DEVELOPMENT AND EXPERIMENTAL EVALUATION OF AN ONTOLOGY TO ONTOLOGY SCHEMA & INSTANCE MATCHING SYSTEM

MASTER OF SCIENCE THESIS
DASKALAKI EVANGELIA

Thesis Supervisor: Prof. Plexousakis Dimitris

This work has been performed at the Foundation of Research & Technology – Hellas, Institute of Computer Science.
The work is partially supported by Plug-IT European project under contract number FP7-3ICT-231413
DEVELOPMENT AND EXPERIMENTAL EVALUATION OF AN
ONTOLOGY TO ONTOLOGY SCHEMA & INSTANCE
MATCHING SYSTEM

Daskalaki Evangelia

Master of Science Thesis

Computer Science Department, University of Crete
Abstract

Ontology, as the means to conceptualize domain knowledge, has become the enabler of the fulfillment of the semantic web vision. It aims to create a platform where information has its semantics and can be understood and processed by computers themselves with minimum human interference, thus, to make data sharable.

Unfortunately, ontologies themselves are heterogeneous and distributed. Defined by different organizations or by different people in the same organization, ontologies can have vastly different characteristics and structures. Therefore, in order to achieve semantic interoperability across ontologies, it is necessary to discover the alignment across ontologies and their instance matches.

The demand for high-quality ontology instance matching is crucial in the context of identity recognition, ontology population and semantic integration. Several frameworks/systems for ontology-based information integration have been proposed. Their research efforts are mainly focused on the integration at the ontology schema level. For those who consider integrating data at the ontology data level for creating or maintain data relations, just a few systems have been proposed.

Our purpose is to offer a multi-strategy ontology to ontology (OtO) matching system, for schema alignment and instance matching, which is domain independent and fully customizable from the domain expert at any level. We argue that, unless a system is fully customizable, it can not claim be domain independent.

The proposed system implements several different ontology matching processes, which can be roughly categorized into the processes that include schema matching algorithms and the processes that include instance matching algorithms.

The schema matching process leverages the lexical similarities, the semantic similarities and the syntactic similarities of the entities of the schema. As far as the instance matching process is concerned, it does not only rely on lexical similarity measures and pair-wise instance comparison. The system utilizes the instance matching process by leveraging (1) the rich semantic knowledge we gain from the output mappings of the schema matching process, (2) the implicit knowledge of domain expert by (semi-)automatically capturing the identification power of the properties and (3) the probability calculation of the result’s truth, in order to accurately and efficiently detect the ontology instances that represent the same real-world entity.

Supervisor:
Dr. Dimitris Plexousakis,
Professor at Computer Science Department,
University of Crete
Ηράκλειο, Οκτώβριος 2011

Υλοποίηση & Πειραματική Αξιολόγηση Συστήματος Συσχέτισης Σχημάτων και Στιγμιότυπων Οντολογιών

Δασκαλάκη Ευαγγελία

Μεταπτυχιακή Εργασία

Τμήμα Επιστήμης Υπολογιστών
Περίληψη

Οι Οντολογίες, ως βασικό μέσο αναπαράστασης γνώσης, έχουν γίνει ο καταλύτης για την εκπλήρωση του οράματος του Σημασιολογικού Ιστού. Ορομα του είναι, να δημιουργήσει μια πλατφόρμα όπου κάθε πληροφορία έχει την σημασιολογία της και μπορεί να γίνει «κατανοητή» και από τους υπολογιστές με την ελάχιστη ανθρώπινη παρέμβαση. Κατ’ επέκταση να γίνουν τα δεδομένα διαμοιράσιμα.

Δυστυχώς, οι οντολογίες είναι εξ ορισμού ετερογενείς. Ακόμη και όταν ορίζεται ο ίδιος τομέας εφαρμογής από διαφορετικά άτομα, οι οντολογίες μπορούν να έχουν πολύ διαφορετικά χαρακτηριστικά και δομές. Ως εκ τούτου, για να επιτευχθεί η σημασιολογική διαλειτουργικότητα μεταξύ των οντολογιών, είναι απαραίτητο να ανακαλυφθούν οι συσχετίσεις στα σχήματα αλλά και στα στιγμιότυπα τους.

Η υψηλής ποιότητας συσχετίσεις στιγμιότυπων δύο οντολογιών είναι ζωτικής σημασίας στο πλαίσιο της αναγνώρισης ταυτότητας, της πληθυσμιακής ενημέρωσης οντολογίας και της σημασιολογικής ολοκλήρωσης. Τα τελευταία χρόνια πολλά συστήματα συσχέτισης οντολογιών έχουν προταθεί. Οι ερευνητικές προσπάθειες του παρελθόντος όμως, έχουν επικεντρωθεί στη σημασιολογική συσχέτιση των σχήματων των οντολογιών. Για όσους ενδιαφέρονται για την ενοποίηση πληροφοριών ή για την δημιουργία και την διατήρηση σχέσεων μεταξύ των δεδομένων, μόνο λίγα συστήματα έχουν προταθεί.

Σκοπός μας είναι να προσφέρουμε ένα σύστημα συσχέτισης οντολογιών, για την συσχέτιση σχήματων και στιγμιότυπων, που θα εμπεριέχει πολλές στρατηγικές συσχέτισης, θα είναι ανεξάρτητο από τους τομείς εφαρμογής και πλήρως παραμετροποιήσιμο από τον τελικό χρήστη. Πεποίθηση μας είναι ότι αν ένα σύστημα δεν είναι πλήρως παραμετροποιήσιμο, δεν μπορεί να είναι και ανεξάρτητο από τον τομέα εφαρμογής, λόγω της πολύ μεγάλης ετερογένειας που υπάρχει.

Το προτεινόμενο σύστημα εμπεριέχει πολλές διαφορετικές στρατηγικές διαδικασιών συσχέτισης, οι οποίες μπορούν να ταξινομηθούν σε στρατηγικές που εμπεριέχουν συσχετίσεις σχήματος και στρατηγικές που εμπεριέχουν συσχετίσεις στιγμιότυπων.

Οι στρατηγικές συσχέτισης σχήματος αξιοποιούν λεξικογραφικές ομοιότητες, σημασιολογικές ομοιότητες και συντακτικές ομοιότητες του σχήματος των οντοτήτων. Όσον αφορά τις στρατηγικές συσχέτισης στιγμιότυπων, αυτές δεν στηρίζονται μόνο στη λεξικογραφική ομοιότητα και στην σύγκριση όλων των ζευγών των στιγμιότυπων. Οι στρατηγικές συσχέτισης στιγμιότυπων χρησιμοποιούν (1) την πλούσια σημασιολογική γνώση που αντλούμε από την έξοδο της διαδικασίας συσχέτισης σχήματος, (2) την άρρητη γνώση των χρηστών από τον
(ημι-)αυτόματο διαχωρισμό των σημαντικών χαρακτηριστικών ενός στιγμιότυπου και (3) τον υπολογισμό της πιθανότητας της αλήθειας του αποτελέσματος. Με την μόχλευση όλων των παραπάνω μπορούμε με ακρίβεια να συμπεράνουμε, αν δυο στιγμιότυπα οντολογιών αντιπροσωπεύουν την ίδια οντότητα.

Επόπτης Καθηγητής:
Δρ. Πλεξουσάκης Δημήτρης
Καθηγητής Τμήματος Επιστήμης Υπολογιστών
Πανεπιστημίου Κρήτης
Ευχαριστίες

Σε αυτό εδώ το μικρό μέρος της μεταπτυχιακής μου εργασίας, μου δίνεται η ευκαιρία να ευχαριστήσω όλους όσους συνέβαλαν για την πραγματοποίηση της.

Θα ήθελα από τον υπεύθυνο καθηγητή μου, κ. Πλεξουσάκη Δημήτρη. Θα ήθελα τον ευχαριστήσω θερμά όχι μόνο για την υποστήριξη, την σωστή καθοδήγηση, τις πολύτιμες συμβουλές και τον χρόνο που μου αφήνετε όποτε χρειαζόμουν βοήθεια, αλλά κυρίως για την εμπιστοσύνη που έδειχνε στο πρόσωπο μου καθ’ όλη την διάρκεια των σπουδών μου, η οποία πάντα μου έδινε ώθηση, ακόμη και στις πιο δύσκολες στιγμές, να συνεχίσω να προσπαθώ.

Θα ήθελα επίσης να ευχαριστήσω πολύ τον Χάρη Κονδυλάκη. Χωρίς την συμπαράστασή του και την βοήθειά του δεν θα υπήρχε η εργασία αυτή. Στην συνέχεια, θα ήθελα να ευχαριστήσω τον ερευνητή του Ινστιτούτου Πληροφορικής κ.Μάρτιν Ντερ για τις ενδιαφέρουσες συζήτησεις μας σχετικά με την εργασία αλλά και για τις καιρικές συμβουλές του.

Αγαπημένη μου φράση είναι ότι «τίποτα δεν είναι τυχαίο σε αυτή τη ζωή». Κάθε μου επίτευγμα γνωρίζω ότι δεν οφείλεται μόνο στις δικές μου προσπάθειες και δυνάμεις. Πίσω από εμένα στην προσπάθεια αυτή, αλλά και σε όλη μου τη ζωή, βρισκόταν σιωπηρά η οικογένεια μου και οι αγαπημένοι μου άνθρωποι, που σαν μια αόρατη δύναμη με σπρώχναν μπροστά να κάνω την υπέρβαση.

Ο αγαπημένος μου πατέρας, δυστυχώς έφυγε νωρίς, όμως για μένα είναι εδώ και με καθοδηγεί σωστά έκανε πάντα. Για την μητέρα μου ο,τι και να πω είναι λίγο, θα αρκεστώ μόνο σε ένα ευχαριστώ που ανά πάσα στιγμή στέκεται δίπλα μου. Ο Μανώλης, ο αδερφός μου, υπήρξε από πολύ μικρή ηλικία το πρότυπο μου, όταν μεγαλώσω να γίνω άξια σαν και αυτόν. Ο Σταμάτης, σύντροφος και συνοδοιπόρος, πάντα στοργικός, υποστηρικτικός και υπομονετικός μου υπενθύμιζε καθ’ όλη την διάρκεια του μεταπτυχιακού τις δυνάμεις μου, για να μην ξεχνιέμαι.

Τέλος θα ήθελα να ευχαριστήσω όλους μου τους φίλους και την υπόλοιπη οικογένεια μου, για την αμέριστη συμπαράστασή τους.
Στην μνήμη του μπαμπά μου Στέλλου,
στη μαμά μου Ελένη,
στον αδερφό μου Μανώλη,
και στο Σταμάτη.
List of Figures

FIGURE 1 : INSTANCE MATCHING REQUIREMENTS ................................................................. 16
FIGURE 2 : SCHEMA AND INSTANCE MATCHING SYSTEM USER INTERFACE .......... 18
FIGURE 3 : THE ASMOV MAPPING PROCESS ................................................................. 20
FIGURE 4 : THE DSSIM MAPPING PROCESS ................................................................. 25
FIGURE 5 : THE A-FLOOD INSTANCE MAPPING PROCESS ........................................... 26
FIGURE 6 : THE COMA++ INSTANCE MAPPING PROCESS ........................................... 27
FIGURE 7 : THE H-MATCH 2.0 INSTANCE MAPPING PROCESS .................................... 30
FIGURE 8 : THE SERIMI INTERLINKING PROCESS ....................................................... 33
FIGURE 9 : THE AGREEMENTMAKER INSTANCE MATCHING CONFIGURATION ....... 34
FIGURE 10 : THE ARCHITECTURE OF ZHISHILINKS ................................................... 35
FIGURE 11 : OTO SYSTEM SEQUENCE DIAGRAM ...................................................... 39
FIGURE 12 : SCHEMA AND INSTANCE MATCHING SYSTEM USER INTERFACE ....... 40
FIGURE 13 : OTO SYSTEM RELATIONAL DB SCHEMA ............................................... 41
FIGURE 15 : SCHEMA MATCHING RESULT ................................................................. 44
FIGURE 16 : CONTINUE WITH INSTANCE MATCHING BUTTON ................................... 48
FIGURE 17 : INSTANCE MATCHING ACTIVITY DIAGRAM ......................................... 49
FIGURE 18 : INSTANCE EXAMPLES ............................................................................. 55
FIGURE 19 : INSTANCE MATCHING RESULT ............................................................... 56
FIGURE 20 : DETAILED MATCHING WINDOW ............................................................. 57
FIGURE 21 : TEST CASE 1 RESULTS ........................................................................... 64
FIGURE 22 : TEST CASE 2 RESULTS ........................................................................... 64
FIGURE 23 : TEST CASE 3 RESULTS ........................................................................... 65
FIGURE 24 : TEST CASE 4 RESULTS ........................................................................... 66
FIGURE 25 : TEST CASE 5 RESULTS ........................................................................... 66
FIGURE 26 : TEST CASE 6 RESULTS ........................................................................... 67
FIGURE 27 : TEST CASE 7 RESULTS ........................................................................... 67
FIGURE 28 : TEST CASE 8 RESULTS ........................................................................... 68
FIGURE 29 : TEST CASE 9 RESULTS ........................................................................... 68
FIGURE 30 : INSTANCES EXAMPLE FROM TEST CASE 10 .......................................... 69
FIGURE 31 : TEST CASE 10 RESULTS ......................................................................... 70
FIGURE 32 : RESULTS FOR TEST CASES 1-10 ........................................................... 71
FIGURE 33 : THRESHOLD FOR TEST CASES 1-10 ....................................................... 71
FIGURE 34 : REFERENCE IN INSTANCES FROM TEST CASE 11 ............................... 72
FIGURE 35 : TEST CASE 11 RESULTS ......................................................................... 73
FIGURE 36 : REFERENCE IN INSTANCES FROM TEST CASE 12 ............................... 73
FIGURE 37 : TEST CASE 12 RESULTS ......................................................................... 74
FIGURE 38 : TEST CASE 13 RESULTS ......................................................................... 74
FIGURE 39 : TEST CASE 14 RESULTS ......................................................................... 75
FIGURE 40 : TEST CASE 15 RESULTS ......................................................................... 75
FIGURE 41 : TEST CASE 16 RESULTS ......................................................................... 76
FIGURE 42 : TEST CASE 17 RESULTS ......................................................................... 77
FIGURE 43 : TEST CASE 18 RESULTS ......................................................................... 77
FIGURE 44 : TEST CASE 19 RESULTS ......................................................................... 78
FIGURE 45 : OVERALL RESULTS FOR TEST CASES 11-19 ........................................... 79
FIGURE 46 : TEST CASE 20 RESULTS ......................................................................... 80
FIGURE 47 : TEST CASE 21 RESULTS ......................................................................... 80
FIGURE 48 : TEST CASE 22 RESULTS ......................................................................... 81
FIGURE 49 : TEST CASE 23 RESULTS ......................................................................... 81
FIGURE 50 : TEST CASE 24 RESULTS ......................................................................... 82
FIGURE 51 : TEST CASE 25 RESULTS ......................................................................... 82
# Table of Contents

**LIST OF FIGURES** ................................................................................................................. 1  
**TABLE OF CONTENTS** ......................................................................................................... 3  

## 1. INTRODUCTION ........................................................................................................ 7  
1.1. The proposed Ontology to Ontology Matching System ....................................... 7  
1.2. Thesis structure ............................................................................................................. 9  

## 2. BACKGROUND KNOWLEDGE ............................................................................. 13  
2.1. Ontologies and Instances .......................................................................................... 13  
2.2. Ontology Matching .................................................................................................... 14  
2.2.1. Schema matching ............................................................................................... 14  
2.2.2. Instance matching .............................................................................................. 14  
2.2.3. Instance matching requirements ........................................................................ 16  
2.3. Related Work ............................................................................................................. 17  
2.3.1. RiMOM .............................................................................................................. 18  
2.3.2. ASMOV (Automated Semantic Mapping of Ontologies with Validation) ...... 20  
2.3.3. FBEM (Feature Based Entity Matching) ........................................................... 22  
2.3.4. DSSim ................................................................................................................ 23  
2.3.5. Anchor-Flood ..................................................................................................... 25  
2.3.6. COMA++ ........................................................................................................... 27  
2.3.7. CODI: Combinatorial Optimization for Data Integration ................................. 28  
2.3.8. H-Match 2.0 ....................................................................................................... 29  
2.3.9. Object-Coref ...................................................................................................... 31  
2.3.10. SERIMI .......................................................................................................... 32  
2.3.11. AgreementMaker .............................................................................................. 33  
2.3.12. LN2R ................................................................................................................ 34  
2.3.13. Zhishi.links ....................................................................................................... 35  

## 3. PRESENTATION OF THE ONTOLOGY TO ONTOLOGY MATCHING PROCESS ............................................................................................................................... 39  
3.1. Loading Ontology ....................................................................................................... 40  
3.2. Schema Matching ....................................................................................................... 42  
3.3. The New Instance Matching Algorithm ...................................................................... 44  
3.3.1. User Interface Inputs .......................................................................................... 45  
3.3.2. Instance Matching Algorithm ............................................................................. 49  
3.3.3. Property weights ................................................................................................. 53
Chapter 1:

1. Introduction
   
   1.1. The proposed Ontology to Ontology matching system
   
   1.2. Thesis structure
1. Introduction

Instance matching is the process of comparing different individuals with the goal of recognizing the same real-world entity. It has been studied for over 50 years. While the majority of the existing research was done for the task of matching database records, modern approaches focus mostly on graph-based data representations extended by additional schema information. This allows us to describe it within the well-established ontology matching framework.

Ontology, as the means to conceptualize domain knowledge, has become the enabler of the fulfillment of the semantic web vision. It aims to create a platform where information has its semantics and can be understood and processed by computers themselves with minimum human interference, thus, to make data sharable.

Unfortunately, ontologies themselves are heterogeneous and distributed. Defined by different organizations or by different people in the same organization, ontologies can have vastly different characteristics and structures. Therefore, in order to achieve semantic interoperability across ontologies, it is necessary to discover the alignment across ontologies and their instances matches.

The demand for high-quality ontology instance matchings is crucial in the context of identity recognition, ontology population and semantic integration. Identity recognition is an emerging topic in the semantic web field and it refers to the capability of detecting whether two different resource descriptions refer to the same real-world entity, namely an instance. Ontology population is one of the main activities of ontology evolution, where the ontology is evolved by acquiring new semantic descriptions of data extracted from heterogeneous data sources. For ontology population, instance matching plays a crucial role to correctly perform the insertion activity and to discover the relationship between new incoming instance and the set of instances already stored in the ontology. Last, but not least, for semantic integration, advanced techniques for ontology instance matching are required to correctly combine data describing individuals in different sources and to improve the accuracy of the schema ontology alignment process.

1.1. The proposed Ontology to Ontology matching system

Several frameworks/systems for ontology-based information integration have been proposed. Their research efforts are mainly focused on the integration at the ontology schema level. For those who consider integrating data at the ontology
data level by creating or maintain data relations, just a few systems have been proposed.

Our purpose is to offer a multi-strategy ontology to ontology (OtO) matching system, for schema alignment and instance matching, which is domain independent and fully customizable from the domain expert at any level. We argue that, unless a system is fully customizable, it cannot claim be domain independent. This hypothesis is based on the vast heterogeneity of the ontologies, as discussed above.

The OtO matching system implements several different ontology matching strategies, which can be roughly categorized into the schema matching and the instance matching strategies.

The schema matching process leverages the lexical similarities, the semantic similarities and the syntactic similarities of the entities of the schema. As far as the instance matching process is concerned the OtO matching system does not only rely on lexical similarity measures and pair-wise instance comparison. It utilizes the instance matching process by leveraging (1) the rich semantic knowledge we gain from the output mappings of the schema matching process, (2) the implicit knowledge of domain expert by (semi-)automatically capturing the identification power of the properties and (3) the probability calculation of the result’s truth, in order to accurately and efficiently detect the ontology instances that represent the same real-world entity.

In our instance matching approach the domain expert can select whether the instance matching will be based on the results of the schema matching. The intuition behind this is that the schema mappings contain semantic knowledge which is important for the efficient instance matching. If for example the schema matching returns a mapping between the “person” entity from ontology A and the “sportperson” entity from Ontology B, then it is wise just to compare the instances of type “person” from ontology A, with the instances of type “sportperson” from ontology B, and not to compare all the instances from ontology A with all the instances from ontology B.

Furthermore, the OtO system uses the keyword rules in order to automatically detect the significant properties and assign them high property weights. By capturing the identification power of the properties, the OtO system can reduce the unnecessary comparisons of the property instances, and only compare the properties of the instances that can determine the identity recognition.

Moreover, the OtO system uses two different filters to determine whether a match is considered as confident or not. The first is the threshold filter. If the average similarity of an instance match is below the threshold then it is not considered as a match, otherwise it is. The second filter is the probability of an instance match to be a true match. Through the use of the property weights, the OtO
system can calculate the probability of an instance match to be true or not. If the probability is very low then the match is not returned. All these features and processes of the OtO matching system are analyzed in detail in section “3. Presentation of the Ontology to Ontology Matching Process”.

Finally, we have evaluated the OtO matching system by using the ISLab Instance Matching Benchmark (IIMB) [12]. The IIMB was introduced in the Ontology Alignment Evaluation Initiative 2009 campaign. The results were successful, since they showed that the OtO matching system returned 0.97 precision, 1 recall and 0.98 F-measure. These results rank the OtO matching system in the first positions among all the other systems that participated in the benchmark. All the test case evaluations are presented in Chapter "4. Experiments and Results”.

1.2. Thesis structure

This Thesis is organized as follows:

Chapter 2 presents the Knowledge Bases involved in the Ontology matching, i.e the schema matching and the instance matching. Furthermore, it analyzes the instance matching requirements. Finally, it reviews the related scientific work done within the past years, as well as the related ontology instance matching frameworks that exist.

In Chapter 3 the process of the OtO matching system is described. All the process stages, from the ontology loading, to the schema matching and the instance matching are analyzed and explained in detail.

Chapter 4 deals with the experiments and their results. It discusses all the outcomes returned by running the 37 test cases from the OAEI 2009 IIMB Instance Benchmark, in comparison with the other instance matching frameworks that took part in the same Benchmark.

The OtO system usage in the EU Project “PlugIT” is analyzed in Chapter 5. In particular, the “PlugIT” project is introduced, and the role of the OtO matching system is explained. Moreover, the customizations that have occurred in the OtO system, for making the OtO system fit in the project, are explained.

Finally, in chapter 6, the strengths and the weaknesses of the OtO matching systems are summarized and the conclusions are drawn.
Chapter 2:

2. Knowledge Base

2.1. Ontologies and Instances

2.2. Ontology Matching
   2.2.1. Schema matching
   2.2.2. Instance matching
   2.2.3. Instance matching requirements

2.3. Related work
   2.3.1. RiMOM
   2.3.2. ASMOV
   2.3.3. FBEM (Feature Based Entity Matching)
   2.3.4. DSSim
   2.3.5. Anchor-Flood
   2.3.6. COMA ++
   2.3.7. CODI: Combinatorial Optimization for Data Integration
   2.3.8. H-Match 2.0
   2.3.9. Object-Coref
   2.3.10. SERIMI
   2.3.11. AgreementMaker
   2.3.12. LN2R
   2.3.13. Zhishi.links
Chapter 2
2. Background Knowledge

2.1. Ontologies and Instances

In this section, we give the definitions related to ontologies and ontology instances.

Ontology is a formal specification of a shared conceptualization [65]. In [35] an ontology is described as a 6-tuple:

\[ O = \{ C, P, H^C, H^P, A^O, I \} \]

where \( C \) and \( P \) are the sets of concepts and properties, respectively. \( H^C \) defines the hierarchical relationships:

\[ H^C \subseteq C \times C. (c_i, c_j) \in H^C \]

denotes that concept \( c_i \) is the subconcept of \( c_j \). Similarly, \( H^P \) defines the hierarchical relationships between each property and its subproperties:

\[ H^P \subseteq P \times P \]

\( A^O \) is a set of axioms and \( I \) is a set of instances of concepts and properties.

Ontologies can also be expressed in description logics (DL) [66], which are a well-known family of knowledge representation mechanism and have been studied for several years. An ontology can be regarded as a typical DL knowledge base that consists of two components: a “TBox” and a “ABox”. Concepts and their relations of the ontology are defined in the TBox as a set of asserted axioms. Individuals (i.e., instances of concepts) are contained in the ABox.

The Web, World Wide Web Consortium (W3C) has proposed a DL-based web ontology language: the OWL [67]. It provides a set of vocabulary as constructs, enabling people to define concepts, properties, individuals, and their relations. Typically, property in OWL has two categories: data type property and object property. Data type properties allow people to describe specific attributes of a concept, such as “age of person”, “address of company”. Meanwhile, object properties enable people to link two concepts with a semantic relation, like “supervise” between “professor” and “student”.

Corresponding to the notions of TBox and ABox in DL, ontology encoded in OWL can also be partitioned into two parts: ontology schema and ontology data. Definitions of concepts, properties and their relations in the owl file(s) are treated as
ontology schema. Instances of these concepts, or individuals, are treated as ontology data.

2.2. Ontology Matching

Ontology matching [68] aim is to (semi-)automatically detect semantic correspondences between heterogeneous ontologies. It is referred to as a means for resolving the problem of semantic heterogeneity. It can be performed at two different levels: schema matching and instance matching.

2.2.1. Schema matching

Schema matching takes two ontologies as input and determines as the output the alignment result between entities of the input ontologies. Given two ontologies O₁ and O₂, a matching (or alignment) finds, for each entity in O₁, a corresponding entity in O₂. O₁ is called the source ontology and O₂ the target ontology. In our schema matching we deal with 1:n matching, i.e. for an entity in the source ontology, find one or more entities in the target ontology. Further, we do not differentiate between ontology alignment and ontology matching.

We formally define an ontology matching result as

\[ M = \{ O , O' , M \} \]

Where \( M = \{ m₁, m₂, m₃, ..., mₙ \} \) denotes a set of mappings. A mapping \( mᵢ \) can be written as \( mᵢ = (e, e', simᵢ) \) where \( e \) and \( e' \) are two entities and \( simᵢ ∈ [0,1] \) denotes the similarity between them.

Often the result can also contain the relation \( r \) between the \( e \) and \( e' \). The relation \( r \) is taken from the set \( \{ ≡, ⊑, ∩, ⊥ \} \), resp. for equivalence, subsumption, overlap and disjointness. In an application about thesauri [69], relations “similar to”, “broader than”, “narrower than”, and even the “related to” relation might also be considered.

2.2.2. Instance matching

In contrast to schema-level integration, instance matching deals with the actual individuals, not with integration of class structures or entity types. Instance matching has to deal with deciding whether two entity descriptions refer to the same individual. The semantic relation between instances is determined based on the overlap (similarity) of their instance annotations. Thus, it crucially depends on measuring the similarity between sets of annotated instances.

Evangelia Daskalaki

Master Thesis | University of Crete | Computer Science
Chapter 2

As stated in [70] the application of instance matching is crucial in the following context:

- **Identity recognition.** Identity recognition is an emerging topic in the Semantic Web field and it refers to the capability of detecting whether two different resource descriptions refer to the same real-world entity, namely an individual. As discussed in [71], in developing any kind of semantic-driven information system possibly based on knowledge exchange and reuse, the phases of ontology conceptualization and population need to be clearly distinguished. Such a distinction is motivated by the observation that the same set of individuals can be used to populate different ontologies also in different domains.

- **Ontology population.** In modern Semantic Web applications, ontology evolution is becoming more and more important due to the need of supporting experts in managing ontology changes through advanced, and possibly automated, techniques[72] [73]. One of the main activities in ontology evolution is ontology population, where the ontology is evolved by acquiring new semantic descriptions of data extracted from heterogeneous data sources. For ontology population, instance matching plays a crucial role to correctly perform the insertion activity and to discover the relationship between new incoming instance and the set of instances already stored in the ontology. In this respect, instance matching has the role of providing a set of semantic mappings between incoming instances and those already stored. The mappings produced by instance matching are exploited to cluster those instances that are recognized as referring to the same real-world entity.

- **Semantic integration.** Due to the increasing popularity of Semantic Web technologies, a novel attention on semantic integration issues has raised. In this respect research effort has been focused on ontology matching and alignment with the aim to enforce advanced techniques for (semi)automatically discovering semantic mappings between possibly distributed and heterogenous ontologies. For semantic integration, advanced techniques for ontology instance matching are required to correctly combine data describing individuals in different sources and to improve the accuracy of the schema ontology alignment process.

In our work we focus on the identity recognition (or instance matching) and if formally described as in [74]:

Evangelia Daskalaki
Master Thesis | University of Crete | Computer Science
Chapter 2

Given two instances $i_1$ and $i_2$, belonging to the same ontology or to different ontologies, instance matching is defined as a function $I_m(i_1; i_2) \rightarrow \{0;1\}$, where 1 denotes the fact that $i_1$ and $i_2$ are referred to the same real-world object and 0 denotes the fact that $i_1$ and $i_2$ are referred to different objects.

2.2.3. Instance matching requirements

In order to find out properly if two individuals are referred to the same real-world object, an instance matching algorithm should satisfy different kinds of requirements. As shown in Figure 1, those can be divided in three main categories [74]:

- **Data value differences.** An instance matching algorithm is required to recognize, as better as possible, corresponding values, even if data contain errors or are represented using different standard formats. This issue has been addressed in the field of record linkage research, and the problem of comparing instances' property values is the same as comparing records' attribute values.

- **Structural heterogeneity.** Instances belonging to different ontologies can not only differ within their properties values, but they can also have different structures. While in record linkage the structure of records is usually given and schema and record matching are different...

![Figure 1: Instance matching requirements](image-url)
problems, in instance matching, schema and instances are more strictly related. Thus, besides the capability to evaluate the level of similarity between property values, instance matching techniques have to go beyond heterogeneous individual representations by identifying the pairs of matching properties between two considered instances.

- **Logical heterogeneity.** A specific ontology matching problem, which is not taken into consideration in record linkage process, is the need to infer implicit knowledge, typically referred to concepts hierarchy within the ontologies.

### 2.3. Related work

The problem of instance matching has been a topic of research for more than 50 years. It is also known as the problem of record linkage [75] [81], Duplicate detection [76], entity resolution [77], merge/purge [89], and object reconciliation or identification [78]. In [80] the authors refer to instances as reference knowledge and the result of the matching process aims for data integration. This last work is focused on the idea of establishing co-reference information and preserving it, in order to build large scale networks of knowledge. Data integration and methods are also discussed in [82] and [83].

While the majority of the existing methods were developed for the task of matching database records [79], modern approaches focus mostly on graph-based data representations extended by additional schema information. This allows us to describe it within the well-established ontology matching framework [68].

The ontology instance matching problem is seen in the literature from two different perspectives. In the first perspective the instance matching results is the means to derive to more accurate schema matching results [84] [85] [86].The notion behind this perspective is that the more significant the overlap of common instances of two ontology concepts is, the more related these concepts are.

On the other hand, the second perspective uses the outcomes of the ontology schema matching in order to utilize the instance matching process [87]. The notion behind the last perspective is that if two ontology concepts are semantically correlated then it is likely that their instances refer to the same individuals, but if the concepts are not related then their instances will also not be related to each other.

In our work we follow the second perspective in order to derive to an instance mapping result by taking into consideration the schema matching result. In the next sections other ontology instance Frameworks are discussed.
2.3.1. RiMOM

RiMOM [34] is a multiple strategy ontology alignment framework based on risk minimization of Bayesian decision. Given two input ontologies at runtime, RiMOM determines which ontology alignment methods to be used, what kinds of information to use in the similarity calculation and how to combine multiple methods as necessary. Figure 2 shows the process flow of RiMOM.

- **Preprocessing.** Given two ontologies, RiMOM generates the description for each entity. Then, it calculates the label similarity factor and the structure similarity factor, which will be used in the following steps.

- **Linguistic-based ontology alignment.** In this step, multiple linguistic-based strategies are executed. Each strategy uses different
ontological information and obtains a similarity result for each entity pair. These strategies will be dynamically selected to be included in different alignment tasks.

- **Similarity combination.** This step combines the similarity results obtained by the selected strategies. The weights in the combination are determined by the label similarity factor and the structure similarity factor.
- **Similarity propagation.** This step considers structural similarity. We use three similarity propagation strategies, namely, Concept-to-Concept, Property-to-Property, and Concept-to-Property.
- **Alignment generation and refinement.** This step fine tunes and outputs the alignment result.

RiMOM adopts a strategy selection method based on the definition of three ontology feature factors: label similarity, structure similarity, and label meaning. These factors are estimated based on the two ontologies to be matched. The matching strategies to be used are those that are suited to the highest factors. For example, if the two ontologies have high label similarity factor, then RiMOM will mostly rely on linguistic based strategies; while if the two ontologies have a high structure similarity factor, it will employ similarity-propagation based strategies on them. However, we note that the association between factors and strategies is predefined. Multiple results are combined using the weighted average of their similarity values, where the weights are predefined experimentally. The OTO system also provides a strategy selection method. However, in contrast with the RiMOM system, the weights are equally assigned. In terms of the final selection of mappings, RiMOM uses a similarity threshold value, while uses additionally to the similarity threshold, the probability of the similarity to be a true match.

For the Instance Matching RiMOM builds two vectors referred to as Name Vector and Virtual Document for each instance. The Name Vector is constructed by accumulating the terms in the Name property values and setting the occurrence of each term as its weight. For Virtual Document, RiMOM first collects the terms of the each instance’s descriptions and annotations then fetch the local information of its neighboring instances to construct a comprehensive vector. The tf-idf value of each term is calculated as its weight. The similarity between two instances is calculated as the weighted sum of their similarity (Cosine Distance) on two kinds of vectors respectively. The difference between the OTO system and the RiMOM is that the OTO system does not produce Vectors for the properties and that the weights of the properties depend on the significance of the property and not on the occurrence of it.

The OTO matching system appears to perform slightly better than RiMOM in the OAEI 2009 IIMB Instance Benchmark [17] which is analyzed in section “5. Experiments and Results”. Particularly the H-mean of the RiMOM precision is 0.96
while for the OtO matching system is 0.97, and the H-mean of the RiMOM recall is 0.98 while for the OtO system is 1.

2.3.2. ASMOV (Automated Semantic Mapping of Ontologies with Validation)

The ASMOV [36] [37] algorithm iteratively calculates the similarity between entities for a pair of ontologies by analyzing four features: the lexical information (measures lexical similarity including identifiers, labels, and descriptions), the internal structure (measures internal structure similarity, that is, similarity between properties of classes and between the domain and range of properties), the external structure (including relations between the parents and the children of classes or properties) and the individuals (that is, the similarity between the sets of class members). All these features are also analyzed from the OtO matching system.

The measures obtained by comparing these four features in ASMOV are combined into a single value using a weighted sum. In the OtO system a weighted sum is only applicable in the instance matching process, while the schema matching is put in another matching category.

![Figure 3: The ASMOV Mapping process](image)

Figure 3 illustrates the ASMOV mapping process, which has been implemented in Java. In the pre-processing phase, the ontologies are loaded into memory using the Jena ARP parser [3] and ASMOV’s ontology modeling component.
Chapter 2

A thesaurus is optionally used to calculate the lexical similarities between each pair of concepts, properties and individuals. ASMOV can be configured to use either the UMLS Metathesaurus [40] or WordNet [32] [33] in order to derive the similarity measures, while in the OtO matching system the WordNet Thesauri is always used in the lexical schema matching process. If a thesaurus is not used in ASMOV, a text matching algorithm is used to compute the lexical distance. The similarities between pairs of entities along the relational structure, internal structure, and extensional dimensions are calculated, and overall similarity measures (or confidence values) are calculated for each pair. From these similarity measures, a pre-alignment is obtained by selecting the entity from one ontology with the highest similarity for a corresponding entity in the other ontology. A threshold of 0.1% is used to ignore spurious non-zero similarity measures.

This pre-alignment then goes through semantic verification, which detects semantically inconsistent mappings and their causes. These inconsistent mappings are removed from the pre-alignment and logged so that the algorithm does not attempt to map the same entities in a subsequent iteration; mappings are removed from the log of inconsistencies when the underlying cause disappears.

As far as the Individual Similarity is concerned ASMOV does not rely on individuals to find the similarity between concepts and properties; however, if individuals are present, the similarity between them is exploited in order to refine the mapping. Two individuals are said to be similar if their internal structures are the same and the corresponding literal values match. If both of these requirements are met for one or more individuals of the source and target entities, then the similarity measure between these entities is set to 1. Moreover, no additional calculations are performed between those entities. Not like to the ASMOV system, the OtO system considers the schema matching and the instance matching as two different strategies where the schema matching supports the instance matching process.

The individual similarity between properties is more complex. As the concepts’ individuals are analyzed, their internal structures will highlight the similarity between properties. In some cases, individuals may have missing properties; these missing properties are analyzed in order to update a list of possible matches between properties. If the possible matches between the source properties and target properties are a one-to-one mapping, then the similarity between those properties are also set to 1; on the other hand, the similarity measure is the probability calculation. Thus, the individual similarity between properties is dictated by the number of occurrences of the target property within the possible matches and the total number of items in the list of possible matches.

The ASMOV system presented above, conducts the instance matching by matching the entire internal structures of every source instance with another target instance. In contrast, the OtO system does not match the entire internal structure,
but only those properties of the structure that are considered as significant properties. Thus the matching process is more efficient in the OtO system. Furthermore, the ASMOV system has the philosophy to conduct instance matching in order to refine the schema matching. On the other hand the OtO system has the philosophy of conducting schema matching in order to utilize the instance matching process. Finally, the OAEI 2009 IIMB Instance Benchmark results showed that the ASMOV performed very well. Particularly the H-mean of the ASMOV precision is 1 while for the OtO matching system is 0.97, and the H-mean of the ASMOV recall is 0.98 while for the OtO system is 1.

2.3.3. FBEM (Feature Based Entity Matching)

The FBEM [41] [42] combines probabilistic as well as ontological methods for deciding whether two records describe the same entity, taking into account intensionsal and extensional aspects of the entities at hand. This approach aims at general-purpose usefulness with special focus on web information systems, and bases on empirical findings about what are the commonly used entity types, and how they are usually described.

Both a reference (matching) entity \(Q\) and candidate (matched) entity \(E\) are considered as a set \(F\) of features \(f\) where each feature \(f\) is a pair of name and value.

All features of any particular entity are enumerated with integer values and denote as \(f_i^Q, f_j^E\) the \(i\)th and \(j\)th features of entities \(Q\) and \(E\) respectively. The following functions are defined:

\[ n(f_i): \text{returns the name part of a feature of an entity.} \]
\[ v(f_i): \text{returns the value part.} \]
\[ f_{i,j} \text{sim}(f_i^Q, f_j^E), \text{a function that computes the similarity of two features} f_i^Q, f_j^E \text{ as follows:} \]
\[ f_{i,j} \text{sim}(f_i^Q, f_j^E) = \begin{cases} 2 \times \lambda \times \mu, & \text{for name}(n(f_i^Q)), name(n(f_j^E)), id(f_i^Q, f_j^E); \\ 2 \times \mu, & \text{for name}(n(f_i^Q)), name(n(f_j^E)); \\ \lambda \times \mu, & \text{for name}(n(f_j^E)), id(f_i^Q, f_j^E); \\ \mu, & \text{for name}(n(f_j^E)); \\ 1, & \text{otherwise}. \end{cases} \]

Equation (1) relies on the following functions and parameters:

\[ \text{sim}(x, y): \text{a suitable string similarity measure between} x \text{ and} y. \]
name(x) : a boolean function indicating whether the feature x denotes one of the possible names of the entity.

id(x, y) : the identity function, returning true if value parts of x and y are identical.

$\mu$ : the factor to which a name feature is considered more important than a non-name feature

$\lambda$: the extra-factor attributed to the presence of the value identity id(x, y).

The Levenstein metric is selected as a similarity measure, and both $\lambda$, and $\mu$ equal to 2.

Similar to the OtO matching system the FBEM implements a vocabulary, small enough to be maintained in a runtime memory, that is used to detect the cases where entity feature name is actually a “name” of the entity, e.g., “name”, “label”, “title”, “denomination”, “moniker”. The difference in the OtO matching system is that the features can also contain the customized weight.

The feature-based entity similarity score is defined as the sum of all the maximum similar feature combinations between Q and E.

Although the FBEM uses the same guideline as the OtO system, the OAEI 2009 IIMB Instance Benchmark results showed that the FBEM did not perform well. Particularly the H-mean of the FBEM precision is 0.16 while for the OtO matching system is 0.97, and the H-mean of the FBEM recall is 0.75 while for the OtO system is 1.

2.3.4. DSSim

DSSim is a multiagent ontology Mapping Framework [43] [44]. Its objective is to produce a method that does not depend on any fine tuned internal parameters for a specific domain or does not assume having large amount of data samples a-priori for machine learning or Bayesian probability assessment. Their hypothesis is that the correctness of different similarity mapping algorithms is always heavily dependent on the actual content and conceptual structure of these ontologies which are different even if two ontologies have been created on the same domain but with different purpose.

DSSim ontology mapping system is based on multiagent architecture [16] where each agent built up a belief for the correctness of a particular mapping hypothesis. Their beliefs are then combined into a more coherent view in order to provide better mappings. DSSim has recognized the need for possible parallel processing for tracks which contain large ontologies e.g. very large cross-lingual resources track. The need of using distributed multi-agent architecture, which is required for scalability purposes once the size of the ontology is increasing. DSSim
mapping process can utilize multi core processors by splitting up the large ontologies into smaller fragments. Both the fragment size and the number of cores that should be used for processing can be set in the “param.xml” file. The process works as follows:

1. Based on the initial parameters divide the large ontologies into n*m fragments.
2. Parse the ontology fragments and submit them into the alignment job queue.
3. Run the job scheduler as long as we have jobs n the queue and assign jobs into idle processor cores.
   3.1. It takes a concept or property from ontology 1 and considers (refer to it from now) it as the query fragment that would normally be posed by a user. DSSim algorithm consults WordNet in order to augment the query concepts and properties with their hypernyms.
   3.2. It takes syntactically similar concepts and properties to the query graph from ontology 2 and build a local ontology graph that contains both concepts and properties together with the close context of the local ontology fragments.
   3.3. Different similarity and semantic similarity algorithms (considered as different experts in evidence theory) are used to assess quantitative similarity values (converted into belief mass function) between the nodes of the query and ontology fragment which is considered as an uncertain and subjective assessment.
   3.4. The similarity matrices are used to determine belief mass functions which are combined using the Dempster’s rule of combination. Based on the combined evidences we select those mappings in which we calculate the highest belief function.
4. The selected mappings are added into the alignment.

The overview of the mapping process is depicted in Figure 4:
The performance of FBEM in the OAEI 2009 IIMB Instance Benchmark showed that the FBEM did not perform well. Particularly the H-mean of the FBEM precision is 0.16 while for the OtO matching system is 0.97, and the H-mean of the FBEM recall is 0.75 while for the OtO system is 1. This can be due to the fact that FBEM was originally conceived as schema mapping system that does not use extensively instances for establishing the mapping.

2.3.5. Anchor-Flood

The A-Flood [45] is a scalable ontology alignment system, which aligns ontology schemas and instances.

Ontology schema matching starts off an anchor. It has a process of collecting small blocks of concepts and related properties dynamically by considering super-concept, sub-concept, siblings and few other neighbors from the anchor point. The size of blocks affect the running time adversely. Therefore, A-Flood incorporates semantic similarity considers intrinsic Information Content (IC) for building blocks of neighboring concepts from anchor-concepts. Local alignment process aligns concepts and their related properties based on lexical information, and structural relations.
Retrieved aligned pairs are considered as anchors for further processing. The process is repeated until there is no more aligned pair to be processed. Hence, it bursts out with a pair of aligned fragment of the ontologies, giving the taste of segmentation. Multiple anchors from different part of ontologies confirm a fair collection of aligned pairs as a whole.

For the Ontology instance Matching, A-Flood considers the semantically linked information that includes linked concepts, properties and their values and other instances as well. They all together form an information cloud to specify the meaning of that particular instance. A-Flood refers to this collective information of association as Semantic Link cloud. The degree of certainty is proportional to the number of semantic link associated to a particular instance by means of property values and other instances. First, a pair of TBox is aligned with the A-Flood algorithm. That means that the A-Flood, like the OtO uses the schema matching results as input for the instance matching. Then, the alignment of the type of an instance to any concept of the neighbors of the type of another instance across ABox is checked. Then the structural similarity among the elements available in a pair of clouds to produce instance alignment is checked. The instance matching algorithm is depicted in Figure 5.

---

**Alg. InstanceMatch** (ABox \(ab_1\), ABox \(ab_2\), Alignment \(A\))

1. for each \(ins_i \in ab_1\)
2. \(cloud_i = makeCloud(ins_i, ab_1)\)
3. for each \(ins_j \in ab_2\)
4. \(cloud_j = makeCloud(ins_j, ab_2)\)
5. if \(\exists a(c_1, c_2) \in A|c_1 \in Block(ins_{1_{type}}) \land c_2 \in Block(ins_{2_{type}})\)
6. \(Sim_{struct}(cloud_i, cloud_j) \geq \delta\)
7. \(imatch = imatch \cup makeAlign(ins_i, ins_j)\)

---

**Figure 5 : The A-Flood instance mapping process**

From the algorithm above it is obvious that the A-Flood does not exploit the weights to differentiate the significance of the properties of each instance, like the OtO system, but runs the comparisons by using all the instance properties. Thus the algorithm is less efficient than the OtO system.

The performance of A-Flood in the OAEI 2009 IIMB Instance Benchmark showed that the system did perform quite well, but not as good as the OtO system. Particularly the H-mean of the A-Flood precision is 0.92 while for the OtO matching system is 0.97, and the H-mean of the FBEM recall is 0.87 while for the OtO system is 1.
2.3.6. COMA ++

COMA [46] [47] is a generic match system that has been developed at the University of Leipzig. The prototype provides a large spectrum of matchers that can be combined in a flexible way. Figure 6 illustrates how the instance-based matchers can be applied for schema matching in COMA++. First schemas and the instance data have to be parsed. Different parsers support different sources, e.g. relational databases and XML files. The parsed instance data are assigned to the generic schema representation. In the following match process matchers are executed. The instance-based matchers are a constraint-based and a content-based matcher. Each matcher generates as result a similarity matrix. This matrix contains pair-wise similarity values for the schema elements. A propagation algorithm is applied on the results of the instance-based matchers to transfer similarities from elements to their surrounding elements. Finally the similarity matrices are combined to derive a mapping.

Before the matching process the constrain based matcher determines the constraints that describe characteristics or patterns of element values. The constraints are assigned to the elements. During the match process these constraints are compared and their similarity determines the similarity of the matched elements. In case of huge sets of instance data this approach has a low effort because rather than comparing the instances, the constraints are compared. The COMA++ distinguishes between the General constraints, the Numerical constraints, and the Pattern constraints.

In the match process the constraint-based matcher determines the similarity of two elements by comparing their previously identified constraints. The matcher uses a synonym table specifying the degree of compatibility between numerical respectively pattern constraints. Additionally, the similarity increases if the average length is alike.

The content-based matcher determines the similarity of two elements by executing a pair-wise comparison of instance values using a similarity function.
result is a similarity matrix with each dimension representing the instances of one element. This matrix is aggregated to one value that defines the similarity of the instance sets and thus the elements. This aggregation is done by applying the following formula where n is the number of instances of e1 and m is the number of instances of e2 and sim is the used string similarity function:

\[
\text{similarity}(e_1, e_2) = \frac{\sum_{i=1}^{n} \max_{j=1..m}(\text{sim}(\text{inst}_{e_1}, \text{inst}_{e_2})) + \sum_{j=1}^{m} \max_{i=1..n}(\text{sim}(\text{inst}_{e_1}, \text{inst}_{e_2}))}{n + m}
\]

The formula uses for every instance of e1 the highest similarity to an instance of e2 and vice versa. These maxima are added and the number of all instances divides the resulting sum.

In COMA++, as in the A-Flood system, the algorithm does not exploit the weights to differentiate the significance of the properties of each instance, like the OtO system does. It conducts the comparisons by using all the instance properties. Thus the algorithm is less efficient than the OtO system’s.

Finally COMA++ did not participate in any Instance-based Benchmark so we don’t have any results to compare with the OtO system.

### 2.3.7. CODI: Combinatorial Optimization for Data Integration

CODI [50] [51] is a probabilistic –logical alignment system. The system provides a declarative framework for the alignment of individuals, concepts, and properties of two heterogeneous ontologies. CODI leverages both logical schema information and lexical similarity measures with well-defined semantics for A-Box and T-Box matching.

CODI is based on the syntax and semantics of Markov logic [48] and transforms the alignment problem to a maximum-a-posteriori optimization problem. This problem needs a-priori confidence values for each matching hypotheses as input. Therefore, CODI implemented an aggregation method of different similarity measures.

As far as instance matching is concerned, CODI implements an approach which utilizes object-properties to determine the instances for which the similarity should be computed. This approach assumes that the TBoxes are one common and two different ABoxes. Consequently, both TBoxes have been integrated beforehand. In a first step the anchor-alignments are computed. Therefore, a small subset of all individuals is computed with each other, compute their lexical similarities, and add
those to the anchor-alignments if their respective similarities are above a threshold.
Then, the first anchor-alignment is taken. For all individuals which are connected
with an object-property-assertion with one of the individuals in the anchor-
alignment again the lexical similarity is computed. They are added to the end of the
anchor-alignments if lexical similarities are higher than the threshold. The anchor-
alignment is a unique set, which means that only new alignments are added. This
procedure is repeated for all following anchor-alignments until the whole set is
through.

The first drawback of the CODI system in comparison with the OtO system is
that instance matching is not applicable when the TBoxes (schemas) are not the
same. Secondly the CODI does not differentiate the instance properties between
significant or not significant as the OtO system does with the use of weights.

2.3.8. H-Match 2.0

HMatch 2.0 [52] [53] is designed as a comprehensive framework for ontology
matchmaking where a number of ontology matching components with different
specialization can be used alone or in combination according to the considered
matching scenario. H-match is build with a modular architecture where each
matching component addresses a specific task and interacts with the other modules
through appropriate interfaces. The HMatchController is responsible for managing
the HMatch 2.0 configuration by selecting the matching components to invoke and
by supervising the execution of the overall matching process. HMatch(L), HMatch(C),
and HMatch(S) provide concept-level matching techniques and each of them is
devoted to enforce a specific pool of matching techniques, namely linguistic,
contextual, and structural matching techniques. Moreover, HMatch(I) is defined to
provide instance-level matching techniques for the similarity evaluation of ontology
instances. Functionalities for manipulating and merging mappings produced by the
various components are also available through the HMatch(M) component.

Mapping operations such as intersection, union, product, and transitive
closure are defined to this end. The Mapping- Manager is responsible for managing
mapping storage, release, and post-processing when required by the other HMatch
2.0 components. Furthermore, HMatch 2.0 includes the HMatch(V) component to
support mapping validation and inference related to the capability of checking the
consistency of the produced mapping sets and to the opportunity of inferring new
mappings through reasoning, respectively. As a further basic feature, HMatch 2.0 is
designed to interface an external reasoning service to support the matching
components (i.e., HMatch(I), HMatch(V)) when required.
In the following, we focus on the instance matching techniques enforced in HMatch 2.0. A detailed description of the instance matching process of HMatch(I) is shown in Figure 7.

Figure 7: The H-Match 2.0 instance mapping process

The matching process starts with the acquisition of the ontology instances to compare in form of ABoxes. After acquisition, a pre-processing stage is performed to tailor the behavior of HMatch(I) according to the tasks that need to be activated. With concept-level mappings acquisition, we mean that concept matching components, namely HMatch(L) and HMatch(C), can be (optionally) invoked when the considered instances are extracted by two different ontologies with heterogeneous TBox schemas. As implicit knowledge exploitation, H-Match 2.0 refers to the (optional) use of implicit knowledge for instance matching. In particular, if the logic-based approach is selected, an instance pre-processing task based on the interaction with an external reasoning service is executed. Finally, as featuring
properties identification, H-Match enforces the choice between the auto-id and the interactive-id modality for property weight specification.

The subsequent matching stage is based on the comparison of instance properties, also called roles, and corresponding property values, also called role fillers. To this end, each instance in HMatch(I) is represented as a tree (instance tree construction) where role fillers are nodes and roles are labeled edges. Matching is then performed by traversing in postorder the instance trees of the two instances and by recursively executing filler similarity or instance similarity on corresponding nodes according to their type. When nodes are datatype values, their similarity is computed by relying on specific matching functions that vary according to the considered datatype (filler similarity evaluation). When nodes are instances, the similarity is evaluated by considering the matching value of all the nodes in their respective sub-trees previously traversed (instance similarity evaluation). The set of resulting instance mappings are finally returned.

From the description of the instance matching process of the H-Match we conclude that it is very similar to the OtO matching system, where both schema matchings and instance property weights are exploited. Still, our system performed better in the OAEI 2009 IIMB Instance Benchmark. This is probably because the OtO system conducts extra filtering in the matching instance results depending on their probability to be an accurate instance match.

### 2.3.9. Object-Coref

ObjectCoref [54] is a Tool for bootstrapping object co-reference resolution on the Semantic Web. The architecture of the approach follows a common self-training framework. Self-training is a major kind of semi-supervised learning, which assumes that there are abundant unlabeled examples in the real world, but the number of labeled training examples is limited.

ObjectCoref builds an initial set of co-referent URIs mandated by the formal and explicit semantics of owl:sameAs, owl:InverseFunctionalProperty, owl:FunctionalProperty, owl:cardinality and owl:maxCardinality.

The semantics of owl:sameAs dictates that all the URIs linked with this property have the same identity. if a property is declared to be inverse functional (IFP), then the object of each property statement uniquely determines the subject (some individual); a functional property (FP) is a property that can have only one unique value for each object; while cardinality (or max-cardinality) allows the specification of exactly (or at most) the number of elements in a relation, in the context of a particular class description, and when the number equals 1, it is somehow similar to the FP, but only applied to this particular class.
Next, ObjectCoref learns the discriminability of pairs of properties based on the co-referent URIs, in order to find more co-referent URIs for extending the training set. The discriminability reflects how well each pair of properties can be used to determine whether two URIs are co-referent or not.

In RDF graphs, each URI is involved in a number of RDF triples whose subject is the URI, and the predicates and objects in these RDF triples form some <property, value> pairs, which can be considered as features for describing such URI. ObjectCoref compares the values between the <property, value> pairs from coreferent URIs, and finds which two properties have similar values and how frequent. The significance is the percentage of the number of coreferent URIs that can found by the discriminant properties in all the coreferent URIs in the training set. If the significance is greater than a given threshold, such the property pair is chosen for further resolution. This is similar to the property weights used in the OTO matching system, with the difference that the weights are not proportional to the frequency of the values, but depends on implicit knowledge of the domain expert.

If new co-referent URIs are found, ObjectCoref selects the accurate ones and adds them into the training set. The whole process iterates several times and terminates when the property discriminability is not significant enough or cannot find more discriminant property pairs.

ObjectCoref does not support schema matching. For that purpose it uses another system, the Falcon-AO [88], in order to use the schema mapping similarities.

### 2.3.10. SERIMI

SERIMI [55] [56] proposes a solution for the instance-matching problem composed of two phases: the selection phase and the disambiguation phase. In the selection phase a traditional information retrieval strategy to generate a set of candidate resources for interlinking is applied. For each instance r in a source dataset A, SERIMI extracts its label (its identifier) and searches for instances in a target dataset B that may have a similar label. The problem that multiple distinct instances in B may share the same label is addressed in the second, disambiguation phase. In the disambiguation phase, SERIMI attempts to filter among the instances found in B, those that actually refer to the same entity in the real world as r.

SERIMI uses existing traditional information retrieval and string matching algorithms for solving the selection phase; their contribution is the similarity measure used in the disambiguation phase. This function is designed to operate even when there is no direct ontology alignment between the source and target datasets being interlinked. Figure 8 shows an overview of the interlinking process.
Figure 8: The SERIMI interlinking process

Given a source and target dataset and a set of source resources (instance of a class) (A), SERIMI obtains the label of these source resources and retrieves candidate resources from the target dataset that share a similar label (B). For each source resource, SERIMI retrieves a set of candidate resources (C). In order to disambiguate a set of candidate, SERIMI applies a function of similarity that selects the resources (D) that are the most similar between all candidate sets (E). These selected resources are the solutions for the interlinking (F). The determination of this optimal cross section is a process based on an underlying assumption that the source resources belong to a homogeneous class of interest (e.g. musician, drugs, country, etc.). Unlike SERIMI system, the OtO matching system does not depend on that assumption and thus is applicable in any case.

2.3.11. AgreementMaker

The AgreementMaker system [57] [58] for ontology matching includes an architecture that facilitates the integration and performance tuning of a variety of matching methods, an evaluation mechanism, which can make use of a reference matching or rely solely on "inherent" quality measures, and a multi-purpose user interface, which drives both the matching methods and the evaluation strategies.

The instance matching in the AgreementMaker can be combined with the schema matching or not, similar to the OtO matching system. The instance matching solution of the AgreementMaker consists of doing a lookup using the label of the instance, the type (when provided), and querying against an index which returns a reasonable number of candidate target instances.
The AgreementMaker computes a similarity between the source instance and the candidate instances. These values are then used to rank the candidates and eventually select the best one. The use of different matchers that compare several features about the instances is made, and then their outputs are combined in order to give a final alignment.

The main features for comparison are: (a) labels using a substring similarity, (b) comments and other literals using a Vector Space Model approach, (c) RDF Statements considering property-value pairs, and (d) the score values returned by the lookup services (e.g. Freebase API, Apache Lucene score).

Figure 9 shows the instance matching configuration.

![Figure 9: The AgreementMaker instance matching configuration](image)

2.3.12. LN2R

The reference reconciliation system (LN2R) [59] is a knowledge-based, unsupervised and based on two methods system, a logical one called L2R and a numerical one called N2R.

The Logical method for Reference Reconciliation (L2R) is based on the translation in first order logic (Horn rules) of some of the schema semantics. L2R [61] is based on the inference of facts of reconciliation (Reconcile(i,j)) and of non-reconciliation (¬Reconcile(I',j')) from a set of fact and a set of rules which transpose the semantics of the data sources and of the schema into logical dependencies between reference reconciliations. Facts of synonymy (SynVals(v1,v2)) and of no synonymy (¬SynVals(u1, u2)) between basic values (strings, dates) are also inferred.
Chapter 2

In order to complement the partial results of L2R, a Numerical method for Reference Reconciliation (N2R) [60] is created. It exploits the L2R results and allows computing similarity scores for each pair of references. N2R has two main distinguishing characteristics. First, it is unsupervised: it does not require any training phase from manually labeled data to set up coefficients or parameters. Second, it is based on equations that model the influence between similarities. In the equations, each variable represents the (unknown) similarity between two references while the similarities between values of attributes are constants. These constants are obtained, either (i) by exploiting a dictionary of synonyms (e.g. WordNet thesaurus) or (ii) by using standard similarity measures on strings or on sets of strings. Furthermore, ontology and data knowledge is exploited by N2R in a filtering step to reduce the number of reference pairs that are considered in the equation system. The functions modeling the influence between similarities are a combination of maximum and average functions in order to take into account the constraints of functionality and inverse functionality declared in the OWL ontology in an appropriate way.

2.3.13. Zhishi.links

Zhishi.links [62] is an instance matching system, which utilizes distributed framework to index and process semantic resources, and uses scalable matching strategies for calculating similarities between two resources. Figure 10 shows the Architecture of Zhishi.links.

![Figure 10: The Architecture of Zhishi.links](image-url)
All the dumps (DBpedia, Freebase and GeoNames) are downloaded beforehand and the interconnection is done locally, even the datasets are very large. The comparison between two resources begins from the string similarity calculation. The more terms two resources’ names share, the more similar they are. If a resource has more than one name (i.e. it has aliases), all names take part in similarity calculations and the highest similarity value is chosen. String similarities are used to filter out the least likely match candidates by setting a proper threshold. Afterwards, semantic similarities are calculated. Generally speaking, similarity scores computed in the previous step increases if two resources have some semantic resemblances (e.g. the same property-value pairs), otherwise penalties are paid. Specifically, functional properties (owl:FunctionalProperty) and inverse-functional properties (owl:InverseFunctionalProperty) have higher weights than ordinary properties.

Finally, match candidates are sorted by their similarity scores. In some cases, two or more match candidates may have the same highest similarity scores. Zhishi.links compares their default labels again and chooses the closest matching pair.

Unlike the OtO matching system, the Zhishi.links does not exploit the significance of the instance properties and the schema matching results.
Chapter 3:

3. Presentation of the Ontology to Ontology Matching Process
   3.1. Load Ontologies
   3.2. Schema Matching
   3.3. The new instance matching algorithm
      3.3.1. User Interface Inputs
      3.3.2. Instance Matching Algorithm
      3.3.3. The key Weight Flag
      3.3.4. User Interface Outputs
Chapter 3
3. Presentation of the Ontology to Ontology Matching Process

In this chapter, we provide a detailed presentation of the matching process. The matching process consists of three process phases, as shown in Figure 11: a) ontology loading, b) schema matching, and c) instance matching, which will be analyzed in the following sections.

![Matching Tool Sequence Diagram](image)

Figure 11: OtO System Sequence Diagram

At this point, it is important to note that the end user of the Ontology to Ontology (OtO) matching system is a user who has domain knowledge about the environment in which the OtO system operates, i.e. the Ontology environment. For example, a common software user does not usually have the knowledge about the difference between schema and instance matching of ontologies, which is fundamental for operating the OtO system. The lack of shared vocabulary between the user and the system in order to communicate, would lead to insufficient customization of the system, followed by insufficient results. For this reason, we refer to end users of the OtO system as “domain expert”.

Evangelia Daskalaki
Master Thesis | University of Crete | Computer Science
39
Chapter 3

In order to begin with the matching process, the domain expert gives her input from the User Interface. The User Interface inputs are divided into three sections (Figure 12): (a) the ontology selections, (b) the schema matching selection and (c) the instance matching selection.

In the ontology selections, the domain expert indicates the source and target ontologies. In the schema matching selections, the domain expert selects whether she wants to proceed with schema matching, the matching algorithm, the threshold and the filtering algorithm. Finally, in the instance matching selections the domain expert selects whether she wants to proceed with instance matching, the type of instances that will take part in the matching procedure, the instance threshold and the string matching algorithms.

![Figure 12: Schema and Instance Matching System User Interface](image)

3.1. Loading Ontology

The first phase of the matching process is the loading of the selected ontologies. The OtO matching system is able to parse ontologies written in OWL (Web Ontology Language) with the aid of the Jena Semantic Web Framework [10]. When the domain expert inserts the source ontology and the target ontology, the system stores the ontologies into the DB as a directed acyclic graph [21]. If the ontologies are already stored in the DB, they are loaded as directed acyclic graphs.
For the purpose of saving ontologies into the DB as directed acyclic graphs, each ontology concept (class) becomes a node in the graph. For the properties of each class, a node is added to the graph and connected to the graph. For each of the classes, associated information such as datatype is stored into the node. Properties that have domain and range are also represented as nodes in the graph. Each of these nodes has a parent node that represents it’s domain class and a child node that represent it’s range class. The graph also contains directed edges from the parent node to the new node as well as from the new node to its child node. IS_A relationships in the Ontology are represented as directed edges from the subclass node to the superclass node [22].

Although the system follows the principles of [1], important changes and updates have occurred since then:

First, an important change of the OtO system is that is uses its own relational DB and does not depend on an external DB as is the case in [1]. The OtO system is a fully independent system which follows the schema in Figure 13.

Furthermore, the previous schema in [1] has been expanded in order to be able to load instances and instance properties. In particular, the ModelInstanceProperties table has been added to the schema, to store all the instance properties, using the combination of modelID, instanceID and propertyName as the composite key (see Figure 13).
3.2. Schema Matching

The second phase of the matching process is the schema matching process. Depending on the domain expert’s selections, the OtO matching system either proceeds with the schema matching process or not. If the user does not select to run a schema matching algorithm, the system assumes that the source schema and the target schema are the same and creates a “mirror mapping”, which is stored into the database. In order to create the mirror mapping the OtO system reads the source schema and creates a mirror of it as the target schema.

If the domain expert has selected to proceed with the schema matching, then the OtO system follows the algorithm that is described in [1], which followed the work done in [2].

In [1], after storing the Ontologies into the DB as a directed acyclic graph, the schema matching algorithm follows a hybrid algorithm for matching the schemas. It is called hybrid algorithm because, depending on the domain expert’s selections, the algorithm runs sequentially a lexicographic matching process, a semantic matching process and syntactic matching process between the schemas. Then it returns a similarity matrix of all the concepts in the schemas (Figure 14).

The inputs of the schema matching algorithm, as shown in Figure 2, are: (a) the algorithm of the schema matching (lexicographic / semantic/ syntactic), (b) the schema threshold and (c) the filtering algorithm. The outputs of the schema matching are the schema mappings along with their similarity measure (Figure 5).
The schema mapping results are presented to the end user in the form of a table with the source element, the target element and the similarity of each match (Figure 15). The domain expert can either accept the matches, or reject them, by unchecking the “accept” checkbox of each match. Furthermore, the domain expert can alter the automatically predefined weights of importance of each property match. The usage of these weights is described in detail in the next section “3.3 The new instance matching algorithm”.

Figure 14: Schema matching Activity Diagram
3.3. The new instance matching algorithm

The third phase of the matching process is the instance-based ontology matching. This is the new feature of the OtO matching system which completes the OtO matching process by also allowing the comparison of the very last components of an ontology, namely the instances. Through this procedure, the OtO matching system will measure the similarity between sets of annotated instances. The goal of this process is to determine instances that represent the same real object or event.

This would be a very expensive procedure in terms of time and memory usage, if all the instance-pairs would be compared to each other, since millions of comparisons would have to be performed. In order to reduce this number, the OtO matching system has introduced a combination of methods and algorithms for better accuracy, efficiency and effectiveness. In particular the instance matching algorithm leverages (1) the rich semantic knowledge we gain from the output mappings of the schema matching process, (2) the implicit knowledge of domain
expert by (semi-)automatically capturing the identification power of the properties with the use of property-weights and (3) the 2-phase filtering method of the results, in order to accurately and efficiently detect the ontology instances that represent the same real-world entity.

As far as the first is concerned, the OtO matching instance algorithm uses the outputs of the process described in “3.2 Schema matching”. As a result only the instances that belong to semantically related classes are included in the instance matching algorithm.

Furthermore, thought the use of the property weights the domain expert can capture the identification power of the properties. This feature is explained in section “3.3.3 Property weights” in detail.

Last but not least, with the use of a 2-phase filtering method the OtO instance matching algorithm can return more accurate results. The first phase of filtering is the similarity threshold and the second filtering phase is done with the use of the “key weight flag”. The “key weight flag” is further analyzed in section “3.3.4 The Key Weight Flag”.

### 3.3.1. User Interface Inputs

The inputs of the instance matching process are:

- the accepted schema matches
- their weights of importance as inserted by the OtO system or altered by the domain expert,
- the instance threshold,
- the selected string matching algorithms [4], [5] and
- the type of instances that will take part in the comparison procedure
  (Anonymous instances/ Parents instances / Thing instances)

As one can notice, the OtO matching system uses the output from the schema matching process. This way the system utilizes the instance matching process by leveraging the rich semantic knowledge we gain from the output mappings of the schema matching process. What’s more, the OtO system uses the implicit knowledge gained from the domain expert capturing the identification power of the properties.

Furthermore the domain expert can insert additional type of the instances that will take part in the instance matching process. In other words, the default
instances that will be included in the matching process are the class instances, but the domain expert can include more. Specifically, she can choose to include the Anonymous instances, the Thing instances or/and the Parents instances.

The Anonymous instances are instances of type <rdf:Description> and do not contain any information about the type of the instance. An example is shown below:

```xml
<rdf:Description rdf:about="#ID5657817362302919422">
  <tbox:cogito-Name rdf:datatype="&xsd;string">Helen Morse</tbox:cogito-Name>
  <tbox:cogito-first_sentence rdf:datatype="&xsd;string">Helen Morse (born 24 January 1948) is an Australian actress who has appeared in films, on television, and on stage.</tbox:cogito-first_sentence>
  <tbox:cogito-tag rdf:datatype="&xsd;string">actor</tbox:cogito-tag>
  <tbox:cogito-domain rdf:datatype="&xsd;string">show</tbox:cogito-domain>
</rdf:Description>
```

A Thing instance is an instance of type Thing. The owl:Thing is a very general predefined class such that every OWL class is a subclass of owl:Thing. As a consequence, all individuals are by definition instances of owl:Thing. An example is shown below:

```xml
<owl:Thing rdf:about="#ID4844303171709206706">
  <hasDataValue rdf:datatype="&xsd;string">Lori Vanadzor</hasDataValue>
</owl:Thing>
```

Last, but not least, the Parents instances of a class B are the instances that are contained in class A, where class A is the parent of class B. For example if class “actor” is a subclass of class “person”, the example below shows two instances from the source ontology and the target ontology, which refer to the same real persons with the difference that in the source ontology the instance is of type actor and in the target ontology the instance is of type person.
As described in section “Presentation of the OtO matching process”, when the domain expert has performed the customization of the inputs, she can press the button “Continue with the instance Matches” at the bottom of the schema matching result table (Figure 16).
Figure 16: Continue with instance matching button
3.3.2. Instance Matching Algorithm

![Instance Matching Activity Diagram](image)

Figure 17: Instance Matching Activity Diagram
Chapter 3

Figure 17 shows the activity diagram of the instance matching algorithm. The algorithm begins by loading the schema mappings. For each class mapping returned from the schema matching, the algorithm returns the source class instances and the target class instances. Depending on the domain expert’s type of instance selections, the system may also return the Anonymous target instances and/or the Parents’ class target instances and/or the Thing target instances in order to proceed with the instances comparison.

For each of the instances of the source and the target mapping class, the algorithm returns the property annotations and their values. If there are no class mappings then the algorithm compares all the instances with each other. If the property annotations of the instances are among the property mappings, then the algorithm checks the property weight of the match. If the weight is over the low limit of 0.0, then the algorithm checks the property values. If the property values are references to other instances then it indicates the correct instance values. Once the values have been indicated the algorithm proceeds with the value matching algorithms.

As reported above in the customizable instance matching inputs, the domain expert can select the string matching algorithms that will be applied on the instance values. The available instance matching algorithms contain character based metrics, token-based metrics, phonetic similarity metrics and semantic-based metrics. In particular:

- **The prefix algorithm** [91]: This matcher looks for common prefixes. The output of the matcher can either be 0 (no match) or 1(exact match). This algorithm has importance weight 0.5, because the matching result is not very indicative in case of skewed and misspelled data.

- **The suffix algorithm** [91]: This matcher looks for common suffixes. The output of the matcher can either be 0 (no match) or 1(exact match). This algorithm has importance weight 0.5, because the matching result is not very indicative in case of skewed and misspelled data.

- **The character frequency algorithm** [6]: This matcher derives the occurrences of characters in the strings and computes the similarities based on the character occurrence list. The result of the algorithm has weight 1.0.

- **the Di-Gram algorithm** [7] [91]: Token-based similarity metric algorithm. Strings are compared according to their set of two-grams, i.e. sequences of two characters. The result of the algorithm has weight 1.0.
Chapter 3

- the **Tri-Gram algorithm** [7] [91]: Token-based similarity metric algorithm. Strings are compared according to their set of three-grams, i.e. sequences of three characters. The result of the algorithm has weight 1.0.

- the **soundex algorithm** [63] [64]: This matcher is the most common phonetic coding scheme. It computes the phonetic similarity between names from their corresponding soundex codes. The weight of the results of this algorithm is 0.5.

- the **edit distance algorithm** [8] [9]: String similarity is computed from the number of edit operations necessary to transform one string to another one. The result of the algorithm has weight 1.0.

- the **single error algorithm** [8] [9]: again the Levenshtein distance is calculated and if it is 1 or less the output of the matcher is 1(match), otherwise it is 0 (no match). The result of the algorithm has weight 1.0.

- The **Wordnet® matching algorithm** [32] [33]: The WordNet® is a large lexical database of English. Nouns, verbs, adjectives and adverbs are grouped into sets of cognitive synonyms (synsets), each expressing a distinct concept. Synsets are interlinked by means of conceptual-semantic and lexical relations. The Wordnet matching algorithm uses the WordNet thesauri to return lexical and semantic relations between the concepts. The result of the algorithm has weight 1.0.

It should be mentioned at this point, that the fixed weights of the available instance matching algorithms are indicative. They are customized after exhaustive testings in the OtO system and by using our own experience. Of course they can be easily customized in the future.

After applying the selected string matching algorithms, a weighted average of the property similarities is calculated with the formula (1):

\[
\text{Property sim} = \sum_{n=0}^{i} \frac{\text{sim}_i \cdot w_i}{\sum_{n=0}^{i} w_i} \tag{1}
\]

Where \( \text{sim} = \{\text{sim}_1, \text{sim}_2, \text{sim}_3, \ldots, \text{sim}_i\} \) are the calculated value similarities from the string matching algorithms and \( w = \{w_1, w_2, w_3, \ldots, w_i\} \) the corresponding weights of each matching algorithm which are fixed from the system.
This process is followed for all the matching property annotations of each instance matching pair and the property similarities are stored in a property similarity matrix.

When there are no more properties to compare, a weighted average of the instance similarity is calculated this time for the instance pair with formula (2). In formula (2) the similarities sim = \{sim_1, sim_2, sim_3,...,sim_p\} represent the similarities of each property pair and the weights w = \{w_1, w_2, w_3,...,w_p\} represent the weights of each matching pair which are given by the domain expert.

\[
\text{Instance sim} = \frac{\sum_{n=0}^{p} \text{sim}_p \cdot \text{w}_p}{\sum_{n=0}^{p} \text{w}_p}
\] (2)

If the calculated instance similarity is above the given instance threshold then the instance pair is an accurate match and might be added into the result map. If it will be added in the matching result or not depends on the “key weight Flag” which is analyzed in the next paragraph.

Unlike the schema matching algorithm, in the instance matching algorithm every source instance is allowed to match with no more than one target instance (1:1). That of course is based on the fact, that the aim of the instance matching algorithm is the identity recognition, so the system returns only the best target instance match for a given source instance. In order to do so, the algorithm keeps in memory the last accurate instance match and its similarity and compares it with the next accurate instance match and its similarity. If the next instance match for the same source instance has a higher similarity than the previous one, then the previous is deleted from the result list and the new one is kept in the list. Otherwise, the new one is deleted and the algorithm continues until it finds all the instance correspondences.
3.3.3. Property weights

The OtO instance matching algorithm leverages the property weights in order to capture the identification power of the properties. The property weight is a real number in the range of [0.0, 1.0]. This number shows how indicative a property match is to determining whether two instances refer to the same real object or not. In other words, the property weight indicates the probability of a property match with the same value to be true particular match. This probability includes two kind of stochastic processes, as also described in [92]: the probability distribution of the property and the probability distribution of the knowledge of the property. The first refers to how many instances share the same property value in the real world and the second refers to how many instances share the same property in a specific knowledge base.

In order to have a better understanding of previously mentioned stochastic processes, let us see an example. Assumingly we are comparing two instances of the ontology class “country”, in order to indentify if these two instances refer to the same real country or not. This “country” class has two properties: (a) the “country-currency” and (b) the “country-name”. For both of the instances the value of the property “country-currency” is “EUR”. The probability distribution of the property “country-currency” would show that 17 countries have as official currency the EUR. This would mean that the probability of these two countries to refer to the same real country would be 5.89%. If the domain expert knew that only 10 out of the 17 are inserted in our knowledge base then the probability would be 10% that these two countries refer to the same real country.

As far as the property “country-name” is concerned the distribution of this property and the distribution of the knowledge of the property is the same; each country has a unique name. So the probability of two countries to refer to the same real country if they share the same name is 100%. Thus, the weight of the property “country-name” would be would be 1.0 and the weight of the property “country-currency” would be 0.1.

The domain expert is expected to alter the automatically inserted weights of each property-match or insert new ones in the matching process, if she considers it as necessary. A first attempt to insert weights to the properties is done automatically from the OtO system, by using keyword rules. This keyword rules are inserted into an external file called “keyword_rules.txt”, which can easily be customized from the domain expert. The domain expert inserts the keywords and their associated weights, separated by comas, in the file. As a result, the property matches that include the specific keywords are given automatically the associated weight amount. Some examples of the keywords are: "name 1.0", "homepage 1.0", "coordinates 1.0", "website 1.0", etc. The other property matches are given zero weight. The
domain expert is asked to correct the weights and insert more or delete the unnecessary ones according to her implicit knowledge.

As described above, through the use of weights the system is able to determine the matches that are crucial for the outcome and separate them from the matches that are not important. This is done in the matching results table (Figure 15), after the schema matching process. In the last column of the table, the domain expert inserts the weights. If the domain expert selects the weight of $w_i=0$, this means that the matching is not important for the result and thus does not match the instance annotations. On the other hand if she selects $w_i=1.0$ this means that the similarity of the property match is of high importance and may indicate that the property values match the entire instance. Through that information, the OtO matching system is able to reduce the number of comparisons conducted, without losing any important information.

3.3.4. The key Weight Flag

The key weight flag is an important feature added into the instance matching algorithm in order to provide more accurate and trustworthy results. The key weight flag is a kind of flag which either allows instance matches with similarity above threshold to be added in the mapping result or not. In order to determine this, the key weight flag is turned into true (green flag to pass) only if the matched instances have at least one property match with weight above or equal 0.5 ($w_i\geq 0.5$).

\[
\text{Count} \left( \left[ \text{w}_1, \text{w}_2, \text{w}_3, \ldots, \text{w}_p \right] \geq 0.5 \right) > 0
\]

If the matched instances have similarity above the threshold but all their property annotation weights are less that 0.5 then the match is not considered as trustable and is not added in the instance matches result.

In order to be more specific, let us see an example of the second case. In Figure 18 shows an example of two instances from the source and the target ontology.
The information we have for the source instance with id “id190e3d46-0ded-4de3-9e9d-9acf1b84d4eb” is that it is an actor, with domain “cinema”, tag “actor”, name “Max Brown” and cogito first sentence “Max Brown (born February 10, 1981) is an English actor”. On the other hand the information we have about the target instance with id “ID1703271437194371702” is that it is an actor, with domain “cinema” and tag “actor”.

For the property matches: “type”, “cogito-tag”, and “cogito-domain” the OtO system will automatically assign a weight of 0.0 (because they are not considered as key information about the instance). If the domain expert decides to alter the weights, she will probably give low weights (below 0.5). This means that even if two instances share these properties and have the same values (similarity = 1.0), it is very likely that the instances will not represent the same real entity. Let’s assume that the domain expert assigns the weight of 0.4 for all three target properties mentioned in our example. The probability for these instances to represent the same real entity is:

\[
\text{Probability} = (\text{sim}_1 \cdot w_1) \cdot (\text{sim}_2 \cdot w_2) \cdot (\text{sim}_3 \cdot w_3) = 1 \cdot 0.4 \cdot 1 \cdot 0.4 \cdot 1 \cdot 0.4 = 0.064
\]

This means that there is 6.4 % probability that these two instances represent the same real world entity, which is an unacceptably low probability.
Chapter 3

The system allows the instance matches to be added in the result only if they have at least one matching property annotation with at least 50% probability to match the real instance entity.

3.3.5. User Interface Outputs

The output of the instance matching algorithm is a set of instance matches, where each source instance is mapped to at most one target instance (1:1). They are presented to the end user in the form of a table where each match is a table row with the source instance id, the target instance id and their similarity. The user can select to see the details (their properties and values) of each matching pair by clicking the “View” button in the end of the table row (Figure 19).

Once the user clicks the “View” button a new window appears with a detailed view of the property annotations and their values of each instance. Figure 20 shows an example of the detailed matching window.

![Figure 19: Instance matching result](image-url)
Figure 20: Detailed Matching window
Chapter 3
Chapter 4:

4. Experiments and Results

4.1. Qualitative Matching Criteria

4.2. Test cases results

4.2.1. Test Case 1 -10

4.2.2. Conclusions for test case 1-10

4.2.3. Test cases 11-19 and results

4.2.4. Conclusions for test cases 11-19

4.2.5. Test cases 20-29 and Results

4.2.6. Conclusions for test cases 20-29

4.2.7. Test cases 30-37 and Results

4.2.8. Conclusions for test cases 30-37

4.2.9. Overall Conclusions of the test cases
4. Experiments and Results

In order to evaluate the matching system and test the instance matching algorithm we have used the ISLab Instance Matching Benchmark (IIMB) [12]. The IIMB is a benchmark that is generated automatically, starting from one data source that is automatically modified according to various criteria. The original data source contains OWL/RDF data about actors, sports, persons, and business firms provided by the OKKAM European project [13].

The benchmark is composed by 37 test cases. For each test case it is required to match the original data source (source ontology) against a new data source (target ontology). The original data source contains 222 different instances and the new data sources contain from 222 to 1722 different instances. Each test case contains a modified ABox (abox.owl + tbox.owl) and the corresponding mapping with the instances in the original ABox (Golden standard).

The IIMB benchmark was chosen because it contains many test sets which introduced different kinds of challenges as listed below:

- Test case 01: Contains an identical copy of the original data source (instance IDs are randomly changed)
- Test case 02 - Test case 10: Value transformations (i.e., typographical errors simulation, use of different standards for representing the same information). In order to simulate typographical errors, property values of each instance are randomly modified. Modifications are applied on different subsets of the instances property values, with varying levels of difficulty (i.e., introducing a different number of errors).
- Test case 11 - Test case 19: Structural transformations (i.e., deletion of one or more values, transformation of datatype properties into object properties, separation of a single property into more properties).
- Test case 20 - Test case 29: Logical transformations (i.e., instantiation of identical individuals into different subclasses of the same class, instantiation of identical individuals into disjoint classes, instantiation of identical individuals into different classes of an explicitly declared class hierarchy).
- Test case 030 - Test case 037: Several combinations of the previous transformations.

Furthermore the IIMB Benchmark was firstly introduced in the Ontology Alignment Evaluation Initiative 2009 campaign [11]. Thus, apart from having a
Golden standard to compare our results we also have other systems’ results to be compared against.

The other systems that participated in the IIMB benchmark 2009 campaign are the Aflood system [14], the ASMOV system [15], the DSSim system [16], the HMatch system [89], the FBEM system [18] and the RiMOM system [17]. All of these systems are analyzed in chapter “2.3 Related work”. The qualitative matching Criteria are going to be presented in the next sections. Furthermore, every test case will be analyzed and the results will be reported.

All the test cases were performed on a computer with a 2,99 GHz Intel® Core™2 Duo CPU and 3,25 GB of RAM, running Microsoft Windows XP. The OtO system was deployed on a Tomcat Apache 5.5 over a mySQL Server 5.1 DBMS.

4.1. Qualitative Evaluation Criteria

The standard evaluation measures for our test cases has been precision, recall [19] and the F-measure [20], computed against the reference alignments (Gold standard). Precision is the fraction of retrieved instances that are relevant, while recall is the fraction of relevant instances that are retrieved. In information retrieval, a perfect precision score of 1.0 (or of 100%) means that every result retrieved by a search was relevant but says nothing about whether all relevant results were retrieved. In contrast a perfect recall score of 1.0 (or of 100%) means that all relevant results were retrieved by the search but says nothing about how many irrelevant results were also retrieved.

The terms “true positives”, “true negatives”, “false positives” and “false negatives” refer to the relationship between the predicted results of an item and the actual results. This is illustrated in the table below:

<table>
<thead>
<tr>
<th>Test outcome</th>
<th>Condition</th>
<th>Positive</th>
<th>Negative</th>
</tr>
</thead>
<tbody>
<tr>
<td>Positive</td>
<td>True Positive</td>
<td></td>
<td>False Positive</td>
</tr>
<tr>
<td>Negative</td>
<td>False Negative</td>
<td></td>
<td>True Negative</td>
</tr>
</tbody>
</table>
Chapter 4

The table shows that the True Positive results are those that were predicted as correct results from the Golden standard and were returned from the test outcome. The False Positive results are those that were predicted as false but were falsely returned from the test outcome. The False Negative results are those that were predicted as correct results but were not returned from the test outcome. Finally the True Negative results are those that were predicted as false from the Golden standard and were correctly not returned from the test outcome.

Precision and recall are defined as the ratios below:

\[
\text{Precision} = \frac{tp}{tp + fp} \quad \text{Recall} = \frac{tp}{tp + fn}
\]

where \( tp \) is the number of True Positive results, \( fp \) is the number of False Positive results and \( fn \) is the number of False Negative results.

The F-measure is a measure of a test's accuracy. It considers both the precision and the recall. The F-measure can be interpreted as a weighted average of the precision and recall. The F-measure reaches its best value at 1.0 and worst score at 0.

\[
F = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}
\]

In all our test cases the evaluation criteria are the Precision, the recall and the F-measure.

4.2. Test cases results

In this section we analyze all the test cases and their results. Furthermore, the customization of the UI is explained. It should be mentioned that in all test cases all the matching algorithms from the section “3.3.2 Instance Matching Algorithm” have been used, except for the WordNet ® matching algorithm. The reason is that for the matching between names and other personal instance data (i.e. date of birth, occupation) which is the case in the IIMB Benchmark, the semantic matching of such a matcher would not be useful.
4.2.1. Test Case 1 -10

In the first test case the schemas and the instances are the same only the ID’s of the instances have changed. The results for the OtO system are 100% precision and recall (Figure 21). In this test case only the default property weight were used.

![Test Case 1 Results](image1)

Figure 21: Test Case 1 Results

In test case 2, the target schema was filled with noisy and skewed data. Many typographical errors have been inserted into the data. In order to deal with that problem, the OtO system was tuned accordingly by lowering the threshold. The minimum threshold that was used was 0.5 and the default weight values were not altered. The results of the OtO system, as well as the other systems appear in Figure 22. Almost all of the systems performed very well, apart from DSSim system.

![Test Case 2 Results](image2)

Figure 22: Test Case 2 Results
In test case 3, again the target instances had many typographical errors. We have tuned the system accordingly by lowering the threshold. The minimum threshold that was used was 0.49 and the default weight values were kept. The results of the OtO system as well as the other systems appear in Figure 23. Almost all of the systems performed very well apart from DSSim. We can assume that the matching algorithms that are used in the DSSim are not very efficient, when the data have typographical errors.

![Figure 23: Test Case 3 Results](image)

Again, in test case 4 the target instance data were skewed with many typographical errors. We have tuned the system accordingly by lowering the threshold. The minimum threshold that was used was 0.42 and the default weight values were not altered. No schema matching algorithm was configured to run, like in the previous test cases. The results of the OtO system as well as the other systems appear in Figure 24. DSSim is once more the only system that did not return good results.
In the next test case the target instances are very skewed with many typographical errors. The strategy that was used in order to overcome the noisy data was to lower the threshold to 0.40. The default weight values were not altered. No schema matching algorithm was configured to run. The result of the OtO system was 100% precision and recall as shown in Figure 25.

Test 6 ran with the same tuning as test case 5. The result of the OtO system was 100% precision and recall as shown in Figure 26.
The data skewness in test case 7 were so intense that apart from keeping the threshold low (t=0.40) we altered the weights of the properties in order to force more properties to be compared. That way we increase the possibilities to return more accurate results. No schema matching algorithm was configured to run. The result of the OtO system was 100% precision and recall as shown in Figure 27.

In the next case, the same tuning as in test case 7 was followed. The results of the OtO system were 100% precision and recall as shown in Figure 28.
Test Case 8

Figure 28: Test Case 8 Results

For test case 9, in order to keep an accurate result with precision and recall 100%, the threshold was reduced to 0.3. Many property weights that had zero value were increased, in order to take part in the matching process. No schema matching process was configured to run. The result of the OtO system was 100% precision and recall as shown in Figure 29.

Test Case 9

Figure 29: Test Case 9 Results

Test case 10 was the last test case from the first category. Here the typographical errors were so intense that in many cases even a human being could not indicate any similarities between matches. After running the instance matching
algorithm with threshold 0.3 and weights added in all the property matches, the OtO matching system returned 3 False Positive results and 3 False Negative out of 222 results.

One of the False Positive is shown in Figure 30. In that case the weights of the annotation properties type, cogito-tag and cogito-domain are very low (w=0.1). On the other hand the weight of the property cogito-Name is 1.0 and cogito_first-sentence is 0.7. As one can see, the skewness is so intense that one could even think that the cogito first-sentence is written in a different language, but this is not the case. The case is that too many typographical errors have been inserted, so that this result is not returned from the OtO system as a match.

The results of the all the systems are shown in Figure 31.

Figure 30: Instances example from test case 10
4.2.2. Conclusions for test case 1-10

In the ten test cases of the IIMB Benchmark considered above, the main focus was to measure how accurate an instance matching system can be, when the data have many typographically errors. The OtO matching system has proved that by tuning the threshold and the property weights it can deal with that problem. It returned the best F-measure results along with the ASMOV and the RiMOM systems, which are 100% precision and recall. The only System that did not show acceptable results in this test case is the DSSim system. Figure 32 shows average precision and recall of tests 1-10 for all the systems. Figure 33 shows the changes that occurred to the threshold from test 1 to test 10.
Figure 32: Results for test cases 1-10

<table>
<thead>
<tr>
<th>Method</th>
<th>Prec.</th>
<th>Rec.</th>
<th>FMeas.</th>
</tr>
</thead>
<tbody>
<tr>
<td>AFlood</td>
<td>1.00</td>
<td>0.99</td>
<td>0.99</td>
</tr>
<tr>
<td>ASMOV</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>DSSim</td>
<td>1.00</td>
<td>0.37</td>
<td>0.54</td>
</tr>
<tr>
<td>HMatch</td>
<td>0.97</td>
<td>0.98</td>
<td>0.97</td>
</tr>
<tr>
<td>FBEM</td>
<td>0.95</td>
<td>0.93</td>
<td>0.94</td>
</tr>
<tr>
<td>RiMOM</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>OTO</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
</tbody>
</table>

Figure 33: Threshold for test cases 1-10
4.2.3. Test cases 11-19 and results

Test cases 11–19 have structural transformations, i.e. deletion of one or more values, transformation of datatype properties into objecttype properties, separation of a single property into more properties. Furthermore references in the instances are introduced, as well as instances are transferred under the owl:Thing type. In all the test cases the threshold was 1.0 because no typographical errors appear in the data. The next paragraphs analyze every test case and its results separately.

Beginning with test case 11, the OtO system was faced with instances in the target ontology that are of type “owl:Thing” and have properties “hasDataValue”. Some other instances in the same ontology have references to these instances. An example is shown in Figure 34. The OtO system had to find the references and calculate the correct similarities.

In order to deal with this, the system was tuned to also consider “Thing instances” in the matching process. This test case ran with the default weights of the system. Figure 35 shows that the OtO system returned 100% precision and recall.

![Reference in instances from test case 11](image-url)
Test case 12 introduces the target instances that are type “owl:Thing” with properties “hasObjValue” referencing to other instance reference (Figure 36). In particular, the OtO system had to find the references values, often by searching into more than three instances. In order to deal with this, the system was tuned to also consider “Thing instances” in the matching process. Furthermore, this test case ran with the default weights of the system. The OtO Matching system again returned 100% precision and recall as shown in Figure 37.

Figure 36: Reference in instances from test case 12
Test Case 12

![Test Case 12 results](image)

Figure 37: Test Case 12 results

In test case 13, all the target instances are of type “owl:Thing”. The OtO system was customized to consider the “Thing instances”. This test case ran with the default weights of the system. The OtO Matching system again returned 100% precision and recall as shown in Figure 38.

Test Case 13

![Test Case 13 results](image)

Figure 38: Test Case 13 results

As in test case 13, in test case 14 all the target instances are of type “owl:Thing”. Furthermore some minor changes have occurred in the target instances were some properties have been removed. The OtO system was customized to
consider the “Thing instances”. The weights have also been altered and more weights have been inserted, so that more properties take part in the matching process. The OtO Matching system returned 100% precision and 100% recall as shown in Figure 39.

![Figure 39: Test Case 14 results](image)

In Test case 15 many changes have occurred in the target instances were some properties have been removed completely. The weights have been altered and more weights have been inserted in order to deal with the loss of information. The OtO Matching system returned the best results (along with ASMOV and RiMOM) with 99% precision and 100% recall as shown in Figure 40.

![Figure 40: Test Case 15 results](image)
In the next test case, intense changes have occurred in the target instances were many properties, even the very important ones, have been removed. The weights have been altered and more weights have been inserted as was done in the previous test case. The OtO Matching system returned 90% precision and 100% recall as shown in Figure 41. Again OtO system returned the best results along with ASMOV and RiMOM.

![Test Case 16](image)

**Figure 41: Test Case 16 results**

Test case 17 showed a severe loss of data. The target instances have left with a few properties to characterize them, which often were very general data (like the property “type”). As a result, the OtO system returned only 62% precision and 99% recall as shown in Figure 42. This result is still the second best result with only 1% percent difference in F-measure from the best result, which was shown from the RiMOM system (Figure 42).
Test Case 17

Figure 42: Test Case 17 results

Test case 18 follows the same philosophy as the previous one, although the results of the OtO system were better. Again the OtO system returned the second best results among all the systems with 99% precision and 100% recall (Figure 43). In order to return these results the OtO system was tuned to consider the “Thing instances” in the matching process. The weights were altered and more have been inserted in order to allow more properties to take part in the matching process.

Figure 43: Test Case 18 results
Moreover, test case 19, as in test case 17 and 18, shows severe loss of data. The results of the precision and recall are 87% and 100% respectively and are shown in Figure 44. These results rank the OtO system in the first position in this test.

![Test Case 19](image)

**Figure 44: Test Case 19 results**

### 4.2.4. Conclusions for test cases 11-19

The OtO Matching system and the RiMOM system show the best results in precision and recall. As shown in Figure 45, the RiMOM system has put its efforts on the precision in contrast to the OtO system were the recall is more important. This depends on the strategy of each system. In our case, we want to minimize the False Negative results by sacrificing some False Positive results. In other words the OtO system prefers not to insert any false data in the DB by losing some correct data.

Another important factor of the success of the OtO system in these test cases is the key weight Flag (“3.3.3. The key Weight Flag”). The key weight Flag helped the algorithm separate the high-confidence results from the low-confidence ones. In Figure 45, the average precision, recall and F-measures of all the systems are shown.
4.2.5. Test cases 20-29 and Results

Test cases 20 – 29 have logical transformations, i.e., instantiation of identical individuals into different subclasses of the same class, instantiation of identical individuals into disjoint classes and instantiation of identical individuals into different classes of an explicitly declared class hierarchy.

Due to the fact that there were no typographical errors, the threshold was configured to 1. Furthermore, the schema matching process ran for all the tests. In detail, each test case is analyzed in the next paragraphs.

Beginning with test case 20, the Anonymous instances appear for the first time. The Anonymous instances are instances stated as `<rdf:Description>` in the abox.owl file, without reporting the type of the instance. The OtO system was configured to calculate the Anonymous instances in the matching process by clicking the “Anonymous Instances” checkbox in the User Interface. The results showed once more that it returned the best precision and recall percentages as did the RiMON the ASMOV and the AFlood systems (Figure 46).
Chapter 4

Figure 46: Test Case 20 Results

In the next test case, all the “Anonymous instances” are turned into “Thing instances” in the target Ontology. Again the OtO system was configured to calculate the “Thing instances” in the matching process by clicking the “Thing Instances” checkbox in the User Interface. The results showed that it returned 100% precision and recall (Figure 47) ranking it once more in the first position.

Figure 47: Test Case 21 Results

In test cases 22-26 instances are randomly turned into Thing instances and Parent Instances. In order to cope with that the OtO system was configured to calculate the Thing instances and the Parent instances in the matching process by
clicking the “Thing Instances” checkbox and the “Parents Instances” checkbox in the User Interface. The results returned 100% precision and recall for the OtO system for all the test cases (Figure 48-52).

![Figure 48: Test Case 22 Results](image)

![Figure 49: Test Case 23 Results](image)
Figure 50: Test Case 24 Results

Figure 51: Test Case 25 Results
In test case 26, the target instances are randomly turned into Parent Instances. In order to cope with that, the OtO system was configured to calculate the Parent instances in the matching process by clicking the “Parents Instances” checkbox in the User Interface. The results returned 100% precision and recall (Figure 52).

In test case 27, the target instances are randomly turned into Parent Instances. In order to cope with that, the OtO system was configured to calculate the Parent instances in the matching process by clicking the “Parents Instances” checkbox in the User Interface. The results returned 100% precision and recall (Figure 53).

In the next test case, instances are randomly turned into Anonymous Instances, Thing instances and Parent Instances. In order to cope with that, the OtO
system was configured to calculate all the above instances in the matching process. The results returned 100% precision and recall (Figure 54).

![Figure 54: Test Case 28 Results](image)

In test case 29, none of the systems returned results, because all the target instances have fixed type “tbox:Item” but the source ontology has no instances for the class “tbox:Item”. Thus, the target instances are not included into the instance matching process. Even the Golden standard has no results reported. So the OtO system returned the correct results once more (Figure 55).

![Figure 55: Test Case 29 Results](image)
4.2.6. Conclusions for test cases 20-29

The OtO system returned 100% in F-measure for all the test cases 20-29. The difficulties faced in those test cases were to customize the system appropriately in order to include all the necessary instances for the matching process. Figure 56 shows the overall results for all the systems. The OtO system and the ASMOV system attained the higher rank. The other systems performed well, except for the FBEM system, which did not return satisfying results.

![Figure 56: Overall results for test cases 20-29](image-url)
4.2.7. Test cases 30-37 and Results

Test cases 30-37 have several combinations of the previous transformations. In the next section each test case will be analyzed separately.

Test case 30 contained only typographical errors in the instance data. In order to overcome the typographical errors, the OtO system was configured by decreasing the threshold to 0.4. Figure 57 shows the overall results.

Test case 31 has data which were very skewed with many typographical errors. Furthermore random properties have been removed. Some of these properties have been critical for the identification of the instances. The OtO project returned 66% precision and 96% recall with threshold 0.62. The overall F-measure was 78% which was the third best result, after RiMOM and ASMOV system.

In all previous test cases OtO had the highest F-measure among all systems. This is the first test case in which OtO did not perform very well, and that was because of the data were biased in such a way that the confidence level of some False Positive results was very low. Figure 58 shows an example of one False Positive result which was not returned by the OtO system. In this case, the only information we have in the target instance is the domain and the type, which are not considered as reliable information to identify a real object. Figure 59 shows the overall scores of all systems for this test case.
Test case 32 had occurrences of typographical errors, missing property values and instances under the parent class of the source instance. The OtO system was
configured with threshold 0.5 and with the matching process including the parents’ instances. Figure 60 presents the overall results.

![Figure 60: Test Case 32 Results](image)

In test case 33, data in the target ontology were intensely skewed. In order to overcome the typographical errors, the threshold was tuned to 0.3. The results were 100% precision and recall for the OtO system (Figure 61).

![Figure 61: Test Case 33 Results](image)

Finally, test cases 34-37 had no typographical errors, but target data were located under Thing instances or Parent instance. Moreover, the target instances had many references to other target instances. As a result, in many cases the OtO
system had to search in over than three instances in order to find the correct property value of an instance (as was the case in test case 12). Finally the number of target instance data in some cases has reached the 1724 instances, which means that the OtO system had to perform millions of comparisons between the properties of the instances. Figure 62 – 65 show the results of all the systems for the test cases 34-37. For all those tests OtO system achieved once more excellent results, which were 100% F-measure.

![Test Case 34 Results](image1)

![Test Case 35 Results](image2)
Figure 64: Test Case 36 Results

Figure 65: Test Case 37 Results
4.2.8. Conclusions for test cases 30-37

The OtO Matching system performed very well in almost all test cases, except for test case 31. Due to that, it was ranked as the second best system in that cluster of test cases after RiMOM system and ASMOV system, with F-measure 0.97. Still the OtO system ranked first in terms of recall, with 0.99. Figure 66 shows the overall results of all the systems.

![Figure 66: Overall results for test cases 30-37](image-url)
4.2.9. Overall Conclusions of the test cases

The OtO matching system reported an F-measure of 100% in 84% of the test cases. This means that in eight out of ten cases the system returned all the correct answers and no false answer (Figure 67).

![Figure 67: Percentages of F-measure >1](image)

Another success of the OtO system is that it was ranked as the first system among all the other systems that took part in the IIMB benchmark, in 32 test cases out of 37 as shown in Figure 68.

![Figure 68: Percentage of 1st position among other systems](image)
As described in section “3.3 The new instance matching algorithm”, a first attempt to identify the significant properties and assign weights to them is performed automatically by the OtO system. In 64% of the test cases, the domain expert did not need to configure the weights, as they were already automatically identified. This means that the instance matching was done fully automatic in less time, since only a few property matchings were conducted (Figure 69).

![Automatic Instance Matching](image)

**Figure 69: Automatic Instance Matching**

Last but not least, the Harmonic mean of the test cases has shown that the OtO system is ranked in the first position in terms of recall among all systems with recall=100% and the second position in terms of precision with precision=97% (after the ASMOV system with 100%) as shown in Figure 70.
Figure 70: Harmonic mean of test cases
Chapter 5:

5. System usage in PlugIT project
   5.1. PlugIT - Project Overview
   5.2. The Role of the OtO matching system in PlugIT
   5.3. The OtO matching system configurations
      5.3.1. OtO matching system as a web service
      5.3.2. Loading the reference ontology
      5.3.3. Matching process
      5.3.4. Matching fixed properties with fixed weights
5. Ontology Matching in the PlugIT project

Apart from the testings with the OAEI benchmark, the OtO system was also customized and used in the PlugIT European project [23]. The plugIT is a project funded by the European Union within the 7th Framework program. It started on March 1, 2009 and ended on August 31, 2011. In the next section, we present an overview of the PlugIT project, as well as some details about the context of use of the OtO system. Moreover, we analyze the customizations that have occurred in the system in order to fit into the project.

5.1. PlugIT - Project Overview

The PlugIT project argues for the necessity to align Business and Information Technology (IT) [25] as well as a change in the role of IT from that of an enabler to that of an industrial sector in its own right. Based on the assumption that businesses in different sectors of the economy will require IT for different reasons and in different ways, this project aspires to develop an IT-Socket that will realize the vision of businesses “plugging-in” to IT (Figure 71).

For the development of the “IT-Socket” a model-based approach [27] [28] [29] is applied as graphical models can be interpreted by the domain expert in the form of semi-formal graphical representations and hence can act as mediator between domain experts and formal semantic representations that can be interpreted by machines.
PlugIT has developed concepts, tools and methods summarized within the “Next Generation Modeling Framework” [26] that allow experts from both business and IT domains to use modeling languages that fit their concrete needs. This development relies on research advances in:

- a tighter involvement of domain experts when expressing formal knowledge by using graphical modeling languages as input,
- different graphical modeling languages for different views and different formal expressiveness on the IT-Socket to provide modeling languages the domain expert is used to work with, as well as
- a domain-specific notation for semantics by integrating formal concepts of semantics with the graphical notations from modeling languages.

PlugIT differentiates between Conceptual Integration and Technical Integration as shown in Figure 72. Conceptual Integration for the “Next Generation Modeling Framework” targets the integration of modeling languages currently existing on the market (e.g. business process modeling languages such as EPC, BPMN, ADONIS®, UML Activity Diagrams, BPEL). The integration within the “Next Generation Modeling Framework” targets the integration of semantic technology into different modeling languages that are required to describe the IT-Socket to enable an alignment based on semantics.

The Technical Integration is developed as the Next Generation Modeling Framework that integrates different modeling tools. The service-oriented middleware is responsible for the technical integration of heterogeneous modeling tools, the semantic kernel is responsible for the conceptual integration of different modeling languages and the Web-modeling platform is responsible for providing a common user interface.

Figure 72: plugIT Regulation Compliance and Technology Compliance
5.2. The Role of the OtO matching system in PlugIT

As described in the project overview, the main inputs of the plugIT project are graphical models from the Business side and the IT side. These models are translated and integrated into formal ontologies and the output is a graphical supported validation of the best fitting models (Figure 73).

![Figure 73: plugIT semantic model alignment](image)

The translation and integration of the models acquires the comparison between them. Figure 74 shows the process of Business and IT alignment. The Reference (IT) model and the Business model are inserted into the IT-Socket of the plugIT project. They are both converted into model ontologies from the Model Ontology Converter by using the Modeling language ontologies of each model and annotating it with the instances of the models. Later on, the model ontologies are annotated with entities from the reference ontologies in order to provide semantic lifting to the models.

After the semantic lifting of the models the OtO matching system [30] is introduced into the process. The OtO matching system matches the instances of the ontologies and returns the similarities, taking into account the annotations from the Reference Ontology and the entire taxonomy of the Reference Ontology. It should be noted at this point, that although the OtO matching system was introduced as a system that operates for the identity recognition, in PlugIT project this is not the
case. The OtO matching system in PlugIT leverages the instance matching algorithm in order to find and return the functionality similarities between the instances of the source ontology and the target ontology.

The resulting mappings are compared from the Process Comparator service, which compares the mappings given along with their similarities and determines the most suitable Business model for the given IT model.

From the process described above, the key role of the OtO matching system is obvious. By indicating the correct similarities the OtO matching system determines if the input models are compatible and if so to which extent they are compatible.

### 5.3. The Ontology to Ontology matching system configurations

For the purpose of addressing the needs of the PlugIT project, the OtO system required some modifications and configurations. These modifications are analyzed in the next sections.

#### 5.3.1. Ontology to Ontology matching system as a web service

First of all the PlugIT project follows the service-oriented Architecture (SOA) [24] and as a result the OtO system was deployed as a web service [31].
Chapter 5

The inputs of the OtO web service are:

- The URL of source ontology
- The URL of target Ontology
- The Similarity Algorithm (‘lexical’ / ‘semantic’ / ‘syntactic’)
- The Similarity threshold (0.0 – 1.0)

The output of the web service is an XML file with the correspondences between the instances of the source ontology and the target ontology. The XML file follows the xsd schema below:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="result">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="source_URI" type="xs:string"/>
        <xs:element name="target_URI" type="xs:string"/>
        <xs:element name="match" minOccurs="0" maxOccurs="unbounded">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="source" type="xs:string"/>
              <xs:element name="target" type="xs:string"/>
              <xs:element name="sim" type="xs:decimal"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Figure 75 shows a dummy web client of the OtO web service, with the web service inputs and the corresponding result.
5.3.2. Loading the reference ontology

Section “5.2 The Role of OtO matching system in PlugIT” mentions that the models are annotated with concepts from the Reference Ontology. Thus, apart from reading the annotations from the model, the OtO system reads the entire Reference Ontology and stores it as a directed acyclic graph in the DB. This serves the purpose of extracting information about the heritage properties of the annotated concepts of the Reference Ontology.

5.3.3. Matching process

The Ontology to Ontology Instance matching web service process follows the steps listed below (Figure 76):

- Reads the source and the target schema and stores all the elements
- Reads the instances and their properties and stores all the instances
- Reads the Reference Ontology and stores all the elements
- For each pair of instances it matches the “label” property and the “seeAlso” property (annotations from the Reference Ontology).
The matching is calculated using string matching algorithms and semantic matching algorithms.

- The semantic matching algorithm uses the Thesauri WordNet® to match the labels of the instances.
- A semantic similarity algorithm is also used to calculate the correspondences of the annotations from the Reference Ontology.

- A weighted average of the similarities is calculated.
- The web service returns all the matches of the instances that are above the threshold.

Figure 76: OtO instance matching web service Architecture
As you can see from the above process, the OtO matching web service does not run any schema matching algorithm to the source and the target ontology. The reason for not running a schema matching algorithm is that the schema of the ontology model gives only information about the graphical components of the models and that is of no use for semantic information we need from the models. While running the OtO web service all the source instances are compared with the target instances. The matching algorithms, as described in section “3.3.2 Instance Matching Algorithm” are:

- **The prefix algorithm** (weight 0.5)
- **The suffix algorithm**: (weight 0.5)
- **The character frequency algorithm** (weight 1.0)
- **the Di-Gram algorithm** (weight 1.0)
- **the Tri-Gram algorithm** (weight 1.0)
- **the soundex algorithm** (weight 0.5)
- **the edit distance algorithm** (weight 1.0)
- **the single error algorithm** (weight 1.0)
- **WordNet® Matcher** (weight 1.0)

In the OtO web service, all algorithms above take part in the matching algorithm and the user cannot select among them, while in the OtO system the domain expert can select the algorithms that will run.

As far as the “seeAlso” properties are concerned, Figure 66 shows that the property values are lexically and semantically compared. If the OtO web service detects that the “seeAlso” property values from the compared instances are concepts from the Reference Ontology, then it checks if they are the same concept. If they are the same concept, then it returns 1 as the “seeAlso” property similarity. If they are not the same concept, then it tries to find any heritage similarities between the concepts, for example IS_A relationship (parent-child concepts) from the reference ontology. If heritage similarities exist then it returns “seeAlso” property similarity= 0.8. In any other case it returns “seeAlso” property similarity= 0.

If the OtO system detects that the “seeAlso” property values from the compared instances are not concepts from the Reference Ontology, then the string matching algorithms mentioned above are used, apart from the WordNet matcher (as is the case in the OtO matching system). A weighted average of the returned similarities is calculated following the formula (4):
Where \( \text{sim} = \{ \text{sim}_1, \text{sim}_2, \text{sim}_3, \ldots, \text{sim}_i \} \) is the calculated property similarities from the matching algorithms and \( \text{w} = \{ \text{w}_1, \text{w}_2, \text{w}_3, \ldots, \text{w}_i \} \) the corresponding weights of each matching algorithm which are fixed from the OtO web-service.

5.3.4. Matching fixed properties with fixed weights

Most of the instances of the source and target ontologies in the PlugIT project have two properties: The first is the “label” property and the second is the “seeAlso” property. The “label” property contains the information of the label of the instance. The “seeAlso” property contains information about the annotation that has been added to the instance from the reference ontology. In other words, the “seeAlso” property has the value of the URI of a concept from the reference ontology. These two predefined properties provide the required information in order to match the models. As a result, the OtO web service has been customized to seclude these properties and compare them.

The “label” property has significance weight 0.6 and the “seeAlso” property has significance weight 0.4, which are both fixed. They are customized after exhaustive testings in the OtO web service with different model ontologies from the PlugIT project. Of course, they can be altered in the future if that is necessary.

When the similarities of the properties are calculated, a weighted similarity of the instance is calculated with the use of formula (3), where \( \text{sim}_l \) is the similarity between the “label” properties of the instance and \( \text{sim}_s \) is the similarity between the “seeAlso” properties of the instance:

\[
\text{Instance sim} = \frac{\text{sim}_l \cdot 0.6 + \text{sim}_s \cdot 0.4}{0.4 + 0.6}
\]
6. Conclusion and Future work
6. Conclusion and Future work

In this Thesis, we presented the OtO matching framework, which is a multi-strategy ontology to ontology matching system. It was shown that it is a domain independent and a fully customizable matching system at any level, i.e. schema level or instance level.

In order to evaluate the matching system and test the instance matching algorithm we have used the ISLab Instance Matching Benchmark (IIMB). The IIMB is a benchmark that is generated automatically, starting from one data source that is automatically modified according to various criteria. The original data source contains OWL/RDF data about actors, sports, persons, and business firms provided by the OKKA European project. The benchmark results have shown that the OtO matching system returned one of the best results. In particular, it returned the best results in recall and the second best results in precision, among seven systems.

Furthermore, the OtO matching system has been deployed in an EU Project, the PlugIT project. After customizing the system to fit in the project, it was used in order to find functional correspondences between two ontologies. The OtO matching system, performed very well in the PlugIT project.

The success of the system is grounded on the carefully designed and analyzed matching process. Specifically speaking, the success is due to the leverage of (1) the rich semantic knowledge we gain from the output mappings of the schema matching process, (2) the implicit knowledge of domain expert by (semi-)automatically capturing the identification power of the properties and (3) the probability calculation of the result’s truth, in order to accurately and efficiently detect the ontology instances that represent the same real-world entity.

On the other hand, the OtO matching system has a few weak points, which give us the opportunity to work on it in the future. The first point is that it is not very efficient when it comes to large ontologies with millions of instances. This would take too long runtime and would have strong requirements on running environment. One way to overcome this problem in the future would be to use the MapReduce Framework [90]. The MapReduce Framework is a programming framework used to simplify data processing across massive data sets.

As far as the User Interface is concerned, we aim for making the OtO matching system more flexible and efficient to use. By more flexible, we mean to give the freedom to the user to either accept the instance matching results, decline them, or to even add more. By efficient, we mean to allow to the user to understand how and why each matching result was chosen as a valid results. That means that for
each result the domain expert can see on what bases the matching result is considered as a true match.

Our next goal is to participate in the next instance matching benchmark of the Ontology Alignment Evaluation Initiative, which is in the 2012 campaign. Until then, we will work on the shortcomings mentioned above, in order to be fully prepared to reach the top once more.
Appendix A
Appendix A

Java Code for Instance Matching Algorithm

```java
public static String instanceSimilaritiesWithWeights(ArrayList checked, ArrayList weights, ArrayList nodeTypes, int[] models, String mapping, String instanceThreshold, int[] strAlgorithm, boolean AnonymousFlag, boolean ParentsFlag, boolean ThingFlag){

    Similarity sim=new Similarity();
    sim.setUseWordNet(false);
    long time = System.currentTimeMillis();
    String previousSName="";
    String previousTName="";
    Float previousSim=(float)0.0;
    String xml="";
    String rdf="";
    String sPropertyName="";
    String sPropertyValue="";
    String tPropertyName="";
    String tPropertyValue="";
    Repository repository = new Repository("webSM","root","admin");

    Similarity similarity=new Similarity();

    DOMParser parser = new DOMParser();
    try {
        parser.parse(new InputSource(new java.io.StringReader(mapping)));
    } catch (SAXException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    }
    Document doc = parser.getDocument();
    String abox=repository.getInstanceNS(models[1]);

    ArrayList <InstanceMatch> instanceMatches=
    new ArrayList <InstanceMatch>();

    if (nodeTypes.contains("6")){//Node.ELEMENTTYPE

        for (int i=0;i<checked.size();i++){

            if(nodeTypes.get(i).equals("6")){//Node.ELEMENTTYPE

                String sourceName=
                getNameFromXML(checked.get(i).toString(),doc,"sourceName");
                String targetName=
                getNameFromXML(checked.get(i).toString(),doc,"targetName");

                ArrayList<Node> sourceInstanceNodes=
                repository.selectNodeInstances(sourceName, models[0]);

                ArrayList<Node> targetInstanceNodes=
                repository.selectNodeInstances(targetName, models[0]);

                for (int j=0;j<sourceInstanceNodes.size();j++){
                    Node sourceInstance=
                    sourceInstanceNodes.get(j);
                    for (int k=0;k<targetInstanceNodes.size();k++){
                        Node targetInstance=
                        targetInstanceNodes.get(k);

                        float similarity = sim.calculateSimilarity(sourceInstance, targetInstance);
                        if (similarity > instanceThreshold) {
                            InstanceMatch instanceMatch = new InstanceMatch();
                            instanceMatch.setSource(sourceInstance);
                            instanceMatch.setTarget(targetInstance);
                            instanceMatch.setSimilarity(similarity);
                            instanceMatches.add(instanceMatch);
                        }
                    }
                }
            }
        }
    }
```
Appendix A

```java
ArrayList<Node> targetInstanceNodes = repository.selectNodeInstances(targetName, models[1]);

if (AnonymousFlag) {
    ArrayList<Node> AnontargetInstanceNodes = repository.selectNodeInstances("Anonymous", models[1]);
    if (AnontargetInstanceNodes!=null) {
        if (targetInstanceNodes==null){
            targetInstanceNodes=AnontargetInstanceNodes;
        }
        else{
            for(int s2=0;s2<AnontargetInstanceNodes.size();s2++) {
                targetInstanceNodes.add(AnontargetInstanceNodes.get(s2));
            }
        }
    }

    else {
        for(int s2=0;s2<AnontargetInstanceNodes.size();s2++) {
            targetInstanceNodes.add(AnontargetInstanceNodes.get(s2));
        }
    }
}

if (ParentsFlag) {
    String parentTargetName = repository.getParentNodeName(targetName, models[1]);
    if (!parentTargetName.equals("noName")) {
        ArrayList<Node> PtargetInstanceNodes = repository.selectNodeInstances(parentTargetName, models[1]);
        if (PtargetInstanceNodes!=null) {
            if (targetInstanceNodes==null){
                targetInstanceNodes=PtargetInstanceNodes;
            }
            else{
                for(int s3=0;s3<PtargetInstanceNodes.size();s3++) {
                    targetInstanceNodes.add(PtargetInstanceNodes.get(s3));
                }
            }
        }
    }
}

if (ThingFlag) {
    ArrayList<Node> TtargetInstanceNodes = repository.selectNodeInstances("Thing", models[1]);
    if (TtargetInstanceNodes!=null) {
        if (targetInstanceNodes==null){
            targetInstanceNodes=TtargetInstanceNodes;
        }
        else{
            for(int s4=0;s4<TtargetInstanceNodes.size();s4++) {
                targetInstanceNodes.add(TtargetInstanceNodes.get(s4));
            }
        }
    }
}
```

Evangelia Daskalaki

Master Thesis | University of Crete | Computer Science 114
for (int l=0;l<sourceInstanceNodes.size();l++){
    Node sInstanceNode=
    sourceInstanceNodes.get(l);
    ModelNodeObject sourceInstanceObj=
    (ModelNodeObject) sInstanceNode.getObject();
    HashMap sInstanceProperties=
    repository.getInstanceProperties(sourceInstanceObj.getName(),models[0]);
    String sname=sourceInstanceObj.getName();

    for (int r=0;r<targetInstanceNodes.size();r++){
        boolean keyweight=false;
        Node tInstanceNode=targetInstanceNodes.get(r);
        ModelNodeObject targetInstanceObj=
        (ModelNodeObject) tInstanceNode.getObject();
        HashMap tInstanceProperties=
        repository.getInstanceProperties(targetInstanceObj.getName(),models[1]);
        String tname=targetInstanceObj.getName();

        int sInstancePropertiesSize=sInstanceProperties.size();
        int tInstancePropertiesSize=tInstanceProperties.size();
        float[][] propertiesSimMatrix=
        new float[sInstancePropertiesSize][tInstancePropertiesSize];
        int y=0; int u=0;
        Set Sourceset = sInstanceProperties.entrySet();
        Set Targetset = tInstanceProperties.entrySet();

        float sumWeights=new Float(0.0);
        for ( Iterator it = Sourceset.iterator(); it.hasNext(); ){
            u=0;
            Map.Entry me = (Map.Entry)it.next();
            sPropertyName=me.getKey().toString();
            sPropertyValue=me.getValue().toString();
            for ( Iterator ite = Targetset.iterator(); ite.hasNext(); ){
                Map.Entry map = (Map.Entry)ite.next();
                tPropertyName=map.getKey().toString();
                tPropertyValue=map.getValue().toString();
                int mappingResult=checkIfMappingFromXML(sPropertyName,tPropertyName,doc);
                if (mappingResult==0){
                    propertiesSimMatrix[y][u]=(float)0.0;
                }
            }
        }
    }
}

Appendix A

Evangelia Daskalaki

Master Thesis | University of Crete | Computer Science
else{
    if (checked.contains(mappingResult)) {

        int index = checked.indexOf(mappingResult);

        String userWeight = (String) weights.get(index);
        float uWeight = new Float(userWeight);
        if (uWeight > (float) 0.0) {

            if (!keyweight) {

                if (uWeight >= (float) 0.50)
                    keyweight = true;

            }

            if (sPropertyValue.indexOf(abox) >= 0) {
                sPropertyValue = returnValue(repository, sPropertyValue, models[0]);
            }

            if (tPropertyValue.indexOf(abox) >= 0) {
                tPropertyValue = returnValue(repository, tPropertyValue, models[1]);
            }

            propertiesSimMatrix[y][u] =
                similarity.computeNameSimilarityByEva(sPropertyValue, tPropertyValue, strAlgorithm);

            propertiesSimMatrix[y][u] = uWeight * propertiesSimMatrix[y][u];

            sumWeights = sumWeights + uWeight;

        }

        else propertiesSimMatrix[y][u] = (float) 0.0;
        break;
    }
}

y = y + 1;
}

float similar =
    Combination.computeSetSimilarity(propertiesSimMatrix, 3, sumWeights);
propertiesSimMatrix = null;
tInstanceProperties = null;
if (similar >= Float.valueOf(instanceThreshold).floatValue() && keyweight == true) {
    if (previousSName.equals(sname) && previousSim < similar) {
        if (instanceMatches != null) {
            xml = removeMatchingInstance(previousSim, previousSName, previousTName, xml); }
    
        xml = xml + addMatchingInstance(similar, sname, tname);
        previousSName = sname;
        previousTName = tname;
        previousSim = similar;
    } else if (previousSName.equals(sname) && previousSim > similar) {
        continue;
    } else {
        xml = xml + addMatchingInstance(similar, sname, tname);
        previousSName = sname;
        previousTName = tname;
        previousSim = similar;
    }
}

xml = "<sourceModel>" + models[0] + "</sourceModel><targetModel>" + models[1] + "</targetModel >" + xml;

time = System.currentTimeMillis() - time;
int minutes = (int) ((time / 1000) / 60);
System.out.println("/	/	/	/	/	/	/	/	/	/	/	/	/	/	/	/	/	/	/	/	/	/	/	/	/	/	/	/	/	/	/	/	/	/	/	/	/	/ TIME FOR THE PROCESS IS : "+ minutes);
sim.setUseWordNet(true); }
else {
    xml = instanceSimilaritiesWithWeightsWithoutMapping(checked, weights, nodeTypes, models, mapping, instanceThreshold, strAlgorithm);
}

return xml;
Bibliography


Bibliography


[23] The PlugIT project, http://plug-it-project.eu


[26] Next Generation Modelling Framework Portal of the 2nd plugIT Prototype: http://83.65.190.84/plugIT/workbench/


[31] Ontology matching web-service endpoint: http://139.91.183.3:8080/MatchingToolWithLabels/services/Match?wsdl


Bibliography


Bibliography

[77] Indrajit Bhattacharya and Lise Getoor: “Entity resolution in graphs”, In Mining Graph Data, Wiley & Sons, 2006
[80] Carlo Meghini, Martin Doerr, Nicolas Spyratos: “Managing Co-reference Knowledge for Data Integration”, EJC 2008: 224-244
[84] Chao Wang, Jie Lu, Guangquan Zhang: “Integration of Ontology Data through Learning Instance Matching”, Web Intelligence 2006: 536-539