Knowledge-based Data Extraction Workbench

For Eclipse

THESIS

Presented in Partial Fulfillment of the Requirements for the Degree Master of Science

in the Graduate School of The Ohio State University

By

Jithendra Kumar Rangaraj, M.Sc.

Computer Science and Engineering Graduate Program

The Ohio State University

2012

Thesis Committee:

Prof. Rajiv Ramnath, Advisor

Prof. Jay Ramanathan
ABSTRACT

Data from various sources are in heterogeneous underlying formats. Combining and extracting data from all such sources will require the user to generate mappings between the different formats which vary on a case by case basis. The solution proposed is a knowledge-based approach to solve the problem of data extraction from different source formats: Build ontologies from source files, find mappings between ontologies, use reasoners to extract data, and convert ontologies to desired formats. There are several ontological tools in the market for converting, mapping and reasoning ontologies. Rather than re-inventing the wheel, the solution proposed is to develop a workbench of all the existing tools in the ontological domain and those that may emerge. This is achieved by an extensible Eclipse plugin based architecture called Knowledge-based Data Extraction (KDE). In addition to the architectural aspects of KDE, this thesis makes the following contributions: a taxonomy of terms used in the ontology domain, identification of the capabilities missing in existing ontological tools, and a scenario-based comparison of two ontology mapping algorithms used in the implementation of the architecture.
Dedication

To my family and friends.
ACKNOWLEDGEMENTS

I take immense pleasure in thanking the people who helped me in realizing my dream of writing this thesis.

I would like to extend my heartfelt and sincere thanks to my advisor, Dr. Rajiv Ramnath, whose guidance, support, motivation and kind heartedness helped me in doing this thesis. With his quench for knowledge and thought provoking questions, he kept directing me along the right path throughout the journey.

I would like to thank my committee member, Dr. Jay Ramanathan, for providing her valuable insights, comments and appreciation. Her comments and feedback helped me in gaining useful insights.

I express my sincere gratitude to Dr. Angelo Alonzo, for his involvement, and support. A special thanks to him for generously sharing his findings with me during the initial stages of this thesis.

Finally, I would like to thank my loving parents for constantly encouraging and supporting me in all my endeavors.
VITA

2004 to 2009 ......................................................... M.Sc. (Integrated 5 years),

Computer Science,

College Of Engineering,

Anna University, India

2009 to 2011 ......................................................... Programmer Analyst,

Cognizant Technology Solutions,

Chennai, India.

2012 to 2012 ......................................................... Graduate Research Assistant,

Department of Computer Science

and Engineering,

The Ohio State University,

USA.

2012 to 2012 ......................................................... College Intern, Cisco Systems,

San Jose, USA

FIELDS OF STUDY

Major Field: Computer Science and Engineering
TABLE OF CONTENTS

ABSTRACT .................................................................................................................. ii
ACKNOWLEDGEMENTS ............................................................................................... iv
VITA ............................................................................................................................... v
LIST OF TABLES ........................................................................................................... ix
LIST OF FIGURES ......................................................................................................... x
CHAPTER 1 ..................................................................................................................... 1
  1.1 Introduction ........................................................................................................... 1
  1.2 Motivation ............................................................................................................ 1
  1.3 Problem Statement .............................................................................................. 3
  1.4 Solution Approach ............................................................................................. 4
  1.5 Contributions ..................................................................................................... 5
  1.6 Organization of the Thesis .................................................................................. 6
CHAPTER 2 ..................................................................................................................... 7
  2.1 Terms and Definitions ......................................................................................... 7
    2.1.1 Ontology ........................................................................................................ 7
    2.1.2 Vocabulary ..................................................................................................... 8
    2.1.3 Resource Description Framework (RDF) ....................................................... 8
    2.1.4 Statement ...................................................................................................... 9
    2.1.5 Uniform Resource Identifier (URI) ............................................................... 9
    2.1.6 RDF triples .................................................................................................. 10
    2.1.7 RDF graph ................................................................................................. 10
    2.1.8 RDF serialization ....................................................................................... 11
    2.1.9 Notation 3 ................................................................................................... 11
    2.1.10 Terse RDF Triple Language (TURTLE) ..................................................... 11
    2.1.11 N Triples ................................................................................................... 11
    2.1.12 Simplified Protocol and RDF Query Language (SPARQL) ....................... 12
    2.1.13 RDF Schema (RDFS) ............................................................................... 12
    2.1.14 RDF in Attributes (RDFa) ....................................................................... 12
    2.1.15 Web Ontology Language (OWL) ............................................................... 12
    2.1.16 Ontology mapping ..................................................................................... 13
  2.2 Algorithms for Ontology Mapping ....................................................................... 15
  2.3 Classification of Ontology Mapping Techniques .................................................. 16
    2.3.1 Matching dimensions ............................................................................... 16
    2.3.2 Tree based on matcher granularity/ input interpretation ............................. 19
    2.3.3 Tree based on kind of input ....................................................................... 21
    2.3.4 Basic Matching Techniques ...................................................................... 22
BIBLIOGRAPHY

CHAPTER 4

4.1 Conclusion ................................................................. 93
4.2 Future Work ............................................................... 94
  4.2.1 Deploying KDE into psife ........................................ 94
  4.2.2 Designing GUI based on cognitive engineering principles and data visualization. ........................................... 95
  4.2.3 Creating the “Best” ontology mapping ............................ 95
  4.2.4 KDE for service-based platform. ................................... 96
  4.2.5 Graphical ontology editing plugins ............................... 96

CHAPTER 3

3.1 Requirements of KDE ................................................... 70
3.2 Architecture of Knowledge-Based Data Extraction System ............ 71
  3.2.1 Controller ............................................................ 72
  3.2.2 Transformation broker ............................................. 73
  3.2.3 Ontology-mapping registry ......................................... 74
  3.2.4 Ontology-mapping engine .......................................... 74
  3.2.5 Reasoning engine .................................................. 75
  3.2.6 Output transformation engine ...................................... 75
3.3 Implementation of KDE .................................................. 76
  3.3.1 Jena framework .................................................... 76
  3.3.2 Transformation tools .............................................. 76
  3.3.3 Ontology mapping tools .......................................... 77
  3.3.4 Reasoning and inference engines .................................... 79
  3.3.5 Output transformation tools ....................................... 80
3.4 Evaluation .................................................................. 81
  3.4.1 Example scenario .................................................. 81
  3.4.2 Manual process .................................................... 82
  3.4.3 Using KDE workbench ............................................. 84
3.5 Future Work ................................................................ 89
  3.5.1 Gold standard ...................................................... 89
  3.5.2 Mappings by S-Match and CROSI ................................. 89

CHAPTER 4 by S-Match and CROSI ........................................... 89

4.1 Conclusion .................................................................. 93
4.2 Future Work ............................................................... 94
  4.2.1 Deploying KDE into psife ........................................... 94
  4.2.2 Designing GUI based on cognitive engineering principles and data visualization. ........................................... 95
  4.2.3 Creating the “Best” ontology mapping ............................ 95
  4.2.4 KDE for service-based platform. ................................... 96
  4.2.5 Graphical ontology editing plugins ............................... 96

BIBLIOGRAPHY ................................................................. 97
LIST OF TABLES

Table 2.1: Full vs. partial structural match (example) ...................................................... 20
Table 2.2: Store and E-commerce website ontology - Class relationship .......................... 30
Table 2.3: Store and Book library ontology - Class Relationships .................................. 30
Table 2.4: Store and University library pre-computed similarity measures ....................... 33
Table 2.5: Similarity measures – Book library and University library ............................... 34
Table 2.6: Concept Lattice Mappings ............................................................................... 37
Table 3.1: Mappings by S-Match ..................................................................................... 87
Table 3.2: Mappings by CROSI ....................................................................................... 88
Table 3.3: Ground truth mapping ..................................................................................... 89
Table 3.4: Mapping counters: SMatch Vs. CROSI ............................................................ 91
Table 3.5: Comparison of the quality metrics S-Match Vs. CROSI ..................................... 92
LIST OF FIGURES

Figure 2.1: A simple RDF triple represented as a RDF graph ........................................10
Figure 2.2: Ontology mapping example--car domain ....................................................15
Figure 2.3: Classification of the schema-matching techniques ........................................18
Figure 2.4: E-commerce and Book Library Ontology .....................................................23
Figure 2.5: Normalized Levenshtein Values .................................................................24
Figure 2.6: Example: Two ontologies in Electronics domain,Tree-based .......................28
Figure 2.7: Classes from retail store ontology. .............................................................29
Figure 2.8: University Library ontology (fragment) .......................................................33
Figure 2.9: Concept Lattice .........................................................................................36
Figure 2.10: Example ontologies representing image documents ....................................38
Figure 2.11: Components of psife architecture .............................................................48
Figure 2.12: Model-based Tool Integration Patterns.......................................................51
Figure 2.13: Snapshot of the initial user interface of the Opinion Mining Tool ...............55
Figure 2.14: Cognitively Engineered User interface .....................................................59
Figure 2.15: Eclipse plugin architecture - Extension point mechanism .........................66
Figure 2.16: Extension Point mechanism- Example .....................................................67
Figure 2.17: Extension Point mechanism- Host plugin’s plugin.xml .............................68
Figure 2.18: Extension Point mechanism- Extender plugin’s plugin.xml .......................68
Figure 3.1: Knowledge-based Data Extraction (KDE) workbench architecture .............72
Figure 3.2: KDE- Snapshot of input selection .............................................................77
Figure 3.3: KDE- Snapshot of ontology mapping algorithm selection ..........................79
Figure 3.4: User interface for selecting the reasoner for querying the ontologies ............80
Figure 3.5: A snapshopt of Company A’s (Seller) data in Excel format .......................81
Figure 3.6: A snapshot of Company B’s (Buyer) data in CSV format ............................81
Figure 3.7: Ontology representation of Company A (Seller) data from File1 .................82
Figure 3.8: Ontology representation of Company B (Buyer) data from File2 ...............84
Figure 3.9: Mapping file for XLWrap corresponding to File1 in TriG syntax ...............85
Figure 3.10: Mapping file for XLWrap corresponding to File2 in TriG syntax ...............85
Figure 3.11: SPARQL query .......................................................................................88
CHAPTER 1

1.1 Introduction

Data is everywhere, floating in the web, on your mobile device, in a hard disk, distributed in a network, in databases, etc., and is collected from various sources through stand-alone applications, sensors, mobile devices, etc. [62, 61]. Evolving web technology has opened a new era of computing by adopting an ontological approach to represent data in a format comprehensible by both humans and machines [39]. With new tools and APIs to deal with ontologies, each one focusing on different aspects such as data conversion, ontology mapping, reasoning, etc., an application could be to see if the tools could be used in a collaborative way. The focus of this thesis is a workbench that allows ontological tools to collaborate with each other, thereby providing the user with a collection of capabilities for the extraction of data from different underlying formats.

1.2 Motivation

Researchers work by analyzing data they have gathered from several sources using various tools to extract knowledge and information [61]. Autonomous systems have data in heterogeneous formats such as CSV, XML, and JSON. With the goal of extracting semantic information from data, a researcher might manually build ontologies from the datasets, use ontology mapping tools to see what kind of mapping the tools generate,
merge ontologies, and use inference engines to reason the datasets. This thesis will
describe a common platform to make use of existing ontological tools and to allow future
tools to be included in the infrastructure for data extraction and analysis from such
sources. Here is a scenario that details how researchers might analyze data.

Yale Heart Study is a heart attack-based survey study website that collects data
about patients regarding their experience during a heart attack. The survey system
collects detailed information from the patients, such as the time of the onset of severe
symptoms, the time when the patient called for help, the mode by which they called for
help, their current medications, the time when the help arrived, and many more details.\(^1\)

The researcher who uses the website for his study on heart attacks has posted
advertisements in websites such as Facebook, Google, and Yahoo. These websites
provide the researcher with information related to the geographical location from which
the clicks to the website originated.

With the data available, how feasible is it to expect the researcher to find a
relation between the weather at a geographical region and the heart attacks that took place
in the region? One approach is to comb the datasets manually, look up the weather
information for the datasets in the web, and try to find correlations. Another approach is
to write an application in a specific programming language that could do the job of
combining the data from all the sources, look in the web for the weather information and,
based on some logic, combine the data sets and display the results in a human readable
format for further analysis. However, these approaches do have shortcomings. Manually
analyzing the data sets is a cumbersome task. An application to combine the data could

\(^1\) In accordance with HIPAA[1] the website does not collect demographic information about the patients.
be a simple one if the data files are in the same format, but due to the varied nature of the data sources the data files will be in different formats. To solve this problem the data has to be converted into a common format, for example CSV. Converting an excel file into a CSV is easy whereas converting an XML or JSON to CSV is difficult. Although this could be achieved by generating mappings from the source formats to CSV, the mappings might vary from on a case by case basis. A simple yet powerful solution to this problem is to build ontologies from the source files and use them to extract the needed data.

1.3 Problem Statement

Researchers tend to work on data collected from heterogeneous sources. As a direct consequence, data they analyze is in different underlying formats. Combining data from different underlying formats is cumbersome. For this, researchers make use of converters, mapping software, visualization tools, etc [61].

The combined data could be used for further analysis by feeding it into systems that perform mathematical calculations to interpret undiscovered information. The market is filled with tools, applications, and frameworks that aid developers in parsing data of different formats, manipulating them, and converting them to desired formats. A great deal of research and work has been invested in such tools that a new system to extract data should be capable of utilizing and adapting the existing tools to work in an integrated manner.
1.4 Solution Approach

The problem described in the previous section can be decomposed into the following categories:

a) Tool Integration: Integrate existing engineering tools that researchers use for knowledge extraction.

b) Data Extraction: Extraction of data from heterogeneous formats (structured).

c) User interaction: Providing the user with an integrated and a flexible access to existing tools by means of a GUI for input.

d) Data Visualization: Visualization of data extracted from the combined form.

The combined data might be structured or unstructured.

Mampilly and Ramnath [4] address the problem of orchestration of engineering tools using the Process-Oriented Service Integration Framework for Eclipse (psife). We will see more details about this thesis in section 2.5. Ragavan and Ramnath [5] address the problem of user interaction and data visualization by developing a GUI based on cognitive engineering principles to provide an integrated access to analysis techniques and to visualize unstructured data. More information about this thesis can be found in section 2.7.

The focus of this thesis is the data extraction from various sources (structured). The solution proposed addresses the problem as follows:

a) Convert the data files into ontologies representing the data.

b) Employ existing ontology mapping tools to generate ontology mappings.

Combine the data based on the ontology mapping to generate bridged ontologies.
c) Use reasoners and inference engines to query the bridged ontologies for the desired information.

We have employed, an extensible plugin system to achieve this. Plugins are written in a standard object-oriented fashion and utilize an inheritance structure to minimize code duplication [2]. Plugins are the solution for extending and integrating existing tools [3]. As new tools and techniques evolve, they can be plugged into the system by conforming to the plugin contract. The data from various sources is converted to RDF triples representing ontologies. The solution exploits all existing tools by wrapping them up into plugins and allowing the user to choose between the available plugins to cater to his needs.

1.5 Contributions

This thesis makes the following contributions. First, it provides an exhaustive taxonomy of terms used in the ontology domain. Second, it identifies missing capabilities in existing ontological tools. Third, it proposes an integrated workbench of tools for data extraction that enables existing tools to collaborate. This is accomplished by an extensible plugin-based data extraction architecture to accommodate existing tools that convert data from various sources to ontologies; perform automatic ontology mapping; and make use of existing reasoners and inference engines to extract information out of the merged ontology. Finally, it provides a scenario-based comparison of the ontology mapping algorithms used during the implementation of the architecture.
1.6 Organization of the Thesis

Chapter 2 provides formal definitions of terms used in the ontology world, elaborates the classification of ontology mapping techniques, discusses the related work, lists the existing tools in the ontology domain and identifies their missing capabilities. Chapter 3 describes and discusses the proposed plugin-based architecture for data extraction, evaluates the architecture by means of an exemplar scenario and provides a scenario-based comparison of the ontology mapping algorithms. Chapter 4 concludes the thesis by summarizing and providing directions for future work.
CHAPTER 2

In this chapter we attempt to define the terms used commonly in the ontology world. We provide a classification of ontology mapping techniques and describe the matching systems that were used to develop the classification. We discuss the existing ontological tools and identify their missing capabilities. Finally, we discuss other related research.

2.1 Terms and Definitions

Although the focus of this thesis is not to provide a complete guide to all the terms used in the semantic web, we attempt to define the most commonly used ones.

2.1.1 Ontology

Although there is no formal definition for ontology, there are a few widely accepted ones.

Thomas Gruber is one of the earliest to come out with the definition of ontology. He says, “Ontology is an explicit specification of a conceptualization” [37].

According to Strassner [6], “An ontology is a formal, explicit specification of a shared, often machine-readable, vocabulary. Its meaning, in the form of entities and relationships between them, intends to describe some knowledge in a given domain.”

- “Formal” denotes that ontology should be expressed using a formal grammar.
• “Explicit” denotes entities, relationships, and constraints defined in a declarative language for knowledge representation.

• “Shared” means that all users of an ontology will represent a concept using the same or equivalent set of entities and relationships.

• Domain refers to the content of the universe of discourse being represented by the ontology.

In simple words, ontology describes knowledge about the domain of interest by means of entities and relationships. It defines a common vocabulary for researchers who need to share information in a domain.

2.1.2 Vocabulary

In semantic web literature, vocabularies are the basic building blocks of reasoning and inference. There is no clear distinction between the terms “Vocabulary” and “Ontology,” yet the general notion is to use the term “Ontology” for a complex and formal collection of terms. Vocabulary defines concepts and relationships (referred to as “terms”). The basic use of vocabularies is to classify terms, characterize relationships, and define constraints on terms.

2.1.3 Resource Description Framework (RDF)

Resource Description Framework (RDF) is a language for representing identifiable resources such as documents on the World Wide Web. In other words, it is the framework for representing information and metadata about resources in the Web. RDF is the standard model for interchange of data across the web and has features that
provide a solution to the problem of merging data with different underlying schemas. It enables structured and semi-structured data to be mixed and shared between applications of different genre.

More formally,

“RDF is an assertional language intended to be used to express propositions using precise formal vocabularies, particularly those specified using RDF Schema, for access and use over the World Wide Web, and is intended to provide a basic foundation for more advanced assertional languages with a similar purpose. The overall design goals emphasise generality and precision in expressing propositions about any topic, rather than conformity to any particular processing model” [9].

2.1.4 Statement

The semantic web is so called because of its ability to represent information in a way that is similar to expressing it in a natural language like English. In general, a statement is used to express information about a subject. Grammatically, a statement can be divided into a subject (the thing that the statement describes), predicate (a specific property of that thing the statement describes), and an object (the value of that property). The RDF model is composed of a set of statements.

2.1.5 Uniform Resource Identifier (URI)

In order to make meaningful statements in RDF, the subject of the RDF statement has to be identified in some unique way. Uniform Resource Identifier (URI) is a unique
way of identifying resources. RFC 3986 defines a URI as “a compact sequence of characters that identifies an abstract or physical resource” [8].

### 2.1.6 RDF triples

The underlying structure of any RDF expression is a collection of RDF triples. RDF provides a flexible method to decompose knowledge into small pieces, called triples, with some rules about the semantics (meaning) of those pieces. An RDF triple is comprised of the three components of a Statement:

1. **Subject (Resource):** This is a URI or blank node.
2. **Predicate (Property):** This is a URI.
3. **Object (Class):** This is a URI, a literal or a blank node.

An RDF triple represents a statement of a relationship between the subject and object. Literal values are raw text that represents the values of the “object” part of an RDF triple.

### 2.1.7 RDF graph

A collection of RDF triples forms an RDF graph. Each triple in the graph is represented by node-arc-node link. Figure 2.1 represents an RDF triple as a RDF Graph.

![RDF Graph Diagram](image-url)

**Figure 2.1** A simple RDF triple represented as a RDF graph
2.1.8 RDF serialization

RDF uses the Extensible Markup Language (RDF/XML) and Notation 3 (N3) in order to serialize the statements in a machine readable manner.

2.1.9 Notation 3

Notation 3 was designed as an alternate means for serializing RDF models with human readability as its goal. The format was developed by the inventor of the World Wide Web, Tim Berners Lee and others in the Semantic Web community. It extends beyond the boundaries of RDF by adding features from first-order logic (N3 Logic), but it turns out to be much slower than RDF.

2.1.10 Terse RDF Triple Language (TURTLE)

Terse RDF Triple Language (TURTLE) is a concrete compact syntax for representing an RDF graph in a textual form. TURTLE is a subset of the N3 form. Turtle has a concise syntax that is easy to write by hand and hence is popular in the developer community. It also has syntax to generate triples.

2.1.11 N Triples

N Triples is a format to represent the subject, predicate, and object notions of a statement in both machine and human readable manner. It is a line-based plain text serialization format for the RDF graphs. It is a subset of TURTLE and removes TURTLE’s concise syntax to represent RDF triples.
2.1.12 Simplified Protocol and RDF Query Language (SPARQL)

As the name suggests, Simplified Protocol and RDF Query Language (SPARQL) is a language for querying data stored in RDF format. The protocol denotes the protocol that is used to convey queries from the query client to the query processor. SPARQL queries are based on Turtle/N3 format.

2.1.13 RDF Schema (RDFS)

RDF properties (predicates of a sentence) represent relationships between resources (subjects) but do not provide mechanisms for describing the properties or the relationships between these properties or other resources. RDF Schema (RDFS) is an RDF vocabulary description language that defines classes and properties in order to describe the classes, properties, and relationships between the resources.

2.1.14 RDF in Attributes (RDFa)

RDFa is a technique that allows adding structured data to HTML documents for machine-readability. It provides a set of markup attributes that augment the HTML code with machine-readable hints.

2.1.14 Web Ontology Language (OWL)

Web Ontology Language (OWL) represents a family of languages that are used to specify ontologies. It specifies formal semantics and RDF-XML-based serializations of the semantic web. It was designed for applications that needed to process information instead of just presenting information. OWL builds on top of RDFS, describing more properties and classes. OWL allows the defining of inverse properties, transitive
properties, disjoint classes, new classes as unions, or intersections of other classes, etc. OWL has three variants in the order of increasing expressiveness, namely OWL Full, OWL DL, and OWL Lite.

- **OWL Lite**: OWL Lite provides support for a classification hierarchy and simple constraints. The expressivity limitations that it specifies are intended for keeping the language constructs minimal in order to make it simple for providing tool support.

- **OWL DL**: OWL DL is so named because of its correspondence with description logics. It provides maximum expressiveness while retaining computational completeness and decidability. Though it includes all OWL language constructs, it has some restrictions (for example, a class cannot be an instance of another class).

- **OWL Full**: OWL Full contains all OWL constructs and provides free, unconstrained use of RDF constructs. It has maximum expressiveness but provides no computational guarantees and hence reasoners are unlikely support every feature of OWL Full.

### 2.1.15 Ontology mapping

Informally, mapping an ontology O1 to another ontology O2 means that, for each entity (Concept C, Relation R, or Instance I) in ontology O1, we try to find a corresponding entity in O2 that has the same intended meaning.

More formally, Kalfoglou and Schrolemmer [11] define an ontology algebraically as a pair \( O = (S, A) \), in which \( S \) denotes (ontological) signature and \( A \) denotes a set of
(ontological) axioms. Here the signature describes the vocabulary modeled as a mathematical structure. For instance, it could consist of a class hierarchy of symbols modeled as a partial ordered set along with the set of relation symbols whose arguments are defined over the concept hierarchical concepts. The interpretation of the vocabulary in some domain of discourse is what the axioms denote.

Ontology mapping can be considered as the task of relating the vocabulary of two different ontologies in the same domain. A total ontology mapping is defined as “morphism,” structure-preserving mappings between ontological structures. Given two ontologies, $O_1 = (S_1, A_1)$ and $O_2 = (S_2, A_2)$, an ontology mapping from $O_1$ to $O_2$ is denoted by a morphism $f : S_1 \rightarrow S_2$ of ontological signatures in such a way that $A_2 \models f(A_1)$, i.e., all the interpretations that satisfy $O_2$’s axioms also satisfy $O_1$’s translated axioms. In order to complete the notion of ontology mapping, we define a weaker notion of ontology mapping as a partial ontology mapping from $O_1$ to $O_2$ if there exists a sub-ontology, $O_1' = (S_1', A_1')$ ($S_1'$ a subset of $S_1$ and $A_1'$ a subset of $A_1$), such that there is a total mapping from $O_1'$ to $O_1$ [11].
Figure 2.2 Ontology mapping example—car domain. The yellow nodes represent classes of elements, the green nodes represent a relationship between the classes, and the red boxes are instantiations of the classes [22].

Figure 2.2 illustrates the concept of ontology mapping between ontologies in the “Car” domain. It contains two ontologies identified by “Ontology 1” and “Ontology 2.” The ontology mapping between the classes and instances of both ontologies is shown in the figure.

2.2 Algorithms for Ontology Mapping

The algorithms for ontology mapping can be considered “exact” and “approximate.” Exact algorithms are the ones that try to achieve exact mappings without worrying about the performance, whereas the approximate ones achieve approximate mappings with improved performance.
2.3 Classification of Ontology Mapping Techniques

Euzenat and Shvaiko [27] have worked on classifying ontology-matching techniques. The actual classification is shown in Figure 2.3. The classification itself is an adapted version of their original schema-matching classification. Here we will see an overview of their work.

2.3.1 Matching dimensions

The techniques and algorithms that exist for ontology matching have been classified based on the following:

1. Input dimensions: The input of the algorithms.
2. Process dimensions: The characteristics of the matching process.
3. Output dimensions: The output of the algorithms.

The classification of ontology matching techniques can be considered as two trees.

1. The first is based on granularity of the matcher and the input interpretation.
2. The second is based on the kind of input, say string, structure, model, or the actual data itself, which is used by elementary matching techniques.

The two trees share their leaves and the nodes at the leaves represent basic matching techniques.

Euzenat and Shvaiko[27] considered the following systems for their classification of ontology mapping: Anchor-PROMPT, Artemis, COMA, COMA++, Cupid, NOM, QOM, OLA, Similarity Flooding in Rondo, CtxMatch, and S-Match. We will give an overview of these systems in section 2.4 after we describe the classification of ontology mapping techniques.
Figure 2.3 represents two trees with their respective roots at the upper and lower layer and leaves shared between the trees in the middle layer. The layers shown in the figure are explained below:

- The upper layer: This layer represents the root and non-leaf nodes of the classification tree (Tree 1) that is based on the granularity of matching techniques and the interpretation of the input information.
- The middle layer: This layer represents classes of elementary (basic) matching techniques that form the leaves of the two trees.
- The lower layer: This layer represents the root and non-leaf nodes of the classification tree (Tree 2) that is based on the kind of input used by elementary matching techniques.

Here we attempt to summarize the meanings of the terms in the classification trees and we refer the reader to Euzenat and Shvaiko [27] for further information.
Figure 2.3 Classification of the ontology mapping techniques. It shows two trees rooted at the upper and lower layer sharing the leaves in the middle layer [27].
2.3.2 Tree based on matcher granularity/ input interpretation

The first tree whose non-leaf nodes form the upper layer of the classification tree is discussed below. The basic matching techniques (leaves) in this tree are classified under Element-level and structure-level techniques.

2.3.2.1 Element-level techniques

Element-level techniques are ontology mapping techniques that don’t consider the relationship between the entities and their instances with other entities or instances classified under this node. The entities and their instances to the finest level of granularity (atomic level) are considered in isolation to compute the mapping. Table 2.1 shows ontology fragments (parts of an ontology) of two different banks. A sample atomic level match in this case is “Address.ZIP = CustomerAddress.PostalCode”.

Basic techniques that come under this category can be further classified under Syntactic and External techniques.

- **Syntactic** are techniques that interpret the input with regard to the structure dictated by a well-defined algorithm. String-based, Language-based and Constraint-based are the basic matching techniques that come under this category. More details and examples of these techniques are discussed in sections 2.3.4.1, 2.3.4.2, and 2.3.4.3, respectively.

- **External** techniques exploit auxiliary (external) resources (human or thesaurus containing relationships) of a domain and common knowledge in order to interpret the input. Linguistic resources (section 2.3.4.4) and Upper level, domain
specific ontologies (section 2.3.4.9) are the basic matching techniques that fall under this category.

2.3.2.2 Structure-level techniques

These are matchers or matching algorithms that take into account the relationship between the instances and their entities. They consider how the instances and their entities appear together in a structure. The level of matching required is decided on a case by case basis. Some cases might require the complete structure to be matched (as in row 1 of Table 2.1). Other cases might require partial mappings (shown in row 2 of Table 2.1).

<table>
<thead>
<tr>
<th>S1 Ontology fragment of Bank1</th>
<th>S2 Ontology fragment of Bank2</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Address</td>
<td>CustomerAddress</td>
<td>Full structural match of Address and CustomerAddress</td>
</tr>
<tr>
<td>Street</td>
<td>Street</td>
<td></td>
</tr>
<tr>
<td>City</td>
<td>City</td>
<td></td>
</tr>
<tr>
<td>State</td>
<td>USState</td>
<td></td>
</tr>
<tr>
<td>ZIP</td>
<td>PostalCode</td>
<td></td>
</tr>
<tr>
<td>AccountOwner</td>
<td>Customer</td>
<td>Partial structural match of AccountOwner (from a finance database) and Customer (sales database)</td>
</tr>
<tr>
<td>Name</td>
<td>Cname</td>
<td></td>
</tr>
<tr>
<td>Address</td>
<td>CAddress</td>
<td></td>
</tr>
<tr>
<td>Birthdate</td>
<td>CPhone</td>
<td></td>
</tr>
<tr>
<td>TaxExempt</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 2.1 Full vs. partial structural match (example). The table shows ontologies fragments of two different banks. “Address” and “AccountOwner” are classes in bank1’s ontology. “CustomerAddress” and “Customer” are classes in bank2’s ontology. The other fields are object properties of the respective classes.
Basic techniques that come under this category can be further classified as Syntactic, External and Semantic.

- **Syntactic**: Data Analysis and Statistics, Graph-based, and Taxonomy based are the basic matching techniques that fall under this category. These techniques are discussed in detail in section 2.3.4.10, 2.3.4.6 and 2.3.4.7.

- **External**: Repository of Structures is the basic technique that is classified under this category and is discussed in detail in section 2.3.4.8.

- **Semantic**: These are techniques that interpret the input using some formal semantics (say, model-theoretic semantic) to justify their results. Exact algorithms are complete and guarantee the discovery of all the possible alignments, whereas approximate algorithms tend to be incomplete. Model-based is the basic technique classified under this category and more information on this technique can be found in section 2.3.4.11.

### 2.3.3 Tree based on kind of input

The second tree whose non-leaf nodes form the lower layer of the classification tree is discussed below.

- The first level is categorized depending on the kind of data the algorithms work on: terminological (strings); structural (structure); extensional (data instances); and semantic (models).

- The second level decomposes these categories further if necessary: terminological can be string-based (considering the terms as sequences of characters) or based on the interpretation of these terms as linguistic objects (linguistic). The structural
methods category is split into two types of methods: those which consider the internal structure of entities, e.g., attributes and their types (internal), and those which consider the relation of entities with other entities (relational).

2.3.4 Basic Matching Techniques

The basic matching techniques that form the leaves of the two trees are discussed in this section.

2.3.4.1 String-based techniques

These techniques try to find mappings based on the string similarity between the input ontologies. The matchers use names and descriptions of the ontologies being considered. Some examples of such techniques include prefix-matching, suffix-matching, edit distance, and n-gram distance techniques.

Prefix and Suffix matching

A prefix-matching technique will match the words “net” and “network” but might also end up matching “hot” and “hotel.” Similarly, a suffix matching technique will match “phone” and “telephone” but might also end up matching “word” and “sword.” But prefix and suffix matching similarity can be useful as a test for strings denoting a more general concept than another (in many languages, adding clauses to a term would restrict its range). For instance, “reviewed article” is more specific than “article”. It can also be used for comparing strings and similar abbreviations, example, “ord” and “order”.

COMA, SF, S-Match, and OLA are techniques that implement prefix-based and suffix-based mapping.
**Edit Distance**

The operations that are usually considered for String edit distance include insertion of a character, replacement of a character by another and deletion of a character. Each operation is assigned a cost and the distance between two strings is the sum of the cost of each operation on the less costly set of operations. The Levenshtein distance is the edit distance with all costs equal to 1\[27].

For example, consider the ontologies shown in the Figure 2.4. The one on the left corresponds to an ontology of an e-commerce website selling cultural goods and the other represents that of a book library. Figure 2.5 shows Levenshtein distance values computed between most of the classes normalized to the maximum length of the strings being compared. A value of zero indicates that the strings are same and a value of one indicates that the strings are totally different and require changes in all positions.

![Figure 2.4](image)

**Figure 2.4**: This figure shows fragments of two ontologies: taxonomy of classes from an e-commerce website selling cultural goods on the left and a book library on the right.
The closest names are Pocket and Novel, and Pocket and Poetry. These names are relatively far from each other (0.67). So, in this case no correspondence can be found from such measures alone. However, the same measure on properties will find the correspondence between author and author, for instance.

<table>
<thead>
<tr>
<th></th>
<th>Science</th>
<th>Children</th>
<th>Book</th>
<th>Person</th>
<th>DVD</th>
<th>Textbook</th>
<th>Product</th>
<th>Pocket</th>
<th>Popular</th>
<th>CD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Politics</td>
<td>0.75</td>
<td>1.00</td>
<td>0.88</td>
<td>0.88</td>
<td>1.00</td>
<td>1.00</td>
<td>0.75</td>
<td>0.75</td>
<td>0.75</td>
<td>1.00</td>
</tr>
<tr>
<td>Thing</td>
<td>0.71</td>
<td>0.75</td>
<td>1.00</td>
<td>1.00</td>
<td>0.88</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>Autobiography</td>
<td>0.92</td>
<td>0.85</td>
<td>0.85</td>
<td>0.92</td>
<td>1.00</td>
<td>0.85</td>
<td>0.92</td>
<td>0.92</td>
<td>0.85</td>
<td>1.00</td>
</tr>
<tr>
<td>Novel</td>
<td>0.86</td>
<td>0.88</td>
<td>0.80</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>0.86</td>
<td>0.67</td>
<td>0.71</td>
<td>1.00</td>
</tr>
<tr>
<td>Biography</td>
<td>1.00</td>
<td>0.89</td>
<td>0.78</td>
<td>0.89</td>
<td>1.00</td>
<td>1.00</td>
<td>0.89</td>
<td>0.89</td>
<td>0.89</td>
<td>1.00</td>
</tr>
<tr>
<td>Essay</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>0.83</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>0.86</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>Volume</td>
<td>0.86</td>
<td>0.75</td>
<td>0.83</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>0.71</td>
<td>0.83</td>
<td>0.71</td>
<td>1.00</td>
</tr>
<tr>
<td>LiteraryCritic</td>
<td>0.93</td>
<td>0.93</td>
<td>1.00</td>
<td>0.86</td>
<td>1.00</td>
<td>0.93</td>
<td>0.86</td>
<td>0.93</td>
<td>0.86</td>
<td>0.93</td>
</tr>
<tr>
<td>Poetry</td>
<td>0.86</td>
<td>0.88</td>
<td>0.83</td>
<td>1.00</td>
<td>0.88</td>
<td>0.71</td>
<td>0.67</td>
<td>0.71</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>Literature</td>
<td>0.80</td>
<td>0.90</td>
<td>1.00</td>
<td>0.80</td>
<td>1.00</td>
<td>0.90</td>
<td>0.80</td>
<td>0.80</td>
<td>0.80</td>
<td>1.00</td>
</tr>
<tr>
<td>Human</td>
<td>0.86</td>
<td>0.88</td>
<td>1.00</td>
<td>0.83</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>0.71</td>
<td>1.00</td>
<td>1.00</td>
</tr>
</tbody>
</table>

Figure 2.5: Normalized Levenshtein distance values computed for the ontologies shown in the figure 2.4.

S-Match, OLA, and Anchor-Prompt are edit distance-based techniques.

N gram

This algorithm returns the number of n-grams (sequence of n characters) between the two input strings. For instance, trigrams for the string article are: art, rti, tic, icl, cle.

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

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

The normalized version of this function is as follows:

\[
\sigma(s, t) = \frac{|ngram(s, n) \cap ngram(t, n)|}{\min(|s|, |t|) - n + 1}
\]
Thus, for example, the trigram similarity between “article” and “aricle” would be $2/4 = 0.5$, while between “article” and “paper” would be 0.

COMA and S-Match are n-gram–based matchers.

### 2.3.4.2 Language-based techniques

These techniques are based on Natural Language Processing techniques that take into account the morphological properties of the input terms. The names from the input are considered as words in some natural language, for example, English. These techniques in conjunction with string-based are employed to improve the results. The following techniques are applied to the entity names by algorithms that are classified under this scheme:

- **Tokenization:** Given input entities, their names are split into tokens and parsed by recognizing punctuation, cases, blanks, underscores, etc. Example: The classes “Publisher of book” and “Literary critics” from the ontologies shown in the Figure 2.4 can be tokenized as `<publisher,of,book>` and `<literary,critics>` respectively.

- **Lemmatization:** The basic forms of the identified tokens are determined by morphological analysis. Example: critics -> critic.

- **Elimination:** Those tokens that are identified as articles, prepositions, conjunctions, etc., are eliminated. For example, the preposition “of” is removed from `<publisher,of,book>` to become `<publisher,book>`.

Once the above are done, either a string based or a distance based matcher can be invoked to identify further similarities.
COMA, Cupid, S-Match, and OLA are based on tokenization and lemmatization. Cupid and S-Match are based on elimination.

2.3.4.3 Constraint-based techniques

Entities are defined by their datatypes, cardinality of attributes, and keys. Constraint-based algorithms consider the internal constraints of such entity definitions. A simple and trivial example could be a mapping based on the equivalence of datatypes, etc. Assume we try to use a constraint-based matcher to determine mappings between the following ontologies:

- **Employee Ontology:**
  - It has data properties EmployeeNo (a literal with type xsd:int), EmployeeName (a literal with type xsd:String), Salary (a literal with type xsd:int), and date_of_birth (a literal with type xsd:date).

- **Personnel Ontology:**
  - Assume the ontology has data properties Pno (a literal with type xsd:int), Pname (a literal with type xsd:string), Dept (a literal with type xsd:String), and Born (a literal with type xsd:date).

If the matcher employed is not used in conjunction with any other matcher, it would say that the data properties EmployeeNo (from the Employee ontology) and Dept (from the Personnel ontology) are the same, because both of them have same datatype, xsd:int. Thus, constraint-based matchers used in isolation might produce invalid results and hence could use the help of another matcher, say a string-based matcher in our case.

OLA and COMA are matchers that can be categorized as constraint-based.
2.3.4.4 Linguistic resources

Matchers that use lexical resources such as lexicons or thesauri are classified under this category. The matching is based on the linguistic relations (e.g., synonyms, hyponyms) between entities.

- Sense based:
  - Thesauri and Lexicons: Matchers use thesauri such as WordNet [14] to recognize meanings of the entity names. WordNet is an electronic lexical database where various meanings of words are put together into sets of synonyms for English and other languages. Ontology mappings can be determined based on the bindings provided by WordNet senses. For example, in Figure 2.6, with morphological preprocessing of labels, “Camera”(in A1) can be determined to be a hypernym for “Digital_Cameras”(in A2). Thus, the mapping should be able to decide that “Digital Cameras” in A2 should be subsumed by “Photo_and_Cameras” in A1.
  - Lexicons: These are matchers that exploit the structural properties of lexicons. Hierarchy-based mappers may use WordNet hierarchies to determine the similarity by the distance between two entities as the number of arcs traversed in the hierarchy [15].

- Dictionaries and directories: Dictionaries store domain-specific terms with relationships such as synonym, hypernym, etc. A matcher will be able to determine that in Figure 2.6 entities “NKN” in A1 and “Nikon” in A2 are synonyms by doing a dictionary look that contains an entry that says “NKN” and
“Nikon” are synonyms. Hence it would deduce that “NKN” and “Nikon” are equivalent classes.

![Ontology Diagram]

**Figure 2.6**: This figure shows two ontologies A1 and A2 in the “Electronics” domain. It shows the classes and subclasses of both the ontologies in a tree format.

Artemis, S-Match, OLA, Cupid, and COMA are matchers that fall under this category.

### 2.3.4.5 Alignment reuse

This category is based on the concept that in order to determine mappings between two ontologies, O1 and O2, if there exists an external source containing mappings of ontologies O1 and O2 (individually) with another ontology O’, then the mapping with the ontology O’ can be reused to obtain the mapping between the ontologies O1 and O2. This approach makes sense only in the case where ontologies O1 and O2 belong to the same domain of interest.

Here is an alternate form of alignment reuse: Suppose we try to map very large ontologies. We break down the ontologies into parts, find out mapping between the
“fragments,” and reuse that information collectively to make it easier to obtain the mapping between the ontologies [16].

As an example, assume that we have mappings between the ontologies (individually) in Figure 2.4 with an ontology fragment from a retail store as shown in Figure 2.7. The mappings between the store ontology and the ontologies of the website and library from Figure 2.4 are shown in the tables 2.2 and 2.3 respectively. “Relationship” denotes the mapping and can take values representing equivalence(=), less general(≤) and more general(≥).

![Diagram of ontology classes](image)

**Figure 2.7**: Classes from retail store ontology (fragment).
Table 2.2: Store and E-commerce website ontology (from figure 2.4)-Class relationships.

<table>
<thead>
<tr>
<th>Store Ontology</th>
<th>E-commerce website ontology</th>
<th>Relationship</th>
</tr>
</thead>
<tbody>
<tr>
<td>Item</td>
<td>Product</td>
<td>=</td>
</tr>
<tr>
<td>Video</td>
<td>DVD</td>
<td>≥</td>
</tr>
<tr>
<td>Audio</td>
<td>CD</td>
<td>≥</td>
</tr>
<tr>
<td>Book</td>
<td>Book</td>
<td>=</td>
</tr>
<tr>
<td>Fiction</td>
<td>Pocket</td>
<td>=</td>
</tr>
</tbody>
</table>

Table 2.3: Store and Book library ontology (from figure 2.4)-Class Relationships.

Since “Book” (from e-commerce website ontology) corresponds to “Book” (from store ontology), and “Book” (from store ontology) corresponds to “Volume” (from library ontology), we can deduce that “Book” (from e-commerce website ontology) and “Volume” (from library ontology) are equivalent. Similarly, we can deduce that “Novel” and “Pocket” are equivalent since they both are equivalent to “Fiction”.

COMA, COMA++, and OLA are matchers that are classified under this category.

2.3.4.6 Graph-based techniques

These algorithms consider the ontologies being mapped as labeled graphs. The similarity comparison between any pair of nodes (from the two ontologies) is based on the analysis of their positions within their respective graphs. The intuition behind this is that, if two nodes from two ontologies are similar, their neighbors must also be similar.
An ontology matching problem is considered an optimization problem and is resolved with the help of graph-matching algorithms. Similarity flooding [17] is a kind of graph-matching algorithm that is based on fixed-point computation. Classes are related to each other through properties. If the properties are similar then the classes they relate must also be similar.

For example, in one of the possible ontology representations of schemas of Figure 2.4, if the “Book” class is related to the “Human” class by the “author” relation in one ontology, and if the “Volume” class is related to the “Writer” class by the “author” relation in the other ontology, then knowing that classes “Book” and “Volumes” are similar, and that relations author and author are similar, we can infer that the classes “Human” and “Writer” may be similar too.

Similarly, in Figure 2.6, if class “Photo_and_Cameras” (in A1) is related to class “NKN” by relation hasBrand, and if class “Digital_Cameras” (A2) is related to class “Nikon” by relation hasMarque in the other ontology, then the inference that “NKN” and “Nikon” may be similar can be made based on the knowledge that classes “Photo_and_Cameras” and “Digital_Cameras” are similar, and also relations hasBrand and hasMarque are similar.

Cupid, COMA, SF, S-Match, and OLA are matchers that fall under this category.

2.3.4.7 Taxonomy-based techniques

Taxonomy-based techniques are a special subset of graph algorithms that consider only the specialization relation (is a). The “is-a” links connect terms that are already similar, and hence the interpretation is that their neighbors may be also similar.
• **Bounded path matching:** Techniques that consider such hierarchical relations try to match paths with links between classes by comparing their position along the paths to identify similar terms. In our example, in Figure 2.6, assume that class “Digital Cameras” (in A2) is subsumed by the class “Photo and Cameras” (in A1). A matcher would identify that “FJFLM” in A1 and “FujiFilm” in A2 as an appropriate match.

Similarly, in Figure 2.4, if “Book” corresponds to “Volume” and “Popular” corresponds to “Autobiography”, then the elements along the paths (“Science” on one side and “Biography” and “Essay” on the other side) must be carefully considered for deciding the correspondence (example: Essay is more general than Science). This technique is primarily guided by two anchors of paths and uses alternative techniques for choosing the best match.

Anchor-Prompt, NOM, and QOM are matchers that fall under this category.

**2.3.4.8 Repository of structures**

The term “structure” here denotes ontologies and their fragments. Similarity measures between every pair of input structures are stored in a repository. Coefficients in the range of [0, 1] denoting the similarity between structures are pre-computed and stored. Input structures that are being matched are compared against the similarity of the structures in the repository. The intuition here is to identify structures that are sufficiently similar to be worth matching in more detail. The process of finding similarity with structures in the repository is a cheaper operation than finding the actual mapping between the structures without similarity.
For example, assume that the repository has similarity measures (Table 2.4) between ontologies in figures 2.7 and 2.8.

![Diagram](image)

**Figure 2.8:** University Library ontology (fragment).

<table>
<thead>
<tr>
<th>Store Ontology</th>
<th>University Library ontology</th>
<th>Similarity Measure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Item</td>
<td>Material</td>
<td>0</td>
</tr>
<tr>
<td>Book</td>
<td>Book</td>
<td>0.78</td>
</tr>
<tr>
<td>Audio</td>
<td>Thesis</td>
<td>0</td>
</tr>
<tr>
<td>Fiction</td>
<td>Novel</td>
<td>0.66</td>
</tr>
</tbody>
</table>

**Table 2.4:** Store and University library pre-computed similarity measures (stored in the repository).

The ontologies from the Figure 2.4 can be compared for similarity with those in the repository to decide whether the matching has to proceed for each pair of input structures. The relationship between the e-commerce website ontology (Figure 2.4) and the store ontology is shown in the Table 2.2. The relationship between book library (Figure 2.4) and university library ontology is shown in Table 2.5.
<table>
<thead>
<tr>
<th>Book library ontology</th>
<th>University Library ontology</th>
<th>Relationship</th>
</tr>
</thead>
<tbody>
<tr>
<td>Volume</td>
<td>Material</td>
<td>≤</td>
</tr>
<tr>
<td>Novel</td>
<td>Novel</td>
<td>=</td>
</tr>
<tr>
<td>Volume</td>
<td>Book</td>
<td>=</td>
</tr>
</tbody>
</table>

Table 2.5: Similarity measures – Book library (Figure 2.4) and University library (Figure 2.8).

Table 2.5 indicates that the class “Volume” is less general to “Material”. Table 2.2 indicates that the class “Item” is equivalent to class “Product”. If we try to match the classes “Volume” (book library) with “Product” (e-commerce website), since the similarity measure between “Item” and “Material” as indicated by Table 2.4 is zero, the matching need not proceed as the structures are not similar. Whereas the matcher can proceed matching the classes “Volume” (book library) and “Book” (e-commerce website) since they are related to classes “Book” (university library) and “Book” (store), respectively, and the similarity measure between “Book” (university library) and “Book” (store) is 0.78 as shown in Table 2.4.

2.3.4.9 Upper-level and domain-specific formal ontologies

These are matching techniques that make use of upper-level or domain-specific ontologies as external sources of knowledge.

- Upper-level ontologies: Systems like Suggested Upper Merged Ontology (SUMO), Descriptive Ontology for Linguistic and Cognitive Engineering (DOLCE), and upper Cyc ontology are some examples. These are semantic matching techniques as these are logic-based systems. This involves transforming
the source ontologies into lightweight ontologies expressed with respect to an upper level ontology and then applying reasoners to detect relations and inconsistencies between the source ontologies.

- Domain-specific formal ontologies focus on a particular domain and contain terms relevant only to that domain and not others. By using specialized resources, e.g., the Formal Model of Anatomy (FMA) [60] in medicine, one can be sure that the concepts in the contextualized resources can be matched accurately to their corresponding concepts in the ontology. However, by using more general resources, there is more probability that an alignment already exists and can be exploited right away.

  For example, let use try to match the anatomy part of two known ontologies CRISP [58] and MeSH [59], with FMA as the background knowledge. The matchers are employed to determine mappings between the concepts of FMA and CRISP or MeSH. For example, assume that in FMA ontology a part of relation exists between the concept of brain (Brain_{FMA}) and head (Head_{FMA}). The concept of brain from CRISP (Brain_{CRISP}) could be anchored to Brain_{FMA}. Similarly the concept of head from MeSH (Head_{MeSH}) could be mapped to Head_{FMA}. Now, with the assumption that Brain_{FMA} is a part of Head_{FMA}, matchers can deduce that Brain_{CRISP} is a part of Head_{MeSH}.

2.3.4.10 Data analysis and statistics techniques

These are techniques that make use of the representative sample of the population in order to detect similarities. This helps in grouping together items or computing
distances between them. Distance-based classification, formal concepts analysis, and correspondence analysis are some data analysis techniques. “Frequency distributions” is categorized as a statistical analysis technique. For more details on this classification, see [27].

**Formal Concept Analysis**

Formal concept analysis is based on the notion that, more constrained the properties (classes) are, the fewer the objects (individuals) that satisfy the constraints. From the data set, a set of individuals and classes can be organized as a concept lattice by computing the closure of the *individuals X classes*. Each concept can be identified by its properties (the intent) and covers the individual satisfying these properties (the extent). This operation starts with the complete lattice of the power set of extent (respectively, intent) and keeps only the nodes which are closed under the connection, i.e., starting with a set of properties, it determines the corresponding set of individuals, which itself provides a corresponding set of properties; if this set is the initial one, then it is closed and is preserved, otherwise, the node is discarded. The result is a concept lattice, like the one computed in Figure 2.9(b) from the table shown in Figure 2.9(a).

![Concept Lattice](image)

**Figure 2.9**: This figure shows the table of individual and classes that are used to build the
concept lattice from the ontologies shown in the Figure 2.4.

The technique is independent of the origin of the entities, i.e. the classes and instances might not belong to the same ontologies. The correspondences/mappings extracted from the concept lattice are shown in the Table 2.6.

<table>
<thead>
<tr>
<th>Class from E-commerce website ontology</th>
<th>Class from Book library ontology</th>
<th>Relation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Science</td>
<td>Essay</td>
<td>=</td>
</tr>
<tr>
<td>Science</td>
<td>Biography</td>
<td>≥</td>
</tr>
<tr>
<td>Essay</td>
<td>Popular</td>
<td></td>
</tr>
<tr>
<td>Science</td>
<td>Autobiography</td>
<td>≥</td>
</tr>
<tr>
<td>Popular</td>
<td>Biography</td>
<td>=</td>
</tr>
<tr>
<td>Popular</td>
<td>Autobiography</td>
<td>=</td>
</tr>
<tr>
<td>Literature</td>
<td>Pocket</td>
<td>≥</td>
</tr>
<tr>
<td>Novel</td>
<td>Pocket</td>
<td>=</td>
</tr>
</tbody>
</table>

Table 2.6: This table shows the mappings extracted from the concept lattice shown in figure 2.9(b).

The result shown is not accurate. However, it is possible to weight these results by first eliminating the redundant correspondences and by providing a confidence according to the size of the extent covered by the correspondence.

2.3.4.11 Model-based techniques

The intuition behind these techniques is that, if two entities are the same, their semantic interpretation must also be same.

These algorithms use the semantic interpretation of the input, Example: model-theoretic semantics.

The following are some example of model-based techniques.
Propositional satisfiability (SAT): SAT deciders are correct and complete decision procedures for propositional satisfiability. The approach here is to translate the mapping problem of mapping the graphs and queries into a propositional formula. Query mapping denotes a pair of nodes and their relationship. The propositional formula is checked for validity. SAT deciders can be used to check exhaustively for all possible mappings.

As an example, consider the ontologies A and B shown in the Figure 2.10. The ontologies represent documents describing the root level class in the hierarchical structure. A WordNet based matcher will be able to identify that the classes “Images” and “Pictures” are the same. Any matcher would guess that “Europe” from ontology A is the same as “Europe” from ontology B. These relations can be translated into propositional connectives as shown in the axioms below:

\[(\text{Images} \leftrightarrow \text{Pictures}) \land (\text{Europe} \leftrightarrow \text{Europe})\]

Note that ‘\(\leftrightarrow\)’ denotes equivalence.

Figure 2.10: Example ontologies representing image documents.
Let ‘C’ denote the set \( \text{Europe} \cap \text{Images} \) and let ‘C1’ denote the set \( \text{Pictures} \cap \text{Europe} \). Let us also suppose that we want to know if C is equivalent (\( \leftrightarrow \)) to C1. Thus, this matching task requires constructing the following formula:

\[
((\text{Images} \leftrightarrow \text{Pictures}) \land (\text{Europe} \leftrightarrow \text{Europe})) \rightarrow \\
((\text{Europe} \land \text{images}) \leftrightarrow (\text{Europe} \land \text{pictures}))
\]

Negation of this formula turns out to be unsatisfiable, and therefore, the equivalence relation holds. Thus besides pruning the incorrect correspondences, this technique also discovers the new ones between complex concepts.

- **Modal SAT**: Propositional SAT allows handling of unary predicates (e.g., classes, XML elements). The idea here is to augment propositional SAT with binary predicates (e.g., attributes), thereby enhancing propositional logics with modal logic operators. The matching problem is translated into a model logic formula and is checked for validity using sound and complete satisfiability search procedures [26].

- **Description Logic-based techniques**: A test of subsumption can be used to determine semantic class relationships. In fact, merging the two ontologies and determining subsumption relationships between each pair of concepts and roles will ensure ontology alignment.

For example, assume two minimal ontologies: Micro-company - a company with at most 5 employees; SME - a firm with at most 10 associates. The logic behind these ontologies can be specified by the formulae:

\[
\text{Micro-company} = \text{Company} \cap \leq 5 \text{ employee}
\]
\[ SME = \text{Firm} \cap \leq_{10} \text{associate} \]

From the above description logic syntax it can be deduced that Company is equivalent to Firm and associate is a sub class of employee.

\[ Company = \text{Firm} \]
\[ associate \subseteq \text{employee} \]

This implies that Micro-company is a subclass of SME.

CtxMatch and S-Match are matchers that fall under this category.

2.4 Ontology Mapping Systems

As mentioned earlier, the following are the systems that were considered by Euzenat and Shvaiko for their classification of ontology mapping techniques:

2.4.1 Anchor-PROMPT

Anchor-PROMPT is a hybrid matching algorithm that takes as input ontology graphs and a set of anchor-pairs of related terms. The anchor-pairs are either defined by the user or obtained from a matcher running string-based techniques or linguistic techniques. The algorithm walks through the input ontologies in the paths specified by the anchors and determines the terms appearing frequently in similar positions on similar paths. Based on the frequency determined and the user input, the algorithm determines the matches [20, 34].
2.4.2 Analysis of Requirements: Tool Environment for Multiple Information Systems (Artemis)

Artemis [35] was designed as a module of Mediator environment for Multiple Information Sources (MOMIS) mediator system for creating global views. MOMIS is a framework to perform information extraction and integration from both structured and semi-structured data sources. Artemis performs the following [34]:

- Affinity-based analysis: It makes use of a common thesaurus to calculate the name, structural, and global affinity coefficients. The common thesaurus is built with the help of ODB-Tools, WordNet, or manual input.
- Hierarchical clustering: This technique makes use of global affinity coefficients to categorize classes into groups based on different levels of affinity. For each cluster, it creates a set of global attributes -- global class. Logical correspondence between the attributes of a global class and source schema’s attributes is determined through a mapping table.

2.4.3 Cupid

Cupid [36] algorithm uses a hybrid approach to mapping ontologies, making use of linguistic and structural mapping techniques. It takes input as graphs and traverses them in a top-down and bottom-up manner to perform the matching process. It consists of three phases:

- Linguistic matching, which calculates linguistic similarity coefficients between the node labels based on the following: morphological normalization,
categorization, string-based techniques (common prefixes, suffixes tests), and a thesaurus look-up.

- Structural matching, which calculates the structural similarity coefficients weighted based on leaves by measuring the similarity between individual element contexts.
- Mapping generation computes weighted similarity coefficients. It generates final mappings by choosing element pairs with weighted similarity coefficients higher than a threshold.

2.4.4 COmBination of MAtching algorithms (COMA)

COMA is a composite schema and ontology matching tool developed at University of Leipzig. It provides a comprehensible and extensible library of matching algorithms. It is a framework for combining the obtained results, and provides a platform for the evaluation of the effectiveness of the various matchers.

It provides a simple, yet powerful GUI for combining the existing matchers and allows creation of workflows of matching steps, enabling a “divide-and-conquer”-like principle to solve complex match tasks in multiple stages [18, 33].

2.4.5 Naive Ontology Mapping (NOM)

NOM [20] inherits the idea of composite matching from COMA [18]. In addition, it adds a set of elementary matchers based on rules exploiting explicitly codified knowledge in ontologies, such as information about super- and subconcepts, super- and subproperties, etc. It also supports a set of instance-based techniques.
2.4.6 CTXMATCH

CTXMATCH uses logic to determine mappings between hierarchical classifications. It takes as input two hierarchical classifications and, between each pair of concepts in both hierarchical classifications, it produces a semantic relation expressed in terms of relation (⊇, ⊆, ≡,*, and ⊥). CTXMatch considers the structural and domain knowledge to determine the mappings [29, 30].

2.4.7 Quick Ontology Mapping (QOM)

QOM employs a dynamic programming approach to perform mapping between ontologies. It uses data structures to categorize and store the identified mappings as “promising” and “less promising” pairs. The dynamic programming approach makes it efficient and lowers the run-time complexity. It is capable of efficiently mapping large-size as well as light-weight ontologies [29, 32].

2.4.8 OWL Lite Aligner (OLA)]

OLA [28] has a set of distance-based algorithms. These algorithms convert definitions of distances (linearly aggregated modulo local matches of entities) into a set of equations based on all the input structures. The algorithms start with base distance measures computed from labels and concrete datatypes and iterates a fixed-point algorithm until no improvement is produced to determine the matching between the ontologies that minimizes the overall distance between them. The alignment generated satisfies some additional criterion (on the alignment obtained and the distance between aligned entities). As a system, OLA considers the alignment as a solution to a clearly stated optimization problem.
2.4.9 S-Match

S-match is a framework for semantic matching that contains semantic algorithms for ontology matching. It takes two ontologies in the form of tree-like structures as input and computes mapping between every pair of nodes. It implements a generic algorithm, a minimal version of the same algorithm, and a structure-preserving semantic matching algorithm. It was designed and developed as a platform for semantic matching, namely a highly modular system with the core of computing relations, where single components can be plugged, unplugged or suitably customized. The possible relations are equivalence (=), more general (⊇), less general (⊆), mismatch (⊥), and overlapping ([23]).

2.4.9.1 Generic semantic matching.

This algorithm generates mappings between every pair of nodes in the input trees. The nodes have labels that have an indented meaning in the real world. It operates in four steps [24]:

- **Step 1**: For all labels L in two trees, it computes the “concepts of labels” [23]. The system has a “Preprocessor” component which takes care of this step.
- **Step 2**: For all nodes N in two trees, it computes “concepts at nodes” [23]. The “Classifier” component of the system executes this step.
- **Step 3**: For all pairs of labels in two trees, compute relations among the concept of labels created by the preprocessor in step 1. “Element level matcher library” component executes this step.

44
• **Step 4**: For all pairs of nodes in two trees, compute relations among the concept of nodes created by the classifier in step 2. “Structure level matcher” component executes this step.

2.4.9.2 Minimal semantic matching.

The mappings generated by the generic matching algorithm might contain some redundant links. The minimal algorithm attempts to filter out the redundant mappings.

2.4.9.3 Structure-preserving semantic matching.

This algorithm is a variant of the generic S-match algorithm and tries to match the structure of the input trees. It maps structurally identical elements between the ontologies.

2.4.10 CROSI Mapping System (CMS)

CMS is an ontology alignment system for CROSI. CROSI stands for Capturing Representing and Operationalizing Semantic Interoperability. It is a structure-matching system that makes use of the rich semantics of the OWL constructs. Its modular architecture allows the system to consult external linguistic resources. It consists of feature generation, feature selection, multi-strategy similarity aggregator, and similarity evaluator. CMS implements a series of mapping techniques, regarded as independent components that make up CMS [25]. CMS assumes that class names are nouns or noun phrases and properties are verbs followed by nouns or adjectives.

• Name Matchers

The name matchers implemented in CMS range from pure syntactical approaches to more semantically enriched ones. They are categorized as follows:
- String distance: Some examples of string distance algorithms incorporated in CMS are Levenshtein distance, Monge-Elkan distance, and Jaro metric. These algorithms act upon tokenized representation of the input.

- Thesaurus: In CMS, thesaurus is incorporated as WordNet and predefined corpora.
  - WordNet employs a hierarchical structure to represent its synset entries. Assume that the names name\textsubscript{i} and name\textsubscript{j} have their corresponding WordNet entries as w\textsubscript{i} and w\textsubscript{j}. The similarity between w\textsubscript{i} and w\textsubscript{j} is approximated as $2 \times h / h_i + h_j$. Here w denotes the least common hypernym of w\textsubscript{i} and w\textsubscript{j}, r the root of the underlying WordNet hierarchy, and $h_i$, $h_j$, h the distances between w\textsubscript{i} and r, w\textsubscript{j} and r, w and r, respectively.

- Semantic Matchers

Semantic matchers in CMS are categorized as:

- Structure-aware: Structure-aware matchers traverse the class hierarchies and accumulate similarities among subclass and subproperty relationships. Assume that we have two classes, c and d, from two ontologies. Let c\textsubscript{i} and d\textsubscript{i} denote their direct parents in respective ontologies. Similarity between c and d is recursively defined as $\text{sim}(c, d) = \alpha \text{sim}_{\text{local}}(c, d) + \beta \text{sim}(c_i, d_i)$, where $\alpha$ and $\beta$ are arbitrary weights and $\text{sim}_{\text{local}}/2$ gives the local similarity with regard to c and d. The local similarity is computed using a combination of one or more of the name matchers.
Intension-aware: These matchers assume that a class is defined by a set consisting of its superclasses and properties. Thus, a class can be represented as a tuple \( c = \{ S, P \} \), where \( S \) is a set of its superclasses and \( P \) is a set of its properties. The semantic similarity between two classes \( c = \{ S_c, P_c \} \) and \( d = \{ S_d, P_d \} \) is defined as finding the similarity between \( S_c \) and \( S_d \) along with \( P_c \) and \( P_d \). The semantic similarity is defined as \( \text{sim}(c, d) = \alpha \text{sim}(S_c, S_d) + \beta \text{sim}_\text{property}(P_c, P_d) \). Here \( \alpha \) and \( \beta \) are arbitrary weights and \( \text{sim}_\text{property}/2 \) computes the property similarity.

2.5 Process-Oriented Service Integration Framework for Eclipse (psife)

Mampilly [4] proposed a flexible and extensible environment for the orchestration of disparate engineering tools using the Eclipse platform called psife. It is based on PFAST, an Eclipse-based Integrated Tool Workbench for Facilities Design. The architecture proposed by Mampilly is shown in Figure 2.11. The integrated solution provides the following:

- Facilities for designing a process model using a simple graphical user interface.
- Interpreting and enacting the process model.
- Integrating the services required by the process model and enabling the constituent services to communicate between each other and the process layer using shared data workspace and control mechanisms.
- Providing a uniform user interface for all services that are invoked by the process model enactment.

Pohl and Weidenhaupt [23] divide process-centered environment into three domains:
• Modeling domain: Activities for defining and maintaining process models by use of a formal language.
• Enactment domain: The actual orchestration of the processes, which denotes the mechanical interpretation of the process model by the process engine.
• Performance domain: Set of actual activities by agents (humans and machines).

Figure 2.11 Components of psife architecture.

The following are the components of the psife architecture.

2.5.1 The process manager

The process manager provides interface for the enactment of services. Process enactment is dictated by the interaction between the enactment and performance domains. The performance domain receives feedback from the enactment domain and executes
accordingly. The process designers define the services needed and are stored in the service catalogs.

2.5.2 The service manager

Tool developers refer to the service catalogs to determine the type of services needed and provide implementation of the services as plugins (tools). The mapping between the actual plugins and services is stored in the service registry. The service manager provides a public interface to the service registry. It provides services for invoking itself, starting and stopping services. The process engine and the plugins can invoke the services provided by the service manager.

2.5.3 The resource manager

The resource manager provides an interface for clients to add and access resources. The resources are associated with a particular schema and each time a new resource is added, if validation is enabled, the resource manager validates it against its corresponding schema. It also provides event notifications to the listeners who register to receive notifications when the resources are changed.

2.5.4 The UI manager

The UI manager is responsible for enabling services to add their menu to the main application window. It provides facilities for enabling and disabling the menus, thereby enabling the tools to control their visibility to the user.
2.6 Tool Integration - ModelCVS

ModelCVS [31] is a model-based tool integration system aiming to provide interoperability between model files generated from different modeling tools. It provides transparent transformation of models between different tools’ modeling languages expressed as MOF-based metamodels.

It also provides versioning capabilities, and thus goes a step further than the existing model transformation approaches. The solution approach we have employed for data extraction from different sources borrows concepts and techniques from ModelCVS.

Issues with model-based tool integration:

- Differences in model data formats. Example: relational databases vs. variants of XMI.
- Differences in modeling scope, e.g., general-purpose UML vs. specific workflow languages.
- Differences in syntax and semantics of languages.

ModelCVS uses a two-level approach separating syntactic and semantic issue.

The following are the two conceptual layers:

- *Architectural model integration patterns* form the first layer, which ensures openness, scalability, and evolvability of a tool integration solution.

- *Semantic technologies* in the form of ontologies for the integration of tool metamodels, as well as for semantic versioning capabilities for models.

Figure 2.11 shows the steps involved in generating the mapping between the input models.
Figure 2.12 Model-based Tool Integration Patterns.

ModelCVS makes use of a bridging language that provides bridging operators for representing the mapping between the model code. A set of integration patterns determine the requirements and working context of the bridging language. Below we will describe two integration patterns responsible for scalability and openness. For information on other patterns, see [21].

2.6.1 Metamodel translation

The modeling languages of the tools overlap conceptually and thus concepts that are equivalent semantically can be used for the translation.

2.6.2 Metamodel modularization

Modularization addresses the scalability issue. It addresses this issue by dividing the metamodel into fragments with each expressing certain aspect of the entire model. The models conforming to such a metamodel are decomposed into fragments in a similar manner.
2.6.3 Tool integration knowledge base

ModelCVS makes use of a repository of language-specific ontologies for each of the existing tools’ domains called Tool integration Knowledge Base. These ontologies contain specific instance data stemming from reference examples of case studies. This enables semi-automatic integration of new tool metamodels containing instances.

2.6.4 Ontology-based metamodel integration

ModelCVS employs an ontological approach to integration of modeling tools. Bridging specifications are created in a semi-automatic way by utilizing the tool integration knowledge base:

2.6.4.1 Metamodel lifting

Lifting is the process of creating ontology from source metadata. The elements in the metamodels are mapped to the concepts in the ontology, thereby performing a step of abstraction and semantical enrichment such that the ontology explicitly expresses the semantics of the modeling concepts whose syntax is defined by the metamodel. The generic ontologies in the tool integration knowledge base are used as a guidance for the ontology creation process and the idiosyncrasies of the specific tools’ languages are captured to create the tool ontologies. The lifting process establishes a common terminology between the models. Thus, the chosen generic ontology should cover the domains of the tool ontologies.
2.6.4.2 Ontology-level integration.

Mapping between ontologies can be automated to a great extent. Based on the relations defined in the generic ontology, the relations between the concepts of specific tools can be deduced, e.g., equivalence, subsumption, or substitutability.

2.6.4.3 Derivation of bridging.

With the mapping between the tool ontologies in place, bridging between the metamodels can be achieved depending on the integration pattern in use. In the case of translation, the bridging operation might create a new model element for each source element. In the case of modularization, it might denote that the two model elements should be merged.

2.6.4.4 Derivation of transformation.

With the bridging between the metamodels in place, the actual mappings between the source models can be derived as executable transformations.

2.7 Opinion Mining Tool – Cognitively designed User Interface

Ragavan [5] in her thesis addresses the need for an open source sense-respond cyberinfrastructure framework that an organization could apply to extract and analyze the “voice of the customer”, from blog comments, helping decision makers develop strategic, tactical and operational planning initiatives. It allows organizations to achieve business process improvement and generate inputs for modeling functional and non-functional requirements for Information Technology (IT) and other initiatives. The central hypothesis is that knowledge can be obtained by mining opinion and sentiment from
comments on blogs and that helps generate business intelligence for an organization. As a step in this direction, an opinion mining tool is designed that integrates information extraction and document search and analysis techniques to provide a concise representation of comments to the user. There has been a lot of research on clustering, sentiment analysis and summarization of text. Combining these three data analysis techniques along with a suitable visualization is an effective technique to explore customer feedback. However, providing easy access to these algorithms for a user from any background is a challenge. The use of a cognitive engineering methodology allows mapping of the mental model of the user to features available in the user interface. Use of a cognitive engineering methodology to develop an interface that provides combined access to these algorithms allows different types of users with varied expertise easy access to these comments.

The basic features of the tool are as follows:

- **Data Extraction**: To pull comment data from multiple web sources.
- **Comment Browser**: To read, search and select a subset of desired comments.
- **Data Analysis**: To analyze the comments and visualize them. The techniques used for comment analysis include a combination of sentiment analysis, clustering and summarization.

### 2.7.1 Initial interface design

The snapshot of the initial version of developed tool shown in Figure 2.13, allows the user browsing, searching, and analysis (cluster, sentiment, summary) of comment data. It tries to provide the user with all the essential functions for comment analysis,
however suffers from issues related to understandability – even an expert user, may not have an understanding of what cluster or sentiment analysis can do for him in terms of achieving his goal.

Figure 2.13: Snapshot of the initial user interface of the Opinion Mining Tool.

Some example scenarios where the understandability of the system becomes difficult for the user and how they can be overcome are listed below:

- The user may not be able to understand what to select by using the comment “browse” feature. The feature requires the user to select the directory containing the comment data extracted to documents that have to be in a particular format with appended metadata. A suitable descriptive label indicating the type of data
that the “browse” function allows would be a good way to intimate the user about the feature’s requirements. An error message should alert the user if he selects an invalid directory.

- The user might not realize that he is allowed to type the URL of the comment file in the “browse” field. From previous experience with browsing files, he might always tend to use the button feature to select files. An alert box with an appropriate message describing the feature would be a good way to intimate the user about this.

- The “search” box with the search criteria can be confusing to the end user, as it is not be clear what it is intended for. The user is required to select the check box corresponding to each search field, which he might not be aware of. A more descriptive label would be a simple solution to this problem.

- The order of Cluster/ Sentiment Analysis button and Advanced Options may be confuse the user about what the Advanced Options pertain to. Also, there may be some confusion as to whether all the advanced options apply to Clustering, Sentiment Analysis or both. A clear separation between the Cluster and Sentiment Analysis by providing separate areas for their respective options would help improve understandability. A tradeoff here is that the common options will get repeated.
2.7.2 Redesign based on Cognitive Engineering

Accounting for the failure scenarios, the user interface was redesigned by keeping in mind the following principles:

- According to Gesalt Psychology, the mind looks for shared properties. Thus the user interface should be designed with the goal of conveying relationships between the GUI elements. Thus, in the “Comment Browser” section, functionalities such as viewing, filtering and searching comments are placed together. The data analysis functions are all grouped together in the “Comment Analysis” section.

- Proximity compatibility principle: When perceiving a display humans tend to visualize its appearance and interpret its motion by comparing it to their expectations or mental model. Different sources of information that are related to the same task are integrated by the mind. Thus, divided attention is needed for each individual source in order to emphasize its importance.

- Hick Hyman’s Law of Decision Complexity: The speed with which an option is selected is strongly influenced by the number of available alternatives. Thus, by reducing the number of options available the mind will quickly decide what to select.

- Memory principle: Some people might tend to forget how to use the interface, whereas others might have too much of knowledge. The visual details available should provide sufficient information to user in order to use the interface. Ideally, there needs to be a right balance between the two.
• Predictive aiding removes a resource demanding cognitive task and replaces it with a simpler perpetual one.

• Gulf of Execution and Evaluation: While using the interface, the user should be able to bridge the gap between his mental goals and the features/functionality of the interface. This implies cognitive processing on the user’s side and is referred to as the “Gulf of Execution”. The design of the interface should be user-centric and goal-based such that it bridges the gulf of execution. The “Gulf of Evaluation” is bridged by comparisons between goals and expectations (cognitive dissonance) and presenting it via a proper output interface.

• Fitts’ Law: The time taken to point at some object is determined by its size and distance from the current position. Thus, the size and the position of the buttons are designed based on this principle.

Applying these principles during interface redesign helps map user goals and expectations to the features that the interface provides. Thus, the resulting interface is based on a concrete model and has implicit understandability, making it intuitive and easy to use for the user.

The screens designed after applying the cognitive engineering principles and methodology are shown in the Figure 2.14.
Figure 2.14: Cognitively Engineered User interface. This figure shows the screens that constitute the user interface redesigned using Cognitive Engineering principles.
Figure 2.14(a) is the initial splash screen that gets loaded when the user launches the tool. The options clearly indicate what the user can achieve using the tool. The extract comment screen, seen in Figure 2.14(b), allows the user to pull out comments from certain website domains. The domain list is preloaded in the tool and the user is allowed to select a domain, specify a URL or add multiple URLs from that domain and extract comments.

Figure 2.14(c), displays the main comment browsing and analysis screen that is improved from the previous version by using cognitive evaluation techniques. Technical terms such as cluster and sentiment analysis are replaced with more intuitive features that are more aligned with user goals. The user can now choose to analyze either All, Positive or Negative comments or type the question he wants answered from the comments in the query box. It is annotated with the principles that were employed for the position and placement of each UI component.

Figure 2.14(d) shows the same screen as that in Figure 2.14(c) except that now it shows the with Cluster Analysis details. Figure 2.14(e) shows the results of comment analysis from the clustering and sentiment analysis algorithms represented in the form a TreeMap.

2.9 Existing Ontological Tools

Existing ontological tools and algorithms may be categorized as follows:
2.9.1 Ontology Building tools

The first step in the process of knowledge-based data extraction is building ontology from source files. This step requires the user to be aware of the implicit ontologies underlying the source files from which data is being extracted. Tools that convert data of different types to RDF triples are available in the market. A few examples are XLWrap(Excel, CSV), Anzo(Excel), java2rdf, jpeg2rdf, etc. The W3C has a comprehensive collection of such tools listed in [38]. The tools might require the user to provide a mapping between the source format to RDF in a language dictated by the converters.

2.9.2 Ontology Mapping Tools

The next step in the process of data extraction by using a knowledge-based approach is to find out mappings between the generated ontologies. Several ontology mapping algorithms have been proposed and implemented as tools and APIs. A few notable ones include Smatch, CROSI, COMA, COMA++, Cupid, QOM, NOM and Artemis. Section 2.3 detailed the ontology mapping categories in which these tools fall. Section 2.4 provides an overview about the above ontology mapping systems. Once the mapping between the ontologies is found a new ontology can be built out of the source ontologies or the source ontologies along with the mappings themselves can be candidates for further processing. The former is called a “complete merge” and the latter is referred as “bridge ontology” [40].
2.9.3 Reasoning Tools

The merged ontology (bridge ontology) can be reasoned by a reasoner or inference engine to extract information. The quality of interpretation depends upon the reasoner employed. Several reasoners and inference engines exist in the market. Pellet[46], Jena Reasoners[45], HermiT[54], Fact++[55], etc. are a few notable ones.

2.9.4 Output Transformation Tools

The reasoned data can be transformed into a format that the user desires. Currently, very few serialization tools exist. Please refer to [41] for a list of tools that are available at W3C.

2.9.5 Miscellaneous Tools

We intend to develop a data-extraction workbench for Eclipse with a uniform user interface which provides an easy access to all the tools. Applications that are capable of allowing the user to edit, reason and visualize ontologies are available in the market. This section discusses two such tools and also describes the capabilities that are missing in them for our solution approach.

2.9.5.1 Protégé

Protégé[53] is an opensource ontology editor developed and maintained by Stanford. It is a Java based desktop application intended for working with ontologies. It
provides an extensible architecture and allows tool developers to incorporate their work as plugins. It provides standard plugins for loading and saving OWL and RDF ontologies, visual ontology editing and reasoning.

**Missing capabilities**

Protégé does not provide capabilities for importing data from various formats (such as CSV, JSON, XLS), converting to ontologies, automatic ontology mapping and conversion to other formats from RDF. Although one can argue that tools that implement these services can be created as separate plugins, providing a uniform user interface becomes difficult.

**2.9.5.2 NeOn Toolkit**

NeOn Toolkit is an Eclipse based ontology engineering environment (however not an Eclipse plugin). It is built on top of Eclipse plugin mechanism and is provided as a standalone application. It has plugins for ontology browsing, editing, aligning, merging, reasoning, versioning, evaluation, etc. Please refer to [47] for a complete list of available plugins.

**Missing capabilities**

The NeOn toolkit has the most functionality with respect to working with ontologies but it also has a few limitations:

- It fails to provide mechanisms for importing data from different underlying formats and exporting data to different formats.
The “Alignment” plugin provided with the toolkit is based on AlignmentAPI[48][49]. Though AlignmentAPI provides features for incorporating new matching algorithms, there are other matching tools and algorithms that have been implemented previously but not based on the AlignmentAPI. The argument here is that the NeOn toolkit does not provide a uniform user interface for the user to select from various matching algorithms except the ones based on AlignmentAPI.

Similarly the “Reasoner” plugin provided with the toolkit contains only Pellet2 and Hermit3 implementations. It does not provide mechanisms to incorporate other reasoners on the fly without modifying the source code of the plugin.

The “SPARQL” plugin provided does not provide the user a choice of the reasoner. Ideally, users must be allowed to choose the reasoner, since different reasoners have different inferencing abilities and our requirements dictate that we provide mechanisms to exploit all of them.

Though such tools can be incorporated as new plugins, they will require separate user interfaces for obtaining input from the user.

2.10 Eclipse Plugin Architecture

The runtime environment for an Eclipse application is called Eclipse equinox, which is the reference implementation of the Open Service Gateway Interface (OSGi) framework. OSGi is a specification that has a component and service model at its core. It enables components and services to be dynamically activated, de-activated, updated and uninstalled. OSGi specification defines an OSGi bundle as a unit of modularization (a
cohesive, self-contained unit), which explicitly defines its dependencies to other modules and services.

Eclipse provides a plugin architecture that can be easily extended. Except for Platform Runtime (kernel), all of the Eclipse Platform's functionality is incorporated as plug-ins. An Eclipse plugin is an OSGi bundle and vice versa. Since everything in Eclipse architecture is a plugin, two kinds of relationships exist between the plugins: Dependency- when a plugin is dependent on another for its working and Extension- when a plugin extends the capabilities of the host plugin to which it connects to. The foundation of the plugin architecture of eclipse is the extension point mechanism. A plugin which would like to allow another plugin to extend its capabilities exposes an extension point. An extension point exposes a set of interfaces which act as a contract. The child plugin which extends the extension point is expected to conform to the contract by implementing the interfaces exposed by the host plugin. The host plugin will make use of the services offered by the child plugin by invoking methods defined in the interfaces.

An Eclipse installation includes a “plugins” folder where individual plugins are deployed in separate folders. A plugin is described in an XML manifest file called plugin.xml, residing in the plugin's folder. The manifest file tells the Eclipse runtime what it needs to know to activate the plugin. In addition, it also includes the extension points from other plugins extended by the current plugin and the extension points that are exposed.

Figure 2.14 provides an overview of the extension point mechanism behind Eclipse’s plugin architecture.
Figure 2.15: This figure provides an overview of the extension point mechanism behind the Eclipse plugin architecture [51].

The Base plugin in Figure 2.15 exposes an extension point and a schema definition for that extension point. The schema definition defines the configuration syntax for the Extender plugin in order to extend the extension point. It might define a set of interfaces that are required to be implemented by the extender plugin. At runtime, Base plugin instantiates the Extender plugin’s classes that implement the interfaces exposed by the base plugin.

Figure 2.16 illustrates the extension point mechanism behind the Eclipse plugin architecture. It shows the extension of the Eclipse workbench user interface by Eclipse Help system’s user interface. Eclipse workbench user interface, “org.eclipse.ui” (host plugin), has extension points called “editors”, “views” and “actionSets”. The menus provided by the workbench’s UI can be extended via the extension-point, “actionSets”. Eclipse help system’s user interface, “org.eclipse.help.ui” (extender plugin) extends
“actionSets” in order to make the help functions available to the user. The help UI plug-in uses the “actionSets” extension-point to extend the workbench UI plug-in by specific help-related menu items, among them, Help->Help Contents and Search->Help.

**Figure 2.16:** This figure shows the extension of the Eclipse workbench UI (host plugin) by the Eclipse help system’s UI (extender plugin) through an extension point “actionSets” [52]. The extender plugin adds menus to the “help” menu.
Figure 2.17: Host plugin’s plugin.xml exposing the “actionSets” extension(1) and the schema definition “actionSets.exsd” (2)[52].

The figure 2.17 shows the plugin.xml corresponding to the Host plugin. It shows the extension point, “actionSets” and its schema definition,”actionSets.exsd”, declared in XML element “extension-point”. The actual schema definition, “actionSets.exsd” can be found in [50].

```xml
<plugin
  id="org.eclipse.ui"
  name="Eclipse UI"
  version="2.1.0"
  provider-name="Eclipse.org"
  class="org.eclipse.ui.internal.UIPlugin">
  <extension;
    point="org.eclipse.ui.actionSets">
    <actionSet
      label="Help" visible="true"
      id="org.eclipse.help.internal.ui.HelppActionSet">
      <action
        label="&Help Contents" icon="icons/visible.gif"
        helpContextId="org.eclipse.help.ui.helpContentsMenu"
        tooltip="Open Help Contents"
        class="org.eclipse.help.ui.internal.HelpContentsAction"
        memberPath="help.helpEnd"
        id="org.eclipse.help.internal.ui.HelppAction">
      </action>
      <!-- Other actionSet elements -->
    </actionSet>
    <action
      label="&Help..." icon="icons/Search_menu.gif"
      helpContextId="org.eclipse.help.ui.helpSearchMenu"
      class="org.eclipse.help.ui.internal.OpenHelpSearchPageAction"
      memberPath="org.eclipse.search_menu/dialogGroup"
      id="org.eclipse.help.ui.OpenHelpSearchPage">
    </action>
  </extension>
</plugin>
```

Figure 2.18: Extender plugin’s plugin.xml[52] denoting the extension of “actionSets” extension point(1). It defines two actions denoted by (2) and (4). The classes (3) and (5)
correspond to the actions (2) and (4) and should implement the interfaces IWorkbenchWindowActionDelegate or IWorkbenchWindowPulldownDelegate as required by [50].

The figures 2.17 and 2.18 show the plugin.xmls of the host and extender plugins. The schema definition requires the extender plugin to implement the interfaces IWorkbenchWindowActionDelegate or IWorkbenchWindowPulldownDelegate[50]. The extender plugin has two action classes, HelpContentAction (Help->Help Contents) and OpenHelpSearchPageAction (Search->Help) that implement either of the two interfaces.

The Base plugin at runtime, reads the Extender plugin’s plugin.xml file and creates menus. The action classes will be invoked by the Base plugin when their corresponding menu is selected by the user. Thus, it can be said that the Extender plugin uses the Base plugin’s functionality for creating menus.
CHAPTER 3

In this chapter we propose an extensible architecture that we call Knowledge-based Data Extraction (KDE) Architecture for the extraction of data from various sources. The solution proposed makes use of a knowledge-based approach for converting data from different formats to ontologies, mapping the ontologies, reasoning, and converting the data to desired formats. The architecture has been designed with the intention of making use of existing tools and new ones that may emerge. We evaluate the architecture by means of an exemplar scenario and provide a scenario-based comparison of two ontology-mapping algorithms, S-Match and CROSI.

3.1 Requirements of KDE

KDE was designed as an Eclipse workbench by keeping in mind the following:

- Data from various underlying formats has to be extracted by providing the user with a uniform view of the data.

- There are tools in the market that perform conversion from source data format to RDF-triples, automatic mapping between the ontologies, reasoning the ontologies, and conversion from RDF triples to desired data formats. The architecture has to be extensible to support tools that exist and those that are yet to come.
There is no one “best” ontology-mapping algorithm that would find all the valid mappings between any two ontologies. The solution approach is to let the user to decide which one to employ.

The user should be presented with an interface that provides him easy access to the available capabilities (tools and algorithms) in the system.

The extracted data should be made available to the user in a format that he desires.

3.2 Architecture of Knowledge-Based Data Extraction System

In this section we will discuss the details of the proposed Knowledge-based Data Extraction (KDE) workbench architecture. It has been designed with the requirements stated in section 3.1. KDE itself is implemented as an eclipse plugin extending the “actionSets” extension point to create GUIs. Figure 3.1 shows the architecture of KDE. It shows the components of KDE and the extension points exposed.
The system itself is implemented as an Eclipse plugin with the components shown within the bright box. The circles are extension points exposed by the components, which can be extended by tools that are wrapped up as plugins.

Below we detail the architectural components shown in Figure 3.1.

3.2.1 Controller

The controller component manages the interactions with the user. It is responsible for providing the GUI and managing the interactions with other components in the system. It provides the user with a set of algorithms and tools for the data extraction process. It communicates to other components the plugins to be invoked based on the user input. The controller pings the components Transformation broker, Ontology-Mapping Engine, Reasoning Engine, and Output Transformation Engine for the plugins that extend their extension points. It assigns unique IDs for each of the extending plugins.
and stores them in the “Internal Plugin Registry.” This registry contains a unique ID for each plugin, the component ID, the plugin name, and the extension point extended. Component ID is assigned to each component during initialization.

3.2.2 Transformation broker

The transformation broker, as the name indicates, is responsible for the transformation of data from the input format to RDF triples (building ontology). For every type of data file, it exposes an extension point. For brevity, Figure 3.1 shows only three extension points: XML, CSV and XLS. The tools that convert data from files in various formats to RDF triples can be wrapped up as plugins. The tools might require the user to provide a mapping file for converting the data to the RDF triples as input. Each extension point exposes an interface that has a “Convert” method, taking the file name and mapping filename as input and returning the RDF representation of the data. The tools might actually expect a different form of the data, say, a file handle, and might not return the triples. Hence, the plugin developers have to employ an adapter pattern to talk to the base plugin. Once an ontology is built from every source file, it is stored in the “Ontology File Mapping Registry.” It is a table that has a unique identifier for each record, the source filename (URI), and the generated ontology. For example, assume that the user wants to extract data from an Excel sheet and a CSV file. XLWrap is a tool that is capable of lifting CSV and Excel files to ontologies, given a mapping file. XLWrap can be wrapped up in a plugin that extends the extension points “XLS” and “CSV.” Based on the filetype, the transformation broker calls the corresponding version of XLWrap to lift the files to ontologies. The ontologies generated are stored in the ontology file mapping registry. If the excel sheet is the first one to be converted, its corresponding record would
have the ID as “1,” filename as name of the file with full path, and the ontology generated as RDF triples. Similarly, the CSV file will have an ID “2” and other details corresponding to it are stored in its corresponding record.

3.2.3 Ontology-mapping registry

The ontology-mapping registry is initialized by the controller. It is a table for storing the mappings between the various input files. Each record contains the unique IDs of the two input files (from the ontology file mapping registry), a mapping algorithm id (controller-generated plugin ID from Internal Plugin Registry), and the mappings generated upon completion of the algorithm. Based on the user input, the controller initializes the ontology-mapping registry with mapping algorithms to use between the input ontologies. As mappings are generated by the ontology-mapping engine, they are stored in the mapping registry. Continuing the example scenario discussed in section 3.2.2, assume that the user selects to apply a “String-based” CROSI algorithm (section 2.4.10) to the input ontologies. The controller creates a record in the ontology-mapping registry with the IDs as 1 and 2, mapping algorithm id as the plugin ID corresponding to CROSI, and an empty mapping. The actual mapping will be populated by the ontology-mapping engine once it is generated.

3.2.4 Ontology-mapping engine

This component is responsible for generating mappings between the ontologies. It exposes an extension point for each kind of leaf in classification elaborated in Section 2.3. The interface contract exposes a mapping method that provides the extender plugin with two input ontologies and expects as output, alignment/mappings. Tools that fall under each category of the classification can extend the corresponding extension point. A
single tool might fall under multiple categories, i.e., it might provide different algorithms that can be classified under different categories. For example, the classification in Section 2.3 has “String-based” and “Linguistic” categories. Therefore, the ontology-mapping engine has extension points called “String-based Extension” and “Linguistic Extension,” as shown in Figure 3.1. CROSI Mapping System can be classified as both “String-based” and “Linguistic” because it provides matchers (implementation of matching algorithms) that are both string-based and WordNet-based. Hence, it can be packaged as a plugin extending both the extension points.

An adapter pattern has to be employed within the extending plugin to enable the tool to talk to the ontology-mapping engine. Once a mapping is generated between each pair of input ontologies, the alignment is stored in the ontology-mapping registry.

3.2.5 Reasoning engine

Once ontology-mapping is done between the ontologies, the controller calls the reasoning engine. It merges the input ontologies and mappings (by reading the ontology-mapping registry) into a bridge ontology. It exposes an extension point called “reasoning extension” into which various reasoners can plug. The interface specification is quite simple, containing a method “Reason,” which takes the ontology, the query to be run against the ontology as input, and returns the result set output as RDF triples. Based on the user’s choice, the reasoning engine loads the registered plugin and calls the Reason method.

3.2.6 Output transformation engine

This engine is responsible for converting the data from RDF triples to any format desired by the user. It exposes an extension point for each kind of output file. The tools
that convert RDF to other formats can be packaged as plugins by extending the extension point. The tools might require a mapping file for the conversion. Hence, the interface contract specifies a “convert” method with the RDF triples, the mapping file, and the output filename as input, and returns a Boolean value based on the success of the conversion. The plugin is expected to store the converted result in the output file specified.

3.3 Implementation of KDE

In this section we describe the tools used to develop KDE.

3.3.1 Jena framework

Jena is a Java-based open source framework for working with RDF graphs. It provides mechanisms to represent, read, serialize, and reason RDF graphs [42]. RDF triples are represented as models in Jena. We have used “Jena models” for representing ontologies throughout our implementation.

3.3.2 Transformation tools

Our implementation of the transformation broker has two extension points, CSV and XLS (Excel). XLWrap [44] is a converter tool that is capable of converting data from CSV and Excel sheets to RDF triples. It expects a mapping file (in TriG [43] format) that specifies the ontology to be built from the source file. Because it is capable of converting data from both CSV and Excel formats, it extends the extension points “CSV Extension” and “XLS Extension.” The transformation broker is responsible for deciding which
version of XLWrap to use based on the user’s input. Figure 3.2 shows the user interface corresponding to the “Build Ontology” step of the data extraction process.

![User interface for the selection of input files and mapping files.](image)

**Figure 3.2** User interface for the selection of the input files and the corresponding mapping files. This data will be used to build ontologies when the user presses next.

### 3.3.3 Ontology mapping tools

There is no one “best” matching algorithm that can be used to map all ontologies. As an example we have employed SMatch(described in section 2.4.9) and CROSI(from section 2.4.10)as plugins to produce mapping between the ontologies. The implementation of ontology-mapping engine exposes two extension points, “graph-based
extension” and “linguistic extension”. SMatch extends the graph-based extension point and CROSI the linguistic extension point.

- SMatch can be integrated with the AlignmentAPI which understands OWL and not Jena models. So the Adapter pattern employed within the SMatch plugin serializes the data into a RDF file, uses OWL API to read and convert it to the format understandable by AlignmentAPI. Once the mapping is generated by SMatch between the two input files, the output is written to a file as RDF Triples. The file is loaded into a Jena model and is returned to the ontology-mapping engine.

- CROSI expects an OWL file as input and hence the input Jena model has to be serialized into an RDF file which will be read by the OWL API to produce an OWL file. This adapter version of the input will be fed as input to CROSI. Once CROSI returns the mappings, the result is loaded into a Jena model and returned to the ontology-mapping engine.

The ontology-mapping engine now writes the mappings generated by the individual plugins to the ontology-mapping registry which will be read by other components in the system. Figure 3.3 show the user interface for this step of the data extraction process.
3.3.4 Reasoning and inference engines

The reasoning engine exposes a single extension point to which the plugins Jena OWL Reasoner and Pellet are connected. These reasoners extend the “reasoning extension” extension point by implementing the interface contract. Since both Jena OWL reasoner and Pellet are capable of working with Jena models there is no adaptation needed within those plugins. Figure 3.4 shows the user interface corresponding to this phase.

- Jena OWL reasoner is a rule-based implementation of OWL Lite which is a subset of OWL Full. We refer the reader to [45] for more details.
- Pellet provides a host of reasoning services. OWL DL reasoning, Incremental Reasoning, etc. We refer the reader to [46] for more information on Pellet.
3.3.5 Output transformation tools

The implementation of the output transformation engine is future work. Currently there are very few tools in the market to transform RDF triples to other formats. In the near future, we believe that there will be many such tools. The W3C resource [41] lists a small collection of the tools available today.
3.4 Evaluation

In this section we attempt to provide an evaluation of the architecture by considering an example scenario. We compare the output of the system with that produced manually by a person.

3.4.1 Example scenario

Company A manufactures goods and sells it to its clients. Company B is one of its clients. Company B being a larger organization buys Company A. Now, an Analyst in company B would like to find out the actual manufacturing cost of the products that were sold by company A and save it in a file in CSV format. Let us assume that the analyst has access to transaction files from both the companies but unfortunately they are in different formats, say Excel and CSV. Figures 3.5 and 3.6 show the actual data files.

<table>
<thead>
<tr>
<th>Transaction ID</th>
<th>Date Qualifier</th>
<th>Date</th>
<th>Time</th>
<th>Time_Format Qualifier</th>
<th>Manufacturing_Location</th>
<th>Manufacturing_Date</th>
<th>ItemQuantity</th>
<th>Manufacturing_Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>D0MMYYYY</td>
<td>27122012</td>
<td>130000</td>
<td>EST</td>
<td>H-HHMMSS</td>
<td>Chennai</td>
<td>12022001</td>
<td>5</td>
</tr>
<tr>
<td>20</td>
<td>MMDYYY</td>
<td>102212</td>
<td>102134</td>
<td>CST</td>
<td>MM-HHSS</td>
<td>Loc Angeles</td>
<td>11032011</td>
<td>6</td>
</tr>
</tbody>
</table>

Figure 3.5 A snapshot of Company A’s (Seller) data in Excel format.

OrderID,Date,Time,TimeCode,Cost,Quantity
10,27122012,130000,EST,23455,5
20,102212,102134,CST,65652,6

Figure 3.6 A snapshot of Company B’s (Buyer) data file in CSV format.
Figures 3.5 and 3.6 represent the data files in Excel and CSV formats from Company A (seller) and Company B (buyer), respectively. For simplicity, we will call the Excel data file as “File1” and the CSV data file as “File2”.

3.4.2 Manual process

The analyst processing the data would figure out that the fields “TransactionID,” “Date,” “Time,” “TimeCode,” and “ItemQuantity” from File1 and the fields “OrderID,” “Date”, “Time,” “TimeCode,” and “Quantity” from File2 correspond to each other. With this information, he could solve the problem in a number of ways. He could pick each record from the File1 manually, match the “OrderID” from File2, pick the Manufacturing cost and other needed fields, and write it to a CSV file.

![Figure 3.7 Ontology representation of Company A (Seller) data from File1.](image-url)
Alternatively, he could seek the help of an Application Developer, who would write a program to read the input files, walk through the records in both the files, match the columns that correspond to each other, and pick the needed columns and dump the output in a CSV file. There are problems associated with this approach. For each kind of data file, mapping and conversion from one format to another would be necessary. For example, if one of the files is in XML and the other is in CSV format, mapping the XML nodes to a column in the CSV file requires manual comparison and cannot be automated for every kind of input file. Alternatively, the developer could use converters to build ontologies from the source files and try to extract data from them. Figures 3.7 and 3.8 represent the ontologies corresponding to the files File1 and File2, respectively. The developer can use converters to convert data files into ontologies, use automatic ontology-mapping tools to figure out possible mapping between the ontologies, load the ontologies and the mappings into a Jena model as a bridge ontology, write SPARQL queries to extract data, and finally use converters to convert to the desired format. The converters, mapping tools, and reasoners understand and work on different input formats. Conversion to and from these formats will be a difficult manual task for the developer.
Figure 3.8 Ontology representation of Company B (Buyer) data from File2.

3.4.3 Using KDE workbench

By using KDE workbench, the developer can convert the file from Excel and CSV formats into ontologies. Once the ontologies are generated, the user can use multiple ontology-mapping tools to determine the best possible mapping between the ontologies and can choose from a set of available reasoners to extract the needed data. The conversion of data between the various tools is done by the workbench.

KDE workbench has XLWrap tool incorporated a plugin. XLWrap expects a mapping file in TriG format to convert source files to ontologies. Figures 3.9 and 3.10 show the mapping files corresponding to the files File1 and File2, respectively.
Figure: 3.9 Mapping file for XLWrap corresponding to File1 in TriG syntax.

```trig
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix prefix: <http://www.w3.org/2003/11/excel#> .
@prefix prefix:xls: <http://www.w3.org/2009/11/xls#> .
@prefix prefix:csv: <http://www.w3.org/2009/11/csv#> .
@prefix prefix:csvData: <http://www.w3.org/2009/11/csvData#> .
@prefix prefix:BuyerOntology: <http://www.w3.org/2009/11/buyerOntology#> .
@prefix prefix:Product: <http://www.w3.org/2009/11/product#> .
@prefix prefix:Transaction: <http://www.w3.org/2009/11/transaction#> .
@prefix prefix:PurchaseOrder: <http://www.w3.org/2009/11/purchaseOrder#> .
@prefix prefix:Data: <http://www.w3.org/2009/11/data#> .
@prefix prefix:Resource: <http://www.w3.org/2009/11/resource#> .
@prefix prefix:Cell: <http://www.w3.org/2009/11/cell#> .
@prefix prefix:Media: <http://www.w3.org/2009/11/media#> .
@prefix prefix:Excel: <http://www.w3.org/2009/11/excel#> .
@prefix prefix:csv: <http://www.w3.org/2009/11/csv#> .
@prefix prefix:xml: <http://www.w3.org/2009/11/xml#> .
@prefix prefix:html: <http://www.w3.org/2009/11/html#> .
@prefix prefix:svg: <http://www.w3.org/2009/11/svg#> .

<Excel File URI>

Figure: 3.10 Mapping file for XLWrap corresponding to File2 in TriG syntax.

```trig
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix prefix:prefix: <http://www.w3.org/2003/11/excel#> .
@prefix prefix:xls: <http://www.w3.org/2009/11/xls#> .
@prefix prefix:csv: <http://www.w3.org/2009/11/csv#> .
@prefix prefix:BuyerOntology: <http://www.w3.org/2009/11/buyerOntology#> .
@prefix prefix:Product: <http://www.w3.org/2009/11/product#> .
@prefix prefix:Transaction: <http://www.w3.org/2009/11/transaction#> .
@prefix prefix:PurchaseOrder: <http://www.w3.org/2009/11/purchaseOrder#> .
@prefix prefix:Data: <http://www.w3.org/2009/11/data#> .
@prefix prefix:Resource: <http://www.w3.org/2009/11/resource#> .
@prefix prefix:Cell: <http://www.w3.org/2009/11/cell#> .
@prefix prefix:Media: <http://www.w3.org/2009/11/media#> .
@prefix prefix:Excel: <http://www.w3.org/2009/11/excel#> .
@prefix prefix:csv: <http://www.w3.org/2009/11/csv#> .
@prefix prefix:xml: <http://www.w3.org/2009/11/xml#> .
@prefix prefix:html: <http://www.w3.org/2009/11/html#> .
@prefix prefix:svg: <http://www.w3.org/2009/11/svg#> .

<Excel File URI>

```
The ontologies that are generated for the files are shown in the Figures 3.7 and 3.8. The mappings generated by employing S-Match and CROSI are shown in the Tables 3.1 and 3.2, respectively. “S.NO” is a counter used to identify a mapping uniquely. “Relationship” denotes the relationship between the input ontologies File1 and File2. SMatch defines the relational operators =, <, and > to denote equivalent class/property, less general, or more general relationships. “Confidence Measure” denotes the degree of confidence about the established relationship.
<table>
<thead>
<tr>
<th>S.No</th>
<th>Class or Property from Ontology 1</th>
<th>Class or Property from Ontology 1</th>
<th>Relationship</th>
<th>Confidence Measure</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Order</td>
<td>PurchaseOrder</td>
<td>=</td>
<td>&quot;1.0&quot;</td>
</tr>
<tr>
<td>2</td>
<td>Order</td>
<td>OrderID</td>
<td>=</td>
<td>&quot;1.0&quot;</td>
</tr>
<tr>
<td>3</td>
<td>TimeCode</td>
<td>Time</td>
<td>&lt;</td>
<td>&quot;1.0&quot;</td>
</tr>
<tr>
<td>4</td>
<td>TimeCode</td>
<td>Quantity</td>
<td>&lt;</td>
<td>&quot;1.0&quot;</td>
</tr>
<tr>
<td>5</td>
<td><strong>TimeCode</strong></td>
<td><strong>TimeCode</strong></td>
<td>=</td>
<td>&quot;1.0&quot;</td>
</tr>
<tr>
<td>6</td>
<td>ManufacturingDate</td>
<td>Quantity</td>
<td>&lt;</td>
<td>&quot;1.0&quot;</td>
</tr>
<tr>
<td>7</td>
<td>ManufacturingDate</td>
<td>Time</td>
<td>&lt;</td>
<td>&quot;1.0&quot;</td>
</tr>
<tr>
<td>8</td>
<td>ManufacturingDate</td>
<td>Date</td>
<td>=</td>
<td>&quot;1.0&quot;</td>
</tr>
<tr>
<td>9</td>
<td>DateQualifier</td>
<td>Date</td>
<td>&lt;</td>
<td>&quot;1.0&quot;</td>
</tr>
<tr>
<td>10</td>
<td>DateQualifier</td>
<td>Quantity</td>
<td>&lt;</td>
<td>&quot;1.0&quot;</td>
</tr>
<tr>
<td>11</td>
<td>DateQualifier</td>
<td>Time</td>
<td>&lt;</td>
<td>&quot;1.0&quot;</td>
</tr>
<tr>
<td>12</td>
<td><strong>ItemQuantity</strong></td>
<td><strong>Quantity</strong></td>
<td>=</td>
<td>&quot;1.0&quot;</td>
</tr>
<tr>
<td>13</td>
<td>TransactionID</td>
<td>Transaction</td>
<td>=</td>
<td>&quot;1.0&quot;</td>
</tr>
<tr>
<td>14</td>
<td>Bill</td>
<td>Bill</td>
<td>=</td>
<td>&quot;1.0&quot;</td>
</tr>
<tr>
<td>15</td>
<td><strong>Time_Format_Qualifier</strong></td>
<td>Time</td>
<td>&lt;</td>
<td>&quot;1.0&quot;</td>
</tr>
<tr>
<td>16</td>
<td><strong>Time_Format_Qualifier</strong></td>
<td>Quantity</td>
<td>&lt;</td>
<td>&quot;1.0&quot;</td>
</tr>
<tr>
<td>17</td>
<td><strong>Time</strong></td>
<td><strong>TimeCode</strong></td>
<td>&gt;</td>
<td>&quot;1.0&quot;</td>
</tr>
<tr>
<td>18</td>
<td><strong>Time</strong></td>
<td>Quantity</td>
<td>&lt;</td>
<td>&quot;1.0&quot;</td>
</tr>
<tr>
<td>19</td>
<td><strong>Time</strong></td>
<td>Date</td>
<td>&gt;</td>
<td>&quot;1.0&quot;</td>
</tr>
<tr>
<td>20</td>
<td><strong>Time</strong></td>
<td><strong>Time</strong></td>
<td>=</td>
<td>&quot;1.0&quot;</td>
</tr>
<tr>
<td>21</td>
<td><strong>Date</strong></td>
<td>Time</td>
<td>&lt;</td>
<td>&quot;1.0&quot;</td>
</tr>
<tr>
<td>22</td>
<td><strong>Date</strong></td>
<td>Quantity</td>
<td>&lt;</td>
<td>&quot;1.0&quot;</td>
</tr>
<tr>
<td>23</td>
<td><strong>Date</strong></td>
<td><strong>Date</strong></td>
<td>=</td>
<td>&quot;1.0&quot;</td>
</tr>
<tr>
<td>24</td>
<td>Manufacturing_Cost</td>
<td>Cost</td>
<td>&lt;</td>
<td>&quot;1.0&quot;</td>
</tr>
</tbody>
</table>

**Table 3.1** Mappings and the confidence measures generated by the algorithm S-Match over the input ontologies shown in Figures 3.7 and 3.8. The relational operators =, <, and > denote equivalent class/property, less general, or more general relationships defined by S-Match.
<table>
<thead>
<tr>
<th>S No</th>
<th>Class or Property from Ontology 1</th>
<th>Class or Property from Ontology 2</th>
<th>Relationship</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Order</td>
<td>PurchaseOrder</td>
<td>equivalentClass</td>
</tr>
<tr>
<td>2</td>
<td>Date</td>
<td>Date</td>
<td>equivalentClass</td>
</tr>
<tr>
<td>3</td>
<td>Time</td>
<td>Time</td>
<td>equivalentClass</td>
</tr>
<tr>
<td>4</td>
<td>Order</td>
<td>PurchaseOrder</td>
<td>equivalentClass</td>
</tr>
<tr>
<td>5</td>
<td><a href="http://www.w3.org/2000/01/rdf-schema#Resource">http://www.w3.org/2000/01/rdf-schema#Resource</a></td>
<td><a href="http://www.w3.org/2000/01/rdf-schema#Resource">http://www.w3.org/2000/01/rdf-schema#Resource</a></td>
<td>equivalentClass</td>
</tr>
<tr>
<td>6</td>
<td>Date</td>
<td>Date</td>
<td>equivalentClass</td>
</tr>
<tr>
<td>7</td>
<td>Time</td>
<td>Time</td>
<td>equivalentClass</td>
</tr>
<tr>
<td>8</td>
<td>Bill</td>
<td>Bill</td>
<td>equivalentClass</td>
</tr>
<tr>
<td>9</td>
<td>Bill</td>
<td>Bill</td>
<td>equivalentClass</td>
</tr>
</tbody>
</table>

Table 3.2 Mappings and the relationships generated by the algorithm CROSI over the input ontologies shown in Figures 3.7 and 3.8.

With the mapping in place, the user can use the query shown in Figure 3.11 and use a reasoner to extract the data.

```sparql
PREFIX rdfs:<http://www.w3.org/2000/01/rdf-schema#>
PREFIX Seller:<http://Seller.org/vocab#>
PREFIX Buyer:<http://Buyer.com/vocab#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
  ?a owl:sameAs ?b .
  ?x a ?a .
  ?y a ?b .
  ?y Buyer:hasOrderId ?a0 .
  ?a0 Buyer:hasValue ?orderId .
  ?y Buyer:hasDate ?a1 .
  ?a1 Buyer:hasValue ?date .
  ?y Buyer:hasCost ?a5 .
  ?a5 Buyer:hasValue ?cost .
  ?x Seller:hasTransactionID ?a2 .
  ?a2 Seller:hasValue ?transId .
  ?x Seller:hasDateQualifier ?a3 .
  ?a3 Seller:hasValue ?datequalifier .
  ?a4 Seller:hasValue ?manucost
FILTER (xsd:byte(?orderId) = ?transId)
}
```

Figure 3.11 SPARQL query used to reason over the bridge ontology, which is composed on the ontologies for file File1 and File2, along with the mappings generated.
3.5 Scenario-Based Comparison of Ontology Mapping Algorithms

We attempt to compare the automatic ontology-mapping algorithms S-Match and CROSI based on the scenario described in Section 3.4. The mappings produced by these algorithms for the ontologies in Figures 3.7 and 3.8 in are shown in Tables 3.1 and 3.2, respectively.

3.5.1 Gold standard

The actual mapping between the ontologies that a human would deduce from the input ontologies is shown in Table 3.3.

<table>
<thead>
<tr>
<th>S No</th>
<th>Class or Property from Ontology 1</th>
<th>Class or Property from Ontology 2</th>
<th>Relationship</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Bill</td>
<td>Bill</td>
<td>equivalentClass</td>
</tr>
<tr>
<td>2</td>
<td>Order</td>
<td>PurchaseOrder</td>
<td>equivalentClass</td>
</tr>
<tr>
<td>3</td>
<td>Date</td>
<td>Date</td>
<td>equivalentClass</td>
</tr>
<tr>
<td>4</td>
<td>TransactionID</td>
<td>OrderID</td>
<td>equivalentClass</td>
</tr>
<tr>
<td>5</td>
<td>Time</td>
<td>Time</td>
<td>equivalentClass</td>
</tr>
<tr>
<td>6</td>
<td>ItemQuantity</td>
<td>Quantity</td>
<td>equivalentClass</td>
</tr>
<tr>
<td>7</td>
<td>TimeCode</td>
<td>TimeCode</td>
<td>equivalentClass</td>
</tr>
</tbody>
</table>

Table 3.3 Ground truth mapping.

3.5.2 Mappings by S-Match and CROSI

Do et al. [57] propose several ways to evaluate alignments qualitatively. One possibility consists of proposing a reference alignment (R) that is the ground truth (a gold standard). The results from the evaluated alignment algorithm (A) can be compared to that reference
alignment. In what follows, alignments \( A \) and \( R \) are considered to be sets of pairs. The following are some metrics used to evaluate alignments [56]:

- **Hamming distance**: Given a reference alignment \( R \), the Hamming distance between \( R \) and some alignment \( A \) is given by
  \[
  H(A, R) = 1 - \frac{|A \cap R|}{|A \cup R|}
  \]
  A Hamming distance value of 0 indicates a better performing algorithm, whereas a value of 1 denotes that the algorithm fails to identify any valid mappings.

- **Precision**: Given a reference alignment \( R \), the precision of some alignment \( A \) is given by
  \[
  P(A, R) = \frac{|A \cap R|}{|A|}
  \]
  The precision value is computed as a ratio of the number of valid mappings identified by the algorithm to the total number of mappings (valid+invalid) it produces. A precision value closer to 1 indicates a better performing algorithm.

- **Recall**: Given a reference alignment \( R \), the recall of some alignment \( A \) is given by
  \[
  \text{Rec}(A, R) = \frac{|A \cap R|}{|R|}
  \]
  The recall value is computed as the ratio of the number of valid mappings identified by the algorithm to the total number of actual mappings that exist between the ontologies (gold standard). The closer the recall value to 1, the better the algorithm runs.

- **Overall**: Given a reference alignment \( R \), the overall of some alignment \( A \) is given by
\[ O(A, R) = 1 - \frac{|A \cup R| - |A \cap R|}{|R|} \]

An overall score of 1 indicates a better performing algorithm. A negative value indicates that the algorithm produces a lot of redundant mappings, whereas a positive value indicates that the algorithm identifies mappings lesser than the actual ones.

Table 3.5 lists a comparison of the quality metric values for the algorithms S-Match and CROSI. S-Match generates a total of 24 mappings, out of which only 6 are found to be valid. The mappings generated by S-Match are shown in the Table 3.1. CROSI generates a total of 9 mappings, out of which 4 are valid and they are shown in bold font in Table 3.2. CROSI also generates a number of duplicate mappings. As can be seen from Table 3.2, the pairs Order-PurchaseOrder, Date-Date, Time-Time, and Bill-Bill are duplicated. The actual number of mappings between the two input ontologies File1 and File2 is 7(|A|). Table 3.4 shows the valid and total number of mappings that SMatch and CROSI generate for the input ontologies.

<table>
<thead>
<tr>
<th>Count</th>
<th>SMatch (Tree based)</th>
<th>CROSI (WordNet)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Valid mappings</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>(</td>
<td>A \cap R</td>
<td>)</td>
</tr>
<tr>
<td>Total</td>
<td>24</td>
<td>9</td>
</tr>
<tr>
<td>(</td>
<td>A</td>
<td>)</td>
</tr>
</tbody>
</table>

**Table 3.4** Number of valid and total mappings generated by SMatch and CROSI.
Table 3.5 Comparison of the quality metric values for the algorithms CROSI and S-Match.

Table 3.5 shows that CROSI, a linguistic matching algorithm, has better hamming distance, precision, and recall values, whereas S-Match has a good recall value for the example under consideration.
CHAPTER 4

4.1 Conclusion

Ontologies offer a solution to the problem of extracting data from heterogeneous formats. Tools that work with ontologies have differences in the format of the input and output. An integrated workbench of tools provides the user with a rich collection of capabilities. The user can focus on extracting the data rather than worrying about the inconsistencies between the tools.

This thesis addresses the problem of data extraction by employing a knowledge-based approach. Initially, we created an exhaustive collection of terms used in the ontology world. Later, we proposed an extensible architecture that provided a means for making use of the existing ontological tools for the process of data extraction. Using an exemplar scenario, we were able compare how a user would extract data manually as opposed to using the system. Finally, we did a scenario-based comparison of two ontology mapping algorithms, S-Match and CROSI.
4.2 Future Work

4.2.1 Deploying KDE into psife

The Knowledge-based Data Extraction (KDE) architecture proposed can be viewed as an architecture to integrate ontological tools. The architecture proposed by Mampilly, psife [4], is capable of dictating orchestration of the engineering tools. By redesigning KDE’s components (Transformation Broker, Ontology-Mapping Engine, Reasoning Engine, and Output-Transformation Engine) as service plugins, we can deploy KDE into psife. The problem here is that psife requires all the service plugins to extend the extension point “services,” i.e., in our case, the plugins XLWrap, S-Match, CROSI, etc. should extend the services extension point. The interaction between these plugins and their orchestration is achieved by the “Process Manager” component of psife. But, KDE requires these plugins to extend the extension points provided by their respective components. In simple words, psife requires plugins that may wish to extend the service plugins should extend the services extension point. Visualizing this as a tree-like structure, the tree will be a two-level tree with psife at the root and all other plugins as the leaves in the second level. If we would like to extend the plugins into another level, psife does not dictate how the service registry will be populated.

In future work, we can extend the architecture of psife to include a “Service Manager” on every service plugin. The service manager on every plugin collects the list of plugins that extend it in an XML file and communicates it to the service manager on the base plugin. In the tree-like structure, the service manager at a node (plugin) will know the list of plugins available in the subtree rooted at that node. This enables the
service manager in psife to have access to all the plugins as well as the plugins extending those plugins. As and when new plugins are added to any plugin in the tree, the service manager of the plugin updates the XML file and sends it to the plugin it extends. Thus, the XML propagates up the hierarchy and reaches the service manager of psife. The service registry will contain the XML file containing the plugin trees, service ID, input and output schema IDs. The process manager can read the service registry and become aware of the plugins currently available in the system.

4.2.2 Designing GUI based on cognitive engineering principles and data visualization.

Most research in cognitive engineering addresses the need to incorporate the mental models of the user into the design and architecture. Understanding the psychology of the user and placing GUI components in the right place would help enhance the usability of the system. As a future enhancement, the current user interface can be redesigned based on cognitive engineering principles. This would consist of an iterative process of redesigning the interface and evaluating it based on the user’s performance.

4.2.3 Creating the “Best” ontology mapping

Certain algorithms (tools) used for ontology mapping provide a confidence measure of the mappings generated. This information can be used to pick the best possible mappings automatically. The ontology-manage engine can perform this by collecting the mapping results of the algorithms, stripping out the mappings with low confidence measure, and eliminating the duplicate mappings. This would assist the user in deciding the correct mappings quickly.
4.2.4 KDE for service-based platform.

Currently, KDE handles tools that can be packaged as plugins and deployed in a single system. We would like to make KDE talk to tools that cannot be packaged as plugins. Such tools could be service-based running on the same or a different system. KDE could be modified to include such services as plugins by substituting a stub in the place of the actual plugin, which can handle the communication between KDE and the services. The communication could be through a socket connection, a web service, or via the concept of shared memory.

4.2.5 Graphical ontology editing plugins

The workbench does not include capabilities for editing the ontologies graphically. We would like to enable the user to edit the ontologies using graphical editors. The abilities of the editors vary and hence we would like to provide the user with a set of graphical editors to choose from. This would mean using the editors as plugins in the system. The ontology mapping engine can be extended to include this functionality by exposing an extension point into which the editors available can be plugged.
BIBLIOGRAPHY


[38] http://www.w3.org/wiki/ConverterToRdf


[41] http://www.w3.org/wiki/ConverterFromRdf


[43] http://www.w3.org/2010/01/Turtle/Trig


[51] https://trac.rtsys.informatik.uni-kiel.de/trac/11ss-eclipse/wiki/Tutorials/Plugins#no1


