Supporting Data Integration Tasks with Semi-Automatic Ontology Construction

Student: Rizkallah Touma
Advisors: Oscar Romero, Petar Jovanovic, and Jovan Varga

Universitat Politecnica de Catalunya,
Barcelona, Spain
{ratouma, oromero, petar, jvarga}@essi.upc.edu

Abstract. Data Integration aims to facilitate the exploitation of heterogeneous data by providing the user with a unified view of data residing in different sources. Currently, ontologies are commonly used to represent this unified view in terms of a global target schema due to their flexibility and expressiveness. However, most approaches still assume a predefined target schema and focus on generating the mappings between this schema and the sources.
In this paper, we propose a solution that supports data integration tasks by employing semi-automatic ontology construction to create a target schema on the fly. To that end, we revisit existing ontology extraction, matching and merging techniques and integrate them into a single end-to-end system. Moreover, we extend the used techniques with the automatic generation of mappings between the extracted ontologies and the underlying data sources. Finally, to demonstrate the usefulness of our solution, we integrate it with an independent data integration system.

Keywords: Data integration, Ontology extraction, Ontology matching

1 Introduction

Large amounts of data are currently made available on the web but the vast majority of them is still published in formats that are not directly exploitable by analytical tools. Moreover, their extremely heterogeneous nature makes any effort to reconcile the data a cumbersome task.

These problems have been tackled by Data Integration (DI) [28] and Data Exchange (DE) [15]. Their main goal is to provide the user with a unified view of different data sources through which these sources can be queried and explored.

DI and DE work with three main artifacts: the data sources, the target schema and the mappings between the schema and the sources. While DE focuses on materializing data in the target schema using these mappings, in DI no actual exchange of data is needed and the data conforming to the target schema is usually virtual [9].

Ontologies have been increasingly used to represent the target schema in DI and DE due to their flexibility and expressiveness [28]. However, all major approaches so far assume a predefined target schema and focus on (semi-)automatically creating the mappings [14] [11]. This assumption is clearly limiting and is not applicable to all data integration tasks.

In this paper, we propose a solution based on semi-automatic ontology construction to support the creation of a target schema and mappings for DI and DE tasks. Ontology construction, as defined in this work, builds on top of three widely studied fields. First, **ontology extraction** [34] is used to extract ontology concepts from the sources, hence unifying the representation of the sources. Second, the heterogeneity of the sources is resolved by applying **ontology matching** techniques [6] to the extracted ontologies. Finally, **ontology merging** [19] is used to produce a single output ontology which represents the target schema of the underlying data sources.

These fields have been typically studied separately, but using them to support data integration tasks requires an end-to-end solution that integrates the three of them together. More importantly, ontology extraction is usually not concerned with keeping mappings between the extracted ontology and the data source. These mappings are essential for data integration as they are used to translate queries over the global schema into queries over source schemata in DI and to materialize the data into the target schema in DE [9]. Hence, the existing ontology extraction techniques must be extended to accommodate the generation of these mappings.

Contributions. The main contributions of the present work are the following:

- We present a system that supports data integration tasks by semi-automatically generating a target schema that captures heterogeneous data sources as well as the mappings between the target schema and the sources,
- We revisit existing ontology extraction, ontology matching and ontology merging techniques and integrate them into a single end-to-end system,
- We extend the used ontology extraction techniques with the generation of mappings between the extracted ontologies and the underlying sources,
- We demonstrate how our system can be used to support data integration tasks by integrating it with an independent data integration system.

1.1 Running Example

Throughout this paper, we use a running example extracted from the TPC-DI benchmark data set[1].The TPC-DI benchmark combines and transforms data extracted from a (fictitious) brokerage firm's On-Line Transaction Processing (OTLP) system along with other sources of data, and loads it into a data warehouse. We use the following subset of the TPC-DI data sources in the running example:

- **Relational database:** the OLTP database includes transactional information about securities market trading and the entities involved. Figure 1 depicts the entities used as part of this running example. The first column in each table is the primary key and columns with underlined names are foreign keys.

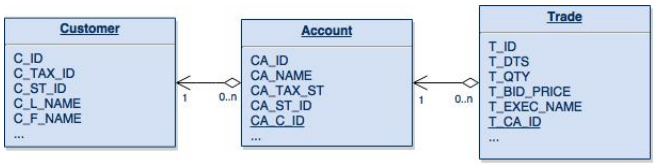


Fig. 1: Running Example - RDB Source

- **XML source:** the XML source represents data extracted from a Customer Management System (CMS) which handles new and updated customer and account information. The partial XML schema used in this running example is depicted in Figure 2.

The remainder of this paper is organized as follows: an overview of related literature is given in Section 2. Our proposed solution is formally defined and detailed in Section 3. A brief look at the implemented prototype is given in Section 4. The results of a practical case study are reported in Section 5. Finally, Section 6 summarizes our work and outlines possibilities for future work.

[1] TPC-DI Data Set: `http://www.tpc.org/tpcdi/default.asp`

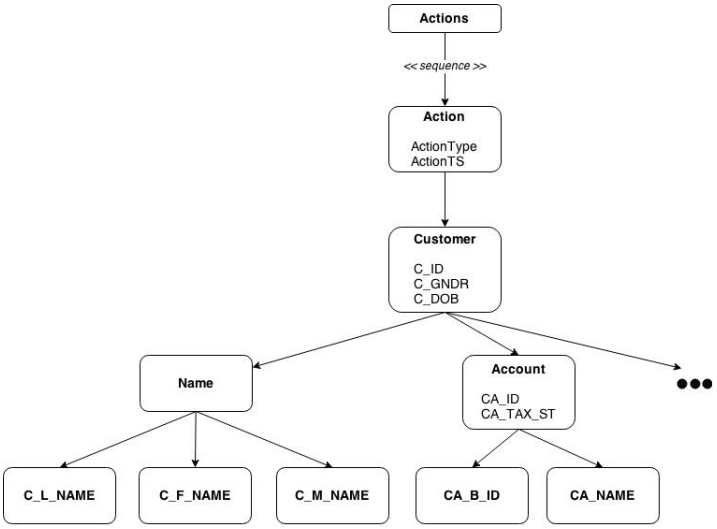


Fig. 2: Running Example - XML Source

2 Related Work

Handling data heterogeneity has been separately studied in two different settings, namely Data Integration (DI) and Data Exchange (DE). On the one side, Data Integration has been defined as a problem of providing the user with a unified view over the disparate data sources, in the form of a global (virtual) schema [28]. Users are then expected to pose their queries in terms of a global schema, while a data integration system automatically reformulates these queries in terms of source schemata, querying data sources individually and finally combining extracted data to answer the user queries.

On the other side, in the scenarios where accessing the data sources at runtime is either not possible (e.g. higher autonomy) or not desirable (e.g. possible congestion of operational processes), user queries should be answered from the previously materialized target data. Consequently, Data Exchange has been defined as a problem of materializing data at the target to best reflect the source data, and hence enable answering user queries solely over the target schema [15].

Both DI and DE are based on the concept of schema mappings, which in general can be seen as assertions that define the correspondences between the schema elements at the sources and at the target. Given that the present work builds on ontology extraction and ontology matching, the rest of the related work section is focused on reviewing approaches in these two fields.

2.1 Ontology Extraction

Ontology extraction can be defined as "the process of acquiring (constructing or integrating) an ontology (semi-) automatically" [34]. Most approaches to ontology extraction, sometimes also called ontology construction or learning, produce an OWL-language ontology[2] and are usually specific to the type of data sources used. In this section, we provide an overview of the techniques used for two of the most common data source types: relational databases and XML sources.

2.1.1 Relational Databases

The basic conversion rules from relational databases to OWL are presented in [32] which includes algorithms to convert tables, columns and constraints. Several published tools im-

[2] OWL: `http://www.w3.org/TR/owl-features/`

plement these rules such as RDBToOnto [4] and Relational.OWL [10]. The main difference between these tools is in the type of databases they support (RDBToOnto supports MySQL and Oracle and is an open source project that can be extended to include other database systems while Relational.OWL only supports MySQL and DB2).

A more complete approach is presented in [30], which classifies the database tables into "roles" based on their primary key and foreign key constraints. They describe the extracted semantic knowledge with a DLR-Lite description logic and keep a mapping between the resulting ontology and the original database in order to allow semantic data retrieval. It is worth mentioning that this is one of the few approaches that keep this type of mappings, which is aligned to the main ideas behind our work.

Other approaches have been based on database reverse engineering and use a set of conversion rules and heuristics in order to convert the database schema into an ontology [7]. The main challenges here are the accuracy of the used heuristics and dealing with database schema not following a disciplined design [30].

2.1.2 XML Sources

Another type of widely studied sources are semi-structured sources such as XML. The most simple technique is to create a direct mapping between XML tags and OWL elements as proposed in [2] but this technique is very limited in the formats of XML schema that it accepts due to the primitive nature of the mappings.

More sophisticated approaches are presented in [41] and [3] which extract ontology concepts from XML tags and create a hierarchy over the extracted concepts. The main difference is that [3] considers integration and synthesis between the extracted terms while [41] only considers syntactic similarity.

An implementation of a similar approach is available through the Ontmalizer[3] project. While most approaches focus on converting XML schema to OWL, Ontmalizer accepts both XML schema and XML documents and converts them to the corresponding OWL representations in a two phase approach; (1) it converts the XML schema to OWL concepts, and (2) using these conversion mappings, it converts valid XML documents according to this schema to OWL instances.

2.1.3 Challenges

Most ontology extraction approaches disregard the mappings between the data source and the resulting ontology due to the added complexity of storing and manipulating them. However, storing these mappings increases the system capability and usability in different fields. For instance, it facilitates the management of the ontology, allows semantic querying of the underlying data source [30] and is crucial for the usability of the system in data integration tasks. Hence, there is a need for an approach where mappings are viewed as first-class citizens and are stored and handled independently from the resulting ontology.

2.2 Ontology Matching

Ontology matching can be viewed as "defining mappings among schema or ontology elements that are semantically related" [6]. Usually, the elements being matched come from two different schema or ontologies.

Given the rapidly growing number of ontologies, ontology matching has been one of the most studied sub-fields of the Semantic Web. Perhaps due to that, it is known by many different names such as ontology matching, mapping, alignment and reconciliation. The difference between these concepts is defined in [19] as follows:

- **Ontology mapping** "deals with relating concepts from different ontologies and is concerned with the representation and storage of mappings between the concepts."

[3] Ontmalizer: `http://www.srdc.com.tr/projects/salus/blog/?p=189`

- **Ontology matching** "is the process of bringing ontologies into mutual agreement by the automatic discovery of mappings between related concepts. The ontologies themselves are unaffected by the alignment process."
- **Ontology merging** "deals with producing a completely new ontology that ideally captures all knowledge from the original ontologies."

Another closely related term is ontology enrichment which generates extensions, such as new concepts, new relations or corrections of the axioms of an existing ontology [17]. While this might seem different at first, it is important to note that the result of an ontology matching process can be, and in many cases is, used to "enrich" one of the ontologies with the concepts of the second one. In the remainder of this work, we exclusively use the term "ontology matching" .

2.2.1 Classification of Ontology Matching Techniques

Ontology matching techniques can be classified into one of the following main categories: terminological, linguistic, structural and semantic [37].

Terminological techniques are the most basic and the oldest. They use string matching techniques in order to determine the similarity between two ontology concepts. The most popular techniques used are edit distance [6,29] and Levenstein distance [24], while [22] proposes a custom-built string metric for ontology matching.

The second type of techniques are **linguistic techniques** which make use of an external knowledge base to determine linguistic similarities between two ontology concepts. The vast majority of the approaches in the studied literature use WordNet for that purpose [6,24,29].

Structural techniques take advantage of the hierarchical nature of ontologies to determine the similarity of two ontology concepts by comparing the neighborhood of the first concept (e.g. siblings) with that of the second [6,22,24].

Finally, the most complex type of techniques used takes advantage of **semantic models** in the matching process such as deduction and inference rules [24]. While these techniques usually yield good results, they have not been as common as other techniques due to their complexity.

Some of the other techniques that do not fall under any of these categories include machine learning [12], probabilistic models [40] and virtual documents [22]. Independently of which technique they use, some approaches apply other techniques in order to improve their performance and efficiently deal with large ontologies such as divide-and-conquer [22], Anchor-Flood [37], clustering [1] and parallelism [20].

It should be noted that the majority of the studied approaches are semi-automatic, meaning that they need user input in order to improve the final matching results, and some, such as [6,26,31], are implemented with a GUI. Appendix A includes a detailed overview of the studied ontology matching approaches.

2.2.2 Challenges

Despite the big amount of research done in ontology matching, many challenges still persist in the field and need further studying [37]. Given that most approaches are still semi-automatic, finding a straight-forward way to **involve the user** in the matching process poses a big challenge, especially when dealing with large ontologies that are not easily traceable or understandable by the user.

On the other hand, automatic matching techniques do not always give precise results and most approaches turn to **combining several matching techniques** in order to improve the matching precision. However, the methods used to combine the results of the techniques are usually primitive and need more research and improvement [37].

Finally, most approaches lack a strong **alignment management** system where alignments can be stored, manipulated and exploited later on. One of the few approaches that

explicitly address this issue is The Alignment API [8] which provides a framework for ontology matching and stores alignments in a custom-developed language.

Having a strong alignment management infrastructure makes it possible to represent complex alignments beyond the capabilities of automatic matchers. Moreover, it provides the alignments for independent usage by third-party applications and hence allows for more reusability of the system.

2.3 Representation of Correspondences

Correspondences are the output of the ontology extraction and matching processes. One type of correspondences relates the extracted ontology to the initial data source while another defines the relations between the concepts of the two ontologies being matched.

To avoid confusion, we will use two different terms for the two types of correspondences:

- **Mappings:** are the output of the ontology extraction process. They define the relation between an ontology concept and the corresponding element in the initial data source,
- **Alignments:** are the output of the ontology matching process. They define the relation between the concepts of the two extracted ontologies.

Due to the differences in the nature and usage of these correspondences, they have been researched separately and here we discuss the main approaches in handling each of them.

2.3.1 Mappings

While most ontology extraction approaches do not store mappings between the extracted ontology and the data source, some independent research has been done on possible formats to store these mappings.

One such format is the R2RML language developed by the W3C consortium [36] which has been extensively studied and developed over the past years. However, it was designed to map relational databases to RDF and it would not work properly for a system intended to handle different types of sources, such as the case in this work.

Another approach is presented in [25] which introduces a general XML-based format that allows to represent mappings from any type of data sources (e.g. relational database, XML documents, natural-language texts) to OWL[4].

2.3.2 Alignments

One of the most advanced techniques on the subject of representing alignments is the EDOAL language developed as part of the Alignment API project [8]. EDOAL (Expressive and Declarative Ontology Alignment Language)[5] is an XML-based language based on OWL that allows to represent complex alignments between the entities of two ontologies. It is specifically useful for cases when expressing equivalence or subsumption between terms is not sufficient.

While the expressiveness of EDOAL exceeds the capabilities of automatic matchers, the ability to represent complex alignments is considered an advantage since the user can manually create such alignments as a post-tuning step.

3 Proposed Solution

The proposed solution is depicted in Figure 3. It uses ontology construction, which encompasses ontology extraction, generation of mappings, ontology matching and ontology merging, to semi-automatically generate a target schema covering several data sources.

[4] Mappings DTD: `http://www.essi.upc.edu/~petar/mappings.html`
[5] EDOAL: `http://alignapi.gforge.inria.fr/edoal.html`

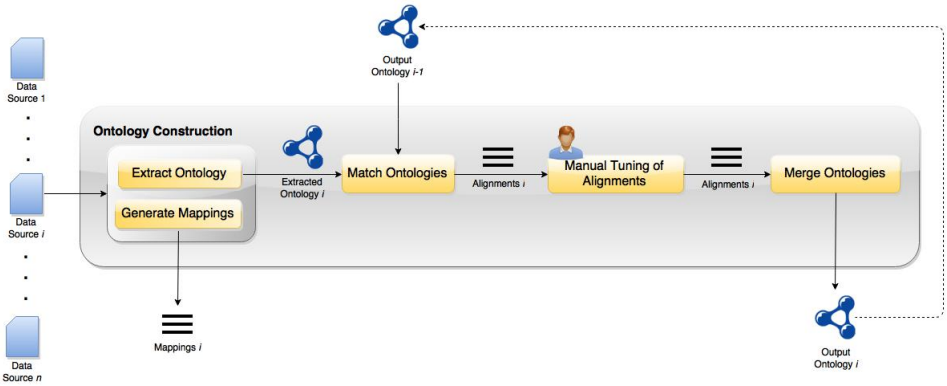


Fig. 3: Proposed Solution

Ontology construction is an iterative process that accepts as input one or more data sources ds. It then pipelines each data source through a series of processes:

- First, an ontology O_{temp} is extracted from ds and a set of *mappings* is generated.
- Second, O_{temp} is matched with the output ontology of the previous iteration O_{out} producing a set of *alignments*.
- These *alignments* are then displayed to the user for manual tuning.
- Finally, the two ontologies O_{temp} and O_{out} are merged using the set of modified alignments $alignments'$.

As a final output, the process produces a single merged ontology O_{out} and a set of mappings between O_{out} and each input ds. The main steps of the ontology construction process are highlighted in Algorithm 1.

Algorithm 1: Ontology Construction

Input : DS: Set of data sources
Output: O_{out}: Output ontology,
$Mappings$: Set of mappings

foreach $ds \in DS$ **do**
  $O_{temp} \leftarrow$ `ExtractOntology`(ds);
  $mappings \cup \leftarrow$ `GenerateMappings`(O_{temp}, ds);
  if *ds is first data source* **then**
    $O_{out} \leftarrow O_{temp}$;
  else
    $alignments \leftarrow$ `MatchOntologies`(O_{temp}, O_{out});
    $alignments' \leftarrow$ `ManualTuningOfAlignments`$(alignments)$;
    $O_{out} \leftarrow$ `MergeOntologies`$(O_{temp}, O_{out}, alignments')$;

3.1 Ontology

Before starting with the formal definition of each of the main processes included in ontology construction, we formalize the concept of ontology. Given the following sets

C : *a set of classes*
P : *a set of properties*
L : *a set of literals*

An ontology statement s can be defined as a triple

$$s = e_1 \times p \times e_2 \;\; where \;\; e_1 \in C,\; p \in P,\; e_2 \in (C \cup L)$$

An ontology O is considered a set of statements [40] and can be defined as

$$O \subset C \times P \times (C \cup L)$$

For simplicity in further definitions, we also define the set of all entities in an ontology E as the union of classes and properties

$$E = C \cup P$$

For the purpose of this work, we use OWL to represent the entities of O. Also, it should be noted that $\forall e \in E$, there's an IRI defining e.

3.2 Ontology Extraction

Ontology extraction is done by mapping the elements in the source to ontology elements. We can write that an extraction process is a function that generates an ontology O starting from a data source ds

$$f : ds \rightarrow O$$

Given the diverse nature of the sources to be extracted, an instance of f should be defined for each type of sources to be tackled. In this paper, we provide the formalization of such a function for relational databases and XML sources but the solution is extensible to include other types of sources. While all of the rules introduced in this section are deterministic, Appendix B includes a discussion on the use of heuristics in ontology extraction.

3.2.1 Relational Database

Assuming a database that has a set T of tables, for each $t \in T$ we define the following sets

$R_t :$ *the set of rows in* t
$C_t :$ *the set of all columns of* t
$PK_t :$ *the set of primary key columns of* t
$\mathbb{FK}_t :$ *the set of foreign keys of* t

We also define the value of a specific row of R_t on a specific column of C_t as

$$r(c) : \forall r \in R_t, \forall c \in C_t$$

Thus, the value of a specific row on C, a subset of C_t, is defined as the concatenation of $r(c)$ on all columns in C

$$r(C) = concat(r(c),\; \forall c \in C),\; \forall r \in R_t,\; \forall C \subseteq C_t$$

Each foreign key $FK_{t,t'} \in \mathbb{FK}_t$ is a subset of one or more columns of C_t. For each $FK_{t,t'}$, the following should hold

$$\forall FK_{t,t'} \in \mathbb{FK}_t,\; \forall r \in R_t :\; \exists t' \in T,\; \exists r' \in R_{t'} \; where \; r(FK_{t,t'}) = r'(PK_{t'})$$

Using the above definitions, we can define the extraction function for a relational database with the values shown in Table 1.

$$f_{rdb} : T \rightarrow O$$

	x	$f_{rdb}(x)$
Schema	$\forall t \in T$	$owl : class$
	$\forall c \in C_t$	$owl : DatatypeProperty\ [rdfs : Domain = f_{rdb}(t),\ rdfs : Range = type(c)]$
	$\forall FK_{t,t'} \in \mathbb{FK}_t$	$owl : ObjectProperty\ [rdfs : Domain = f_{rdb}(t),\ rdfs : Range = f_{rdb}(t'),\ owl : Cardinality = 1]$
Instances	$\forall r \in R_t$	$r(PK_t) \times rdf : type \times c \ \ where\ c = f_{rdb}(t)$
	$\forall r(c),\ \forall c \in C_t : r(c) \neq NULL$	$i \times p \times r(c) \ \ where\ i = f_{rdb}(r),\ p = f_{rdb}(c)$
	$r(FK_{t,t'}),\ \forall FK_{t,t'} \in \mathbb{FK}_t : r(FK_{t,t'}) \neq NULL$	$i \times p \times i' \ \ where\ i = f_{rdb}(r),\ p = f_{rdb}(FK_{t,t'}),\ i' = f_{rdb}(r')$

Table 1: RDB Extraction Function

Some elements map to a main axiom and one or more additional axioms, presented in square brackets, that are properties of the main axiom. For instance, for each property we set the domain and range of that property as shown in Table 1. These mappings are based on the extraction algorithms presented in [32].

Example. Applying the extraction rules presented in Table 1 to the database depicted in Figure 1 results in the following ontology:

```
<rdf:RDF
  xmlns:tpcrdb="http://www.tpc.org/tpc-di/rdb"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <owl:Class rdf:about="tpcrdb#Account"/>
  ...
  <owl:Class rdf:about="tpcrdb#Customer">
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:Cardinality rdf:datatype="xsd#int">1</owl:Cardinality>
        <owl:onProperty>
          <owl:ObjectProperty rdf:about="tpcrdb#CA_C_ID_FK"/>
        </owl:onProperty>
      </owl:Restriction>
    </rdfs:subClassOf>
  </owl:Class>
  ...
  <owl:DatatypeProperty rdf:about="tpcrdb#C_L_NAME">
    <rdfs:domain rdf:resource="tpcrdb#Customer"/>
    <rdfs:range rdf:resource="xsd#string"/>
  </owl:DatatypeProperty>
  ...
  <owl:ObjectProperty rdf:about="tpcrdb#CA_C_ID_FK">
    <rdfs:domain rdf:resource="tpcrdb#Account"/>
    <rdfs:range rdf:resource="tpcrdb#Customer"/>
  </owl:ObjectProperty>
  ...
</rdf:RDF>
```

Note how every table in the database is mapped to an OWL class while every column is mapped to a datatype property and every foreign key to an object property. The extraction process also results in the following RDF instance:

```
<account rdf:about="tpcrdb#account_2184">
  <ca_id rdf:datatype="xsd#long">2184</ca_id>
  <ca_st_id rdf:datatype="xsd#string">ACTV</ca_st_id>
  <account_customer_fk>
    <customer rdf:about="tpcrdb#customer_849">
      <c_id rdf:datatype="xsd#string">849</c_id>
      <c_ctry rdf:datatype="xsd#string">ESP</c_ctry>
      ...
    </customer>
  </account_customer_fk>
  ...
</account>
```

3.2.2 XML

Assuming that we have an XML Schema (XSD)[6], we can define the following sets for XSD

ST_{xsd} : *the set of simple types in* XSD
CT_{xsd} : *the set of complex types in* XSD

Then, each complex type $ct \in CT_{xsd}$ consists of

LE_{ct} : *the set of local elements defined in* ct
A_{ct} : *the set of attributes defined in* ct

And assuming we have an XML document which is valid to the schema XSD, we define

E_{xml} : *the set of elements in* XML
A_{xml} : *the set of attributes of all elements of* XML

Using the above definitions, we define the extraction function of an XML source with the values shown in Table 2. These are an extension of the mappings presented in [2].

$$f_{xml} : XSD \cup XML \rightarrow O$$

	x	$f_{xml}(x)$
Schema	$\forall st \in ST_{xsd}$	$rdfs : Datatype$
	$\forall ct \in CT_{xsd}$	$owl : Class$
	$\forall e \in E_{ct}, \forall ct \in CT_{xsd} :\ e.type = simple$	$owl : DatatypeProperty\ [rdfs : Domain = f_{xml}(ct),\ rdfs : Range = f_{xml}(e.type)]$
	$\forall e \in E_{ct}, \forall ct \in CT_{xsd} :\ e.type = complex$	$owl : ObjectProperty\ [rdfs : Domain = f_{xml}(ct),\ rdfs : Range = f_{xml}(e.type)]$
	$\forall a \in A_{ct}, \forall ct \in CT_{xsd}$	$owl : DatatypeProperty\ [rdfs : Domain = f_{xml}(ct),\ rdfs : Range = f_{xml}(a.type)]$
	$minOccurs$	$owl : minCardinality$
	$maxOccurs$	$owl : maxCardinality$
Instances	$\forall e \in E_{xml}$	$e.node \times rdf : type \times c \ \ where\ c = f_{xml}(e.type)$
	$\forall e \in E_{xml} :\ e.isLocal \wedge e.type = simple$	$i \times p \times e.value\ where\ i = f_{xml}(e.parent),\ p = f_{xml}(e)$
	$\forall e \in E_{xml} :\ e.isLocal \wedge e.type = complex$	$i \times p \times i'\ where\ i = f_{xml}(e.parent),\ p = f_{xml}(e),\ i' = f_{xml}(e.node)$
	$\forall a \in A_{xml}$	$a.parent \times p \times a.value\ where\ p = f_{xml}(a)$

Table 2: XML Extraction Function

Example. Applying the extraction rules presented in Table 2 to the XML schema depicted in Figure 2, produces the following ontology:

```
<rdf:RDF
  xmlns:tpcxml="http://www.tpc.org/tpc-di/xml"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <owl:Class rdf:about="tpcxml#Action"/>
  ...
  <owl:Class rdf:about="tpcxml#Customer"/>
  ...
  <owl:ObjectProperty rdf:about="tpcxml#hasAccount">
    <rdfs:domain rdf:resource="tpcxml#Customer"/>
    <rdfs:range rdf:resource="tpcxml#Account"/>
  </owl:ObjectProperty>
```

[6] XSD: http://www.w3.org/TR/xmlschema11-1/

```
  ...
  <owl:DatatypeProperty rdf:about="tpcxml#C_ID">
    <rdfs:domain rdf:resource="tpcxml#Customer"/>
    <rdfs:range rdf:resource="xsd#int"/>
  </owl:DatatypeProperty>
  ...
  <owl:DatatypeProperty rdf:about="tpcxml#CA_B_ID">
    <rdfs:domain rdf:resource="tpcxml#Account"/>
    <rdfs:range rdf:resource="xsd#int"/>
  </owl:DatatypeProperty>
  ...
</rdf:RDF>
```

Note how each element with a complex type is mapped to an OWL class. Similarly, each element with a simple type as well as each attribute is mapped to an OWL datatype property. Finally, an object property is created between each element and its sub-elements. The extraction process also produces the following RDF instance:

```
<Action rdf:about="tpcxml#Action_INS969036">
  <ActionType rdf:datatype="xsd#string">NEW</ActionType>
  <hasCustomer>
    <Customer rdf:about="tpcxml#Customer_INS969036">
      <hasAccount>
        <Account rdf:about="tpcxml#Account_INS969036">
          <CA_ID rdf:datatype="xsd#string">15092</CA_ID>
          ...
        </Account>
      </hasAccount>
      ...
    </Customer>
  </hasCustomer>
  ...
</Action>
```

3.3 Extraction Mappings

Given the extraction rules presented in Section 3.2, we need to capture the mappings between the extracted ontology and the corresponding data source. Regardless of the source's type, a mapping can be viewed as a triple

$$(x,\ f(x),\ access(x))$$

$$\begin{aligned} where\ \ & x: \textit{ the element in the data source},\\ & f(x): \textit{ the corresponding ontology element},\\ & access(x): \textit{ the access route to element } x \end{aligned}$$

The value of $access(x)$ depends on the type of the source. For relational databases, it includes the connection information to the database and the projection attributes from the corresponding table, which should always include the primary key columns. On the other hand, for XML sources it consists of the directory to the XML file and the XPath expression to the element x.

Example. Given the two previous extracted ontologies, the generated mappings are presented below using the format taken from [25]. This format was chosen due to its flexibility in capturing mappings with different types of sources, as discussed in Section 2.3.1.

```
<OntologyMapping>
   <Ontology type="owl:DatatypeProperty">
      tpcrdb:C_L_NAME
   </Ontology>
   <Mapping sourceKind="relational"
      connectionName="tpcdi">
       <Tablename>Customer</Tablename>
```

```
        <Projections>
           <Attribute>C_ID</Attribute>
           <Attribute>C_L_NAME</Attribute>
        </Projections>
    </Mapping>
</OntologyMapping>
...
<OntologyMapping>
  <Ontology type="owl:Class">
    tpcxml:Customer
  </Ontology>
  <Mapping sourceKind="xml"
    connectionName="tpcdi\cust_mgmt.xml">
    <Tablename>/Actions/Action</Tablename>
    <Projections>
      <Attribute>/Customer</Attribute>
    </Projections>
  </Mapping>
</OntologyMapping>
...
```

The first mapping in the above example maps an ontology property to a relational database and the value of $access(x)$ is a SQL query that can be formulated as

$$SELECT < Attribute_1/ >, ..., < Attribute_n/ >$$
$$FROM < TableName/ >$$

The second mapping maps an ontology class to an XML source and the $access(x)$ function is an XPath expression formulated as

$$XPath: \ / < TableName/ > / < Attribute/ >$$

3.4 Ontology Matching

Assuming we have two extracted ontologies O_1 and O_2, the matching process consists of finding all alignments between two entities $e_1 \in E_1$ and $e_2 \in E_2$. While alignments between two literals are theoretically possible, they are rarely produced in practice and hence are not considered.

Another important concept in the alignment is the relationship r between e_1 and e_2. The relationships that a matching process can find belong to a closed predefined set R. Usually, R contains at least the following relationships

$$R = \{equivalence, subsumption\}$$

Hence, an alignment a can be seen as an ontology statement

$$a = e_1 \times r \times e_2 \ where \ e_1 \in E_1, \ e_2 \in E_2, \ r \in R$$

We also define A as the set of all alignments between O_1 and O_2. A fully automatic system would find any alignment $\forall \ a \in A$, but in practice, most ontology matching systems only find a subset of A automatically and need to be manually enriched with further alignments by the user.

For the purposes of this work, we restrict the possible alignments to the DL-Lite profile of OWL (OWL 2 QL)[7]. OWL 2 QL is expressive enough to represent conceptual models such as UML and Entity-Relation diagrams and at the same time restricts the expressiveness of the alignments to avoid contradictions. Remarkably, in OWL 2 QL the usual TBOX reasoning tasks are polynomial in the size of the TBOX while yielding LOGSPACE with regard to the size of data for query answering (i.e. reducible to querying a RDBMS).

[7] OWL: http://www.w3.org/TR/2004/REC-owl-features-20040210/

In order to achieve this, we restrict the subsumption relations between two ontology entities $E_{left} \sqsubseteq E_{right}$ as defined by the following grammar

$$E_{left} \rightarrow c_1 \mid \exists Q_1$$
$$E_{right} \rightarrow c_2 \mid \exists Q_2 \mid \neg c_2 \mid \neg \exists Q_2$$
$$Q_1 \rightarrow p_1$$
$$Q_2 \rightarrow p_2$$
$$where\ (c_1 \in C_1 \wedge c_2 \in C_2) \vee (c_2 \in C_1 \wedge c_1 \in C_2),$$
$$(p_1 \in P_1 \wedge p_2 \in P_2) \vee (p_2 \in P_1 \wedge p_1 \in P_2)$$

Example. Continuing with our running example, the matching process between the two example ontologies presented in Section 3.2 and using Falcon-AO matchers (see Section 4) results in the following alignments:

```
<rdf:RDF>
  <onto1>
    <Ontology rdf:about="tpc:rdb">
      <location>file:/tpc_di_rdb.owl</location>
    </Ontology>
  </onto1>
  <onto2>
    <Ontology rdf:about="tpc:xml">
      <location>file:/tpc_di_xml.owl</location>
    </Ontology>
  </onto2>
  <Cell>
    <entity1>
      <edoal:Class rdf:about="tpc:rdb#Account"/>
    </entity1>
    <entity2>
      <edoal:Class rdf:about="tpc:xml#Account"/>
    </entity2>
    <relation>=</relation>
    <measure rdf:datatype=":&xsd#float">1.0</measure>
  </Cell>
  ...
  <Cell>
    <entity1>
      <edoal:Property rdf:about="tpc:rdb#C_L_NAME"/>
    </entity1>
    <entity2>
      <edoal:Property rdf:about="tpc:xml#C_L_NAME"/>
    </entity2>
    <relation>=</relation>
    <measure rdf:datatype=":&xsd#float">1.0</measure>
  </Cell>
  ...
</rdf:RDF>
```

In the above example, the element *measure* contains a float number reflecting the certainty of the alignment. These alignments are represented using EDOAL which was chosen because it is based on OWL and at the same time offers more flexibility in capturing more complex alignments that can be manually created by the user (see Section 2.3.2).

3.5 Ontology Merging

Assuming two ontologies O_1 and O_2 and a set of alignments A resulting from a matching process, the ontology merging process can be defined as a function

$$merge : O_1 \times O_2 \times A \rightarrow O_{merged}$$

The values of the *merge* function are the following

$$\forall s \in O_1 \rightarrow s \qquad \forall s \in O_2 \rightarrow s \qquad \forall a \in A \rightarrow a$$

In other words, the ontology merging process is simply combining the statements from the two input ontologies with the statements representing the alignments between them.

It is worth mentioning that the mappings between the ontologies and the sources do not change when merging the ontologies. This follows the principle of storing atomic mappings and composing them only when it is needed by the tool that uses them [11, 14].

Example. To finish with our running example, given the two ontologies extracted in Section 3.2 and the alignments generated in Section 3.4, the final output ontology of the merging process is the following:

```
<rdf:RDF
  xmlns:tpcrdb="http://www.tpc.org/tpc-di/rdb"
  xmlns:tpcxml="http://www.tpc.org/tpc-di/xml"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <owl:Class rdf:about="tpcrdb#Customer"/>
  <owl:Class rdf:about="tpcrdb#Account">
    <owl:equivalentClass
      rdf:resource="tpcxml#Account"/>
  </owl:Class>
  ...
  <owl:DatatypeProperty rdf:about="tpcrdb#C_L_NAME">
    <rdfs:domain rdf:resource="tpcrdb#Customer"/>
    <rdfs:range rdf:resource="xsd#string"/>
    <owl:equivalentProperty
      rdf:resource="tpcxml#C_L_NAME"/>
  </owl:DatatypeProperty>
  ...
</rdf:RDF>
```

In the above example, $owl : equivalentClass$ and $owl : equivalentProperty$ statements were added to the corresponding classes and properties. These statements are an OWL translation of the alignments resulting from the matching process (see Section 3.4).

4 Solution Prototype

The general architecture of the developed solution prototype is depicted in Figure 4. As it can be seen, the solution's main processes (ontology extraction, matching and merging) are completely independent of each other. Moreover, there is a dedicated output layer which encapsulates the logic needed to interact with the solution's output artifacts.

Ontology Extraction Module. The first step in the ontology construction process is extracting an ontology from the input data source. Simultaneously, this module generates the mappings between the extracted ontology and the source based on the formalization given in Section 3.2.

Ontmalizer was used in extracting ontologies from XML sources. Its source code was significantly modified in order to incorporate the generation of the mappings between the extracted ontology and the source.

On the other hand, a custom extraction module was developed to deal with relational databases. This is due to the fact that most available tools either produce RDF triples instead of OWL ontologies or are stand-alone tools whose source code is not available for modification. Currently, the module supports MySQL databases and is extensible to handle other common types of relational databases such as SQL Server and Oracle. Further implementation details and information on how to extend this module are available in Appendix C.

Ontology Matching Module. This module matches the extracted ontology with the output ontology from the previous iteration. It produces the alignments between the two ontologies as formalized in Section 3.4.

The Alignment API 4.0 [8] was used as the backbone of this module. It provides a general framework where any independent matcher can be integrated. In this prototype, three different techniques from Falcon-AO [22] were integrated to match the input ontologies:

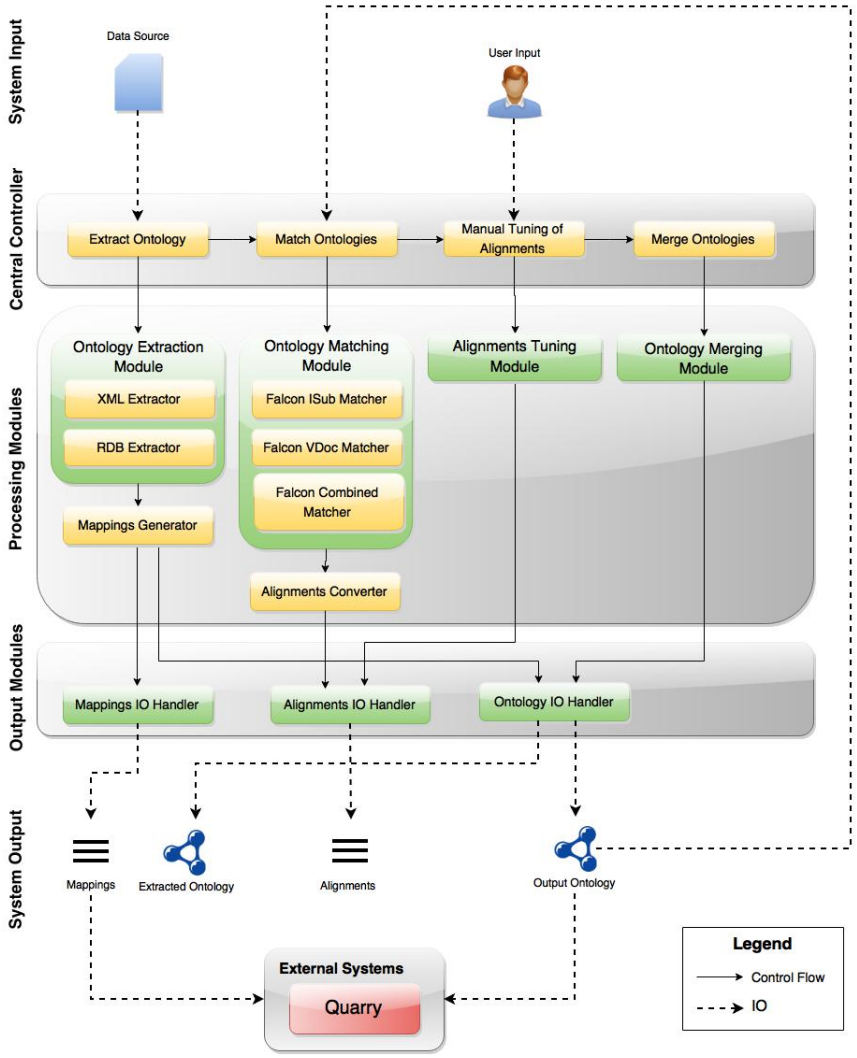


Fig. 4: Solution Architecture

- I-Sub matcher: a novel string matching technique presented in [39],
- V-Doc matcher: the "virtual documents" technique based on TF.IDF concept of information retrieval [42],
- Combined matcher: combines the above two matchers together with a graph comparison algorithm [21]. This matcher also includes a partitioning algorithm in order to handle larger ontologies.

Since the alignments generated in the prototype are represented in EDOAL, a conversion module was implemented that converts the alignments resulting from Falcon-AO to the EDOAL language. Such a module must be implemented for each matching tool to be integrated into the solution. Further implementation details and information on how to extend this module are available in Appendix D.

Alignments Tuning Module. This module provides an interface where the alignments are displayed to the user who can further add, modify or delete them. This module is needed given that most ontology matching approaches are semi-automatic (see Section 2.2). Hence, the user needs to correct the automatic results and manually add complex alignments that cannot be automatically detected by ontology matchers. Currently, this module only allows

the user to review the alignment sets resulting from each of the three integrated matchers and to choose which set to use in the ontology merging process.

Ontology Merging Module. As a final step, this module makes use of the alignments generated by the previous module in order to merge the ontologies following the rules presented in Section 3.5. It outputs the merged ontology which is used as input to the ontology matching module in the next iteration.

Output Modules. The output modules in the prototype were separated from the processing modules. Jena was used to read / write ontologies throughout the prototype. The Alignment API's output modules were used to interact with the EDOAL alignments generated during the matching process. Finally, an output module was developed for the extraction mappings using DOM and SAX Parser.

4.1 Integration with Quarry

Quarry is a data integration system that automates the physical design of a data warehousing system from high-level information requirements and provides tools to efficiently accommodate multidimensional schema and ETL process designs [25].

As can be seen from Figure 5, the user defines these high-level requirements through the Requirements Elicitor component, which takes as input a domain ontology capturing the underlying data sources. After the definition of the requirement, the Requirements Interpreter component uses the mappings between the domain ontology and the data sources to automatically generate the multidimensional schema and ETL flows that correspond to the requirement.

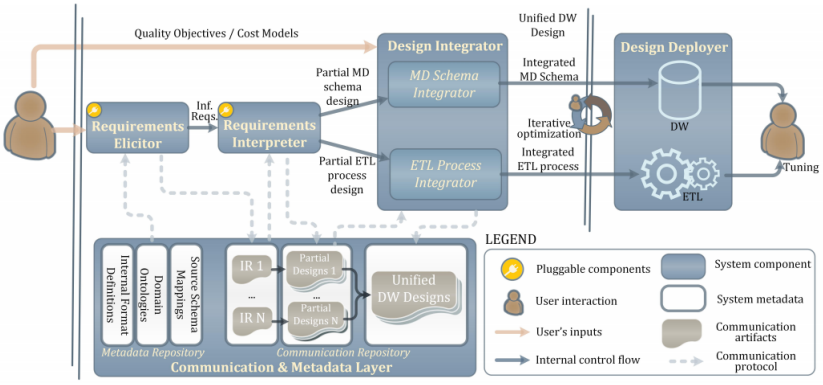


Fig. 5: Quarry System Architecture

However, Quarry does not produce such an ontology but rather assumes that both the ontology and the mappings are pre-existing. We integrated our prototype with Quarry and used it as a pre-processing step to generate the domain ontology and the mappings, thus eliminating the requirement of having a predefined target schema, in this case represented by the domain ontology. A step-by-step explanation of how Quarry takes advantage of the output produced by the prototype of our solution is available in Appendix E.

Similarly, the output artifacts generated by our prototype can be consumed by any other data integration system. The output ontology is a standard OWL ontology that can be parsed by any OWL-compliant library. Moreover, the independent IO modules that were developed as part of the prototype can be used by third-party systems to interact with the generated mappings.

5 Case Study

As a case study, we evaluated our prototype on a subset of the TPC-DI benchmark data set which contains an XML and a relational database source, as introduced in Section 1.1.

5.1 Ontology Extraction

To evaluate the extraction module, we compared the ontology resulting from our prototype O_{auto} with an ontology that was manually built out of the same sources as part of [25] O_{man}. We used the following metrics in the comparison

$$Recall = \frac{|E_{auto} \cap E_{man}|}{|E_{man}|} \qquad AK = 1 - \frac{|E_{auto} \cap E_{man}|}{|E_{auto}|}$$

Recall measures the percentage of the entities in the manual ontology that were automatically extracted while *AK* (*Additional Knowledge*) measures the percentage of the entities that we extracted that are not present in the manual ontology. The results of these metrics are shown in Table 3.

	Recall	Additional Knowledge (AK)
RDB	64.4%	2.4%
XML	100%	34.2%

Table 3: Extraction Case Study Results

The low *Recall* in the case of RDB extraction as well as the high *AdditionalKnowledge* for XML extraction are a result of design decisions in the manual ontology. As can be noted from Figure 6, which depicts an extract of O_{auto} and O_{man}, O_{auto} retains the same normalization level as the initial data sources while O_{man} does not.

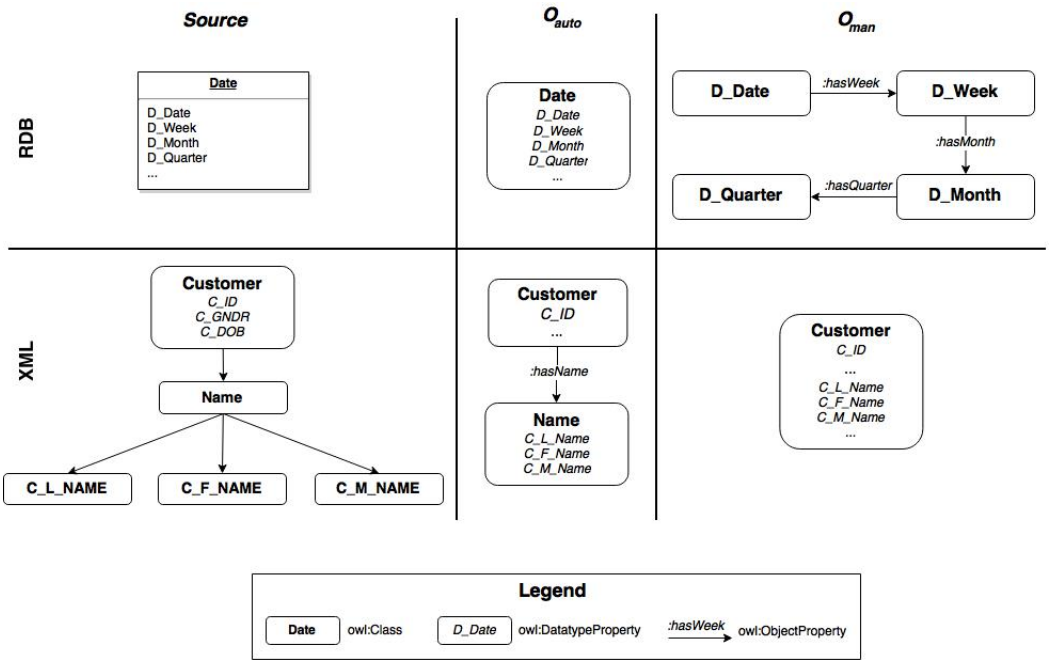


Fig. 6: Extraction Case Study Comparison

In the case of the relational database, O_{man} includes a different OWL class for each column of the "Date" table, thus converting a denormalized source to a more normalized design. While this results in a better conceptual design, the extraction rules cannot automatically produce such an output given that the needed functional dependencies are not available in the source.

As for the XML source, O_{man} denormalizes the design of the source by disregarding some XML elements, as can be noted by the absence of a "Name" class. This results in a different number of OWL classes in O_{man} than in O_{auto}, depending on the level of normalization followed by the designer, and explains the results presented in Table 3.

In contrast, the design of O_{auto} captures the design of the underlying sources more faithfully by retaining the same normalization level. Denormalization is an irreversible step that should be taken at the logical or physical levels and denormalizing a source at the conceptual level, such as the case with the XML source in O_{man}, is a faulty design decision. The ontology is meant to act as a conceptual schema of the underlying sources and should capture as much semantic knowledge from these sources as possible.

While having a semi-automatic approach would resolve these issues as it would allow the users to tune the design of the automatically extracted ontology, the purpose of the solution proposed in this work is achieving the highest degree of automation possible in the ontology extraction and construction processes.

This highlights the impact that the quality of the source's design will have on the design of the automatically extracted ontology. In the case of relational databases for instance, the database must be in at least Boyce-Codd Normal Form (BCNF) to produce quality results. Similarly, the XML schema must include a separate complex type for each domain concept in order for them to be converted into OWL classes in the resulting ontology.

5.2 Ontology Matching

To evaluate the matching module, we compared the alignments generated by each of the three integrated matchers (see Section 4) with a manually created set of alignments. Table 4 shows the experimental results presented with the standard information retrieval metrics.

	Precision	Recall	F-Measure
I-Sub	95.8%	82.1%	88.4%
V-Doc	72.2%	46.4%	56.5%
Combined	100%	57.1%	72.7%

Table 4: Matching Case Study Results

Both the I-Sub and the Combined matchers gave results comparable to the tests presented in [39] and [22] respectively. On the other hand, the V-Doc matcher scored considerably lower than the results reported in [42].

These results can be explained by looking at the ontologies extracted from the used TPC-DI data sources. The ontologies contain many similarly-named concepts and properties, which is ideal for a string-comparison approach such as I-Sub. However, in cases where this is not true, V-Doc may perform better given that it takes advantage of the hierarchical structure and neighborhood information of a concept.

It should be noted that these matchers were integrated into the prototype due to their representativeness of the various types of ontology matching techniques (see Section 2.2) and can be replaced with any other ontology matcher. More complete experiments on these matchers are available in [39], [22] and [42].

6 Conclusions and Future Work

In this work, we have demonstrated how semi-automatic ontology construction can be used to support data integration in eliminating the requirement of a predefined target schema.

We proposed a solution that integrates existing ontology extraction, matching and merging techniques into a single end-to-end system. Moreover, we extended the common functionality of ontology extraction techniques with the generation of mappings between the extracted ontologies and the data sources, a crucial artifact in the data integration field.

We have also developed a prototype of the proposed solution and shown how its output artifacts can be used as a pre-processing step for Quarry, a data integration system. The prototype uses Falcon-AO for the matching process and is easily extensible to include other matching techniques.

In the future, we will develop new modules to extract ontologies from other common types of data sources, such as text documents. Moreover, the manual alignments tuning module will be implemented with a graphical interface where users can tune the results of the matching process and express complex alignments that the automatic matchers cannot discover. Finally, we plan on extending the generated mappings beyond the representation of instance-level changes in order to capture schema evolution of the underlying data sources.

References

1. Algergawy, A., Massmann, S., Rahm, E.: A clustering-based approach for large-scale ontology matching. In: Proceedings of ADBIS. pp. 415 – 428. Springer (2011)
2. Bohring, H., Auer, S.: Mapping XML to OWL ontologies. In: Proceedings of 13. Leipziger Informatik-Tage (LIT 2005). pp. 147 – 156. GI (2005)
3. Castano, S., De Antonellis, V., De Capitani di Vimercati, S., Melchiori, M.: Semi-automated extraction of ontological knowledge from XML data sources. In: Proceedings of DEXA. pp. 852–860. IEEE (2002)
4. Cerbah, F.: Learning highly structured semantic repositories from relational databases: The RDBToOnto tool. In: In Proceedings of ESWC. pp. 777 – 781. Springer (2008)
5. Cotterell, M.E., Medina, T.: A Markov model for ontology alignment. CoRR (2013)
6. Cruz, I.F., Palandri Antonelli, F., Stroe, C.: AgreementMaker: Efficient matching for large real-world schemas and ontologies. VLDB pp. 1586 – 1589 (2009)
7. Dadjoo, M., Kheirkhah, E.: An approach for transforming of relational databases to OWL ontology. International Journal of Web & Semantic Technology 6(1) (2015)
8. David, J., Euzenat, J., Scharffe, F., Dos Santos, C.T.: The Alignment API 4.0. Semantic Web Journal 2, 3–10 (2011)
9. De Giacomo, G., Lembo, D., Lenzerini, M., Rosati, R.: On reconciling data exchange, data integration, and peer data management. In: Proceedings of ACM PODS. pp. 133 – 142. ACM (2007)
10. De Laborda, C.P., Conrad, S.: Relational.OWL - a data and schema representation format based on OWL. In: Proceedings of CRPIT. pp. 89–96. Australian Computer Society (2005)
11. Dessloch, S., Hernández, M.A., Wisnesky, R., Radwan, A., Zhou, J.: Orchid: Integrating schema mapping and etl. In: Proceedings of ICDE. pp. 1307–1316. IEEE (2008)
12. Doan, A., Madhavan, J., Domingos, P., Halevy, A.: Learning to map between ontologies on the semantic web. In: WWW2002. ACM (2002)
13. Ehrig, M., Staab, S.: Qom - Quick ontology mapping. In: Proceedings of ISWC. pp. 683–697. Springer (2004)
14. Fagin, R., Haas, L.M., Hernández, M., Miller, R.J., Popa, L., Velegrakis, Y.: Clio: Schema mapping creation and data exchange pp. 198–236 (2009)
15. Fagin, R., Kolaitis, P.G., Miller, R.J., Popa, L.: Data exchange: Semantics and query answering. Theoretical Computer Science 336, 89–124 (2005)
16. Fridman, N., Musen, M.: PROMPT: Algorithm and tool for automated ontology merging and alignment. Proceedings of AAAI pp. 450–455 (2000)
17. Georgiu, M., Groza, A.: Ontology enrichment using semantic wikis and design patterns. Studia Universitatis Babes-Bolyai, Informatica 56(2) (2011)

18. Giunchiglia, F., Shvaiko, P., Yatskevich, M.: S-Match: An algorithm and an implementation of semantic matching. In: The Semantic Web: Research and Applications, vol. 3053, pp. 61–75. Springer (2004)
19. Granitzer, M., Sabol, V., Onn, K.W., Lukose, D., Tochtermann, K.: Ontology alignment - a survey with focus on visually supported semi-automatic techniques. Future Internet 2, 238–258 (2010)
20. Gross, A., Hartung, M., Kirsten, T., Rahm, E.: On matching large life science ontologies in parallel. In: Proceedings of DILS. pp. 35–49. Springer (2010)
21. Hu, W., Jian, N., Qu, Y., Wang, Y.: GMO: A graph matching for ontologies. In: Proceedings of K-Cap. pp. 43–50 (2005)
22. Hu, W., Qu, Y., Cheng, G.: Matching large ontologies: A divide-and-conquer approach. Data and Knowledge Engineering 67, 140–160 (2008)
23. Jauro, F., Junaidu, S., Abdullahi, S.: Falcon-AO++ - an improved ontology alignment system. International Journal of Computer Applications (2014)
24. Jean-Mary, Y.R., Shironoshita, E.P., Kabuka, M.R.: Ontology matching with semantic verification. Journal of Web Semantics 7, 235–251 (2009)
25. Jovanovic, P., Romero, O., Simitsis, A., Abelló, A., Candón, H., Nadal, S.: Quarry: Digging up the gems of your data treasury. In: Proceedings of EDBT. pp. 549–552. OpenProceedings (2015)
26. Kolli, R., Doshi, P.: OPTIMA: tool for ontology alignment with application to semantic reconciliation of sensor metadata for publication in sensormap. In: Proceedings of ICSC. pp. 484–485. IEEE Computer Society (2008)
27. Lambrix, P., Tan, H.: SAMBO - a system for aligning and merging biomedical ontologies. Journal of Web Semantics 4, 196–206 (2006)
28. Lenzerini, M.: Data integration: A theoretical perspective. In: Proceedings of ACM PODS. pp. 233–246. ACM (2002)
29. Li, J., Tang, J., Li, Y., Luo, Q.: RiMOM - a dynamic multistrategy ontology alignment framework. IEEE TKDE 21, 1218–1232 (2009)
30. Lubyte, L., Tessaris, S.: Automatic extraction of ontologies wrapping relational data sources. In: Proceedings of DEXA. vol. 5690, pp. 128–142. Springer (2009)
31. Maedche, A., Motik, B., Silva, N., Volz, R.: MAFRA - A mapping framework for distributed ontologies in the semantic web. In: Proceedings of IC3K. pp. 235–250. Springer (2002)
32. Mallede, W., Marir, F., Vassilev, V.: Algorithms for mapping RDB schema to RDF for facilitating access to deep web. In: Proceedings of WEB. IARIA XPS Press (2013)
33. Ngo, D., Bellahsene, Z.: YAM++: A multi-strategy based approach for ontology matching task. In: Proceedings of IC3K. pp. 421–425. Springer (2012)
34. Petasis, G., Karkaletsis, V., Paliouras, G., Krithara, A., Zavitsanos, E.: Ontology population and enrichment: State of the art. Knowledge-Driven Multimedia Information Extraction and Ontology Evolution pp. 134–166 (2011)
35. Robin, C.R.R., Uma, G.V.: A novel algorithm for fully automated ontology merging using hybrid strategy. European Journal of Scientific Research 47(1), 1450–216 (2010)
36. Sequeda, J.: On the semantics of R2RML and its relationship with the direct mapping. In: Proceedings of the International Semantic Web Conference. pp. 193–196. CEUR-WS (2013)
37. Shvaiko, P., Ome Euzenat, E.: Ontology matching: State of the art and future challenges. IEEE TKDE 25(1), 158–176 (2011)
38. Spohr, D., Hollink, L., Cimiano, P.: A machine learning approach to multilingual and cross-lingual ontology matching. In: Proceedings of ISWC. pp. 665–680. Springer (2011)
39. Stoilos, G., Stamou, G., Kollias, S.: A string metric for ontology alignment. In: Proceedings of ISWC. pp. 624–637. Springer (2005)
40. Suchanek, F.M., Abiteboul, S., Senellart, P.: PARIS: Probabilistic alignment of relations, instances, and schema. VLDB 5, 157–168 (2011)
41. Wang, S., Zeng, Y., Zhong, N.: Ontology extraction and integration from semi-structured data. In: Proceedings of AMT. pp. 39–48. Springer (2011)
42. Zhang, H., Hu, W., Qu, Y.: Constructing virtual documents for ontology matching using MapReduce. In: Proceedings of JIST. pp. 48–63. Springer (2012)

Appendix A Overview of Ontology Matching Approaches

The studied ontology matching approaches have been categorized according to several dimensions: input format, used matching techniques, scope and size of ontology. In this appendix, we briefly describe each of these dimensions before giving an overview of the approaches in Table 5.

- **Input Format:** describes the format of the ontologies that each approach supports. The most popular choices are RDF, OWL or both. It is worth mentioning that even if an approach accepts ontologies in OWL as input, it does not mean that it utilizes the full capacity of the language in the matching process.
- **Matching Techniques:** describes the ontology matching techniques that the approach uses. They are classified into one of the following categories: terminological, linguistic, structural, and semantic (see Section 2.2.1).
- **Scope:** indicates whether the approach was designed as a general-purpose or a domain-specific tool. Most of the studied approaches were designed as general-purpose tools with the exception of [27] and [24] which are targeted towards biomedical ontologies.
- **Size of ontology:** indicates whether the approach applies any specific technique to deal with larger ontologies. While all approaches work on large ontologies, their performance is usually considerably worse than those who apply a specific technique to handle large ontologies.

Paper	Input	Matching Techniques				Scope	Size	Notes
		Terminological	Linguistic	Structural	Semantic			
SAMBO [27]	OWL	Edit distance, n-gram	WordNet	Structural similarity	-	Biomedical		-
Falcon-AO [22]	RDFS, OWL	I-sub	-	Divide-and-conquer, virtual docs	-		Large	-
Falcon-AO++ [23]	RDFS, OWL	*same as Falcon-AO with user input*					Large	-
RiMOM [29]	OWL	Edit distance	WordNet	Similarity propagation	Dynamic matcher selection			Matcher chosen based on pre-calculated factors, inefficient with large ontologies
ASMOV [24]	OWL	String equality, Levenstein distance	WordNet	Hierarchical similarities	-	Biomedical		Verification to detect semantic inconsistencies
AnchorFlood [37]	RDFS, OWL	String equality, Winkler-based similarity	WordNet	Iterative similarity propagation	-		Large	Starts from core "anchor" concepts (100% similarity) and "floods" neighbors
AgreementMaker [6]	RDFS, OWL	TF.IDF, edit distance, substrings	WordNet	Descendant and sibling similarity	-			User must be domain expert
PARIS [40]	RDFS	Probabilistic model, iterative instance, class matching	-	-	-			Ontologies should have a lot of instances in common
OPTIMA [26]	RDFS, OWL	Expectation-maximization, lexical similarity	-	Structural similarity	-			-

QOM [13]	RDFS	Dice coefficient, Levenstein distance, cosine sim, weighted average of sims	-	Structural knowledge	-		Large	Prunes matches below a specific threshold
Gross et al. [20]	-	*Intra-matcher, inter-matcher parallelism*					Large	Simple matching and partitioning techniques
S-Match [18]	-	Tokenization, String comparison	-	Node matching	Semantic Matching			-
GLUE [12]	-	*Mult-strategy machine learning, relaxation labeling*						Similarity measure can be changed
ONION [19]	RDFS	String matching	WordNet, document corpus	-	-			-
MAFRA [31]	RDFS	String matching	-	-	Semantic bridging, distribution			Similarity measures implemented are very basic
PROMPT [16]	RDFS, OWL	*Linguistic similarity, user feedback*						General algorithm without specific similarity measure
Cotterell et al. [5]	-	Levenstein distance	-	Edge confidence (Markov chain)	-			-
YAM++ [33]	-	*Machine learning, Similarity Flooding*						-
Algergawy et al. [1]	XML Schema, OWL	TF.IDF	-	Clustering, Vector-space model	-		Large	-
Spohr et al. [38]	-	String matching	-	Structural features	-			Cross-lingual matching, machine learning
Robin et al. [35]	OWL	String matching	WordNet	-	-			Uses heuristics in the matching process

Table 5: Summary of Ontology Matching Approaches

Appendix B Introducing Heuristics into Ontology Extraction

The mappings presented in Section 3.2 follow deterministic rules and give accurate results in the resulting ontology. However, capturing some advanced features from the sources cannot be done by applying deterministic rules only but rather requires the use of heuristics.

This appendix includes a brief description of some cases where introducing heuristics into the extraction process would allow the solution to capture further knowledge from the underlying sources.

1. **Relational Databases.** In the case of relational databases, the following features can be captured by the use of heuristics [30]:
   (a) One important feature that cannot be captured by deterministic rules is inheritance. The basic heuristic to determine if a table t is a sub-table of table t' is to check if the primary key of t is also a foreign key from t referencing t'. However, this rule is not 100% accurate and it might identify inheritance in some cases where there is actually no inheritance relation.
   (b) Another feature that can be captured by heuristics is to determine if a table t is used to break an n-to-m relationship between two or more other tables. Such tables are usually characterized by a primary key that consists of several foreign keys referencing multiple tables.
   Once identified, these tables should not be translated to OWL classes, as they do not represent any real-world object, but rather to object properties between the set of tables that they reference.
2. **XML Sources.** As for XML sources, applying heuristics would help in identifying elements of the XML schema that do not correspond to real-world objects such as the element "Actions" in the running example (see Section 1.1) which is merely a list of "Action" elements.
   Once identified, these elements should be eliminated from the extraction process as they do not represent any real-world objects.

Given that these rules are heuristic, they might yield inaccurate results and including them in the ontology extraction process requires a semi-automatic approach where the user can tune the resulting ontology. This defeats the purpose of the solution proposed in this work which aims for the highest degree of automation possible in the ontology extraction and construction processes.

Appendix C Ontology Extraction Module Implementation

This appendix includes technical details about the implementation of the Ontology Extraction Module and how it can be extended. Figure 7 depicts a package diagram of the module as it is implemented in the prototype. Abstract classes are depicted with a list of the abstract methods that they include.

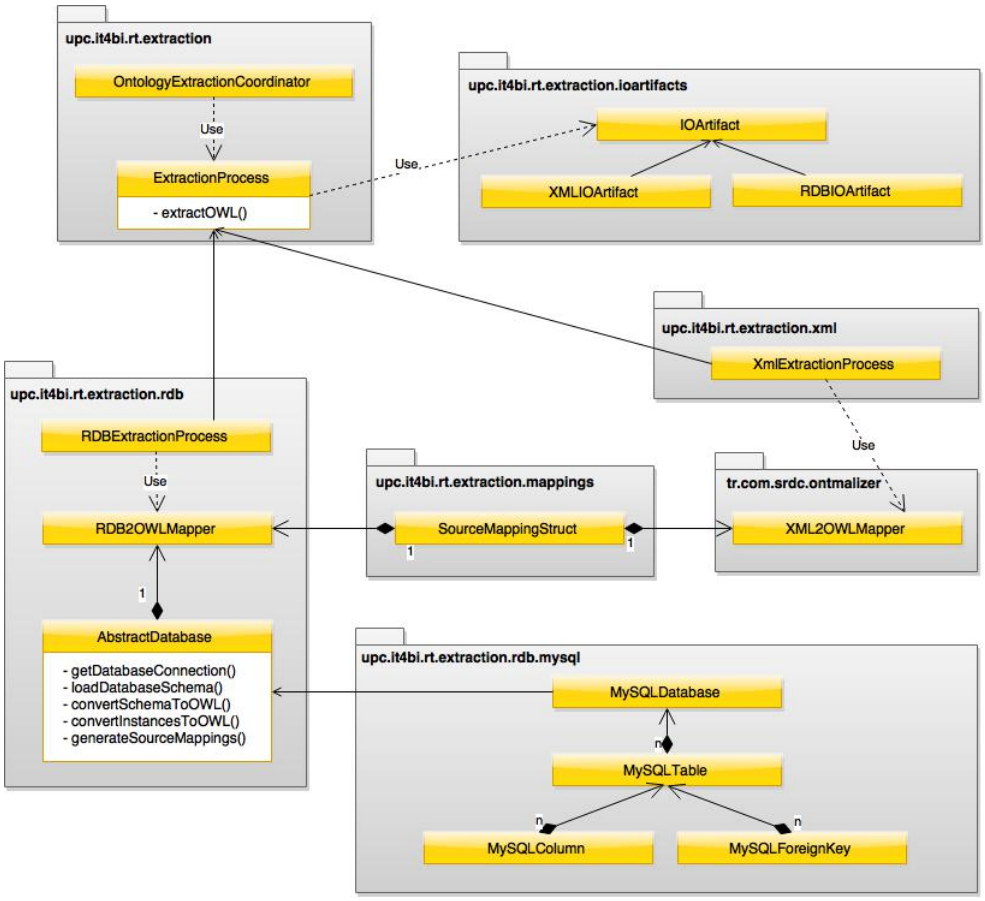


Fig. 7: Ontology Extraction Module - Package Diagram

The module was designed to be easily extensible. As we can see in Figure 7, the *Ontology ExtractionCoordinator* uses an abstract *ExtractionProcess* class to extract an ontology from a source represented by the $IOArtifact$ class. In turn, both *ExtractionProcess* and $IOArtifact$ are extended through inheritance to accommodate the extraction process from each supported type of data sources.

It can also be noted from Figure 7 that the source mappings are integrated within the module through composition. Both the $XML2OWLMapper$ and $RDB2OWLMapper$ classes contain an instance of *SourceMappingStruct*, which corresponds to the mappings between the extracted ontology and the underlying data source.

The Ontology Extraction Module can be extended in two different directions; adding support to a new type of data source or adding support to a new type of relational database.

1. **Supporting a new source type.**
   (a) Create a subclass of $IOArtifact$ corresponding to the new source type to be supported. The new class should contain information related to the source (e.g. $XMLIO$

Artifact contains the file path of the XSD schema file while *RDBIOArtifact* contains the connection information to the database).

(b) Create a subclass of *ExtractionProcess* overriding its abstract method, *extractOWL*(). The new class can either directly contain the extraction of the ontology or can use a third-party tool to perform the actual extraction (such as the case with using *XML2OWLMapper* in the case of XML sources).

(c) The generation of the mappings must be embedded directly into the extraction process. Thus, the overriden version of the *extractOWL*() method must simultaneously generate the source mappings.

2. **Supporting a new database type.**

(a) Create a subclass of *AbstractDatabase* overriding all of its abstract methods. These methods perform complementing tasks that are coordinated by the *RDB2OWLMapper* class in order to extract the ontology from the relational database.

(b) As can be seen in Figure 7, the *upc.it4bi.rt.extraction.rdb.mysql* package contains a class representing each main concept in a relational database (i.e. table, column, foreign key). Each of these classes contains the extraction method of the corresponding element from the database. The developer can decide whether to follow this structure for other types of databases or to incorporate the entire extraction process in the newly created subclass of *AbstractDatabase*.

Appendix D Ontology Matching Module Implementation

This appendix includes technical details about the implementation of the Ontology Matching Module and how to extend it. Figure 8 depicts a package diagram of the module as it is implemented in the prototype. Abstract classes are depicted with a list of the abstract methods that they include.

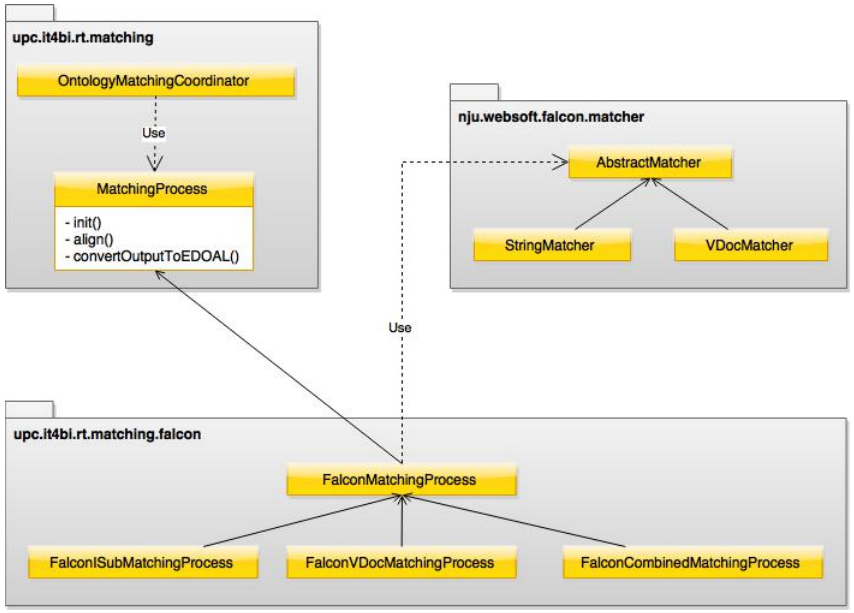


Fig. 8: Ontology Matching Module - Package Diagram

Similarly to the Ontology Extraction Module, this module was designed to be easily extended. As we can see in Figure 8, the *OntologyMatchingCoordinator* uses an abstract *MatchingProcess* class to match two ontologies. In turn, *MatchingProcess* is extended through inheritance to integrate any external matcher, such as the case with *FalconMatching Process* and its subclasses, which correspond to three different matching techniques from the Falcon-AO tool.

1. **Integrating a new matcher.**
   (a) Create a subclass of *MatchingProcess* and override all of its abstract methods. These methods, depicted in Figure 8, represent, respectively, the initialization of the matching process, the matching process itself, and the conversion of the output to EDOAL.
   (b) This new class is expected to act as a wrapper around the new matching tool to be integrated. For instance, in the case of *FalconMatchingProcess*, the *init*() method is used to pass any initialization parameters to *AbstractMatcher* while the *align*() method invokes the actual matching process. On the other hand, the *convertOutputToEDOAL*() method includes a custom implementation that converts the output of the matching process to the EDOAL representation used throughout the prototype.
   (c) Add the name of the new class to the project properties so it can be automatically used in future ontology construction processes.

Appendix E Integration with Quarry

Quarry automatically generates the multidimensional schema and ETL flows of a data warehousing system from high-level information requirements. [25]. In order to support this functionality, it makes use of a pre-defined domain ontology capturing the underlying data sources as well as of mappings between this ontology and the sources. This appendix includes a step-by-step explanation of how our prototype can be used as a pre-processing step to generate the domain ontology and mappings that Quarry requires.

Step 1: Creating a new ontology construction project. We launch our prototype in order to create a new ontology construction project. Figure 9 shows the interface of the prototype where the sources to be integrated can be added.

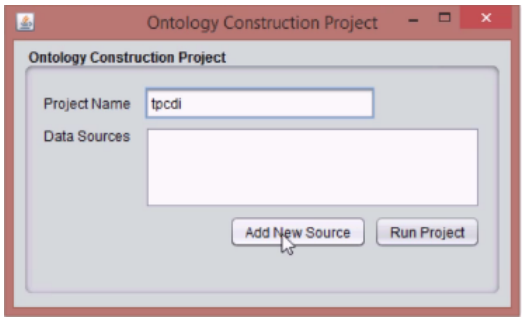
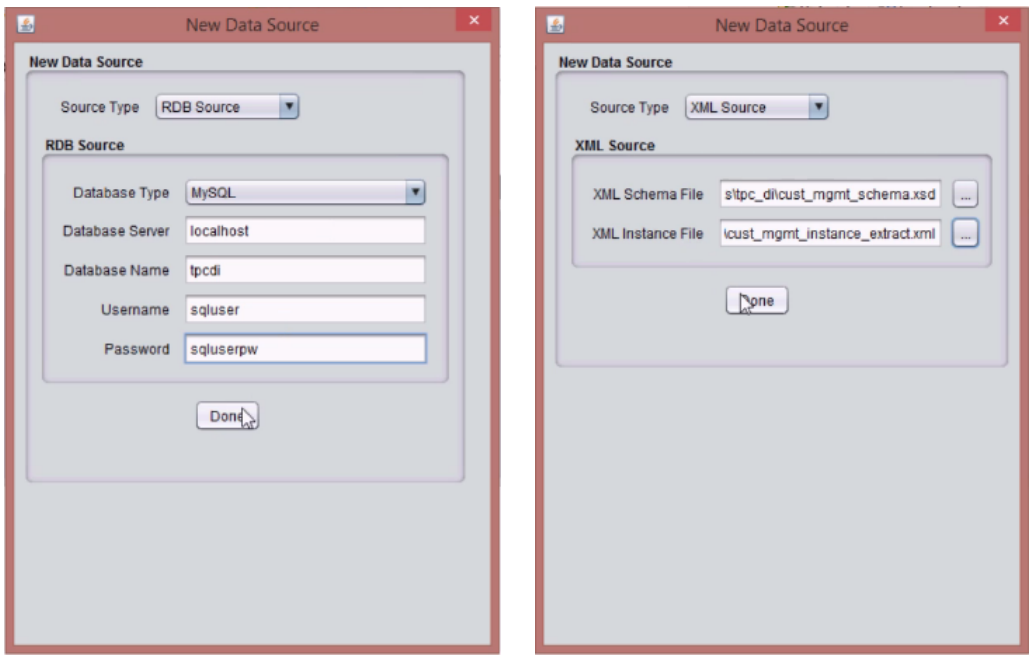


Fig. 9: Creating a new Ontology Construction Project

Step 2: Adding data sources to the project. After creating the project, we proceed to add the sources one by one. The pop-up window that appears includes a drop down list of the supported source types. Figure 10 shows how a relational database and an XML source can be added. The sources used in this example are the ones taken from the TPC-DI data set as explained in Section 1.1.



(a) Relational Database Source **(b)** XML Source

Fig. 10: Adding a New Source to the Ontology Construction Project

Step 3: Reviewing the ontology matching results. After adding all of the data sources and running the project, the prototype automatically runs the ontology matching process using each of the integrated matchers separately. The sets of alignments resulting from running each matcher are then displayed to the user as depicted in Figure 11.

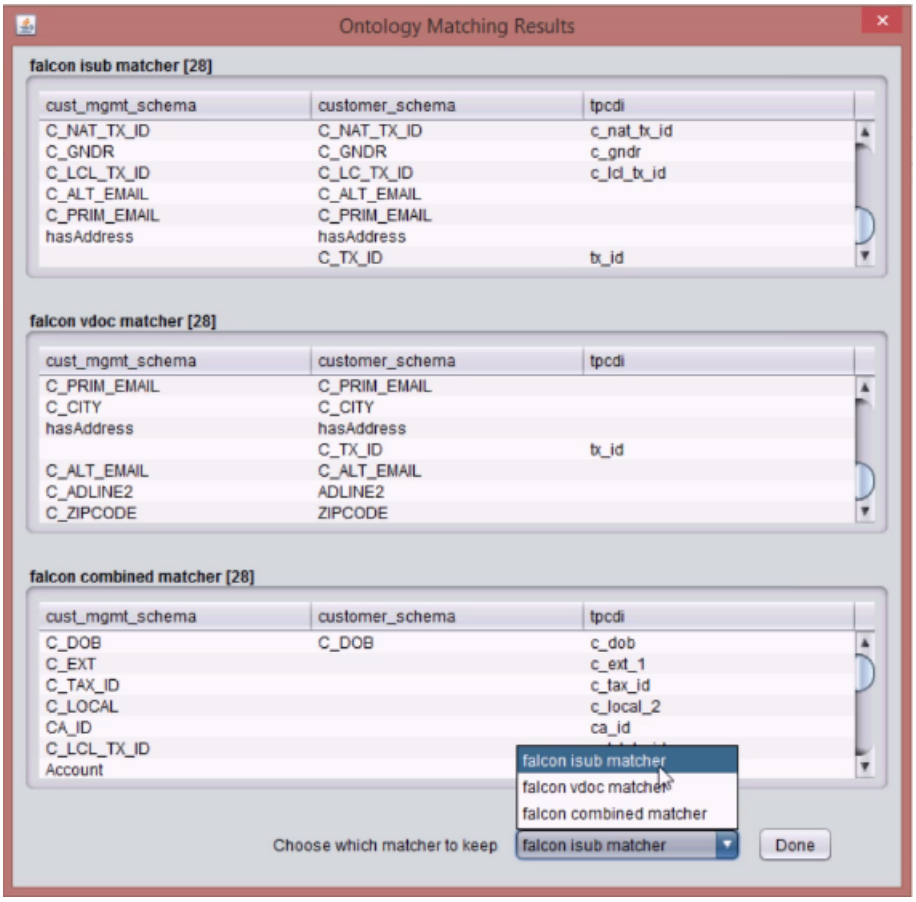


Fig. 11: Reviewing Ontology Matching Results

Through the interface depicted in Figure 11, the user can review the matching results of each of the integrated matchers and decide which set of alignments to use in the ontology merging process.

It should be noted that once the Manual Alignments Tuning module is fully implemented, the user will be able to tune the alignments (e.g. delete, edit or add new alignments) through this interface and not only review them.

Step 4: Finalizing the ontology construction process. After the user chooses which set of alignments to use, the prototype runs the ontology merging process and outputs several files depicted in Figure 12. The two highlighted files, which correspond to the final output ontology and the mappings between the ontology and the sources, are the ones to be used as input by Quarry.

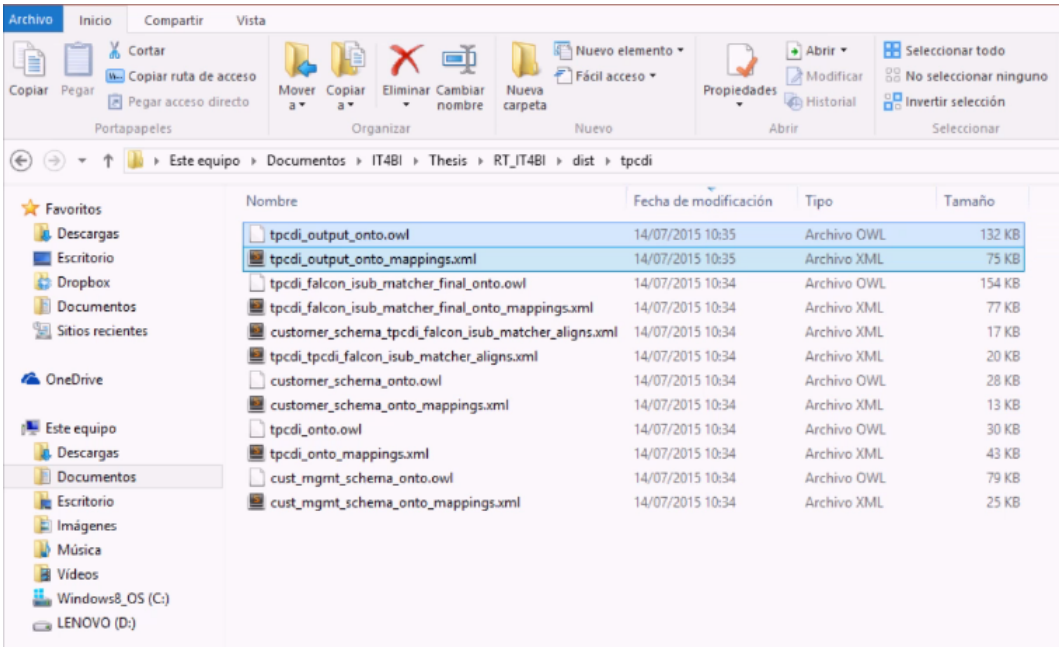


Fig. 12: Ontology Construction Output Files

Step 5: Creating a new Quarry project. After running the ontology construction process and obtaining the necessary input for Quarry, we create a new Quarry project by uploading the domain ontology and the mappings between the ontology and the data sources, as depicted in Figure 13.

Fig. 13: Creating a new Quarry Project

Step 6: Creating a new requirement. After the project is created, we create a new high-level requirement. To do so, Quarry displays a visualization of the domain ontology uploaded when creating the project as shown in Figure 14. Creating a requirement involves two main steps; defining a fact and defining at least one dimension. Figure 15 shows how a fact can be created while 16 shows how a dimension can be added with Quarry.

In this example, assume we want to analyze the average *bid price* of a *trade* by customer for trades that are of type *sell*. The involved concepts from the ontology shown in Figure 14 are thus *trade*, *customer* and *trade type*.

Fig. 14: Creating a New Requirement

In Figure 15, we define as a fact the average *bid price* of a *trade*. Whereas in Figure 16, we are defining a restriction on *trade type* to constraint it to trades of type *sell* only. Similarly, we define *customer* as a dimension without any restrictions.

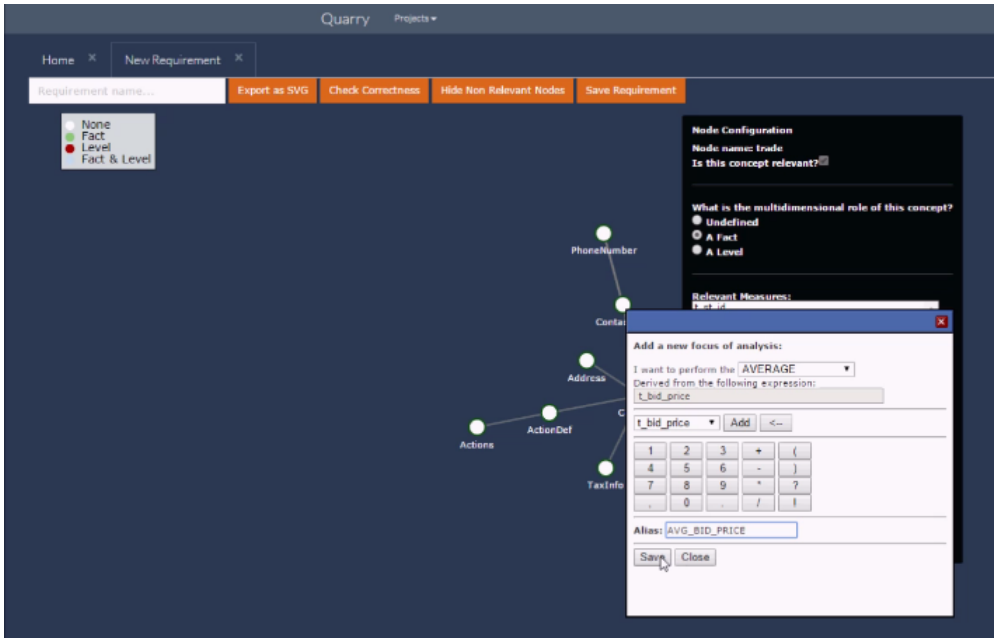


Fig. 15: Defining a New Requirement Fact

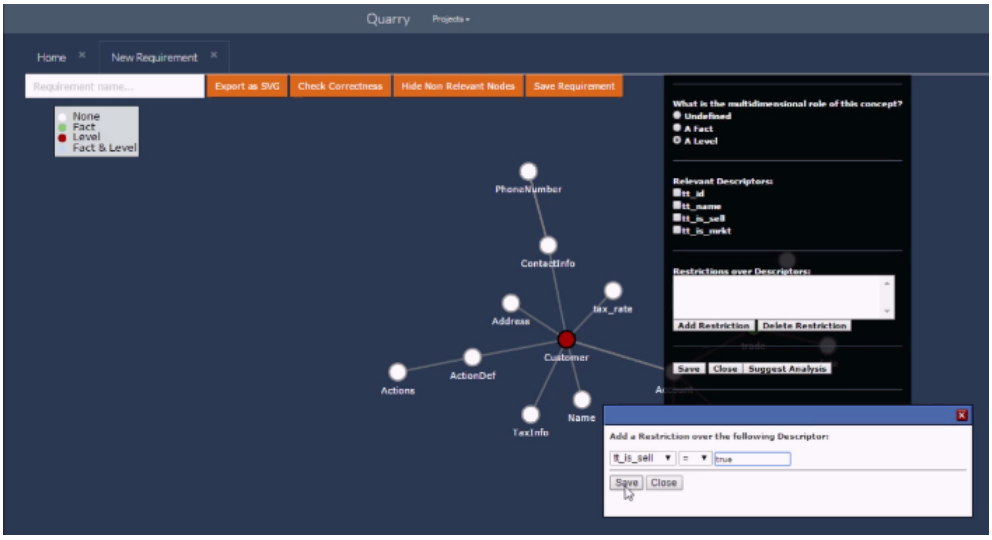


Fig. 16: Defining a New Requirement Dimension

Step 7: Multidimensional Schema (MDS). After defining the requirement, Quarry automatically generates the MDS representing this requirement in terms of the elements and attributes of the initial data sources (e.g. XML elements, database tables and columns). Figure 17 shows an example of such an MDS.

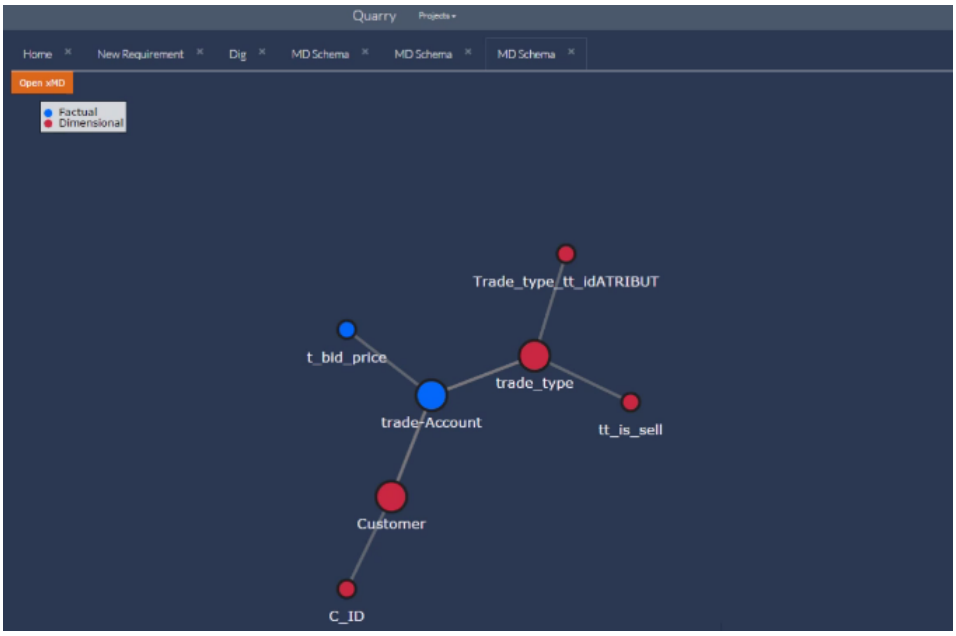


Fig. 17: Multidimensional Schema in Quarry

Step 8: ETL Flow. Moreover, Quarry also generates the ETL flow that transforms the data from the sources into the target schema. This is where the mappings generated by our prototype and uploaded when creating the project are used by Quarry. Figure 18 shows the ETL flow corresponding to the MDS depicted in Figure 17.

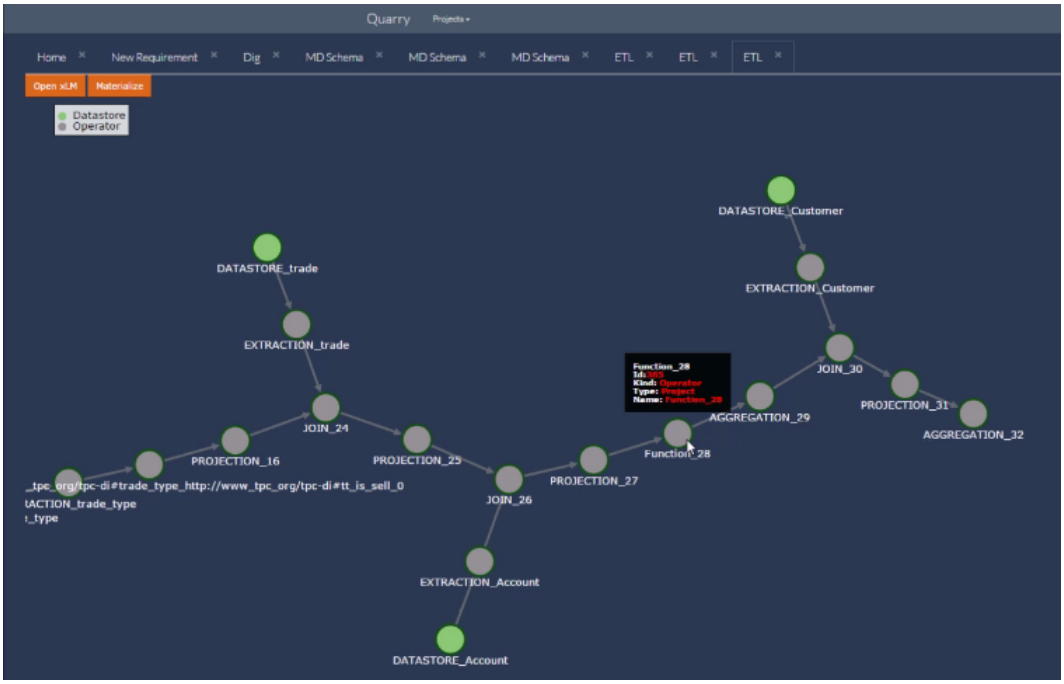


Fig. 18: ETL Flow in Quarry

Through this example, we have clarified how the outputs generated by our prototype, i.e. the output ontology and the mappings, can be used by Quarry. Quarry assumes that these artifacts are pre-defined and users previously needed to create them manually prior to using it.

Similarly, the output artifacts generated by our prototype can be consumed by any other data integration system. The output ontology is a standard OWL ontology that can be parsed by any OWL-compliant library. Moreover, the independent IO modules that were developed as part of the prototype can be used by third-party systems to interact with the generated mappings.