_S$ and the respective corresponding mappings $\mathcal{CS}$;
Output: a merged ontology $\mathcal{O}_M$;
// Initialization Phase Build an empty initial merge model $\mathcal{I}_M$;
1 Pares source ontologies $\mathcal{O}_S$;
2 Create a map model $\mathbb{M}$ based on given correspondence $\mathcal{CS}$ from $\mathcal{M}$;
3 Integrate the correspondence entities;
4 Translate the axioms;
// Partitioning Phase $\mathcal{P} \Leftarrow PivotFinder(\mathcal{CS}, \mathcal{O}_S)$;
5 $\mathcal{CL} \Leftarrow Partitioner(\mathcal{I}_M, \mathcal{P})$;
// Combining Phase Assign the properties to $\mathcal{CL}$;
6 $subOntologies \Leftarrow intraCombination(\mathcal{CL})$;
7 Apply local refinements on $subOntologies$;
8 $\mathcal{O}_M \Leftarrow intraCombination(subOntologies)$;
9 Apply global refinements on $\mathcal{O}_M$;
10 **return** $\mathcal{O}_M$

First, an empty initial merged model is built (*line 1*). The source ontologies are parsed into $\mathcal{I}_M$ and the map model $\mathbb{M}$ is built (*lines 2-3*). The corresponding entities from $\mathbb{M}$ are integrated in $\mathcal{I}_M$ and the axioms are translated (*lines 4-5*). In the partitioning step, first, the pivot sets $\mathcal{P}$ are detected (*line 6*). Then, the initial merge model $\mathcal{I}_M$ is divided based on $\mathcal{P}$ to create a set of blocks $\mathcal{CL}$ (*line 7*). In the combining phase, first, the properties are assigned to blocks (*line 8*). Then, the intra-combination is applied to blocks to create k sub-ontologies. This process is followed by applying local refinements (*lines 9-10*). After that, the created sub-ontologies are combined in the intra-combination step, followed by applying the global refinements to create the merged ontology (*lines 11-12*). Finally, the merged ontology $\mathcal{O}_M$ is returned to the user (*line 13*).

4.8 Example

Figure 4.4 shows three sample ontologies. The first ontology $\mathcal{O}_1$ has 22 axioms, 8 classes, and 3 properties. The second ontology $\mathcal{O}_2$ has 25 axioms, 8 classes, and 4 properties. The third ontology $\mathcal{O}_3$ has 47 axioms, 13 classes, and 6 properties. The correspondences between the source ontologies (for classes and properties) are shown in Table 4.2. In this section, we detail the process of merging the given source ontologies and their respective corresponding sets. Each following step corresponds to the line numbers of Algorithm 4.1.

- **Step 1**: An empty $\mathcal{I}_M$ is built.
- **Step 2**: The axioms of all three source ontologies are imported in $\mathcal{I}_M$. In this stage, $\mathcal{I}_M$ contains all 22 axioms of $\mathcal{O}_1$, 25 axioms of $\mathcal{O}_2$, and 46 axioms of $\mathcal{O}_3$. All these source ontologies as a whole have 19 taxonomic and 32 non_taxonomic relations.
- **Step 3**: We have 9 pairs of corresponding entities. Thus. the map model $\mathbb{M}$ is built with 8 elements. Since pairs

$Abstract^{\mathcal{O}_1} \equiv PaperAbstract^{\mathcal{O}_2}$ and $PaperAbstrac^{\mathcal{O}_2} \equiv Abstract^{\mathcal{O}_3}$ are considered as one joint entry in $\mathcal{I}_M$. Thus, $\mathcal{I}_M$ includes: $\mathbb{M} = \{(c_6, c_{12}), (c_8, c_{26}), (c_7, c_{13}, c_{25}), (c_{16}, c_{22}), (c_{15}, c_{21}), (p_8, p_{11}), (p_6, p_{12})\}$.

- **Step 4**: The corresponding entities are integrated. Thus, $\mathcal{I}_M$ includes new 8 integrated entities: $c_{(6)(12)}, c_{(8)(26)}, c_{(7)(13)(25)}, c_{(16)(22)}, c_{(15)(21)}, p_{(8)(11)}, p_{(6)(12)}$. The original entities are deleted from $\mathcal{I}_M$. The max cardinality is 3 here.

- **Step 5**: All axioms from the source ontologies (existing in $\mathcal{I}_M$) will be translated if their entities exist in $\mathbb{M}$. For instance, axiom $Student \sqsubseteq ConferenceParticipant$ is replaced with $Student \sqsubseteq ConferenceParticipant\_Member$. Figure. 4.5 shows the translated axioms in $\mathcal{I}_M$. As a whole, 36 axioms out of 93 in $\mathcal{I}_M$ are translated. Indeed, 38.7% of axioms are translated, since the overlap (number of corresponding entities on the total entities) between the source ontologies is relatively high (24.14).

- **Step 6**: To find the pivot classes, we measure the reputation degree of each element of $\mathbb{M}$. The cardinality $Card$, connectivity $Conn$ and the reputation degree of each $cs \in \mathcal{CS}$ are shown in Table 4.3. Based on the reputation degree, the set of pivot classes is ordered as: $\mathcal{P} = \{cs_3, cs_5, cs_1, cs_4, cs_2\}$.

- **Step 7**: $\mathcal{I}_M$'s axioms should be divided into different blocks. To this end, the first element of $\mathcal{P}$, which has the highest reputation degree, is selected to be placed in the first block $\mathcal{L}_1$. Thus, $cs_3 = (c_7^{\mathcal{O}_1}, c_{13}^{\mathcal{O}_2}, c_{25}^{\mathcal{O}_3})$ take place at block $\mathcal{L}_1$. All is-a connected entities of cs_3 elements, i.e., connected entities of c_7, c_{13}, c_{25} are added to $\mathcal{L}_1$ (see $\mathcal{L}_1$ in Figure. 4.6). The next element in $\mathcal{P}$ is cs_5. The entities of cs_5 have not been added in the previous block ($\mathcal{L}_1$) yet. So, a new block should be generated for it. Thus, c_{15} and c_{21} with their connected entities construct block $\mathcal{L}_2$ (see $\mathcal{L}_2$ in Figure 4.6). The next elements of $\mathcal{P}$, i.e. cs_1, cs_4 and cs_2 have been already assigned to a block. Thus, no new block will be generated. As a whole, two blocks $\mathcal{CL} = \{\mathcal{L}_1, \mathcal{L}_2\}$ with 10 and 8 classes are created, respectively. Note that c_{14} is unconnected on the class hierarchy level. Thus, it is not added to any blocks. Moreover, although `Acceptance` and `Rejection` have an is-a connection to `Decision`, but since `Decision` is not connected to any other classes, these three classes can not be added to any blocks.

- **Step 8**: Each block is augmented with the properties. Figure 4.7 shows the blocks with their properties. Properties p_2, p_5 and $p_{(6)(12)}$ are marked as distributed properties and are not added to any blocks. As a whole, there is no is-a distributed axioms between these two blocks. The two is-a axioms of `Decision` are marked as unconnected distributed axioms. There are only 3 non-taxonomic distributed axioms.

- **Step 9**: Two sub-ontologies are built for $\mathcal{L}_1$ and $\mathcal{L}_2$ of $\mathcal{CL}$. The output generation is based on the user-selected format.

- **Step 10**: Both sub-ontologies are checked for the refinements based on the user-selected GMRs. For instance, let us suppose the user selects *R1*, *R15*, and *R16*. In all blocks, these three GMRs are satisfied, so no more refinement takes place.

TABLE 4.2: Corresponding pairs between the source ontologies given in Figure 4.4.

Corresponding Pairs	Corresponding Pairs
$ShortPaper^{\mathcal{O}_1} \equiv Poster^{\mathcal{O}_2}$	$SubjectArea^{\mathcal{O}_1} \equiv Topic^{\mathcal{O}_3}$
$Abstract^{\mathcal{O}_1} \equiv PaperAbstract^{\mathcal{O}_2}$	$PaperAbstrac^{\mathcal{O}_2} \equiv Abstract^{\mathcal{O}_3}$
$ExternalReviewer^{\mathcal{O}_2} \equiv ExternalReviewer^{\mathcal{O}_3}$	$Reviewer^{\mathcal{O}_2} \equiv Reviewer^{\mathcal{O}_3}$
$ConferenceParticipant^{\mathcal{O}_1} \equiv Member^{\mathcal{O}_3}$	$assignExternalReviewer^{\mathcal{O}_2} \equiv invites\_co\_reviewers^{\mathcal{O}_3}$
$isReviewedBy^{\mathcal{O}_2} \equiv isReviewedBy^{\mathcal{O}_3}$	-

TABLE 4.3: A map model $\mathbb{M}$ and its elements with their reputation degree.

cs	$cs's\ elements$	$Card(cs)$	$Conn(cs)$	$reputation(cs)$
cs_1	$c_6^{\mathcal{O}_1}, c_{12}^{\mathcal{O}_2}$	2	$(0.75 \times 1 + 0.25 \times 2) + (0.75 \times 1 + 0.25 \times 1) = 2.25$	$2 \times 2.25 = 4.5$
cs_2	$c_8^{\mathcal{O}_1}, c_{26}^{\mathcal{O}_3}$	2	$(0.75 \times 0 + 0.25 \times 1) + (0.75 \times 1 + 0.25 \times 2) = 1.5$	$2 \times 1.5 = 3$
cs_3	$c_7^{\mathcal{O}_1}, c_{13}^{\mathcal{O}_2}, c_{25}^{\mathcal{O}_3}$	3	$(0.75 \times 1 + 0.25 \times 0) + (0.75 \times 1 + 0.25 \times 0)$ $+(0.75 \times 1 + 0.25 \times 2) = 2.75$	$3 \times 2.75 = 8.25$
cs_4	$c_{16}^{\mathcal{O}_2}, c_{22}^{\mathcal{O}_3}$	2	$(0.75 \times 0 + 0.25 \times 2) + (0.75 \times 1 + 0.25 \times 1) = 1.5$	$2 \times 1.5 = 3$
cs_5	$c_{15}^{\mathcal{O}_2}, c_{21}^{\mathcal{O}_3}$	2	$(0.75 \times 0 + 0.25 \times 4) + (0.75 \times 1 + 0.25 \times 5) = 3$	$2 \times 3 = 6$

- **Step 11**: All refined sub-ontologies should be merged to create $\mathcal{O}_M$. The inter_relatedness degree between two blocks is 3. In this step, the distributed axioms are added to the $\mathcal{O}_M$. The classes which could not be added in Step 7 to any blocks are added now to $\mathcal{O}_M$.
- **Step 12**: The $\mathcal{O}_M$ is checked for the global refinements. *R1* and *R16* are fulfilled in the merged ontology. However, *R15* is not satisfied. Property p_6p_{12} has multiple domains, so the oneness refinement *R15* is applied. Thus, as a possible solution (see Appendix B), we create a new class as the union of all its domains ($Paper$ and $Review$). We then add this new single class as a domain of property p_6p_{12}. As a whole, 1 global refinement action has been done here and no local refinement.
- **Step 13**: The final $\mathcal{O}_M$ is saved based on the user-selected output-format and returned to the user. $\mathcal{O}_M$ has 23 classes, 11 properties, and 93 axioms.

4.9 Summary

In this chapter, we proposed a partitioning-based method for merging ontologies. We used a light-weight partition method with low computational processing to generate blocks of ontologies. Each block is refined and merged separately. Then they combined to produce the final merged ontology. We have provided the workflow, the algorithm, and an example carrying on our method. The main characteristics of our method summarized as:

- The scalability is achieved by decomposing the merging n ontologies into merging k blocks, where the number of blocks is smaller than the source ontologies.
- The partitioning process has a low overhead, but it helps for parallel merging and refining individual blocks.
- Two-step refinements are performed to assure the quality of the merged result.

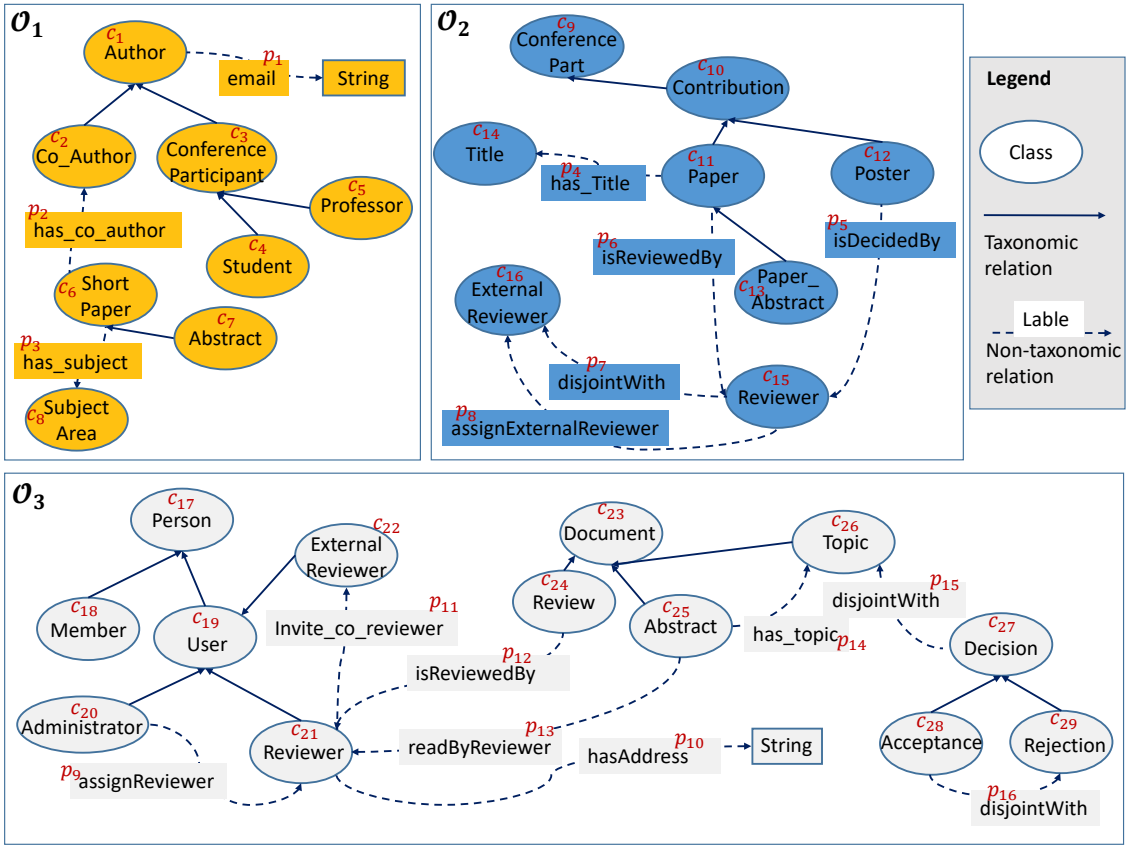


FIGURE 4.4: Three sample source ontologies.

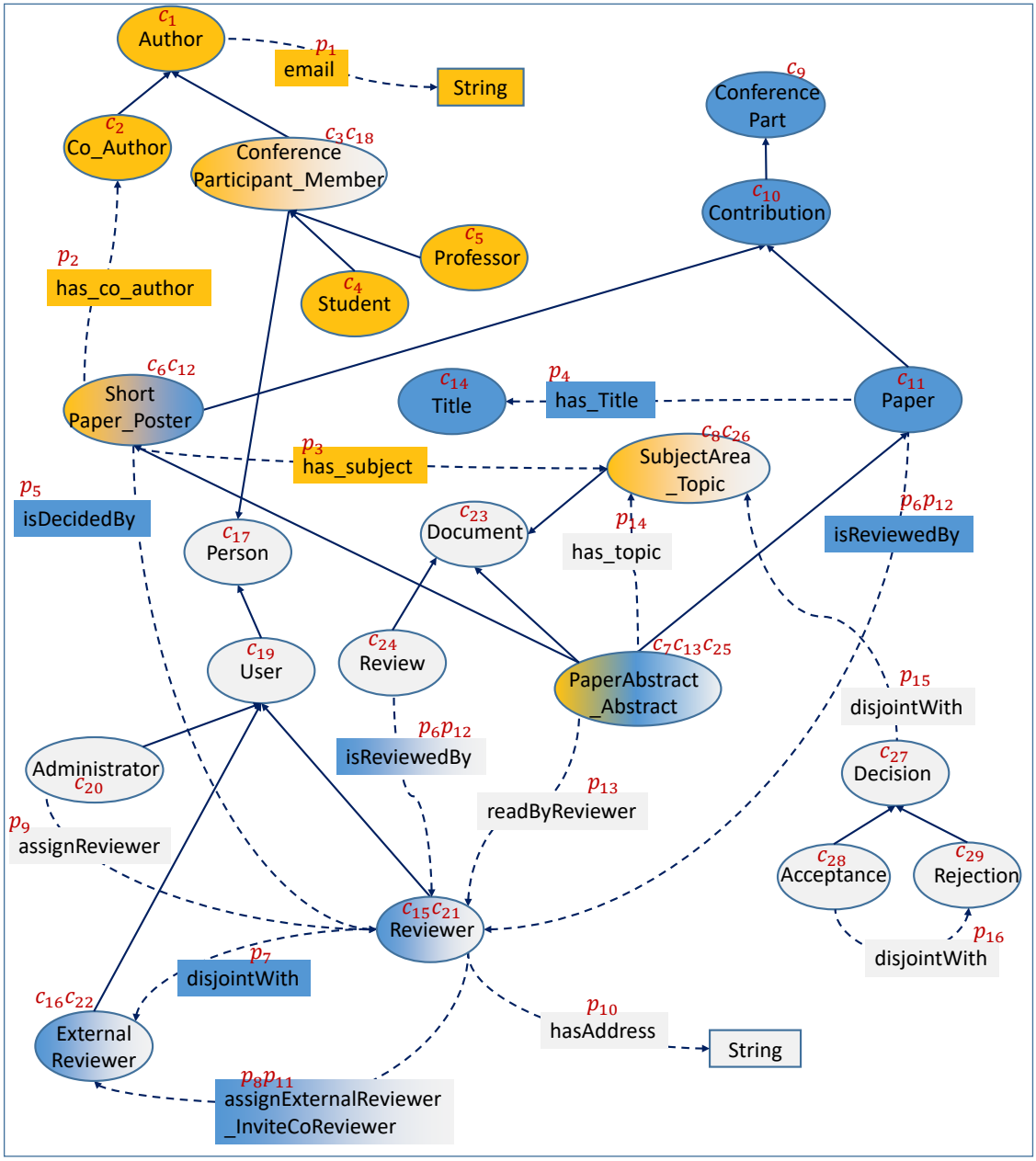


FIGURE 4.5: The initial merge model $\mathcal{I}_M$ for the given source ontologies.

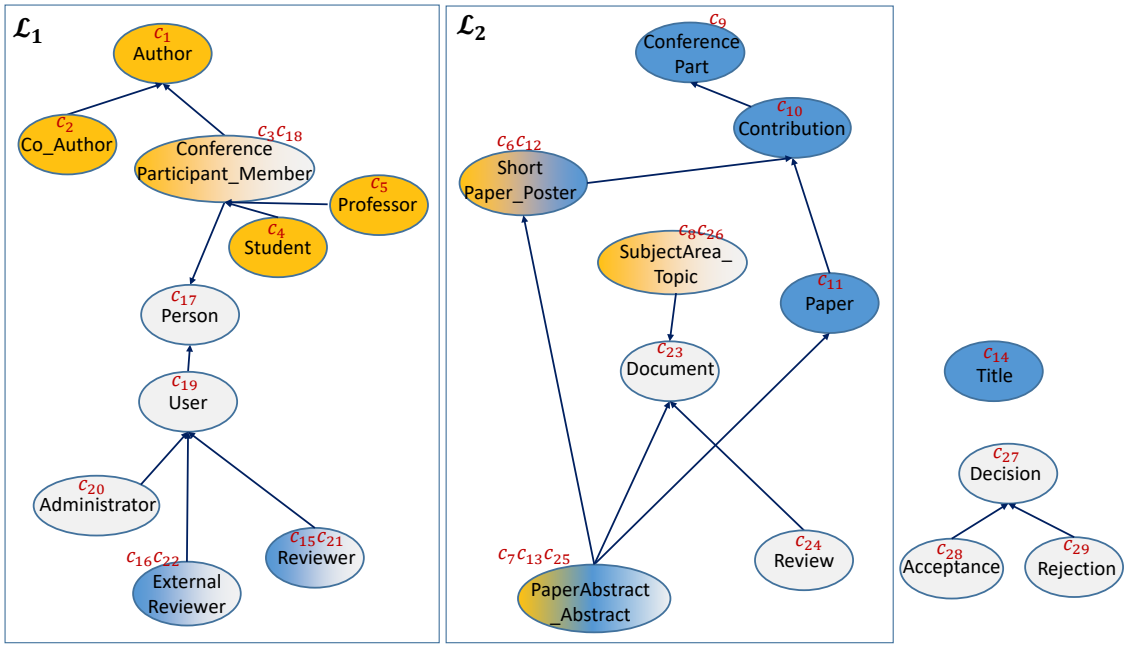


FIGURE 4.6: Generating two blocks for the given source ontologies.

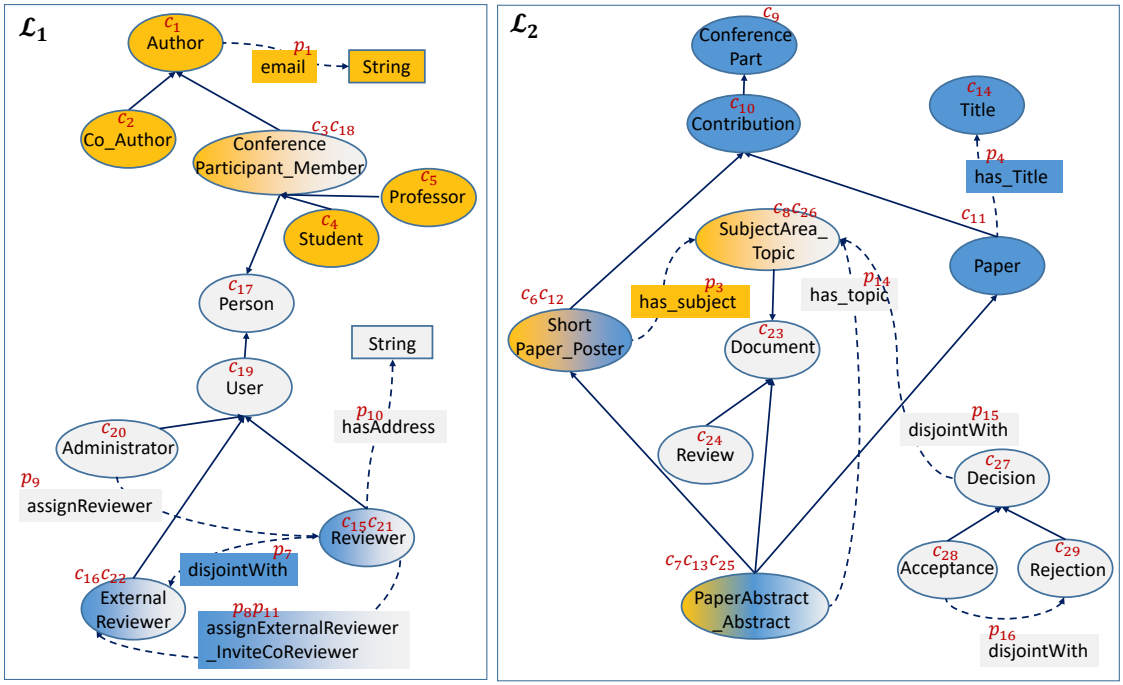


FIGURE 4.7: Augmenting the blocks with non-taxonomy relations.

In Chapter 10, we will present a set of experimental tests on our method.

5 Generic Merge Requirements

This chapter starts with an introduction about GMRs in Section 5.1. The GMR classification and overview are presented in Section 5.2 and Section 5.3, respectively. The compatibility checker framework is presented in Section 5.4, which generates a set of all possible compatible sets based on the user-selected GMRs. We look at the ranking of these sets in Section 5.5. The resolution of conflicts between GMRs has been investigated in Section 5.6. Finally, a summary of this chapter is given in Section 5.7. Moreover, a list of used notations, symbols, and nomenclature is shown in Table 5.1. The contents of this chapter have been previously published in [BGKR20c; BKR19b; GBKR20].

TABLE 5.1: The used notations, symbols, and nomenclature in Chapter 5.

Notation	Description
$\mathcal{O}_S$	a set of source ontologies
$\mathcal{O}_M$	a merged ontology
c_i	a class in an ontology
c'	an integrated class
p_i	a property of an ontology
I_i	a individual of an ontology
$\sqsubseteq$	subClassOf relation between two classes
GMR	Generic Merge Requirements
R1-R20	individual GMRs
$\mathcal{U}$	a set of user-selected GMRs
$\mathcal{U}^C$	a compatible subset of $\mathcal{U}$
$\mathcal{U}^{EC}$	an extra compatible set of GMRs related to $\mathcal{U}$
$\mathcal{G}$	GMRs interactions graph
V	a set of vertices of in the graph
E	a set of edges of in the graph
$\mathcal{K}$	number of vertices in the clique
$\mathcal{K}^C$*-Clique*	compatible clique with $\mathcal{K}$ vertices
$\mathcal{K}^C$*-max-Clique*	compatible clique with maximum $\mathcal{K}$ vertices
$\mathcal{RS}$	a set of compatible sets with $\mathcal{U}$
rs	a compatible set with $\mathcal{U}$
l	number of rs in $\mathcal{RS}$
S_i	scope of changes by a GMR on $\mathcal{O}_M$
$\mu(R_j)$	a set of axioms getting effect by applying $R_j \in$ GMRs on $\mathcal{O}_M$
$\parallel$	compatibility between two GMRs
$\nparallel$	incompatibility between two GMRs
f_d	compatibility degree of a GMR
$\lvert rs_z \rvert$	the number of GMRs in the compatible set rs_z
$\lvert \mathcal{U} \rvert$	the number of GMRs in $\mathcal{U}$
$\lvert \mathcal{U} \cap rs_z \rvert$	the number of GMRs that contains in both rs_z and $\mathcal{U}$
$Score_i(rs_z)$	scoring rs_z
$Total\_Score(rs_z)$	total scoring of ranking the rs_z
$\Psi(\mathcal{U})$	the number of GMRs' aspect in $\mathcal{U}$
$\Psi(rs_z)$	the number of GMRs' aspect in rs_z
$\Psi(\mathcal{U} \cap rs_z)$	the number of common aspects in both rs_z and $\mathcal{U}$
$\lvert GMRs_{Aspect} \rvert$	the total number of aspects of GMRs
$\mathcal{SH}$	Subsumption Hierarchy
$depth(v_i)$	deep of datatype v_i on $\mathcal{SH}$
$\mathcal{RG}$	attributed Restriction Graph for detecting and solving OWL restriction
cases $\mathbb{A}$-$\mathbb{N}$	different solution for restriction conflicts

5.1 Introduction

One aspect that different approaches to merging ontology differ from each other is the set of criteria they aim to fulfill; that is, the requirements that they expect the merged ontology to meet. We analyzed the literature and determined which criteria, here called Generic Merge Requirements (GMRs), are used by different approaches. In summary:

Definition 5.1. *Generic Merge Requirements (**GMRs**) is a set of requirements and criteria that a merged ontology expects to achieve.*

By utilizing GMRs, we can provide a flexible merging approach, where users can actively choose which requirements are important to them, instead of allowing only a very indirect choice by picking a merge system that uses their preferred set of criteria, something that is not transparent to the user. Thus, we take a step toward user-requirement driven ontology merging by allowing the user to customize the GMRs when creating merged ontologies.

Unfortunately, not all GMRs are compatible with each other. For example, one might want to preserve all the classes contained in the original ontologies in the merged ontology. On the other hand, one could wish to achieve class acyclicity. Likely, these goals conflict. Therefore, once the user has selected which GMRs are important, a system is needed that can check whether these GMRs are compatible and can be met simultaneously. In this chapter, we provide the first insight into their compatibility and describe a graph-based method to determine maximum supersets of compatible GMRs. We then rank the suggested sets and return the sorted results to the user.

5.2 GMRs Classification

To discover the most commonly used GMRs, we have studied and analyzed three different areas or researches, including (i) ontology merging methods [DB10; JRGHB09; JERS+11; NM03; PB03; RR14], (ii) ontology merging benchmarks [MFH16; RR12], and (iii) ontology engineering [NM+01; PVSFGP12]. Overall, we classify the GMRs according to three dimensions subdivided into six aspects (see Figure 5.1). The three dimensions we identified are:

- **Integrity**: This dimension refers to the degree of knowledge coverage in the merge process through (i) the *completeness* aspect, and to the amount of knowledge redundancy by (ii) the *minimality* aspect.
- **Model Properties**: Within this dimension, the principles of creating a new ontology model are investigated. In this dimension, (i) *acyclicity*, and (ii) *connectivity* satisfaction aspects are considered.
- **Logic Properties**: The inference of the expected knowledge with involved constraints is analyzed in this dimension. This includes (i) *deduction*, and (ii) *constraint* satisfaction aspects.

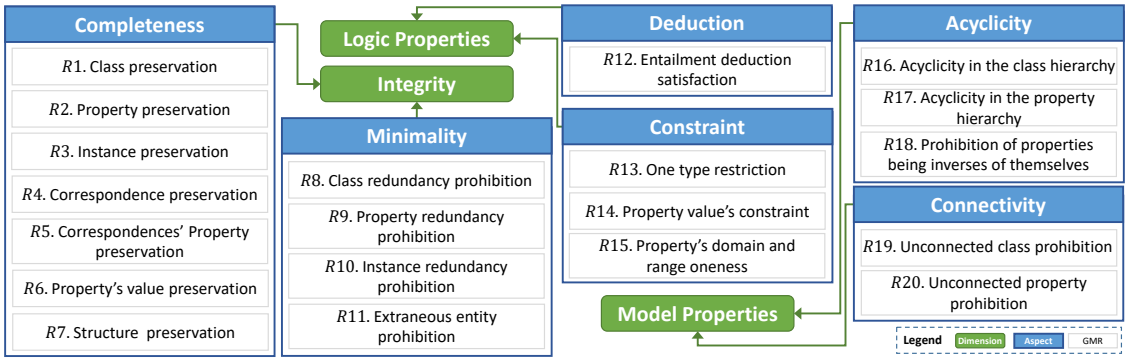


FIGURE 5.1: Generic Merge Requirements (GMRs) classification into three dimensions and six aspects.

TABLE 5.2: Corresponding pairs between the source ontologies given in Figure 5.2.

Corresponding Pairs	Corresponding Pairs
$Review_{c_1}^{\mathcal{O}_1} \equiv Review_{c_{10}}^{\mathcal{O}_2}$	$Reviewer_{c_2}^{\mathcal{O}_1} \equiv Reviewer_{c_{11}}^{\mathcal{O}_2}$
$Document_{c_6}^{\mathcal{O}_1} \equiv ConferenceDocument_{c_{19}}^{\mathcal{O}_2}$	$Paper_{c_5}^{\mathcal{O}_1} \equiv Paper_{c_{13}}^{\mathcal{O}_2}$
$Person_{c_4}^{\mathcal{O}_1} \equiv Person_{c_{12}}^{\mathcal{O}_2}$	$writtenBy_{p_1}^{\mathcal{O}_1} \equiv has\_author_{p_{14}}^{\mathcal{O}_2}$
$has\_id_{p_6}^{\mathcal{O}_1} \equiv has\_id_{p_{16}}^{\mathcal{O}_2}$	-

5.3 GMR Overview

In this section, to explain each GMR in practice, we have designed two sampled source ontologies, as shown in Figure 5.2. The first ontology $\mathcal{O}_1$ has 9 classes (c_1-c_9), 13 properties (p_1-p_{13}), and 4 instances (I_1-I_4). The second source ontology $\mathcal{O}_2$ contains 10 classes (c_{10}-c_{19}) and 12 properties (p_{14}-p_{25}). In both, we include a group of property restrictions alongside with literals. The highlighted entities in each source ontology have corresponding entities from the subsequent ontology. Table 5.2 shows the corresponding pairs between these two ontologies. We have provided two different versions of the merged ontologies to reflect the variety of applying GMRs, as shown in Figure 5.3. Through this example, we will provide explanations of each GMR.

We determined six aspects that GMRs can be grouped into:

Completeness refers to knowledge preservation and coverage:

R1. Class preservation: Each class in (all/target)[1] source ontologies should have a mapped class in the merged ontology [RR14; MFRW00; CKP08; TBL08; FRP14; JERS+11; MTFH14; NM03; MG18b; SBH08; RPSY09; DB10; PB03; MFH16; RR12; UA10; ALL10].

➔ In $\mathcal{O}_{M_1}$, classes c_3, c_{13}-c_{15} are missing. In $\mathcal{O}_{M_2}$, class c_{15} is missing. So, *R1* fulfilled neither in $\mathcal{O}_{M_1}$ nor in $\mathcal{O}_{M_2}$.

[1]Preserving all entities from all source ontologies or a preferred one.

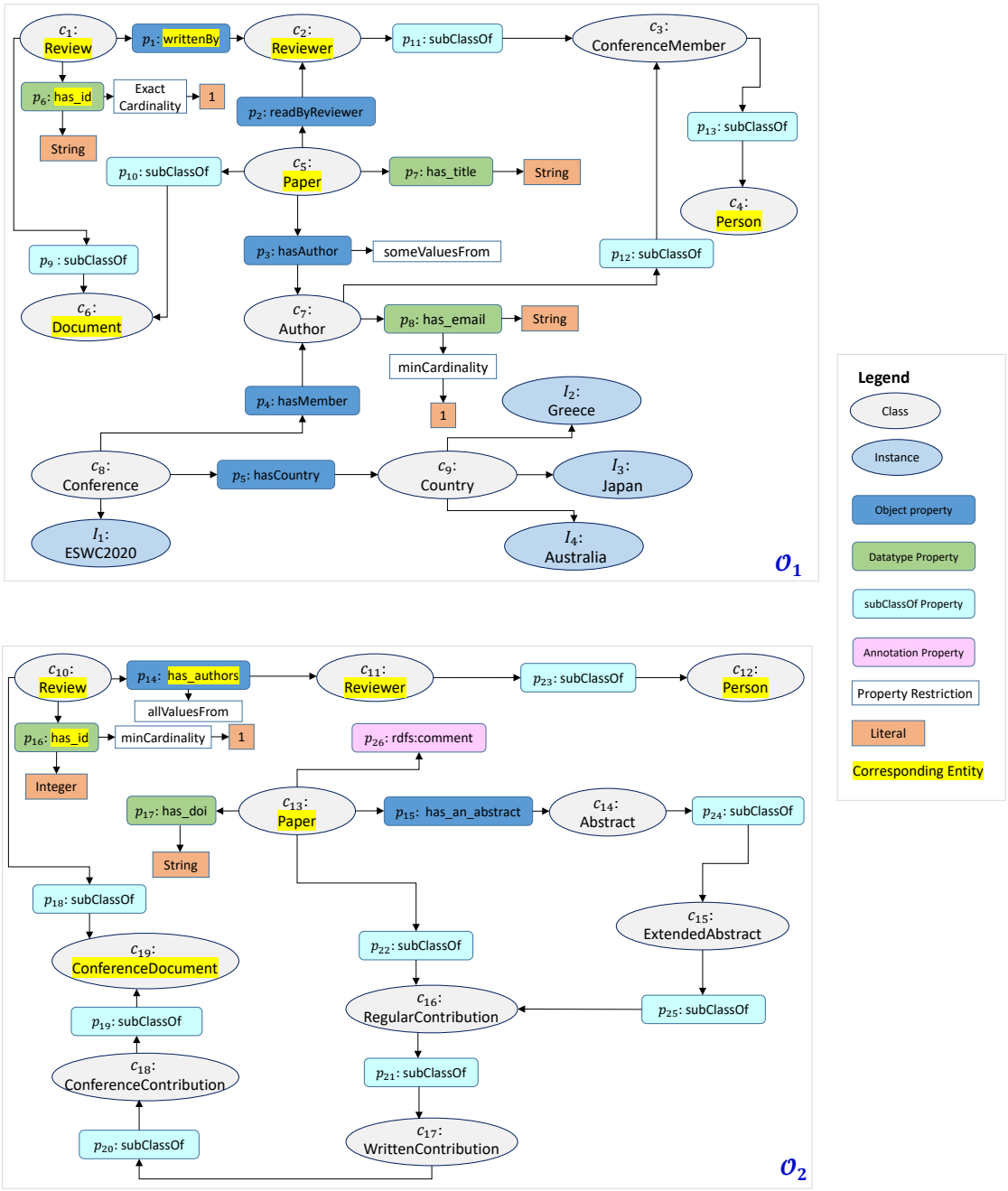


FIGURE 5.2: Two sample source ontologies.

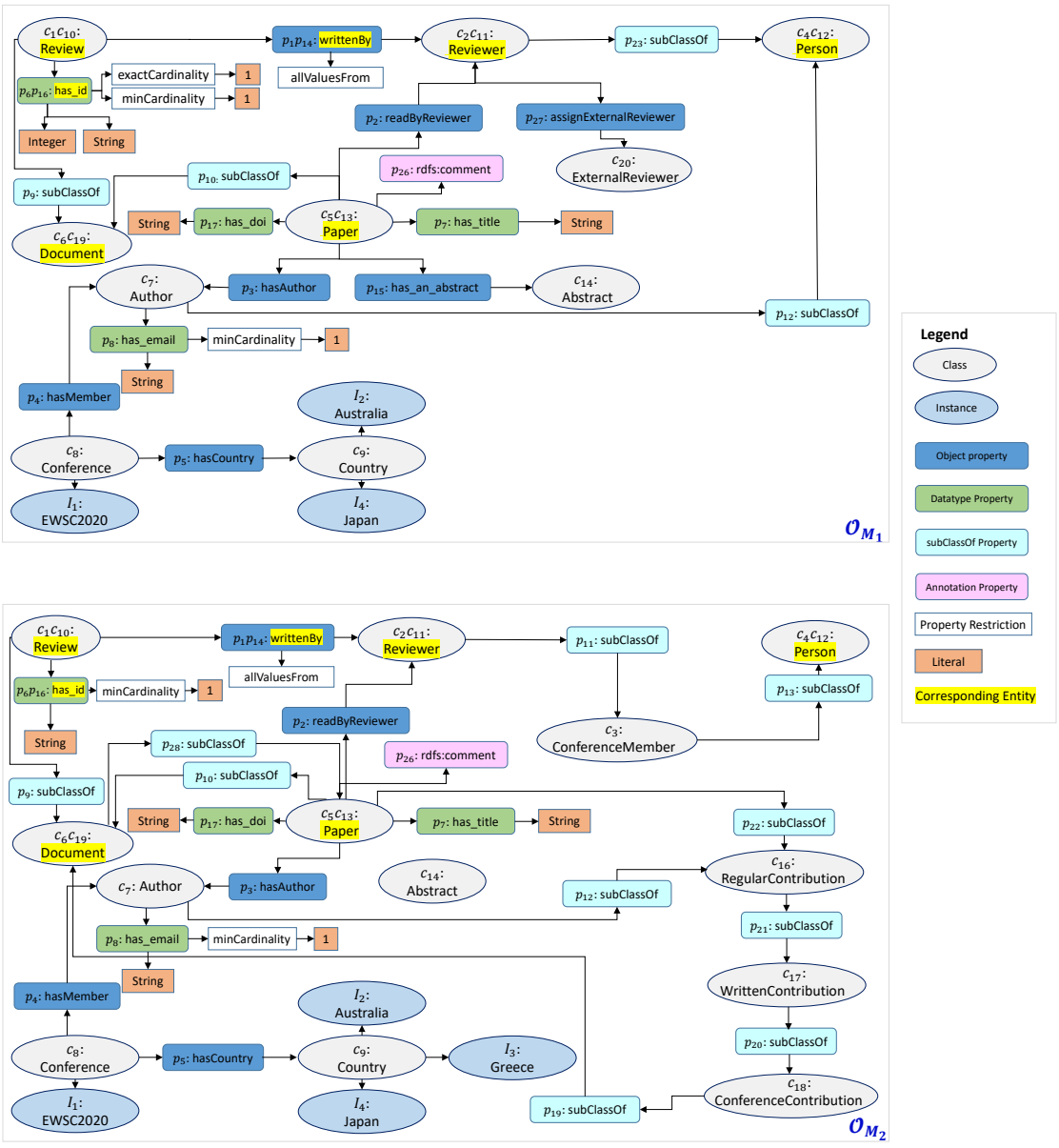


FIGURE 5.3: Two different merged ontologies for the given source ontologies in Figure 5.2.

R2. Property preservation: Each property from the (all/target) source ontologies is explicitly in or implied by the merged ontology [RR14; CKP08; TBL08; FRP14; PB03; EGED09].

➔ In $\mathcal{O}_{M_1}$, properties p_{11}, p_{13}, p_{18}-p_{22}, p_{24}, and p_{25} are missing. In $\mathcal{O}_{M_2}$, properties p_{15}, p_{24}, and p_{25} are missing. So, *R2* fulfilled neither in $\mathcal{O}_{M_1}$ nor in $\mathcal{O}_{M_2}$.

R3. Instance preservation: All instances of (all/target) source ontologies should be preserved in the merged ontology [RR14; CKP08; FRP14; SM01; PB03].

➔ In $\mathcal{O}_{M_1}$, the instance I_2 is missing, whereas $\mathcal{O}_{M_2}$ fulfilled *R3*.

R4. Correspondence preservation: If two entities of the source ontologies are corresponding[2], both should map to the same merged entity in the merged ontology [NM03; PB03; RR14; RR12].

➔ In both $\mathcal{O}_{M_1}$ and $\mathcal{O}_{M_2}$, all corresponding classes and properties are mapped to the same merged entity. Thus, *R4* is fulfilled in both merged ontologies.

R5. Correspondences' property preservation: If any of the corresponding entities from the source ontologies has a certain property, the merged entity should also have this property [NM03; PB03].

➔ For the merged class c_5, in $\mathcal{O}_{M_1}$, the property p_{22} is missing, and in $\mathcal{O}_{M_2}$, the property p_{15} is missing. So, *R5* did not fulfill in $\mathcal{O}_{M_1}$ and $\mathcal{O}_{M_2}$, as the properties of the correspondence classes are missed.

R6. Property's value preservation: Properties' values from the (all/target) source ontologies should be preserved in the merged ontology [NM03; PB03]. In case of conflicts, a resolution strategy is required.

➔ In $\mathcal{O}_{M_1}$, all values from properties are preserved. However, p_6p_{16} faced with a conflict. In $\mathcal{O}_{M_2}$, the value of p_6p_{16} is missing as during the creating the $\mathcal{O}_{M_2}$, the conflicts are solved.

R7. Structure preservation: If two entities are connected via a certain property in a source ontology, their mapped entities in the merged ontology should be connected via the respective mapped property [DB10; ALL10], thus preserving the source ontologies' structures in the merged ontology. In [SJRG14], this GMR is called a *conservativity principle*. It states that the merged ontology should not induce any change in the concept hierarchies of the source ontologies.

➔ In $\mathcal{O}_{M_2}$, class c_7 does not have the same parent as source ontology $\mathcal{O}_1$, as it does not have parent c_3. Thus, *R7* is not fulfilled in $\mathcal{O}_{M_2}$. In $\mathcal{O}_{M_1}$ all classes have the same parents as the source ontologies' structure.

Minimality refers to knowledge redundancy and controlling of semantic overlap:

[2]This can be equality, similarity, or is-a correspondences. In each case, the same type of corresponding should be preserved.

R8. Class redundancy prohibition: A class from the (all/target) source ontologies should have at most one mapping in the merged ontology [TBL08; PK19; PC19; GAC10; SBH08; LT06; DB10; MFH16; PB03; RR14; UA10] .

➔ *R8* is fulfilled as there are no redundant classes in $\mathcal{O}_{M_1}$ and $\mathcal{O}_{M_2}$.

R9. Property redundancy prohibition: A property from the (all/target) source ontologies should have at most one mapping in the merged ontology [MFH16; CKP08; EGED09].

➔ There are no redundant properties in $\mathcal{O}_{M_1}$ and $\mathcal{O}_{M_2}$. Thus, *R9* is fulfilled.

R10. Instance redundancy prohibition: An instance from the (all/target) source ontologies should have at most one mapping in the merged ontology [SM01; MTFH14].

➔ *R10* is fulfilled in $\mathcal{O}_{M_1}$ and $\mathcal{O}_{M_2}$ as there are no redundant instances.

R11. Extraneous entity prohibition: No additional entities other than source ontologies' entities should be added in the merged result [PB03].

➔ In $\mathcal{O}_{M_1}$, property p_{27} and class c_{20} do not belong to the given source ontologies. So, *R11* does not fulfill.

Deduction refers to the deduction satisfaction with *R12*:

R12. Entailment deduction satisfaction: The merged ontology is desirable to be able to entail all entailments of the (all/target) source ontologies [JRGHB09; TBL08]. As the semantic consequences of the integration, it can include more entailments, but it should at least not miss knowledge from the source ontologies.

➔ The subClassOf relation $c_7\ subClassOf\ c_3$, an entailment from $\mathcal{O}_1$ cannot be entailed by $\mathcal{O}_{M_2}$. So, *R12* did not fulfill in $\mathcal{O}_{M_2}$.

Constraint reflects the satisfaction of the ontology constraints:

R13. One type restriction: Two corresponding entities should follow the same data type [PB03]; e.g., if the range of `author_Id` in one of the source ontology is `String` and in the other one is `Integer`, then the range of the merged entity `author_Id` cannot have both types.

➔ In $\mathcal{O}_{M_1}$, there is a conflict between `String` and `Integer` of p_6p_{16}. Thus, *R13* did not fulfill for $\mathcal{O}_{M_1}$, whereas $\mathcal{O}_{M_2}$ fulfills *R13*.

R14. Property value's constraint: If the (all/target) source ontologies place some restriction on a property's values (e.g., in terms of cardinality or by enumerating possible values), this should be preserved without conflict in the merged ontology [PB03; JRGHB09].

➔ The property restriction `someValuesFrom` on p_3, origin from $\mathcal{O}_1$, is missing in both merged ontologies. Moreover, in $\mathcal{O}_{M_1}$, there is a cardinality conflict in p_6p_{16}. So, *R14* did not fulfill in both merged ontologies.

R15. Property's domain and range oneness: The merge process should not result in multiple domains or ranges defined for a single property. This rule is recast from the ontology modeling issues in [PVSFGP12].

➔ *R15* is fulfilled in both $\mathcal{O}_{M_1}$ and $\mathcal{O}_{M_2}$.

Acyclicity refers to controlling the chain problem in the merged ontology:

R16. Acyclicity in the class hierarchy: A cycle of is-a relationships implies equality of all of the classes in the cycle since is-a is transitive. Therefore, the merge process should not produce a cycle in the class hierarchy [CKP08; ZRL17; RPSY09; LT06; DB10; JERS+11; NM+01; PB03; PVSFGP12; RR14; FMB12].

➔ $\mathcal{O}_{M_1}$ fulfilled *R16*, however, in $\mathcal{O}_{M_2}$ there is a cycle as $c_5c_{13} \sqsubseteq c_{16} \sqsubseteq c_{17} \sqsubseteq c_{18} \sqsubseteq c_6c_{19} \sqsubseteq c_5c_{13}$, where $\sqsubseteq$ shows the subClassOf relations.

R17. Acyclicity in the property hierarchy: The merge process should not produce a cycle between properties concerning the is-subproperty-of relationship [PVSFGP12; FMB12].

➔ *R17* is fulfilled in $\mathcal{O}_{M_1}$ and $\mathcal{O}_{M_2}$.

R18. Prohibition of properties being inverses of themselves: The merged process should not cause an inverse recursive definition on the properties [PVSFGP12].

➔ In $\mathcal{O}_{M_1}$ and $\mathcal{O}_{M_2}$, there is no violation of *R18*.

Connectivity refers to the hierarchy connectivity satisfaction:

R19. Unconnected class prohibition: The merge process should not make the classes unconnected [JERS+11; PB03; PVSFGP12]. Every class that had some connections in the source ontologies before the merge process should not be unconnected after the merge process in the merged ontology. In *CoMerger*, we consider only connections on the is-a hierarchy.

➔ In $\mathcal{O}_{M_2}$, class c_{14} is unconnected. Thus, *R19* did not fulfill in $\mathcal{O}_{M_2}$, whereas $\mathcal{O}_{M_1}$ fulfilled it.

R20. Unconnected property prohibition: The merge process should not make the properties unconnected [NM03; PVSFGP12]. Every property that had some connections in the source ontologies before the merge process should not be unconnected after the merge process in the merged ontology.

➔ *R20* is fulfilled in both $\mathcal{O}_{M_1}$ and $\mathcal{O}_{M_2}$.

5.4 Proposed Approach for Checking GMR Compatibility

In the previous section, we introduced a set of GMRs. However, all GMRs are hard to meet simultaneously for two reasons. First, users will not be interested in all of them. Depending on the application they want to create an ontology for, some GMRs might be important to them, while others are trivial. Second, not all GMRs are compatible, e.g., it is difficult to apply acyclicity and property preservation at the same time. Thus, an

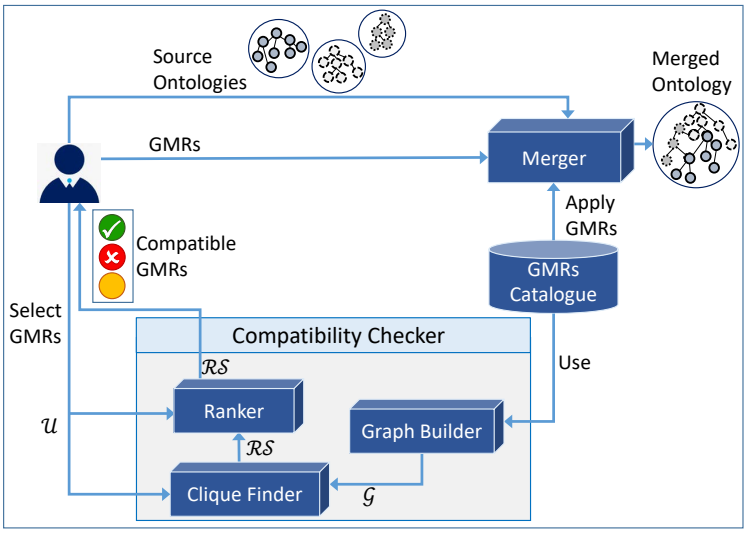


FIGURE 5.4: The workflow of GMRs' compatibility checker within the ontology merge system.

informal definition of (in-)compatibility between two GMRs is given below, which later we specify a formal definition:

Definition 5.2. *Two GMRs are (potentially) compatible with each other if they both can be applied simultaneously at the merged ontology. Otherwise, they are (potentially) incompatible.*

In this section, we describe our approach to finding maximum compatible supersets of user-specified GMR. Basically, what we do is first finding the subsets of GMRs specified by the user that are compatible, and second, extending those by further GMRs (out of the GMRs the user had not selected) while maintaining compatibility. Our intuition is that, first, as much as possible of what the user wanted should be met, and second, adding further GMRs will, in general, improve the quality of the merged ontology.

We propose such a compatibility checker framework within the ontology merge system, as shown in Figure 5.4. The source ontologies are merged based on the user-selected GMRs. Users can ask for the compatibility checker of the selected GMRs. To achieve this, we build a graph $\mathcal{G}$ of the interaction between the GMRs. We then recast the problem at hand to selecting the maximum superset of the user-selected GMRs on the graph. The compatibility checker method takes as input a set of user-selected GMRs, namely $\mathcal{U}$, with the GMRs interaction's graph $\mathcal{G}$. The method uses a clique finder algorithm to detect a set of compatible sets $\mathcal{RS} = \{rs_1, rs_2, ..., rs_z\}$. These results are ranked, sorted, and returned to the user. More precisely, our framework performs the following steps:

- A graph $\mathcal{G}$ is built based on the interactions between GMRs.
- The compatible subsets of the user-selected GMRs are extracted from the $\mathcal{G}$. Then, they will be extended to the maximal compatible superset.
- The detected supersets are ranked and ordered based on criteria.
- An ordered list of ranked compatible sets is returned to the user.

In the next subsections, we explain:

- Building the GMRs interaction's graph $\mathcal{G}$
- Extracting a superset of the graph based on the user-selected GMRs

5.4.1 Building GMRs Interactions Graph $\mathcal{G}$

To build the graph $\mathcal{G}$, we use the *Graph Builder* component in Figure 5.4. It takes as input the GMRs catalogue and creates the graph $\mathcal{G}$. The GMRs' interaction graph $\mathcal{G} = (V, E)$ demonstrates the interaction between GMRs, where V is the set of vertices representing the GMRs (described in Section 5.3), and E is the set of edges. In this graph, two GMRs are connected via an edge if they are compatible. To define the compatibility of GMRs (existence of an edge between two GMRs in $\mathcal{G}$), we set two different conditions:

5.4.1.1 Condition I

The scope of changes by a GMR on the merged ontology can reveal the (in-)compatibility. Two GMRs are compatible with each other if they do not modify the same scope of entities. We distinguish between two scopes:

- **Direct scope**: It is the main scope that is affected by applying a GMR.
  - ➔ *Example*: Applying *R1* adds missing classes, so the direct scope of *R1* is the classes.
- **Indirect scope**: It is the scope that might be affected by the changes made on the direct scope.
  - ➔ *Example*: Applying *R8* deletes the redundant classes (direct scope is redundant classes). However, as a side effect of this operation, this might cause the properties connected to those classes to become unconnected, or their instance to be orphaned. Thus, the indirect scopes of *R8* are properties and instances.

We classify the scopes of changes by GMRs into four categories:

- **Scope 1- Classes**: All classes in the merged ontology, which is the union of:
  - *Scope* 1_1- *Classes origin from source ontologies*: These groups of classes belong to classes of the source ontologies.
  - *Scope* 1_2- *Redundant classes*: These groups are the classes origin from source ontologies, which are repeated.
  - *Scope* 1_3- *Extra classes*: These groups of classes are extra classes that do not belong to any source ontologies. They are created through the merge process.
- **Scope 2- Properties**: All properties in the merged ontology, which is the union of:
  - *Scope* 2_1- *Properties origin from source ontologies*: These groups of properties belong to the properties of the source ontologies.

TABLE 5.3: Scope of changes by applying GMRs in the merged ontology.

Scope	Sub-Scope	Explanation
Scope 1- Classes	Scope 1_1	Classes origin from source ontologies
	Scope 1_2	Redundant classes
	Scope 1_3	Extra classes
Scope 2- Properties	Scope 2_1	Properties origin from source ontologies
	Scope 2_2	Redundant properties
	Scope 2_3	Extra properties
Scope 3- Instances	Scope 3_1	Instances origin from source ontologies
	Scope 3_2	Redundant instances
	Scope 3_3	Extra instances
Scope4- Values of properties	-	Values of properties

  - *Scope* 2_2*- Redundant properties*: These properties origin from source ontologies that are repeated.
  - *Scope* 2_3*- Extra properties*: These groups of properties are extra properties that do not belong to any source ontologies. They are created through the merge process.

- **Scope 3- Instances**: All instances in the merged ontology, which is the union of:
  - *Scope* 3_1*- Instances origin from source ontologies*: These groups of instances belong to instances of the source ontologies.
  - *Scope* 3_2*- Redundant instances*: These instances origin from source ontologies that are repeated.
  - *Scope* 3_3*- Extra instances*: These groups of instances are extra instances that do not belong to any source ontologies. They are created through the merge process.
- **Scope 4- Value of properties**: This scope is related to the values of properties in the merged ontology.

A brief overview of the scopes of applying GMRs has been shown in Table 5.3.

5.4.1.2 Condition II

Let us illustrate our intuition for requiring the second condition with an example by considering the interaction between *R2* (property preservation) and *R5* (correspondences' property preservation). *R2* may make changes to the properties, and *R5* possibly makes changes to the properties of the corresponding classes. So, both GMRs apply changes on the same set of entities, i.e., properties origin from the source ontologies (Scope 2_1). However, it cannot be concluded that both GMRs are incompatible because the operations that both carry on the merged ontology do not have any contradiction. *R2* uses the add operation to preserve the missing properties. *R5* also uses the add operation to add missing properties of the corresponding classes. Therefore,

both these actions can be performed simultaneously in the merged ontologies without conflict. As a whole, three types of operations are performed to meet the GMRs and ensure their fulfillment:

- Add:
  - ➔ *Example: R1* (class preservation) uses the add operation to preserve the missing classes in the merged ontology.
- Delete:
  - ➔ *Example: R8* uses the delete operation to be free of redundant classes.
- A combination of add and delete:
  - ➔ *Example: R4* (correspondence preservation) uses add and delete operations, where for two corresponding classes c_1 and c_2 that are not mapped to the integrated class c', first, c_1 and c_2 should be deleted, then c' added.

Table 5.4 shows the scopes and operations of each GMR. For some GMRs, there may be different possible operations. We followed only one of possible solutions in *CoMerger*, and marked the alternative one by the symbol *.

Although applying each GMR may change direct and indirect scopes, their operations carry on the direct scope, only. Therefore, to determine the compatibility of the GMRs, the type of operations that each GMR performs on the direct scope should be considered. In this regards, when two GMRs change the same set of entities, they can still be compatible if both use the same operation.

Let $\mu(R_j)$ be a set of entities that get affected by applying $R_j \in$ GMRs to the merged ontology, i.e., their direct scope. Given the conditions mentioned above, we define the compatibility of between R_j and $R_k \in$ GMRs as:

Definition 5.3. *R_j is compatible with R_k ($R_j \parallel R_k$) if R_j and R_k modify the different scope of entities in the merged ontology, i.e., $\mu(R_j) \neq \mu(R_k)$. If $\mu(R_j) = \mu(R_k)$, the type of operation of applying R_j and R_k should be the same.*

Accordingly, there could be four variants on the scope of changes and the types of operation, depicted in Figure 5.5, as:

***Case A*- Same Scopes and Same Operations**: In this case, the scope of entities affected by applying R_j and R_k, is the same. Moreover, R_j and R_k use the same type of operations. Since both GMRs use the same operation on the same set of entities, they are compatible with each other.

- ➔ *Example*: *R2* $\parallel$ *R7*.
  R2 makes changes in the properties. *R7* makes changes in the is-a properties. So, these two GMRs address the same set of properties origin from the source ontologies (Scope 2_1). Moreover, *R2* uses the add operation to add the missing properties. *R7* uses the add operation to add the missing is-a properties in order to preserve the hierarchy structure of the source ontologies in the merged ontology.

TABLE 5.4: The scopes and operations of each GMR. The symbol ⋆ indicates an alternative solution.

GMR	Direct Scope	Indirect Scope	Operation	Description
R1	$S1_1$	-	add	It adds missing classes of the source ontologies.
R2	$S2_1$	-	add	It adds missing properties of the source ontologies.
R3	$S3_1$	-	add	It adds missing instances of the source ontologies.
R4	$S1_1$	$S2$ $S3$	add & delete	If two corresponding classes c_1 and c_2 are not mapped to the one integrated class c', first, c_1 and c_2 is deleted, then c' will be added.
	$S2_1$	$S1$ $S3$	add & delete	It follows the procedure the same as the *R4*-scope 1-1 but one the properties.
R5	$S2_1$	-	add	It adds missing properties of the corresponding classes.
R6	$S4$	-	add	It adds missing values of the properties.
R7	$S2_1$	-	add	It adds is-a properties to the respective class.
R8	$S1_2$	$S2, S3$	delete	It deletes redundant classes.
R9	$S2_2$	$S1, S3$	delete	It deletes redundant properties.
R10	$S3_2$	$S1$	delete	It deletes redundant instances.
R11	$S1_3$ $S2_3$ $S3_3$	$S1, S2,$ $S3$	delete	It deletes extra entities.
R12	$S1_1$ $S2_1$ $S3_1$	- - -	add	It adds some entities to achieve the entailment the same as the source ontologies.
R13	$S4$	-	delete	It keeps only one of the data types and deletes the other one.
R14	$S4$	-	delete	It keeps only one value of the property and deletes the other one.
R15	$S1_1$	$S2, S3$	add & delete	It might add multiple domains or ranges as the unionOf to the property.
			delete⋆	It might delete multiple domains or ranges and only keep one of them.
R16	$S2$	$S1$	delete	It might delete some properties to be free of cycles.
	$S1$	$S2, S3$	delete⋆	It might delete some classes to be free of cycles.
R17	$S2$	$S1$	delete	It deletes properties to be free of the cycle on the properties' hierarchy.
R18	$S2$	$S1$	delete	It deletes the inverse of properties.
R19	$S2$	-	add	It might add is-a relations to connect the unconnected classes.
	$S1$	$S2, S3$	delete⋆	It might delete unconnected classes.
R20	$S2$	$S1$	delete⋆	It might delete the unconnected properties.
		-	add	It might use the add operation to connect the unconnected properties to the classes.

Case B- **Same Scopes with Different Operations**: In this case, the set of entities, getting effect by applying R_j and R_k, is the same. But, R_j and R_k use different types of operations. Since both use the different operations on the same set of entities, they are incompatible.

➔ *Example*: *R2* $\nparallel$ *R17*.
Both *R2* and *R17* change properties. *R2* uses the add operation to preserve missing properties, whereas *R17* may delete some properties to achieve acyclicity. Thus, it may happen that applying *R17* reverses the changes made by *R7* and vice versa.

Case C- **Different Scopes with the Same Operations**: In this case, the set of entities, getting effect by applying R_j and R_k, is different. Moreover, both use the same type of operations. Since both GMRs using the same operation but on different sets of entities, they are compatible with each other.

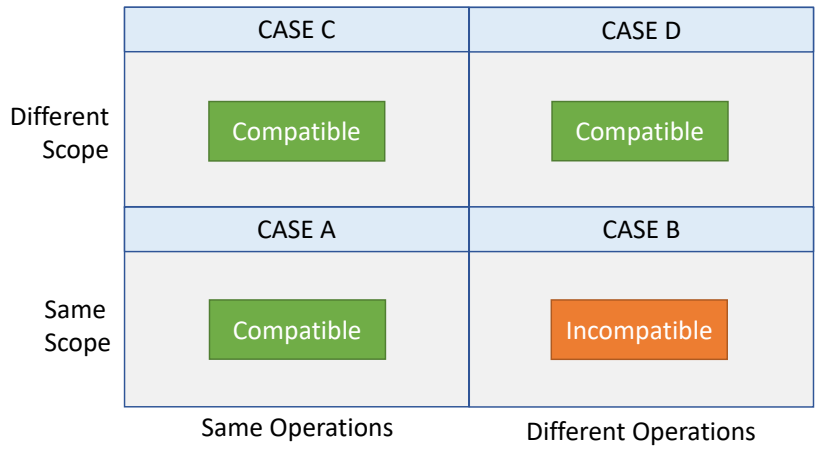


FIGURE 5.5: Compatibility along with four cases by the same or different scopes and operations of applying GMRs.

➔ *Example*: *R1* || *R2*.
Preserving the classes in the merged ontology causes changes in the classes in *R1* (Scope 1_1). However, preserving the properties modifies the properties in *R2* (Scope 2_1). These two GMRs do not change the same group of entities. Moreover, both use the add operation to apply these GMRs. Since both GMRs use the same operation but on different sets of entities, they are compatible with each other.

Case D- **Different Scopes and Different Operations**: In this case, applying R_j and R_k is performed on different sets of entities. Moreover, both GMRs use different types of operations. Since both use different operations on the different entity sets, they are completely separated and do not affect each other. Therefore, they are compatible.

➔ *Example*: *R1* || *R11*.
R1 makes changes in the classes origin from source ontologies (Scope 1_1). *R11*, in addition to changing properties, modifies the extra classes (Scope 1_3). So, the scopes of changes in these two GMRs are on the different entity sets. *R1* uses the add operation, while *R11* uses delete operation. Since both use the different operations on different sets of entities, they are completely separated. So, these two GMRs are compatible with each other.

Considering the scope and the operation of each GMR stated above, we can conclude their interaction. For the GMRs with multiple scopes, Table 5.5 shows the detail of interactions where the effect of the individual scope is considered separately. The concise version of this table is shown in Table 5.6, in which R_j is considered to be compatible with R_k if the intersection of all its sub-scopes is compatible. As mentioned earlier, to build the edges in the GMRs interaction graph $\mathcal{G}$, we use the (in-)compatible interactions, in which there is an edge between two GMRs, if they are compatible. When there is no edge between two GMRs, it indicates the incompatibility. Thus, the graph $\mathcal{G}$ has edges between the compatible GMRs, as has been shown in Table 5.6.

TABLE 5.5: Compatibility interaction between GMRs, when the sub-scopes are considered separately. The last column f_d shows the compatibility degree.

GMR	Compatible GMRs	f_d
R1	*R2, R3, $R4_{(S2_1)}$, R5-R14, R16-R20*	0.94
R2	*R1, R3, $R4_{(S1_1)}$, R5-R15, $R16_{(S2_2,S2_3)}$, $R17_{(S2_2,S2_3)}$, $R18_{(S2_2,S2_3)}$, R19, R20*	0.88
R3	*R1, R2, R4-R20*	1
$\boldsymbol{R4_{(S1_1)}}$	*R2, R3, R5-R11, $R12_{(S2_1,S3_1)}$, R13, R14, R16-R20*	0.91
$\boldsymbol{R4_{(S2_1)}}$	*R1, R3, R6, R8-R11, $R12_{(S1_1,S3_1)}$, R13-R15, $R16_{(S2_2,S2_3)}$, $R17_{(S2_2,S2_3)}$, $R18_{(S2_2,S2_3)}$ $R19_{(S2_2,S2_3)}$, $R20_{(S2_2,S2_3)}$*	0.73
R5	*R1-R3, $R4_{(S1_1)}$, R6-R15, $R16_{(S2_2,S2_3)}$, $R17_{(S2_2,S2_3)}$, $R18_{(S2_2,S2_3)}$, R19, R20*	0.88
R6	*R1-R5, R7-R12, R15-R20*	0.94
R7	*R1-R3, $R4_{(S1_1)}$, R5, R6, R8-R15, $R16_{(S2_2,S2_3)}$, $R17_{(S2_2,S2_3)}$, $R18_{(S2_2,S2_3)}$, R19, R20*	0.79
R8	*R1-R7, R9-R20*	1
R9	*R1-R8, R10-R18, $R19_{(S2_1,S2_3)}$, $R20_{(S2_1,S2_3)}$*	0.94
R10	*R1-R9, R11-R20*	1
$\boldsymbol{R11_{(S1_3)}}$	*R1-R10, R12-R20*	1
$\boldsymbol{R11_{(S2_3)}}$	*R1-R10, R12-R18, $R19_{(S2_1,S2_2)}$, $R20_{(S2_1,S2_2)}$*	0.94
$\boldsymbol{R11_{(S3_3)}}$	*R1-R10, R12-R20*	1
$\boldsymbol{R12_{(S1_1)}}$	*R1-R3, $R4_{(S2_1)}$, R5-R11, R13, R14, R16-R20*	0.94
$\boldsymbol{R12_{(S2_1)}}$	*R1-R3, $R4_{(S1_1)}$, R5-R11, R13-R15, $R16_{(S2_2,S2_3)}$, $R17_{(S2_2,S2_3)}$, $R18_{(S2_2,S2_3)}$, R19, R20*	0.87
$\boldsymbol{R12_{(S3_1)}}$	*R1-R11, R13-R20*	1
R13	*R1-R5, R7-R12, R14-R20*	0.97
R14	*R1-R5, R7-R13, R15-R20*	0.97
R15	*R2, R3, $R4_{(S2_1)}$, R5-R11, $R12_{(S2_1,S3_1)}$, R13, R14, R16-R20*	0.91
$\boldsymbol{R16_{(S2_1)}}$	*R1, R3, $R4_{(S1_1)}$, R6, R8-R11, $R12_{(S1_1,S3_1)}$, R13-R15, R17, R18, $R19_{(S2_2,S2_3)}$, $R20_{(S2_2,S2_3)}$*	0.78
$\boldsymbol{R16_{(S2_2)}}$	*R1-R15, R17, R18, $R19_{(S2_1,S2_3)}$, $R20_{(S2_1,S2_3)}$*	0.87
$\boldsymbol{R16_{(S2_3)}}$	*R1-R15, R17, R18, $R19_{(S2_1,S2_2)}$, $R20_{(S2_1,S2_2)}$*	0.87
$\boldsymbol{R17_{(S2_1)}}$	*R1, R3, $R4_{(S1_1)}$, R6, R8-R11, $R12_{(S1_1,S3_1)}$, R13-R16, R18, $R19_{(S2_2,S2_3)}$, $R20_{(S2_2,S2_3)}$*	0.78
$\boldsymbol{R17_{(S2_2)}}$	*R1-R16, R18, $R19_{(S2_1,S2_3)}$, $R20_{(S2_1,S2_3)}$*	0.94
$\boldsymbol{R17_{(S2_3)}}$	*R1-R16, R18, $R19_{(S2_1,S2_2)}$, $R20_{(S2_1,S2_2)}$*	0.94
$\boldsymbol{R18_{(S2_1)}}$	*R1, R3, $R4_{(S1_1)}$, R6, R8-R11, $R12_{(S1_1,S3_1)}$, R13-R17, $R19_{(S2_2,S2_3)}$, $R20_{(S2_2,S2_3)}$*	0.78
$\boldsymbol{R18_{(S2_2)}}$	*R1-R17, $R19_{(S2_1,S2_3)}$, $R20_{(S2_1,S2_3)}$*	0.94
$\boldsymbol{R18_{(S2_3)}}$	*R1-R17, $R19_{(S2_1,S2_2)}$, $R20_{(S2_1,S2_2)}$*	0.94
$\boldsymbol{R19_{(S2_1)}}$	*R1-R3, $R4_{(S1_1)}$, R5-R15, $R16_{(S2_2,S2_3)}$, $R17_{(S2_2,S2_3)}$, $R18_{(S2_2,S2_3)}$, R20*	0.87
$\boldsymbol{R19_{(S2_2)}}$	*R1-R8, R10-R15, $R16_{(S2_1,S2_3)}$, $R17_{(S2_1,S2_3)}$, $R18_{(S2_1,S2_3)}$, R20*	0.87
$\boldsymbol{R19_{(S2_3)}}$	*R1-R10, $R11_{(S1_3,S3_3)}$, R12-R15, $R16_{(S2_1,S2_2)}$, $R17_{(S2_1,S2_2)}$, $R18_{(S2_1,S2_2)}$, R20*	0.87
$\boldsymbol{R20_{(S2_1)}}$	*R1-R3, $R4_{(S1_1)}$, R5-R15, $R16_{(S2_2,S2_3)}$, $R17_{(S2_2,S2_3)}$, $R18_{(S2_2,S2_3)}$, R19*	0.87
$\boldsymbol{R20_{(S2_2)}}$	*R1-R8, R10-R15, $R16_{(S2_1,S2_3)}$, $R17_{(S2_1,S2_3)}$, $R18_{(S2_1,S2_3)}$, R19*	0.87
$\boldsymbol{R20_{(S2_3)}}$	*R1-R10, $R11_{(S1_3,S3_3)}$, R12-R15, $R16_{(S2_1,S2_2)}$, $R17_{(S2_1,S2_2)}$, $R18_{(S2_1,S2_2)}$, R19*	0.87

The compatibility degree f_d for each GMR R_j is driven by Equation 5.1. $f_d(R_j)$ is the number of compatible GMRs with R_j divided by the total number of GMRs, where it is 35 and 20 in Table 5.5 and Table 5.6, respectively. The last column of these tables shows the compatibility degree of each GMR.

$$f_d(R_j) = \frac{|compatible\ GMR\ with\ R_j|}{|GMR|} \tag{5.1}$$

Figure 5.6 shows the sorted GMRs based on their compatibility degrees when the intersection of sub-scopes is considered. *R3*, *R8*, and *R10* are compatible with all other

TABLE 5.6: Compatibility interaction between GMRs, when the intersection of sub-scopes are considered. The last column f_d shows the compatibility degree.

GMR	Compatible GMRs	f_d
R1	*R2, R3, R5-R14, R16-R20*	0.89
R2	*R1, R3, R5-R15, R19, R20*	0.79
R3	*R1, R2, R4-R20*	1
R4	*R3, R6, R8-R11, R13, R14*	0.74
R5	*R1-R3, R6-R15, R19, R20*	0.79
R6	*R1-R5, R7-R12, R15-R20*	0.89
R7	*R1-R3, R5, R6, R8-R15, R19, R20*	0.79
R8	*R1-R7, R9-R20*	1
R9	*R1-R8, R10-R18*	0.89
R10	*R1-R9, R11-R20*	1
R11	*R1-R10, R12-R18*	0.89
R12	*R1-R3, R5-R11, R13, R14, R19, R20*	0.74
R13	*R1-R5, R7-R12, R14-R20*	0.95
R14	*R1-R5, R7-R13, R15-R20*	0.95
R15	*R2, R3, R5-R11, R13, R14, R16-R20*	0.84
R16	*R1, R3, R6, R8-R11, R13-R15, R17, R18*	0.63
R17	*R1, R3, R6, R8-R11, R13-R16, R18*	0.63
R18	*R1, R3, R6, R8-R11, R13-R17*	0.63
R19	*R1-R3, R5-R8, R10, R12-R15, R20*	0.68
R20	*R1-R3, R5-R8, R10, R12-R15, R19*	0.68

GMRs. *R13* and *R14* have high compatibility as the scope of their changes is different from the others. *R16*, *R17*, and *R18* are the least compatible.

5.4.2 Clique Finder

Given the GMRs interaction graph $\mathcal{G}$ and the set $\mathcal{U}$ containing the GMRs the user is interested in, we aim to find the maximum subset of V containing all vertices out of $\mathcal{U}$ and no incompatible nodes. This may not always be achievable since the user might have chosen incompatible GMRs already. In this case, we search for a maximum subset of V in $\mathcal{G}$ that preserves as many nodes out of $\mathcal{U}$ as possible and contains compatible nodes only. Thus, the *Clique Finder* component in Figure 5.4 takes as input a set of user-selected GMRs $\mathcal{U}$ alongside with the GMRs' interaction graph $\mathcal{G}$ (see Equation 5.2). It returns a set of all possible compatible sets, namely $\mathcal{RS} = \{rs_1, rs_2, ..., rs_l\}$.

$$\mathcal{RS} \Leftarrow Clique\_Finder(\mathcal{G}, \mathcal{U}) \tag{5.2}$$

Each suggested compatible set $rs \in \mathcal{RS}$ contains (all/part) of the user-selected compatible GMRs, and compatible GMRs additionally all other. For the given

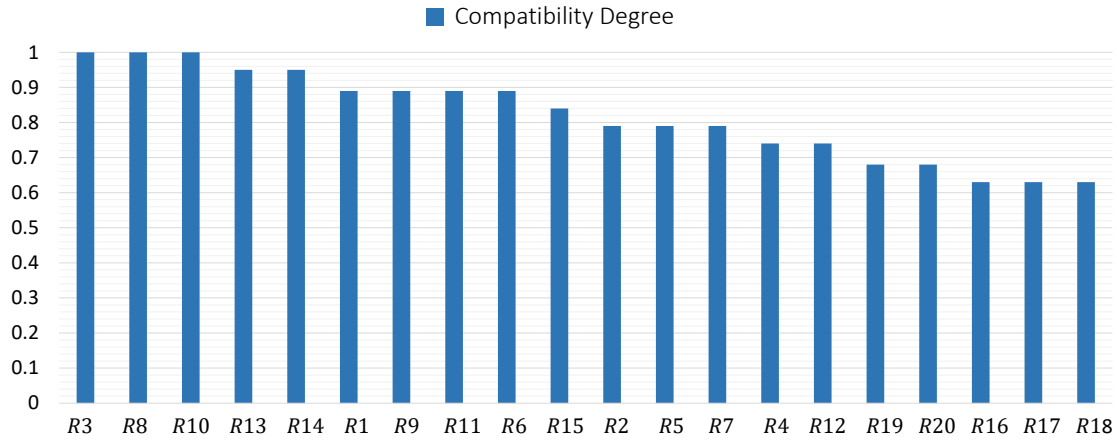


FIGURE 5.6: Sorted GMRs based on their compatibility degree.

user-selected GMRs, $\mathcal{U}$, each suggested compatible set rs is formulated in Equation 5.3.

$$rs = \mathcal{U}^C \cup \mathcal{U}^{EC} \tag{5.3}$$

where, $\mathcal{U}^C$ is a compatible subset of $\mathcal{U}$, and $\mathcal{U}^{EC}$ is an extra compatible set of GMRs related to $\mathcal{U}$. To obtain the compatible set rs, we recast the problem at hand as clique extraction on the GMRs' interaction graph $\mathcal{G}$, where it needs to be the maximal best match based on the user-selected GMRs. A clique is a set of fully connected vertices. We thus extract a compatible clique $\mathcal{K}^C$-*Clique*, where $\mathcal{K}$ indicates the number of vertices in the clique, and C denotes that the clique is compatible.

Definition 5.4. *The $\mathcal{K}^C$-Clique is a compatible clique iff between all vertices only the compatible relations exist.*

Compatible relations between GMRs are encoded by edges in the GMRs interaction graph $\mathcal{U}$. $\mathcal{K}^C$-*Clique* includes compatible GMRs from $\mathcal{U}$ (called $\mathcal{U}^C$) and additional compatible GMRs related to $\mathcal{U}$'s elements (called $\mathcal{U}^{EC}$). $\mathcal{K}^C$-*max-Clique* is a clique containing at least $\mathcal{K}$ vertices that is not a subset of any other cliques. To compute the $\mathcal{K}^C$-*max-Clique*, we use the adapted CLIQUES algorithm described in [TTT06].

To avoid enumerating all possible subgraphs, we set two constraints on the clique extraction:

1. If a clique does not contain at least $\mathcal{K}$ vertices, then neither the clique nor any other sub-cliques can contain a $\mathcal{K}^C$-*Clique*, because, if the clique does not have the required number of vertices, it cannot be a $\mathcal{K}^C$-*Clique*.
2. Only vertices in a $\mathcal{K}^C$-*max-Clique* of $\mathcal{G}$ can form a $\mathcal{K}^C$-*Clique*, because a vertex which is not in a $\mathcal{K}^C$-*max-Clique* cannot be in any $\mathcal{K}^C$-*Clique*.

The first constraint contributes to reducing the search space, and the second one narrows the result to the maximal desired compatible GMRs. Moreover, Definition 5.4 ensures that the selected GMRs are compatible.

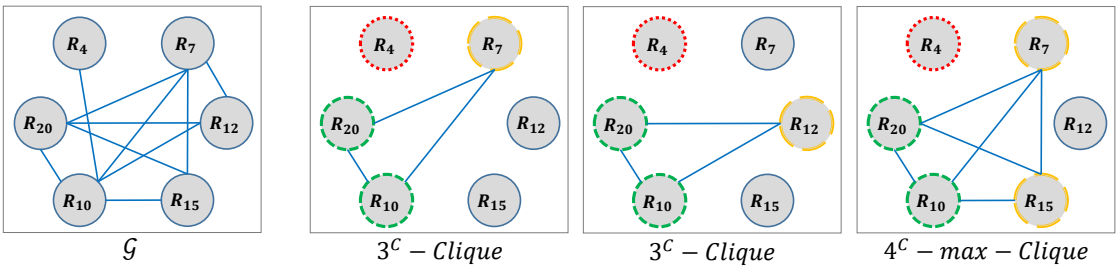


FIGURE 5.7: From left to right: part of GMRs interaction graph $\mathcal{G}$; two different compatible $3^C - Clique$; a compatible 4^C-*max-Clique* for $\mathcal{U} = \{R4, R10, R20\}$. Green: user-selected compatible GMRs; Red: user-selected incompatible GMRs; Orange: extra compatible GMRs.

5.4.2.1 Example

To illustrate the clique finder function with an example, Figure 5.7, left, shows part of the GMRs interaction graph $\mathcal{G}$ for six GMRs. An edge between two GMRs indicates their compatibility. Let us consider that user selects *R4*, *R10*, and *R20*, i.e., $\mathcal{U} = \{R4, R10, R20\}$. For the given $\mathcal{U}$, three different cliques have been shown, in which the user-selected compatible GMRs, the incompatible $\mathcal{U}$'s elements, and the additional compatible GMRs related to $\mathcal{U}$ are indicated with green, red, and yellow circles, respectively.

Two possible presented 3^C-*Clique* are compatible cliques, where two user-selected GMRs are considered, and the incompatible one did not include. However, they are not a maximal clique. The 4^C-*max-Clique* is the best match to the $\mathcal{U}$ and indicates the maximal compatible subset on the sketched $\mathcal{G}$.

5.5 Ranking the Compatible Sets

For each set of user-selected GMRs, there are different possible compatible GMRs sets. Let $\mathcal{RS} = \{rs_1, rs_2, ..., rs_l\}$ be all possible compatible sets based on the user-selected GMRs. To figure out which $rs_z \in \mathcal{RS}$ is the best choice, the *Ranker* component in Figure 5.4 rates the elements of $\mathcal{RS}$ based on different criteria. Therefore, the ranking process assigns a confidence degree to each suggested compatible set.

Assume that the user selected *R7*, *R9*, *R10*, and *R16*, i.e., $\mathcal{U} = \{R7, R9, R10, R16\}$. The algorithm described earlier finds three possible compatible sets, as:

- $rs_1 = \{R1, R3, R6, R8, R9, R10, R11, R16, R17, R18\}$,
- $rs_2 = \{R1, R3, R8, R9, R10, R11, R13, R14, R16, R17, R18\}$, and
- $rs_3 = \{R2, R3, R5, R6, R7, R8, R9, R10, R11, R15\}$.

Thus, $\mathcal{RS} = \{rs_1, rs_2, rs_3\}$[3]. To determine which rs is the best choice, we rank all compatible sets with three different criteria:

[3]For the given $\mathcal{U}$, there are 18 different maximum compatible sets. To make the example concise, we consider 3 compatible sets, only.

1. **The number of user-selected GMRs in each compatible set:** The intersection of the compatible set rs_z and the user-selected GMRs $\mathcal{U}$, $rs_z \cap \mathcal{U}$, comprises all elements which are contained in both rs_z and $\mathcal{U}$. Thus, we count the number of elements that are available in both rs_z and $\mathcal{U}$. Let us consider that

   - $|rs_z|$ is the number of GMRs in the compatible set rs_z,
   - $|\mathcal{U}|$ is the number of GMRs in the user-selected GMRs ($\mathcal{U}$),
   - $|\mathcal{U} \cap rs_z|$ is the number of GMRs contained in both rs_z and $\mathcal{U}$, and
   - $|GMRs|$ shows the total number of GMRs in our system.

   Given these notations, Equation 5.4 ranks each suggested compatible set based on the user preference (in the first part) and the power of the rs_z itself (in the second part of the equation).

$$Score_1(rs_z) = \frac{|\mathcal{U} \cap rs_z|}{|\mathcal{U}|} + \frac{|rs_z|}{|GMRs|} \tag{5.4}$$

   For the given example of this section, rs_1 has $|rs_1| = 10$, $|\mathcal{U}| = 4$, $|\mathcal{U} \cap rs_1| = 3$, and $|GMRs| = 20$. Thus, the $Score_1$ for rs_1 is:

$$Score_1(rs_1) = \frac{|\mathcal{U} \cap rs_1|}{|\mathcal{U}|} + \frac{|rs_1|}{|GMRs|} = \frac{3}{4} + \frac{10}{20} = 1.25$$

   Similarly, this score for rs_2 and rs_3 is $Score_1(rs_2) = 1.3$ and $Score_1(rs_3) = 1.25$, respectively.

2. **The number of user-selected aspects in each compatible set:** GMRs have been categorized in different aspects (see Section 5.3), which users can select. Not only the number of user-selected GMRs has an effect on the ranking of each compatible set rs, but also the user intended aspects should be taken into account. Therefore, the extent to which each suggested compatible set rs_z covers the user's intended aspects should be considered. Let us consider that

   - $\Psi(\mathcal{U})$ is the number of GMRs' aspects in $\mathcal{U}$,
   - $\Psi(rs_z)$ is the number of GMRs' aspects in rs_z,
   - $\Psi(\mathcal{U} \cap rs_z)$ is the number of common aspects in both rs_z and $\mathcal{U}$, and
   - $|GMRs_{Aspect}|$ shows the total number of aspects in the GMRs catalouge.

   Given these notations, Equation 5.5 ranks each suggested compatible set based on the user preference aspect (in the first part) and the power of rs_z's aspect itself (in the second part of the equation).

$$Score_2(rs_z) = \frac{\Psi(\mathcal{U} \cap rs_z)}{\Psi(\mathcal{U})} + \frac{\Psi(rs_z)}{|GMRs_{Aspect}|} \tag{5.5}$$

   In the current example, rs_1 has $\Psi(\mathcal{U}) = 3$, $\Psi(rs_1) = 3$, $\Psi(\mathcal{U} \cap rs_1) = 3$, and $|GMRs_{Aspect}| = 6$. Therefore, the $Score_2$ for rs_1 is:

$$Score_2(rs_1) = \frac{\Psi(\mathcal{U} \cap rs_1)}{\Psi(\mathcal{U})} + \frac{\Psi(rs_1)}{|GMRs_{Aspect}|} = \frac{3}{3} + \frac{3}{6} = 1.5$$

Similarly, this score for rs_2 and rs_3 is $Score_2(rs_2) = 1.67$ and $Score_2(rs_3) = 1.5$, respectively.

3. **Compatibility degree of each GMR:** Up to now, the proposed metrics consider the quantity measure, not quality. This results in obtaining an equal value for those sets that contain the same number of GMRs and aspects and the same number of common GMRs and aspects with the user-selected set. In the running example, there is the same number of GMRs and aspects in rs_1 and rs_3, i.e., $|s_1| = 10$, $|s_3| = 10$, $\Psi(s_1) = 3$, and $\Psi(s_3) = 3$. Also, the number of common GMRs and aspects in these sets with user-selected ones is the same. Therefore, they obtained the same values for $Score_1$ and $Score_2$. However, these two sets are distinct. To reflect the difference between them, the specific characteristics of each GMR belonging to the suggested sets should be considered.

   As an indicator to represent a difference between GMRs, we use the compatibility degree of each GMR, as shown in Table 5.6. For $R_j \in GMRs$, the compatibility degree is the number of compatible GMR with R_j divided by the total number of GMRs, as shown in Equation 5.1. Therefore, the average value of the compatibility degree of each element in the suggested compatible set is used as the third ranking criteria. For the given $rs_z = \{R_i, ..., R_m\}$, the average compatibility degree of Rs in rs_z is shown in Equation 5.6.

$$Score_3(rs_z) = \Sigma_{j=i}^{m} f_d(R_j) \times \frac{1}{|rs_z|} \tag{5.6}$$

   In the given example, $Score_3(rs_1) = 0.845$, $Score_3(rs_2) = 0.86$, and $Score_3(rs_3) = 0.89$.

Thus, the total rank for each rs_z is defined by Equation 5.7.

$$Total\_Score(rs_z) = w_1 \times Score_1 + w_2 \times Score_2 + w_3 \times Score_3 \tag{5.7}$$

Considering empirical values of 0.8, 0.1, and 0.1 for w_1, w_2, and w_3, respectively, the total score for our current example is $Total\_Score(rs_1) = 1.23$, $Total\_Score(rs_2) = 1.29$, and $Total\_Score(rs_3) = 1.24$. The values are normalized between 0 and 1 and presented in the descending order to the user. Figure 5.8 shows all the maximum compatible sets $\mathcal{RS}$ for the given example. The rank values for each set has been normalized and ordered in the GUI.

5.6 Conflict Resolution

In ontologies, the entities can represent real-world constraints, such as type, cardinality, or value restrictions. However, two ontology developers may model the same or overlapping entities to describe the common real-world objects with different

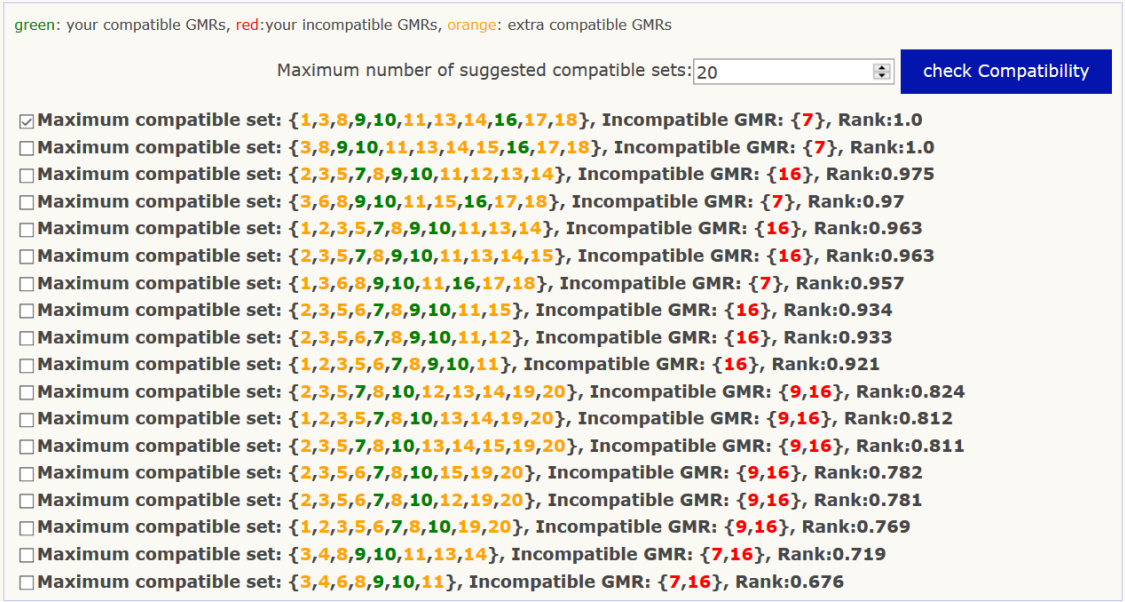


FIGURE 5.8: All maximum compatible sets for the user-selected GMRs {*R7*, *R9*, *R10*, *R16*}.

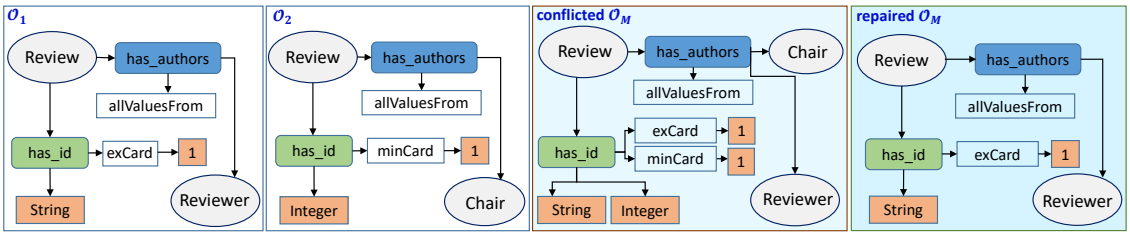


FIGURE 5.9: From left to right: Fragments of two source ontologies $\mathcal{O}_1$ and $\mathcal{O}_2$, the conflicting merged ontology and the repaired one.

restrictions. When two different restrictions are combined in the merged ontology, conflict can happen easily. As we stated before, the conflicts between restrictions in the merged ontology should be resolved. Thus, in this section, we take a deeper look at the resolution of occurring conflict by *R13* (one type restriction) and *R14* (property values' constraint).

5.6.1 Conflicts Occurring by One type Restriction

A datatype property should have at most one range. This has been called the one-type restriction [PB03]. A conflict can happen in the merged ontology when two corresponding entities from different source ontologies have different data types[4]. For example, in the ontology fragments in Figure 5.9, `has_id` from $\mathcal{O}_1$ and $\mathcal{O}_2$ contains two different datatypes: `String` and `Integer`. In the merged result $\mathcal{O}_M$, the two corresponding `has_id` are integrated into one entity. However, the type entities remain

[4]The one type conflict can happen only on datatype properties.

separate, so `has_id` is the origin of two type relationships, which indicates a one type conflict.

The first step toward reconciling one type conflict is to determine which alternative data type can be used in the merged ontology. To this end, we build a *Subsumption Hierarchy* $\mathcal{SH}$ over all supported datatypes in OWL Full. The subsumption relations between the datatypes in $\mathcal{SH}$ are built based on the general data types conversions[5]. Starting from depth zero at the root, the most general datatype is placed at the next level. After that, more precise datatypes are considered.

Upon on $\mathcal{SH}$, the (in-)compatibility between two datatypes can be found as:

Definition 5.5. *Two data types are **compatible** if there is a path in $\mathcal{SH}$ between them that does not go through the root. Otherwise, they are **incompatible**.*

- Conflict solution between two compatible datatypes: Let us consider $depth(v_i)$ shows the level of datatype v_i in the $\mathcal{SH}$. For example, the depth of `Float` in $\mathcal{SH}$ is less than the depth of `Double` because `Double` is more precise than `Float`. In this regard, we define substitution between two datatypes as:

  Definition 5.6. ***Substitution** of two compatible data types $v_i, v_j \in \mathcal{SH}$ with $depth(v_i) < depth(v_j)$, is the type of v_i, since v_i is a more general type in $\mathcal{SH}$.*

  If v_i and v_j are compatible and have the same depth, e.g., are siblings, the substitution is the parent type of both in $\mathcal{SH}$.

- Conflict solution between two incompatible datatypes: If v_i and v_j are incompatible, then no substitution can be performed on them. In this case, we follow the proposed solution, instantiation, in the early work of the schema merging aspects in [BDK92]:

  Definition 5.7. *The **instantiation** between two incompatible datatypes is a new type that inherits from both.*

  This resolution creates an entirely new type that inherits from both data types and replaces the two type-of relationships from the respective property by one type-of relationship to the new type. Therefore, for two contradicting values, an instantiation of them is a new inherited type of both.

The proposed approach is valid when the values of the restrictions are data types, i.e., `String`, `Integer`, `Float`. If they are class types (e.g., Man, Woman, Person), we follow the semantic relatedness strategy, which we will discuss in the next section.

5.6.2 Conflicts Occurring by Property Value's Constraint

An ontology may restrict the maximum or a minimum number of occurrences that a given entity can take part in a relationship or enumerate the possible values of properties. However, when the source ontologies place restrictions on property's values,

[5]We assumed the OWL/RDF data types could be mapped to Java data types and considered the general data types conversions from: `https://docs.oracle.com/javase/specs/jls/se7/html/jls-5.html#jls-5.5`

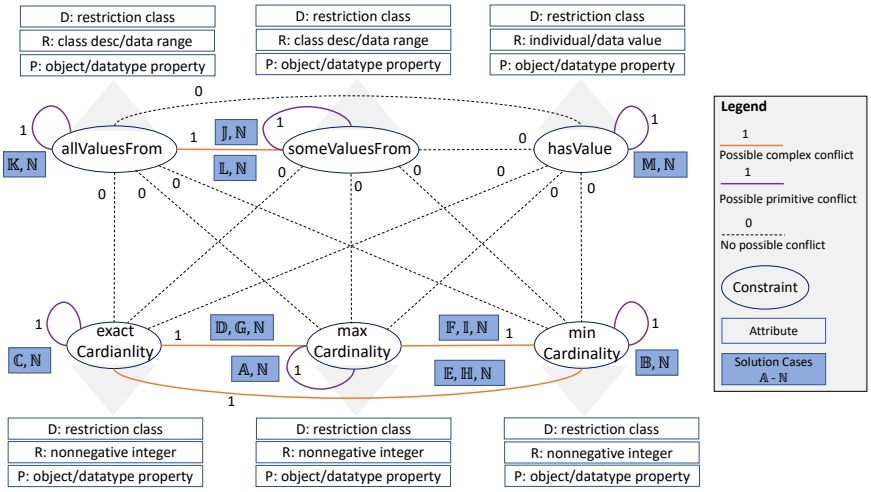


FIGURE 5.10: The attributed Restriction Graph $\mathcal{RG}$ for six OWL Restriction types. Three types of interactions (possible complex conflict, possible primitive conflict, no possible conflict) and adjusted solutions (cases $\mathbb{A}$-$\mathbb{N}$) are presented.

the merged ontology may exhibit conflicts. To detect and reconcile value and cardinality restriction conflicts, we build an attributed Restriction Graph $\mathcal{RG}$ for the six OWL restrictions `allValuesFrom`, `someValuesFrom`, `hasValue`, `exactCardinality`, `maxCardinality`, and `minCardinality`. Figure 5.10 shows the graph $\mathcal{RG} = (V, E)$. It is an undirected labeled graph, where V is a set of vertices, and E is a set of edges. The vertices correspond to the values and cardinality restrictions, while the edges show the interactions between the vertices. Each vertex holds three attributes: Domain (D), Range (R), and the properties (P) on which the constraint can be applied. A constraint links a Domain to a Range and can be applied on object or datatype properties. Domain (D) and Properties (P) attributes for our vertices have the same values. Thus, we construct the edges based on the Range (R) attribute, as given by Definition 5.8 and 5.9. The interactions between vertices can reveal three different states: (1) no conflict (isolated), (2) primitive, or (3) complex conflict.

Definition 5.8. *If the Range (R) attributes of two vertices in the $\mathcal{RG}$ are the same, depending on their values and ranges, there is a possibility of **conflict** for them. However, two restrictions with different Range (R) attributes are **isolated** from each other and can not have any conflicts.*

When there is a possible conflict between vertices, the edge between these two vertices holds label 1. Otherwise, labels of the edges are 0.

Definition 5.9. *A **primitive** conflict is a possible conflict between the same restriction types. A possible conflict over different restriction types is called a **complex** conflict.*

In the $\mathcal{RG}$ depicted in Figure 5.10, all recursive violet-colored edges are types of possible primitive conflicts. Orange edges between two vertices in $\mathcal{RG}$ depict possible complex conflicts. Each primitive or complex conflict on the values or cardinality constraints requires a reconciliation method. We developed such methods and derived detailed

solutions (see Table 5.7) to all 21 interaction restriction cases given by the cases $\mathbb{A}$-$\mathbb{N}$ in Figure 5.10. A summary of the resolution is:

- **Cardinality restriction conflicts solution**: We use the greatest lower and least upper bound methods adapted to the individual cases.
- **Value restriction conflicts solution**: When the value restriction is on a data property, we follow the approach described in Section 5.6.1. If the value restriction is related to an object property, we apply the *semantic relatedness* solution, in this way, if two values are semantically related, following the generalization of them, we choose the super class out of them. If the values are siblings, we select their parent value. When there is no semantic relatedness for two values (i.e., they are not on the same hierarchy), no automatic reconciliation can be made.

5.7 Summary

Each ontology merging system aims to explicitly or implicitly fulfill a set of Generic Merge Requirements (GMRs) that their merged ontologies should meet. In this chapter, we contribute to analyzing them. In particular:

- We provide a comprehensive list of GMRs in the literature, classified and adapt them in the context of the ontology merging domain.
- Since not all GMRs can be fulfilled at the same time, we propose a graph-based framework to determine the GMRs compatibility interaction systematically. The intuition behind using the graph theory is to facilitate the encoding of the GMRs' compatibility via the graph presentation and reveal the other possible compatible requirements.
- We propose an automatic ranking method to rates the suggested compatible sets. The sorted list of them is returned to the user. It helps users to select their appropriate set easily among all possible existence sets.
- Our proposed framework allows users to specify the most important GMRs for their specific task at hand and to detect a maximum compatible superset. This result can then be used to select an appropriate merge method or to parameterize a generic merge method.
- We tackle (i) one type conflicts by building a subsumption hierarchy on data types and performing substitution or instantiation on them, (ii) cardinality restriction conflicts with least upper and greatest lower bound method, (iii) value restriction conflicts by utilizing the semantic relatedness.
- The proposed framework for checking the compatibility between the GMRs can be easily extended for the newly embedded GMRs, where building the GMR interaction's graph and obtaining their compatibility can be performed in the same procedure for the new adapted GMRs.
- We embed the GMRs within the *CoMerger* system, where the users can access to the logged information of applying GMRs on their merged ontologies.

TABLE 5.7: Conflicts between six OWL restrictions. Case $\mathbb{A}$-$\mathbb{N}$ corresponds to solution cases in Figure 5.10.

Case	Appearance	Conflict Type	Solution
$\mathbb{A}$	$value(p^i_{maxCardinality}) \neq value(p^j_{maxCardinality})$	primitive	Taking the greatest lower bound to cover both
$\mathbb{N}$	$value(p^i_{maxCardinality}) = value(p^j_{maxCardinality})$	no conflict	-
$\mathbb{B}$	$value(p^i_{minCardinality}) \neq value(p^j_{minCardinality})$	primitive	Taking the least upper bound to cover both
$\mathbb{N}$	$value(p^i_{minCardinality}) = value(p^j_{minCardinality})$	no conflict	-
$\mathbb{C}$	$value(p^i_{excatCardinality}) \neq value(p^j_{excatCardinality})$	primitive	No automatic solution, applying user's preference
$\mathbb{N}$	$value(p^i_{excatCardinality}) = value(p^j_{excatCardinality})$	primitive	-
$\mathbb{D}$	$value(p^i_{maxCardinality}) = value(p^j_{excatCardinality})$	complex	Taking the exact number to cover both
$\mathbb{E}$	$value(p^i_{minCardinality}) = value(p^j_{excatCardinality})$	complex	Taking the exact number to cover both
$\mathbb{F}$	$value(p^i_{minCardinality}) = value(p^j_{maxCardinality})$	complex	Taking the least upper bound from minCardinality and greatest lower bound from maxCardinality to cover both
$\mathbb{G}$	$value(p^i_{maxCardinality}) \neq value(p^j_{excatCardinality})$	complex	Taking the exact number if they are in the same range, otherwise no automatic solution
$\mathbb{H}$	$value(p^i_{minCardinality}) \neq value(p^j_{excatCardinality})$	complex	Taking the exact number, if they are in the same range, otherwise no automatic solution
$\mathbb{I}$	$value(p^i_{minCardinality}) \neq value(p^j_{maxCardinality})$	complex	Taking the least upper bound from minCardinality and greatest lower bound from maxCardinality to cover both
$\mathbb{J}$	$value(p^i_{someValuesFrom}) \neq value(p^j_{someValuesFrom})$	primitive	for object property: utilizing semantic relatedness, taking superclass; for datatype property: utilizing subsumption hierarchy, taking more general type
$\mathbb{N}$	$value(p^i_{someValuesFrom}) = value(p^j_{someValuesFrom})$	no conflict	-
$\mathbb{K}$	$value(p^i_{allValuesFrom}) \neq value(p^j_{allValuesFrom})$	primitive	Applying the same solution as Case $\mathbb{J}$
$\mathbb{N}$	$value(p^i_{allValuesFrom}) = value(p^j_{allValuesFrom})$	no conflict	-
$\mathbb{L}$	$value(p^i_{allValuesFrom}) \neq value(p^j_{someValuesFrom})$	complex	Applying the same solution as Case $\mathbb{J}$
$\mathbb{N}$	$value(p^i_{allValuesFrom}) = value(p^j_{someValuesFrom})$	no conflict	-
$\mathbb{M}$	$value(p^i_{hasValue}) \neq value(p^j_{hasValue})$	primitive	No automatic solution, applying user's preference
$\mathbb{N}$	$value(p^i_{hasValue}) = value(p^j_{hasValue})$	no conflict	-

6 Handling Inconsistencies

In this chapter, first, we provide an introduction to inconsistency handling in the ontology merging process in Section 6.1. In Section 6.2, we present the preliminaries along with an example of an inconsistent merged ontology and the Subjective Logic theory. We then illustrate our strategy on applying Subjective Logic to handle inconsistencies of the merged ontology in Section 6.3. We demonstrate the whole workflow and the algorithm of our approach in Section 6.4 and Section 6.5, respectively. In Section 6.6, we provide concise insights into the creation of a repair plan. Finally, an illustrative example of applying our method and a summary of this chapter are given in Section 6.7 and Section 6.8, respectively.

Moreover, a list of used notations, symbols, and nomenclature is presented in Table 6.1. The contents of this chapter have been previously published in [BKR19c; BKR19a].

TABLE 6.1: The used notations, symbols, and nomenclature in Chapter 6.

Notation	Description
$\mathcal{O}_S$	a set of source ontologies
$\mathcal{O}_i$	one of the source ontologies
$\mathcal{O}_M$	a consistent merged ontology
$\mathcal{O}'_M$	an inconsistent merged ontology
$\mathcal{M}$	a mapping set between the source ontologies
n	the number of source ontologies
$\mathcal{P}$	a proposition
x	an axiom
X	a set of axioms
$\overline{X}$	the trustworthiness of axiom sets
e	an entity of an ontology
w	an opinion
$w_x^{\mathcal{O}_i}$	an opinion of axiom x from $\mathcal{O}_i$
$b_x^{\mathcal{O}_i}$	a belief of axiom x from $\mathcal{O}_i$
$d_x^{\mathcal{O}_i}$	disbelief of axiom x from $\mathcal{O}_i$
$u_x^{\mathcal{O}_i}$	an uncertainty of axiom x from $\mathcal{O}_i$
$a_x^{\mathcal{O}_i}$	the atomicity of axiom x from $\mathcal{O}_i$
r	number of positive observations
$r_x^{\mathcal{O}_i}$	number of positive observations about x by $\mathcal{O}_i$
s	number of negative observations
$s_x^{\mathcal{O}_i}$	number of negative observations about x by $\mathcal{O}_i$
A	an agent
$\mathcal{C}_{un}$	a set of unsatisfiable concept in $\mathcal{O}'_M$
$Root_{\mathcal{C}_{un}}$	a set of root unsatisfiable concepts in $\mathcal{O}'_M$
$\mathcal{J}$	a set of justifications
$\mathcal{J}_d$	a justification
$\models$	entail
$\not\models$	not entail
$\subseteq$	subset relationship
$\subsetneq$	proper (or strict) subset relationship
$\sqsubseteq$	a subclass relationship between two classes
$\Psi_{x_j}(\mathcal{J})$	axiom frequency for x_j in $\mathcal{J}$
$\Psi_{\mathcal{O}_j}(\mathcal{J})$	number of conflicting axiom sets belong to $\mathcal{O}_i$
$\Gamma(\mathcal{O}_i)$	total number of axioms in $\mathcal{O}_i$
$\Gamma_{x_j}(\mathcal{O}_i)$	number of axioms in $\mathcal{O}_i$ containing elements of x_j
$f_{x_j}(\mathcal{O}_i)$	the fraction of axioms in $\mathcal{O}_i$ containing elements of x_j
α	the provenance of the axioms
β	the provenance of the elements of the axioms

6.1 Introduction

Ontologies reflect their creators' view on the domain and are thus somewhat subjective. In the ontology merging process, even if the source ontologies are consistent, the resulting merged ontology may be inconsistent due to differing world views encoded into the source ontologies. These inconsistencies need to be resolved if one wants to make use of the merged ontology. The resolution is, however, a challenging problem.

Most studies on inconsistency handling of ontologies (cf. [HVHH+05; KPSCG06; LPSV06; SPF+12]) have assumed an isolated environment that an ontology is created stand-alone from other existing ontologies. Whereas, in the context that an ontology is built based on the existing ontologies, such as during the ontology merging process, occurred inconsistencies can be handled by analyzing the respective resources ontologies. The inconsistencies arising in the single development environment are mostly the result of modeling errors. Whereas, the inconsistencies occurred in the merged ontology might stem from different perspectives of the source ontologies for the given domain, each of them correct in their own right. Thus, we need to determine which source ontologies' axioms are the most trustable and can act as a comprehensive solution for handling the inconsistencies. This can be a considerable challenge even for experts already in the face of modestly sized ontologies.

To handle the inconsistency of the merged ontologies, we utilize a Subjective Logic-based approach. Subjective Logic captures opinions about the world in belief models and provides a set of operations for combining opinions. It provides an effective environment to manage and combine beliefs over a set of mutually exclusive assertions. It is applicable when the problem at hand is characterized by considerable uncertainty and incomplete knowledge. In this regards, we apply Subjective Logic to rank and estimate the trustworthiness of conflicting axioms that cause inconsistencies within a merged ontology. We have implemented this approach in a prototypical tool that automatically can rank the conflicting axiom in an inconsistent merged ontology. Moreover, through our tool, we suggest the user a set of revised operations for the conflicting axioms in an inconsistent merged ontology. We will demonstrate the feasibility and effectiveness of our method by the experimental results carried in Chapter 12.

6.2 Preliminaries

Before we introduce our method, we outline preliminaries for unsatisfiable concepts [BCM+03], incoherent [FHP+06] and inconsistent [BCM+03] ontologies along with justification [HPS09] for an inconsistent ontology:

Definition 6.1. ***Unsatisfiable concepts*** $\mathcal{C}_{un}$ *are concepts that cannot have any individuals.*

The term "concept" here refers to the classes based on Definition 4.1 of an ontology in Chapter 4. We keep the term in this chapter to be uniform with the unsatisfiable concept's definition in literature reviews.

An unsatisfiable root concept [KPSCG06] is an unsatisfiable class in which a contradiction found in the class definition does not depend on the unsatisfiability of another class in ontology. We use function $Root_{C_{un}}$ to retrieve all the unsatisfiable root concepts for the given inconsistent merged ontology. If there are a large number of unsatisfiable concepts in an inconsistent ontology, restricting the repair efforts to the root concepts may reduce the necessary effort.

A common error for an ontology is incoherence. It indicates that there are unsatisfiable concepts which is interpreted as an empty set in all the models of its terminology.

Definition 6.2. *An ontology $\mathcal{O}$ is **incoherent** iff there exists an unsatisfiable concept in $\mathcal{O}$.*

The incoherence can be considered as a kind of inconsistency in the TBox, i.e., the terminology part of an ontology. An incoherent ontology has an incoherent TBox. Indeed, incoherence in ontologies corresponds to inconsistency in knowledge bases in classical logic, where a knowledge base is a finite set of classical formulae. A knowledge base is inconsistent if and only if there is no model satisfying all its formulae, as given below:

Definition 6.3. *An ontology $\mathcal{O}$ is **inconsistent** iff there is no model of $\mathcal{O}$, i.e., $\mathcal{O}$ is unsatisfiable.*

We denote an inconsistent merged ontology by $\mathcal{O}'_M$ while $\mathcal{O}_M$ denotes a consistent one.

The incoherence does not provide the classical sense of the inconsistency because there might exist a model for an incoherent ontology [FHP+06]. Thus, we need classical inconsistency for ontologies.

An inconsistent ontology does not necessarily imply that it is incoherent, and vice versa. There exists different combinations of the inconsistency and the incoherence. Let us consider c_1, c_2, and c_3 as sample classes in ontology and a as an instance (individual). There could be four variants on inconsistency and incoherence relations, as has been stated in [FHP+06]:

- An inconsistent and coherent ontology: the two disjoint concepts c_1 and c_2 share an instance a (see Case I in Figure 6.1).
- A consistent and incoherent ontology: the two disjoint concepts c_1 and c_2 share a sub-concept c_3 (see Case II in Figure 6.1).
- An inconsistent and incoherent ontology: the two disjoint concepts c_1 and c_2 share a sub-concept c_3, which has an instance a (see Case III in Figure 6.1).
- An inconsistent but coherent TBox: the two disjoint concepts c_1 and c_1 share a sub-concept, which is a nominal a (see Case IV in Figure 6.1).

The propositional logic reasoning community, such as [BS05], have long used the notion of *minimal conflict sets* for determining the minimal unsatisfiable subsets of a set of clauses. These conflict sets are used to gain insight into why a set of clauses is unsatisfiable. In essence, minimal conflict sets are akin to justifications, given by Definition 6.4.

Definition 6.4. *Let $\mathcal{O}$ be an ontology entailing axiom x, i.e., $\mathcal{O} \models x$. $\mathcal{J}_d$ is a **justification** for x in $\mathcal{O}$ if $\mathcal{J}_d \subseteq \mathcal{O}$, and $\mathcal{J}_d \models x$, and for all $\mathcal{J}'_d \subsetneq \mathcal{J}_d$ $\mathcal{J}'_d \not\models x$.*

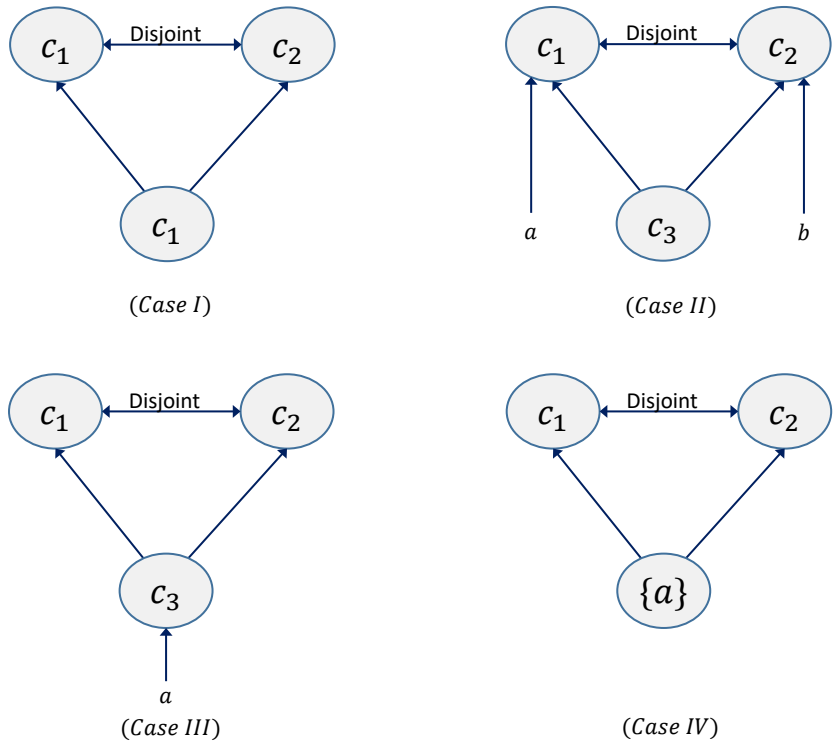


FIGURE 6.1: Examples of various inconsistency and incoherence [FHP+06].

Intuitively, a justification is a minimal subset of an ontology that causes it to be inconsistent. Alternatively, in [SC+03], the authors used the term MUPS (Minimal Unsatisfiable Preserving Sub-Tboxes) for a justification of unsatisfiable concepts in ontologies. A MUPS for a concept is a minimal fragment of the knowledge base in which the concept is unsatisfiable. In this chapter, we use the term justification rather than MUPS[1]. There may be more than one justification for a single unsatisfiable concept, since there may be different inconsistency-causing clashes responsible for its unsatisfiability.

Given the above definitions, in this chapter, we assume the following notations:

- A set of unsatisfiable concepts is denoted by $\mathcal{C}_{un} = \{c_1, c_2, ..., c_u\}$.
- The ontology justification set $\mathcal{J}$ is the set of all justifications and is given by $\mathcal{J} = \{\mathcal{J}_1, \mathcal{J}_2, ..., \mathcal{J}_l\}$ where $\mathcal{J}_d \in \mathcal{J}$. There may be multiple, potentially overlapping justifications in $\mathcal{J}$.
- Each justification $\mathcal{J}_d \in \mathcal{J}$ includes several axioms, denoted by $\mathcal{J}_d = (x_1, x_2, ..., x_z)$.
- Each axiom $x_j \in \mathcal{J}_d$ has a set of entities $\{e_1, e_2, ..., e_t\}$, which we call the involved entities in x_j. For example, c_1 and c_2 are involved entities in the axiom $x_1 : c_1 \sqsubseteq c_2$.

[1]On a practical level, justifications are used by many ontology development and debugging environments such as Protégé [HTR06].

- We represent all axioms belonging to ontology justification set by X and call them the conflicting axioms set. Indeed, they are a group of axioms that conflicts with each other and cause a concept to be unsatisfiable.
- The trustworthiness of X is indicated by $\overline{X}$.

6.2.1 Example: an Inconsistent Ontology

In this section, we aim to show the introduced notations within an example. Let us consider four source ontologies *conference*, *confOf*, *edas*, and *ekaw* from the conference track of OAEI benchmark[2]. The mappings between these source ontologies are generated by the SeeCOnt tool [ABKD15]. We merged the given source ontologies by our proposed approach in Chapter 4 without applying refinements.

In Figure 6.2, we show the class hierarchy of the built merged ontology, which is taken from Protégé[3] version 5.5. The reasoner shows that the merged ontology is inconsistent and contains unsatisfiable concepts. In the figure, the unsatisfiable concepts are shown in red. As a total, the merged ontology has 68 unsatisfiable concepts; 3 of which are unsatisfiable root concepts, as shown by c_1, c_2, and c_3 in Figure 6.2. For these 3 unsatisfiable root concepts, there are 10 different justification sets, containing 69 duplicated conflicting axioms. In the figure, we show as an instance, 5 justification sets ($\mathcal{J}_1$-$\mathcal{J}_5$), containing 19 distinct (32 duplicated) conflicting axioms (x_1-x_{19}). Note that some axioms appear in several justification sets (cf. x_6 in $\mathcal{J}_2$ and $\mathcal{J}_4$).

Based on our notation, we have:

- $C_{un} = \{c_1, c_2, ..., c_{68}\}$
- $Root_{C_{un}} = \{\texttt{Person}, \texttt{Track}, \texttt{Working\_event}\}$
- $\mathcal{J} = \{\mathcal{J}_1, \mathcal{J}_2, ..., \mathcal{J}_{10}\}$
- $X = \{x_1, x_2, ..., x_{69}\}$

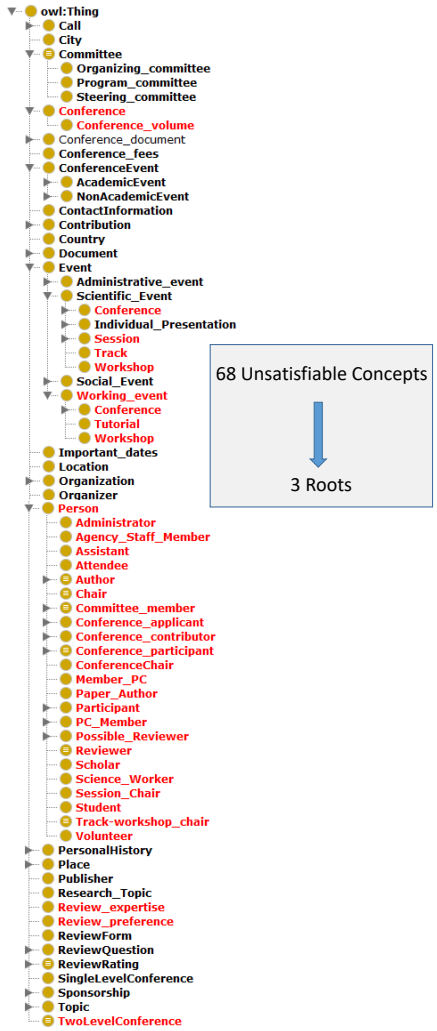
To transform the inconsistent merged ontology into a consistent one, part of the conflicting axioms should be revised. Deciding on this issue is somehow complex. Processing justification sets and repairing ontologies is frequently referred to as ontology debugging, where we aim to find the hitting sets of the justifications (sometimes called diagnoses). We contribute to utilizing the Subjective Logic theory to find the least trustworthy axioms to be revised. In Section 6.7, we will present how our method can be applied to this example.

In the next subsection, we present the background knowledge of Subjective Logic theory.

6.2.2 Subjective Logic Theory

In traditional binary logic, propositions are considered only in two states. Thus, believing a proposition about an aspect of the world can be presented by discrete values of TRUE and FALSE. However, because our knowledge about the world never is perfect,

[2] http://oaei.ontologymatching.org/2019/conference/index.html
[3] https://protege.stanford.edu/



Person c_1

x_1. Person SubClassOf hasName exactly 1 rdfs:Literal
x_2. hasName Domain Conference
x_3. Conference DisjointWith Person — $\mathcal{J}_1$

x_1. Person SubClassOf hasName exactly 1 rdfs:Literal
x_2. hasName Domain Conference
x_4. Conference SubClassOf Scientific_Event
x_5. Scientific_Event SubClassOf Event
x_6. Event DisjointWith Person — $\mathcal{J}_2$

Working_event c_2

x_7. Working_event SubClassOf hasTopic some Topic
x_8. isTopicOf InverseOf hasTopic
x_9. isTopicOf Range Conference or Paper
x_{10}. Confernce SubClassOf hasName exactly 1 rdfs:Literal
x_{11}. hasName Domain Person
x_{12}. Paper SubClassOf Contribution
x_{13}. Working_event SubClassOf Event
x_3. Conference DisjointWith Person
x_{14}. Contribution DisjointWith Event — $\mathcal{J}_3$

x_7. Working_event SubClassOf hasTopic some Topic
x_8. isTopicOf InverseOf hasTopic
x_9. isTopicOf Range Conference or Paper
x_{10}. Confernce SubClassOf hasName exactly 1 rdfs:Literal
x_{11}. hasName Domain Person
x_{12}. Paper SubClassOf Contribution
x_{13}. Working_event SubClassOf Event
x_{14}. Conference DisjointWith Event
x_6. Event DisjointWith Person — $\mathcal{J}_4$

Track c_3

x_{15}. Track SubClassOf Scientific_Event
x_{16}. Scientific_Event SubClassOf Event
x_{17}. Track SubClassOf Conference_part
x_{18}. Conference_part SubClassOf Paper
x_{19}. Paper SubClassOf Contribution
x_{14}. Contribution DisjointWith Event — $\mathcal{J}_5$

FIGURE 6.2: Example of an inconsistent ontology: The concepts shown in red are unsatisfiable. The ontology contains 68 unsatisfiable concepts, which among 3 are roots, shown by c_1-c_3. Five justification sets $\mathcal{J}_1$-$\mathcal{J}_5$, including a set of conflicting axioms x_1-x_{19}, are presented.

human expressions may not always determine with absolute certainty whether the given proposition is TRUE or FALSE. Additionally, when the truth of a statement is evaluated by one individual (source), it can not represent a general and objective opinion. This indicates that capturing our perception of reality by traditional logic is applicable to the idealistic world, only.

Thereby, with our imperfect knowledge, we can only have an opinion about the proposition. In this view, we can express our opinion with degrees of belief or disbelief. Moreover, our unawareness of the proposition can be presented as a vacuous belief, filling the vacancy in the absence of both belief and disbelief. In this field, Subjective Logic theory [Jøs16] emerges to express subjective opinions and operates on our subjective perception about the world. The logic uses individual opinions about the truth of propositions as variables.

Since opinions are subjective, they have ownership assigned whenever relevant. The owner of an opinion is called an agent. In the formulation of the opinion in Subjective Logic, superscripts indicate ownership, and subscripts indicate the proposition to which the opinion applies. For instance, $w_{\mathcal{P}}^{A}$ is an opinion held by agent A about the truth of proposition $\mathcal{P}$. The agents can express their opinions with degrees of belief (b), disbelief (d), uncertainty (u), and atomicity (a) about a given proposition ($\mathcal{P}$). In particular:

- *Belief* (b) is the belief that the proposition is TRUE.
- *Disbelief* (d) is the belief that the proposition is FALSE.
- *Uncertainty* (u) is the amount of uncommitted belief.
- *Atomicity* (a) or a priori probability is the base rate in the absence of all evidence.

These components satisfy $b + d + u = 1.0$ and $b, d, u, a \in [0, 1]$. An opinion denoted by $w_{\mathcal{P}}^{A} = (b, d, u, a)$ expresses the belief of the agent A in the truth of the proposition $\mathcal{P}$.

Binomial opinions are represented in [Jøs16] on a triangle, as shown in Figure 6.3. A point inside the triangle indicates a (b, d, u) triple. The triangle has three axes: belief, disbelief, and uncertainty. The axes run from one edge to the opposite vertex. The atomicity is shown as a point on the base line. Given these values, we can demonstrate the probability expectation (t_x) for an opinion. It is formed by projecting the opinion point onto the base, parallel to the base rate director line. As an instance, the opinion $w_x = (0.2, 0.5, 0.3, 0.6)$ with probability expectation $t_x = 0.38$ is shown in the Figure 6.3. A strong positive opinion can be represented by a point towards the bottom right belief vertex. As a whole, if the opinion point is located at the left or right bottom vertex in the triangle, the opinion is equivalent to boolean TRUE or FALSE. The characteristics of various binomial opinion classes are listed below. A binomial opinion:

- where $b = 1$ is equivalent to binary logic TRUE,
- where $d = 1$ is equivalent to binary logic FALSE,
- where $b + d = 1$ is equivalent to a traditional probability,
- where $b + d < 1$ expresses degrees of uncertainty, and
- where $b + d = 0$ expresses total uncertainty.

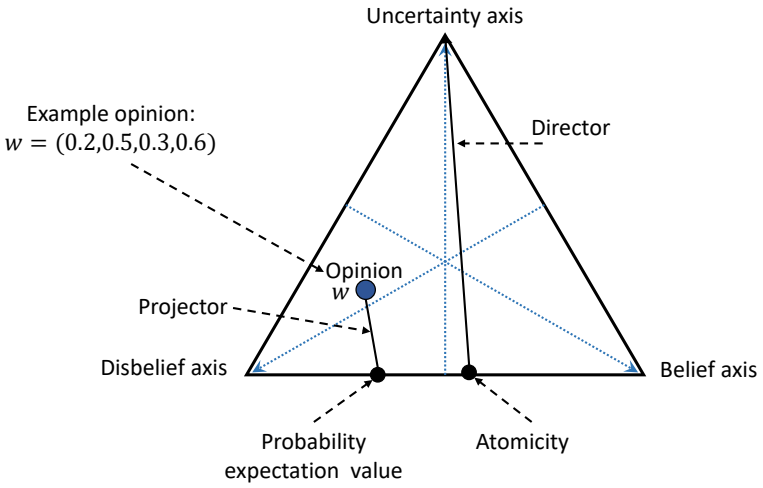


FIGURE 6.3: Opinion triangle with example opinion [Jøs16].

In the context of merging ontologies, it is the creators of the source ontologies that have opinions about the trustworthiness of certain axioms. Since we do not have direct access to the creators, we substitute their opinions by the ontologies that they created in order to reflect their subjective views. Thus, for us, the source ontologies reflecting the creators' belief play the agent role.

Let $\mathcal{P}$ be a proposition such as:

"Axiom x is trustworthy in the merged ontology $\mathcal{O}_M$".

Then, the binomial *opinion* w of agent $\mathcal{O}_i \in \mathcal{O}_s$ about the proposition $\mathcal{P}$ is equivalent to a beta distribution for the information source x [Jøs16]. An opinion of axiom x from agent $\mathcal{O}_i$ is given by $w_x^{\mathcal{O}_i}$. It is expressed by belief $b_x^{\mathcal{O}_i}$, disbelief $d_x^{\mathcal{O}_i}$, uncertainty $u_x^{\mathcal{O}_i}$, and atomicity $a_x^{\mathcal{O}_i}$ with a tuple given by Equation 6.1, where $b_x^{\mathcal{O}_i} + d_x^{\mathcal{O}_i} + u_x^{\mathcal{O}_i} = 1.0$ and $b_x^{\mathcal{O}_i}, d_x^{\mathcal{O}_i}, u_x^{\mathcal{O}_i}, a_x^{\mathcal{O}_i} \in [0, 1]$.

$$w_x^{\mathcal{O}_i} = (b_x^{\mathcal{O}_i}, d_x^{\mathcal{O}_i}, u_x^{\mathcal{O}_i}, a_x^{\mathcal{O}_i}) \tag{6.1}$$

Opinions are formed on the basis of positive and negative evidence. Let $r_x^{\mathcal{O}_i}$ and $s_x^{\mathcal{O}_i}$ be the number of positive and negative past observations about x by agent $\mathcal{O}_i$, respectively. Then, $b_x^{\mathcal{O}_i}, d_x^{\mathcal{O}_i}$, and $u_x^{\mathcal{O}_i}$ are calculated upon them, as it is shown in Equations 6.2, 6.3, and 6.4.

$$b_x^{\mathcal{O}_i} = \frac{r_x^{\mathcal{O}_i}}{r_x^{\mathcal{O}_i} + s_x^{\mathcal{O}_i} + 2} \tag{6.2}$$

$$d_x^{\mathcal{O}_i} = \frac{s_x^{\mathcal{O}_i}}{r_x^{\mathcal{O}_i} + s_x^{\mathcal{O}_i} + 2} \tag{6.3}$$

$$u_x^{\mathcal{O}_i} = \frac{2}{r_x^{\mathcal{O}_i} + s_x^{\mathcal{O}_i} + 2} \tag{6.4}$$

where, 2 in Equations 6.2, 6.3, and 6.4 indicated binomial opinions. Thus, the opinion's probability expectation value is computed in Equation 6.5 as the trustworthiness of axiom x by agent $\mathcal{O}_i$:

$$t_x^{\mathcal{O}_i} = b_x^{\mathcal{O}_i} + a_x^{\mathcal{O}_i} \times u_x^{\mathcal{O}_i} \tag{6.5}$$

In order to apply this logic to our problem, we need to formulate $r_x^{\mathcal{O}_i}$ and $s_x^{\mathcal{O}_i}$ as a basis to computing $b_x^{\mathcal{O}_i}, d_x^{\mathcal{O}_i}, u_x^{\mathcal{O}_i}$, and $a_x^{\mathcal{O}_i}$. We contribute to them in the next section. Upon them, we can calculate the trustworthiness of the conflicting axioms. We will also present how the combination of opinion can be held by Subjective Logic operators, considering dependencies across them.

In general, over the last decade, Subjective Logic has been increasingly used in the Semantic Web. It is successfully applied in a variety of applications, such as ontology alignments [HS08; Jus04], annotation evaluation [CNF12; CVHF10], recommendation systems [PK12], and ontology inconsistency handling [SPF+12]. The last one, similar to us, used the Subjective Logic to solve ontology inconsistencies but in a single development environment. Moreover, the authors only used the atomicity value and omitted the belief, disbelief, and uncertainty values. Additionally, they did not consider the agent's opinion combination.

6.3 Proposed Method for Inconsistency Handling by Subjective Logic

We handle inconsistency of the merged ontology by utilizing the Subjective Logic theory. To achieve this, we formulate negative s and positive r observation along with atomicity a in Section 6.3.1, Section 6.3.2, and Section 6.3.3, respectively. We then demonstrate how different opinions can be combined in Section 6.3.4, followed by considering their dependency in Section 6.3.5.

6.3.1 Negative Observation

To determine the negative observation r for the trustworthiness of the axiom x_j belonging to the conflicting axiom sets X, we use:

- $\Psi_{x_j}(\mathcal{J})$: the axiom frequency for x_j in the justification set $\mathcal{J}$
- $\Psi_{\mathcal{O}_i}(X)$: the number of conflicting axiom sets, belonging to $\mathcal{O}_i$

With the first criterion, we want to observe how many times the axiom x_j appears in $\mathcal{J}$'s elements. Thus, we count the number of axiom x_j in $\mathcal{J}$. With the second one, we aim

to reflect the view of the $\mathcal{O}_i$. Upon them, we determine the negative observation s for axiom x_j by agent $\mathcal{O}_i$ in Equation 6.6.

$$s_{x_j}^{\mathcal{O}_i} = \frac{\Psi_{x_j}(\mathcal{J})}{\Psi_{\mathcal{O}_i}(X)} \tag{6.6}$$

This idea is similar to the notion of *arity* of the axiom in [SC+03]. Moreover, it is already used in [KPSCG06] to accelerate the process of getting rid of unsatisfiable concepts in an inconsistent ontology.

6.3.2 Positive Observation

To determine the positive observations *r* of the axiom $x_j \in X$, we use the provenance information of the axiom. Note that each axiom in the merged ontology $\mathcal{O}_M$ is derived from one or several source ontologies. Therefore, we can base the positive observation r for x_j by the agent $\mathcal{O}_i$ as the multiplication of:

- *Provenance information*: the existence of the axiom x_j in the source ontology $\mathcal{O}_i$
- *Effect*: the impact of changes of the axiom x_j in the merged ontology $\mathcal{O}_M$

In our prototype, the existence of an axiom in a source ontology is determined by searching their equivalent entities based on the given mappings $\mathcal{M}$. This could be extended to more powerful logic-based approaches.

The second criterion (*effect*) reflects how much the ontology gets affected if the axiom x_j is removed or altered. For this, we determine how often the elements e_{x_j} of axiom x_j have been referenced in other axioms in the ontology. We formulate:

- $\Gamma_{x_j}(\mathcal{O}_i)$: the number of axioms in $\mathcal{O}_i$ that contains elements of x_j
- $\Gamma(\mathcal{O}_i)$: the total number of axioms in $\mathcal{O}_i$

Then, the fraction of axioms in $\mathcal{O}_i$ that contains elements of x_j indicates their effect, as shown in Equation 6.7.

$$f_{x_j}(\mathcal{O}_i) = \frac{\Gamma_{x_j}(\mathcal{O}_i)}{\Gamma(\mathcal{O}_i)} \tag{6.7}$$

The significance of this strategy is based on the following intuition: if the elements in the axiom are used and referred often in other axioms, changing or removing axioms related to these elements may be undesired. For example, if a certain class is heavily instantiated, or if a certain property is heavily used by many classes, then the user needs to be aware of changing this axiom.

Therefore, the positive observation *r* for axiom x_j by agent $\mathcal{O}_i$ is determined by *provenance* $\times$ *effect*, as shown in Equation 6.8.

$$r_{x_j}^{\mathcal{O}_i} = \begin{cases} \alpha \times f_{x_j}(\mathcal{O}_i) & \text{if } x_j \in \mathcal{O}_i \\ \beta \times f_{x_j}(\mathcal{O}_i) & \text{if } x_j \notin \mathcal{O}_i \end{cases} \tag{6.8}$$

The provenance of the axioms is represented by the α and β parameters, which can be determined by the user. If the given axiom does not belong to the source ontology, i.e., $x_j \notin \mathcal{O}_i$, but at least one element of the axiom exists in $\mathcal{O}_i$, then $f_{x_j}(\mathcal{O}_i)$ is multiplied with β parameter. Otherwise, it is multiplied with α parameter.

6.3.3 Atomicity

The atomicity metric (or namely, base rate or a priori probability) plays an important role in the absence of evidence for belief, disbelief, and uncertainty. It reflects prior knowledge about the phenomenon at hand. When there is no other evidence for belief, disbelief, and uncertainty, we use the available indicator of ontology's characteristics in order to determine the atomicity in our context.

To this end, we utilize the centrality measure of the axioms' elements. Centrality measures are designed to rank the entities according to their positions in the graph. They are interpreted as the prominence of entities embedded in an intended structure. In [ABKD15], centrality measures have been successfully utilized to rank the importance of ontology's entities. Following this idea, we use them to express atomicity. This leads us to use degree centrality [Nie74], which calculates the number of connections of a node. In a directed graph, there are an in-degree and an out-degree centrality that calculate the number of input and output links, respectively. In the context of ontologies, we indicate the in-degree and out-degree with the number of super- and subclasses for a given class.

Therefore, for axiom x_j containing t elements $x_j = \{e_1, e_2, ..., e_t\}$, the atomicity a by agent $\mathcal{O}_i$ is shown in Equation 6.9. $SubClass(e_g)$ and $SuperClass(e_g)$ return the sub- and superclasses of $e_g \in x_j$, respectively.

$$a_{x_j}^{\mathcal{O}_i} = \frac{1}{|e| \in \mathcal{O}_i} \times \sum_{g=1}^{t} |SubClass(e_g) \cup SuperClass(e_g)|,\ e_g \in x_j \tag{6.9}$$

Through Equation 6.9, the atomicity is calculated by the total number of super- and subclasses of the elements divided by the total number of elements $|e|$ in $\mathcal{O}_i$. It is evident that the atomicity for an entity increases as it has a larger number of connections in the class hierarchy.

Summing up, by formulating the negative s and positive r observation alongside with atomicity a, we can obtain the belief, disbelief, and uncertainty to specify the opinion of agent $\mathcal{O}_i$ for the axiom x_j. As a result, the trustworthiness of the axiom x_j can be achieved by Equation 6.5.

6.3.4 Combining Opinions

Up to now, we have been looking at the opinions of individual agents. The *consensus* ($\oplus$) operator [Jøs16] of Subjective Logic computes the combination of opinions of different agents. Combining the opinions lets to express more trustworthy opinions that are agreed upon by multiple agents. The consensus operator combines opinions of two or more independent and possibly conflicting opinions about the same proposition into a single one that reflects both opinions in a fair and equal way.

Let $w_x^{\mathcal{O}_1} = (b_x^{\mathcal{O}_1}, d_x^{\mathcal{O}_1}, u_x^{\mathcal{O}_1}, a_x^{\mathcal{O}_1})$ and $w_x^{\mathcal{O}_2} = (b_x^{\mathcal{O}_2}, d_x^{\mathcal{O}_2}, u_x^{\mathcal{O}_2}, a_x^{\mathcal{O}_2})$ be opinions respectively held by $\mathcal{O}_1$ and $\mathcal{O}_2$ about the same proposition x. Then, the consensus ($\oplus$) for these two opinions $w_x^{\mathcal{O}_1\mathcal{O}_2} = w_x^{\mathcal{O}_1} \oplus w_x^{\mathcal{O}_2}$ is shown in Equation 6.10.

$$w_x^{\mathcal{O}_1} \oplus w_x^{\mathcal{O}_2} = \left(\frac{b_x^{\mathcal{O}_1} u_x^{\mathcal{O}_2} + b_x^{\mathcal{O}_2} u_x^{\mathcal{O}_1}}{k}, \frac{d_x^{\mathcal{O}_1} u_x^{\mathcal{O}_2} + d_x^{\mathcal{O}_2} u_x^{\mathcal{O}_1}}{k}, \frac{u_x^{\mathcal{O}_1} u_x^{\mathcal{O}_2}}{k}, \frac{a_x^{\mathcal{O}_1} u_x^{\mathcal{O}_2} + a_x^{\mathcal{O}_2} u_x^{\mathcal{O}_1} - (a_x^{\mathcal{O}_1} + a_x^{\mathcal{O}_2}) u_x^{\mathcal{O}_1} u_x^{\mathcal{O}_2}}{u_x^{\mathcal{O}_1} + u_x^{\mathcal{O}_2} - 2u_x^{\mathcal{O}_1} u_x^{\mathcal{O}_2}} \right) \tag{6.10}$$

where $k = u_x^{\mathcal{O}_1} + u_x^{\mathcal{O}_2} - u_x^{\mathcal{O}_1} u_x^{\mathcal{O}_2}$ such that $k \neq 0$ and $a_x^{\mathcal{O}_1\mathcal{O}_2} = (a_x^{\mathcal{O}_1} + a_x^{\mathcal{O}_2})/2$ when $u_x^{\mathcal{O}_1}, u_x^{\mathcal{O}_2} = 1$.

Thus, by utilizing the consensus operator over the opinions from different agents about the conflicting axioms, we can achieve a more robust opinion.

6.3.5 Applying Conditional Opinions to Reflect the Dependencies

As we stated before, justifications in the ontology justification set $\mathcal{J} = \{\mathcal{J}_1, \mathcal{J}_2, ..., \mathcal{J}_l\}$ might have overlap, i.e., the axioms in the $\mathcal{J}_d \in \mathcal{J}$ might exist in other $\mathcal{J}$s (see the example in Section 6.2.1). In our prototype, we use the sorted order of justifications set based on the axiom frequency. The described approach so far does not consider the effect of the ranked value for the axioms that are already calculated in one $\mathcal{J}_d$, once the other $\mathcal{J}$s want to be ranked. To overcome this drawback, we use the conditional theory of Subjective Logic [Jøs16] in order to reflect the effect of dependent opinions.

Let us explain our intuition with an example: Suppose the ontology justification set includes three justifications, as $\mathcal{J} = \{\mathcal{J}_1, \mathcal{J}_2, \mathcal{J}_3\}$. Let us consider that axioms are repeated in multiple $\mathcal{J}$s, such as:

$$\mathcal{J} = \{(x_1^{\mathcal{J}_1}, x_2^{\mathcal{J}_1}, x_3^{\mathcal{J}_1}), (x_4^{\mathcal{J}_2}, x_5^{\mathcal{J}_2}, x_6^{\mathcal{J}_2}, x_7^{\mathcal{J}_2}), (x_3^{\mathcal{J}_3}, x_4^{\mathcal{J}_3}, x_5^{\mathcal{J}_3}, x_7^{\mathcal{J}_3}, x_8^{\mathcal{J}_3})\}$$

where $\mathcal{J}_1 \cap \mathcal{J}_2 = \emptyset$ but $(\mathcal{J}_1, \mathcal{J}_2) \cap \mathcal{J}_3 \neq \emptyset$. The opinions for axioms belonging to $\mathcal{J}_1$ and $\mathcal{J}_2$ are calculated as independent opinions using Equation 6.1. However, some axioms of $\mathcal{J}_3$ have already obtained ranked values from $\mathcal{J}_1$ and $\mathcal{J}_2$. Although we can use the previous ranked values from $\mathcal{J}_1$ and $\mathcal{J}_2$ for x_3, x_4, x_5, x_7 in $\mathcal{J}_3$, it might happen that these axioms get differently ranked concerning the remaining axioms in $\mathcal{J}_3$. Therefore, in an incremental process, we calculate a new value in each $\mathcal{J}$, but we also consider the effect of the previous ranked values for axioms in other $\mathcal{J}$s.

In this case, the proposition $\mathcal{P}$, for instance, for axiom x_3 writes as:

"$x_3 \in \mathcal{J}_3$ is trustworthy if $x_3 \in \mathcal{J}_1$ already has a trustworthiness of 0.3".

This idea translates to "IF x THEN y", which is equal to the probability of the proposition y given that the proposition x is TRUE. More precise expression is:

$$p(\text{IF } x \text{ THEN } y) = p(y|x)$$

This has been represented in [Jøs16] by $w_{y||x}$ with the conditional *deduction* ($\circledcirc$) operator.

Let $\mathbb{X} = \{x, \bar{x}\}$ and $\mathbb{Y} = \{y, \bar{y}\}$ be two binary frames where there is a degree of relevance between $\mathbb{X}$ and $\mathbb{Y}$. Let us consider that

- $w_x = (b_x, d_x, u_x, a_x)$ is an agent's opinion about x being true;
- $w_{y|x} = (b_{y|x}, d_{y|x}, u_{y|x}, a_{y|x})$ is an agent's opinion about y being true given that x is true;
- $w_{y|\bar{x}} = (b_{y|\bar{x}}, d_{y|\bar{x}}, u_{y|\bar{x}}, a_{y|\bar{x}})$ is an agent's opinion about y being true given that x is false.

The conditional *deduction* ($\circledcirc$) operator is a ternary operator (i.e., requires 3 input arguments), as shown in Equation 6.11.

$$w_{y||x} = w_x \circledcirc (w_{y|x}, w_{y|\bar{x}}) = (b_{y||x}, d_{y||x}, u_{y||x}, a_{y||x}) \tag{6.11}$$

where,

$$\begin{aligned}
b_{y||x} &= (b_x b_{y|x} + d_x b_{y|\bar{x}} + u_x(b_{y|x}a_x + b_{y|\bar{x}}(1 - a_x))) - a_y K \\
d_{y||x} &= (b_x d_{y|x} + d_x d_{y|\bar{x}} + u_x(d_{y|x}a_x + d_{y|\bar{x}}(1 - a_x))) - (1 - a_y)K \\
u_{y||x} &= (b_x u_{y|x} + d_x u_{y|\bar{x}} + u_x(u_{y|x}a_x + u_{y|\bar{x}}(1 - a_x))) + K \\
a_{y||x} &= a_y
\end{aligned} \tag{6.12}$$

Indicator K can be determined according to three different selection criteria detailed in [Jøs16].

If y relates to multiple x variables, they are combined using the consensus operator, as shown by Equation 6.13.

$$w_{y||(x_1,x_2,..x_j)} = w_{y||x_1} \oplus w_{y||x_2} \oplus ... \oplus w_{y||x_j} \tag{6.13}$$

In our example, several axioms in $\mathcal{J}_3$ are already repeated in $\mathcal{J}_1$ and $\mathcal{J}_2$, i.e., $w_{y||(x_3,x_4,x_5,x_7)}$.

The deduction operator requires that $w_{y|x}$ and $w_{y|\bar{x}}$ be formulated in our context. Suppose $w_{x_j}^{\mathcal{J}_1} = (b_{x_j}^{\mathcal{J}_1}, d_{x_j}^{\mathcal{J}_1}, u_{x_j}^{\mathcal{J}_1}, a_{x_j}^{\mathcal{J}_1})$ and $w_{x_f}^{\mathcal{J}_2} = (b_{x_f}^{\mathcal{J}_2}, d_{x_f}^{\mathcal{J}_2}, u_{x_f}^{\mathcal{J}_2}, a_{x_f}^{\mathcal{J}_2})$ are already calculated

by Equation 6.1. Let for the sake of simplicity $y = x_f^{\mathcal{J}_2}$ and $x = x_j^{\mathcal{J}_1}$. Then, following the probabilistic conditional $p(y|x) = \frac{p(y \wedge x)}{p(x)}$, we re-write $p(y|x)$ as:

$$w_{y|x} = \frac{w_{y \wedge x}}{w_x} \tag{6.14}$$

where $w_{y \wedge x} = (b_{y \wedge x}, d_{y \wedge x}, u_{y \wedge x}, a_{y \wedge x})$ [Jøs16] is shown by Equation 6.15.

$$\begin{aligned} b_{y \wedge x} &= b_y b_x, & d_{y \wedge x} &= d_y + d_x - d_y d_x, \\ u_{y \wedge x} &= b_y u_x + u_y b_x + u_y u_x, & a_{y \wedge x} &= \frac{b_y u_x a_x + u_y a_y b_x + u_y a_y u_x a_x}{b_y u_x + u_y b_x + u_y u_x} \end{aligned} \tag{6.15}$$

Division operation $\frac{w_y}{w_x}$ is defined in [Jøs16], as represented in Equation 6.16.

$$\begin{aligned} b_{y \backslash x} &= b_y - b_x, & d_{y \backslash x} &= \frac{a_y(d_y + b_x) - a_x(1 + b_x - b_y - u_x)}{a_y - a_x}, \\ u_{y \backslash x} &= \frac{a_y u_y - a_x u_x}{a_y - a_x}, & a_{y \backslash x} &= a_y - a_x \end{aligned} \tag{6.16}$$

We apply the same approach to $w_{y|\bar{x}}$, where $w_{\bar{x}}$ is defined in [Jøs16] as $w_{\bar{x}} = (d_x, b_x, u_x, 1 - a_x)$.

In the next section, we present our workflow to handle the inconsistency of the merged ontology.

6.4 Inconsistency Handling Workflow

Figure 6.4 shows our workflow to handle the inconsistency of the merged ontology. It includes an iterative process in five main steps:

1. The merged ontology is tested by off-the-shelf reasoners such as Pellet [SPG+07], whether it is consistent and coherent.
2. For the inconsistent and incoherent merged ontology, containing unsatisfiable concepts, a set of conflicting axioms are identified using the reasoner.
3. Trustworthiness of each axiom is computed based on our Subjective Logic-based approach. The degree of trust for each conflicting axiom will be presented to the user.
4. For the lowest trustworthiness axiom, a revised plan is generated, and possible suggestions to repair the ontology are provided to the user.
5. The plan can be applied automatically, or users can make their desired changes before applying them.

After revising the conflicting axioms by performing a set of processes such as deleting or rewriting, the merged ontology is tested again with the consistency test of the reasoner.

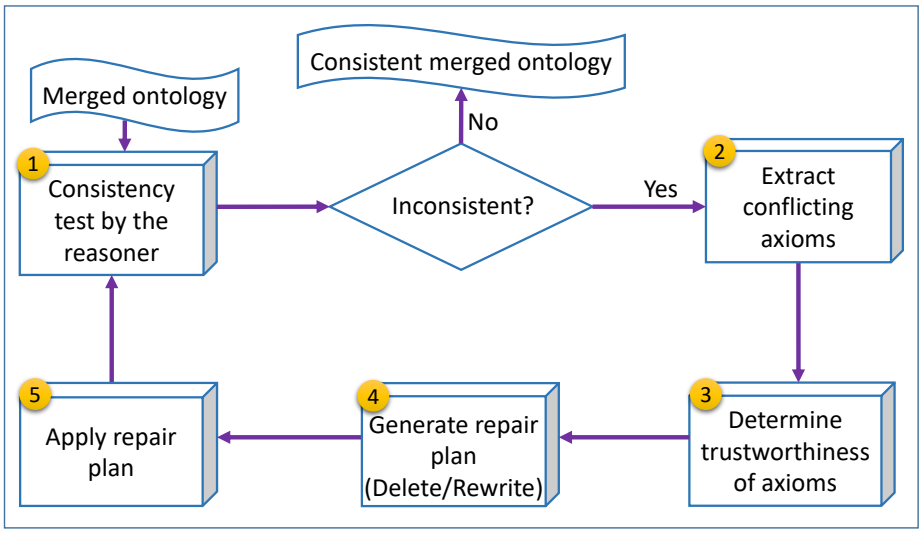


FIGURE 6.4: Our workflow to handle inconsistency of the merged ontology.

If applicable, the process will be iterated. Sometimes, more than one iteration is required to make the inconsistent merged ontology into the consistent ones.

6.5 Algorithm

Algorithm 6.1 describes our proposed approach to rank the conflicting axioms in order to determine the degree of trustworthiness of them. The algorithm takes as input an inconsistent merged ontology $\mathcal{O}'_M$ with a set of source ontologies $\mathcal{O}_S$. It generates a consistent merged ontology $\mathcal{O}_M$.

At first, unsatisfiable concepts C_{un} and their justification sets $\mathcal{J}$ are extracted (*lines 2-3*). After that, for each justification $\mathcal{J}_d \in \mathcal{J}$, the algorithm checks whether it has some shared axioms with other $\mathcal{J}$s, i.e., $(\mathcal{J}_d \cap \{\mathcal{J}_1, \mathcal{J}_2, ..., \mathcal{J}_{d-1}\} \neq \emptyset)$ (*line 7*). If so, the opinion for $x_j \in \mathcal{J}_d$ is calculated as the conditional deduction in *line 8*. Otherwise, it is calculated as *line 10*. Then, the opinion combination of all agents takes place *(line 13)*. The ranked value (trustworthiness) for each axiom $x_j \in \mathcal{J}_d$ is represented by $\bar{x}_j^{\mathcal{J}_d}$ and calculated in *lines 14-15* . Based on the total ranked axioms, shown by $\overline{X}$, a repair plan is generated and presented to the users (*line 18*). Users can apply our suggested plan directly or edit it before applying on the $\mathcal{O}'_M$ (*line 19*). The merged ontology is checked again for the consistency tests. If it is still inconsistent, the whole process will be repeated. Otherwise, $\mathcal{O}_M$ is returned as a consistent merged ontology.

6.6 Repair plan

In the repair process, a plan should be generated whose primary goal is to suggest a repair solution for the least trustworthy axioms while preserving the more trustworthy axioms. To this end, a library of error patterns extracted from [KPSCG06] has been embedded in our system. For the least trustworthy axiom of each justification, we check

Algorithm 6.1: The Subjective Logic-based algorithm for handling inconsistencies of the merged ontology.

Input: an inconsistent merged ontology $\mathcal{O}'_M$, a set of source ontologies $\mathcal{O}_S$;
Output: a consistent merged ontology $\mathcal{O}_M$;

while $(\mathcal{O}'_M \; is \; inconsistent)$ **do**
  $C_{un} \Leftarrow$ extract unsatisfiable concepts from $\mathcal{O}'_M$;
  $\mathcal{J} \Leftarrow$ extract justifications for C_{un};
  for $(\forall \, \mathcal{J}_d \in \mathcal{J})$ **do**
    for $(\forall \, x_j \in \mathcal{J}_d)$ **do**
      for $(\forall \, \mathcal{O}_i \in \mathcal{O}_S)$ **do**
        if $(\mathcal{J}_d \cap \{\mathcal{J}_1, \mathcal{J}_2, ..., \mathcal{J}_{d-1}\} \neq \emptyset)$ **then**
          $w_{x_j}^{\mathcal{O}_i} \Leftarrow w_{y||(x_j)}^{\mathcal{O}_i}$ according to Equation 6.11;
        else
          $w_{x_j}^{\mathcal{O}_i} \Leftarrow (b_{x_j}^{\mathcal{O}_i}, d_{x_j}^{\mathcal{O}_i}, u_{x_j}^{\mathcal{O}_i}, a_{x_j}^{\mathcal{O}_i})$ according to Equations 6.2, 6.3, 6.4, and 6.9;
        end
      end
      $w_{x_j}^{\mathcal{O}} = w_{x_j}^{\mathcal{O}_1} \oplus ... \oplus w_{x_j}^{\mathcal{O}_n}$ according to Equation 6.10;
      $t_{x_j}^{\mathcal{O}} = b_{x_j}^{\mathcal{O}} + a_{x_j}^{\mathcal{O}} \times u_{x_j}^{\mathcal{O}}$ according to Equation 6.5;
      $\bar{x}_j^{\mathcal{J}_d} \Leftarrow t_{x_j}^{\mathcal{O}}$;
    end
  end
  $plan \Leftarrow GenerateRepairPlan(\overline{X})$;
  $\mathcal{O}'_M \Leftarrow ApplyPlan(\mathcal{O}'_M, plan)$;
end
$\mathcal{O}_M \Leftarrow \mathcal{O}'_M$;
return $\mathcal{O}_M$

if any of the axioms has a pattern corresponding to one in the library. If so, we suggest it to the user as a replacement axiom. As a whole, a set of candidate changes is presented to the user, in which the user can apply the repair plan on a particular or all proposed axiom(s). Note that the generated plan can be applied automatically, or the user can customize it before applying.

It might happen that the user-selected plan cannot get rid of all inconsistencies in one iteration in the ontology. Therefore, the mentioned procedures will be repeated until a consistent merged ontology will be achieved.

6.7 Example: Applying Our Method on an Inconsistent Ontology

Let us consider the four source ontologies from Section 6.2.1 that are merged by the merger component in Figure 6.5. As described before, the reasoner component

in Figure 6.5 shows that the merged ontology $\mathcal{O}_M$ is inconsistent and has three unsatisfiable root concepts $C_{un} = \{c_1, c_2, c_3\}$, where c_1=`Person`, c_2=`Working_event`, and c_3=`Track`. There are 10 justification sets for the three unsatisfiable concepts. We show three $\mathcal{J}$s in the figure, including:

$$\mathcal{J} = \{(x_1^{\mathcal{J}_1}, x_2^{\mathcal{J}_1}, x_3^{\mathcal{J}_1}), (x_1^{\mathcal{J}_2}, x_2^{\mathcal{J}_2}, x_4^{\mathcal{J}_2}, x_5^{\mathcal{J}_2}, x_6^{\mathcal{J}_2}), (x_7^{\mathcal{J}_3}, x_8^{\mathcal{J}_3}, x_9^{\mathcal{J}_3}, x_{10}^{\mathcal{J}_3}, x_{11}^{\mathcal{J}_3}, x_{12}^{\mathcal{J}_3}, x_{13}^{\mathcal{J}_3}, x_3^{\mathcal{J}_3}, x_{14}^{\mathcal{J}_3})\}$$

A remedy for inconsistencies arises in the merged ontology would require the removal or rewriting a minimal part of the ontology in order to make the merged ontology consistent. To compute which axioms are less trustworthiness, we follow the presented Subjective Logic-based approach, as shown in the ranker component in Figure 6.5.

Axioms in the first $\mathcal{J}$ are considered independent since there are no previous justifications for them. Thus, they follow the independent opinion process, as it is shown in Figure 6.5. In this regard, proposition $\mathcal{P}$, for instance, for $x_1 \in \mathcal{J}_1$ is written as:

"the information source $x_1^{\mathcal{J}_1}$ is trustworthy in the merged ontology $\mathcal{O}_M$".

We show the opinion of each ontology in Table 6.2. Based on the given values, the probability exception values are calculated for each opinion in the last column.

TABLE 6.2: Example opinions for x_1 from four source ontologies $\mathcal{O}_1$-$\mathcal{O}_4$.

Ontology	Opinion	Value	Probability exception
$\mathcal{O}_1$	$w_{x_1^{\mathcal{J}_1}}^{\mathcal{O}_1} = (b_{x_1^{\mathcal{J}_1}}^{\mathcal{O}_1}, d_{x_1^{\mathcal{J}_1}}^{\mathcal{O}_1}, u_{x_1^{\mathcal{J}_1}}^{\mathcal{O}_1}, a_{x_1^{\mathcal{J}_1}}^{\mathcal{O}_1})$	$w_{x_1^{\mathcal{J}_1}}^{\mathcal{O}_1} = (0.3, 0.5, 0.2, 0.3)$	$t_{x_1^{\mathcal{J}_1}}^{\mathcal{O}_1} = 0.26$
$\mathcal{O}_2$	$w_{x_1^{\mathcal{J}_1}}^{\mathcal{O}_2} = (b_{x_1^{\mathcal{J}_1}}^{\mathcal{O}_2}, d_{x_1^{\mathcal{J}_1}}^{\mathcal{O}_2}, u_{x_1^{\mathcal{J}_1}}^{\mathcal{O}_2}, a_{x_1^{\mathcal{J}_1}}^{\mathcal{O}_2})$	$w_{x_1^{\mathcal{J}_1}}^{\mathcal{O}_2} = (0.7, 0.1, 0.2, 0.2)$	$t_{x_1^{\mathcal{J}_1}}^{\mathcal{O}_2} = 0.74$
$\mathcal{O}_3$	$w_{x_1^{\mathcal{J}_1}}^{\mathcal{O}_3} = (b_{x_1^{\mathcal{J}_1}}^{\mathcal{O}_3}, d_{x_1^{\mathcal{J}_1}}^{\mathcal{O}_3}, u_{x_1^{\mathcal{J}_1}}^{\mathcal{O}_3}, a_{x_1^{\mathcal{J}_1}}^{\mathcal{O}_3})$	$w_{x_1^{\mathcal{J}_1}}^{\mathcal{O}_3} = (0.4, 0.3, 0.3, 0.8)$	$t_{x_1^{\mathcal{J}_1}}^{\mathcal{O}_3} = 0.64$
$\mathcal{O}_4$	$w_{x_1^{\mathcal{J}_1}}^{\mathcal{O}_4} = (b_{x_1^{\mathcal{J}_1}}^{\mathcal{O}_4}, d_{x_1^{\mathcal{J}_1}}^{\mathcal{O}_4}, u_{x_1^{\mathcal{J}_1}}^{\mathcal{O}_4}, a_{x_1^{\mathcal{J}_1}}^{\mathcal{O}_4})$	$w_{x_1^{\mathcal{J}_1}}^{\mathcal{O}_4} = (0.2, 0.6, 0.2, 0.3)$	$t_{x_1^{\mathcal{J}_1}}^{\mathcal{O}_4} = 0.26$

The combined opinion for $x_1^{\mathcal{J}_1}$ is calculated as:

$$w_{x_1^{\mathcal{J}_1}}^{\mathcal{O}} = w_{x_1^{\mathcal{J}_1}}^{\mathcal{O}_1} \oplus w_{x_1^{\mathcal{J}_1}}^{\mathcal{O}_2} \oplus w_{x_1^{\mathcal{J}_1}}^{\mathcal{O}_3} \oplus w_{x_1^{\mathcal{J}_1}}^{\mathcal{O}_4} = (0.48, 0.46, 0.06, 0.30)$$

Thus, the opinion's probability exception value for axiom x_1 based on Equation 6.5 is:

$$t_{x_1^{\mathcal{J}_1}}^{\mathcal{O}} = b_{x_1^{\mathcal{J}_1}}^{\mathcal{O}} + a_{x_1^{\mathcal{J}_1}}^{\mathcal{O}} \times u_{x_1^{\mathcal{J}_1}}^{\mathcal{O}} = 0.48 + 0.30 \times 0.06 = 0.50$$

The opinions for $x_2, x_3 \in \mathcal{J}_1$ are calculated accordingly.

Axioms in $\mathcal{J}_2$ already have values in $\mathcal{J}_1$, so we follow the conditional deduction, as they are dependent on $\mathcal{J}_1$. For instance, for axiom x_1 in $\mathcal{J}_2$, the proposition $\mathcal{P}$ is written as:

"$x_1^{\mathcal{J}_2}$ is trustworthy iff $t_{x_1^{\mathcal{J}_1}}^{\mathcal{O}} = 0.50$, and $t_{x_2^{\mathcal{J}_1}}^{\mathcal{O}} = 0.27$".

Then, the conditional opinion for this axiom is calculated as:

$$w_{x_1^{\mathcal{J}_2}||(x_1^{\mathcal{J}_1}, x_2^{\mathcal{J}_1})} = w_{x_1^{\mathcal{J}_2}||x_1^{\mathcal{J}_1}} \oplus w_{x_1^{\mathcal{J}_2}||x_2^{\mathcal{J}_1}} = 0.2.$$

The remaining axioms follow the same process. As a result, $x_1 \in \mathcal{J}_1$, $x_6 \in \mathcal{J}_2$, and $x_{14} \in \mathcal{J}_3$ will be presented to the user for revision, as they are the least trustworthy in their respective justification sets.

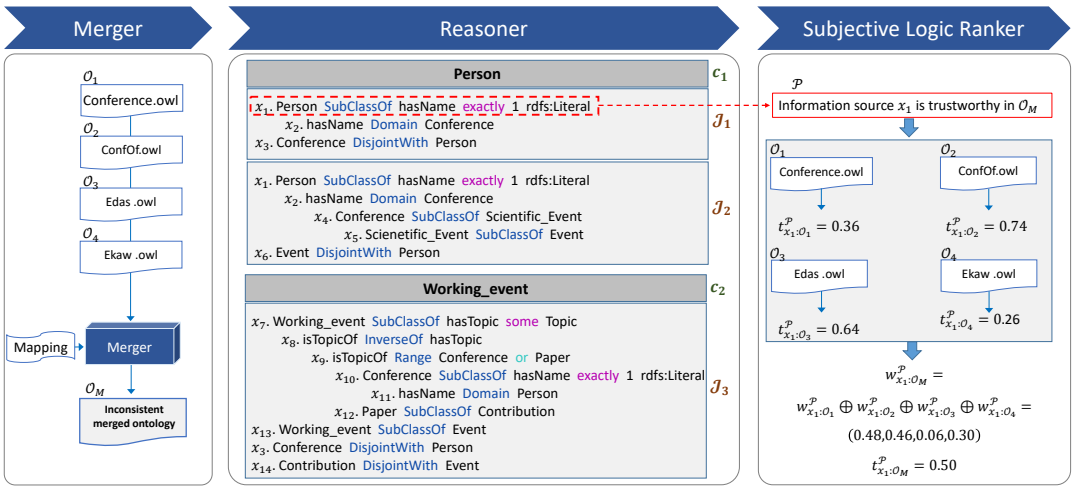


FIGURE 6.5: Ontologies are merged, and the result is inconsistent; The reasoner detects the justification sets and unsatisfiable concepts of the merged ontology; the Subjective Logic-based method ranks the axioms belonging to the justifications. Opinions from four source ontologies are combined by the consensus ($\oplus$) operator for one proposition $\mathcal{P}$.

6.8 Summary

The main characteristics of our proposed method summarized as:

- We tackle inconsistencies arising in the merged ontology, concerning the respective source ontologies.
- We utilize the Subjective Logic theory to combine several criteria in order to rank the conflicting axioms by belief, disbelief, and atomicity values.
- We provide the systematic formulation for the positive $r_x^{\mathcal{O}_i}$ and negative $s_x^{\mathcal{O}_i}$ observation alongside with atomicity $a_x^{\mathcal{O}_i}$ in the context of an inconsistent merged ontology.
- The opinions of the agents have been combined in our prototype by the consensus operators of the Subjective Logic.
- When conflicting axioms belonging to a justification set are dependent on other sets, we employ the conditional ranking.

7 Quality Assessment for the Merged Ontology

This chapter starts with an introduction to the proposed quality evaluation method in Section 7.1, followed by preliminaries and relevant background knowledge in Section 7.2. In Section 7.3, we present a set of evaluation principles. Our proposed framework for the evaluation of the quality of the merged ontology dimensions is described in Section 7.4, followed by our quality assessment workflow in Section 7.5. Finally, we present a summary of this chapter in Section 7.6.

Moreover, a list of used notations, symbols, and nomenclature is shown in Table 7.1. The contents of this chapter have been previously published in [BGKR20b].

TABLE 7.1: The used notations, symbols, and nomenclature in Chapter 7.

Notation	Description
$\mathcal{O}$	an ontology
$\mathcal{O}_S$	a set of source ontologies
$\mathcal{O}_M$	a merged ontology
$\mathcal{M}$	a mapping between source ontologies
$\mathcal{O}_k(L)$	ontology using logical language L under commitment k
$E_{\mathcal{O}_M}$	quality evaluation function
D	an evaluation dimension
S	a set of inputs for the evaluation function
$\dot{c}$	coefficient of measurement error
mp	a measurement procedure of evaluating quality indicators
m	the output (numerical value) of $E_{\mathcal{O}_M}$
L	logical language
$\mathbb{C}$	conceptualization
$I_k(L)$	intended model
k	commitment to certain I for L
$\triangle$	a set of relevant entities
W	a set of possible worlds
R	a set of intensional relations
TP	True Positive
FP	False Positive
FN	False Negative
TN	True Negative
$n_{TP}^{\mathcal{O}_k(L)}$	number of TP of $\mathcal{O}_k(L)$
$n_{FP}^{\mathcal{O}_k(L)}$	number of FP of $\mathcal{O}_k(L)$
$n_{FN}^{\mathcal{O}_k(L)}$	number of FN of $\mathcal{O}_k(L)$
$\mathbb{P}$	Precision
$\mathbb{R}$	Recall
$P1$-$P22$	principles of the evaluation standards
CQ	Competency Question
GMR	General Merge Requirement
R1-R20	individual GMRs

7.1 Introduction

The ontology merging process plays an essential role within different Semantic Web applications. The merged ontologies have a central role in realizing real-world applications. Thus, there is a strong need to develop evaluation methods that can measure their quality.

Most studies in ontology merging lack sufficient experimental evaluation on the merged result. Some of these studies evaluate the accuracy of their generated alignment on the merge scenario (cf. [MFBB10; MTFH14]). The few proposed benchmarks [MFH16; RR12] are restricted to a few criteria, only, and thus can not be adequately used for the quality evaluation of merged ontologies. Ontology merging systems such as [RR14; FRP14; JERS+11] that use criteria-based techniques to evaluate their methods are usually limited to a few measures and do not fully cover quality aspects. Manual evaluation of merged ontologies is a complex, error-prone, and labor-intensive task for users or experts, especially for large-scale ontologies. Automatic evaluation of merged ontologies can provide a wide range of criteria in different quality aspects. Thus, it helps end-users and experts to obtain comprehensive knowledge on strengths and weaknesses in order to build trust for sharing and reusing merged ontologies. If applicable, they can drive further improvements before involving merged ontologies in their desired application.

To evaluate merged ontologies systematically, we adapt evaluation dimensions from two well-known ontology evaluation frameworks [DRFBSAG+11; GCCL05] and customize them in the context of ontology merging. These two works introduced structural and functional evaluation dimensions. Moreover, the usability in [GCCL05], and the reliability, operability, and maintainability dimensions in [DRFBSAG+11] are presented. We build our criteria on top of these classifications, formulate them, and analyze how these dimensions can be practically evaluated on merged ontologies. Associated indicators introduced in each dimension are designed by considering evaluation standards in [BBL76; DGFE16]. The novelty of this research is to bring these dimensions with systematic formulations in the context of the evaluation of merged ontologies and make them feasible in practice. A set of experimental tests are conducted in Chapter 13 to demonstrate the feasibility of our assessment.

7.2 Preliminaries & Background

This section introduces the used notations and briefly discusses relevant background knowledge.

An ontology merging evaluator takes as input a merged ontology alongside respective source ontologies, their mappings, and user parameters. It generates a set of quality indicator values as output. Precisely:

Definition 7.1. *An **ontology merging evaluator** measures the quality of the merged ontology $\mathcal{O}_M$ based on a set of source ontologies $\mathcal{O}_S$, their mappings $\mathcal{M}$, and user parameters concerning a set of evaluation quality characteristics.*

An ontology is a fairly complex structure with several interdependent measurable features. It is often more practical to focus on the evaluation of different levels of the ontology separately rather than evaluating the ontology as a whole. Thus, a set of various quality characteristics is required for measuring different quality levels [GCCL05]. Evaluating merged ontologies for a particular characteristic depends on the evaluation of its set of associated sub-characteristics. Likewise, the evaluation of a particular sub-characteristic depends on its associated indicators. Thus, quality indicators are statistical measures that indicate output quality. Each quality indicator is a unit of measurement of quality evaluation. In this view, the quality is presented in different levels of abstraction, characteristics (*dimension*), sub-characteristics (*aspect*), and sub-sub-characteristics (*indicator*):

Definition 7.2. *A **dimension** shows the top-level particular quality evaluation of merged ontologies. It consists of several **aspects**, each depending on associated **indicators**.*

Given these preliminaries, the remaining part of this section will address:

- The difference between ontology evaluation and ontology ranking
- Insufficiency of the global scoring
- Different domains of evaluation
- Towards a customizable evaluation of merged ontologies

7.2.1 Ontology Ranking vs. Ontology Quality Evaluation

Ontology ranking may be considered similar to ontology quality evaluation in that they both try to evaluate ontologies based on a set of quality indicators. However, they are performed for different purposes.

- Approaches to ontology ranking aim to select the best ontology among many candidates for a given purpose in a project or a particular task. Thus, in this respect, ontology ranking essentially is similar to a decision-making problem.
- Ontology evaluation aims to identify the strengths and flaws of a given ontology. Hence, users and developers can make informed decisions according to their needs for the evaluated ontology.

7.2.2 Insufficiency of Global Scoring

The principle of software quality in [BBL76] shows that there is no single quality indicator that can give a universally useful rating of software quality. The authors showed that calculating and understanding the value of a single, overall indicator for software quality may be more trouble than it is worth. Usually, a global score is calculated by obtaining the weighted average of the scores of associated quality indicators. However, obtaining an overall quality rating with the union over the single indicators suffers two main drawbacks:

1. **Weight optimization**: Considering equal weights for all associated indicators is not a fair way to obtain an overall quality rating. Because quality indicators are

not equally important for users or associated applications and projects. Users, indeed, are interested in some indicators, not all. Determining different weights is a challenging task. It mostly needs a *preference ordering function* [GCCL05], achievable by user intervention to prioritize indicators and apply local constraints. Subsequently, for any evaluated ontology, this preference ordering function should be set individually by the user.

2. **Conflicting quality indicators**: High quality of all indicators is hard to achieve simultaneously as they are usually in conflict. For instance, achieving a higher value for the coverage indicator often conflicts with obtaining low redundancy because preserving all properties from source ontologies in the merged ontology frequently causes path redundancy. Thus, coverage and redundancy indicators might not be simultaneously fully achieved. Moreover, users generally find it difficult to quantify their preferences in such conflict situations [BBL76].

Therefore, by assessing the quality of merged ontologies, our goal is not to assign a global score for the evaluation of the merged ontology. As suggested in [BBL76], the best use for indicators is as individual anomaly indicators to be used as guides for further developments and usages.

7.2.3 Evaluation Domains

In our quality evaluation framework, the merged results and the underlying merge method are evaluated. The merge method influences the merge result. Thus, our quality evaluation covers both parts:

- **Evaluation of the merge method**: By analyzing merged ontologies, it can be observed how well the underlying merge method is performed. For example, the entity coverage percentage in merged ontologies depends on how the merge method is carried out. Thus, the accuracy of the merge method is indicated with some aspects of our framework.
- **Evaluation of the merge result**: Through some aspects of our framework, the correctness of the merge result is evaluated independently of the underlying merge method. It is related to the evaluation of the merged ontologies themself. For instance, whether the merged ontology is consistent, or includes cycles, is related to evaluating the merge result.

7.2.4 A Customizable Evaluation

There is no single best or preferred quality indicator for ontology evaluation [BGM05]. Instead, choosing an evaluation indicator should depend on the purpose of the evaluation, the application in which the ontology is to be used, and the aspects of the ontology we are trying to evaluate. Thus, users should be able to choose the desired quality indicators for their given purposes or applications. A practical step is taken toward this customizable evaluation. To achieve this, we provide users a variety of quality indicators, by which they can customize the required indicators for evaluating a given merged ontology based on the aim of the evaluation. For each adjusted indicator,

users receive detailed analysis and can observe the weakness and strengths of the evaluated ontology.

7.3 Evaluation Standards

To develop an evaluation framework for the merging process and its result, i.e., the merged ontology, we follow the well established Goal-Question-Metric (GQM) approach [Bas92] from the software evaluation field. GQM is based on the assumption that the evaluation of any system should be an evaluation of fitness for purpose. Thus, the evaluation process should be preceded by the identification of the engineering goals behind the system or technology to be evaluated. The goals are defined in an operational, tractable way by refining them into a set of quantifiable questions that are used to extract the appropriate information. The questions are then used to define a specific set of metrics. Since our aim is to develop a general framework and not specified it to a special system, we also seek a general goal. The goals and defining questions are described in the following:

Goal 1. The merged ontology should provide a comprehensive knowledge concerning the given source ontologies.

Q1. Is the knowledge from the source ontologies reflected in the merged ontology?

Q2. Do the evaluated aspects of the quality of the merged ontology present a comprehensive evaluation?

Goal 2. The merge process should be user friendly and customizable to user requirements.

Q3. Can the merged ontology reflect the user requirements?

Q4. Is the result of the evaluation clear for users?

According to the GQM paradigm, these questions can be used to define the set of criteria that should be employed to evaluate the merged ontology with respect to the presented goals. We will introduce these criteria in Section 7.4. Moreover, we considered existing evaluation standards [DGFE16; BBL76] and took them into account in designing our quality indicators. These works are summarized as follows:

- In [DGFE16], four standards are introduced to provide a framework for determining a good evaluation for the evaluation society. Standards claim to substantiate the idea of professional evaluation. The evaluation includes four principles, (1) usefulness, (2) feasibility, (3) fairness, and (4) accuracy. Each principle is presented with a group of sub-statements. We consider all the principles above and adapt the respective sub-statements. Note that some sub-statements cannot be aligned in our context, as they are related to human evaluation, such as human error checking.
- In [BBL76], a conceptual framework is established for analyzing the characteristics of software quality. We adapt the respective characteristics as sub-statements within the principles introduced in [DGFE16].

Accordingly, we address the following principles and sub-statements (given by Pi) for evaluating the quality of merged ontologies:

- **Usefulness principle**:

  P1. Detecting involved and affected variables: Issues affecting evaluation and those affected by it should be identified beforehand to take them into account as far as possible when creating the evaluation.

  ➔ In our context, analyzing the source ontologies should be considered, since they are affected by evaluation and merge results. The application that wants to use the evaluated ontology should be taken into account as well because it affects evaluation.

  P2. Transparency of the impact of influenced variables: The impact of those involved in and those affected by evaluation should be transparent and clear. These values can be used for the classification and interpretation of the results.

  ➔ The importance degree of the source ontologies and the application that wants to use the merged ontology should be clear to use these degrees in concluding the result and interpreting the output.

  P3. Clarification of evaluation purposes: The purpose of the evaluation is varied with quality dimensions. Thus, the purpose of each aspect of quality dimensions should be clarified separately for the users.

  ➔ The purpose of each evaluation aspect should be clarified and defined for the user.

  P4. Competence and credibility of the underlying system: The system (environment or framework) that conducts evaluations should be competent. Thus, the result of the evaluation can be acceptable.

  ➔ The reliability of the evaluation tool and framework should be approved.

  P5. Selection and scope of indicators: The selection and scope of indicators should be considered based on evaluation goals.

  ➔ Users should be able to adjust the scope of the evaluation. Thus, they can adjust desired quality indicators based on the purpose of the given task.

  P6. Automation: The feasibility of the automated evaluation is admired. However, it depends on the supported aspects of the framework. An automated ontology evaluation is a necessary precondition [BGM05] for the healthy development of automated ontology processing techniques.

  ➔ Evaluation is desired to be automatic as possible for the given quality dimension.

  P7. Completeness of the output: The result of the evaluation should be complete and comprehensible.

➔ The output of evaluation should provide all essential quality aspects by covering different quality indicators in order to be comprehensive and complete.

P8. Clarity of the result: The result of the evaluation should be clear and understandable for the user.

➔ Presenting the result of the evaluation should be understandable for the user. The understandability of the result can be augmented with a user-friendly GUI with ease of use feature.

P9. Use and benefits of evaluation: The results should be useful, and the users should get benefit from the evaluation.

➔ The evaluation criteria are intended to emphasize existing gaps and weaknesses to provide better insight for the user and suggest a possible solution when it is applicable. Accordingly, the user could get benefits from evaluation results.

- **Feasibility principle**:

P10. Appropriate methods: Evaluation procedures should be chosen in such a way that, on the one hand, the evaluation is carried out professionally in accordance with the requirements. On the other hand, the effort for those involved and those affected is kept in an appropriate ratio to the intended benefit of the evaluation.

➔ The evaluation methods should consider the requirements and keep in balance the effort for analyzing source ontologies (those involved in) and the application that wants to use the merged ontology (those affected by) with an adequate ratio to the desired evaluation output.

P11. Efficiency of evaluation: The effort for evaluation should be in reasonable proportion to the benefits of the evaluation.

➔ The amount of effort on producing the desired results should be feasible in practice (with a reasonable complexity), and the output of the evaluation should be in appropriate with this complexity.

- **Fairness principle**:

P12. Formal disposal of indicators: The quality indicators, process, and goal of the evaluation should be written and available for users.

➔ Each quality indicator is expected to have an exact and systematic definition, certified goal, and precise implementation of the process in practice. They should be available for users.

P13. Disclosure of results: The results of the evaluation should be made available.

➔ The output of the evaluation should be available for users.

P14. Comprehensive and fair examination: Evaluations should examine and present the strengths and weaknesses of the object that wants to be evaluated as fairly and comprehensively as possible.

➔ In the quality evaluation of the merged ontology, both strengths and weaknesses should be reported to users. The evaluation should be comprehensive by covering various aspects.

P15. Impartial implementation and output: The evaluation process and result should be impartial and unbiased with respect to quality indicators or environments.

➔ The merge evaluation techniques should be independent of the underlying merge method and environment.

- **Accuracy principle**:

P16. Description of the object to be evaluated: The theoretical aspect of the object that wants to be evaluated should be described and documented precisely and comprehensively.

➔ The theoretical aspect of the given merged ontology and the way that it is created upon should be comprehensively and accurately described.

P17. Description of purposes and procedures: The purposes and procedures of the evaluation itself, including the underlying evaluation methods, should be documented and described so precisely that they can be understood and assessed.

➔ For each aspect that the user wants to evaluate, there should be a well-documented description of its objective and implementation of the evaluation's function.

P18. Context analysis: The context of the object that wants to be evaluated should be analyzed in a sufficiently comprehensive and detailed manner and taken into account when interpreting results.

➔ The merged ontology should be evaluated via some criteria for a given context.

P19. Declaration of indicators: The quality indicators that are used for evaluation should be documented with sufficient accuracy so that the reliability and appropriateness of them can be assessed.

➔ On what basis the indicators are defined should be documented.

P20. Valid and reliable indicators: The quality indicators should be valid and reliable.

➔ On what basis the indicators are defined should be valid.

P21. Justified assessments and conclusions: The evaluative statements made in an evaluation should be based on explicit criteria and target values. Conclusions should be explicitly justified based on the given inputs so that they can be understood and assessed.

➔ Concluding the quality of the merged ontology should be based on the respective source ontologies, their mapping, and parameters.

P22. Meta-evaluation: Meta-evaluation should evaluate evaluations. In order to make this possible, evaluations should be documented, archived, and made accessible as far as possible in a suitable form.

➔ A meta-evaluation on the output of the evaluation framework should be carried.

We consider these sub-statements of principles in designing our quality indicators. In Section 13.7 of Chapter 13, we demonstrate to which extent our evaluation framework fulfills the principles.

7.4 Proposed Quality Indicators for the Evaluation of Merged Ontology

In this section, we first present how we extend the existing ontology evaluation framework in our context in Section 7.4.1. We then show the quality evaluation function in Section 7.4.2. Finally, we present the quality indicators in Section 7.4.3.

7.4.1 Extending Ontology Evaluation Frameworks

We use well-known existing ontology evaluation frameworks [DRFBSAG+11; GCCL05] and adapt in the evaluation of merged ontologies. Their main features are:

- Gangemi et al. [GCCL05] provided a comprehensive framework for ontology evaluation. At first, the authors created a meta-ontology to provide a meta-theoretical foundation. In this model, ontologies are considered semiotics objects to distinguish the different aspects of ontology engineering in practice. Upon the created meta-ontology, the authors built three evaluation dimensions: structural, functional, and usability. In the structural dimension, an ontology is an (information) object. In the functional dimension, it is a language (information object + intended conceptualization). From the usability viewpoint, the ontology's meta-language (the profile about the semiotic context of an ontology) is considered.
- OQuaRE framework [DRFBSAG+11] evaluates the quality of ontologies based on the software quality standard, ISO/IEC 25000, to make quality evaluation reproducible. ISO/IEC 25000 [IEC05] is a standard for Software product Quality Requirements and Evaluation known as SQuaRE. This standard defines a complete evaluation process of a software product. It suggests a series of quality characteristics that should be used for measuring quality. The authors considered ontologies as software artifacts. Thus, ontologies are viewed as the result of the application of a construction process and are evaluated as products, independently of the particular development process. In this regard, the OQuaRE framework provides five adapted dimensions: structural, functional, reliability, operability, and maintainability.

Next, we explain the introduced dimensions and their associated aspects. For each introduced quality aspect, we illustrate to which extent they are applicable for evaluating merged ontologies.

1. **Structural dimension**: It focuses on the syntax and logical properties of an ontology.
    - *Aspects*: The structural dimension is evaluated against several aspects, such as cohesion, redundancy, consistency. The units of measurements are calculated through a wide range of graph property indicators such as depth, breadth, cycle, and density.
    - *Adaptation*: In Chapter 5, we have identified Generic Merge Requirements (GMRs) as important criteria within the ontology merge process. Thus, for this dimension, we focus on them. Indeed, GMR classification is conducted based on analyzing three different research areas, including ontology merging methods, benchmarks, and ontology engineering. Thus, it comprehensively considers all topological properties of the structural characteristics of merged ontologies. Note that some aspects of evaluation frameworks mentioned above [GCCL05; DRFBSAG+11] such as redundancy or cycle are also included in GMRs. Moreover, we include the consistency measurement of merged ontologies as proposed in Chapter 6.
2. **Functional dimension**: It is related to the intended use and semantics of a given ontology. The evaluation within this dimension is based on the degree to which functional requirements are accomplished, that is, the appropriateness for its intended purpose.
    - *Aspects*: In [GCCL05], the functional dimension is evaluated against task and topic assessment, user satisfaction, modularity, and agreement assessment aspects. Precision and recall are calculated for each based on how these aspects are achieved against the built ontology. In [DRFBSAG+11], several aspects, such as reference ontology, knowledge reuse, and inferencing, are introduced.
    - *Adaptation*: To make the functional evaluation feasible in practice within our framework, we use the task assessment aspect and will embed other aspects in our future work. We use query evaluation and Competency Questions answering to measure the intended use and semantics of merged ontologies. This adaption is made in such a way that the tasks achieved by the source ontologies and those obtained from merged ontologies are compared to quantify precision and recall.
3. **Usability dimension**: It focuses on the ontology profile and communication context of an ontology.
    - *Aspects*: This dimension is evaluated against a set of annotation aspects in all steps of the ontology life-cycle.
    - *Adaptation*: Annotation information on the life-cycle of merged ontologies might not always be available. This is because merged ontologies can be

created by a merge method or human intervention, which might not be available during the evaluation process. Thus, we consider those quality indicators on the ontology and entities annotation, which can be estimated in reality for a given merged ontology and might be affected by the ontology merging process.

4. **Reliability dimension**: It evaluates the capability of an ontology that maintains the level of performance under stated conditions for a given period of time.
    - *Aspects*: This dimension is evaluated by recoverability and availability aspects.
    - *Adaptation*: We consider these aspects in the quality indicators of the usability dimension, such as availability of the ontology URI, and meta-information on ontology annotation.
5. **Operability dimension**: It evaluates the effort needed to use an ontology, and in the individual assessment of such use, by a stated or implied set of users.
    - *Aspects*: The operability dimension is measured through learnability, ease of use, and helpfulness aspects.
    - *Adaptation*: The introduced aspects require a user-study, which is considered this dimension for our future work.
6. **Maintainability dimension**: It evaluates the capability of ontologies that can be modified for changes in environments, requirements, or functional specifications.
    - *Aspects*: The maintainability dimension is evaluated against several aspects, including modularity, reusability, analysability, changeability, modification stability, and testability. These aspects are defined with several quality indicators, such as the mean number of properties per class, depth of subsumption hierarchy, and the number of children.
    - *Adaptation*: We do not consider maintainability as a separate dimension. Because, the aspects of the maintainability dimension are evaluated by the quality indicators of topological properties such as the mean number of properties per class, or depth of classes. Such indicators are related to the structural dimension, which we consider in the structural dimension. Moreover, considering the statistical properties of ontology elements are not an appropriate way to evaluate the maintainability aspects such as modularity, changeability, or reusability. For example, the authors claimed that the number of properties affects reusability because having a more precisely defined ontology makes its knowledge more reusable. Likewise, they claimed that if the mean number of properties per class is large, it can be concluded that this ontology has good modularity. It is hard for us to accept these arguments straightforwardly.

The criteria that we adapted above are according to the goals and questions mentioned in Section 7.3. In particular, the structural quality dimension is based on question one (Q1), in which the coverage of the merged ontology is evaluated in terms of structure and

entity preservation. Regarding question two (Q2), we seek to provide a comprehensive evaluation by combining different quality dimensions. The quality criteria in the functional dimension are derived according to Q3. By adapting the usability dimension, we aim to concern the clarity and usability of the merged ontology (concerning Q4).

In the following, before describing each dimension in detail, a quality evaluation function applicable for all dimensions is presented and the respective parameters are clarified separately for each introduced dimension.

7.4.2 Quality Evaluation Function

Inspired by the general function of the structural dimension evaluation in [GCCL05], we present the quality evaluation function applicable for all introduced dimensions.

The evaluation of the merged ontology ($E_{\mathcal{O}_M}$) is the process of achieving a quality value (m) for the given quality dimension (D) on a set of inputs (S) by running a measurement procedure (mp). In a formal way, the quality function for evaluating the merged ontology $E_{\mathcal{O}_M}$ is given in Equation 7.1.

$$E_{\mathcal{O}_M} = (D, S, mp, \dot{c}) \tag{7.1}$$

in which:

- D (Dimension) is the quality dimension we want to measure.
- S (Set of input) is a set of inputs.
- mp (Measurement procedure) is the procedure executed to perform the measurement and achieve the quality value.
- $\dot{c}$ (Coefficient of measurement error) adjusts for context-related variations on measurement procedure.

Thus, the output value (m) of the quality evaluation function ($E_{\mathcal{O}_M}$) is obtained by applying a measurement procedure (mp) for a dimension (D) to a set of input (S) with considering a coefficient ($\dot{c}$), if available (see Equation 7.2).

$$mp_{D,S,\dot{c}} \xrightarrow[\text{yields}]{} m \tag{7.2}$$

The set of output values from quality indicators is the intended result for this function. The output shows information about the strengths and weaknesses of the given merged ontology. Upon that, users can draw further improvement in order to increase the quality of the merged ontology.

In Table 7.2, we show the variables of the quality evaluation function for each dimension separately. We specify the respective input S, measurement procedure mp, and output value m, as follows:

- Structural dimension:

TABLE 7.2: Variables of the quality evaluation function for each quality dimension.

Dimension	Input	Measurement Procedure	Output
Structural	Source ontologies Merged ontology Mappings	Counting function	Numerical value
Functional	Source ontologies Merged ontology Competency Question/Query	Counting function	Percentage value in range [0,1]
Usability	Source ontologies Merged ontology	Conditional boolean function	Boolean data (True/False)

  - S: The inputs of structural dimension evaluation are the set of source ontologies, the merged ontology, and the respective mappings.
  - mp: The usual measuring procedure, as stated in [GCCL05], is counting by a function that relates a set of entities to a numerical value in terms of the given quality indicator. The counting procedure might require a non-trivial algorithm. In Section 7.4.3.1, we present the counting function for each quality indicator. For some of them, we utilize a reasoner to perform that.
  - m: The counting procedure on evaluating the structural dimension generates numerical (absolute) values.

- Functional dimension:
  - S: Evaluation of the functional dimension requires as the input the merged ontology and the respective source ontologies along with a set of Competency Questions or queries for the given domain.
  - mp: We use a counting procedure to quantify precision and recall for the matching of merged ontology with its intended semantics or use.
  - m: Since the precision and recall are calculated as the fraction of two values, the output is obtained as a percentage value in the range [0,1].
- Usability dimension:
  - S: The input of this dimension is the merged ontology with respective source ontologies.
  - mp: The measurement procedure is carried with a conditional boolean function to validate whether the respective quality indicator in the merged ontology is satisfied (true) or not (false).
  - m: The functions for evaluating the existence, correctness, and satisfying the associated quality indicators return a Boolean data type with true or false values.

Next, we present our quality metrics for each evaluation dimension.

7.4.3 Associated Quality Indicators of Evaluation Dimensions

As we discussed in Section 7.4.1, we build our criteria on top of the classification introduced in [GCCL05; DRFBSAG+11], including structural, functional, and usability-related dimensions. Designing each associated quality indicator is carried by considering the standard principles in [BBL76; DGFE16] (see Section 7.3). To evaluate the merged ontology in a systematic way, we contribute to formulating the indicators within each dimension in the context of the ontology merging.

7.4.3.1 Measuring the Structural Dimension

The structural dimension focuses on evaluating syntax (e.g., graph structure), and formal semantics of the merged ontology. In this form, the topological and logical properties of the merged ontology are measured by means of an indicator.

As stated in Section 7.4.2, the quality evaluation function of the structural dimension returns numerical (absolute) values. Structural evaluation functions do not produce any relative values, such as those represented in [DRFBSAG+11]. Representing the relative values disputes:

- **Finding the appropriate ratio**: A relative value is obtained by a division operator, in which the dividend is divided by the divisor. To achieve a relative value for each quality indicator, the division operation calls into question how the divisor is determined. For instance, to present a relative value for the acyclicity indicator in a merged ontology, we are required to present the number of detected cycles based on the total number of entities, is-a relations, or the number of classes. All of them can be feasible since each represents a particular aspect of this quality indicator. However, none of them can properly illustrate the acyclicity indicator.
- **Impact of the individual anomaly**: For any given indicator, the influence of each detected anomaly is different from others. Thus, each anomaly requires a specific weight. Moreover, an equal weight can not be considered for all anomalies as they have different priorities. This implies assigning an individual weight for each anomaly within a quality indicator. For example, in the acyclicity indicator, all cycles cannot be treated with the same weight. The detected cycles might have different priorities and important levels. Indicating the effect of each anomaly effectively is hard to be accomplished.

These challenges prevent the fair presentation of relative values. Therefore, considering absolute values can overtake them. In this regards, for each quality indicator, we present the number of occurrences that the indicator is not satisfied in the merged ontology, a so-called anomaly. For instance, for the acyclicity indicator, we present the absolute number of cycles in the merged ontology; for the class coverage indicator, we present the absolute number of classes from the source ontologies that do not preserve in the merged ontology. To make this evaluation more helpful for the user, the name of those elements that do not meet the quality indicators will also be listed.

To classify the quality indicators in this dimension, we take advantage of GMR classification from Chapter 5. This also covers the criteria in the early work on ontology

evaluation [DB10]. GMRs consider the topological and structural properties of merged ontologies. In this follow, in the structural dimension, we evaluate integrity, logic properties, and model properties aspects alongside the consistency aspect, which are described as next.

7.4.3.1.1 Measures for integrity. The aim of this aspect is to evaluate the integrity of a merged ontology based on the knowledge provided by the source ontologies. This includes:

- *Coverage* [RR12] or *completeness* [DB10] to measure the percentage of entities from the source ontologies that are covered by the merged ontology [DB10]. We evaluate the coverage by:
  - *R1. Class*, *R2. Property*, and *R3. Instance preservation*: These indicators can be evaluated by counting the numbers of classes, properties, and instances from the source ontologies that do not have a mapped class, property, and instance in a merged ontology, respectively.
  - *R4. Correspondence preservation*: For *R4*, the evaluation function counts the numbers of given correspondences from source ontologies that are not mapped to the same entity in the merged ontology.
  - *R5. Correspondences' property preservation*: The evaluation function counts the numbers of corresponding entities' properties that are not included for the merged entities.
  - *R6. Property's value preservation*: The evaluation function counts the numbers of properties' values from source ontologies that are not preserved in the merged ontology.
  - *R7. Structure preservation*: The evaluation function counts the numbers of entities from the merged ontology that do not follow the same structure for each entity with respect to the source ontologies.
- *Redundancy* or namely, *minimality* checks that no redundant entity appears in the merged ontology [DB10; MFH16; RR12]. We evaluate the redundancy by:
  - *R8. Class*, *R9. Property*, and *R10. Instance redundancy prohibition*: The evaluation functions for these indicators count the numbers of classes, properties, and instances from the source ontologies that have more than one mapped entity in the merged ontology.
  - *R11. Extraneous entity prohibition*: For this indicator, the evaluation function counts the numbers of entities that are not present in the source ontologies but are in the merged ontology.

7.4.3.1.2 Measures for logic properties. Within this aspect, we evaluate the logical properties of the merged ontology, which can get the effect by the merge process:

- *Deduction satisfaction* refers to *R12*, *Entailment deduction satisfaction* indicator. With the help of the embedded reasoner, the evaluation function counts the numbers

of entailments from the merged ontology that are not included in the source ontologies' entailment. We follow subsumption, equivalence, and satisfiability entailments tests from [BPS11].

- *Constraint satisfaction* evaluates the ontology constraints by:
  - *R13. One type restriction*: The evaluation function counts the numbers of corresponding entities with different data types that do not have a compromised type in their respective merged entity.
  - *R14. Property value's constraint*: The evaluation function counts the numbers of source ontologies' restrictions on the properties' values (e.g., cardinality, enumerating values) that have not been satisfied in the merged ontology.
  - *R15. Property's domain and range oneness*: The evaluation function counts the numbers of properties that have multiple domains or ranges in the merged ontology.

7.4.3.1.3 Measures for model properties. The principles of creating a new ontology model are evaluated in this aspect, including:

- *Acyclicity* to evaluate the existence of cycles in a merged ontology by:
  - *R16*: *Acyclicity in the class hierarchy*, and *R17. Acyclicity in the property hierarchy*: For these indicators, the evaluation function counts the numbers of cycles between classes and properties concerning subClassOf and subPropertyOf relationships in the merged ontology, respectively.
  - *R18. Prohibition of properties being inverses of themselves*: An inverse recursive definition might happen by combining the corresponding entities and their existing inverse relationships. Thus, to evaluate this indicator, the evaluation function counts the numbers of inverse recursive definitions on the properties in the merged ontology.
- *Connectivity* to evaluate the satisfaction of hierarchical connectivity by:
  - *R19. Unconnected class*, and *R20. Unconnected property prohibition*: To evaluate *R19* and *R20* indicators, the evaluation function counts the number of unconnected classes and properties in the merged ontology, respectively.

7.4.3.1.4 Measures for consistency. Another aspect of the structural dimension introduced in [GCCL05; DRFBSAG+11] is the evaluation of ontology consistency. As mentioned earlier, an *inconsistent ontology* [HPS09] is an ontology that cannot have any models and entails everything. Thus, through an embedded reasoner, we detect whether an ontology is consistent. For an inconsistent ontology, the evaluation function counts the number of inconsistent classes and present them to users. As stated in the method presented in Chapter 6, we considered the knowledge of the source ontologies into account in order to rank conflicting axioms. We then present users the ranked axioms alongside suggestions for possible repairs to make the merged ontology consistent.

7.4.3.2 Measuring the Functional Dimension

The functional dimension is associated with the intended use of a given merged ontology and its components, i.e., its function. It focuses on the relations holding between the ontology graph and its intended meaning or semantics. The functional dimension is coincident with the main purpose of an ontology, i.e., specifying a given conceptualization, or a set of contextual assumptions about a world, which is expressed as a set of concepts, their definitions, and their inter-relationships [Usc96].

Functional measures have been quantified by precision and recall in [GCCL05]. In the context of ontology evaluation, this definition is adapted by choosing an appropriate domain for positive and negative responses from matching between the structure of the ontology and the intended usage and semantics. Given a logical language L that implicitly commits to a conceptualization $\mathbb{C}$, an ontology's purpose is to capture all models of L that are compatible with $\mathbb{C}$ and only those. These models are called intended models $I_k(L)$, k being the commitment to a certain interpretation I for L. In this view, an ontology $\mathcal{O}$ using L is a logical theory designed in such a way that the set $\mathcal{O}_k(L)$ of its models relative to $\mathbb{C}$ under the commitment k is a suitable approximation of the set $I_k(L)$. Thus, a conceptualization $\mathbb{C}$ of the ontology, as shown in Figure 7.1, (a), is demonstrated with a set of relevant entities $\triangle$, a set of possible world W, and a set of intensional relations R, given by $\mathbb{C} = (\triangle, W, R)$. From this perspective, precision $\mathbb{P}$ and recall $\mathbb{R}$ are defined by Equation 7.3 and Equation 7.4, respectively, where, TP=True Positive, FP=False Positive, and FN=False Negative.

$$\mathbb{P} = \frac{n_{TP}^{\mathcal{O}_k(L)}}{n_{TP}^{\mathcal{O}_k(L)} + n_{FP}^{\mathcal{O}_k(L)}} \tag{7.3}$$

Precision $\mathbb{P}$ is the proportion of intended models ${}_{TP}^{\mathcal{O}_k(L)} \subseteq I_k(L)$ (True Positives) over $\triangle$, on the sum of all $\mathcal{O}$ models $\mathcal{O}_k(L)$, which can include False Positives ${}_{FP}^{\mathcal{O}_k(L)} \nsubseteq I_k(L)$.

$$\mathbb{R} = \frac{n_{TP}^{\mathcal{O}_k(L)}}{n_{TP}^{\mathcal{O}_k(L)} + n_{FN}^{\mathcal{O}_k(L)}} \tag{7.4}$$

Recall $\mathbb{R}$ is the proportion of intended models ${}_{TP}^{\mathcal{O}_k(L)} \subseteq I_k(L)$ (True Positives) over $\triangle$, on the sum of all intended models $I_k(L)$, which can include False Negatives ${}_{FN}^{\mathcal{O}_k(L)} \subseteq I_k(L)$.

An ontology can accept unintended models, resulting in a lower precision, or can miss intended models, resulting in a lower recall. Accordingly, the merged ontology can be labeled with one of the following cases:

When the matching between the merged ontology and its intended use or semantics quantifies

- high precision and max recall, the functionality of the merged ontology can be marked by a GOOD label;

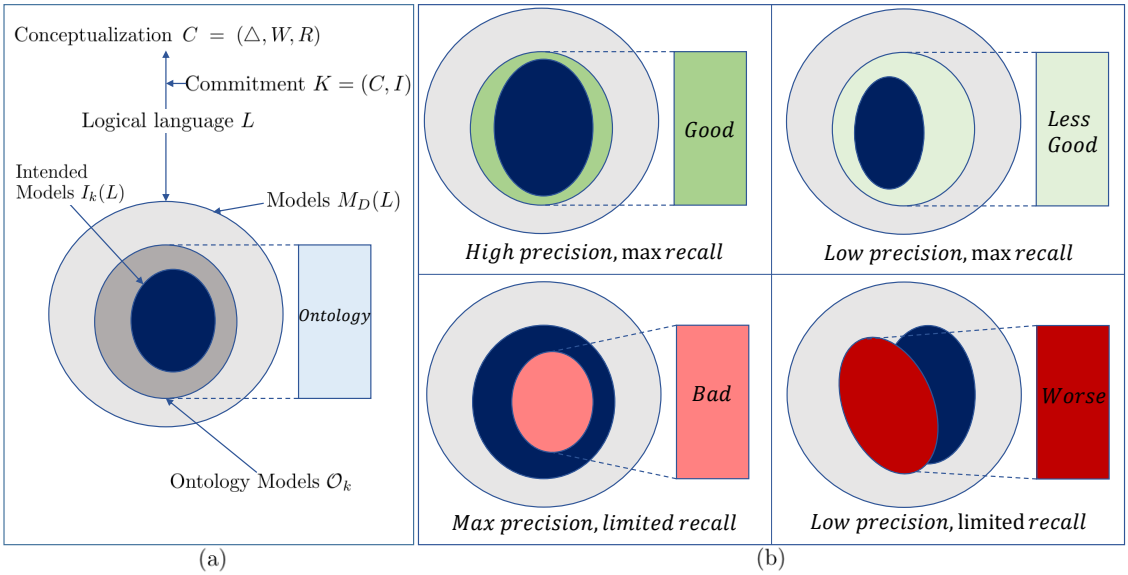


FIGURE 7.1: (a) The relationship between an ontology and a conceptualization. (b) Precision and recall of an ontology evaluation [GCCL05].

- low precision and max recall, the functionality of the merged ontology can be marked by a LESS GOOD label;
- max precision and limited recall, the functionality of the merged ontology can be marked by a BAD label;
- low precision and limited recall, the functionality of the merged ontology can be marked by a WORSE label.

Figure 7.1,(b) shows the possible cases resulting from this category.

The remaining challenges here are:

1. How to measure the intended use and semantics of a merged ontology in practice?
2. How to systematically determine true or false positive and negative responses to be able to calculate precision and recall?

We make our contribution in dealing with these two challenges in the next.

7.4.3.2.1 Measuring the intended use and semantics of the merged ontology. An intended conceptualization corresponds to the expertise of an ontology's intended users. The expertise boundary is provided by the task that should be accomplished with the help of the ontology [GCCL05]. Then, an ontology should be aimed at capturing at least the schema of that expertise. Ontology engineers have to elicit expertise from the associated developer, or assume a set of data (e.g., here, ontology) as a qualified expression of expertise and task. Thus, precision and recall of an ontology can be measured against: a) experts' judgment, or b) a data set assumed as a qualified expression of experts' judgment. We find two scenarios to accomplish it:

1. **Competency Questions answering**: One approach to capture the intended use of an ontology is to use Competency Questions (CQ)s [GCCL05]. A set of CQs is complete in the sense that if the ontology can provide correct answers to all questions, then the ontology can serve its intended purpose [Usc96].
2. **Query analyzer**: Query can provide the environment to capture the intended semantics of the merged ontology based on its respective source ontologies. Thus, we can measure the intended semantics by providing a list of queries which the source ontologies can or cannot answer, and then compare with the achieved answers from the merged ontology.

We will explain these two scenarios in detail in Section 13.2 of Chapter 13.

7.4.3.2.2 Measuring the true or false positive and negative responses. We determine true or false positive and negative responses in the context of the merged ontology. The responses from the merged ontology should be compared with the responses achieved from source ontologies. Thus, we define TP, FP, TN, and FN based on the expected answers of the source ontologies in both environments (tests with CQs or queries).

The exact formulation of TP, FP, TN, and FN in CQ and query scenarios will be explained in Section 13.2 of Chapter 13.

7.4.3.3 Measuring the Usability Dimension

This dimension focuses on the ontology profile to address the communication context of the merged ontology. An ontology profile is a set of ontology annotations, the metadata about an ontology, and its elements. We evaluate this dimension with two aspects (1) annotation about ontology itself, and (2) annotation about ontology's entities. In each aspect, we focus on those quality indicators concerning usability profiling in the context of the ontology merging.

7.4.3.3.1 Measures for annotation about the ontology itself. During the merge process, metadata of the source ontologies are combined in the merged ontology. Thus, in this aspect, we evaluate ontology annotation indicators in two directions: (i) existence, and (ii) correctness of the following indicators:

- *Ontology URI*: The ontology URI should not include file extension such as *.owl*, *.rdf*, or *.ttl* [AGL12]. Thus, the evaluation function for this indicator checks whether the merged ontology has a URI and the URI is correct.
- *Ontology namespace*: The evaluation function for this indicator checks that the merged ontology has a namespace. If so, the namespace should have the correct syntax.
- *Ontology declaration*: The ontology should not be failing to declare the `owl:Ontology` tag where the ontology metadata should provide it [PVGPSF14]. Thus, for this indicator, the evaluation function checks the existence and correctness of the ontology declaration.

- *Ontology license*: The merged ontology should contain a proper license. This license should be compatible with all licenses from the respective source ontologies, which requires modeling compatibility of different licenses.

7.4.3.3.2 Measures for annotation about ontology's entities. The existence of entity annotations has already been covered by $R2$ in the structural dimension (in the view of considering annotation as the properties). Here, we evaluate the remaining related aspects on entities' annotations, which might get an effect while creating the merged ontology:

- *Label uniqueness*: Since the ontology merging techniques combine the corresponding entities, they make remarkable changes to the labels of the corresponding entities. Thus, following [NM03], for this indicator, the evaluation function observes whether the created labels are unique.
- *Unify naming*: The name of entities from the source ontologies might be heterogeneous with each other. Thus, we aim to observe how the merge process follows the same convention in the merged ontology. Unify naming will increase the readability of the merged ontology. Consequently, readability influences usability quality. For this indicator, the evaluation function checks whether all entity's names follow the same naming conventions in the merged ontology, such as capitalization, or singular versus plural, as proposed in [NM+01; PVGPSF14].
- *Entity type declaration*: During the merge process, new entities might be created, or existing ones are combined into one integrated entity. It is important to check whether these entities have been explicitly declared with their respective types [HHP+10; PVGPSF14]. Thus, the evaluation function for this indicator checks whether all entities explicitly have been declared in the merged ontology.

7.5 Ontology Merging Quality Assessment Workflow

Figure 7.2 presents our proposed workflow for assessing the quality of the merged ontology. The inputs are source ontologies $\mathcal{O}_S$, the respective mappings $\mathcal{M}$, the merged ontology $\mathcal{O}_M$, and a set of Competency Questions (CQ)s or queries as user parameters. The output is a set of quality values. The ontologies and mapping are parsed, then, the evaluator engine assesses the quality of the merged ontology for the given dimension. Afterwards, the result is presented to the user through a GUI. If the evaluation of an indicator detects some anomalies, further analyzes will be presented to users. They can automatically repair some detected anomalies through the repair engine.

In practice, users can customize the set of evaluation criteria based on their needs. They can assess the quality of the merged ontology concerning criteria introduced in structural, functional, usability, and consistency dimensions. Moreover, users can compare the source and merged ontologies through the SPARQL endpoint. There is a set of embedded query patterns for users. However, they can customize their own query and compare the results of the merged and source ontologies simultaneously.

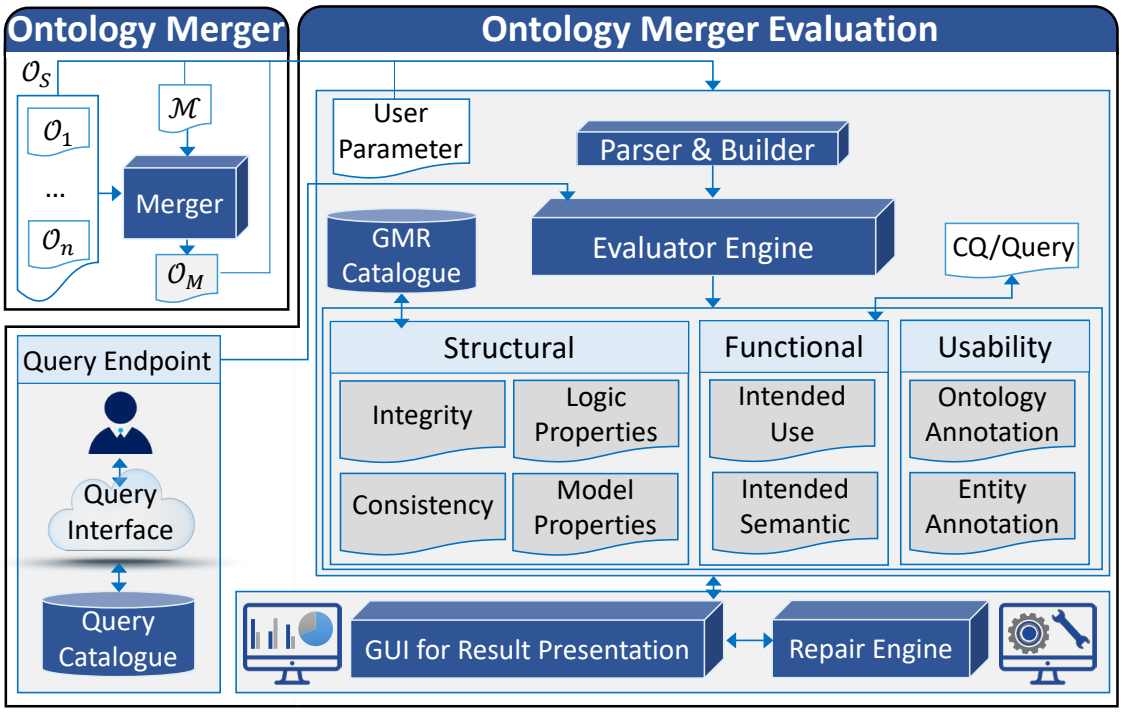


FIGURE 7.2: Workflow of ontology merging quality assessment.

7.6 Summary

In this chapter, we presented a quality assessment framework for merged ontology within structural, functional, and usability dimensions. Each quality dimension is presented with a set of aspects and associated quality indicators. An overview of our evaluation's dimensions is shown in Table 7.3. For each dimension, we present the associated aspects and quality indicators. Moreover, in the last column, we show the evaluation type of each indicator (as introduced in Section 7.2.3).

TABLE 7.3: Dimension, aspect, indicator, and type of the ontology merging evaluation task.

Dimension	Aspect	Indicator	Type
Structural	Integrity	Coverage (*R1-R7*)	merge method
		Redundancy (*R8-R11*)	merge method
	Logic Properties	Deduction(*R12*)	merge method
		Constraint (*R13-R15*)	merge method
	Model Properties	Acyclictly (*R16-R18*)	merge result
		Connectivity (*R19-R20*)	merge result
	Consistency	number of inconsistent classes	merge result
Functional	Intended Use	Competency Questions Answering	merge result
	Intended Semantics	Query Analyzer	merge method
Usability	Ontology Annotation	Ontology URI, namespace declaration, license	merge result
	Entity Annotation	Label uniqueness, unify naming entity type declaration	merge result

Part III

Evaluation

8 Experimental Evaluation

This chapter starts with presenting the dataset in Section 8.1, which is used in the experimental test of the thesis. After that, the detail of the implementation is presented in Section 8.2. Finally, an overview of the experimental test is shown in Section 8.3.

8.1 Datasets

To evaluate the applicability of our approach, we have aimed to use in our experiments a wide variety of ontologies in terms of number and size. We have selected twelve datasets, given by d_1-d_{12} (see Table 8.1). Our datasets include sets of ontologies from the conference (d_1-d_6), anatomy (d_7) and large biomedical (d_8) tracks of the OAEI benchmark[1] along with ontologies from BioPortal[2] in the domains of biomedical resource (d_9), health (d_{10}), and the union of both (d_{11}) as well as a combination of several subdomains (d_{12}). The dataset includes a variety of ontologies with different axioms' size ($134 \leq |Sig(\mathcal{O}_S)| \leq 30364$) and numbers ($2 \leq n \leq 55$).

We conducted our tests with two different types of correspondences (mappings):

- A perfect mapping $\mathcal{M}$ from the OAEI benchmark and BioPortal.
- An imperfect mapping $\mathcal{M}'$ which is produced by an ontology matching system.

While the first shows the general potential of the approach, the latter shows its applicability in a realistic setting where typically no perfect mapping is available.

Table 8.1 shows our datasets. The number of source ontologies n in each dataset is shown in *column 2*. The name of source ontologies and their axioms size $Sig(\mathcal{O}_S)$ are shown in *columns 3-4*. The number of corresponding classes $\mathcal{M}^{(')}_{|C|}$ and properties $\mathcal{M}^{(')}_{|P|}$ for imperfect and perfect mappings has been presented in *columns 5-8* in Table 8.1, respectively. To generate the imperfect mapping $\mathcal{M}'$, we use SeeCOnt [ABKD15] with F-Measure 50%-80%.

[1] http://oaei.ontologymatching.org/2019/

[2] https://bioportal.bioontology.org/;accessedat01.10.2019

TABLE 8.1: Dataset statistics: The number (n), name, and size ($Sig(\mathcal{O}_S)$ of source ontologies with the number classes $|C|$ and properties $|P|$ of their imperfect ($\mathcal{M}'$) and perfect ($\mathcal{M}$) mappings.

id	n	Source ontologies $\mathcal{O}_S$	$\lvert Sig(\mathcal{O}_S)\rvert$	$\mathcal{M}'_{\lvert C\rvert}$	$\mathcal{M}'_{\lvert P\rvert}$	$\mathcal{M}_{\lvert C\rvert}$	$\mathcal{M}_{\lvert P\rvert}$
d_1	2	cmt | conference	318 | 408	7	2	11	3
d_2	2	ekaw | sigkdd	341 | 193	8	1	11	0
d_3	3	cmt | conference | confOf	- | - | 335	14	5	22	11
d_4	4	conference | confOf edas | ekaw	- | - 903 | -	33	14	40	13
d_5	4	cmt | ekaw iasted | sigkdd	- | - 539 | -	29	8	33	5
d_6	7	cmt | conference | confOf edas | ekaw | iasted | sigkdd	- | - | - - | - | - | -	57	25	68	27
d_7	2	human | mouse	30364 | 11043	1175	2	1490	0
d_8	2	FMA_samll | NCI_small	16690 | 2472	2472	0	2480	0
d_9	7	AHSO | BNO | CABRO EPILONT | RAO RNPRIO | RO	341 | 281 | 175 741 | 1401 219 | 2274	6	0	7	0
d_{10}		ADAR | AHSO | AO BNO | CABRO | CSSO CWD | EPILONT | ICO IDO | NEOMARK3 | NPI OF | PCAO | PHARE PSO | RAO	17857 | - | 872 - | - | 3472 134 | - | 9438 4665 | 2513 | 401 5007 | 637 | 2169 1879 | 1401	218	48	147	0
d_{11}	19	d_9 and d_{10}'s ontologies	-	219	48	155	0
d_{12}		d_9 and d_{10}'s ontologies and AEO | AHOL | AMINO-ACID BFO | BHO | BMT | BOF BP | BSPO | BT | CKDO CMPO | CN | DIAB DIAGONT | EOL | FBbi GDCO | GFO | GRO | HIO INO | LHN | MCBCC MEDO | OBIWS | ONLIRA OPB | PEO | PPO | REPO RNPRIO | ROS | SHR UNITSONT | UO | VIVO	1920 | 2506 | 572 575 | 2619 | 3020 1993 | 714 | 1969 2322 | 591 | 21144 1842 | 5885 | 446 4617 | 5383 | 682 479 | 2736 | 5392 3769 | 1828 | 3280 355 | 2241 | 548 | 3996 551 | 5401 | 697 219 | 1726 | 825 340 | 3629 | 4862	967	172	676	0

8.2 Implementation

The proposed approach has been implemented in *ECLIPSE*[3] IDE using a verity of software and languages. *JAVA*[4] and *OWL-API* [HB11][5] have been mainly used in the core of the method's implementation. The GUI has been set with *JavaScript*, *HTML*, *Tomcat Server*[6], *CSS*, and *JS*. *Pellet* reasoner [SPG+07] (version 3) and *OWL-API explanation* [7] are used to handle the inconsistency of the merged ontologies. Moreover, the *SPARQL* language is used for the query endpoint. All the experiments were carried out on Intel core i7 with 12 GB internal memory on Windows 10 with Java compiler 1.8. The tool is publicly available on `http://comerger.uni-jena.de/` and distributed under an open-source license[8] along with the merged ontologies.

CoMerger allows the user to load the source ontologies in *OWL* format. The mapping between the source ontologies can be automatically determined by an embedded matcher, or it can be provided by the user in *RDF* format. The merged ontology created by *CoMerger* can be in *RDF/XML* or *OWL/XML* format.

8.3 Overview of Experimental Tests

To evaluate and validate the applicability of the proposed methods, we conducted a series of experiments on each aspect:

1. *Merging Multiple Ontologies with an N-ary Strategy*: In Chapter 10, we will analyze and evaluate our proposed n-ary method within three different tests: observing the characteristics of the merged ontologies under different scenarios, analyzing the constructed logic of the merged ontologies via a set of Competency Questions, and comparing the n-ary method with binary strategy.
2. *Utilizing User-Driven Generic Merge Requirements*: In Chapter 11, we will show two use case studies on the compatibility of user-selected GMRs and on the conflict resolution.
3. *Inconsistency Handling of the Merged Ontology*: In Chapter 12, we will show the characteristics of the inconsistencies of the merged ontology. We then will investigate the quality of the result by a Competency Questions test on two different versions of the consistent merged ontologies. Finally, we will show the time performance of the proposed method.
4. *Assessing the Quality of the Merged Ontology*: In Chapter 13, we aim to present a practical assessment of the proposed evaluation framework. To this end, we will measure the quality of the merged ontologies in the structural, functional, and usability dimensions. After that, we will present the time performance of

[3] ECLIPSE, `https://www.eclipse.org/`
[4] JAVA, `https://www.java.com/`
[5] `http://owlapi.sourceforge.net/`
[6] `http://tomcat.apache.org/`
[7] `http://owl.cs.manchester.ac.uk/research/explanation/`
[8] `https://github.com/fusion-jena/CoMerger`

the evaluation process and the overall result demonstration. We then illustrate a total analysis of tested datasets. Finally, we will present the extent to which the evaluation standard has been achieved.

9 *CoMerger*: Proposed Tool

This chapter presents the overview of *CoMerger* tool in Section 9.1. The architecture of the tool and its components are demonstrated in Section 9.2 and Section 9.3. The chapter is concluded by presenting the different GUI of the system and a summary in Section 9.4 and Section 9.5, respectively. The content of this chapter has been partially published in [BGKR20a].

9.1 *CoMerger* Overview

Over the decade, several ontology merging tools [CKP08; MTFH14; PC19; PK19; RR14; NM03] have been developed. However, none of them meets all three requirements: methods in [MTFH14; NM03; PC19; PK19; RR14] are restricted to merging two ontologies at a time and are thus not sufficiently scalable. A set of pre-defined merge requirements is implemented in [NM03; RR14], and thus they lack customization. Approaches in [CKP08; MTFH14; PC19; PK19; RR14] lack the ability for users to assess the quality of the merged results and do not provide inconsistency handling. Lastly, to the best of our knowledge, none of them are available as web-based applications.

We propose *CoMerger* as the first step towards a comprehensive merging tool focussing on four important aspects:

- compatibility checking of the user-selected Generic Merge Requirements (GMR)s,
- merging multiple ontologies with adjusting a set of user-selected GMRs,
- assessing the quality of the merged ontology, and
- inconsistency handling of the result.

Next, the architecture of *CoMerger* tool and the interaction between the mentioned aspects will be presented.

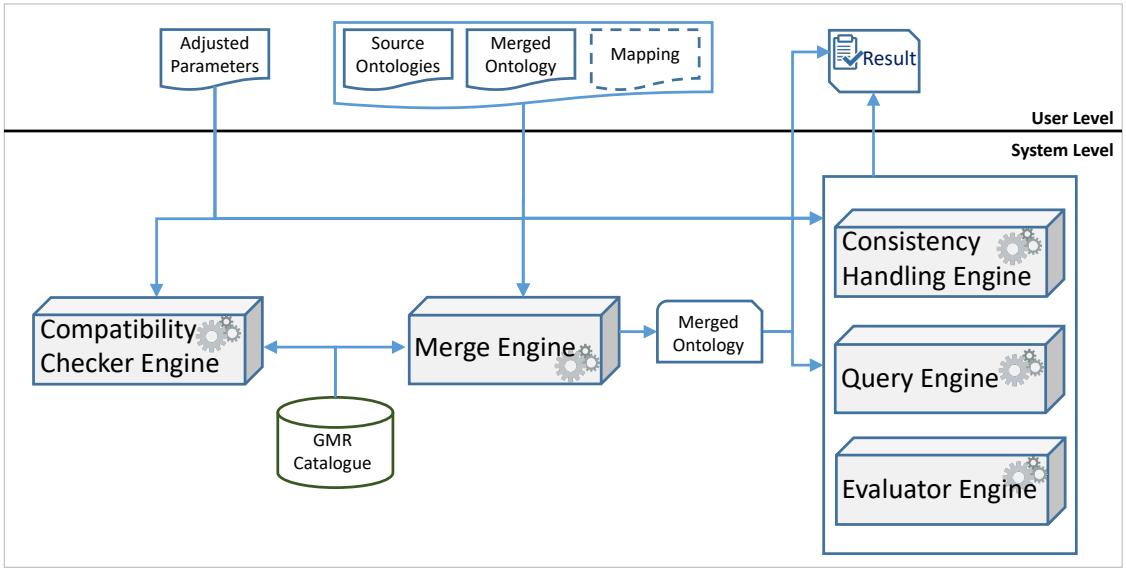


FIGURE 9.1: *CoMerger* architecture.

9.2 *CoMerger* Architecture

Figure 9.1 shows the architecture of *CoMerger* and the interaction between its components. It depicts the data flow between the *CoMerger* components in two levels. In the *system level*, the communication between the components is sketched. The *user level* allows a user to interact with the tool through a friendly GUI with different scenarios in merging ontologies, evaluating the merged ontology and check its consistency, ask for the compatibility of their selected GMRs, and perform a set of SPARQL queries on the merged and source ontologies. In all scenarios, users should provide the required inputs and adjust the parameters based on their needs. The respective results are presented to the users in each scenario.

The user uploads a set of source ontologies alongside with the mappings. The tool can read a set of RDF alignment, containing the similarity relations between entities with at least a given similarity value. If no mapping is given, *CoMerger* automatically detects the correspondences by using embedded ontology matching methods. Currently, two ontology matching approaches are embedded in our tool: SeeCOnt method [ABKD15] and a string matching based on the Jaccard similarity coefficient [Jac01].

Moreover, the user is able to select the required Generic Merge Requirements (GMR)s, including, e.g., entities preservation, one type restriction, acyclicity, and connectivity. The *Compatibility Checker* engine determines whether it is possible to meet all requirements simultaneously or there are contradictions. The engine suggests a compatible superset of the GMRs given by the user. After parsing the source ontologies and their mappings, the merged ontology is automatically generated by the *Merge* engine with taking into account the user-selected GMRs.

Moreover, the quality of the merged ontology can be evaluated via the *Evaluator* engine according to the user-selected evaluation aspects. There is a possibility to evaluate

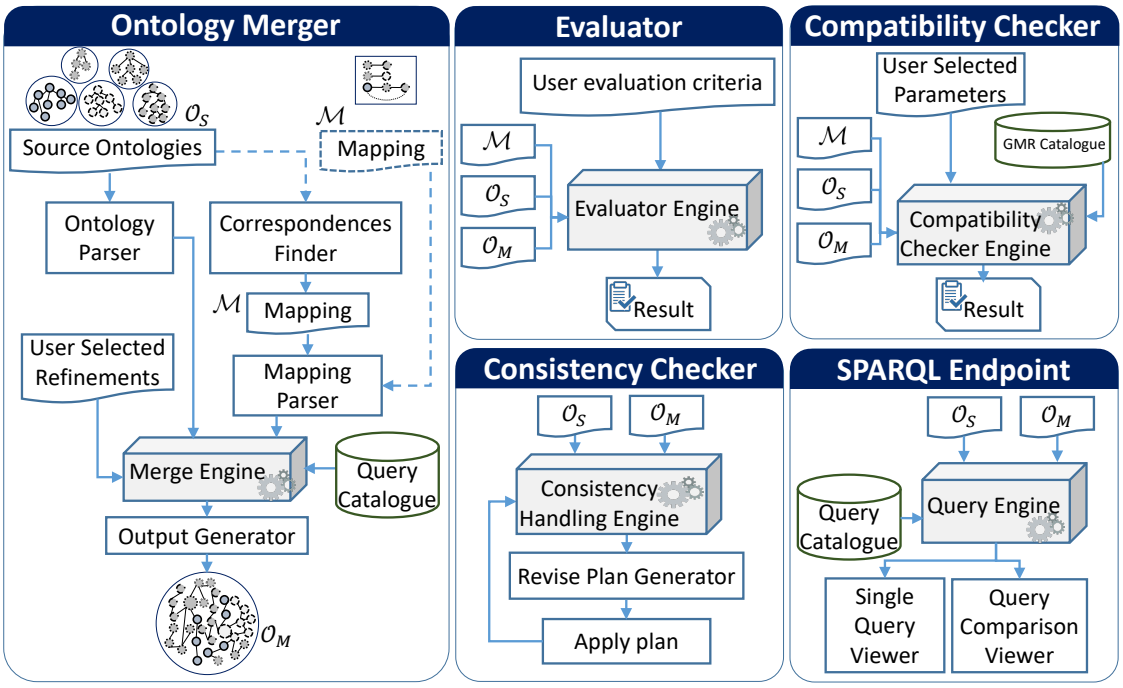


FIGURE 9.2: *CoMerger* components.

the quality of any given merged ontology independent of the merge process via a separate interface, *Evaluator*. Besides the quality criteria, the *Consistency Handling* engine can validate whether the merged result is consistent and provide support in repairing the inconsistencies. Additionally, through the embedded SPARQL endpoint of the *Query* engine, the user can compare query results on the merged and source ontologies simultaneously or individually.

9.3 *CoMerger* Component

Figure 9.2 presents the individual components of *CoMerger* tool, each is described as follows:

9.3.1 GMRs Compatibility Checker

The tool enables the flexible ontology merging process, in which the users can adjust a set of GMRs. However, not all GMRs are compatible. Thus, the compatibility checker component in *CoMerger* verifies which GMRs can be met simultaneously. We utilized a graph-based method, described in Chapter 5, to capture the maximum compatible superset of user-selected GMRs.

9.3.2 Multiple Ontologies Merger

Our proposed merge method, described in Chapter 4, takes as input a set of source ontologies $\mathcal{O}_S$ alongside the respective mappings $\mathcal{M}$ and automatically generates a

merged ontology $\mathcal{O}_M$. At first, the n ($n \geq 2$) source ontologies are divided into k ($k << n$) blocks and a local refinement is applied to them. After that, the blocks are combined to produce the merged ontology followed by a global refinement. The user can adjust a set of refinement operations via the embedded GMRs. Moreover, the tool logs the knowledge-level of the ontology merging process, which can be further analyzed by the users. Through our tool, in addition to reusing the final merged ontology, users can access the k created local sub-ontologies.

9.3.3 Merged Ontology Evaluator

To assess the quality and correctness of the merged ontology, we provided a comprehensive set of evaluation criteria, described in Chapter 7. It covers a variety of characteristics of each individual aspect of the merged ontology in three dimensions: (1) structural criteria via the evaluation of the General Merge Requirement (GMR)s, (2) functional measurements by the intended use and semantics of the merged ontology, and (3) usability evaluation on ontology and entity annotation. The evaluation criteria also represent an analytic view on how well the created merged ontology reflects the given source ontologies. The merged ontology can be evaluated independently of the merge method by the separated interface in *CoMerger*. Moreover, users can compare the results of queries on the merged ontology versus the source ontologies through the embedded SPARQL endpoint in the tool.

9.3.4 Consistency Checker

The merged ontology should be free of any inconsistencies. However, since the encoded knowledge of source ontologies may model different world views, it can easily happen that the merged ontology is inconsistent. It needs to be resolved if one wants to make use of the merged ontology in further applications. Thus, we developed a Subjective Logic-based method, described in Chapter 6, to rank the conflicting axioms, which caused inconsistencies in the merged ontology. The rank function concerns the degree of trustworthiness of the source ontologies knowledge. Upon that, the tool suggests the remedies of changes such as deleting or rewriting a part of conflicting axioms to turn the inconsistent merged ontology into a consistent one. The whole process can be accomplished automatically, or a user can review the system's suggestions and make necessary changes before applying them.

9.4 *CoMerger* GUI

This section describes the GUI ability of the mentioned components, as:

1. **GMRs Compatibility Checker**: Figure 9.3 shows the GUI of adjusting the desired GMRs. Users can set the number of suggested compatible sets and ask for the compatibility of the selected GMRs.
2. **Multiple Ontologies Merger**: Figure 9.4 shows the GUI of the merge component. User can upload the source ontologies and preferably their alignment, and adjust the desired GMRs along with the evaluation aspects. Once the ontologies are

merged, users can download the merged ontology, its sub-ontologies (the created blocks), the result of the evaluation, and the log file.

3. **Merged Ontology Evaluator**: The users can directly evaluate the merged ontology through the provided GUI in Figure 9.5. The merged ontology along with respective source ontologies should be uploaded, and the desired evaluation aspects need to be adjusted. Other GUIs of this component will be shown in Chapter 13.
4. **Consistency Checker**: Independent of the merge process or after that, users can ask for checking the consistency of the merged ontology by adjusting the required parameters (see *Consistency Test* setting in Figure 9.5). The result of consistency checking is shown in Figure 9.6 for a sample merged ontology. For an inconsistent merged ontology, a repair plan will be presented.

9.5 Summary

Despite the effort of many research studies, the developed ontology merging systems still suffer specific problems. In [CKP08; NM03], many user interactions are required, which might not be feasible for large-scale ontologies. iPrompt [NM03] requires user interaction for all entity merging, and in [CKP08], the enumerated schemas should be manually refined by users. To scale to many sources, the merging systems in [MTFH14; NM03; PC19; PK19; RR14] are insufficient due to merging only two ontologies at a time. No inconsistency handling is provided in [CKP08; MTFH14; PC19; PK19; RR14]. In [NM03; RR14], a set of fixed GMRs is implemented without user customization. To the best of our knowledge, besides iPrompt, the other mentioned systems are not publicly accessible and reproducible. Moreover, none of them are available as a web-based application.

In this chapter, we present *CoMerger*, a customizable online tool for building a consistent quality-assured merged ontology. The tool can merge multiple ontologies at the same time by adjusting user preference of merge requirements, check the quality of the merged ontology, and support for inconsistency handling of the result. Our web-based application is supported by many modern web browsers. The host server (VM) for the tool includes 8 cores with CPU 2.39 GHz and 16 GB RAM. The processing time based on the size and number of source ontologies is reasonable. For instance, merging 17 ontologies with 51461 axioms took 140 seconds with a home internet (44 Mbps speed) in the Firefox 72.0.2 web browser. Users can opt for a local installation of the tool to omit delays due to network communication, which the code is publicly available in our repository[1].

[1] `https://github.com/fusion-jena/CoMerger/tree/master/SourceCode`

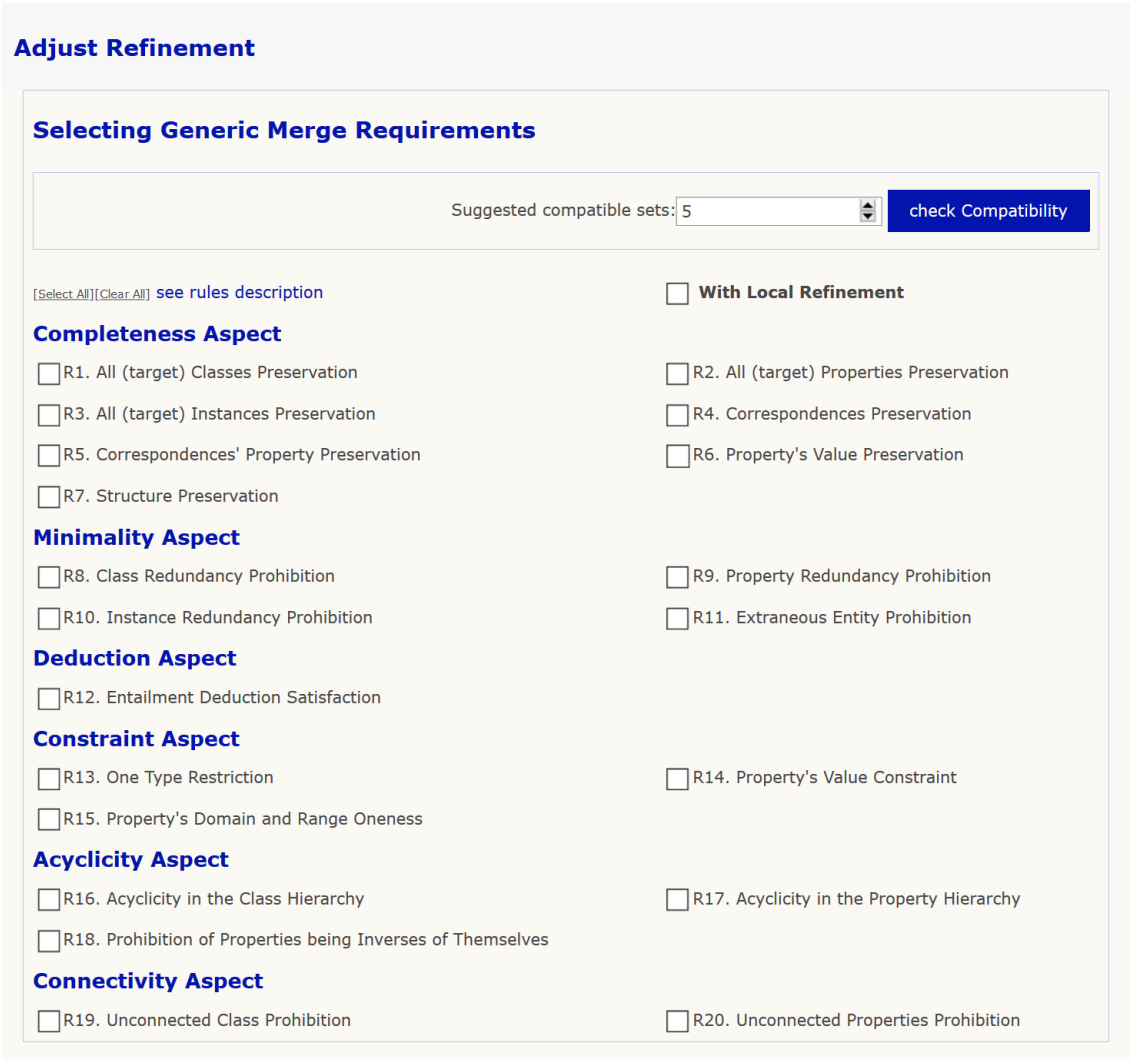


FIGURE 9.3: GUI of adjusting the desired GMRs.

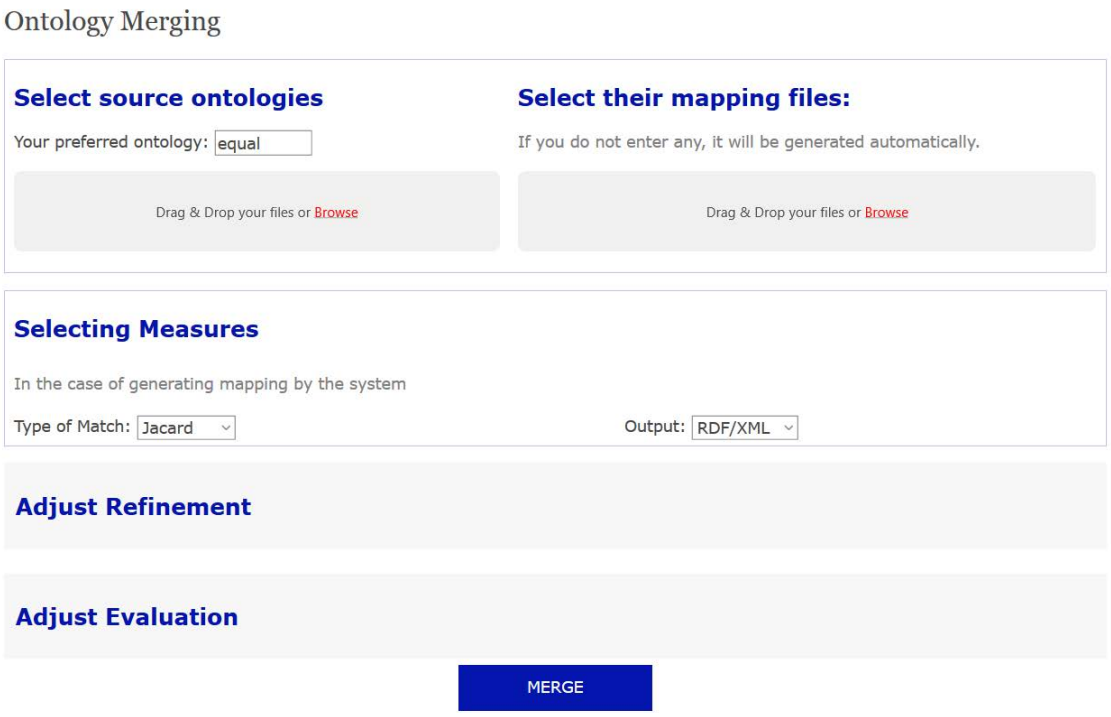


FIGURE 9.4: Ontology merging GUI.

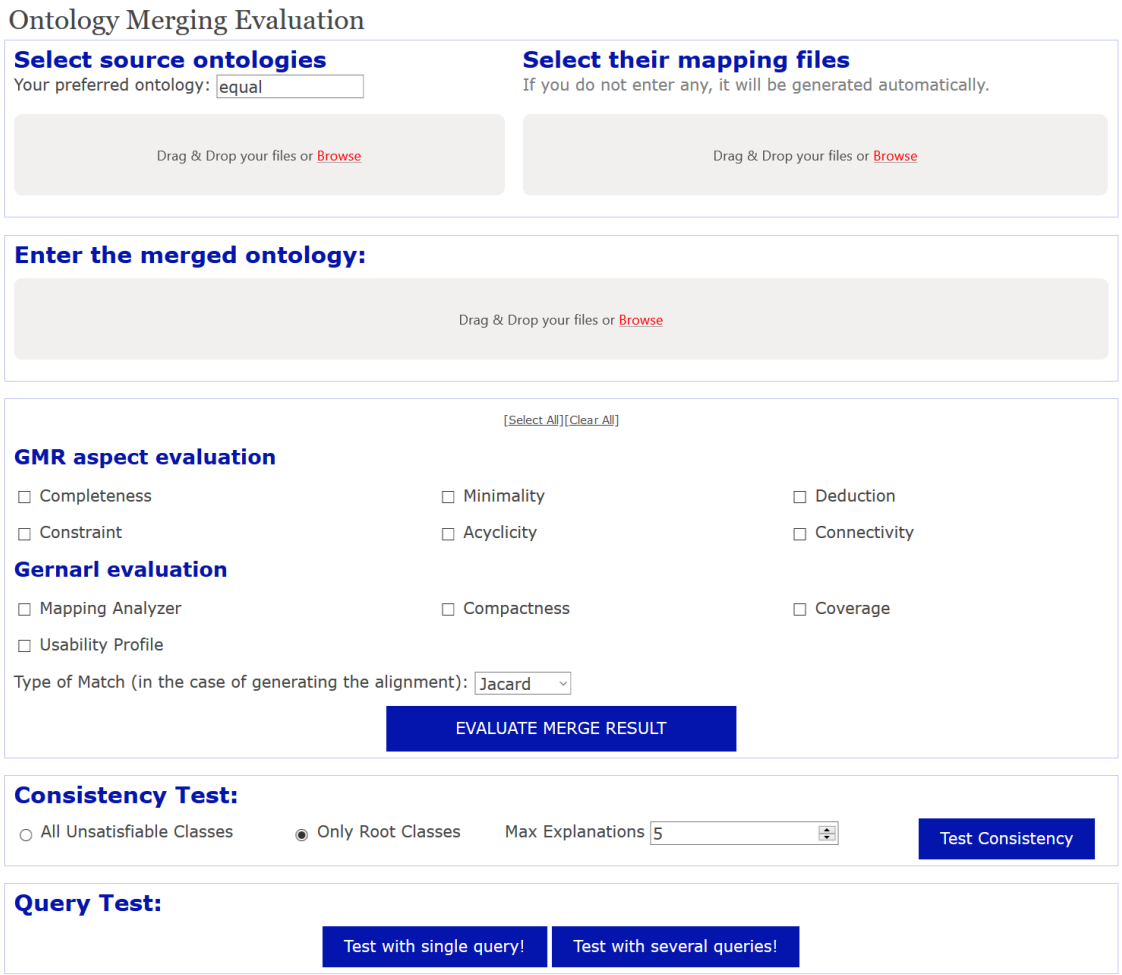


FIGURE 9.5: The GUI for customizable evaluation of the merged ontology.

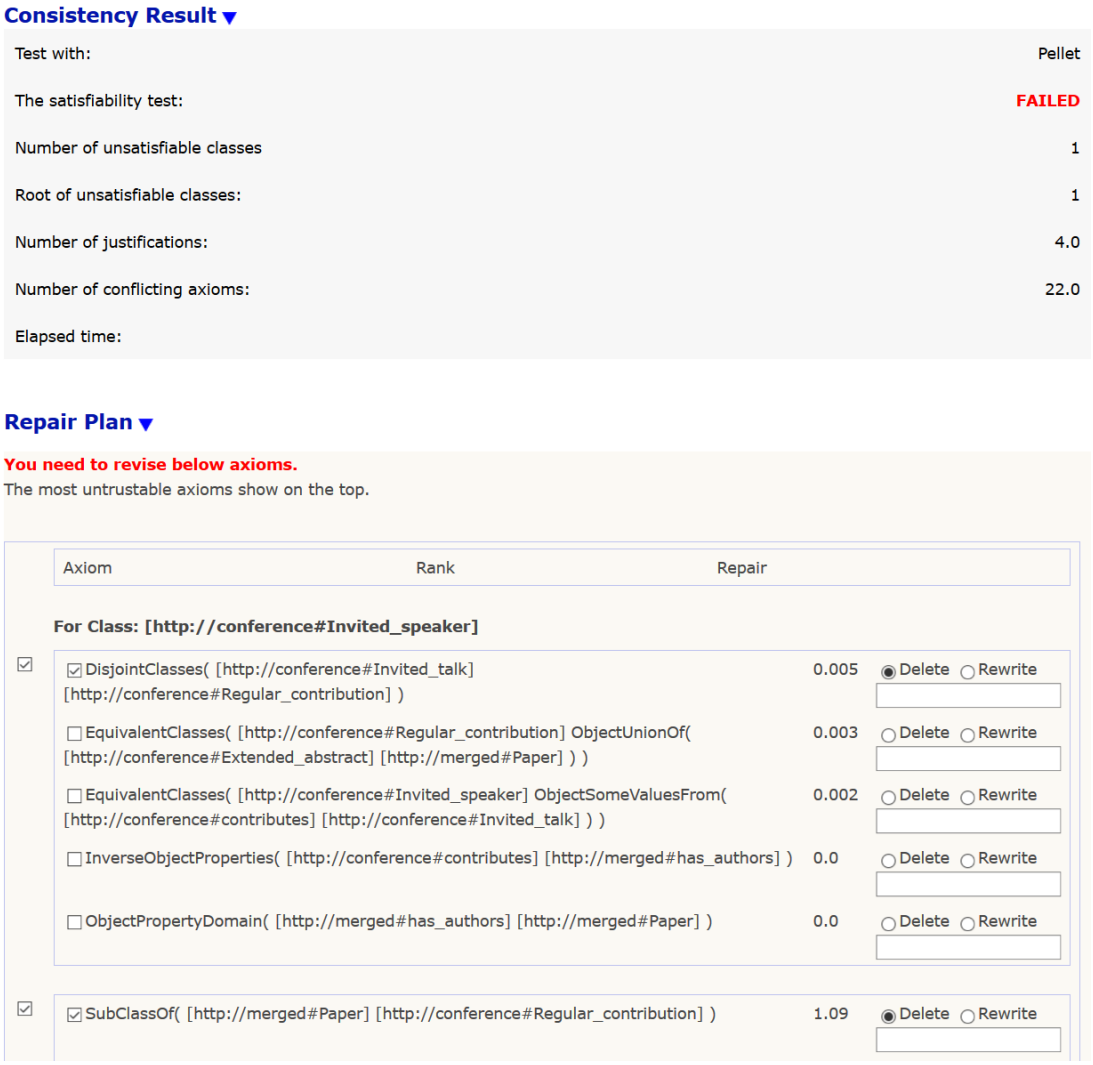


FIGURE 9.6: The result of the consistency test with a repair plan for an inconsistent merged ontology.

10 Experimental Tests on the N-ary Merge Method

In this chapter, we present the experimental test on the n-ary merge method, described in Chapter 4. At first, we show the setting of our experimental tests in Section 10.1. We then provide an experimental evaluation of the proposed approach for merging a variety of ontologies in Section 10.2, showing the effectiveness of our approach over purely binary approaches. In particular, we present three tests: In the first test, we observe the characteristics of the n-ary merged ontologies in Section 10.2.1. In the second test, we analyze the constructed logic of the merged ontology by answering a group of Competency Questions in Section 10.2.2. Comparing n-ary and binary merge methods is demonstrated in the third test in Section 10.2.3. In Section 10.3, a summary of this chapter is presented.

The list of used notations, symbols, and nomenclatures has been shown in Table 10.1. The results described in this chapter have been partially published in [BKR20a; BKR20b].

TABLE 10.1: The used notations, symbols, and nomenclatures in Chapter 10.

Notation	Description
$\mathcal{O}$	an ontology
$\mathcal{O}_S$	a set of source ontologies
$\mathcal{O}_M$	a merged ontology
$Sig(\mathcal{O})$	a signature of an ontology
C	a set of classes in an ontology
P	a set of properties of an ontology
I	an individual of an ontology
str	degree of the structure preservation
on	oneness- properties that have multiple domains or ranges
C_u	unconnected classes in an ontology
cyc	cycles in the class hierarchy
R_G	global refinements on the merged ontology
R_L	local refinements on the blocks
$\mathcal{M}$	a perfect mapping set between the source ontologies
$\mathcal{M}'$	an imperfect mapping set between the source ontologies
Cor	all corresponding entities
$Card$	the max cardinality of a corresponding set
n	number of source ontologies
k	number of blocks
$taxo\_rel$	taxonomy relation of a class
non_taxo_rel	non-taxonomy relation of a class
w_t	weight degree of taxonomy relation
w_{nt}	weight of non-taxonomy relation
ds	distributed axioms between two blocks
tr	translated axioms
ov	amount of overlap between source ontologies
$Mer.$	number of merge operations
N	n-ary merge strategy
B	balanced merge strategy
L	ladder merge strategy
V_1-V_{12}	twelve versions of the merged ontologies
d_1-d_{12}	twelve tested datasets
CQ	Competency Questions
GMR	General Merge Requirement
R1-R20	individual GMRs

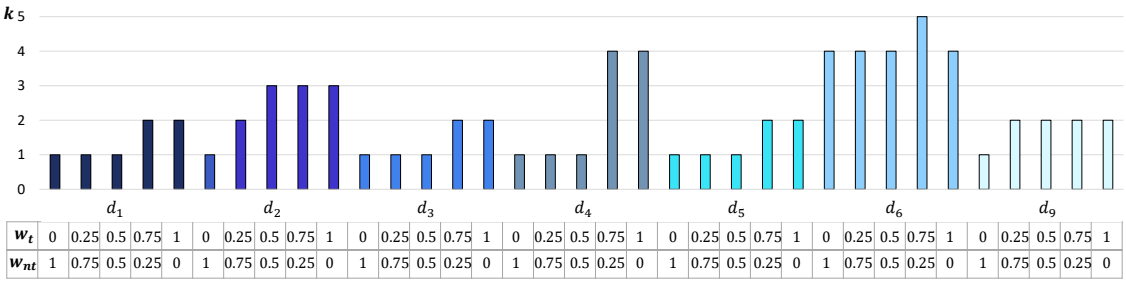


FIGURE 10.1: Effect of taxonomic weight w_t and non-taxonomic weight w_{nt} on the number of blocks k.

TABLE 10.2: Number of taxonomic and non-taxonomic relations for the datasets.

	d_1	d_2	d_3	d_4	d_5	d_6	d_7	d_8	d_9	d_{10}	d_{11}	d_{12}
$\|taxo\_rel\|$	91	134	155	299	414	628	9916	10629	1223	6295	6827	26432
$\|non\_taxo\_rel\|$	424	217	621	1118	521	1489	18976	28533	2848	41461	42755	110450

10.1 Test Setting

In this section, we present: (1) which values for the parameters have been set in Section 10.1.1, (2) how we build the binary merged ontologies in Section 10.1.2, (3) how we created different versions of the merged ontologies in Section 10.1.3, and (4) which refinements are used in the creation of merged ontologies in Section 10.1.4.

10.1.1 Adjusting Parameters

In the experimental results of this section, the values of w_t and w_{nt} were empirically determined to 0.75 and 0.5, respectively, but we make no claim that these are optimal values. The reason for choosing these values is the tested ontologies are augmented with a large number of nan-taxonomic relations (see Table 10.2). When the weight of nan-taxonomic relations is as important as the taxonomic relations, the classes with less number of taxonomic relations would be selected as the pivot classes. Note that, assigning classes to the blocks is carried based on the taxonomic relations. In this view, when a class with less number of taxonomic relations is considered as the pivot of the block, then only a few classes will be added to that block. As a result, blocks will be smaller, and the number of created blocks will be bigger and even more than the number of source ontologies. Thus, we set a higher weight to taxonomic relations.

We also carried a test for all values of w_t and w_{nt} in the range [0,1] with distance 0.25, where the $w_t + w_{nt} = 1$. The results of different weights on the number of blocks are shown in Figure 10.1. The datasets which are not shown, achieved $k = 1$ for all weight setting. The number of blocks depends on the characteristics of ontologies. Thus, in Table 10.2, we show the number of taxonomic and non-taxonomic relations of merged ontologies in each dataset. Given the characteristics that they have, the number of blocks is acceptable.

10.1.2 Adjusting Binary Methods

Binary strategies allow the merging of two ontologies at a time [BLN86]. They are called *ladder* strategies when a new ontology is integrated with an existing intermediate result at each step. A binary strategy is *balanced* when the ontologies are divided into pairs at the start and are integrated in a symmetric fashion.

Let us consider an example when $\mathcal{O}_1$, $\mathcal{O}_2$, $\mathcal{O}_3$, and $\mathcal{O}_4$ are source ontologies.

- In the *balanced-binary* merge, first, $\mathcal{O}_1$ and $\mathcal{O}_2$ are merged, which results in an intermediated merged ontology namely $\mathcal{O}_{12}$. Then $\mathcal{O}_3$ and $\mathcal{O}_4$ are merged, which creates $\mathcal{O}_{34}$. After that, the created $\mathcal{O}_{12}$ and $\mathcal{O}_{34}$ are merged to produce the final result.
- In the *ladder-binary* merge, first, $\mathcal{O}_1$ and $\mathcal{O}_2$ are merged. Then, $\mathcal{O}_{12}$ is merged with $\mathcal{O}_3$, which results in creating $\mathcal{O}_{123}$. After that, $\mathcal{O}_{123}$ is merged with $\mathcal{O}_4$ to create the final result.

In our experimental tests, we follow the mentioned producers to conduct the binary merges.

10.1.3 Building Different Versions of Merge

We evaluated our approach under different conditions V_1-V_{12} (see Table 10.3):

1. Using the perfect mapping ($\mathcal{M}$) versus an imperfect mapping ($\mathcal{M}'$)
2. Applying (✓) the local refinement process or not (✗)
3. Applying (✓) global refinements or not (✗)

We follow these conditions for n-ary, balanced, and ladder merge strategies. Thus, we generated six versions (V_1-V_6) of merged ontology using the n-ary method, three versions (V_7-V_9) of binary balanced, and three versions (V_{10}-V_{12}) of the binary ladder. For the binary merges, we consider the imperfect mapping, only. Because, at each merge process, the mapping for the created intermediate merged result and one of the source ontologies is generated on the fly with the ontology matching tool. There is no perfect mapping for the intermediate merge ontology with one of the ontologies in $\mathcal{O}_S$.

10.1.4 Adjusting Refinements

To apply local and global refinements, we select a subset of GMRs (*R1-R3*, *R7*, *R15*, *R16*, *R19*) from Chapter 5. A brief reminder: *R1*, *R2*, *R3*, and *R7* are related to class, property, instance, and structure preservation, respectively. *R15* emphasizes on the oneness of properties, i.e., without multiple domains or ranges. *R16* is relevant to class acyclicity and *R19* expresses the degree of connectivity in $\mathcal{O}_M$. We use these criteria to observe how well the merged ontologies are structured.

TABLE 10.3: The settings for generating twelve variants of the merged ontologies.

	N-ary						Balanced			Ladder		
	V_1	V_2	V_3	V_4	V_5	V_6	V_7	V_8	V_9	V_{10}	V_{11}	V_{12}
Mapping type	$\mathcal{M}$	$\mathcal{M}$	$\mathcal{M}$	$\mathcal{M}'$	$\mathcal{M}'$	$\mathcal{M}'$	$\mathcal{M}'$	$\mathcal{M}'$	$\mathcal{M}'$	$\mathcal{M}'$	$\mathcal{M}'$	$\mathcal{M}'$
Global refinement	✓	✓	✗	✓	✓	✗	✓	✓	✗	✓	✓	✗
Local refinement	✓	✗	✗	✓	✗	✗	✓	✗	✗	✓	✗	✗

10.2 Experimental Results

In the first test, we observe the characteristics of the n-ary merged ontologies. In the second test, we analyze the constructed logic of the merged ontology by answering a group of Competency Questions (CQs). Comparing binary merge and n-ary methods is demonstrated in the third test. The inconsistency test will be carried in Chapter 12.

From the existing n-ary merge approaches, Porsche [SBH08] does not target ontologies, but XML schemas, and the code of OmerSec [MFBB10] is not available. Thus, we were not able to perform a comparison with existing works. We mainly focus on tests associated with the characteristics and performance.

10.2.1 Characteristics of the N-Ary Merged Result

In this section, first, we present which criteria we used for the evaluation of the results. We then present the result and analyze them.

10.2.1.1 Evaluation criteria

To evaluate the characteristics of the created $\mathcal{O}_M$, we use three evaluation criteria categories:

- **Integrity**: in [DB10], the integrity aspect of a created merged ontology is defined with three measures:
  - The *compactness* metric represents the size of the merge result. It can be useful for general information about the O_M in the desired application.
  - The *coverage*, or namely *completeness*, is the percentage of entities presented in the $\mathcal{O}_S$ that is included in the $\mathcal{O}_M$. It is related to the degree of information preservation. This includes classes C, properties P, instances I, and the structurality str coverage. The latest one refers to preserving the structure of the merged ontology w.r.t. source ontologies. These metrics are related to the evaluation of *R1-R3*, and *R7* from Chapter 5.
  - The *redundancy*, or namely *minimality*, checks that no redundant entities appear in the $\mathcal{O}_M$. Since in all created versions of the merged ontologies, no redundant entities have been found, we do not include this metric in results.

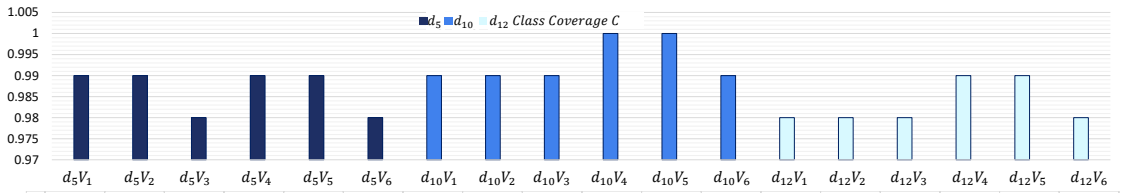
    There is a difference between duplicated or redundant entities. Duplicated are those entities (with the same name and characteristics), which are repeated

more than one time in the ontologies. For example, class `Paper` appears two times in the ontology. In the implementation of *CoMerger*, classes in OWL ontologies are defined as SET, which by nature, contains no repeated elements. Thus, in all tested ontologies, there are no duplicated entities. On the other hand, redundant means that two entities (possibly with different names) are the same real entity in the world. For example, classes `Document` and `ConferenceDocument` are referred to one real entity in the world. So, they are the same entities with different names. If an ontology contains these two classes, it has redundancy. One way to detect the entities referring to the same real entity in the world is the mappings given by experts or generated by an ontology alignment system. Given the alignment between source ontologies, the mapped entities should not appear separately in the merged ontologies. If so, we call them redundant entities. Since for a group of corresponding entities, our approach merges them into an integrated entity, there are no redundant entities in all tested ontologies, too.

- **Evaluation of applying the GMRs**: we evaluate to what extent the refinements play a role. For *R15*, we count the number of properties that have multiple domains or ranges and present it as $|on|$. For *R19*, we consider only those unconnected classes in the $\mathcal{O}_M$, which were connected in the $\mathcal{O}_S$ given by $|C_u|$. For *R16*, we calculate how many cycles in the class hierarchy in the $\mathcal{O}_M$ exist ($|cyc|$).
- **Merge process characteristic**: we address the characteristic of the merge process by measuring:
  - Number of created blocks (k)
  - Percentage of axioms with taxonomic distributed on the total axioms ($ds_{tax}\%$)
  - Percentage of axioms with non-taxonomic distributed relations on the total axioms ($ds_{non\_tax}\%$)
  - Percentage of the axioms with taxonomic relations which are unconnected on the total axioms ($ds_{un}\%$)
  - Percentage of the translated axioms on the total axioms ($tr\%$)
  - Number of local refinement actions in intra-combinations ($|R_L|$)
  - Number of global refinement actions in inter-combinations ($|R_G|$)

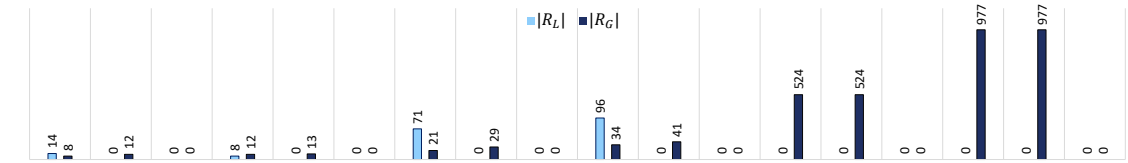
10.2.1.2 Results

The full results of running six versions of our n-ary method on twelve datasets have been presented in Tables 10.6 and 10.7 on the OAEI and BioPortal datasets, respectively. To highlight analyzing the various aspects, we extracted some of these tests in Figure 10.2 and Figure 10.3, but in analyzing the result, we make a conclusion on the whole result. Indeed, these figures are a summary for better visualization. Moreover, we show the results of considering perfect or imperfect mapping in Figure.10.4, the analyzing of the blocks' number on the whole datasets in Figure 10.5, and characteristic of the distributed axioms in Figure 10.6. In particular, we present the achieved results, as follows:



	d_5V_1	d_5V_2	d_5V_3	d_5V_4	d_5V_5	d_5V_6	$d_{10}V_1$	$d_{10}V_2$	$d_{10}V_3$	$d_{10}V_4$	$d_{10}V_5$	$d_{10}V_6$	$d_{12}V_1$	$d_{12}V_2$	$d_{12}V_3$	$d_{12}V_4$	$d_{12}V_5$	$d_{12}V_6$
P	1	1	1	1.01	1	1	0.9	0.9	0.9	1.01	1.01	1	0.95	0.95	0.95	1.03	1.03	1.01
I	1	1	1	1	1	1	0.93	0.93	0.93	0.98	0.98	0.98	0.96	0.96	0.96	0.96	0.96	0.96
st	0	0	0	0	0	1	0	0	71	0	0	87	0	0	176	0	0	579

FIGURE 10.2: Class C, property P, and instance I coverage with the number of unpreserved structure str coverage of six versions of n-ary merge for three sample datasets.



	d_1V_1	d_1V_2	d_1V_3	d_1V_4	d_1V_5	d_1V_6	d_6V_1	d_6V_2	d_6V_3	d_6V_4	d_6V_5	d_6V_6	$d_{12}V_1$	$d_{12}V_2$	$d_{12}V_3$	$d_{12}V_4$	$d_{12}V_5$	$d_{12}V_6$
$\|on\|$	0	0	4	0	0	4	0	0	14	0	0	24	0	0	44	0	0	85
$\|C_u\|$	0	0	6	0	0	8	0	0	8	0	0	11	0	0	252	0	0	229
$\|cyc\|$	0	0	1	0	0	0	0	0	4	0	0	0	0	0	4	0	0	13

FIGURE 10.3: Number of local $|R_L|$ and global $|R_G|$ refinements, oneness $|on|$, unconnected classes $|C_u|$ and cycle $|cyc|$ of six versions of the n-ary merge.

- In Figure 10.2, we show the degree of information preservation on the six versions of our n-ary merge method. The percentage of class coverage $C\%$ is shown on the chart, while the percentage of the property $P\%$, instance $I\%$ coverage, and the absolute number of unpreserved structure $|str|$ are drawn in the table view under each chart.
- In Figure 10.3, we show the number of local $|R_L|$ and global $|R_G|$ refinement actions. In this figure, we also present the statistics about the evaluation of selected GMRs with $|on|$, $|C_u|$, and $|cyc|$.
- In Figure 10.4, we show the percentage of translated axioms for all datasets along with the number of corresponding entities in perfect $\mathcal{M}$ or imperfect $\mathcal{M}'$ mapping.
- In Figure 10.5, we demonstrate the number of created blocks k for all datasets using perfect $\mathcal{M}$ (V_1-V_3) or imperfect $\mathcal{M}'$ mappings (V_4-V_6).
- In Figure 10.6, we present the taxonomic distributed axioms. Moreover, we present the statistic of the axioms, which are unconnected on the is-a hierarchy level. We also show distributed non_taxonomic relations. All values are shown as the percentage of the total axioms

10.2.1.3 Analyzing the results

To analyze the result of these two figures, we examine the result of different versions.

- Investigating the effect of considering no refinements (V_3/V_6) compared to applying refinements either locally or globally (V_1, V_2, V_4, V_5): We can conclude

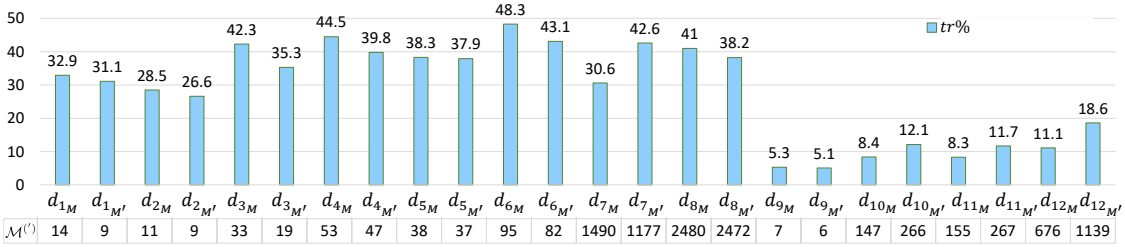


FIGURE 10.4: Comparing translated axioms $tr\%$ with corresponding entities in $\mathcal{M}$ and $\mathcal{M}'$.

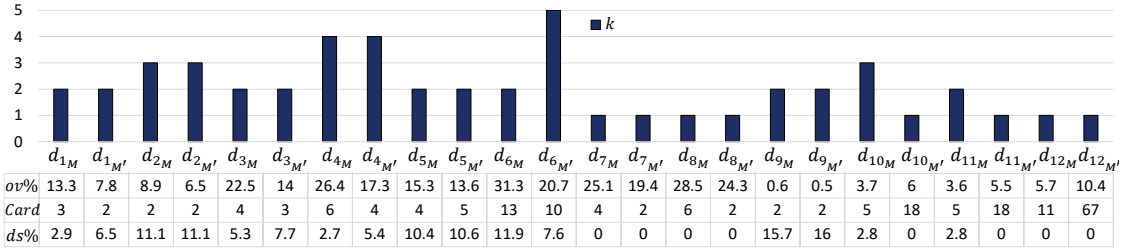


FIGURE 10.5: Number of blocks k versus class overlap $ov\%$, max cardinality $Card$, and distributed axioms $ds\%$.

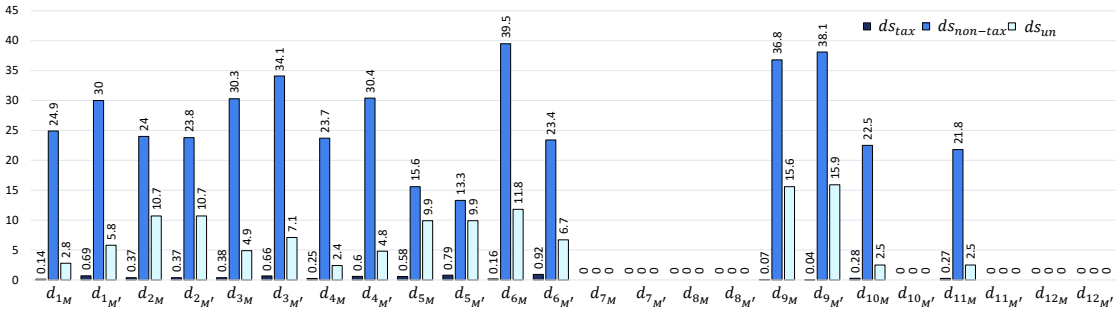


FIGURE 10.6: Comparing distributed axioms: taxonomic distributed axioms ds_{tax}, non-taxonomic distributed axioms ds_{non_tax}, and unconnected taxonomic distributed axioms ds_{un}.

that applying refinement in 76 out of 120 cases (see Tables 10.6 and 10.7) leads to considerable improvements in class coverage, structure preservation, oneness, unconnected classes, and acyclicity, whereas in 40 cases, the same results are achieved. For example, if no refinement is applied, 8 classes became unconnected in d_1, or 24 properties with multiple domains and ranges in d_6 are generated, or 13 cycles exist in d_{12}, or 71 unpreserved structures happen in d_{10}.

- Analyzing using perfect (V_1-V_3) versus imperfect mappings (V_4-V_6): As the results in Tables 10.6 and 10.7 show, we observe that out of 12 datasets, a perfect mapping causes 7 fewer cases of unconnected entities (e.g., in d_1 and d_6); 3 fewer cases of cycles (e.g., in d_{12}); 7 fewer cases of properties oneness (e.g., in d_{12}); 6 cases preserving better structure (e.g., in d_5, d_{10} and d_{12}); 3 fewer cases of local refinements (e.g., in d_6); 9 fewer cases of global refinements (e.g., in d_1, d_6, and d_{12}).

On the other hand, using imperfect mapping causes 1 fewer case of unconnected entities in d_{12}; 6 fewer cases of cycles (e.g., in d_1 and d_6); 4 fewer cases of local refinements (e.g., in d_1); and 2 fewer cases of global refinements.

- Analyzing applying local refinements (V_1/V_4) versus no local refinements (V_2/V_5): In regards to the results of Tables 10.6 and 10.7, the effect of applying local refinements V_1/V_4 rather than no local refinements V_2/V_5 cannot be observed in the view of the mentioned criteria. However, applying refinement actions in the local or global level have different computational complexity, since the respective search spaces substantially differ. For instance, finding or repairing a cycle in a small set of classes (local sub-ontologies) is far less expensive than among all classes of the merged ontology.
- Analyzing translated axioms with respect to the corresponding entities: As the result in Figure 10.4 shows, using perfect or imperfect mappings with different numbers of corresponding entities has a direct effect on the number of translating $|tr|$ axioms.
- Analyzing the number of blocks k: As the results in Figure 10.5 shows, the value of k affects the number of distributed axioms. Thus, we also report the percentage of the distributed is-a axioms on the total axioms $ds\%$. Determining k in our method mostly depends on the amount of overlap $ov\%$ between the source ontologies' classes and the cardinality $Card$ value on corresponding classes. The overlap is calculated by the ratio of the number of corresponding classes on the total classes. For each dataset, we show the maximum cardinality between the corresponding entities. Considering $ov\%$ and $Card$, the values of k in our datasets is reasonable ($1 \leq k \leq 5$), which shows the feasibility of our approach.
- Analyzing distributed axioms: As the results in Figure 10.6 shows, in all datasets, the percentage of the taxonomic distributed axioms ds_{tax} is less than 1%. These axioms are mostly related to the *objectUnionOf* axioms, where the union classes are distributed over the blocks. For instance, $A\ subClassOf\ (C\ or\ D)$; which $C \in \mathcal{L}_1$, and $D \in \mathcal{L}_2$. The distributed axioms, which are a type of non-taxonomic relations (ds_{non_tax}) are mostly high. The reason is the ontologies have many non_taxonomic relations, such as *equivalent* or *disjoint* axioms, which their entities based on the taxonomic relations are distributed between the blocks. We observed that, in each dataset, there are some taxonomic-based axioms which do not belong to any blocks (showing by ds_{un}). For instance, $A\ subClassOf\ B$, but none of the A or B do not already assign to any blocks. Thus, this axiom is marked as ds_{un}. Because the superclass of these types of axiom (here A) does not connect to any other classes, and indeed, it is only connected to the root. The datasets $d_7, d_8, d_{10_{M'}}, d_{11_{M'}}$, and d_{12} have one block and do not have any distributed axioms.

10.2.2 Answering Competency Questions

Competency Questions (CQs) are a list of questions used in the ontology development life cycle, which an ontology should answer. By using CQs tests, we aim to observe which created $\mathcal{O}_M$ can provide superior answers to the CQs. To this end, we used a set

TABLE 10.4: Answering CQs on the different versions of merged ontologies.

id	*Complete%*	*Semi% -Complete*	*Partial%*	*Wrong%*	*Unknown%*	*Null%*	*Total Correct%*
d_1V_1, V_2	**20**	6.7	**3.3**	**3.3**	66.7	**0**	**30**
d_1V_3		**10**	0		**63.3**	3.4	
d_1V_4, V_5					66.7	**0**	
d_1V_6					**63.3**	3.4	
d_2V_1-V_6	**13.3**	**6.7**	**13.3**	**0**	**66.7**	**0**	**33.3**
d_3V_1, V_2	36.7	26.7	**6.7**	**0**	**26.7**	**3.2**	**70**
d_3V_3	33.3	**30**	3.3			6.7	66.7
d_3V_4, V_5	**40**	23.3		3.3		3.4	
d_3V_6	36.7					6.7	63.3
d_3V_7, V_8	**40**					3.4	66.7
d_3V_9	36.7					6.7	63.3
d_3V_{10}, V_{11}	26.7	13.3	0	10		23.3	40
d_3V_{12}	23.3					26.7	36.7
d_4V_1-V_3	33.3	**20**	10	**0**	36.7	**0**	**63.3**
d_4V_4-V_6	**36.7**	13.3	**13.3**	3.3	**33.4**		
d_4V_7-V_9	33.3					3.4	60
d_4V_{10}-V_{12}	23.3	10	3.3	10		20	36.7
d_5V_1-V_3	**23.3**	13.4	**23.3**	**0**	40	**0**	60
d_5V_4-V_9		**16.7**			**36.7**		**63.3**
d_5V_{10}-V_{12}		13.3	16.7			10	53.3
d_6V_1-V_3	**40**	**23.3**	**10**	**0**	26.7	**0**	**73.3**
d_6V_4-V_9			6.7		**23.3**	6.7	70
d_6V_{10}-V_{12}	33.3	6.7	0	6.7		30	40

of CQs (see Appendix A) in the conference domain and ran them for the datasets in the conference domain.

10.2.2.1 Evaluation criteria

We compare the CQ-results for each dataset with all possible answers from the $\mathcal{O}_M$ with respect to its $\mathcal{O}_S$ on that dataset. We achieved five different answers, as explained below:

- The ***complete*** answer of $\mathcal{O}_M$ indicates a full correct answer similar to the $\mathcal{O}_S$'s answer.
- Among all answers of the $\mathcal{O}_S$, if the number of found answers in $\mathcal{O}_M$ is higher or equal than the number of not found, we marked it as a ***semi-complete*** answer.
- Among all answers of the $\mathcal{O}_S$, if the number of found answers in $\mathcal{O}_M$ is lower than the number of not found, we marked it as a ***partial*** answer.
- An answer is marked as ***wrong*** if the result of the CQ from the merged ontology is not the same as the answers from the source ontologies. For example, the wrong answer might happen, when all source ontologies return, for instance, true for a true or false query, but the merged ontology returns false. Alternatively, in the questions related to the is-a hierarchy, the merged ontology shows a different hierarchy than the hierarchy of the source ontologies.

- If CQ's entities exist in the ontology, but no further knowledge exists about them, we mark them by a ***null*** answer.
- If the ontology does not have any knowledge about the CQ, we indicate this by an ***unknown*** answer.

10.2.2.2 Results

The results of running our CQs on the conference datasets are presented in Table 10.4. Each CQ has been converted manually to a SPARQL query and run against the $\mathcal{O}_S$ and the different versions of the $\mathcal{O}_M$. We show the values based on the percentage of the total number of CQs. Values in boldface show the best result in each dataset: highest values in *complete*, *semi-complete*, *partial*, and *total-complete* answers; lowest values in *wrong*, *unknown*, and *null* answers. The last column shows a sum value on the *complete*, *semi-complete*, and *partial* answers given by the *total correct*.

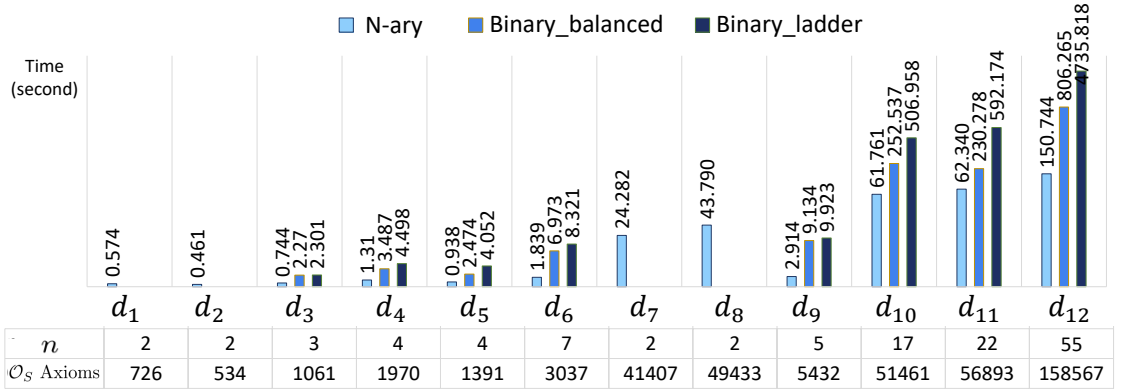
10.2.2.3 Analyzing

From this test, we can observe the following points.

- Analyzing applying local or global refinements: in some cases can provide more *complete* answers (cf. d_3V_1, V_2, $d_3V_4V_5$), more *partial* answers (cf. d_1V_1, V_2,d_3V_1, V_2), and less *null* answers (cf. in d_1V_1, V_2, d_1V_4, V_5, d_3V_1, V_2).
- Analyzing using perfect mappings: it causes more *semi-complete* answers in d_3 and d_4, more *partial* answers in d_1 and d_3, less wrong answers in d_3 and d_4, and less *null* answers in d_3 and d_6.
- Analyzing using imperfect mappings: it causes more *complete* answers in d_3 and d_4, more *semi-complete* answers in d_5, more *partial* answers in d_4, less unknown answers in d_4-d_6, less *null* answers in d_4, and less *unknown* answers in d_6. One of the reasons why the perfect mappings in some cases are worse than imperfect mapping is, the entities inside the ontology in OAEI mapping correspond to each other. This self-mapping causes the structure of the $\mathcal{O}_M$ to be mixed up and makes it unable to provide correct CQs' answering.
- Comparing the n-ary (V_4-V_6) and binary (V_7-V_{12}) strategies: it reveals that the n-ary merge can achieve the same quality result as binary methods, and even better results in d_4 in terms of achieving more *complete* answers and less *null* answers rather than binary approaches.
- Comparing binary-balanced (V_7-V_9) and binary-ladder (V_{10}-V_{12}): The difference between two versions of the binary merges is mostly related to the order of selecting and merging them, which affects the final merged output. This test also reveals that binary-balanced (V_7-V_9) and binary-ladder (V_{10}-V_{12}) in all cases have different outputs and results, i.e., the final merged output is different. Consequently, the answers to the CQs are different.

TABLE 10.5: Comparing n-ary (N), balanced (B), and ladder (L) merge strategies with the number of corresponding entities $|Cor|$, translated axioms $|tr|$, global refinements $|R_G|$, merge processes $|Mer.|$.

	d_4			d_6			d_{10}			d_{11}			d_{12}		
	N	B	L	N	B	L	N	B	L	N	B	L	N	B	L
$\|Cor\|$	47	65	64	82	127	128	266	369	351	267	388	386	1139	2186	2159
$\|tr\|$	790	1270	1339	1310	2791	3462	6949	15816	31420	6960	16060	35480	30035	66344	154002
$\|R_G\|$	21	27	23	41	48	51	143	166	191	143	177	196	977	1533	1560
$\|Mer.\|$	1	3		1	6		1	16		1	18		1	54	



	d_1	d_2	d_3	d_4	d_5	d_6	d_7	d_8	d_9	d_{10}	d_{11}	d_{12}
n	2	2	3	4	4	7	2	2	5	17	22	55
$\mathcal{O}_S$ Axioms	726	534	1061	1970	1391	3037	41407	49433	5432	51461	56893	158567

FIGURE 10.7: Runtime performance: numbers of source ontologies and axioms versus the required time for the merge process in second. Binary merges run on datasets with $n > 2$.

This test also reveals how datasets are robust. For instance, different versions of d_2 are mostly identical and are not different. On the other hand, some datasets such as d_3 have quiet different output on each created version of the merged ontology.

Summing up, the ontologies in the conference domain are small in size and lack complete knowledge modeling. Because of this, 100% complete answers are hard to achieve. However, our created merged ontologies can provide up to 73.3% totally correct answers for the given CQs in this domain. This shows the applicability of our method.

10.2.3 Binary versus N-ary

While the CQ test shows that the quality of the result of the n-ary approach can compete with the binary approaches, in the third test, we compare performance metrics. We conduct an experimental test by a series of binary merges on the eight datasets that have more than two source ontologies. Note that, in the previous test, we compared the quality of the output. However, in this test, we examine:

1. The required operations
2. The runtime performance

10.2.3.1 Evaluation criteria

We evaluate the criteria for:

- The operation complexity: We compare the number of required refinement actions $|R_G|$ in Table 10.5. We also present the number of merge processes by $|Mer.|$ in the table.
- The runtime performance: We demonstrate the method's *scalability* by illustrating the performance test results. Here, the runtime performance is evaluated based on the number of ontologies versus the required time for the merge process in N-ary(V_5), Balanced (V_8), and Ladder (V_{11}).

10.2.3.2 Results

Table 10.8, Table 10.9, and Table 10.10 compare the n-ary, balanced, and ladder merge strategies for different versions of the test setting. Table 10.8 shows the results when both local and global refinements are applied. Table 10.9 shows the results when only global refinements are applied. Table 10.10 shows the results when no refinement is applied. In all tables, we show the results for those datasets which have more than two source ontologies. Table 10.5 shows a summary of Table 10.9 to emphasize on analyzing the result.

Figure 10.7 shows the total runtime in seconds for the merge processes in n-ary (V_5), binary-balanced (V_8), and binary-ladder (V_{11}). We ran each test ten times and presented the average values. The processing time of the binary merge does not include the time for creating the respective mappings.

10.2.3.3 Analyzing the results

From the test, we derive the following points:

- Analyzing corresponding entities $|Cor|$ and translated axioms: The number of total corresponding entities $|Cor|$ during the whole process of the merge is quite different. In each test of a binary merge, only the correspondences between two entities can be integrated into a new entity. However, in the n-ary merge, the corresponding entities from multiple source ontologies can be integrated simultaneously into the new entity. For this reason, the number of corresponding entities in the binary merge in 7 out of 8 datasets is much higher than the n-ary approach. Consequently, the required amount of combining them into new entities and translating their axioms $|tr|$ is high in all tested datasets. For instance, in d_6, the number of translated axioms in the n-ary method is 1310, while in the binary-ladder strategy, it is 3462. Therefore, the n-ary approach has great speed-up.
- Analyzing global refinements: In 7 out of 8 datasets, the n-ary approach requires fewer refinement actions compared to binary merges. For instance, in d_{12}, the n-ary method runs 977 actions, while in the binary-balanced is 1533.
- Analyzing local refinements: The same conclusion can be derived from comparing the required local refinements.
- Analyzing the number of merges: We also present the number of merge processes by $|Mer.|$ in the table. While the n-ary approach only uses one iteration for all tests,

ladder and balanced methods require $n-1$ merge process, e.g., in d_{12}, 54 times the whole process of the merge should be run.

- Analyzing the scalability: In Figure 10.7, we present the number of source ontologies $\mathcal{O}_S$ and their axioms, to emphasize that there is a linear dependency on the number and size of $\mathcal{O}_S$ w.r.t. merge processing time. The result quantifies that the n-ary merge is on average 4 (9) times faster than the balanced (ladder) binary merge, respectively. This concludes that using n-ary rather than binary methods is more valuable and effective when the number of ontologies gets higher. For example, in d_3 with 3 source ontologies, n-ary is 3 times faster than binary strategies, in d_6 with $n = 7$, n-ary is 4 times faster than both binary approaches, and in d_{12} with $n = 56$, it is 31 times faster than the binary ladder. Binary approaches limit when merging multiple ontologies, while the n-ary method has significantly better performance. This emphasizes the effectiveness of our approach in merging multiple ontologies.

Overall our results show that the n-ary strategy achieves comparable results in terms of quality but outperforms binary approaches in terms of runtime and complexity.

10.3 Summary

In this chapter, we have conducted a set of experimental tests to analyze the effectiveness of our proposed method described in Chapter 4. We used a set of datasets and compared our approach with a series of binary merge strategies (ladder and balanced). For each dataset, we created twelve different merged ontologies by using the perfect mapping versus an imperfect mapping and applying the refinement process on only the local level, only the global level, and on both levels. We applied a subset of GMRs on merged ontologies and reported the absolute number of occurrences that a GMR did not fulfill in merged ontologies. Moreover, we evaluated them by integrity criteria (compactness, coverage, and redundancy [DB10]).

By analyzing the characteristics of merged ontologies, we observed that:

- Applying refinements in most cases leads to considerable improvements in the class coverage, structure preservation, oneness, unconnected classes, and acyclicity.
- In most cases, the perfect mappings cause better results in terms of structural measures. However, in some datasets, the result achieved by using imperfect mappings is impressive.
- The number of corresponding entities in the perfect or imperfect mappings leads to a different number of translated axioms.
- The number of blocks affects the number of distributed axioms. It gets effect by the amount of overlap between the source ontologies classes and the cardinality of the corresponding classes.

By analyzing merged ontologies with a set of Competency Questions (CQs), we observed that:

- Ontologies built by utilizing a set of refinements could provide better answers to the CQs.
- In some datasets, ontologies built by perfect mappings provide better answers, while in some others, the ontologies built by the imperfect mapping have superior answers.
- Ontologies built via the n-ary merge method achieve the same quality result as those by the binary methods, and even better results in some cases.

By comparing binary merges and the n-ary strategy in terms of operational complexity and runtime performance, we observed that:

- The number of translating axioms in binary merges is much more than in the n-ary technique. Because in the n-ary merge, not only a pair but a group of corresponding entities can be integrated into one step.
- Merged ontologies achieved by the n-ary method requires fewer refinement actions compare to the binary merge methods.
- The n-ary approach only uses one iteration for all tests. In ladder and balanced methods, $n - 1$ merge process is required (n is the number of source ontologies).
- In terms of time performance on tested datasets, the n-ary merge is on average 4 and 9 times faster than the balanced and ladder binary merge, respectively.

TABLE 10.6: Characteristics of the n-ary merged result- OAEI dataset.

id	Integrity							Model Pr.			Merge Pr.	
	Compactness			Coverage				$\|on\|$	$\|C_u\|$	$\|cyc\|$	$\|R_L\|$	$\|R_G\|$
	C	P	I	C	P	I	str					
d_1V_1	77	120	0	1	1	-	0	0	0	0	14	8
d_1V_2	77	120	0	1	1	-	0	0	0	0	-	12
d_1V_3	76	120	0	0.97	1	-	0	4	6	1	-	0
d_1V_4	82	121	0	1	1	-	0	0	0	0	8	12
d_1V_5	82	121	0	1	1	-	0	0	0	0	-	13
d_1V_6	81	121	0	0.98	1	-	0	4	8	0	-	0
d_2V_1	112	61	0	1	1	-	0	0	0	0	17	1
d_2V_2	112	61	0	1	1	-	0	0	0	0	-	1
d_2V_3	112	61	0	1	1	-	0	0	1	0	-	0
d_2V_4	115	60	0	1	1	-	0	0	0	0	18	3
d_2V_5	115	60	0	1	1	-	0	0	0	0	-	4
d_2V_6	115	60	0	1	1	-	0	2	2	0	-	0
d_3V_1	98	147	0	0.98	1	-	0	0	0	0	21	11
d_3V_2	98	147	0	0.98	1	-	0	0	0	0	-	16
d_3V_3	97	147	0	0.97	1	-	0	6	7	1	-	0
d_3V_4	109	154	0	0.98	1	-	0	0	0	0	14	15
d_3V_5	109	154	0	0.98	1	-	0	0	0	0	-	17
d_3V_6	108	154	0	0.97	1	-	0	6	9	0	-	0
d_4V_1	202	167	114	0.99	1	1	0	0	0	0	30	12
d_4V_2	201	167	114	0.99	1	1	0	0	0	0	-	18
d_4V_3	199	167	114	0.98	1	1	0	6	8	1	-	0
d_4V_4	226	165	114	0.99	0.99	1	0	0	0	0	26	18
d_4V_5	226	165	114	0.99	0.99	1	0	0	0	0	-	21
d_4V_6	224	165	114	0.98	0.99	1	0	9	9	0	-	0
d_5V_1	248	156	4	0.99	1	1	0	0	0	0	61	7
d_5V_2	247	156	4	0.99	1	1	0	0	0	0	-	9
d_5V_3	246	156	4	0.98	1	1	0	3	5	0	-	0
d_5V_4	253	155	4	0.99	1.01	1	0	0	0	0	64	13
d_5V_5	253	154	4	0.99	1	1	0	0	0	0	-	17
d_5V_6	251	153	4	0.98	1	1	1	9	5	0	-	0
d_6V_1	338	274	118	0.99	1.05	1	0	0	0	0	71	21
d_6V_2	336	274	118	0.98	1.05	1	0	0	0	0	-	29
d_6V_3	334	274	118	0.98	1.05	1	0	14	8	4	-	0
d_6V_4	390	283	118	0.99	1.03	1	0	0	0	0	96	34
d_6V_5	390	281	118	0.99	1.02	1	0	0	0	0	-	41
d_6V_6	386	280	118	0.98	1.01	1	1	24	11	0	-	0
d_7V_1	4526	4	0	1	1	-	0	0	0	0	-	9
d_7V_2	4526	4	0	1	1	-	0	0	0	0	-	9
d_7V_3	4526	4	0	1	1	-	0	0	7	2	-	0
d_7V_4	4873	2	0	1	1	-	0	0	0	0	-	7
d_7V_5	4873	2	0	1	1	-	0	0	0	0	-	7
d_7V_6	4873	2	0	1	1	-	0	0	7	0	-	0
d_8V_1	7290	87	0	1	1	-	0	0	0	0	-	1949
d_8V_2	7290	87	0	1	1	-	0	0	0	0	-	1949
d_8V_3	7285	87	0	1	1	-	4	0	1916	29	-	0
d_8V_4	7721	87	0	1	1	-	0	0	0	0	-	1925
d_8V_5	7721	87	0	1	1	-	0	0	0	0	-	1925
d_8V_6	7712	87	0	1	1	-	9	0	1916	0	-	0

TABLE 10.7: Characteristics of the n-ary merged result- BioPortal dataset.

id	Integrity							Model Pr.			Merge Pr.	
	Compactness			Coverage				$\|on\|$	$\|C_u\|$	$\|cyc\|$	$\|R_L\|$	$\|R_G\|$
	C	P	I	C	P	I	str					
d_9V_1	1145	101	31	1	0.72	1	0	0	0	0	8	17
d_9V_2	1145	101	31	1	0.72	1	0	0	0	0	-	17
d_9V_3	1144	101	31	1	0.72	1	0	14	2	0	-	0
d_9V_4	1146	101	31	1	0.72	1	0	0	0	0	0	17
d_9V_5	1146	101	31	1	0.72	1	0	0	0	0	-	17
d_9V_6	1145	101	31	1	0.72	1	0	14	2	0	-	0
$d_{10}V_1$	5042	2197	843	0.99	0.9	0.93	0	0	0	0	41	106
$d_{10}V_2$	5042	2197	843	0.99	0.9	0.93	0	0	0	0	-	116
$d_{10}V_3$	5018	2197	843	0.99	0.9	0.93	71	21	5	2	-	0
$d_{10}V_4$	4957	2394	882	1	1.01	0.98	0	0	0	0	-	143
$d_{10}V_5$	4957	2394	882	1	1.01	0.98	0	0	0	0	-	143
$d_{10}V_6$	4928	2378	882	0.99	1	0.98	87	22	7	3	-	0
$d_{11}V_1$	5564	2245	870	0.99	0.89	0.94	0	0	0	0	40	110
$d_{11}V_2$	5564	2245	870	0.99	0.89	0.94	0	0	0	0	-	120
$d_{11}V_3$	5539	2245	870	0.99	0.89	0.94	75	21	5	2	-	0
$d_{11}V_4$	5490	2469	909	1	1.01	0.98	0	0	0	0	-	143
$d_{11}V_5$	5490	2469	909	1	1.01	0.98	0	0	0	0	-	143
$d_{11}V_6$	5461	2453	909	0.99	1	0.98	87	22	7	3	-	0
$d_{12}V_1$	15822	3818	1262	0.98	0.95	0.96	0	0	0	0	-	524
$d_{12}V_2$	15822	3818	1262	0.98	0.95	0.96	0	0	0	0	-	524
$d_{12}V_3$	15729	3818	1262	0.98	0.95	0.96	176	44	252	4	-	0
$d_{12}V_4$	15080	3589	1262	0.99	1.03	0.96	0	0	0	0	-	977
$d_{12}V_5$	15080	3589	1262	0.99	1.03	0.96	0	0	0	0	-	977
$d_{12}V_6$	14944	3540	1262	0.98	1.01	0.96	579	85	229	13	-	0

TABLE 10.8: Comparing n-ary (V_4), balanced (V_7), and ladder(V_{10}) merge strategies.

id	Method	$\|C\|$	$\|P\|$	$\|I\|$	$\|Cor\|$	$\|tr\|$	$\|R_L\|$	$\|R_G\|$	$\|Mer.\|$
d_3	N-ary	109	154	0	19	374	14	15	1
	Balanced	110	154	0	22	566	37	16	2
	Ladder	110	154	0	22	566	38	16	2
d_4	N-ary	226	165	114	47	790	26	18	1
	Balanced	226	166	114	65	1270	26	27	3
	Ladder	226	167	114	64	1339	122	21	3
d_5	N-ary	253	155	4	37	527	64	13	1
	Balanced	254	156	4	47	863	162	16	3
	Ladder	255	156	4	47	997	214	15	3
d_6	N-ary	390	283	118	82	1310	96	34	1
	Balanced	395	285	118	127	2785	398	37	6
	Ladder	396	285	118	128	3406	521	42	6
d_9	N-ary	1146	101	31	6	278	0	17	1
	Balanced	1146	101	31	6	360	2	17	6
	Ladder	1146	139	31	6	483	0	17	6
d_{10}	N-ary	4957	2394	882	266	6949	-	143	1
	Balanced	4984	2194	843	368	15790	272	147	16
	Ladder	5003	2195	843	351	31415	949	173	16
d_{11}	N-ary	5490	2469	909	267	6960	-	143	1
	Balanced	5142	2248	870	730	19024	102	155	18
	Ladder	5510	2266	870	387	35489	951	178	18
d_{12}	N-ary	15080	3589	1262	1139	30035	-	977	1
	Balanced	15285	3227	1175	4136	82041	2095	1517	54
	Ladder	15384	3398	1175	2156	153871	3831	1541	54

TABLE 10.9: Comparing n-ary(V_5), balanced (V_8), and ladder (V_{11}) merge strategies.

id	Method	$\|C\|$	$\|P\|$	$\|I\|$	$\|Cor\|$	$\|tr\|$	$\|R_G\|$	$\|Mer.\|$
d_3	N-ary	109	154	0	19	374	17	1
	Balanced	110	154	0	22	566	18	2
	Ladder	110	154	0	22	566	18	2
d_4	N-ary	226	165	114	47	790	21	1
	Balanced	226	166	114	65	1270	27	3
	Ladder	226	167	114	64	1339	23	3
d_5	N-ary	253	154	4	37	527	17	1
	Balanced	254	154	4	47	865	19	3
	Ladder	254	154	4	47	994	19	3
d_6	N-ary	390	281	118	82	1310	41	1
	Balanced	394	284	118	127	2791	48	6
	Ladder	394	283	118	128	3462	51	6
d_9	N-ary	1146	101	31	6	278	17	1
	Balanced	1146	101	31	6	360	17	6
	Ladder	1146	139	31	6	483	17	6
d_{10}	N-ary	4957	2394	882	266	6949	143	1
	Balanced	4982	2193	843	369	15816	166	16
	Ladder	5003	2193	843	351	31420	191	16
d_{11}	N-ary	5490	2469	909	267	6960	143	1
	Balanced	5507	2246	870	388	16060	177	18
	Ladder	5511	2265	870	386	35480	196	18
d_{12}	N-ary	15080	3589	1262	1139	30035	977	1
	Balanced	15450	3282	1175	2186	66344	1533	54
	Ladder	15384	3395	1175	2159	154002	1560	54

TABLE 10.10: Comparing n-ary(V_6), balanced (V_9), and ladder (V_{12}) merge strategies.

id	Method	$\|C\|$	$\|P\|$	$\|I\|$	$\|Cor\|$	$\|tr\|$	$\|Mer.\|$
d_3	N-ary	108	154	0	19	374	1
	Balanced	109	154	0	22	567	2
	Ladder	109	154	0	22	567	2
d_4	N-ary	224	165	114	47	790	1
	Balanced	224	166	114	65	1268	3
	Ladder	224	167	114	64	1339	3
d_5	N-ary	251	153	4	37	527	1
	Balanced	252	153	4	47	868	3
	Ladder	252	153	4	47	1001	3
d_6	N-ary	386	280	118	82	1310	1
	Balanced	389	283	118	127	2786	6
	Ladder	390	282	118	128	3440	6
d_9	N-ary	1145	101	31	6	278	1
	Balanced	1145	101	31	6	372	6
	Ladder	1146	139	31	6	513	6
d_{10}	N-ary	4928	2378	882	266	6949	1
	Balanced	4951	2177	843	365	15866	16
	Ladder	4970	2176	843	350	31733	16
d_{11}	N-ary	5461	2453	909	267	6960	1
	Balanced	5475	2231	870	386	15801	18
	Ladder	5474	2246	870	385	35742	18
d_{12}	N-ary	14944	3540	1262	1139	30035	1
	Balanced	14868	3225	1171	3617	80151	54
	Ladder	15111	3346	1175	2135	156293	54

11 Experimental Tests on GMRs

This chapter presents the empirical analysis of the Generic Merge Requirements introduced in Chapter 5. We have implemented the GMRs within the *CoMerger*. The detail of the implementation is discussed in Appendix B. We conduct two use case studies on (i) compatibility of user-selected GMRs in Section 11.1, (ii) conflict resolution in Section 11.2. For the ranking of suggested sets, w_1, w_2, and w_3 have been empirically adjusted to 0.8, 0.1, and 0.1, respectively, where w_1 is the weight on the user-selected GMRs, w_2 is the weight of user-selected aspects, and w_3 is the weight on the compatibility degree of the GMRs. We present a summary of this chapter in Section 11.3.

The list of used notations, symbols, and nomenclatures in this chapter has been shown in Table 11.1. The results described in this chapter have been partially published in [BGKR20c; BKR19b; GBKR20].

TABLE 11.1: The used notations, symbols, and nomenclature in Chapter 11.

Notation	Description
$\mathcal{O}_S$	a set of source ontologies
$\mathcal{O}_M$	a merged ontology
c_i	a class in an ontology
p_i	a property of an ontology
I_i	an individual of an ontology
$\sqsubseteq$	subClassOf relation between two classes
GMR	Generic Merge Requirements
R1-R20	individual GMRs
$\mathcal{U}$	a set of user-selected GMRs
$\mathcal{K}$	number of vertices in the clique
$\mathcal{K}^C$*-Clique*	compatible clique with $\mathcal{K}$ vertices
$\mathcal{K}^C$*-max-Clique*	compatible clique with maximum $\mathcal{K}$ vertices
$\mathcal{RS}$	a set of compatible sets with $\mathcal{U}$
rs	a compatible set with $\mathcal{U}$
CQ	Competency Question

11.1 Use Case Study on Compatibility Checker

In this section, our goal is to show given a set of user-selected GMRs, there is a superset of compatible GMRs that can be fulfilled simultaneously. For this purpose, we use two given merged ontologies $\mathcal{O}_{M_1}$ and $\mathcal{O}_{M_2}$ from Chapter 5. We analyze three user-selected GMRs and then discuss the extent to which they can be fulfilled simultaneously.

11.1.1 First Use Case

Let us consider that the user selects four GMRs as $\mathcal{U} = \{R2, R3, R8, R16\}$. In $\mathcal{O}_{M_2}$, *R3* and *R8* are fulfilled. However, properties p_{15}, p_{24}, and p_{25} are missing, so, *R2* did not fulfill. Moreover, there is a cycle in $c_5c_{13} \sqsubseteq c_{16} \sqsubseteq c_{17} \sqsubseteq c_{18} \sqsubseteq c_6c_{19} \sqsubseteq c_5c_{13}$, which indicates that *R16* also did not fulfill in $\mathcal{O}_{M_2}$. *R2* and *R16* are incompatible, because *R2* adds the missing properties and wants to keep all properties. *R16*, on the other hand, deletes is-a properties to be free of cycles. Applying *R2* in $\mathcal{O}_{M_2}$ adds properties p_{15}, p_{24}, and p_{25}. Therefore, all properties can be preserved in the merged ontology. However, by applying *R16*, property p_{28} will be deleted to be free of cycles. This makes the *R2* fails. In this case, if *R2* wants to add p_{28}, a cycle will be generated. So, *R2* can not completely be fulfilled in the merged ontology. It can add three missing properties, but one property (p_{28}) cannot be preserved.

Therefore, our system suggests as the best possible compatible set $rs_1 = \{$*R1*, *R3*, *R8*, *R9*, *R10*, *R11*, *R13*, *R14*, *R16*, *R17*, *R18*$\}$ and $rs_2 = \{$*R3*, *R8*, *R9*, *R10*, *R11*, *R13*, *R14*, *R15*, *R16*, *R17*, *R18*$\}$, in which *R2* is not considered. Based on our proposed method, these two sets have the same scores of 1.0. Thus, given the user-selected GMRs, there is a superset of compatible GMRs that can be fulfilled simultaneously. The next possible compatible set is a case where *R16* is not considered, but *R2* is kept. So, the system suggests $rs_3 = \{$*R2*, *R3*, *R5*, *R7*, *R8*, *R10*, *R12*, *R13*, *R14*, *R19*, *R20*$\}$ with a score of 0.986.

For the given $\mathcal{U}$, the 3^C-Cliques are $\{R3, R8, R16\}, \{R2, R8, R16\}$, and $\{R2, R3, R8\}$, and a 2^C-Clique is $\{R3, R8\}$. Table 11.2 shows all $\mathcal{K}^C$-max-Cliques, which are all possible maximal compatible sets for the user-selected GMRs. rs_1-rs_6, rs_8, and rs_9 are 11^C-max-Cliques, while rs_7, rs_{10}-rs_{16} are 10^C-max-Cliques, and rs_{17} and rs_{18} are 8^C-max-Clique and 7^C-max-Clique, respectively.

11.1.2 Second Use Case

Let us consider that the user selects three GMRs as $\mathcal{U} = \{R3, R6, R13\}$. In $\mathcal{O}_{M_1}$, *R3* is fulfilled. *R13* applies one type restriction. So, in $\mathcal{O}_{M_1}$, only one of the types for property $p_6p_{16} : has\_id$ should be preserved. However, applying *R13* prevents *R6* from being met, because all values of property p_6p_{16} are preserved. Therefore, *R6* and *R13* have a conflict with each other and cannot be fulfilled simultaneously in $\mathcal{O}_{M_1}$. Given the user-selected GMRs, our method suggests the best set as $\{$*R2*, *R3*, *R5*, *R7*, *R8*, *R10*, *R12*, *R13*, *R14*, *R19*, *R20*$\}$ in which *R6* does not include.

For the given $\mathcal{U}$, there are two 2^C-Cliques as $\{R3, R13\}$, and $\{R3, R6\}$. Table 11.3 shows all possible maximum compatible sets ($\mathcal{K}^C$-max-Clique) for $\mathcal{U}$.

TABLE 11.2: All possible compatible maximum sets for user-selected GMRs $\mathcal{U} = \{R2, R3, R8, R16\}$. Green (no-line): user-selected compatible GMRs; Red (double-underline): user-selected incompatible GMRs; Orange (underline): extra compatible GMRs.

$\mathcal{RS}$	$\mathcal{K}$	Compatible	Incompatible	Score
rs_1	11	{R1, R3, R8, R9, R10, R11, R13, R14, R16, R17, R18}	{R2}	1.0
rs_2	11	{R3, R8, R9, R10, R11, R13, R14, R15, R16, R17, R18}	{R2}	1.0
rs_3	11	{R2, R3, R5, R7, R8, R10, R12, R13, R14, R19, R20}	{R16}	0.986
rs_4	11	{R2, R3, R5, R7, R8, R9, R10, R11, R12, R13, R14}	{R16}	0.975
rs_5	11	{R1, R2, R3, R5, R7, R8, R10, R13, R14, R19, R20}	{R16}	0.973
rs_6	11	{R2, R3, R5, R7, R8, R10, R13, R14, R15, R19, R20}	{R16}	0.973
rs_7	10	{R3, R6, R8, R9, R10, R11, R15, R16, R17, R18}	{R2}	0.97
rs_8	11	{R1, R2, R3, R5, R7, R8, R9, R10, R11, R13, R14}	{R16}	0.963
rs_9	11	{R2, R3, R5, R7, R8, R9, R10, R11, R13, R14, R15}	{R16}	0.963
rs_{10}	10	{R1, R3, R6, R8, R9, R10, R11, R16, R17, R18}	{R2}	0.957
rs_{11}	10	{R2, R3, R5, R6, R7, R8, R10, R15, R19, R20}	{R16}	0.944
rs_{12}	10	{R2, R3, R5, R6, R7, R8, R10, R12, R19, R20}	{R16}	0.943
rs_{13}	10	{R2, R3, R5, R6, R7, R8, R9, R10, R11, R15}	{R16}	0.934
rs_{14}	10	{R2, R3, R5, R6, R7, R8, R9, R10, R11, R12}	{R16}	0.933
rs_{15}	10	{R1, R2, R3, R5, R6, R7, R8, R10, R19, R20}	{R16}	0.931
rs_{16}	10	{R1, R2, R3, R5, R6, R7, R8, R9, R10, R11}	{R16}	0.921
rs_{17}	8	{R3, R4, R8, R9, R10, R11, R13, R14}	{R2, R16}	0.719
rs_{18}	7	{R3, R4, R6, R8, R9, R10, R11}	{R2, R16}	0.676

11.1.3 Third Use Case

Let us consider that the user selects six GMRs as $\mathcal{U} = \{R1, R2, R3, R8, R10, R19\}$.

In $\mathcal{O}_{M_1}$, classes c_3, c_{13}-c_{15} are missing. Applying *R1* adds these classes in the $\mathcal{O}_{M_1}$. Moreover, properties p_{11}, p_{13}, p_{18}-p_{22}, p_{24}, and p_{25} are missing. So, applying *R2* adds these properties in the $\mathcal{O}_{M_1}$. Moreover, *R3* adds the missing instance I_2 in the $\mathcal{O}_{M_1}$. *R8*, *R10*, and *R19* are fulfilled in $\mathcal{O}_{M_1}$.

In $\mathcal{O}_{M_2}$ class c_{15} and properties p_{15}, p_{24}, and p_{25} are missing. Applying *R1* and *R2* adds the missing classes and properties in $\mathcal{O}_{M_2}$. *R3*, *R8*, and *R10* are fulfilled in $\mathcal{O}_{M_2}$. However, in the origin $\mathcal{O}_{M_2}$, the class c_{14} was unconnected. But, *R1* and *R2* actions cause that now c_{14} be connected. So, *R19* is fulfilled.

A 6^C-Clique is $\{R1, R2, R3, R8, R10, R19\}$, 5^C-Cliques are $\{R2, R3, R8, R10, R19\}$ and $\{R1, R2, R3, R8, R10\}$, 4^C-Cliques are $\{R2, R3, R8, R10, R19\}$ and $\{R2, R3, R8, R10, R19\}$, and a 3^C-Clique is $\{R3, R8, R10\}$. So, the user-selected GMRs in this case study are compatible with each other, however, our system suggests all possible supersets of other compatible GMRs with $\mathcal{U}$, as shown in Table 11.4.

TABLE 11.3: All possible compatible maximum sets for user-selected GMRs $\mathcal{U} = \{R3, R6, R13\}$. Green (no-line): user-selected compatible GMRs; Red (double-underline): user-selected incompatible GMRs; Orange (underline): extra compatible GMRs.

$\mathcal{RS}$	$\mathcal{K}$	Compatible	Incompatible	Score
rs_1	11	{R2, R3, R5, R7, R8, R10, R12, R13, R14, R19, R20}	{R6}	1.0
rs_2	11	{R2, R3, R5, R7, R8, R9, R10, R11, R12, R13, R14}	{R6}	0.989
rs_3	11	{R1, R2, R3, R5, R7, R8, R10, R13, R14, R19, R20}	{R6}	0.987
rs_4	11	{R2, R3, R5, R7, R8, R10, R13, R14, R15, R19, R20}	{R6}	0.987
rs_5	11	{R1, R3, R8, R9, R10, R11, R13, R14, R16, R17, R18}	{R6}	0.987
rs_6	11	{R3, R8, R9, R10, R11, R13, R14, R15, R16, R17, R18}	{R6}	0.986
rs_7	11	{R1, R2, R3, R5, R7, R8, R9, R10, R11, R13, R14}	{R6}	0.976
rs_8	11	{R2, R3, R5, R7, R8, R9, R10, R11, R13, R14, R15}	{R6}	0.976
rs_9	10	{R2, R3, R5, R6, R7, R8, R10, R15, R19, R20}	{R13}	0.956
rs_{10}	10	{R3, R6, R8, R9, R10, R11, R15, R16, R17, R18}	{R13}	0.956
rs_{11}	10	{R2, R3, R5, R6, R7, R8, R9, R10, R11, R15}	{R13}	0.946
rs_{12}	10	{R2, R3, R5, R6, R7, R8, R10, R12, R19, R20}	{R13}	0.913
rs_{13}	10	{R2, R3, R5, R6, R7, R8, R9, R10, R11, R12}	{R13}	0.902
rs_{14}	10	{R1, R2, R3, R5, R6, R7, R8, R10, R19, R20}	{R13}	0.9
rs_{15}	10	{R1, R3, R6, R8, R9, R10, R11, R16, R17, R18}	{R13}	0.9
rs_{16}	8	{R3, R4, R8, R9, R10, R11, R13, R14}	{R6}	0.89
rs_{17}	10	{R1, R2, R3, R5, R6, R7, R8, R9, R10, R11}	{R13}	0.89
rs_{18}	7	{R3, R4, R6, R8, R9, R10, R11}	{R13}	0.804

11.2 Use Case Study on Conflict Resolution

We have conducted a preliminary analysis of the conflict resolution of owl restrictions on three pairs of ontologies adapted from the *conference* domain of the OAEI benchmark[1] provided by the OntoFarm project [ZS17]. Table 11.5 shows the number of `maxCardinality`, `minCardinality`, `exactCardinality`, `AllValuesFrom`, `someValuesFrom`, and `hasValue` of the merged ontologies. The last two columns show the number of primitive and complex conflicts in each merged ontology.

We observed how easily the small ontologies could cause conflicts when being merged, as they are augmented with various constraints. We applied the proposed methods given in Section 5.6 of Chapter 5 to solve existing conflicts. We then compared the conflicting merged ontology with the revised one with a set of Competency Questions suitable for the tested ontologies (see Appendix A). The results are presented in Table 11.6. The merged ontology that was revised by our approach could achieve homogenous answers, whereas the conflicting one returns contradicting answers. This test demonstrates that applying our method on the conflicting merged ontology can provide homogenous answers and shows the applicability of our method in practice.

[1] `http://oaei.ontologymatching.org/2019/conference/index.html`

TABLE 11.4: All possible compatible maximum sets for user-selected GMRs $\mathcal{U} = \{R1, R2, R3, R8, R10, R19\}$. Green (no-line): user-selected compatible GMRs; Red (double-underline): user-selected incompatible GMRs; Orange (underline): extra compatible GMRs.

$\mathcal{RS}$	$\mathcal{K}$	Compatible	Incompatible	Score
rs_1	11	{R1, R2, R3, R5, R7, R8, R10, R13, R14, R19, R20}	-	1.0
rs_2	10	{R1, R2, R3, R5, R6, R7, R8, R10, R19, R20}	-	0.963
rs_3	11	{R2, R3, R5, R7, R8, R10, R12, R13, R14, R19, R20}	{R1}	0.918
rs_4	11	{R2, R3, R5, R7, R8, R10, R13, R14, R15, R19, R20}	{R1}	0.907
rs_5	10	{R2, R3, R5, R6, R7, R8, R10, R15, R19, R20}	{R1}	0.882
rs_6	10	{R2, R3, R5, R6, R7, R8, R10, R12, R19, R20}	{R1}	0.881
rs_7	11	{R1, R2, R3, R5, R7, R8, R9, R10, R11, R13, R14}	{R19}	0.875
rs_8	10	{R1, R2, R3, R5, R6, R7, R8, R9, R10, R11}	{R19}	0.838
rs_9	11	{R2, R3, R5, R7, R8, R9, R10, R11, R12, R13, R14}	{R1, R19}	0.793
rs_{10}	11	{R1, R3, R8, R9, R10, R11, R13, R14, R16, R17, R18}	{R2, R19}	0.791
rs_{11}	11	{R2, R3, R5, R7, R8, R9, R10, R11, R13, R14, R15}	{R1, R19}	0.782
rs_{12}	10	{R2, R3, R5, R6, R7, R8, R9, R10, R11, R15}	{R1, R19}	0.757
rs_{13}	10	{R2, R3, R5, R6, R7, R8, R9, R10, R11, R12}	{R1, R19}	0.756
rs_{14}	10	{R1, R3, R6, R8, R9, R10, R11, R16, R17, R18}	{R2, R19}	0.754
rs_{15}	11	{R3, R8, R9, R10, R11, R13, R14, R15, R16, R17, R18}	{R1, R2, R19}	0.698
rs_{16}	10	{R3, R6, R8, R9, R10, R11, R15, R16, R17, R18}	{R1, R2, R19}	0.672
rs_{17}	8	{R3, R4, R8, R9, R10, R11, R13, R14}	{R1, R2, R19}	0.618
rs_{18}	7	{R3, R4, R6, R8, R9, R10, R11}	{R1, R2, R19}	0.582

TABLE 11.5: Three sample merged ontologies. Number of OWL restrictions and primitive and complex conflicts are shown.

id	Merged Ontology	Owl Restriction						Conflict	
		max Cardinality	*min Cardinality*	*exact Cardinality*	*allValues From*	*someValues From*	*has Value*	Primitive	Complex
$\mathcal{O}_{M_1}$	cmt conference	3	5	5	1	12	0	0	3
$\mathcal{O}_{M_2}$	cmt confOf	2	8	14	10	7	0	1	1
$\mathcal{O}_{M_1}$	conference confOf	1	6	10	19	8	0	0	1

11.3 Summary

This chapter analyses the compatibilities between the Generic Merge Requirements (GMRs) introduced in Chapter 5 through use case studies. For the given user-selected GMRs, we show possible supersets of compatible GMRs that can be fulfilled simultaneously. For each superset, we indicate the length of the clique and the incompatible GMRs, if available. Moreover, the suggested supersets are ranked based on our proposed criteria, and their achieved scores are presented. Investigating the extent to which the users agree with the ranked list of compatible GMRs is on our future agenda.

To analyze our proposed method for conflict resolution of owl restrictions, we have

TABLE 11.6: CQs answers on the conflicted and revised merged ontology $\mathcal{O}_M$ given in Table 11.5.

CQs	conflicted $\mathcal{O}_{M_1}$	revised $\mathcal{O}_{M_1}$	conflicted $\mathcal{O}_{M_2}$	revised $\mathcal{O}_{M_2}$	conflicted $\mathcal{O}_{M_3}$	revised $\mathcal{O}_{M_3}$
CQ_1	YES	YES	YES	YES	YES	YES
CQ_2	Person	Person	Person	Person	Person	Person
CQ_3	exCard 1, minCard 1	exCard 1	exCard 1, minCard 1	exCard 1	minCard 1	minCard 1
CQ_4	YES	YES	YES	YES	YES	YES
CQ_5	Conference	Conference	Conference	Conference	Conference	Conference
CQ_6	exCard 1, maxCard 2	exCard 1	exCard 1	exCard 1	maxCard 2	maxCard 2
CQ_7	NO	YES	NO	NO	YES	YES
CQ_8	Reviewer, Meta-Reviewer	Reviewer	YES	YES	YES	YES
CQ_9	Paper	Paper	Contribution	Contribution	Contribution	Contribution
CQ_{10}	int	int	string, int	string/int	int	int

conducted a preliminary test on a set of merged ontologies. For each merged ontology, we present the number of their owl restrictions along with the number of primitive and complex conflicts. Through a set of Competency Questions (CQs), we compared the conflicting merged ontology with the revised one obtained by our approach. The revised merged ontologies provide homogenous answers to the given CQs.

12

Experimental Tests on Inconsistency Handling of Merged Ontologies

This chapter is devoted to present a set of experimental tests on the proposed inconsistency handling method, introduced in Chapter 6. To show the applicability of our method, we conducted three tests. In Section 12.1, we show the characteristics of the inconsistencies of the merged ontology, if applicable. We also present the effect of restricting the process to the root unsatisfiable concepts in the justification sets. In Sections 12.2, we investigate the quality of the result by a Competency Questions test on two versions of consistent merged ontologies. In Section 12.3, we examine the time performance of our proposed method. Finally, in Section 12.4, a summary of this chapter is presented.

In our experimental tests, we have used the Pellet reasoner[1] [SPG+07] alongside with OWL-API explanation[2]. The maximum explanation for the reasoner has been adjusted to 5. Moreover, α and β parameters have been set to 1 and 0.5, respectively. These parameters determined empirically, but we make no claim that these are optimal values. The parameters can be determined by the users in our tool. The experimental tests in this chapter have already been carried out in other datasets in [BKR19a]. While in this chapter, we carry out the tests on the introduced datasets in Chapter 8, in which their merged ontologies have been built in Chapter 10. Moreover, a list of used notations, symbols, and nomenclature is presented in Table 12.1.

[1] Pellet-owlapiv3 version 2.3.2

[2] `http://owl.cs.manchester.ac.uk/research/explanation/`

TABLE 12.1: The used notations, symbols, and nomenclature in Chapter 12.

Notation	Description
$\mathcal{O}_S$	a set of source ontologies
$\mathcal{O}_M$	a consistent merged ontology
$\mathcal{O}'_M$	an inconsistent merged ontology
$\mathcal{M}$	a mapping set between the source ontologies
X	a set of axioms
$\mathcal{C}_{un}$	a set of unsatisfiable concept in $\mathcal{O}'_M$
$Root_{\mathcal{C}_{un}}$	a set of root concepts
$\mathcal{J}$	a set of justifications
d_1-d_{12}	twelve tested datasets
CQ	Competency Question
TP	True Positive
FP	False Positive
FN	False Negative
TN	True Negative

12.1 Characteristics of Inconsistent Ontologies

In this section, we aim to examine the characteristics of the inconsistency on different versions of merged ontologies, build in Chapter 10, for our datasets from Chapter 8. The results of this test are presented in Table 12.2, as:

- The first column shows the versions of the merged ontologies that have been tested.
- If the merged ontology is consistent and does not have any unsatisfiable concepts, we show it by ✓symbol. Otherwise, it failed (✗) under one of the following cases[3]:
  - ✗$_{CaseI}$: the merged ontology is inconsistent (there is no model of ontology), but it is coherent (it does not have any unsatisfiable concepts). So, in this case, the reasoner cannot extract a model of the ontology.
  - ✗$_{CaseII}$: the merged ontology is consistent, but it is incoherent and contains unsatisfiable concepts.
- We present the number of unsatisfiable concepts C_{un}, the size of the justification set $\mathcal{J}$, and the total number of conflicting axioms $|X|$, if available.
- We show the statistic values if we restrict the experiments only to the root unsatisfiable concepts $Root_{C_{un}}$ (columns 3-5) or consider all unsatisfiable concepts C_{un} (columns 6-8).
- In the last column, we show the total number of axioms in the tested ontologies. To make the table concise, for rows containing several ontologies, we show the average value of their axioms. For instance, in the second row, the merged ontologies d_1V1 and d_1V_2 have 721 axioms, and d_1V_3 has 717 axioms. We show their average value, i.e., 720 in the second row.

  Note that, the number of total conflicting axioms can be more than the total number of axioms in the ontology, as the conflicting axioms might be duplicated in different justification sets, cf. the example of Section 6.2.1 in Chapter 6 where justification sets contain duplicated axioms.

[3]The other cases from Section 6.2 of Chapter 6 did not occur on our datasets. Thus, we do not report them in this test.

TABLE 12.2: The characterization of the consistency test: consistent without unsatisfiable concepts (✓); inconsistent without unsatisfiable concepts (✗_{CaseI}); consistent with unsatisfiable concepts (✗_{CaseII}), including the number of root ($Root_{C_{un}}$) or all unsatisfiable concepts (C_{un}), number of justification set $\mathcal{J}$, and number of all axioms (X) belonging to the justification sets. The average number of axioms for each group of ontologies are shown in the last column. The symbol $\star$ shows that the reasoner could not return the answer within a predetermined maximum time.

id	Test	root only			all			| axiom |
		$\lvert Root_{C_{un}}\rvert$	$\lvert\mathcal{J}\rvert$	$\lvert X\rvert$	$\lvert C_{un}\rvert$	$\lvert\mathcal{J}\rvert$	$\lvert X\rvert$	
d_1V_1-V_3	✓	-	-	-	-	-	-	720
d_1V_4, V_5	✗_{CaseII}	1	2	12	1	2	12	721
d_1V_6	✗_{CaseII}	1	4	22	1	4	22	715
d_2V_1-V_3	✓	-	-	-	-	-	-	535
d_2V_4, V_5	✗_{CaseII}	1	1	7	6	12	105	533
d_2V_6	✗_{CaseII}	1	2	13	6	19	154	532
d_3V_1-V_3	✓	-	-	-	-	-	-	1027
$d_3V_4, V_5, V_7, V_8, V_{10}, V_{11}$	✗_{CaseII}	2	3	18	2	3	18	1043
d_3V_6, V_9, V_{12}	✗_{CaseII}	2	6	33	2	6	33	1038
d_4V_1, V_2	✗_{CaseII}	5	18	60	12	50	226	2045
d_4V_3	✗_{CaseII}	5	19	64	12	54	247	2041
d_4V_4, V_5	✗_{CaseII}	3	10	69	68	335	2355	2059
d_4V_7, V_8	✗_{CaseII}	4	15	123	11	50	458	2084
d_4V_{10}, V_{11}	✗_{CaseII}	5	20	146	65	320	2260	2066
d_4V_6, V_9, V_{12}	✗_{CaseII}	15	56	392	68	336	2388	2063
d_5V_1-V_4	✓	-	-	-	-	-	-	1401
$d_5V_5, V_7, V_8, V_{10}, V_{11}$	✗_{CaseII}	20	58	402	69	341	2420	1413
d_5V_6, V_9, V_{12}	✗_{CaseII}	32	94	632	83	415	2984	1408
$d_6V_1V_2$	✗_{CaseII}	42	156	1092	175	862	6122	3013
d_6V_3	✗_{CaseII}	42	162	1210	175	895	6124	3020
d_6V_4-V_{12}	✗_{CaseII}	53	196	1545	192	1015	6195	3120
d_7V_1-V_3	✓	-	-	-	-	-	-	42386
d_7V_4, V_5	✗_{CaseII}	3	5	31	520	2548	17839	42013
d_7V_6	✗_{CaseII}	3	5	31	520	2550	17852	42006
d_8V_1, V_2	✗_{CaseII}	119	331	3846	1021	$\star$	$\star$	51856
d_8V_3	✗_{CaseII}	119	335	3872	1021	$\star$	$\star$	49961
d_8V_4-V_6	✗_{CaseI}	-	-	-	-	-	-	51508
d_9V_1-V_{12}	✓	-	-	-	-	-	-	5352
$d_{10}V_1$-V_{12}	✗_{CaseI}	-	-	-	-	-	-	48911
$d_{11}V_1$-V_{12}	✗_{CaseI}	-	-	-	-	-	-	50982
$d_{12}V_1$-V_3	$\star$	$\star$	$\star$	$\star$	$\star$	$\star$	$\star$	155073
$d_{12}V_4$-V_{12}	✗_{CaseI}	-	-	-	-	-	-	146814

In all tests, only one iteration of all process was required to make the inconsistent merged ontologies into the consistent ones. From the results in Table 12.2, we draw the following points:

- Analyzing restrictions on the root or all unsatisfiable concepts: In d_2, d_4-d_{12}, the number of root unsatisfiable concepts is less than the number of all unsatisfiable concepts. For example, in $d_4 V_4, V_5$, there are only 3 unsatisfiable root concepts, but 68 total unsatisfiable concepts. Consequently, the number of justifications and conflicting axioms is less when we restrict the experiments only to the root unsatisfiable concepts. This causes low computational complexity. In d_1 and d_3, these values are the same.

- Effect of using perfect or imperfect mapping on the consistency of the result: In d_1-d_3, and d_7, the merged ontology built by the perfect mappings is consistent, whereas, the merged ontologies created by the imperfect mappings in these datasets are inconsistent. Moreover, in d_4, the merged ontologies using perfect or imperfect mappings in both cases are consistent. However, the merged ontology built by the perfect mappings has less number of conflicting axioms. This can demonstrate the power of using perfect mappings on the result, if available. In d_5, the merged ontology used the perfect mapping is a consistent one. Surprisingly, we observe that the merged ontology, built by the imperfect mapping that applied local and refinements (V_4), is also consistent. Whereas, V_5 (with only global refinements) and V_6 (with no refinements) are inconsistent ones. Applying local and global refinement in this test could help that the merged ontology be consistent.

- Comparing applying refinements or no refinements on the result: Besides the effect of the consistency of the result in d_5 that described before, we observe that the merged ontologies built by applying local or global refinements have less number of (root or all) unsatisfiable concepts in d_4 and d_5 and less number of conflicting axioms in d_2, d_3, d_6, and d_7. This observation demonstrates the strength of refinements in the result.

- Comparing the merged ontologies built via the n-ary approach (V_1-V_3) or binary methods (V_7-V_{12}) with imperfect mappings: There is no difference in the consistency or inconsistency of the result, because it depends on the type of mappings used by the merged ontologies.

- Observing the ability of the reasoner in the large scale ontologies: We set a maximum time as 43000 seconds (an empirical threshold) and observe whether the reasoner can return the answer within this time. If no, we mark this case by $\star$ symbol in Table 12.2. In tests with all unsatisfiable concepts in d_8, the reasoner failed to return the justification sets for 1021 concepts in the maximum limited time. Moreover, in d_{12}, the reasoner could not return any answer about the consistency or inconsistency of the merged ontology. This is computationally an expensive task for a complex knowledge base. For instance, d_{12} has more than one hundred fifty thousand axioms. We have tested these ontologies with another reasoner, but the same outcome is achieved. Our goal was to contribute to the ranking of the conflicting axioms that cause inconsistencies, not on improving the reasoner's

ability. Thus, we consider these cases as the limitation of the reasoner, not our method. Moreover, comparing the behavior of different reasoners is out of the scope of this research. For this, we refer to the existing survey, such as [KP17].

12.2 Answering Competency Questions

Competency Question (CQ)s are a list of questions used in the ontology development life cycle, which the ontology should answer. To this end, we have provided a list of CQs (see Appendix A) in the conference domain and run them on the conference datasets (d_1-d_6). We run this test on those ontologies that could not successfully pass the consistency test. For them, we create two different consistent and coherent merged ontologies, as:

- $\mathcal{O}_{M_1}$: The consistent and coherent merged ontology, which used our ranking method and applied our proposed plan.
- $\mathcal{O}_{M_2}$: The consistent and coherent merged ontology, which was achieved by human intervention.

For the CQs test, we aim to retrieve precision and recall of answering the given CQs by the merged ontologies. Thus, we determine positive and negative CQs along with TP (True Positive), FP (False Positive), and FN (False Negative) responses, as follows:

- *Positive CQ*: The answer to the CQ must be in the source ontologies.
- *Negative CQ*: The answer to the CQ must not be in the source ontologies.
- TP: If the merged ontology correctly answers a positive CQ, we mark the answer to this CQ by TP.
- FN: If the merged ontology incorrectly answers a positive CQ, we mark the answer to this CQ by FN.
- FP: If the merged ontology provides a correct answer to a negative CQ, we mark the answer to this CQ by FP.
- TN: If the merged ontology incorrectly answers a negative CQ, we mark the answer to this query by TN.

Note that we use the same test for evaluating the functional dimension in Chapter 13.

Figure 12.1 shows the result. The precision is achieved uniquely 1 for all tested ontologies. Thus, we only show the values of the recall in the figure. We observe that different versions of the merged ontologies achieved the same result. Thus, in the figure, we show one entry for the group of ontologies that have the same result.

By analyzing the result, we observe that out of 48 tests on the merged ontologies, in 6 cases, $\mathcal{O}_{M_1}$, the consistent merged ontology created by our method, achieved the same result with the human-created one ($\mathcal{O}_{M_2}$). Moreover, $\mathcal{O}_{M_1}$ achieved better recall in 9 cases. In 33 cases, the human-created one ($\mathcal{O}_{M_2}$) was better than $\mathcal{O}_{M_1}$ (up to 0.07 percentage). The results achieved from this test show the feasibility and reliability of our method.

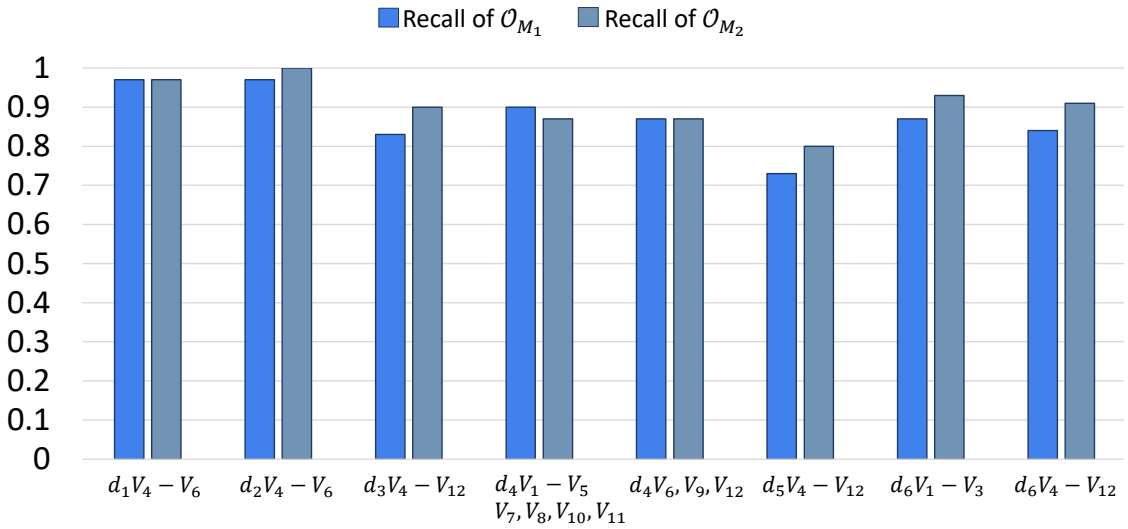


FIGURE 12.1: Competency Question-based experimental tests: The values of the recall for the consistent merged ontology achieved by our method $\mathcal{O}_{M_1}$ and the human-created $\mathcal{O}_{M_2}$ are presented.

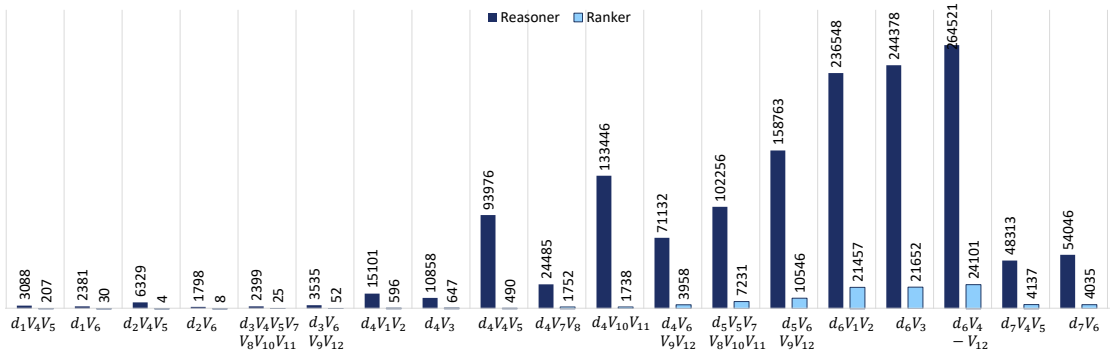


FIGURE 12.2: The scalability test: Time performance in milliseconds of processing the unsustainable root concepts by the reasoner and ranker.

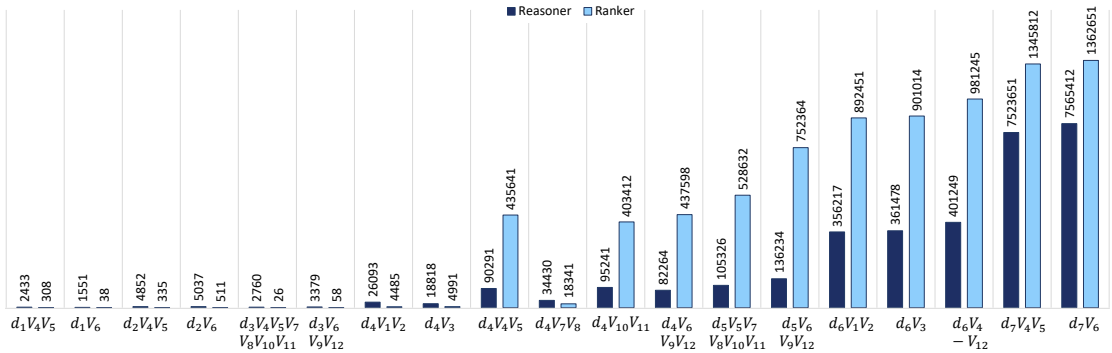


FIGURE 12.3: The scalability test: Time performance in milliseconds of processing all unsatisfiable concepts by the reasoner and ranker.

12.3 Scalability

To evaluate the run time performance of our proposed approach, we observed three processing time of:

- testing the consistency of the ontology by the reasoner and extracting the unsatisfiable concepts C_{un} alongside with their justification sets $\mathcal{J}$,
- ranking the axioms belonging to justification sets by our proposed method, and
- generating the resolution plan.

We present the results in Figure 12.2 and Figure 12.3 in milliseconds. We observed that time for generating plan in all merged ontology were less than 50 milliseconds. Thus, we do not report them in the figures to keep the presentation of the results concise and straightforward. Figure 12.2 shows the result of running the test by considering only the root unsatisfiable concepts, whereas Figure 12.3 presents the runtime performance of processing all unsatisfiable concepts. In both, we show the average time for different versions of each dataset, as has been categorized in Table 12.2.

From the results, we can draw the following points:

- Comparing the runtime performance of processing root or all unsatisfiable concepts: We observed that in d_7, there is a large difference between processing the root or all unsatisfiable concepts. Because there are only 3 roots, and the reasoner and ranker have to process only 3 unsatisfiable concepts with 31 conflicting axioms. But if we consider all unsatisfiable concepts, the reasoner and ranker should process 17839 axioms for 520 unsatisfiable concepts. In such cases, restricting the process to the root concepts can improve the time considerably. Nevertheless, this argument can not be straightforwardly generalized for all tests. In all cases, the number of root unsustainable concepts alongside with the conflicting axioms is less than when we process all unsatisfiable concepts. So, it is expected to have less time processing. However, this is not the case. Because in the test using the root concepts, the reasoner should extract the potential root concepts among all unsatisfiable concepts. This is, however, a time-consuming process. For this reason, the processing time of the reasoner in some cases (cf. $d_4V_{10}V_{11}$) is less when we do not extract the root concepts. As a whole, out of 19 group tests, 12 times the reasoner could be faster when we restrict the test to the root concepts, and 7 times processing all unsatisfiable concepts was faster. Moreover, we observed that the ranking process is faster in all cases if we restrict the process to the root concepts as it depends on the number of conflicting axioms that must be processed.
- Analyzing the time performance of the reasoner on the large scale ontologies: In all tests except d_8 and d_{12}, the reasoner could provide the answer within the maximum limited time (43000 seconds). In d_8V_1 testing with root unsatisfiable concepts, the time for reasoner and ranker were 17710713 and 72435 milliseconds, respectively. However, in processing all unsatisfiable concepts, the reasoner could not extract the justification sets in the maximum limited time as it needs to find justification sets for 1021 unsatisfiable concepts. Indeed, the reasoner got stuck to retrieving justification sets because the given ontology is big and complex. In $d_{12}V_1$-V_3, the

reasoner failed to return any answer for the consistency test within the limited processing time.

- Comparing the processing time of the reasoner and ranker: The processing time of the ranker mainly depends on the number of conflicting axioms that must be processed as well as the number of source ontologies alongside the number of merged ontologies' axioms. In the tests restricted to the root unsatisfiable concepts, the processing time of the reasoner is more than the ranker. However, in the test with considering all unsatisfiable concepts, when the number of processing axioms is increased, the time for ranking is more than the reasoner time. As the test demonstrated, mostly, the time is increased by the number of conflicting axioms set. Given a complex, time-consuming task overall, the processing time by the ranker seems acceptable.

12.4 Summary

In this chapter, we present a set of experimental tests on the proposed method of inconsistency handling. In particular,

- We present the characteristics of the inconsistent merged ontologies and show the effect of restricting the process to the root unsatisfiable concepts in justification sets. In most datasets, the number of root unsatisfiable concepts is less than the number of all unsatisfiable concepts. Therefore, narrowing the repair process to root concepts can cause low computational complexity. However, this is not straightforward because finding the root concepts is a time-consuming process.
- Generating the merged ontologies by using the perfect mappings rather than imperfect mappings in some cases causes the result to be consistent, and in other cases, causes that the inconsistent merged ontologies have less number of conflicting axioms. This shows the benefit of utilizing perfect mappings, if available.
- The inconsistent ontologies that used a set of refinements during the merge process have less number of conflicting axioms compared to the ontologies built without any refinements.
- In dealing with large scale merged ontologies, the reasoner failed to return the justification sets or an answer about (in-)consistency of merged ontologies.
- We investigate the quality of the result by a set of Competency Questions on two versions of consistent merged ontologies: the consistent merged ontology which used our method and the one which is achieved by human intervention. The test indicates that consistent merged ontologies created by our method are rival with the human-created one.
- We examine the time performance on finding justification sets, ranking the axioms, and generating the resolution plan. The processing time is increased by the number of conflicting axioms that caused inconsistencies.

13

Experimental Tests on the Quality Assessment of the Merged Ontology

To demonstrate the applicability of our evaluation framework, we conducted a series of experiments utilizing the datasets from Chapter 8. We assess how our introduced quality framework can be applied for the ontology merging domain in practice. The experiments are carried on structural, functional, and usability quality dimensions on the merged ontologies that are built by our presented approach from Chapter 4. We measure the quality of the merged ontology in three dimensions. Analyzing the result:

- In structural quality assessment, it shows how well the merged ontology is structured, concerning the syntax and topological properties of the merged ontology.
- In functional quality assessment, it shows how well the intended use and semantics of the merged ontologies are satisfied.
- In usability quality assessment, it shows how well the merged ontology is profiled to address the communication context of the merged ontology.

We evaluate the structural quality indicators along with GMRs in Section 13.1, followed by evaluating the functional and usability quality indicators in Sections 13.2 and 13.3, respectively. We present the time performance of the evaluation process and the overall result demonstration in Section 13.4 and Section 13.5, respectively. We illustrate a total analysis of tested datasets in Section 13.6. We analyze to which extent the evaluation standard has been achieved in Section 13.7, and finally, in Section 13.8, we present a summary of this chapter.

In this chapter, we test the merged ontologies that are built without any refinements to observe different possible anomalies in the tested ontologies. Comparing the effect of different mappings has been done in Chapter 10. Thus, in this chapter, we only consider the perfect mappings for the given source ontologies. In a nutshell, the evaluated ontologies in this chapter are version three (V_3) of the merged ontologies in Chapter 10.

The results described in this chapter have been partially published on another dataset in [BGKR20b]. Moreover, a list of used notations, symbols, and nomenclature is shown in Table 13.1.

TABLE 13.1: The used notations, symbols, and nomenclature in Chapter 13.

Notation	**Description**
$\mathcal{O}_S$	a set of source ontologies
$\mathcal{O}_M$	a merged ontology
n	number of source ontologies
TP	True Positive
FP	False Positive
FN	False Negative
TN	True Negative
$\mathbb{P}$	Precision
$\mathbb{R}$	Recall
$\Sigma\, R$	total number of unsatisfied GMRs
$\Sigma\, e$	total number of anomalous entities
A, B, C	classes
I'	an individual
$\sqsubseteq$	subclass relations between two classes
d_1-d_{12}	twelve tested datasets
$P1$-$P22$	principles of the evaluation standards
CQ	Competency Question
GMR	General Merge Requirement
R1-R20	individual GMRs

TABLE 13.2: Occurrence of unsatisfied GMRs. Σ R shows the number of unsatisfied GMRs, and Σ e shows the total number of anomalous entities.

id	*R1*	*R2*	*R3*	*R7*	*R12*	*R14*	*R15*	*R16*	*R18*	*R19*	Σ R	Σ e
d_1	1	0	0	0	0	0	4	1	0	6	4	12
d_2	0	0	0	0	0	3	0	0	0	1	2	4
d_3	1	0	0	0	0	0	6	1	1	7	5	16
d_4	2	0	0	0	1	2	6	1	1	8	7	21
d_5	1	0	0	0	0	20	3	0	0	5	4	29
d_6	2	0	0	0	-	20	14	4	1	8	6	49
d_7	0	0	0	0	0	2	0	2	0	7	3	11
d_8	0	0	0	4	533	25	0	29	0	1916	5	2507
d_9	1	44	0	0	0	4	14	0	0	2	5	65
d_{10}	3	286	41	71	-	26	21	2	2	5	9	457
d_{11}	3	313	41	75	-	26	21	2	2	5	9	488
d_{12}	0	27	0	176	-	187	44	4	5	252	7	695

13.1 Quality Evaluation of the Structural Dimension

In this test, we observe the quality of the merged ontologies for our datasets with the structural indicators. The measurement procedures for the evaluation function of each indicator are presented in Appendix B. The entailment test based on [BPS11] includes subsumption, equivalence, and satisfiability tests. We report the numbers of subsumption and equivalence entailment tests from the source ontologies which are not entailed by the merged ontology. For the satisfiability test, we refer to the consistency test in Chapter 12.

The evaluation functions of the indicators in this dimension return the absolute numbers. Thus, in Table 13.2, we show the absolute number for each indicator of GMRs in our datasets. The GMRs with zero values for all datasets are not presented in the table. Moreover, we show the total numbers of GMRs that are not satisfied in each dataset, given by Σ R. It indicates how many GMRs out of 20 are not satisfied for that dataset. Additionally, we present the total numbers of entities that have anomalies among the represented GMRs, given by Σ e.

Note that the statistical numbers of structural quality for all datasets are presented in a table view (see Table 13.2). However, the user will receive the result through a GUI. Figure 13.1 shows the result in such a GUI for the merged ontology in dataset d_3. The GUI shows the practical output that a user can see. For each indicator, the output in GUI presents:

- Evaluation dimension's name
- Indicator's name
- Indicator's description

- Fulfilled indicator; completely satisfied indicator for the tested ontology
- Failed indicator; not satisfied indicator at least for one entity of the tested ontology
- Frequency; the number of affected entities for the failed indicator
- Ontology entities affected; list of anomalous entities for the failed indicator
- Repair option; possible repair solution for detected anomalies (see Appendix B)

From the results in Table 13.2, we can observe:

- The merged ontologies in each dataset differ in the quality assessment employing GMRs.
- The total number of unsatisfied GMRs (given by $\Sigma\ R$) can demonstrate how well the merged ontology is structured. For example, the merged ontologies in second and seventh datasets (d_2 and d_7) only in 2 out of 20 GMRs have anomalies, indicating that they are well-structured than others. Several other merged ontologies, such as in d_4 and d_{11}, have anomalies in many GMRs and are poorly structured compared to others. We also show the number of entities affected by the failed indicators (given by $\Sigma\ e$). From these total numbers ($\Sigma\ R$ and $\Sigma\ e$), the user can observe the weaknesses of the merged ontology and drive further improvements before using the merged ontology in the desired application. Note that, if the merged ontology is inconsistent, the entailment test by the reasoner for *R12* cannot be performed. For this reason, there is no value for *R12* in d_7, d_{10}-d_{12}.
- Part of the results focuses on how well the merge process is performed. For instance, *R1* indicates how many classes from the source ontologies are not preserved in the merged ontology. It is occurred during the merge process by the underlying merge method.
- Another group of indicators indicates that the respective source ontologies itself have some anomalies. For example, in the merged ontology of dataset d_8, many classes do not have any connections in the is-a hierarchy. As a result, merging them carries with it this flaw.
- Overall, all datasets have anomalies in at least one GMR. This indicates the need for refinements, where we discussed the effect of applying refinements in detail in Chapter 10.

13.2 Quality Evaluation of the Functional Dimension

We have provided two different tests to evaluate the intended use and semantics of the merged ontology:

13.2.1 Quality Assessment of Intended Use with Competency Questions Testing

In this test, we evaluate the functional measure by analyzing user-provided Competency Questions (CQ)s in order to assess the intended use of the merged ontologies. Indeed,

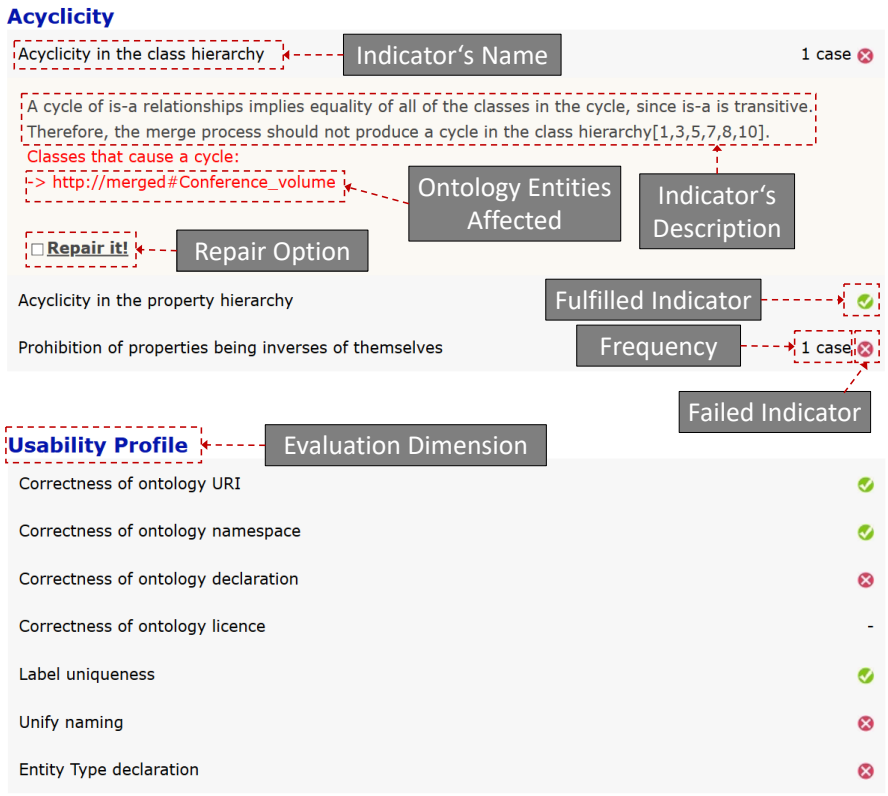


FIGURE 13.1: The GUI for structural quality evaluation of the merged ontology in dataset d_3.

CQs are domain-specific and provided by the user. Thus, for the conference domain, we have provided a list of CQs (see Appendix A) and applied them on the conference datasets (d_1-d_6). We aim to observe how the modeling of the merged ontology is aligned with its intended use.

To quantify the precision and recall for the matching between the merged ontology and its intended use in CQs tests, we determine positive and negative CQs along with TP (True Positive), FP (False Positive), and FN (False Negative) responses, as follows:

- *Positive CQ*: It is the CQ that at least one of the source ontologies can answer. Note that for this CQ, the merged ontology is able to answer this CQ, and even the answer of merged ontology can be more comprehensive than the source ontologies because the merge process aims to achieve more comprehensive knowledge.
- *Negative CQ*: It is the CQ that can not be answered by any source ontologies.
- TP: If the merged ontology correctly answers a positive CQ, we mark the answer to this CQ by TP.
- FN: If the merged ontology incorrectly answers a positive CQ, we mark the answer to this CQ by FN.

TABLE 13.3: True or false positive and negative responses.

	Positive CQ (Intended Answer)	negative CQ (Non-Intended Answer)
Correct Answer	True Positives TP	False Positives FP
Wrong Answer	False Negatives FN	True Negatives TN

- FP: If the merged ontology provides a correct answer to a negative CQ, we mark the answer to this CQ by FP.
- TN: If the merged ontology incorrectly answers a negative CQ, we mark the answer to this query by TN[1].

Table 13.3 shows this. These values are adapted in this way that positive and negative queries are defined based on the source ontologies and TP, FP, and FN are calculated on the merged ontologies upon them. Thus, the results achieved by merged ontology is compared with the knowledge provided by the source ontologies.

The results are shown in Figure 13.2, where precision $\mathbb{P}$ and recall $\mathbb{R}$ are calculated based on TP, FP, and FN for each dataset. All merged ontologies evaluated in this test achieved precision 1 because the FP of all of them is zero. Indeed, if none of the source ontologies cannot answer the negative CQ (the CQs that must not be answered by the source ontologies), the merged ontology could not answer it. Because during the merge process, no further information than source ontologies is added to the merged ontology. If a merged ontology is built by human intervention, that might bring some new knowledge than source ontologies knowledge. In this case, non-zero values would be possible for FP. Moreover, FN is sometimes non-zero because the answers from the merged ontology are wrong in comparing the answer of source ontologies. Same as the assumption in Chapter 10, the wrong answer might happen, when all source ontologies return, for instance, true for a true or false query, but the merged ontology returns false. Alternatively, in the questions related to the is-a hierarchy, the merged ontology shows a different hierarchy than the hierarchy of the source ontologies.

As a whole, the recall of all tested ontologies varied between 0.93 and 1, whereas, the precision achieved uniquely 1. If we interpret high and max values with values above 0.5 (our empirical threshold), the merged ontologies are achieved GOOD labels in all tested datasets.

13.2.2 Quality Assessment of Intended Semantics with Query Testing

In this test, we aim to evaluate the intended semantics of the ontology with respect to the source ontologies. To this end, we created two types of queries similar to [HVHTT05] on individuals and is-a relations:

- In the is-a-based queries, for each subclass-of relation like '$A \sqsubseteq B$', we make a true query '$A \sqsubseteq B$?', and a false query like '$A \sqsubseteq C$?', which '$B \neq C$' and '$A \not\sqsubseteq C$'.

[1]To calculate precision and recall equations, we do not need TN.

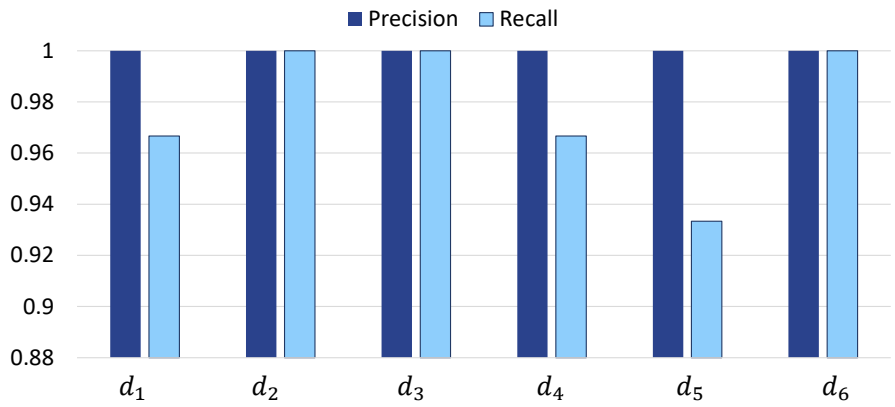


FIGURE 13.2: Functional quality's evaluation for the intended use via Competency Questions. All tests achieved GOOD labels.

- In the individual-based queries, for each individual I' of concept A, we create a positive individual query like *"is I' a A?"*, and a negative individual query like *"is I' a B?"*, where "$B \neq C$" and B is another concept selected randomly.

To quantify the precision and recall for the matching between the merged ontology and its intended semantics in the query tests, we determine intended and non-intended answers along with TP, FP, and FN, as follows:

- *Intended answer*: We expect that the answer from the merged ontology for the true query is true, and for the false query is false. If so, we mark them as intended answers.
- *Non-intended answer*: If the answer from merged ontology for the true query is false and for the false query is true, we mark them as non-intended answers.
- TP: If the answer of the merged ontology is correct compared to the intended answer, we mark the answer to this query by TP.
- FN: If the answer of the merged ontology is incorrect compared to the intended answer, we mark the answer to this query by FN.
- FP: If the answer of the merged ontology is equal to the non-intended answer, we mark the answer to this query by FP.
- TN: If the answer of the merged ontology is equal to the non-intended answer, we mark the answer to this query by TN.

Table 13.3 shows this. Our adaption is made explicitly in the context of the ontology merging process. Note that true and false queries are defined based on the knowledge provided by the source ontologies. Thus, this definition considers the knowledge of source ontologies and compares the knowledge of the merged ontology with the respective source ontologies to obtain TP, FN, and FP.

The presented results in Figure 13.3 shows the precision and recall of running a total of 268 queries on our tested datasets. The number of running queries for each dataset is

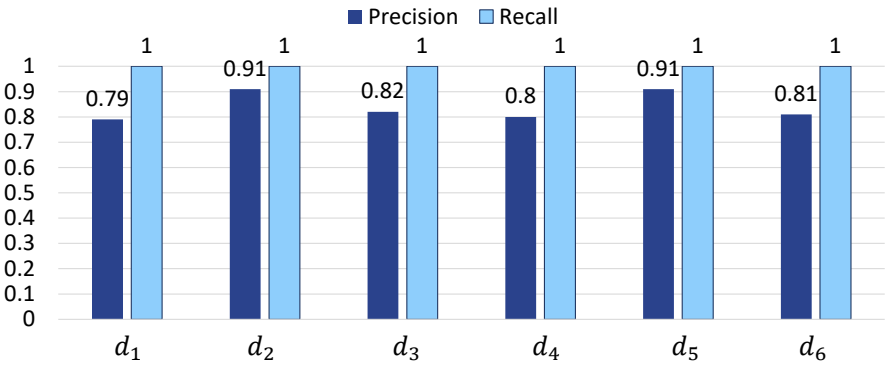


FIGURE 13.3: Functional measure's evaluation for intended semantics via queries. All tests achieved GOOD labels.

shown under each dataset. Remarkably, all merged ontologies could provide a correct answer for some negative queries. Thus, the precision of all tested ontologies varied between 0.79 and 0.91, whereas, the recall achieved uniquely 1. Interpreting high and max values with values above 0.5 shows GOOD labels for all tested datasets. The test demonstrates that the intended semantics concerning the source ontologies is high.

13.3 Quality Evaluation of the Usability Dimension

In this test, we aim to observe the usability quality of the merged ontologies. The evaluation functions for the quality indicators in this dimension return a boolean data type to show whether the associated indicators are satisfied or not. Table 13.4 shows the evaluation of usability indicators of the merged ontologies in all datasets. The GUI for this dimension has been shown at the bottom of Figure 13.1.

From the results presented in Table 13.4, we can observe that all merged ontologies contain the correct URI and namespace with unique labels and decelerated entities, whereas none of them include ontology deceleration. The merged ontologies in the conference (d_1-d_6) and anatomy (d_7) datasets do not follow unified naming conventions. Moreover, all datasets failed to declare the *owl:Ontology* tag where the ontology metadata should provide it. Our datasets do not include any license. Thus they are not reported in the table. Analyzing conflicts between licenses is left open for future work.

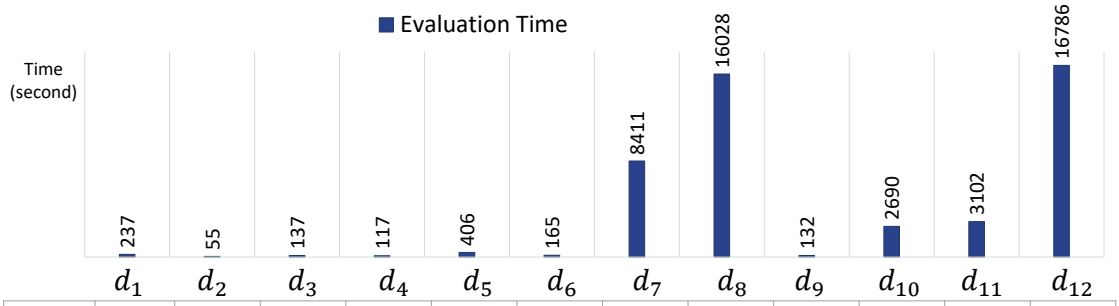
Overall, for all tested datasets, 53 out of 72 cases in Table 13.4 are satisfied, which illustrates well-profiles for ontologies with a proper communication context.

13.4 Time Performance

Figure 13.4 shows the required time for evaluating the merged ontologies of structural and usability dimensions together. We ran each test ten times and presented the average values.

TABLE 13.4: Evaluating the usability quality of the merged ontologies.

id	Annotation of ontology itself			Annotation of ontology's entities		
	Ontology URI	Name Space	Ontology Declaration	Label Unique	Entity Declaration	Unify Naming
d_1-d_7	✓	✓	✗	✓	✓	✗
d_8-d_{12}	✓	✓	✗	✓	✓	✓



	d_1	d_2	d_3	d_4	d_5	d_6	d_7	d_8	d_9	d_{10}	d_{11}	d_{12}
n	2	2	3	4	4	7	2	2	5	17	22	55
$\mathcal{O}_S$Axioms	726	534	1061	1940	1391	3037	41407	49433	5432	51461	56893	158567
$\mathcal{O}_M$Axioms	717	534	1026	2041	1395	3020	42384	49961	5321	47488	49952	154749

FIGURE 13.4: Runtime performance: numbers (n) and axioms' size of source ontologies $\mathcal{O}_S$ with the axioms' size of merged ontologies $\mathcal{O}_M$ versus the required time for evaluation of the structural and usability dimensions in second.

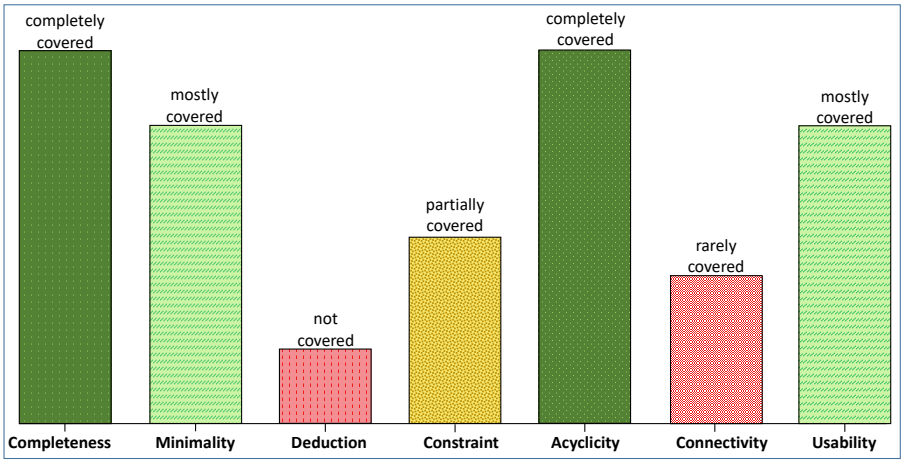


FIGURE 13.5: Overall result's view of aspects for the structural and usability dimensions of a sample merged ontology.

In this figure, we present the number of source ontologies $\mathcal{O}_S$ and their axioms along with the number of axioms of the merged ontologies, to emphasize that there is a linear dependency on the size of input data with respect to the processing time of evaluation. Overall, considering the size of input data for quality evaluation, the time of processing is reasonable.

13.5 Overall Result Demonstrations

To help the user to give an overall glance of all aspects, we provide an interface for whole indicators in five intervals:

- *Completely-Covered*: When all indicators of the aspect are fulfilled.
- *Not-Covered*: When none of the indicators of the aspect are fulfilled.
- *Mostly-Covered*: When the number of fulfilled indicators in the aspect is higher than the number of failed indicators.
- *Rarely-Covered*: When the number of fulfilled indicators in the aspect is lower than the number of failed indicators.
- *Partially-Covered*: When the number of fulfilled indicators in the aspect is equal to the number of failed indicators.

Aspects with one indicator (e.g., deduction) use interval *Completely-Covered* or *Not-Covered* only. Figure 13.5 shows this demonstration for a sample merged ontology.

13.6 Total analyzing

By looking at all results obtained for the tested datasets, we can observe their strengths and weaknesses. For instance, the merged ontology in dataset d_1 failed in 4 out of 20 structural quality indicators and in 2 out of 6 usability quality indicators, showing its weakness. From another point of view, the merged ontology in d_1 succeeded in 16 out of 20 structural quality indicators and in 4 out of 6 usability quality indicators, showing its strength. The merged ontology has achieved a GOOD label in the evaluation of the intended use via the CQs test and the evaluation of intended semantics via query test. Such an analysis can be obtained for other merged ontologies. All the results obtained for structural, functional, and usability dimensions are shown in Figure 13.6.

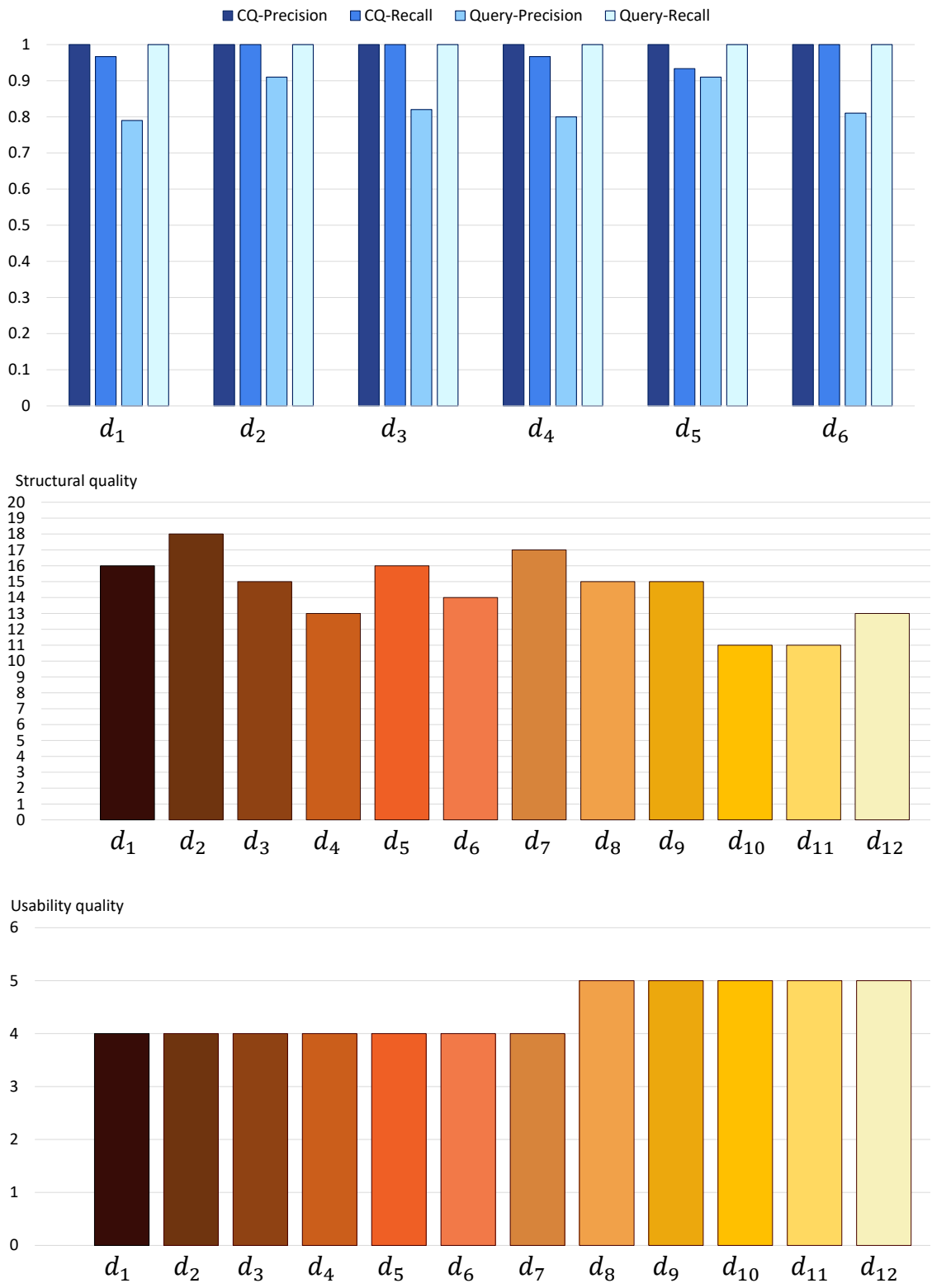


FIGURE 13.6: Total Analyzing: Top: functional quality evaluation in CQ and query tests in GOOD, LESS-GOOD, BAD, and WORSE intervals; Middle: number of fulfilled indicators in the structural quality evaluation among 20 indicators; Down: number of fulfilled indicators in the usability quality evaluation among 6 indicators.

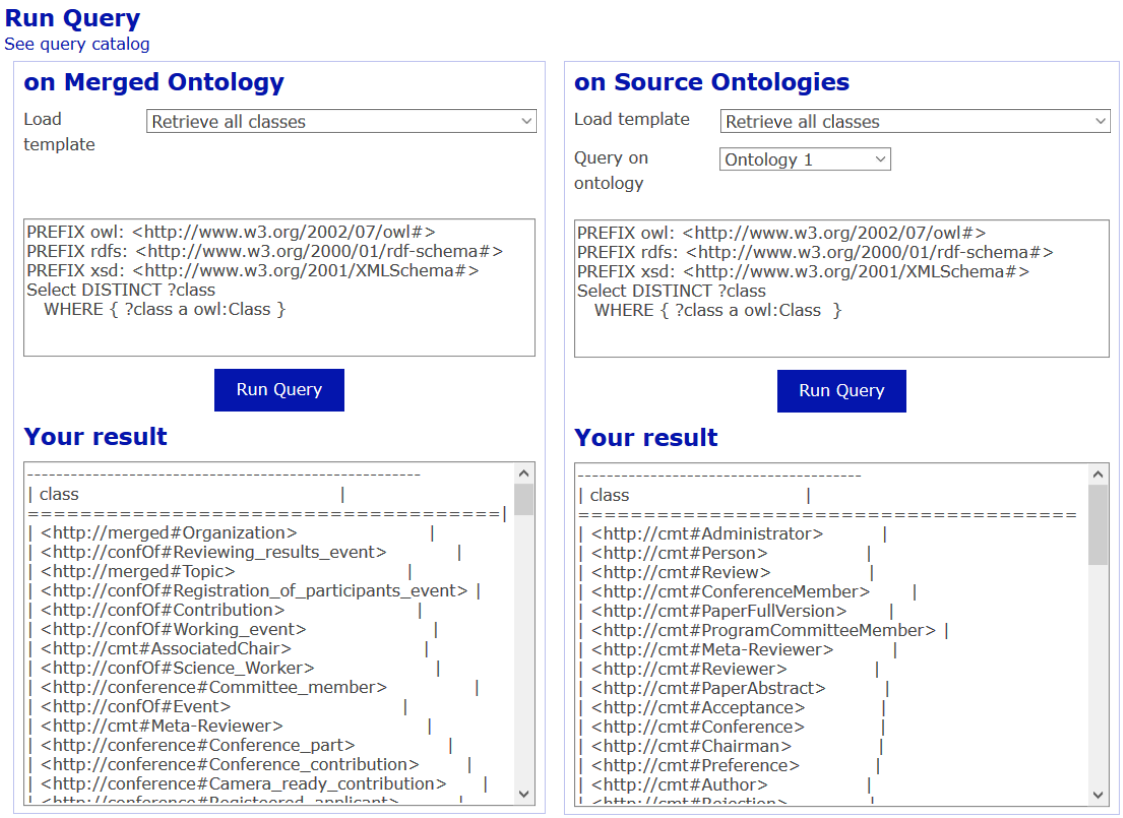


FIGURE 13.7: Query endpoint GUI with the separate results for the merged and source ontologies.

As stated in Section 7.2.2 of Chapter 7, we aimed not to assign a global score to ontologies but to identify their strengths and weaknesses. This is why we have not concluded the experiments saying which ontology is the best in the tested domain; instead, the main features of each merged ontology have been described. The merged ontologies that have been evaluated through structural, functional, and usability dimensions have an acceptable quality, and the decision has to be made by the users or developers based on their needs. They can be improved in several directions in order to increase their quality.

The experimental tests in this chapter show that our evaluation method can detect the main strengths and weaknesses of the evaluated ontologies. Besides, our evaluation framework can compare the merged ontology against the respective source ontologies. The users can investigate the knowledge of the merged ontologies via the SPARQL endpoint in our platform to compare the result against the knowledge of the source ontologies. Figure 13.7 shows the output of running sampled queries on the merged and source ontologies simultaneously. The evaluation of structural and usability dimensions can also be carried by our user-friendly GUI, as shown in Figure 9.5 in Chapter 9.

13.7 Analyzing the Fulfilment of the Principles of Evaluation Standards

This section analyzes the extent to which the principles of evaluation standards are fulfilled in our framework.

- **Usefulness principle**:

  P1. Detecting the involved and affected variables (fulfilled): We detected and took into account the analysis of the source ontologies (in all three quality evaluation dimensions) and the application in which a merged ontology is to be used (in the functional dimension).

  P2. Transparency of the impact of influenced variables (unfulfilled): Since we did not obtain an overall quality rating (see Section 7.2.2), the importance of the source ontologies and the application that wants to use the merged ontology did not consider.

  P3. Clarification of evaluation purposes (fulfilled): The purpose of each evaluation aspect is clarified and defined for the user.

  P4. Competence and credibility of the underlying system (unfulfilled): Analyzing the evaluation framework within a use case application or with user study is on our future work.

  P5. Selection and scope of indicators (fulfilled): Users are able to adjust the scope of the evaluation. They can adjust desired quality indicators based on the purpose of the given task.

  P6. Supported scope and automation (fulfilled): Evaluation is an automatic process for structural and usability dimensions. The functional dimension requires a set of user-provided queries or CQs.

  P7. Completeness of the output (fulfilled): All evaluation framework provides a set of essential quality aspects by covering different quality indicators. It does not rely on only one single aspect. Thus, it is comprehensive.

  P8. Clarity of the result (fulfilled): The result is understandable with our GUI, showing detailed easy understand analysis to the user.

  P9. Use and benefits of evaluation (fulfilled): We emphasized on the weakness and those elements that need to be repaired individually for the user. Thus, users can get benefit from the results and draw further improvements.

- **Feasibility principle**:

  P10. Appropriate methods (fulfilled): The efforts for analyzing source ontologies (those involved one) and the application that wants to use the merged ontology (those affected) are in balance with the desired evaluation output.

  P11. Efficiency of evaluation (fulfilled): In our experimental tests, we have shown that applying all framework is feasible in practice. The complexity of our

evaluation function is liner since they use counting procedures. Thus, the output is achieved in a reasonable time and complexity.

- **Fairness principle**:

  P12. Formal disposal of indicators (fulfilled): In our evaluation framework, each indicator has an exact and systematic definition, certified goal, and precise implementation of the process in practice. They are available for users.

  P13. Disclosure of results (fulfilled): The outputs of evaluation are available for users through a GUI or as a text file.

  P14. Comprehensive and fair examination (fulfilled): The evaluation of the quality of the merged ontology shows both strengths and weaknesses to the user. The evaluation is comprehensive by covering various quality aspects.

  P15. Impartial implementation and output (fulfilled): The merge evaluation techniques can work well independently of the underlying merge method.

- **Accuracy principle**:

  P16. Description of the object to be evaluated (fulfilled): The theoretical aspects of the merged ontology, such as entities' size, are accessible through our tool.

  P17. Description of purposes and procedures (fulfilled): The evaluation functions are documented well.

  P18. Context analysis (fulfilled): The functionality indicators can evaluate the merged ontology based on the given context.

  P19. Declaration of indicators (fulfilled): The formulation of each indicator is defined well.

  P20. Valid and reliable indicators (fulfilled): The quality indicators are built upon well-known literature studies.

  P21. Justified assessments and conclusions (unfulfilled): The evaluation criteria can provide an analytic view on how well the created merged ontology reflects the given source ontologies. But we did not make a final conclusion.

  P22. Meta-evaluation (unfulfilled): A meta-evaluation on the output of the evaluation framework is on our future agenda.

13.8 Summary

In this chapter, we conduct a set of experimental tests on structural, functional, and usability dimensions of the proposed evaluation framework. In particular,

- In the evaluation of the structural dimension, we show the topological properties of merged ontologies by analyzing the extent to which the GMRs are satisfied. Various merged ontologies differ in the quality assessment employing GMRs.

- We evaluate merged ontologies in the functional dimension by using intended use and semantics. Our tested ontologies achieve high intended use and semantics concerning the source ontologies for the given Competency Questions and queries.
- In the evaluation of the usability dimension, we show the communication context of merged ontologies. The most tested ontologies are well-profiled.
- The processing time of evaluation in our dataset is reasonable concerning the number of their axioms.
- We provide an interface to ease the interpretation of the results for users. The interface presents each aspect of evaluation in five intervals from completely-covered to not-covered.
- Most principles of evaluation standards in our evaluation framework are fulfilled.

Part IV

Conclusion

14 Summary

This chapter briefly reiterates our assumptions, the solution we provided, and the results of the evaluation. Thereafter, we bring the dissertation to a close by reviewing its achievements and the extent to which the research hypothesis is satisfied.

In this thesis, we explained, and with the aid of a deep literature review, demonstrated that four key aspects of the ontology merging filed deserve attention, since:

- the binary merge approaches are not scalable enough,
- the merge approaches need to adjust user merge requirements,
- the merged ontology should be free of inconsistencies, and
- the quality of the merged ontology should be evaluated systematically.

We contribute to the issues stated above in Chapters 4-7. In particular:

- We have proposed a partitioning-based method to merge multiple ontologies with an n-ary strategy. The approach seeks to partition the n source ontologies into k blocks ($k << n$), then merges and refines the individual blocks, and finally combines and globally refines blocks to generate the merged ontology.
  - The partitioning process follows a structural similarity strategy with low computational complexity. The approach utilizes a set of pivot classes to accelerates the partitioning process and assigns the classes in the blocks based on their structural similarity, which no similarity membership function is required. Indeed, the adjacent classes (on the hierarchy level of the source ontologies) with their corresponding classes are located in the same blocks. This increases the intra-block similarity of blocks and decreases the inter-block similarity. In this way, the n source ontologies are partitioned into the k blocks, which leads to having smaller and more tractable merging subtasks (concerning *H2*).

  - As we have shown in Section 4.6, the blocks have been combined via intra- and inter-combining processes. Local and global refinements have been carried to fulfill the user-selected requirements (satisfying hypothesis *H4*).
  - We used the ontologies from OAEI benchmark and BioPortal. We have merged multiple ontologies up to 56 source ontologies, including different axioms' size ($134 \leq |Sig(\mathcal{O}_S)| \leq 30364$). We have tested the method by generating six different merged ontologies of applying local or global refinements and considering perfect or imperfect mappings and analyzed the effect of such variables.
  - We conducted a series of experiments on the binary and n-ary merge strategies intended to demonstrate that same or better quality result in terms of merged characteristics and answering Competency Questions (Section 10.2.1 and Section 10.2.2) and scalability in terms of operational complexity and time performance (Section 10.2.3) of the n-ary method. This proof-of-concept satisfied the thesis' hypothesis (*H1*) that the n-ary merge method is prior to the binary merge strategies.

- We have identified and provided a comprehensive list of Generic Merge Requirements (GMRs) by analyzing three fields in the literature: ontology merging approaches, ontology merging benchmark, and ontology engineering. We adapted them in the context of the ontology merging domain. In addition, we classified them into integrity, model, and logic properties dimensions.
  - Users can select a set of GMRs during the merge process. Since all GMRs are not compatible together, users can ask our system to suggest a set of possible GMRs according to their choices. We utilized a graph-based framework to determine the compatibilities between the user-selected GMRs. We also established a method to rank the suggested compatible sets and sort them.
  - With a use case study in Section 11.1, we have shown that given a set of user-selected GMRs, there is a superset of compatible GMRs that can be fulfilled simultaneously (satisfying hypothesis *H3*).
  - We also had an insight into the conflict arising by one type restriction and property values' constraint. We proposed a method to build a subsumption hierarchy on data types and performing substitution or instantiation on them to tackle one type conflict. We used the least upper and greatest lower bound method to reconcile the cardinality restriction and utilized the semantic relatedness to deal with value restriction conflicts.

- Concerning hypothesis *H5*, we tackled the inconsistencies of the merged ontology with respect to the knowledge of the source ontologies in Chapter 6.
  - We utilized the Subjective Logic theory to find the trustworthy of the axioms that arise inconsistencies. Subjective Logic can captures opinions (trustworthy) about the world (in our context, the conflicting axioms) in belief models and provides a set of operations for combining opinions (of the source ontologies).

  - When there is a dependency between the conflicting axioms of several justification sets, we used the adapted version of the conditional operation in Subjective Logic.
  - The proposed method is implemented in the *CoMerger*, which it can automatically rank the conflicting axioms and suggest a set of axioms to the users to be revised. The suggested plan can be applied automatically, or a user can revise them before applying.
  - We have shown in Section 12.2, the inconsistent merged ontologies which became to a consistent one by our approach are competitive with the one which became consistent by the human intervention (in several cases achieved a same or better result in the Competency Questions tests), which demonstrated the reliability of our method.

- In Chapter 7, we proposed a comprehensive framework to assess the quality of the merged ontology by structural, functional, and usability dimensions (satisfying hypothesis *H6*).
  - We defined each dimension with a set of aspects and quality indicators summarized in Table 7.3. We formulated the structural measures via consistency test and General Merge Requirement (GMR)s. We defined the systematic formulation to evaluate the functional measures against the intended use and semantics of the merged ontology and provided criteria for the usability dimension.
  - We have shown an empirical analysis on each dimension in Section 13.1, Sections 13.2, and Section 13.3.
  - Our framework has been adapted by the prior works [GCCL05; DRFBSAG+11] in the context of evaluating the merged ontology, in which the quality of the merged ontology is evaluated by concerning the knowledge of the source ontologies.
  - The proposed criteria have been implemented in *CoMerger*, which can be evaluated independently of the merge process.

- We have developed *CoMerger* (Chapter 9), a web-based application which comprised all methods stated above. *CoMerger* is an open-source tool and can be installed locally, too.

Our proposed approaches have their limitations. The GMRs compatibility checker framework is conservative and finds potential conflicts independent of given ontologies. Moreover, all GMRs that we have implemented in the *CoMerger* are not fully automatic and require the user intervention, such as *R16* or *R20* (see Appendix B). Furthermore, although we have proposed the conflict solution strategies for between some GMRs, the full resolution for all is hard to be achieved. For example, the conflict between the *R2* (property preservation) and *R16* (class acyclicity), any possible solution to compromise them synonymously cannot fully resolve their conflict. Another limitation of our approach is the time performance of the inconsistency handling when the ontologies are large. Besides, the evaluation of the functional dimension is narrowed to converting

the given Competency Questions to the SPARQL queries manually. Overcoming this limitation is out of the defined scope of the thesis hypothesis.

15 Future Work

In this chapter, we examine a set of important directives for future work. The goal of this thesis was to develop an efficient, n-ary ontology merge method that scales for multiple ontologies. The merge method can be customized for user requirements. Moreover, we took a step towards checking the inconsistency and quality of the merged result. We built *CoMerger* that implements all our proposed methods. In this thesis, we focused on the core elements of each aspect (concerning our hypothesis) and evaluated the feasibility of our proposed methods. We expect that the contributions of this thesis can be extended and improved in several directions. In the rest of this chapter, we briefly identify possible future directions.

Merging Multiple Ontologies with an N-ary Strategy: We provided a parameterizable merge platform allowing users to influence the merge result interactively through the Generic Merge Requirements (GMRs). A future research agenda in this direction is finding strategies for such user interaction in the context of an application or a use-case. Moreover, adapting our approach to merging data on the schema-level of Linked Open Data (LOD) scenarios is another future research direction. Furthermore, taking advantage of the parallelization potential of the approach could represent an interesting topic of research.

Utilizing User-Driven Generic Merge Requirements: The Generic Merge Requirements (GMRs) within our systems can be easily adapted for new GMRs, in which their compatibilities can be detected through our framework. Not all potential GMRs conflicts will materialize in each concrete merged ontology. Investigation of how the approach can be extended to take this into account is left for future work. A further future plan is a study about the extent to which the users agree with the ranked suggested sets. In addition, the study of crucial GMRs in a specific domain could be a potential future direction.

Inconsistency Handling of the Merged Ontology: The future work of this aspect could be on studying the exploitation of domain knowledge. We believe using more

trustable background knowledge such as MeSH[1] [Lip00] and SNOMED-CT[2] [Don06] (as domain-specific knowledge resources in the biomedical domain) or WordNet [Mil98] (as a generic knowledge resource) might significantly improve the opinion's probability expectation value.

Assessing the Quality of the Merged Ontology: Evaluating the merged ontology in the functional dimension has been carried out by a set of user-given Competency Questions, where human intervention is required to convert them to the SPARQL queries. The possibility of evaluating this dimension automatically would help many use-case scenarios. This precedes to integrating an automatic conversion from natural language to the SPARQL-queries, such as the works in [NNBU+13; DDS+16]. Furthermore, evaluating the source ontologies before the merge process might give useful insights to the users. Moreover, analyzing conflicts between licenses in our framework is left open for future work. Despite efforts in other domains [HSP+19; GLR+14; Gor11], to the best of our knowledge, there is no study on the license compatibility checker of the ontologies. Thus, it would open the door for the compatibility checker of the source ontologies in the ontology merging scenarios.

***CoMerger* tool:** The tool could be extended with respect to several dimensions. First, other existing ontology matchers can be embedded in our open-source tool so that users can choose their desired way of generating the mapping between the source ontologies. Second, providing an integration plug-in with popular ontology editor tools such as Protégé [NSD+01] might be useful in practice. The further future plan can be the evaluation of the ease-of-use through a user study.

[1] Medical Subject Headings
[2] Systematised Nomenclature of Medicine, Clinical Terms

Part V

Appendix

A

Competency Questions on Conference domain

Table A.1 shows a set of Competency Questions (CQs) for the conference domain of the OAEI benchmark used for our experimental tests. We drive these CQs based on the information provided by *cmt*, *conference*, *confOf*, *edas*, *ekaw*, *iasted*, *sigkdd* ontologies from the conference domain.

Table A.2 shows a set of Competency Questions (CQs) used in the experimental test of conflict resolution.

TABLE A.1: The Competency Questions (CQs) for the conference domain of the OAEI benchmark. CQ_1-CQ_{30} are positive CQs, while CQ_{31}-CQ_{35} are negative CQs.

No.	Competency Questions
	Positive CQs
CQ_1	Which type of documents can include in the conference document?
CQ_2	Who can participate in the conference?
CQ_3	Can "Reviewer" be the type of person in the conference domain?
CQ_4	What is the role of the conference participant at the conference?
CQ_5	Can a publisher be related to the conference domain?
CQ_6	Who issues the conference proceeding?
CQ_7	Who assigns the external reviewer?
CQ_8	Can the conference proceeding have an ISBN?
CQ_9	Can an author be a conference member?
CQ_{10}	Which type of contribution exists in the conference domain?
CQ_{11}	Which type of fee exists?
CQ_{12}	Which events can exist at the conference?
CQ_{13}	Which social events can a conference have?
CQ_{14}	Which organization can be at the conference?
CQ_{15}	Who assigns the reviewer?
CQ_{16}	Can a poster be part of the contribution to the conference?
CQ_{17}	Which type of paper can exist in the conference document?
CQ_{18}	Can the tutorial be part of the conference?
CQ_{19}	Which countries can participate?
CQ_{20}	What can be the decision for a submitted paper to the conference?
CQ_{21}	Who can be a conference member?
CQ_{22}	Which type of conference committee exists?
CQ_{23}	Can information for the participant be part of the conference document?
CQ_{24}	Can a workshop be part of the conference?
CQ_{25}	Can a short paper be a contribution to the conference?
CQ_{26}	Which type of session exists in the conference?
CQ_{27}	Does the conference has a demo session?
CQ_{28}	Which audiovisual_equipment conference has?
CQ_{29}	Which type of deadline exists in the conference?
CQ_{30}	Which type of award can be given at the conference?
	Negative CQs
CQ_{31}	How should discussions on the poster held?
CQ_{32}	Is there any information regarding the past venue of the conference?
CQ_{33}	Is there any relation between an author and a company
CQ_{34}	Do authors have a CV and accessible during the conference presentation?
CQ_{35}	Is there any visa information available for the participant?

TABLE A.2: The Competency Questions (CQs) used in conflict resolution test.

No.	**Competency Questions**
CQ_1	Does a person have an email?
CQ_2	Who can have an email?
CQ_3	How many emails can a person have?
CQ_4	Does a conference have a name?
CQ_5	Who can have a name?
CQ_6	How many names can a conference have?
CQ_7	Has a review only a reviewer as an author?
CQ_8	By whom is a review written by?
CQ_9	What can have a title?
CQ_{10}	Which range does the title have?

B

GMR Implementation

In this appendix, we show the prototype implemented in each GMR. In particular, we explain:

- how we determine that a GMR is not fulfilled in merged ontologies, and
- how the unfulfilled GMR can be repaired.

We assume an ontology $\mathcal{O}$ contains a set of entities $\mathcal{E}$ including classes C, properties P, and instances I. The full list of used notations of this appendix has been shown in Table B.1.

TABLE B.1: The used notations and symbols in Appendix B.

Notation	Description
$\mathcal{O}_S$	source ontologies
$\mathcal{O}_M$	merged ontology
c_j	a class
c_j^T	a parent of a class
c_j^D	a child of a class
p_j	a property
c_j^p	a respective class (domain/range) of a property
I_j	an instance
c_j^I	the respective class of an instance
e_j	an entity
$\mathcal{E}$	all entities
X	all axioms
α	an axiom

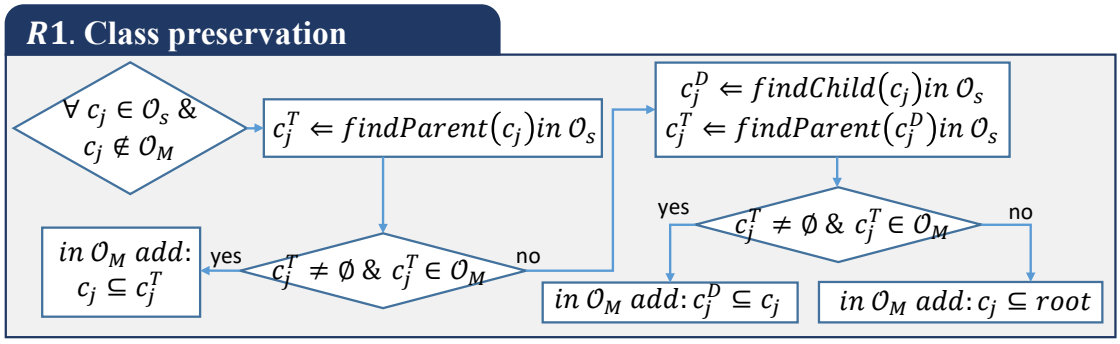


FIGURE B.1: *R1*- Repair solution.

***R1*. Class preservation:**

- **Detecting *R1***: We check whether all classes (or their mapped classes) from the source ontologies exist in the merged ontology. If no, we mark them as missing classes.
- **Repairing *R1***: Any missed class (c_j) from source ontologies, i.e., $c_j \in \mathcal{O}_S$ but $c_j \notin \mathcal{O}_M$, should be added to the merged ontology. To perform this process:
  1. One parent (c_j^T) of this class in $\mathcal{O}_S$ should be found.
  2. If there exists a parent for it ($c_j^T \neq \emptyset$) and if this parent already exists in the merged ontology ($c_j^T \in \mathcal{O}_M$), this class is added as a child of its detected parent.
  3. If there is no parent for it ($c_j^T = \emptyset$), or the parent c_j^T does not exist in $\mathcal{O}_M$, then repeat this process by considering the child of c_j, i.e., one child of the missing class (c_j^D) should be found. If it exists in $\mathcal{O}_M$, c_j^D is added as a parent of c_j.
  4. Otherwise, it should be added to the root.

  Figure B.1 shows the repair process of *R1*.

***R2*. Property preservation:**

- **Detecting *R2*:** We check whether all properties (or their mapped properties) from the source ontologies exist in the merged ontology. If no, we mark them as missing properties.
- **Repairing *R2*:** Any missed property (p_j) from source ontologies, i.e., $p_j \in \mathcal{O}_S$ but $p_j \notin \mathcal{O}_M$, should be added to the merged ontology. To perform this process:
  1. The respective class (c_j^p) for the missing property from $\mathcal{O}_S$ should be found. It can be a domain or range of that property. Note that, domains or ranges of the properties are the type of class.
  2. If c_j^p exists in the merged ontology, we add in $\mathcal{O}_M$: $c_j^P\ hasProperty\ p_j$.
  3. If c_j^p does not exist in the merged ontology, the property p_j cannot be added to the merged ontology. This situation will be warned to the user.

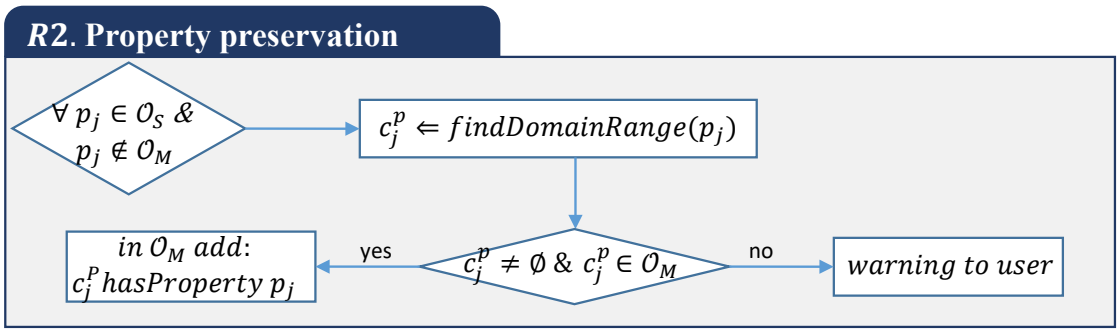


FIGURE B.2: *R2*- Repair solution.

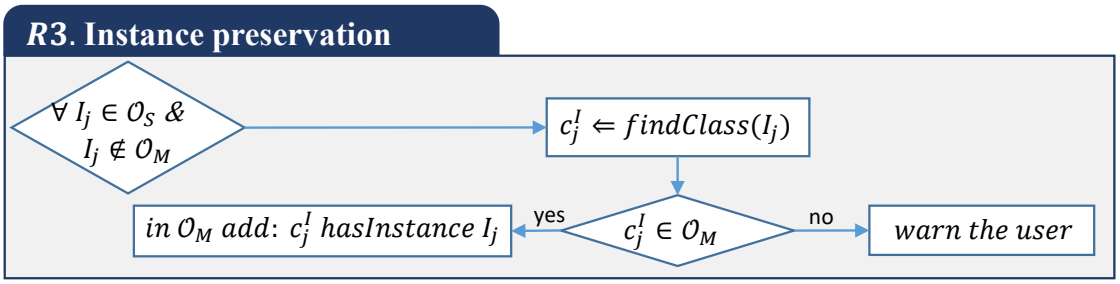


FIGURE B.3: *R3*- Repair solution.

Figure B.2 shows the repair process of *R2*. This process carries for all types of properties such as objects and data properties.

***R3*. Instance preservation:**

- **Detecting *R3*:** We check whether all instances from the source ontologies exist in the merged ontology. If no, we mark them as missing instances.
- **Repairing *R3*:** Any missed instance (I_j) from the source ontologies, i.e., $I_j \in \mathcal{O}_S$ but $I_j \notin \mathcal{O}_M$, should be added to the merged ontology. To perform this process:
  1. The respective class c_j^I of the missing instance I_j should be found.
  2. If c_j^I exists in the merged ontology, the instance (I_j) is added to its detected class (c_j^I).
  3. If c_j^I does not exist, we warn to the user that this instance could not be added to the merged ontology.

  Figure B.3 shows the repair process of *R3*.

***R4*. Correspondence preservation:**

- **Detecting *R4*:** We check whether all corresponding entities are integrated into one entity in the merged ontology or not. If no, we mark them.
- **Repairing *R4*:** For those entities which have some correspondences, but they did not merge into one entity, we combine them in an integrated entity. We add this

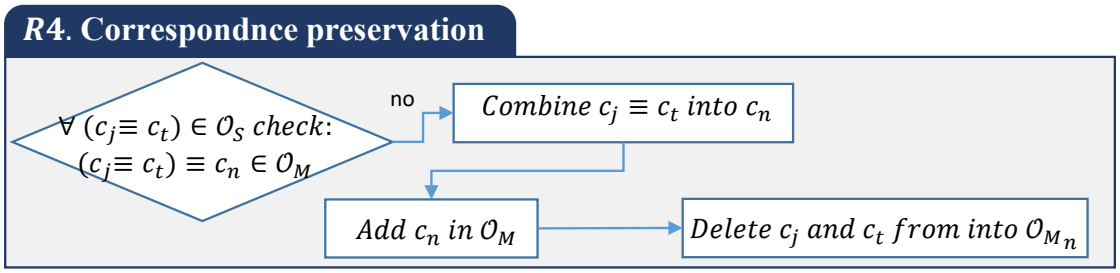


FIGURE B.4: *R4*- Repair solution.

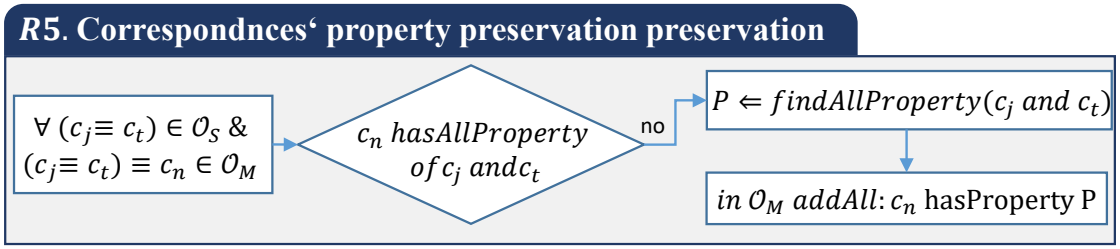


FIGURE B.5: *R5*- Repair solution.

new entity to the merged ontology, then delete those entities from the merged ontology. Figure B.4 shows the repair process of *R4*.

***R5*. Correspondence's property preservation:**

- **Detecting *R5*:** For all corresponding classes that they merged into an integrated entity in $\mathcal{O}_M$, we check whether this integrated entity has all properties of its corresponding entities. If no, we mark them.
- **Repairing *R5*:** For those marked entities, we add the properties of the corresponding entities to the integrated entity in $\mathcal{O}_M$. Figure B.5 shows the repair process of *R5*.

***R6*. Value preservation:**

- **Detecting *R6*:** For all corresponding entities with two different values, we check whether their integrated entity has both values. Moreover, if both values have a conflict, we mark them.
- **Repairing *R6*:** If an integrated entity does not have both values of its corresponding entities, we set both values for the integrated one. However, if their values have a conflict with each other, we need user interaction to solve it. Figure B.6 shows the repair process of *R6*.

***R7*. Structure preservation:**

- **Detecting *R7*:** In the merged ontology, we check whether each class has the same ancestor as the source ontologies. If no, we mark them.

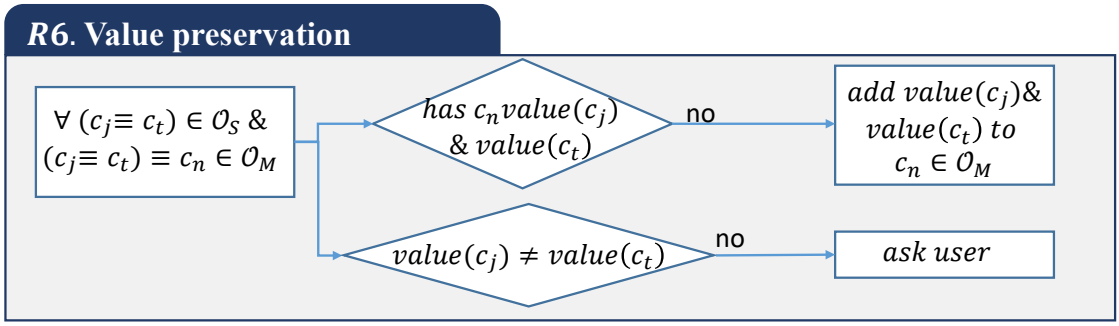


FIGURE B.6: *R6*- Repair solution.

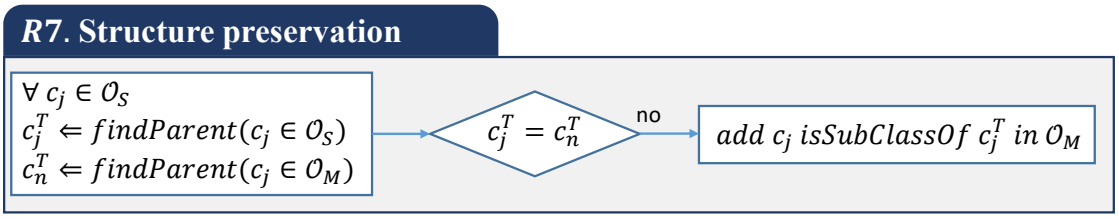


FIGURE B.7: *R7*- Repair solution.

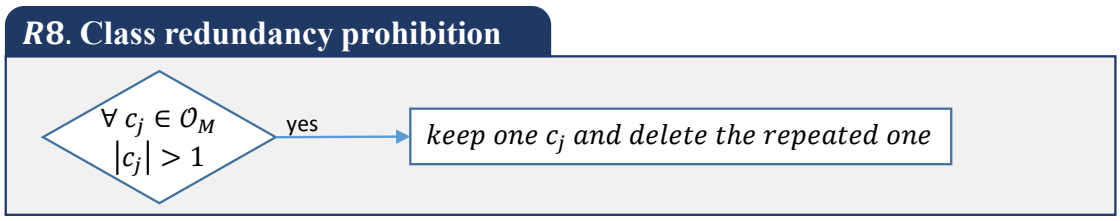


FIGURE B.8: *R8*- Repair solution.

- **Repairing *R7*:** For any marked class c_j, we add a new is-a relationship from class c_j to its respective parent (belong to $\mathcal{O}_S$). This process is carried, only if the parent of c_j exists in $\mathcal{O}_M$. Figure B.7 shows the repair process of *R7*.

***R8*. Class redundancy prohibition:**

- **Detecting *R8*:** If there is any class c_j, which is redundant (duplicated) in the merged ontology, we mark it.
- **Repairing *R8*:** For any marked class c_j, we keep one of them and delete the repeated one. Figure B.8 shows the repair process of *R8*.

***R9*. Property redundancy prohibition:**

- **Detecting *R9*:** If there is any property p_j, which is redundant (duplicated) in the merged ontology, we mark it.
- **Repairing *R9*:** For any marked property p_j, we keep one of them and delete the repeated one. Figure B.9 shows the repair process of *R9*.

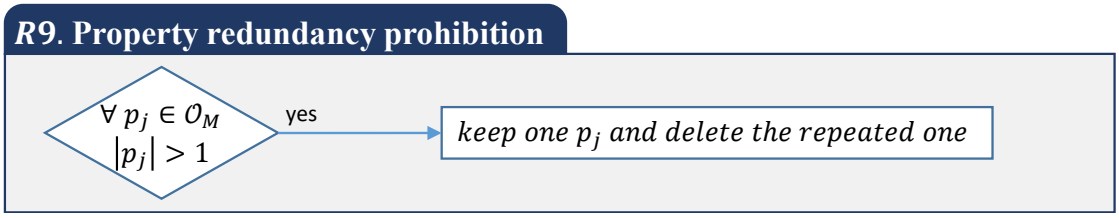


FIGURE B.9: *R9*- Repair solution.

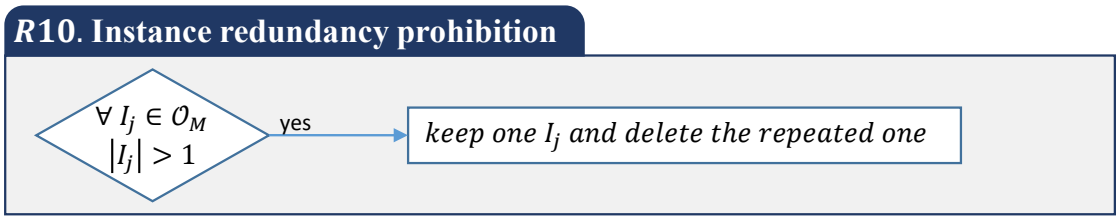


FIGURE B.10: *R10*- Repair solution.

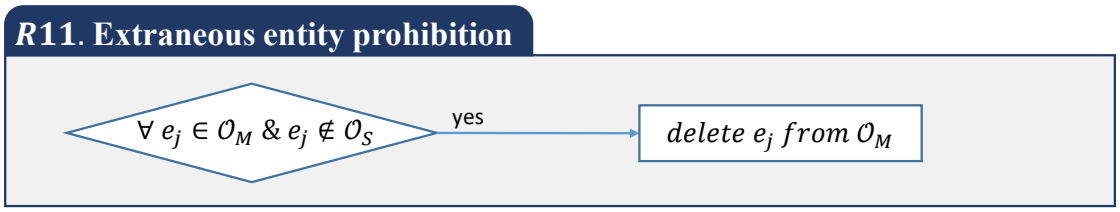


FIGURE B.11: *R11*- Repair solution.

***R10*. Instance redundancy prohibition:**

- **Detecting *R10*:** If there is any instance I_j, which is redundant (duplicated) in the merged ontology, we mark it.
- **Repairing *R10*:** For any marked instance I_j, we keep one of them and delete the repeated one. Figure B.10 shows the repair process of *R10*.

***R11*. Extraneous entities prohibition:**

- **Detecting *R11*:** For all entities belong to $\mathcal{O}_M$, we check whether they exist in $\mathcal{O}_S$. If no, we mark them.
- **Repairing *R11*:** Any extra marked entity is deleted from $\mathcal{O}_M$. Figure B.11 shows the repair process of *R11*.

***R12*. Entailments deduction satisfaction:**

- **Detecting *R12*:** This is related to subsumption and equivalence entailments. For both, we follow the same process. First, we get subclass and equivalence axioms from the source ontologies. Then, we ask a reasoner to check whether the merged ontology can entail those axioms. If no, we mark them.

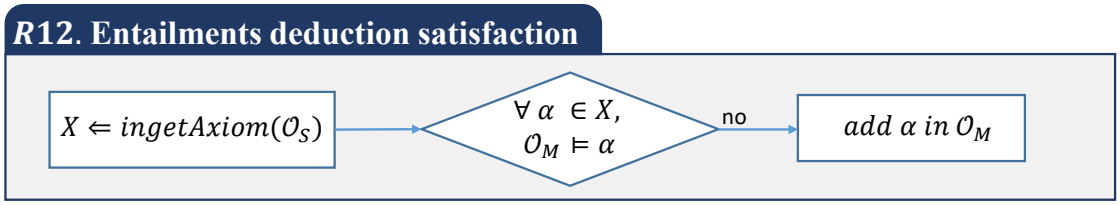


FIGURE B.12: *R12*- Repair solution.

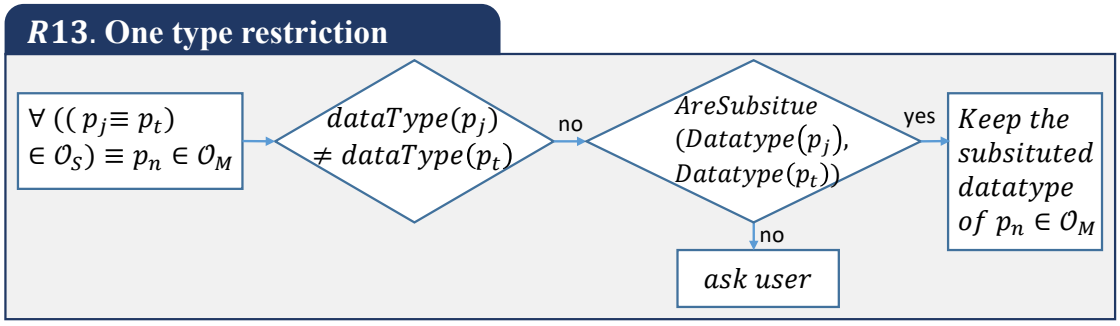


FIGURE B.13: *R13*- Repair solution.

- **Repairing *R12*:** We add those not-entailed axioms in $\mathcal{O}_M$. Figure B.12 shows the repair process of *R12*.

***R13*. One type restriction:**

- **Detecting *R13*:** We check for each integrated data type property in the merged ontology $(((p_j \equiv p_t) \in \mathcal{O}_S) \equiv p_n \in \mathcal{O}_M)$, whether they have the same datatype properties. If in the source ontologies, they have different values $(dataType(p_j) \neq dataType(p_t))$, the new integrated one p_n, cannot have both types at the same time. So, we mark it.
- **Repairing *R13*:** For any marked property, we check if both types are homogenous together, we only keep the substitute one. e.g., CHAR and STRING are homogenous together and we keep only the more general type, i.e., type STRING for p_n. If no, we ask the user. Figure B.13 shows the repair process of *R13*.

***R14*. Property value's constraint:**

- **Detecting *R14*:** We check all following constraint types:

  ObjectMaxCardinality, ObjectMinCardinality, ObjectExactCardinality, DataMaxCardinality, DataMinCardinality, DataExactCardinality, ObjectSomeValuesFrom, ObjectAllValuesFrom.

  For all entities belonging to source ontologies, we check the value of property of each constraint type, then we check the value of its mapped entity in the merged ontology. If they have different values, we mark them.
- **Repairing *R14*:** For any marked entity e_j, we keep the substitute value in $\mathcal{O}_M$. Figure B.14 shows the repair process of *R14*.

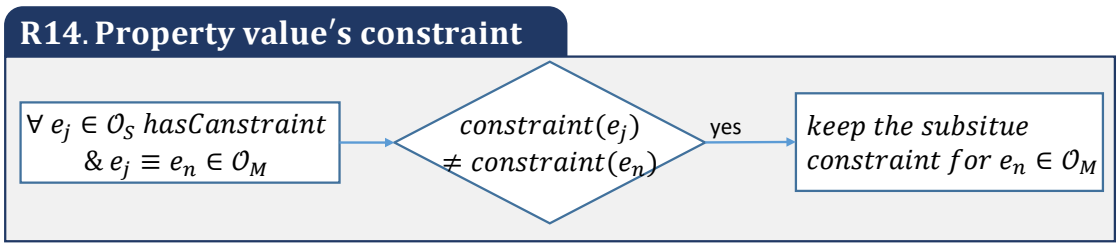


FIGURE B.14: *R14*- Repair solution.

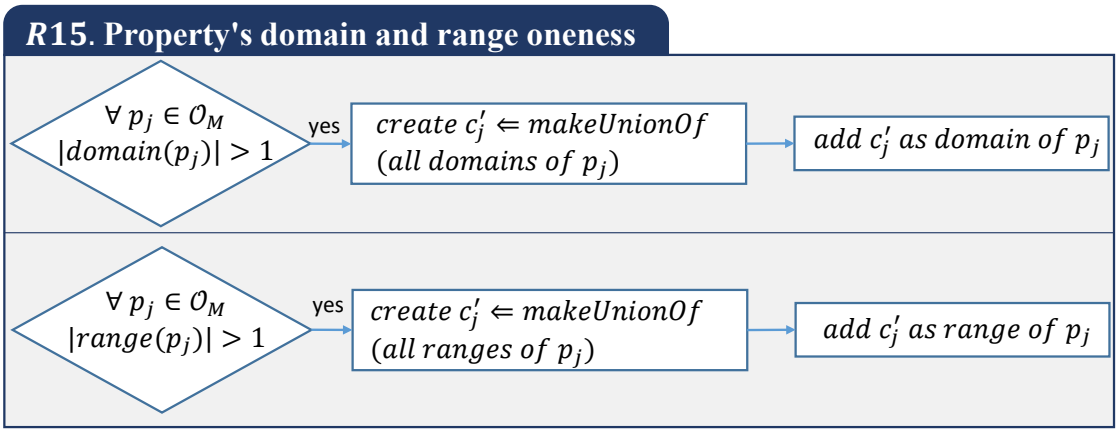


FIGURE B.15: *R15*- Repair solution.

***R15*. Property's domain and range oneness:**

- **Detecting *R15*:** If a property p_j in the merged ontology has multiple domains or ranges ($|domainRange(p_j)| > 1$), we mark it.
- **Repairing *R15*:** For any marked property p_j, we create a new class as the union of all its domains or ranges. We then add this new class as domain/range of property p_j. Figure B.15 shows the repair process of *R15*.

***R16*. Acyclicity in the class hierarchy:**

- **Detecting *R16*:** There are two types of cycles:
  - Self-cycle: To detect the self-cycle, we check for any class c_j, this class should not appear to the list of its parents. If so, we mark it.
  - Recursive-cycle: During the visiting of all parents of a class, if we visit more than one time a parent, we mark it as a cycle.
- **Repairing *R16*:** We delete the respective axiom that caused a self-cycle in the merged ontology. To repair the recursive-cycle, we need user interaction. Figure B.16 shows the repair process of *R16*.

***R17*. Acyclicity in the property hierarchy:**

- **Detecting *R17*:** There are two types of cycles in the property hierarchy:

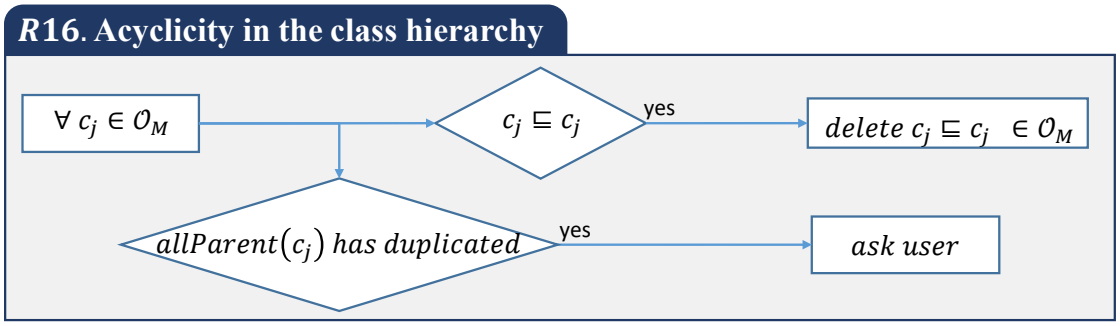


FIGURE B.16: *R16*- Repair solution.

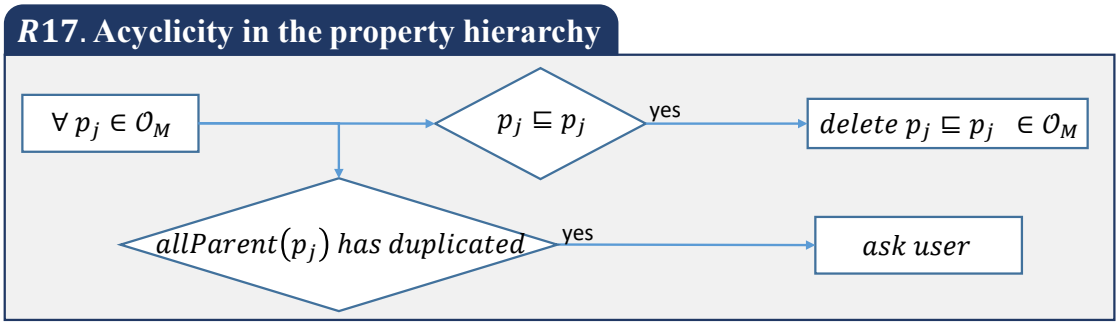


FIGURE B.17: *R17*- Repair solution.

  - Self-cycle: To detect this type of cycle, we check for any property p_j, this property should not appear as its subPropertyOf. If so, we mark it.
  - Recursive-cycle: During the visiting subPropertyOf hierarchy for property p_j, if we visit more than one time a property, we mark it as a cycle.

- **Repairing *R17*:** We delete the respective axiom that caused a self-cycle on the property hierarchy in the merged ontology. To repair the recursive-cycle, we need user interaction. Figure B.17 shows the repair process of *R17*.

***R18*. Prohibition of properties being inverses of themselves:**

- **Detecting *R18*:** We check whether in the merged ontology, there is a property that is inverse of itself. If so, we mark it.
- **Repairing *R18*:** We delete the related inverseOf axiom of the marked property in the merged ontology. Figure B.18 shows the repair process of *R18*.

***R19*. Unconnected class prohibition:**

- **Detecting *R19*:** If there is any class (c_j) which does not have any connections to the other classes in the is-a hierarchy, i.e. $SubClass(c_j)\&SuperClass(c_j) = \emptyset$, we mark it.
- **Repairing *R19*:** The repair process includes:
  1. One of sub or superclass of c_j from $\mathcal{O}_S$, called c_j^T should be found.

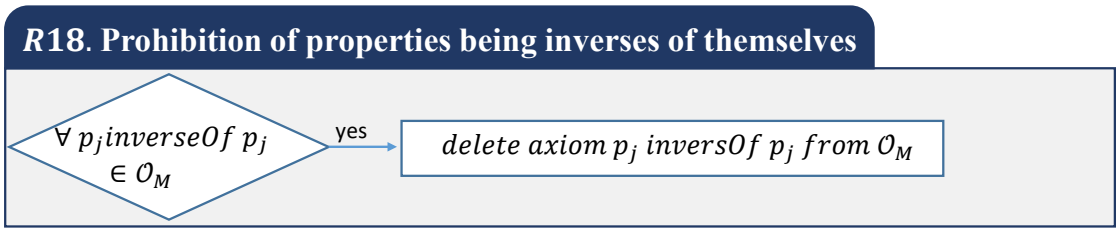


FIGURE B.18: *R18*- Repair solution.

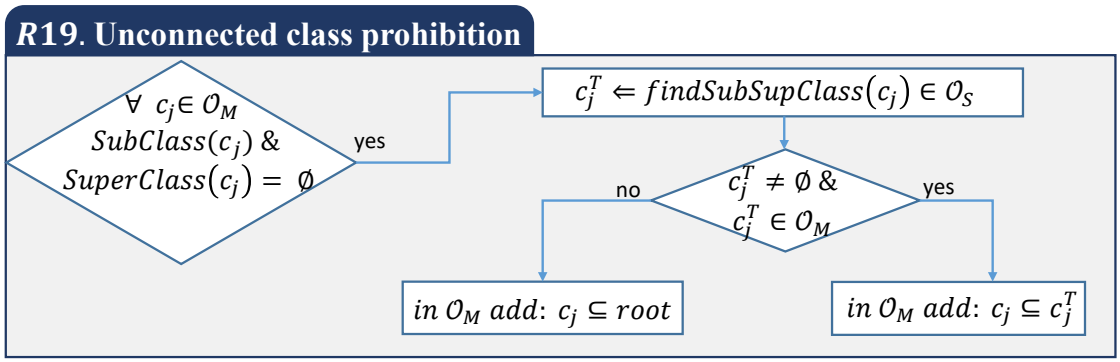


FIGURE B.19: *R19*- Repair solution.

2. If c_j^T is not null and exists in the merged ontology, we add c_j to c_j^T with an is-a relationship.
3. Otherwise, we add c_j to the root of the merged ontology.

Figure B.19 shows the repair process of *R19*.

***R20*. Unconnected property prohibition:**

- **Detecting *R20*:** If there is any property p_j which does not have any connections to the other properties in the subPropertyOf hierarchy, we mark it.
- **Repairing *R20*:** The repair process includes:
  1. One of sub or super property of p_j from $\mathcal{O}_S$, called p_j^T should be found.
  2. If p_j^T exists in the merged ontology, we add p_j to p_j^T with a subPropertyOf relationship.
  3. Otherwise, ask the user.

Figure B.20 shows the repair process of *R20*.

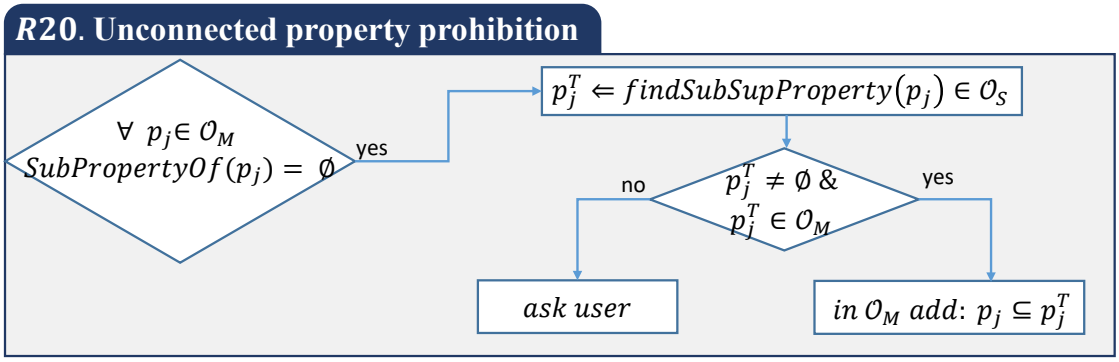


FIGURE B.20: *R20*- Repair solution.

C

User Help

In this appendix, we give a practical guideline about the user interaction with the *CoMerger* tool. In particular, we present:

1. a guideline to merging ontologies in Section C.1,
2. an instruction to assess the quality of the merged ontology in Section C.2),
3. required steps to verifying the consistency of the merged result in Section C.3),
4. necessary processes of checking the compatibility of the selected Generic Merge Requirements (GMR)s in Section C.4),
5. an instruction of comparing ontologies through the SPARQL endpoint in Section C.5).

C.1 Merging Ontologies

Figure C.1 shows the GUI of the ontology merging process in *CoMerger* tool. First, the MERGER tab should be selected. After that, the required input parameters should be adjusted. To this end, a set of source ontologies and their mappings should be uploaded. If no mapping is available for them, the system can generate them automatically. However, the user should determine the type of matcher and the format of the output. Moreover, the user can adjust a set of refinements and evaluation criteria to be applied to the merged ontology. Note that, there is a possibility to perform them after the merge process as well. Finally, with pressing the MERGE button, the merged ontology will be generated.

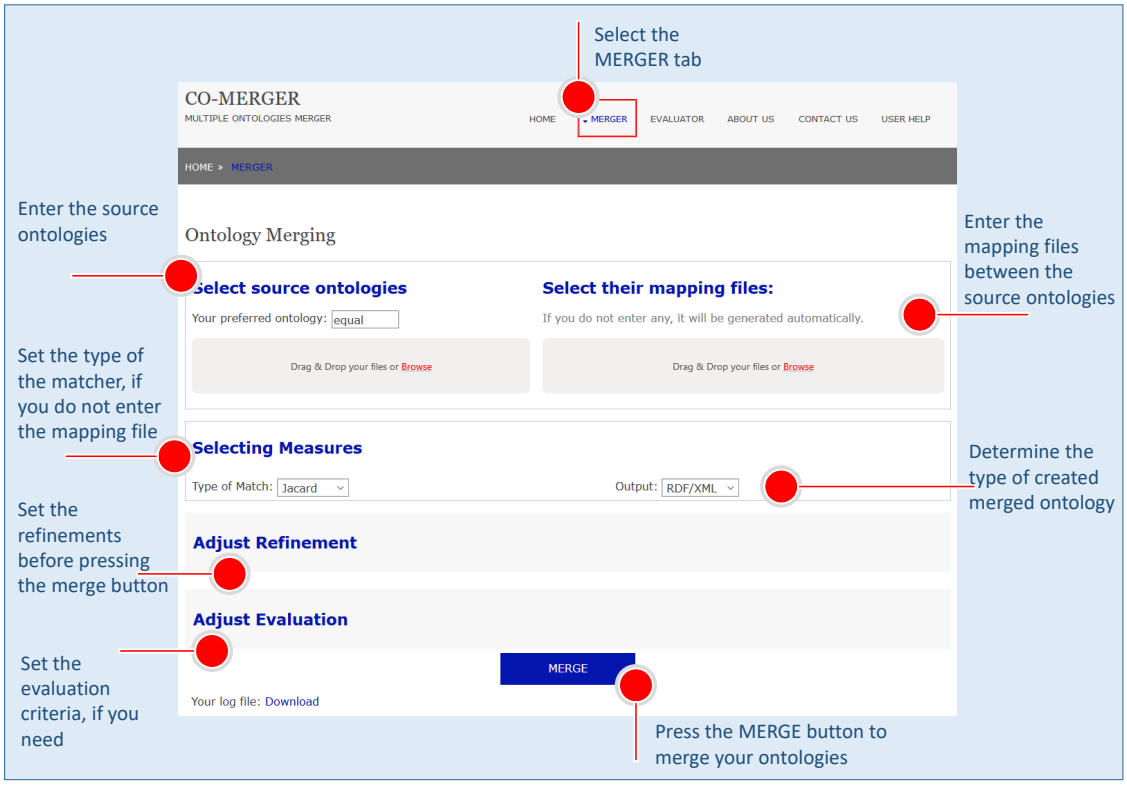


FIGURE C.1: Merger GUI.

After performing the merge process, the system guides the user to the result page, as shown in Figure C.2. In this page, the merged ontology, the created sub-ontologies, the log information on each step of refinements, and the evaluation result can be download.

Moreover, at this stage, the user is able to apply the refinement or evaluation, if they did not perform already. Besides, the user can run the consistency test, or the query test on the respective merged ontology.

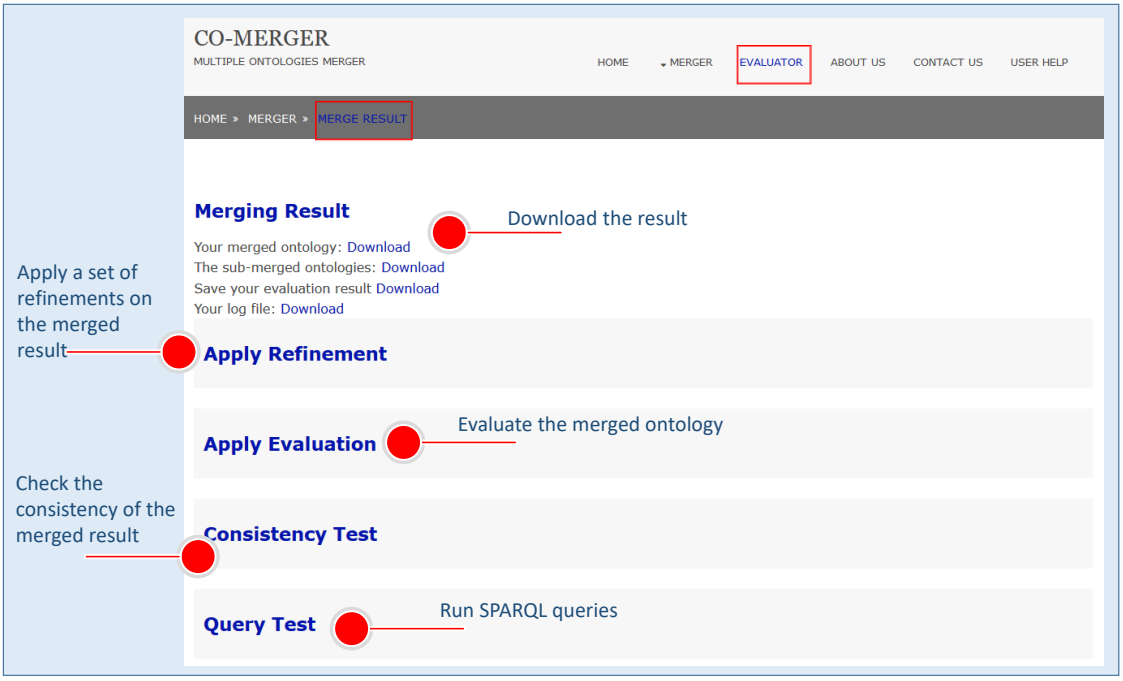


FIGURE C.2: Merging Result GUI.

C.2 Quality Assessment

Figure C.3 shows the setting for the evaluation criteria. The user can select or deselect all criteria once or (de-)select them individually.

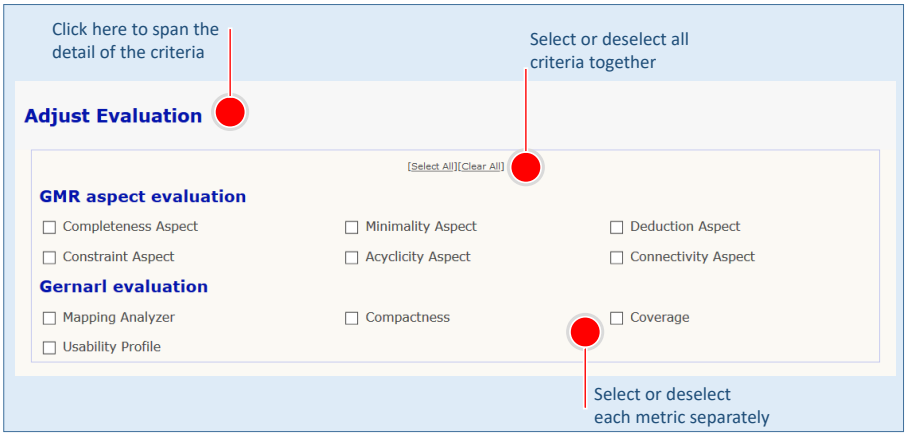


FIGURE C.3: Setting of the evaluation criteria.

Figure C.4 presents the evaluation results. It includes the overall result representation of the selected evaluation aspects. Moreover, the detailed analysis of the selected aspect will be shown to the user.

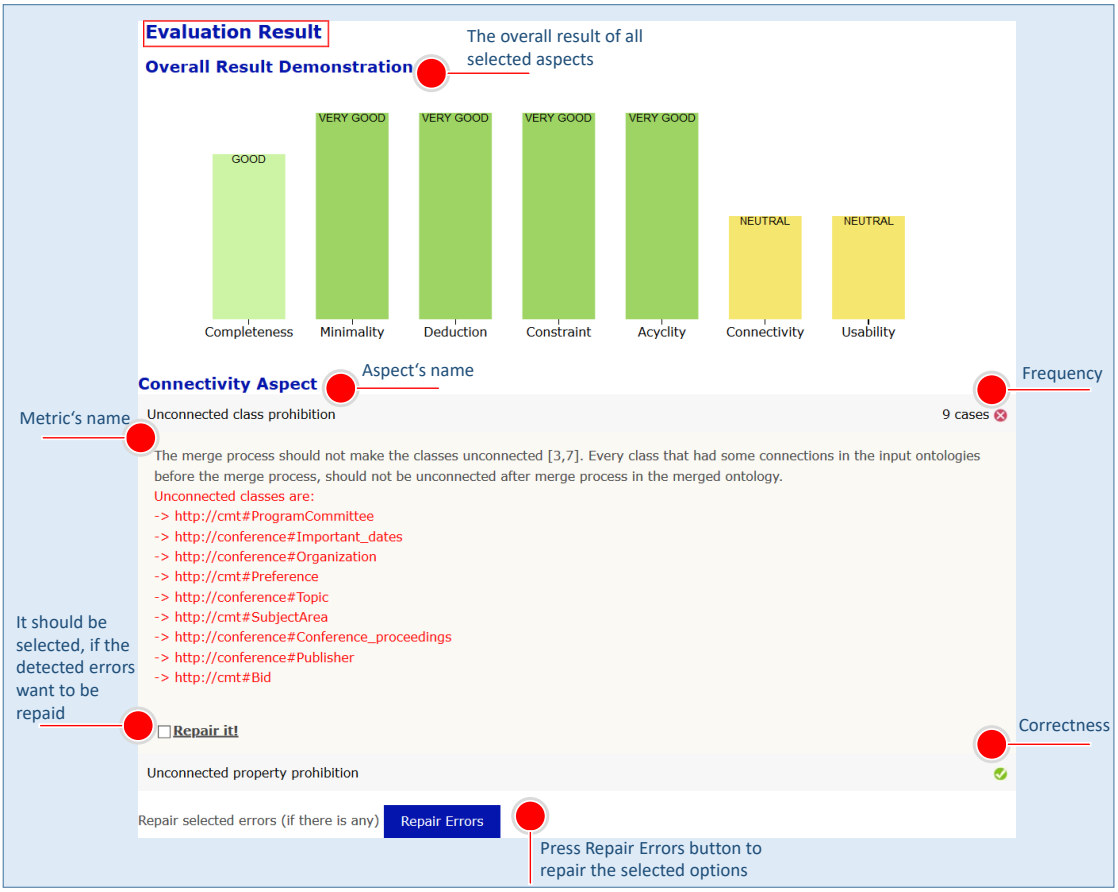


FIGURE C.4: Evaluation results.

Users are able to evaluate the quality of any given merged ontology directly. It does not require to run the merge process beforehand. To this end, through the *Evaluator* tab, the quality of a merged ontology can directly be assessed, as shown in Figure C.5. The merged ontology and respective source ontologies, along with their mapping (if available) should be uploaded. The user can evaluate the merged result by adjusting the set of criteria, run a consistency test directly, or perform a query test.

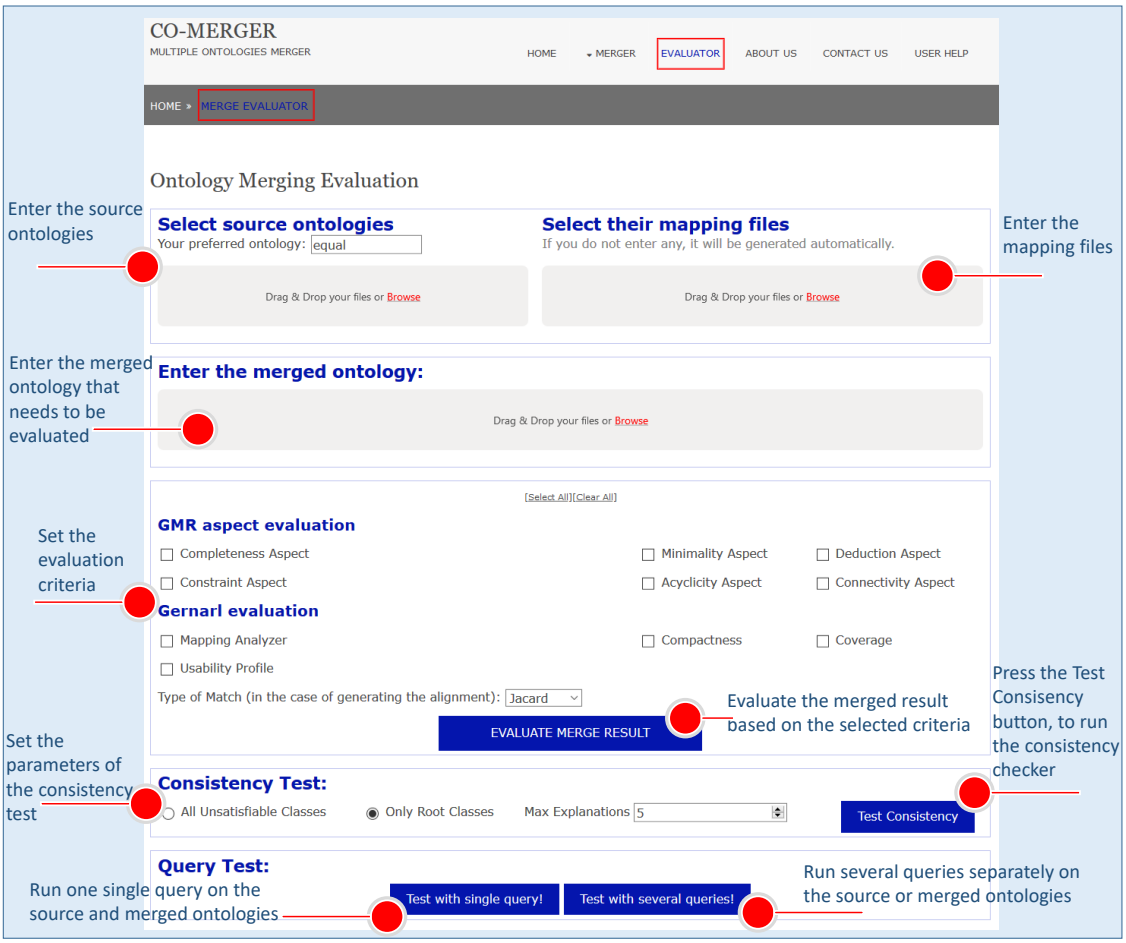


FIGURE C.5: Evaluator GUI.

C.3 Consistency Checker

Figure C.6 shows the parameter setting in order to perform the consistency test. Figure C.7 presents the result of the consistency test.

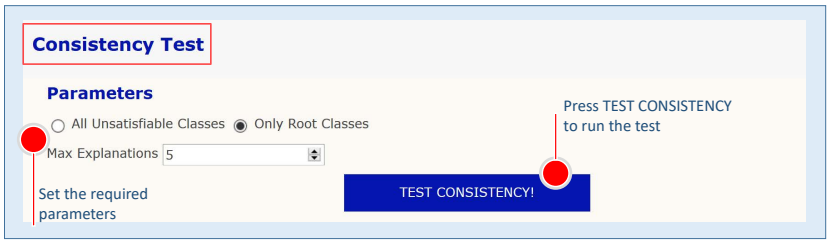


FIGURE C.6: Parameter setting of consistency test.

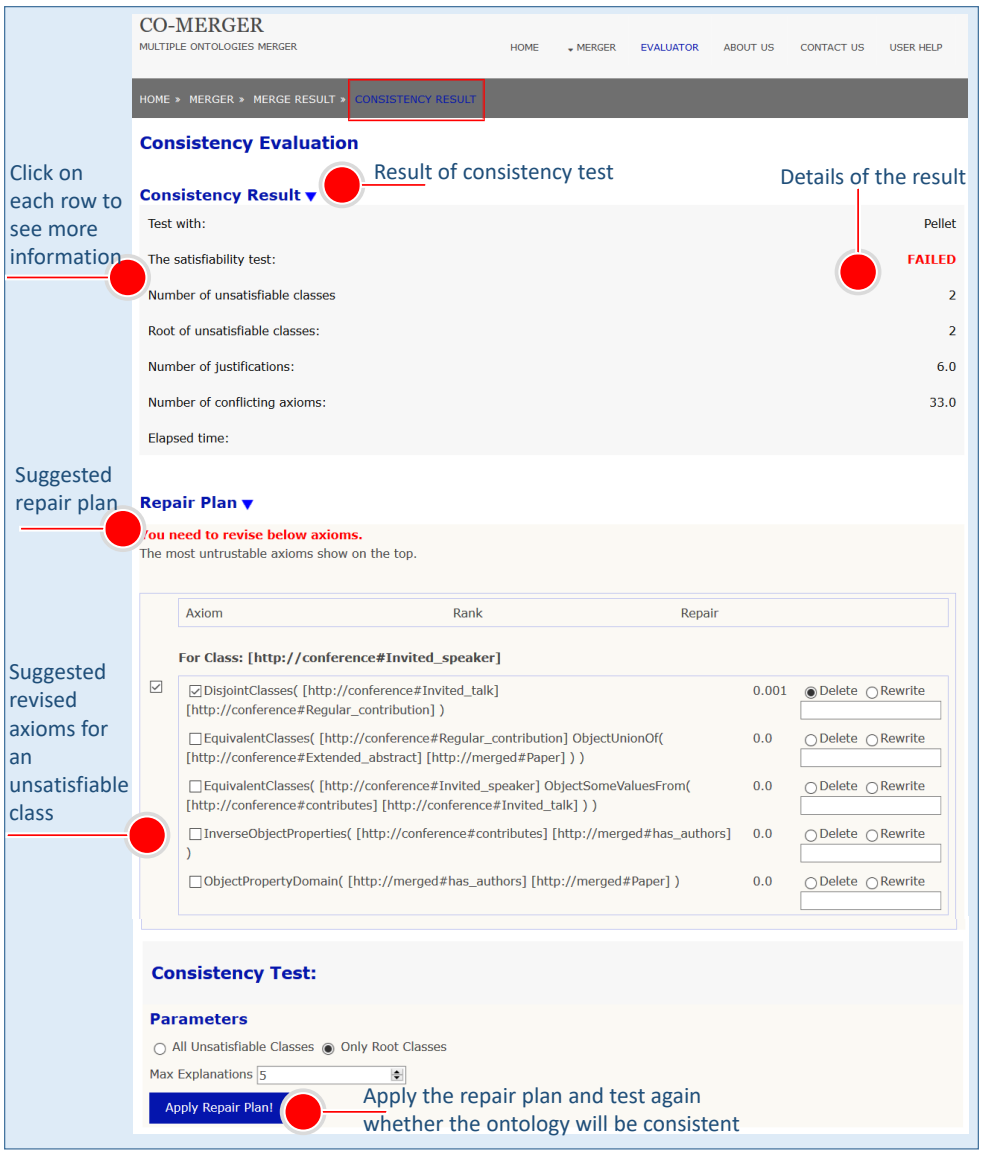


FIGURE C.7: Consistency test result.

C.4 Compatibility Checker

Figure C.8 shows the compatibility checker test, and Figure C.9 shows a sample result of this test.

FIGURE C.8: Compatibility checker GUI.

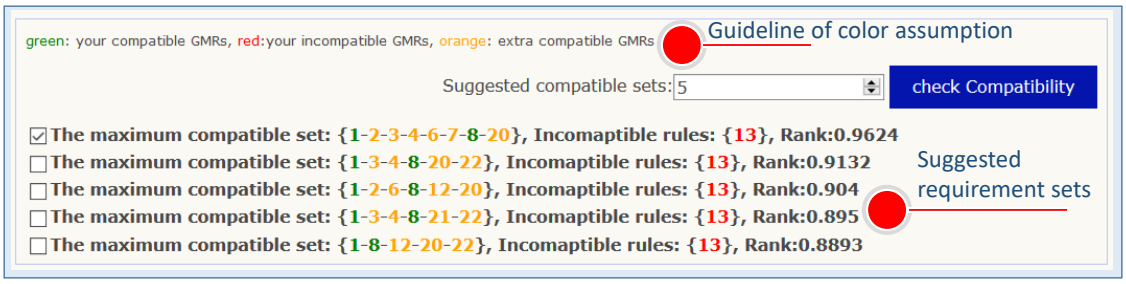


FIGURE C.9: Result of compatibility checker.

More information about GMRs can be achieved via the Requirement page. This is accessible under the USER HELP submenu (see Figure C.10)

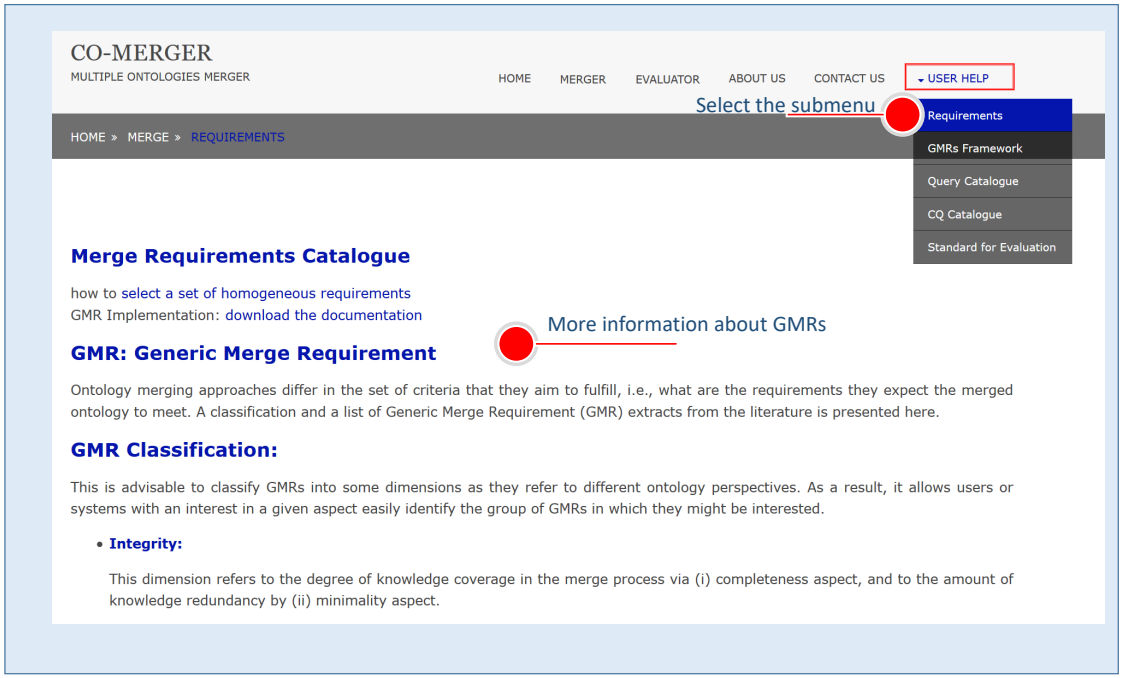


FIGURE C.10: Generic Merge Requirements (GMR)s information page

C.5 SPARQL Query Endpoint

Figure C.11 shows the detail of running a single SPARQL query both on the source and merged ontologies. Users can use ready templates or write their queries.

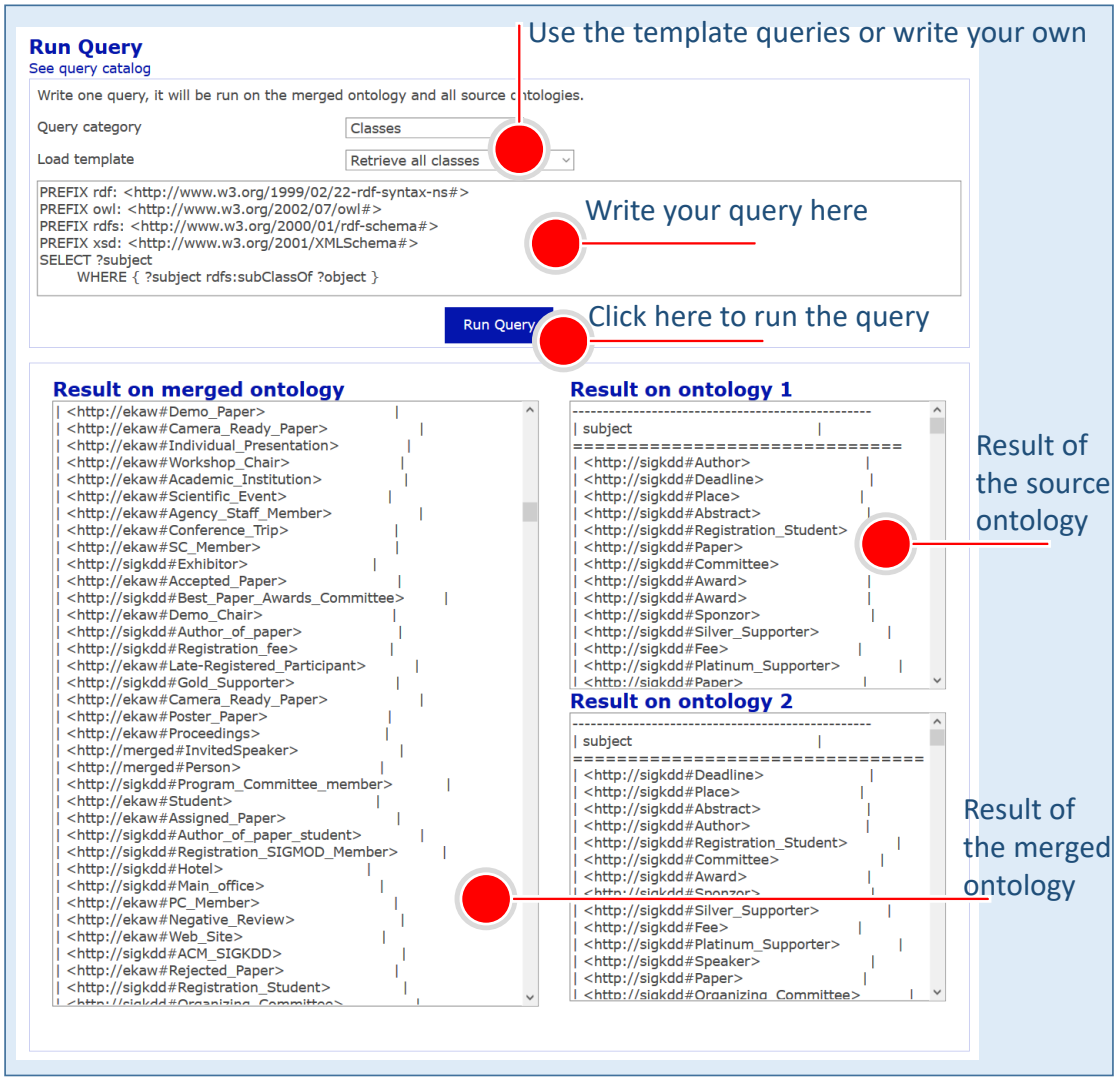


FIGURE C.11: Running a single query on the source and merged ontologies, simultaneously.

Figure C.12 shows the possibility of running different queries on the source or merged ontologies.

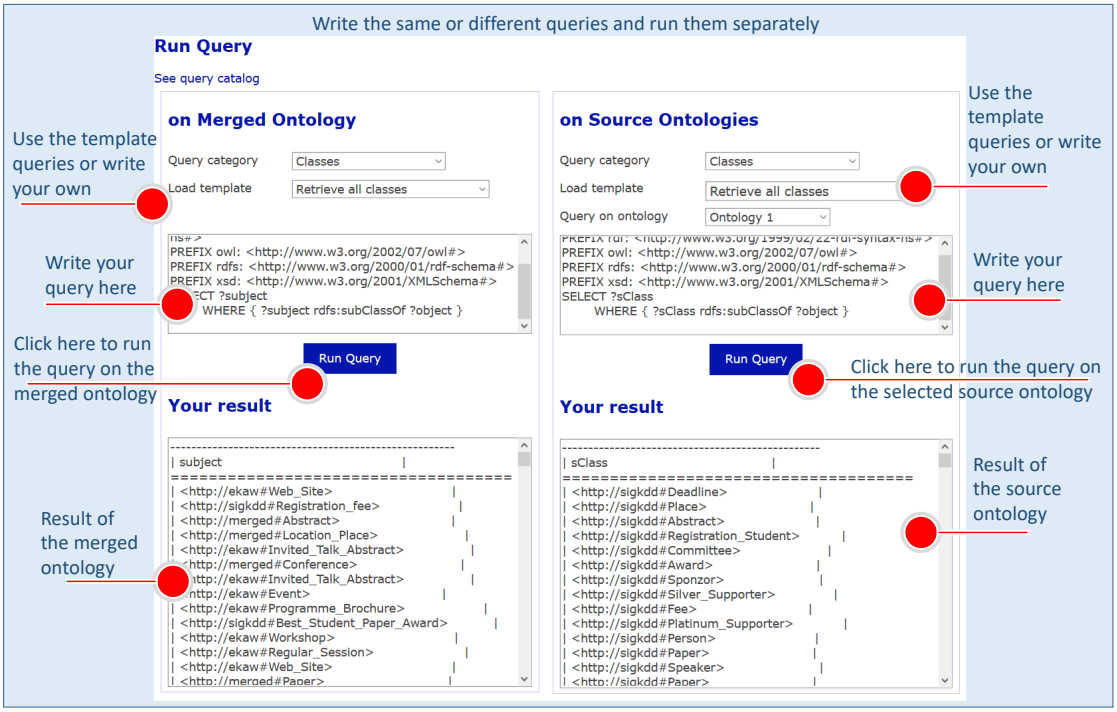


FIGURE C.12: Running different queries on the source or merged ontologies, separately.

Bibliography

[ABKD15] A. Algergawy, S. Babalou, M. J. Kargar, and S. H. Davarpanah, "SeeCOnt: A new seeding-based clustering approach for ontology matching," in *Advances in Databases and Information Systems*, 2015, ch. 17, ISBN: 9783319231341. DOI: http://dx.doi.org/10.1007/978-3-319-23135-8_17.

[ADMR05] D. Aumueller, H.-H. Do, S. Massmann, and E. Rahm, "Schema and ontology matching with COMA++," in *Proceedings of the 2005 ACM SIGMOD international conference on Management of data - SIGMOD '05*, 2005, pp. 906–908, ISBN: 1595930604. DOI: http://dx.doi.org/10.1145/1066157.1066283.

[AFF+19] A. Algergawy, D. Faria, A. Ferrara, I. Fundulaki, I. Harrow, S. Hertling, E. Jimenez-Ruiz, N. Karam, A. Khiat, P. Lambrix, *et al.*, "Results of the ontology alignment evaluation initiative 2019," in *CEUR Workshop Proceedings*, vol. 2536, 2019, pp. 46–85.

[AGL12] P. Archer, S Goedertier, and N Loutas, "Study on persistent uris, with identification of best practices and recommendations on the topic for the mss and the ec," *ISA Programme*, 2012.

[ALL10] F. F. de Araujo, F. L. R. Lopes, and B. F. Lóscio, "MeMO: A clustering-based approach for merging multiple ontologies," in *Workshops on Database and Expert Systems Applications*, IEEE, 2010, pp. 176–180, ISBN: 9781424480494. DOI: http://dx.doi.org/10.1109/DEXA.2010.50.

[BALKR17] S. Babalou, A. Algergawy, B. Lantow, and B. König-Ries, "Why the mapping process in ontology integration deserves attention," in *Proceedings of the 19th International Conference on Information Integration and Web-based Applications & Services - iiWAS '17*, ACM, 2017, pp. 451–456, ISBN: 9781450352994. DOI: http://dx.doi.org/10.1145/3151759.3151834.

[Bas92] V. R. Basili, "Software modeling and measurement: The goal/question/metric paradigm," Technical report, University of Maryland at College Park, College Park, MD, USA, Tech. Rep., 1992.

[BBL76] B. W. Boehm, J. R. Brown, and M. Lipow, "Quantitative evaluation of software quality," in *Proceedings of the 2nd international conference on Software engineering*, IEEE Computer Society Press, 1976, pp. 592–605.

[BCM+03] F. Baader, D. Calvanese, D. McGuinness, P. Patel-Schneider, and D. Nardi, *The description logic handbook: Theory, implementation and applications*. Cambridge university press, 2003, ISBN: 9780511711787. DOI: http://dx.doi.org/10.1017/CBO9780511711787.

[BDK92] P. Buneman, S. Davidson, and A. Kosky, "Theoretical aspects of schema merging," in *International Conference on Extending Database Technology*, Springer, 1992, ch. 11, pp. 152–167, ISBN: 3540552707. DOI: http://dx.doi.org/10.1007/BFb0032429.

[BGKR20a] S. Babalou, E. Grygorova, and B. König-Ries, "CoMerger: A customizable online tool for building a consistent quality-assured merged ontology," in *In 17th Extended Semantic Web Conference (ESWC'20), Poster and Demo Track*, 2020.

[BGKR20b] ——, "How good is this merged ontology?" In *In European Semantic Web Conference, Poster and Demo Track*, Springer, 2020, pp. 13–18. DOI: http://dx.doi.org/10.1007/978-3-030-62327-2_3.

[BGKR20c] ——, "What to do when the users of an ontology merging system want the impossible? towards determining compatibility of generic merge requirements," in *International Conference on Knowledge Engineering and Knowledge Management*, Springer, 2020, pp. 20–36. DOI: http://dx.doi.org/10.1007/978-3-030-61244-3_2.

[BGM05] J. Brank, M. Grobelnik, and D. Mladenic, "A survey of ontology evaluation techniques," in *Proceedings of the conference on data mining and data warehouses (SiKDD 2005)*, Citeseer Ljubljana, Slovenia, 2005, pp. 166–170.

[BKR19a] S. Babalou and B. König-Ries, "A subjective logic based approach to handling inconsistencies in ontology merging," in *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*, Springer, 2019, pp. 588–606. DOI: http://dx.doi.org/10.1007/978-3-030-33246-4_37.

[BKR19b] ——, "GMRs: Reconciliation of generic merge requirements in ontology integration," *In SEMANTICS Poster and Demo Track*, 2019.

[BKR19c] ——, "On using subjective logic to build consistent merged ontologies," *In SEMANTICS Poster and Demo Track*, 2019.

[BKR20a] ——, "Towards building knowledge by merging multiple ontologies with comerger: A partitioning-based approach," *arXiv preprint arXiv:2005.02659*, 2020.

[BKR20b] ——, "Towards multiple ontology merging with CoMerger," in *In International Semantic Web Conference (ISWC'20), Posters, Demos, and Industry Tracks*, 2020.

[BLKJ14] J. J. Budovec, C. A. Lam, and C. E. Kahn Jr, "Informatics in radiology: Radiology gamuts ontology: Differential diagnosis for the semantic web," *Radiographics*, vol. 34, no. 1, pp. 254–264, 2014, ISSN: 02715333. DOI: http://dx.doi.org/10.1148/rg.341135036.

[BLN86] C. Batini, M. Lenzerini, and S. B. Navathe, "A comparative analysis of methodologies for database schema integration," *ACM Computing Surveys (CSUR)*, vol. 18, no. 4, pp. 323–364, 1986, ISSN: 03600300. DOI: http://dx.doi.org/10.1145/27633.27634.

[Bod04] O. Bodenreider, "The unified medical language system (umls): Integrating biomedical terminology," *Nucleic acids research*, vol. 32, no. suppl 1, pp. D267–D270, 2004, ISSN: 13624962. DOI: http://dx.doi.org/10.1093/nar/gkh061.

[BPS11] S. Bail, B. Parsia, and U. Sattler, "Extracting finite sets of entailments from owl ontologies," in *Proceedings of the 24th international workshop on Description Logic, Barcelona, Spain*, Citeseer, 2011.

[BS05] J. Bailey and P. J. Stuckey, "Discovery of minimal unsatisfiable subsets of constraints using hitting set dualization," in *International Workshop on Practical Aspects of Declarative Languages*, Springer, 2005, ch. 14, pp. 174–186, ISBN: 9783540243625. DOI: http://dx.doi.org/10.1007/978-3-540-30557-6_14.

[CKP08] L. Chiticariu, P. G. Kolaitis, and L. Popa, "Interactive generation of integrated schemas," in *Proceedings of the 2008 ACM SIGMOD international conference on Management of data - SIGMOD '08*, 2008, pp. 833–846, ISBN: 9781605581026. DOI: http://dx.doi.org/10.1145/1376616.1376700.

[CNF12] D. Ceolin, A. Nottamkandath, and W. Fokkink, "Automated evaluation of annotators for museum collections using subjective logic," in *IFIP International Conference on Trust Management*, 2012, ch. 18, ISBN: 9783642298516. DOI: http://dx.doi.org/10.1007/978-3-642-29852-3_18.

[CR16] E. G. Caldarola and A. M. Rinaldi, "An approach to ontology integration for ontology reuse," in *IEEE 17th International Conference on Information Reuse and Integration (IRI)*, 2016, pp. 384–393, ISBN: 9781509032075. DOI: http://dx.doi.org/10.1109/IRI.2016.58.

[CVHF10] D. Ceolin, W. R. Van Hage, and W. Fokkink, "A trust model to estimate the quality of annotations using the web," *WebSci*, 2010.

[DA07] S Deelers and S Auwatanamongkol, "Enhancing k-means algorithm with initial cluster centers derived from data partitioning along the data axis with the highest variance," *International Journal of Computer Science*, vol. 2, no. 4, pp. 247–252, 2007.

[DB10] F. Duchateau and Z. Bellahsene, "Measuring the quality of an integrated schema.," in *Conceptual Modeling – ER*, 2010, ch. 19, pp. 261–273, ISBN: 9783642163722. DOI: http://dx.doi.org/10.1007/978-3-642-16373-9_19.

[DDF+90] S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, and R. Harshman, "Indexing by latent semantic analysis," *Journal of the American society for information science*, vol. 41, no. 6, pp. 391–407, 1990, ISSN: 00028231. DOI: http://dx.doi.org/10.1002/(SICI)1097-4571(199009)41:6%3C391::AID-ASI1%3E3.0.CO;2-9.

[DDS+16] M. Dubey, S. Dasgupta, A. Sharma, K. Höffner, and J. Lehmann, "Asknow: A framework for natural language query formalization in sparql," in *European Semantic Web Conference*, Springer, 2016, ch. 19, pp. 300–316, ISBN: 9783319341286. DOI: http://dx.doi.org/10.1007/978-3-319-34129-3_19.

[DGFE16] E. Degeval-Gesellschaft Für Evaluation, "Standards für evaluation," *Die Deutsche Bibliothek – CIP, Einheitsaufnahme DeGEval – Gesellschaft für Evaluation e.V. Standards für Evaluation- Erste Revision 2016, Mainz, 2017*, 2016, ISSN: 978-3-941569-06-5.

[Don06] K. Donnelly, "Snomed-ct: The advanced terminology and coding system for ehealth," *Studies in health technology and informatics*, vol. 121, p. 279, 2006.

[Doo+18] D. M. Dooley *et al.*, "FoodOn: A harmonized food ontology to increase global food traceability, quality control and data integration," *npj Science of Food*, vol. 2, 1 2018, ISSN: 23968370. DOI: http://dx.doi.org/10.1038/s41538-018-0032-6.

[DRFBSAG+11] A. Duque-Ramos, J. T. Fernández-Breis, R. Stevens, N. Aussenac-Gilles, *et al.*, "OQuaRE: A SQuaRE-based approach for evaluating the quality of ontologies," *Journal of Research and Practice in Information Technology*, vol. 43, no. 2, p. 159, 2011.

[dSSS07] M. d'Aquin, A. Schlicht, H. Stuckenschmidt, and M. Sabou, "Ontology modularization for knowledge selection: Experiments and evaluations," in *International Conference on Database and Expert Systems Applications*, Springer, 2007, ch. 85, pp. 874–883, ISBN: 9783540744672. DOI: http://dx.doi.org/10.1007/978-3-540-74469-6_85.

[DTI07] P. Doran, V. Tamma, and L. Iannone, "Ontology module extraction for ontology reuse: An ontology engineering perspective," in *Proceedings of the sixteenth ACM conference on Conference on information and knowledge management*, ACM, 2007, ch. 9781595938039, pp. 61–70. DOI: http://dx.doi.org/10.1145/1321440.1321451.

[EGED09] N. M. El-Gohary and T. E. El-Diraby, "Merging architectural, engineering, and construction ontologies," *Journal of Computing in Civil Engineering*, vol. 25, no. 2, pp. 109–128, 2009, ISSN: 08873801. DOI: http://dx.doi.org/10.1061/(ASCE)CP.1943-5487.0000048.

[Fah17] M. Fahad, "Merging of axiomatic definitions of concepts in the complex owl ontologies," *Artificial Intelligence Review*, vol. 47, no. 2, pp. 181–215, 2017, ISSN: 02692821. DOI: http://dx.doi.org/10.1007/s10462-016-9479-5.

[FFKJ19] M. T. Finke, R. W. Filice, and C. E. Kahn Jr, "Integrating ontologies of human diseases, phenotypes, and radiological diagnosis," *Journal of*

the American Medical Informatics Association, vol. 26, no. 2, pp. 149–154, 2019. DOI: https://doi.org/10.1093/jamia/ocy161.

[FHP+06] G. Flouris, Z. Huang, J. Z. Pan, D. Plexousakis, and H. Wache, "Inconsistencies, negations and changes in ontologies," in *AAAI*, vol. 21, 2006, pp. 1295–1300.

[FMB12] M. Fahad, N. Moalla, and A. Bouras, "Detection and resolution of semantic inconsistency and redundancy in an automatic ontology merging system," *Journal of Intelligent Information Systems*, vol. 39, no. 2, pp. 535–557, 2012, ISSN: 09259902. DOI: http://dx.doi.org/10.1007/s10844-012-0202-y.

[FRP14] D. H. Fudholi, W. Rahayu, and E. Pardede, "Code (common ontology development): A knowledge integration approach from multiple ontologies," in *IEEE 28th International Conference on Advanced Information Networking and Applications*, 2014, pp. 751–758, ISBN: 9781479936304. DOI: http://dx.doi.org/10.1109/AINA.2014.92.

[GAC10] A. Guzmán-Arenas and A.-D. Cuevas, "Knowledge accumulation through automatic merging of ontologies," *Expert Systems with Applications*, vol. 37, no. 3, pp. 1991–2005, 2010, ISSN: 09574174. DOI: http://dx.doi.org/10.1016/j.eswa.2009.06.078.

[GBKR20] E. Grygorova, S. Babalou, and B. König-Ries, "Toward owl restriction reconciliation in merging knowledge," in *In 17th Extended Semantic Web Conference (ESWC'20), Poster and Demo Track*, 2020.

[GCCL05] A. Gangemi, C. Catenacci, M. Ciaramita, and J. Lehmann, "Ontology evaluation and validation: An integrated formal model for the quality diagnostic task," *On-line: http://www.loa-cnr.it/Files/OntoEval4OntoDev_Final. pdf*, 2005.

[GLR+14] G. Governatori, H.-P. Lam, A. Rotolo, S. Villata, G. A. Atemezing, and F. L. Gandon, "Checking licenses compatibility between vocabularies and data.," *COLD*, vol. 1264, 2014.

[Gor11] T. F. Gordon, "Analyzing open source license compatibility issues with carneades," in *Proceedings of the 13th International Conference on Artificial Intelligence and Law*, 2011, pp. 51–55, ISBN: 9781450307550. DOI: http://dx.doi.org/10.1145/2018358.2018364.

[Gru+93] T. R. Gruber *et al.*, "A translation approach to portable ontology specifications," *Knowledge Acquisition*, vol. 5, no. 2, pp. 199–220, 1993, ISSN: 10428143. DOI: http://dx.doi.org/10.1006/knac.1993.1008.

[GW12] B. Ganter and R. Wille, *Formal concept analysis: mathematical foundations*. Springer Science & Business Media, 2012.

[HB11] M. Horridge and S. Bechhofer, "The owl api: A java api for owl ontologies," *Semantic Web*, vol. 2, no. 1, pp. 11–21, 2011, ISSN: 15700844. DOI: http://dx.doi.org/10.3233/SW-2011-0025.

[HCZQ11] W. Hu, J. Chen, H. Zhang, and Y. Qu, "How matchable are four thousand ontologies on the semantic web," *The Semantic Web: Research*

and Applications, pp. 290–304, 2011, ISSN: 9783642210334. DOI: http://dx.doi.org/10.1007/978-3-642-21034-1_20.

[HHP+10] A. Hogan, A. Harth, A. Passant, S. Decker, and A. Polleres, "Weaving the pedantic web.," *LDOW*, vol. 628, 2010.

[HPS09] M. Horridge, B. Parsia, and U. Sattler, "Explaining inconsistencies in owl ontologies," in *Scalable Uncertainty Management*, 2009, ch. 11, pp. 124–137, ISBN: 9783642043871. DOI: http://dx.doi.org/10.1007/978-3-642-04388-8_11.

[HQC08] W. Hu, Y. Qu, and G. Cheng, "Matching large ontologies: A divide-and-conquer approach," *Data & Knowledge Engineering*, vol. 67, no. 1, pp. 140–160, 2008, ISSN: 0169023X. DOI: http://dx.doi.org/10.1016/j.datak.2008.06.003.

[HS08] D. Hooijmaijers and M. Stumptner, "Improving integration with subjective combining of ontology mappings," in *International Symposium on Methodologies for Intelligent Systems*, 2008, ch. 60, pp. 552–562, ISBN: 9783540681229. DOI: http://dx.doi.org/10.1007/978-3-540-68123-6_60.

[HS14] H. Hlomani and D. Stacey, "Approaches, methods, metrics, measures, and subjectivity in ontology evaluation: A survey," *Semantic Web Journal*, vol. 1, no. 5, pp. 1–11, 2014.

[HSP+19] G. Havur, S. Steyskal, O. Panasiuk, A. Fensel, V. Mireles, T. Pellegrini, T. Thurner, A. Polleres, and S. Kirrane, "Automatic license compatibility checking.," in *SEMANTICS Posters and Demos*, 2019.

[HTR06] M. Horridge, D. Tsarkov, and T. Redmond, "Supporting early adoption of owl 1.1 with protege-owl and fact++.," in *Proceedings of the Second OWL Experiences and Directions Workshop (OWLED-06)*, 2006.

[HVHH+05] P. Haase, F. Van Harmelen, Z. Huang, H. Stuckenschmidt, and Y. Sure, "A framework for handling inconsistency in changing ontologies," in *International semantic web conference*, Springer, 2005, ch. 27, pp. 353–367, ISBN: 9783540297543. DOI: http://dx.doi.org/10.1007/11574620_27.

[HVHTT05] Z. Huang, F. Van Harmelen, and A. Ten Teije, "Reasoning with inconsistent ontologies.," in *Proceedings of the 19th international joint conference on Artificial intelligence*, vol. 5, 2005, pp. 454–459.

[IEC05] I. IEC, "Iso/iec 25000–software engineering–software product quality requirements and evaluation (square)–guide to square," *Systems Engineering*, vol. 41, 2005.

[Jac01] P. Jaccard, "Étude comparative de la distribution florale dans une portion des alpes et des jura," *Bull Soc Vaudoise Sci Nat*, vol. 37, pp. 547–579, 1901.

[JERS+11] S. P. Ju, H. E. Esquivel, A. M. Rebollar, M. C. Su, *et al.*, "CreaDO–a methodology to create domain ontologies using parameter-based ontology merging techniques," in *10th Mexican International Conference on Artificial Intelligence*, IEEE, 2011, pp. 23–28.

[JHQ+09] Q. Ji, P. Haase, G. Qi, P. Hitzler, and S. Stadtmüller, "Radon—repair and diagnosis in ontology networks," in *European Semantic Web Conference*, Springer, 2009, pp. 863–867.

[Jøs16] A. Jøsang, *Subjective logic*. Springer, 2016.

[JRASC18] E. Jiménez-Ruiz, A. Agibetov, M. Samwald, and V. Cross, "We divide, you conquer: From large-scale ontology alignment to manageable subtasks with a lexical index and neural embeddings," in *CEUR Workshop Proceedings*, vol. 2288, 2018, pp. 13–24.

[JRGH12] E. Jiménez-Ruiz, B. C. Grau, and I. Horrocks, "On the feasibility of using owl 2 dl reasoners for ontology matching problems.," in *ORE*, 2012.

[JRGHB09] E. Jiménez-Ruiz, B. C. Grau, I. Horrocks, and R. Berlanga, "Ontology integration using mappings: Towards getting the right logical consequences," in *ESWC*, Springer, 2009, ch. 16, pp. 173–187, ISBN: 9783642021206. DOI: http://dx.doi.org/10.1007/978-3-642-02121-3_16.

[Jus04] K. Juszczyszyn, "A subjective logic-based framework for aligning multiple ontologies," in *International Conference on Knowledge-Based and Intelligent Information and Engineering Systems*, Springer, 2004, ch. 159, pp. 1194–1200, ISBN: 9783540232063. DOI: http://dx.doi.org/10.1007/978-3-540-30133-2_159.

[KF01] M. Klein and D. Fensel, "Ontology versioning on the semantic web," in *SWWS*, 2001, pp. 75–91.

[KJH+05] J. Kim, M. Jang, Y.-G. Ha, J.-C. Sohn, and S. J. Lee, "MoA: Owl ontology merging and alignment tool for the semantic web," in *International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems*, Springer, 2005, ch. 100, pp. 722–731. DOI: http://dx.doi.org/10.1007/11504894_100.

[KP17] A. Khamparia and B. Pandey, "Comprehensive analysis of semantic web reasoners and tools: A survey," *Education and Information Technologies*, vol. 22, no. 6, pp. 3121–3145, 2017, ISSN: 13602357. DOI: http://dx.doi.org/10.1007/s10639-017-9574-5.

[KPSCG06] A. Kalyanpur, B. Parsia, E. Sirin, and B. Cuenca-Grau, "Repairing unsatisfiable concepts in owl ontologies," in *European Semantic Web Conference*, 2006, ch. 15, pp. 170–184, ISBN: 9783540345442. DOI: http://dx.doi.org/10.1007/11762256_15.

[KPSH05] A. Kalyanpur, B. Parsia, E. Sirin, and J. Hendler, "Debugging unsatisfiable classes in owl ontologies," *Journal of Web Semantics*, vol. 3, no. 4, pp. 268–293, 2005, ISSN: 15708268. DOI: http://dx.doi.org/10.1016/j.websem.2005.09.005.

[KS03] Y. Kalfoglou and M. Schorlemmer, "Ontology mapping: The state of the art," *The Knowledge Engineering Review*, vol. 18, pp. 1–31, 1 2003, ISSN: 02698889. DOI: http://dx.doi.org/10.1017/S0269888903000651.

[KVS06] K. Kotis, G. A. Vouros, and K. Stergiou, "Towards automatic merging of domain ontologies: The hcone-merge approach," *Journal of Web*

Semantics, vol. 4, no. 1, pp. 60–79, 2006, ISSN: 15708268. DOI: http://dx.doi.org/10.1016/j.websem.2005.09.004.

[LBBH15] K. M. Livingston, M. Bada, W. A. Baumgartner, and L. E. Hunter, "KaBOB: Ontology-based semantic integration of biomedical databases," *BMC bioinformatics*, vol. 16, no. 1, p. 1, 2015, ISSN: 14712105. DOI: http://dx.doi.org/10.1186/s12859-015-0559-3.

[Lip00] C. E. Lipscomb, "Medical subject headings (mesh)," *Bulletin of the Medical Library Association*, vol. 88, no. 3, p. 265, 2000.

[LPSV06] J Lam, J. Z. Pan, D. Sleeman, and W. Vasconcelos, "Ontology inconsistency handling: Ranking and rewriting axioms," *Technical report aucs/tr0603, University of Aberdeen*, 2006.

[LT06] P. Lambrix and H. Tan, "SAMBO- a system for aligning and merging biomedical ontologies," *Journal of Web Semantics*, vol. 4, no. 3, pp. 196–206, 2006, ISSN: 15708268. DOI: http://dx.doi.org/10.1016/j.websem.2006.05.003.

[Mac+67] J. MacQueen *et al.*, "Some methods for classification and analysis of multivariate observations," in *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, Oakland, CA, USA, vol. 1, 1967, pp. 281–297.

[MFBB10] N. Maiz, M. Fahad, O. Boussaid, and F. Bentayeb, "Automatic ontology merging by hierarchical clustering and inference mechanisms," in *Proceedings of I-KNOW*, 2010, pp. 1–3.

[MFH16] M. Mahfoudh, G. Forestier, and M. Hassenforder, "A benchmark for ontologies merging assessment," in *Knowledge Science, Engineering and Management*, 2016, ch. 44, pp. 555–566, ISBN: 9783319476490. DOI: http://dx.doi.org/10.1007/978-3-319-47650-6_44.

[MFRW00] D. L. McGuinness, R. Fikes, J. Rice, and S. Wilder, "An environment for merging and testing large ontologies," in *KR*, 2000, pp. 483–493.

[MG18a] A. Makwana and A. Ganatra, "A better approach to ontology integration using clustering through global similarity measure.," *Journal of Computer Science*, vol. 14, no. 6, pp. 854–867, 2018, ISSN: 15493636. DOI: http://dx.doi.org/10.3844/jcssp.2018.854.867.

[MG18b] ——, "A known in advance, what ontologies to integrate? for effective ontology merging using k-means clustering," *International Journal of Intelligent Engineering and Systems*, vol. 11, p. 72, 4 2018, ISSN: 21853118. DOI: http://dx.doi.org/10.22266/ijies2018.0831.08.

[Mil98] G. A. Miller, *WordNet: An electronic lexical database*. MIT press, 1998.

[MRB03] S. Melnik, E. Rahm, and P. A. Bernstein, "Rondo: A programming platform for generic model management," in *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, ACM, 2003, pp. 193–204, ISBN: 158113634X. DOI: http://dx.doi.org/10.1145/872757.872782.

[MTFH14] M. Mahfoudh, L. Thiry, G. Forestier, and M. Hassenforder, "Algebraic graph transformations for merging ontologies," in *Model and Data*

Engineering, Springer, 2014, ch. 16, pp. 154–168, ISBN: 9783319115863. DOI: http://dx.doi.org/10.1007/978-3-319-11587-0_16.

[MVH+04] D. L. McGuinness, F. Van Harmelen, *et al.*, "Owl web ontology language overview," *W3C recommendation*, vol. 10, no. 10, p. 2004, 2004.

[Nie74] J. Nieminen, "On the centrality in a graph," *Scandinavian journal of psychology*, vol. 15, no. 1, pp. 332–336, 1974, ISSN: 00365564. DOI: http://dx.doi.org/10.1111/j.1467-9450.1974.tb00598.x.

[NM+01] N. F. Noy, D. L. McGuinness, *et al.*, *Ontology development 101: A guide to creating your first ontology*, 2001.

[NM03] N. F. Noy and M. A. Musen, "The prompt suite: Interactive tools for ontology merging and mapping," *International Journal of Human-Computer Studies*, vol. 59, no. 6, pp. 983–1024, 2003, ISSN: 10715819. DOI: http://dx.doi.org/10.1016/j.ijhcs.2003.08.002.

[NNBU+13] A.-C. Ngonga Ngomo, L. Bühmann, C. Unger, J. Lehmann, and D. Gerber, "Sorry, i don't speak sparql: Translating sparql queries into natural language," in *Proceedings of the 22nd international conference on World Wide Web*, 2013, pp. 977–988, ISBN: 9781450320351. DOI: http://dx.doi.org/10.1145/2488388.2488473.

[NSD+01] N. F. Noy, M. Sintek, S. Decker, M. Crubézy, R. W. Fergerson, and M. A. Musen, "Creating semantic web contents with protege-2000," *IEEE intelligent systems*, vol. 16, no. 2, pp. 60–71, 2001, ISSN: 15411672. DOI: http://dx.doi.org/10.1109/5254.920601.

[PB03] R. A. Pottinger and P. A. Bernstein, "Merging models based on given correspondences," in *Proceedings 2003 VLDB Conference*, Elsevier, 2003, pp. 862–873, ISBN: 9780127224428. DOI: http://dx.doi.org/10.1016/B978-012722442-8/50081-1.

[PC19] M Priya and A. K. Cherukuri, "A novel method for merging academic social network ontologies using formal concept analysis and hybrid semantic similarity measure," *Library Hi Tech*, vol. 38, p. 399, 2 2019, ISSN: 07378831. DOI: http://dx.doi.org/10.1108/LHT-02-2019-0035.

[PDT06] P. Plessers and O. De Troyer, "Resolving inconsistencies in evolving ontologies," in *European Semantic Web Conference*, Springer, 2006, ch. 17, pp. 200–214, ISBN: 9783540345442. DOI: http://dx.doi.org/10.1007/11762256_17.

[PHZY17] M. Paixao, M. Harman, Y. Zhang, and Y. Yu, "An empirical study of cohesion and coupling: Balancing optimization and disruption," *IEEE Transactions on Evolutionary Computation*, vol. 22, no. 3, pp. 394–414, 2017, ISSN: 1089778X. DOI: http://dx.doi.org/10.1109/TEVC.2017.2691281.

[PK12] G. Pitsilis and S. J. Knapskog, "Social trust as a solution to address sparsity-inherent problems of recommender systems," *arXiv preprint arXiv:1208.1004*, 2012.

[PK19] M Priya and C. A. Kumar, "An approach to merge domain ontologies using granular computing," *Granular Computing*, pp. 1–26, 2019, ISSN: 23644966. DOI: http://dx.doi.org/10.1007/s41066-019-00193-3.

[PSK05] B. Parsia, E. Sirin, and A. Kalyanpur, "Debugging owl ontologies," in *Proceedings of the 14th international conference on World Wide Web*, ACM, 2005, pp. 633–640, ISBN: 1595930469. DOI: http://dx.doi.org/10.1145/1060745.1060837.

[PTP+18] D. Porello, N. Troquard, R. Penaloza, R. Confalonieri, P. Galliani, and O. Kutz, "Two approaches to ontology aggregation based on axiom weakening.," in *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence*, 2018, ISBN: 9780999241127. DOI: http://dx.doi.org/10.24963/ijcai.2018/268.

[PVGPSF14] M. Poveda-Villalón, A. Gómez-Pérez, and M. C. Suárez-Figueroa, "Oops! (ontology pitfall scanner!): An on-line tool for ontology evaluation," *International Journal on Semantic Web and Information Systems*, vol. 10, no. 2, pp. 7–34, 2014, ISSN: 15526283. DOI: http://dx.doi.org/10.4018/ijswis.2014040102.

[PVSFGP10] M. Poveda Villalon, M. C. Suárez-Figueroa, and A. Gómez-Pérez, "A double classification of common pitfalls in ontologies," *Workshop on Ontology Quality (OntoQual 2010), Co-located with EKAW*, 2010.

[PVSFGP12] M. Poveda-Villalón, M. C. Suárez-Figueroa, and A. Gómez-Pérez, "Validating ontologies with oops!" In *Knowledge Engineering and Knowledge Management*, 2012, ch. 24, pp. 267–281, ISBN: 9783642338755. DOI: http://dx.doi.org/10.1007/978-3-642-33876-2_24.

[QKL07] C. Quix, D. Kensche, and X. Li, "Generic schema merging," in *International Conference on Advanced Information Systems Engineering*, Springer, 2007, ch. 10, pp. 127–141, ISBN: 9783319981765. DOI: http://dx.doi.org/10.1007/978-3-540-72988-4_10.

[Rah16] E. Rahm, "The case for holistic data integration," in *Advances in Databases and Information Systems*, 2016, ch. 2, pp. 11–27, ISBN: 9783319440385. DOI: http://dx.doi.org/10.1007/978-3-319-44039-2_2.

[RB01] E. Rahm and P. A. Bernstein, "A survey of approaches to automatic schema matching," *The VLDB Journal*, vol. 10, no. 4, pp. 334–350, 2001, ISSN: 10668888. DOI: http://dx.doi.org/10.1007/s007780100057.

[RC15] J. Raad and C. Cruz, "A survey on ontology evaluation methods," in *Proceedings of the 7th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management*, 2015, ISBN: 9789897581588. DOI: http://dx.doi.org/10.5220/0005591001790186.

[RCC92] D. A. Randell, Z. Cui, and A. G. Cohn, "A spatial logic based on regions and connection.," *KR*, vol. 92, pp. 165–176, 1992.

[RDH+04] A. Rector, N. Drummond, M. Horridge, J. Rogers, H. Knublauch, R. Stevens, H. Wang, and C. Wroe, "Owl pizzas: Practical experience of

teaching owl-dl: Common errors & common patterns," in *International Conference on Knowledge Engineering and Knowledge Management*, Springer, 2004, ch. 5, pp. 63–81, ISBN: 9783540233404. DOI: http://dx.doi.org/10.1007/978-3-540-30202-5_5.

[RKB+08] P. N. Robinson, S. Köhler, S. Bauer, D. Seelow, D. Horn, and S. Mundlos, "The human phenotype ontology: A tool for annotating and analyzing human hereditary disease," *The American Journal of Human Genetics*, vol. 83, no. 5, pp. 610–615, 2008, ISSN: 00029297. DOI: http://dx.doi.org/10.1016/j.ajhg.2008.09.017.

[RPSY09] A. Radwan, L. Popa, I. R. Stanoi, and A. Younis, "Top-k generation of integrated schemas based on directed and weighted correspondences," in *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*, ACM, 2009, pp. 641–654, ISBN: 9781605585512. DOI: http://dx.doi.org/10.1145/1559845.1559913.

[RR12] S. Raunich and E. Rahm, "Towards a benchmark for ontology merging.," in *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*, vol. 7567, 2012, ch. 20, pp. 124–133, ISBN: 9783642336171. DOI: http://dx.doi.org/10.1007/978-3-642-33618-8_20.

[RR14] ——, "Target-driven merging of taxonomies with ATOM," *Information Systems*, vol. 42, pp. 1–14, 2014, ISSN: 03064379. DOI: http://dx.doi.org/10.1016/j.is.2013.11.001.

[SAN+11] L. M. Schriml, C. Arze, S. Nadendla, Y.-W. W. Chang, M. Mazaitis, V. Felix, G. Feng, and W. A. Kibbe, "Disease ontology: A backbone for disease semantic integration," *Nucleic acids research*, vol. 40, no. D1, pp. D940–D946, 2011, ISSN: 03051048. DOI: http://dx.doi.org/10.1093/nar/gkr972.

[SBH08] K. Saleem, Z. Bellahsene, and E. Hunt, "Porsche: Performance oriented schema mediation," *Information Systems*, vol. 33, no. 7, pp. 637–657, 2008, ISSN: 03064379. DOI: http://dx.doi.org/10.1016/j.is.2008.01.010.

[SC+03] S. Schlobach, R. Cornet, *et al.*, "Non-standard reasoning services for the debugging of description logic terminologies," in *Ijcai*, vol. 3, 2003, pp. 355–362.

[SJRG14] A. Solimando, E. Jiménez-Ruiz, and G. Guerrini, "Detecting and correcting conservativity principle violations in ontology-to-ontology mappings," in *International Semantic Web Conference*, Springer, 2014, ch. 1, pp. 1–16, ISBN: 9783319119144. DOI: http://dx.doi.org/10.1007/978-3-319-11915-1_1.

[SM01] G. Stumme and A. Maedche, "FCA-Merge: Bottom-up merging of ontologies," in *IJCAI*, vol. 1, 2001, pp. 225–230.

[SPF+12] M. Sensoy, J. Z. Pan, A. Fokoue, M. Srivatsa, and F. Meneguzzi, "Using subjective logic to handle uncertainty and conflicts," in *IEEE 11th International Conference on Trust, Security and Privacy in Computing*

and Communications, 2012, pp. 1323–1326, ISBN: 9781467321723. DOI: http://dx.doi.org/10.1109/TrustCom.2012.294.

[SPG+07] E. Sirin, B. Parsia, B. C. Grau, A. Kalyanpur, and Y. Katz, "Pellet: A practical owl-dl reasoner," *Journal of Web Semantics*, vol. 5, pp. 51–53, 2 2007, ISSN: 15708268. DOI: http://dx.doi.org/10.1016/j.websem.2007.03.004.

[TBL08] D. Thau, S. Bowers, and B. Ludäscher, "Merging taxonomies under rcc-5 algebraic articulations," in *Proceedings of the 2nd international workshop on Ontologies and information systems for the semantic web*, ACM, 2008, pp. 47–54, ISBN: 9781605582559. DOI: http://dx.doi.org/10.1145/1458484.1458492.

[TTT06] E. Tomita, A. Tanaka, and H. Takahashi, "The worst-case time complexity for generating all maximal cliques and computational experiments," *Theoretical Computer Science*, vol. 363, pp. 28–42, 1 2006, ISSN: 03043975. DOI: http://dx.doi.org/10.1016/j.tcs.2006.06.015.

[UA10] O. Unal and H. Afsarmanesh, "Semi-automated schema integration with SASMINT," *Knowledge and information systems*, vol. 23, no. 1, pp. 99–128, 2010, ISSN: 02191377. DOI: http://dx.doi.org/10.1007/s10115-009-0217-z.

[UKMZ98] M. Uschold, M. King, S. Moralee, and Y. Zorgios, "The enterprise ontology," *The knowledge engineering review*, vol. 13, no. 01, pp. 31–89, 1998, ISSN: 02698889. DOI: http://dx.doi.org/10.1017/S0269888998001088.

[Usc96] M. Uschold, "Building ontologies: Towards a unified methodology," in *Proceedings of 16th Annual Conference of the British Computer Society Specialists Group on Expert Systems*, Citeseer, 1996.

[Wil09] R. Wille, "Restructuring lattice theory: An approach based on hierarchies of concepts," in *International Conference on Formal Concept Analysis*, Springer, 2009, ch. 23, pp. 314–339, ISBN: 9783642018145. DOI: http://dx.doi.org/10.1007/978-3-642-01815-2_23.

[ZPVOS18] O. P. Zalamea Patino, J. Van Orshoven, and T. Steenberghen, "Merging and expanding existing ontologies to cover the built cultural heritage domain," *Journal of Cultural Heritage Management and Sustainable Development*, vol. 8, no. 2, pp. 162–178, 2 2018, ISSN: 20441266. DOI: http://dx.doi.org/10.1108/JCHMSD-05-2017-0028.

[ZRL17] L.-Y. Zhang, J.-D. Ren, and X.-W. Li, "OIM-SM: A method for ontology integration based on semantic mapping," *Journal of Intelligent & Fuzzy Systems*, vol. 32, no. 3, pp. 1983–1995, 2017, ISSN: 10641246. DOI: http://dx.doi.org/10.3233/JIFS-161553.

[ZS17] O. Zamazal and V. Svátek, "The ten-year ontofarm and its fertilization within the onto-sphere," *Journal of Web Semantics*, vol. 43, pp. 46–53, 2017, ISSN: 15708268. DOI: http://dx.doi.org/10.1016/j.websem.2017.01.001.