Automatic Semantic Enrichment of Data Services

Bruno C. N. Oliveira
bruno.cno@posgrad.ufsc.br
Department of Informatics and Statistics
Federal University of Santa Catarina
Florianópolis, Santa Catarina, Brazil

Alexis Huf
alexis.huf@posgrad.ufsc.br
Department of Informatics and Statistics
Federal University of Santa Catarina
Florianópolis, Santa Catarina, Brazil

Ivan Salvadori
ivan.salvadori@posgrad.ufsc.br
Department of Informatics and Statistics
Federal University of Santa Catarina
Florianópolis, Santa Catarina, Brazil

Frank Siqueira
frank.siqueira@ufsc.br
Department of Informatics and Statistics
Federal University of Santa Catarina
Florianópolis, Santa Catarina, Brazil

ABSTRACT

In recent years, many approaches and tools have emerged to assist in the semantic enrichment of Web services. Many researchers have been directing efforts in providing semantic annotations and enriching service descriptions. However, the provisioning of semantically enriched representations through data services has been little considered in the literature, despite of its significant impact on the effectiveness of data integration solutions. In this paper we present a novel method aimed at enriching data provided by services, enabling them to dynamically provide semantic representations. We introduce an architecture, called OntoGenesis, capable of i) constructing and evolving domain ontologies for data services and ii) aligning the properties of such ontologies with external data sources, so as to enhance the reuse of well-known concepts in order to facilitate further integrations. Experiments using real-world bilingual datasets show the applicability of our proposal and the obtained results reveal that alignments achieve satisfactory levels of precision and recall.

CCS CONCEPTS

• **Information systems** → **Web services**; *Web applications*; *Ontologies*; • **Applied computing** → Computing in government;

KEYWORDS

Web Services; Data Services; Semantic Web; Ontology Construction; Property Matching.

ACM Reference Format:
Bruno C. N. Oliveira, Alexis Huf, Ivan Salvadori, and Frank Siqueira. 2017. Automatic Semantic Enrichment of Data Services . In *iiWAS '17: The 19th International Conference on Information Integration and Web-based Applications & Services, December 4–6, 2017, Salzburg, Austria.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3151759.3151783

iiWAS '17, December 4–6, 2017, Salzburg, Austria

ACM ISBN 978-1-4503-5299-4/17/12…$15.00
https://doi.org/10.1145/3151759.3151783

1 INTRODUCTION

The number of Web services, scattered in public and private networks, that are meant to provide data already stored in some data source has been constantly increasing. Such Web services are also referred to as data services and are useful for providing access interfaces to data sources that cannot be completely disclosed [8]. Additionally, such services can also employ Semantic Web technologies [4] in order to provide data in a sophisticated machine-readable format and, therefore, be easily reused by third parties and integrated to complex Web applications. Several researchers have been discussing the benefits of employing Semantic Web services; for instance, enhance data interoperability [23] and assist in automating tasks such as service discovery, selection, composition, etc. [18, 26].

The implementation and adoption of Semantic data services in real-world applications, however, are limited mainly due to the format of stored data. Such data is usually stored in a syntactic form, i.e., only the structure of the data is specified, but not the semantics. In addition, various major challenges contribute to this lack of adoption. Some common issues include the time and effort demanded in the construction of domain ontologies and the semantic annotation of data services. These are complex tasks that require domain expert knowledge. Concerns over agreement in semantic modeling must also be taken into account. In practice, assuming that data provided by services will always be defined by a universal ontology is not realistic [12]. It brings a new challenge concerning Semantic data services development and integration, due to the existing heterogeneous ontologies describing the same real-world entity.

Since building an ontology for a data source is a difficult and time consuming task by its nature, some support tools [7, 9, 19, 22] have been developed to help users in the ontology construction process. However, these tools often require availability of data dumps for generating a domain ontology. This limitation hinders the adoption of such tools in SOA (Service-Oriented Architecture) applications, in which availability of data depends on the service interface. Extensional ontology matching techniques [11] attempt to solve the problem of ontology heterogeneity using instance data to infer equivalences at the schema level. On the other hand, data services are susceptible to changes, and ontologies that describe data are required to evolve in parallel, otherwise they become inconsistent. Moreover, in order to perform extensional matching, ontology

matchers generally assume that the two ontologies that are to be matched have already been created and associated with a large set of instances. This becomes a challenge when such data is partially available (due to the service interface) and when instance data of both ontologies are unrelated. Yao et al. [30] provide a mechanism to create a unified ontology based on a set of JSON documents provided by a Web service. Nevertheless, the generated ontology does not leverage semantic concepts defined in external sources, aiming to reuse existing concepts and minimize heterogeneity issues. In addition, previous works have proposed approaches to enrich services with semantics, most of them focusing on service descriptions [6, 14, 16]. In contrast, few proposals address the enrichment of data provided by services. For instance, Salvadori et al. [23] propose a method to enrich representations of data-based microservices with `owl:sameAs` and `rdfs:seeAlso` links. A framework aimed at identifying alignments between heterogeneous ontologies is also proposed. A drawback in this approach is that microservices must already employ a domain ontology and provide semantic data.

This work proposes an architecture, called OntoGenesis, aimed at generating domain ontologies and automatically enriching data services with semantic concepts defined in such ontologies. The benefits of our proposal are two-fold. Firstly, OntoGenesis provides a way to gradually build domain ontologies from syntactic representations provided by data services and to reuse well-known concepts by identifying data intersection with external sources. Secondly, the architecture enables the migration of syntactically defined data services toward Semantic data services. The evaluation of our proposal relies on open government data regarding police reports, published by the Public Security Secretariat of the state of São Paulo (SSP/SP)[1] in Brazil, as well as ontologies and instances of DBpedia[2] and GeoNames[3] datasets. Results show that our approach can achieve suitable F-Measure scores as well as better performance in comparison with other state-of-the-art ontology matchers.

The remainder of this paper is organized as follows: Section 2 summarizes the main concepts that are required for understanding this work. Section 3 presents an overview of the semantic enrichment approach for data service representations, while the OntoGenesis and its main components are detailed in Section 4. Section 5 presents the evaluation study and the obtained results. Related research efforts found in the literature are discussed in Section 6. Finally, Section 7 draws the conclusions and presents some perspectives for future work.

2 BACKGROUND

2.1 Data Services

To make data publicly available on the Web, data stores are often wrapped by Web services, which provide a Web interface for handling data. Such Web services hereafter will be called as data services or just services.

The main feature of data services is the manipulation of data in the sense that they act as data providers, allowing abstraction of access to data sources. Bianchini et al. [5] define a data service s as an operation, method or query to access data from a given data source. These services are modeled as a set of: i) service inputs s_i, which consist in parameters that are needed to invoke the service and access data; and ii) outputs s_o, representing data that is accessed through service s. Data access is usually resource (i.e., entity) oriented, that is, a consumer requests a resource to a service, passing s_i, and receives a representation of such resource, s_o. The output representation can be seen as a snapshot of the state of a resource at a given time, available in different formats, such as XML, JSON, HTML, etc. It is worth noting that data services are not tied to any particular technology; they can be implemented, for example, using SOAP or REST technology stacks.

A crucial factor that hinders service integration occurs in the conceptual level, since data services often employ different terminologies (for the attribute names, for instance), even though providing information about the same real-world concept. To overcome this issue, Semantic Web technologies [4] may be applied to data services, resulting in Semantic data services. This approach is aimed at providing machine-readable descriptions of data, therefore facilitating its integration and reuse. According to McIlraith et al. [18], Semantic Web services should expose information about available services, their properties, execution interfaces, pre- and post-conditions, in a sophisticated machine-readable format. Regarding Semantic data services, managed resources, as well as their properties and relationships, should also be semantically enriched. This means that representations provided by such services should be associated with semantic concepts.

On the other hand, Lira et al. [15] consider Semantic data services as access points to data that is natively stored as RDF (Resource Description Framework) triples in a particular data source. In contrast, we argue that Semantic data services provide semantic representations, regardless of how data is stored and maintained. Thus, enriching data service representations means to provide semantic data taking advantage of RDF formats (such as JSON-LD).

2.2 Ontology Construction and Matching

Ontologies play an important role in the Semantic Web, especially with regard to Web services. In general, ontology engineers and domain experts manually develop domain ontologies to provide a domain-specific model suitable for describing the semantics of a service. Most effort involved in semantically enriching services is, therefore, in the construction of ontologies as well as in adapting and evolving them in accordance with demanded changes. Ontology Learning (OL) [17] is a topic interested in automating and thereby facilitating the creation of ontologies by ontology engineers and domain experts. In this way, ontological elements, such as concepts and relations, are extracted from different resources. Some researchers, such as Alfaries [1], examine existing techniques and tools available for (semi-)automatically learning domain ontologies from Web service resources.

Although OL offers mechanisms to automate the ontology construction process, it is important to reuse ontological elements from existing ontologies. This allows further integrations and logic-based reasoning to be performed by software agents. In this sense, ontology matching techniques emerge to solve ontology heterogeneity issues, identifying equivalence relations – usually expressed by OWL equivalences – between different ontologies. Euzenat and

[1] http://www.ssp.sp.gov.br/transparenciassp/
[2] Datasets available at: http://wiki.dbpedia.org/Downloads2015-10
[3] Datasets available at: http://download.geonames.org/export/dump/

 DOI: 10.1145/3151759.3151783

Shvaiko [11] identify four basic techniques for ontology matching. The name-based technique considers only the name of ontology elements (e.g., labels of properties and classes). The structured-based technique considers the structure of ontology elements (e.g., subclass relations). Semantic-based techniques, on the other hand, usually leverage reasoner tasks so as to infer the equivalences between different ontologies. Finally, the extensional techniques use the ontology instances in order to identify similar individuals and thereby match classes and properties.

As data services provide information from data sources, extensional matching techniques can be adapted to SOA in such way that information can be added as instances of service ontology. Thus, besides the construction of domain ontologies for data services, this work is concerned with the extensional matching technique to find out alignments between service ontologies and external ontologies, focusing on their properties. Such alignments are often expressed by the `owl:equivalentProperty` axiom [2].

3 SEMANTICALLY ENRICHING DATA REPRESENTATIONS

This work focuses on dynamically enriching syntactic representations, provided by data services, with semantics. This goal is achieved by associating concepts defined in domain ontologies with representations provided by data services, in order to provide Linked Data. To this end, it is necessary to include a semantic adapter in the data service in order to perform such associations and create the new semantic representations. Figure 1 presents an overview of the workflow for the data service semantic enrichment. Firstly, a consumer sends a request to a data service. After processing the request, the service sends to an Enricher a syntactic representation of the response (serialized in XML or JSON, for example). The Enricher extracts all elements from such representation and constructs a domain ontology for the service, including classes, data type and object properties, as well as identified equivalent property links to external concepts.

The Enricher should output the domain ontology along with semantic associations, a.k.a semantic mappings, between the syntactic attributes from service representations and the new ontology concepts. A semantic mapping (SM) can be defined as a 3-tuple $\mathcal{SM} = \{a, c, t\}$, where a is the attribute of a representation, c is the concept represented in the ontology, and t is its type (a class or an object data/property). As an example, suppose that it is created a datatype property c, where c = `"http://service1-ontology#name"`, for the attribute a = `"name"` of a certain representation. The semantic mapping generated shall be $\mathcal{SM}$ = {`"name"`, `"http://service1-ontology#name"`, `"Datatype property"`}. SMs are useful for generating semantic representations for the data service. In this way, according to the set of semantic mappings yielded by the Enricher, the syntactic representation is automatically converted to JSON-LD by a semantic adapter, which in turn is sent back to the consumer.

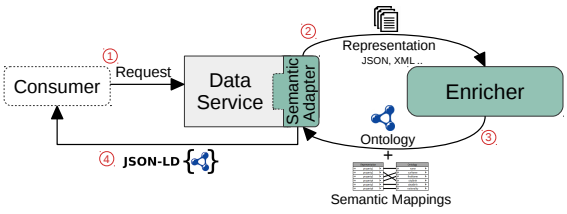

Figure 1: Schema of Semantic Enrichment of Data Service Representations.

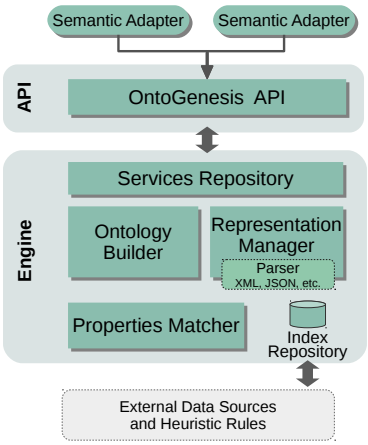

Figure 2: OntoGenesis Architecture.

As illustrated in Figure 1, the Semantic Adapter is a component attached to a data service, which intercepts the responses and returns JSON-LD representations to consumers. It can also be seen as a connector responsible for the communication between the data service and the Enricher. The next section presents a detailed discussion concerning the Semantic Adapter as well as the architecture designed for the enricher.

4 ONTOGENESIS

This section presents OntoGenesis, an architecture for semantically enriching data services representations. The OntoGenesis architecture is divided in three major components, as depicted in Figure 2. The first one, the OntoGenesis Engine, is responsible for constructing an ontology for the data service and for yielding semantic mappings in accordance with the syntactic attributes of the data service representations. The second component, called OntoGenesis API, is a Web API which provides a communication interface for accessing functionalities provided by the OntoGenesis Engine. Finally, the Semantic Adapter is a lightweight library for assisting data services in providing semantic representations by means of the OntoGenesis API. It is important to notice that both Engine and API components act as the Enricher depicted in Figure 1.

4.1 OntoGenesis Engine

As shown in Figure 2, the OntoGenesis Engine comprises five main components: *Services Repository*, *Representation Manager*, *Ontology Builder*, *Index Repository* and *Properties Matcher*.

The *Services Repository* manages information about the registered data services. The information stored includes the service name, its URI address, and the semantic features created by OntoGenesis (i.e, the domain ontology and the semantic mappings). Furthermore, this component assists in the operation of the other

components providing the necessary information about the data services registered in OntoGenesis.

The *Representation Manager* aims at extracting the elements from a data service representation, such as attributes and their values, useful for the ontology construction process. To this end, it provides a common abstraction to any data format, so that specific parsers can be seamlessly encapsulated in this component, allowing the OntoGenesis Engine to properly deal with different data formats, such as JSON, XML, CSV, among others.

Since the results of the Representation Manager are used by other components, format-specific details are discarded using the common abstraction. We employ an object-inspired abstraction, in which a representation consists of a set of objects, each consisting of a map from attribute names to a set of attribute values. As for attribute values, they are divided into object values (creating a tree structure) and primitive values[4] (numbers, strings and booleans).

The *Ontology Builder* analyzes the syntactic elements extracted in the Representation Manager to construct a domain ontology for the service producing the data. If a domain ontology has already been constructed from a previous representation sent by the service, the Ontology Builder updates the domain ontology with the new identified elements. Therefore, the ontology evolves as new representations are provided to OntoGenesis by the data service. Figure 3 depicts a sample of a primary OWL ontology in RDF/XML (b) based on a given JSON representation (a). The values contained in the JSON represent real data – published by the SSP/SP – of a police report with information regarding a person involved (e.g., a victim, witness or perpetrator).

The attribute names existing in the representation are mapped to a property instance in the ontology. The type of this property is determined as follows:

(1) `owl:ObjectProperty` if used exclusively with object values of a representation (e.g., lines 6 and 11 of Figure 3 (a) and (b) respectively);
(2) `owl:DatatypeProperty` if used exclusively with primitive values (e.g., lines 7 and 20 of Figure 3 (a) and (b) respectively);
(3) only `rdf:Property` otherwise.

The URI of the property instance is determined by a simple concatenation of the attribute name with the ontology prefix, which is defined in accordance with the service URI provided by the Services Repository component.

Classes are generated from two sources. First, any `owl:ObjectProperty` instance p originates a new class C (e.g., line 10 of Figure 3 (b)), as well as the triple $\langle p$ `rdfs:range` $C\rangle$ (line 13). Any property q extracted from object values of the attribute name corresponding to p will also take the triple $\langle q$ `rdfs:domain` $C\rangle$ (lines 20-31). The second source is the name of the endpoint from where the representation originated. Such class R generated in this manner will be the `rdfs:domain` of all properties that correspond to attribute names found in the root objects of the representations sharing the same endpoint name (e.g, line 15). The rationale for this is that, within a typical Web service, endpoints (e.g., a resource method in JAX-RS[5]) will serve entities

```
1  {
2    "PoliceReport": {
3       "reportID": "2015-10004-794",
4       "location": "Train Station ...",
5       ...
6       "personInvolved": {
7          "name": "CARLOS ALBERTO DOS SANTOS",
8          "docID": "015****18",
9          "birthDate": "12-21-1966",
10         "nationality": "Brazilian",
11         "placeOfBirth": "Sao Paulo-SP",
12         "gender" : "Male",
13          ...
14      }
15   }
16 }
```

(a)

```
1  <rdf:RDF
2     xmlns="http://example-service/ontology#"
3     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
4     xmlns:owl="http://www.w3.org/2002/07/owl#"
5     xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
6     xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
7     xml:base="http://example-service/ontology">
8    <owl:Ontology rdf:about="http://example-service/ontology"/>
9    <owl:Class rdf:ID="PoliceReport"/>
10   <owl:Class rdf:ID="PersonInvolved"/>
11   <owl:ObjectProperty rdf:ID="hasPersonInvolved">
12     <rdfs:domain rdf:resource="#PoliceReport"/>
13     <rdfs:range rdf:resource="#PersonInvolved"/>
14   </owl:ObjectProperty>
15   <owl:DatatypeProperty rdf:ID="reportID">
16     <rdfs:domain rdf:resource="#PoliceReport"/>
17     <rdfs:range rdf:resource="xsd:string"/>
18   </owl:DatatypeProperty>
19    ...
20   <owl:DatatypeProperty rdf:ID="name">
21     <rdfs:domain rdf:resource="#PersonInvolved"/>
22     <rdfs:range rdf:resource="xsd:string"/>
23   </owl:DatatypeProperty>
24   <owl:DatatypeProperty rdf:ID="docID">
25     <rdfs:domain rdf:resource="#PersonInvolved"/>
26     <rdfs:range rdf:resource="xsd:string"/>
27   </owl:DatatypeProperty>
28   <owl:DatatypeProperty rdf:ID="birthDate">
29     <rdfs:domain rdf:resource="#PersonInvolved"/>
30     <rdfs:range rdf:resource="xsd:date"/>
31   </owl:DatatypeProperty>
32    ...
33 </rdf:RDF>
```

(b)

Figure 3: Sample of: (a) JSON representation of a Police Report[6] and (b) Ontology built from such representation.

of the same type. Furthermore, such endpoint name can be inferred by a tool automating registry and representation submission to OntoGenesis.

In parallel with the ontology construction process, the Ontology Builder also yields the Semantic Mappings ($\mathcal{SM}$). For each ontology property p of a type t created in accordance with an attribute a of the service representation, it generates the 3-tuple $\mathcal{SM} = \{a, p, t\}$, useful for converting such representation to JSON-LD.

Despite the fact that the primary ontology produced by the Ontology Builder supplies semantic concepts related to the data provided by the service, such concepts are only known by the data service. In order to allow richer integration with other existing Semantic Web applications or services, it is necessary that the constructed ontology reuses (or aligns to) concepts defined by well-known and open ontologies/vocabularies. To this end, OntoGenesis aims at finding out equivalent concepts (specifically properties)

[4]JSON arrays are considered as multiple values for the same attribute name, i.e. the array ordering is discarded.
[5]https://jcp.org/en/jsr/detail?id=339

[6]Representation attributes were translated from Portuguese to ease understanding.

between the constructed ontology and external sources in order to expand the ontology and thereafter afford further reasoning tasks.

The *Index Repository* component stores, in a key-value database, indexes for literal data from external data sources and from the data services for which OntoGenesis will construct ontologies. There are two types of indexes. Data originated from services is kept in an index with the form $p \rightarrow o$, where p is an RDF predicate and o is the lexical form of an observed literal value. This index is not efficient for querying purposes, but allows efficient enumeration of o. Regarding the second type of index, used for external sources, a major requirement is fast processing of similarity-based queries (SBQ) in which, given a string w, a string s in the index with $d(w,s) \leq n$ is selected, where d is a text similarity function. In this paper, we consider the Levenshtein distance with a threshold of 1 ($d_L(w,s) \leq 1$) for the sake of simplicity.

To support querying the existence of s with $d_L(w,s) \leq n$, the second index is $p \rightarrow M$, where M is a DFA (Deterministic Finite Automata). The language of M ($\mathcal{L}(M)$) is the set of all tokens extracted from some object o of some triple in the external RDF sources. For such query, one simply needs to compute a Levenshtein automata $LEV_n(w)$ and verify if $\mathcal{L}(M) \bigcap \mathcal{L}(LEV_n(w)) \neq \emptyset$. Since the intersection automaton concurrently executes the intersected automata, such verification of empty intersection can be done directly without the need to materialize $LEV_1(w)$ [24].

Heuristic rules are also employed as sources of information for the alignment process. Such rules are patterns, that can be expressed as regular expressions, bound to a property p. One simple example of a rule $\mathcal{R}$ is `dbo:date` → `[0-9]{4}-[0-9]{2}-[0-9]{2}`. When the terms of a property p_1 (from a data service) match, for instance, with such $\mathcal{R}$, then p_1 can be considered as an equivalent property of `dbo:date`. Since the index for external sources consists of a DFA M, instead of a tree, heuristic rules can be merged directly into M, causing any string that matches a rule to be considered part of the index.

The *Properties Matcher* is the core component of OntoGenesis Engine and aims at properly matching the set of terms of each property provided by a data service, with the set of terms of each property existing in external data sources. The matching mechanism is based on the overlapping of data existing among properties in different datasets. In order to calculate the degree of overlapping data between two properties and find out more accurate equivalent properties, we define a strength value for each performed property matching. We provide an equation to compute the equivalent property strength for two properties p_1 and p_2 as follows:

$$Strength(p_1, p_2) = \frac{|\mathcal{V}p_1 \bigcap_s \mathcal{V}p_2|}{|\mathcal{V}p_1|} \quad (1)$$

where $\mathcal{V}p_1$ is the set of terms provided by a data service pertaining to a given property p_1, and $\mathcal{V}p_2$ is the set of values of a property p_2 provided by an external source, or recognized by any heuristic rule bound to p_2. The intersection $\bigcap_s$ uses a similarity-based algorithm to check whether a term $t_1 \in \mathcal{V}p_1$ is similar to a term $t_2 \in \mathcal{V}p_2$ and thereby can be considered as a co-occurrence. Besides Levenshtein, other similarity measures could be used, possibly requiring a change in the external sources index for efficient similarity-based query.

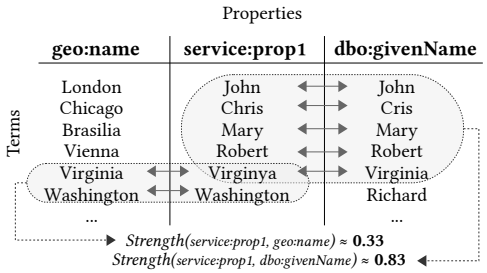

Figure 4: Example of overlapping property values.

Based on the overlapping strength, we identify the degree of correspondence between two properties, so the higher the overlapping strength, the most likely that the properties are equivalent.

Figure 4 illustrates the equation (1) in a scenario with three properties. `service:prop1` is a property from an ontology automatically constructed for a data service, while the other two properties come from external data sources. The rounded hatched rectangles represent $\mathcal{V}p_1 \bigcap_{d_L \leq 1} \mathcal{V}p_2$. Although `geo:name` has an intersection with the generated property, we can observe that the strength of this relation (0.33) is smaller than that between `service:prop1` and `dbo:givenName` (0.83). In view of that, we can define a threshold for the strength so that property pairs without significant support for their equivalence can be discarded.

Algorithm 1 presents the main steps for determining property equivalences from comparing property values provided by data services with external data sources. The procedure receives as input the ontology O constructed for the data service; the indexes $\mathcal{I}$ and $\mathcal{I}'$ containing, respectively, data from service representations and external sources; and the strength threshold represented by α. The algorithm starts matching all properties from the data services with properties obtained from external sources (lines 2-4). For each P_1, P_2 pair, the algorithm counts how many of the values enumerated in $\mathcal{V}p_1$ (line 3) are also present in $\mathcal{V}p_2$ (lines 5-10). This counting employs the Levenshtein automata imitation technique [24] to avoid the enumeration of $\mathcal{V}p_2$, which in addition to the values of $\mathcal{I}'$, includes the values that match possible heuristic rules bound to p_2. The strength is computed in line 11, in accordance to equation (1). In what follows, a triple stating that P_1 and P_2 are equivalent is created (l. 12) and it is added to the ontology if strength satisfies the threshold α, or removed from the ontology otherwise (lines 14-16).

In order to improve efficiency, the implementation of the algorithm in OntoGenesis Engine was done in such a way that the strength result is stored in memory and updated to each new set of values coming from the data service. Thus, it is not necessary to recalculate the intersection of the whole set, but only of the new terms received, to update the strength.

4.2 OntoGenesis API

The OntoGenesis API is a RESTful API meant to be accessible for all data services that need to be semantically enriched. This API exposes two main features: i) register the data service, and ii) invoke the OntoGenesis Engine to semantically enrich received representations from registered data services. It is worth noting that OntoGenesis API is an intermediate layer of communication

Algorithm 1 Properties Matching

```
1:  procedure MatchProperties(O, I, I', α)
2:      for each P1 ∈ GetProperties(I) do
3:          Vp1 ← GetValues(I, P1)
4:          for each P2 ∈ GetProperties(I') do
5:              Vp2 ← GetValues(I', P2)
6:              overlap ← 0
7:              for each v ∈ Vp1 do
8:                  if v ∈ Vp2 then
9:                      overlap ← overlap + 1
10:                 end if
11:             end for
12:             strength ← overlap / |Vp1|
13:             equivProp ← ⟨P1 owl:equivalentProperty P2⟩
14:             if strength ≥ α then
15:                 add equivProp to O
16:             else if equivProp ∈ O then
17:                 remove equivProp from O
18:             end if
19:         end for
20:     end for
21: end procedure
```

between services – through semantic adapters – and the main components of the OntoGenesis Engine.

When a given consumer interacts with a registered service, the latter sends the resource requested by the user to the OntoGenesis API for semantic enrichment. The OntoGenesis API invokes the OntoGenesis Engine and returns to the data service its new ontology, along with the semantic mappings. Therefore, legacy data services that provide purely syntactic representations are able to register with OntoGenesis and be dynamically enriched with semantic data.

The OntoGenesis API supports custom configurations, such as a threshold, for the equivalent properties strength, and a representation buffer size. The former must be configured with values from 0 to 1 and it is used during execution of the properties matching algorithm (Algorithm 1). The representation buffer size represents how many data service representations will be sent to the OntoGenesis Engine at the same time. When it is customized with a value greater than one, representations from each service are stored in a buffer before they are sent to the Engine. Thus, instead of forwarding representations one by one, it sends a batch of representations to the Engine in order to reduce the calls and therefore improve the performance of the semantic enrichment process.

Finally, OntoGenesis API offers an interface that allows loading new external data sources into the Index Repository. Accordingly, users can send new datasets to be loaded, so that their content will be considered at runtime by the Properties Matcher in forthcoming service requests.

4.3 Semantic Adapter

The Semantic Adapter is a component that is attached to a data service that wishes to provide semantically enriched representations. As shown in Figure 1, it is responsible for the interaction between the data service and the OntoGenesis API. In addition to being

```
{   "@context": {
       "PoliceReport": "http://example-service/ontology/PoliceReport",
       "reportID": "http://example-service/ontology/reportID",
       ...
       "personInvolved": "http://example-service/ontology/hasPersonInvolved",
       "name": "http://example-service/ontology/name",
       "docID": "http://example-service/ontology/docID",
       "birthDate": "http://example-service/ontology/birthDate",
       "nationality": "http://example-service/ontology/nationality",
       "placeOfBirth": "http://example-service/ontology/placeOfBirth",
       ...
    },
    "@id": "http://example-service/policeReport?reportID=2015-10004-794",
    "@type": "PoliceReport",
}
```

Figure 5: Excerpt of a JSON-LD context

imported into the data service implementation, its only required configuration is the URI of the OntoGenesis API.

By using this component, the service registration is performed automatically when the attached service starts. In addition, the Semantic Adapter intercepts all consumer requests coming to the data service and transparently invokes the OntoGenesis API, that delegates to the Engine the construction of a domain ontology and semantic mappings. Based on these artifacts, a new semantic representation serialized in JSON-LD is generated by the adapter and returned to the service clients. Therefore, instead of responding a syntactic representation, the data service is able to serve linked data to its consumers.

The Semantic Adapter also takes into consideration the `@id`, `@type` and `@context` syntax tokens [13] to create JSON-LD documents. The `@id` uniquely identifies the resource data that is being described. The `@type` specifies the subject of the JSON-LD document and the `@context` is used to map terms (usually a short word) that expands to an URI – defined in the ontology. These terms are kept the same as declared in the syntactic representation, so as to have little impact on the document syntax. A key benefit of adopting this feature is to allow both semantic-capable and syntactic consumers (i.e., those unable to process semantic data) to process the new outputs of the data service, since the attribute labels do not change. Figure 5 shows a sample of JSON-LD context mapping the JSON representation and ontology illustrated in Figure 3.

We provide a non-intrusive implementation of the Semantic Adapter for the JAX-RS specification. This implementation transparently diverts responses with representations generated by the data service to the OntoGenesis API, and replaces them with semantically enriched ones. Moreover, the Semantic Adapter provides the data service with an endpoint to share the constructed ontology in accordance with its URI base. Semantic applications can thereby retrieve this ontology and perform reasoning tasks.

5 EVALUATION

In this section we describe the scenario in which the evaluation was performed, the experimental methodology and the obtained results. The aim of this evaluation is to show the applicability of OntoGenesis and measure its performance using real world data. Therefore, we analyze the recall, precision and F-measure obtained by the proposed algorithm. In the end, we briefly describe a test performed with two ontology matchers in order to compare them with our properties matching approach.

Table 1: Experimental results for DBpedia subset size.

Subset Size	Total Time Mean with CI*	Highest F-Measures Mean
20%	103.84 ± 5.70 ms	0.7097
40%	121.25 ± 7.30 ms	0.7503
60%	154.86 ± 12.99 ms	0.6542

* CI = 95% Confidence Interval

Table 2: Experimental results with 95% Confidence Interval.

α^*	$\mathcal{SM}$ and Ontology Construction	Properties Matching	JSON-LD Generation
0.4	19.10 ± 1.17 ms	19.57 ± 2.48 ms	3.57 ± 0.32 ms
0.6	17.98 ± 0.97 ms	18.05 ± 1.42 ms	3.39 ± 0.26 ms
0.8	19.58 ± 1.08 ms	21.22 ± 1.75 ms	4.23 ± 0.35 ms

*α = Strength threshold

5.1 Scenario

The scenario is based on open government data published by SSP/SP, which discloses police reports in a non-structured format. Two categories of reports were selected: "intentional homicide" and "suspicious death", due to the large attention given to these types of crime/death. The police report provides information on those involved (victims and perpetrators), such as name, document number, birth date, nationality, place of birth, gender, among others. In spite of being a substantial initiative of information transparency to society, SSP/SP does not publish information in a suitable format to be integrated and reused by other data sources or even analyzed by researchers. In order to enrich this information, we have deployed a data service that provides non-semantic data of those who have been involved in any police report filed in the year of 2015.

As external data sources of OntoGenesis, we have used subsets of both DBpedia and Geonames. DBpedia is a huge cross-domain database that offers diverse Linked Data extracted from information boxes in Wikipedia pages. Geonames is a geographical database containing more than 160 million RDF triples available on the Web, and also comprises a vocabulary for representing information on places, such as latitude, longitude, name, etc. A pair of subsets of Geonames was utilized due to their potential for data intersection: Places in Brazil and Country names. In addition, we have selected the DBpedia-Person dataset, which provides more than 8 million triples representing attributes of a person, such as name, birth date, birth place, among others.

5.2 Methodology

The deployed data service provides OntoGenesis with JSON representations that may contain up to ten attributes of a person. Nine of these attributes have equivalent properties in external sources and only one has no correspondences. Therefore, this one must have a datatype property in the ontology, but with no equivalent property. It is worth mentioning that all representation attributes provided by the data service were labeled in Portuguese, literally as in the data source, whereas the properties of external sources remained in English.

We analyzed the behavior of our approach setting three different values for the threshold of equivalent properties strength: 0.4, 0.6 and 0.8. For experimental purposes, we have configured the representation size buffer to 1, that is, OntoGenesis will process each request at a time. When it receives a representation from the data service, OntoGenesis extracts all values, checks overlapping of terms with external sources and, finally, updates the data service ontology with new properties and equivalent properties, in accordance with the configured strength threshold.

The experiments were performed using random person data, with 7 rounds of 100 requests sent by the data service to OntoGenesis for each of the thresholds. All experiments were performed on an Intel i7 processor running at 2.5GHz, equipped with 8GiB of memory, using Oracle Java Development Kit (JDK) 8 and Apache Jena 3.3.0 for RDF and ontology processing. In order to allow the replication of experiments, source code and instructions for setting up the evaluation as well as all the results gathered in the experiments are available in a public repository[7].

5.3 Subset Size

Although the person subset of DBpedia was entirely loaded into the Index Repository, its huge size can significantly impact on memory consumption, since its dump file comprises more than 1GB. Aiming at finding out an appropriate subset size for evaluation purpose, we have performed a previous experiment using the 20%, 40% and 60% most frequent terms on DBpedia. Four execution rounds of 100 requests have been done for each size using the same configurations detailed in section 5.2 with the strength threshold set to 0.8.

Table 1 shows the average processing time observed by the data service for each request, along with their confidence intervals; and the mean of the highest F-Measures (within the 100 requests) computed for each execution. Results show that a larger subset size has little impact on processing times. Moreover, no significant difference was observed among the F-Measures computed for each subset size, varying nearly 0.1 or less. However, since more false positives were found in the largest subset, one inaccurate equivalent property was defined, which led to a slight decrease of F-Measure with a subset size of 60%. In view of that, and in order to maintain a suitable subset size and high F-Measure, 40% of the most frequent terms of DBpedia have been used to perform the following experiments.

5.4 Threshold Comparison

Table 2 presents the mean processing time for the semantic mappings ($\mathcal{SM}$) and ontology construction; properties matching process; and JSON-LD generation, along with their confidence interval for each request and considering the 7 rounds of execution. One can notice that there is no significant alteration in execution time between the adopted thresholds. Additionally, the processing time is independent of the number of requests, since previous requests are not reprocessed, but only the strength of equivalent properties is recalculated.

Figure 6 shows the average of precision and recall per request, considering the 7 rounds of execution and their confidence interval

[7] https://github.com/brunocnoliveira/iiwas2017-ontogenesis-experiments

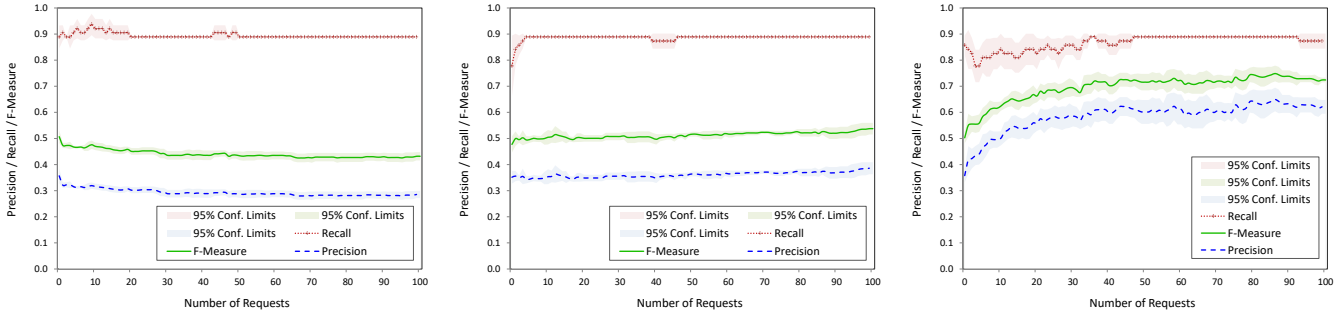

Figure 6: Precision, Recall and F-Measure curves for strength threshold (a) α = 0.4; (b) α = 0.6; and (c) α = 0.8.

in the shadowed area surrounding the lines. It also depicts the F-measure (or F_1 score), representing the harmonic mean of precision and recall. Figure 6 (a) shows that in the scenario with a strength threshold of 0.4, the recall is kept stable, reaching almost 0.9 after 20 requests, while the precision slightly decreases to 0.3. In the second scenario, with a threshold of 0.6 (Figure 6 (b)), the recall increases before executing 10 requests, while the precision slightly increases, reaching almost 0.4. On the other hand, in the scenario with a threshold of 0.8 (Figure 6 (c)), the precision reaches nearly 0.6, with recall and F-Measure of approximately 0.85 and 0.7, respectively.

In all scenarios, after some time, OntoGenesis was capable of enriching, with equivalent properties, 8 out of 9 properties that could be enriched (approx. 0.89 of recall). In addition, it has also found false positive equivalent properties for the same datatype property. The reason behind this is that there are few different properties sharing the same frequent terms. For instance, `pessoa:nome` (a property of the data service ontology that represents a person's name) has correspondences with `dbpedia:name`, `dbpedia:surname` and `dbpedia:givenName`. Nevertheless, we consider only `dbpedia:name` as the correct match. In the same way, `pessoa:naturalidade` (a property of the data service ontology related to the place of birth) has correspondences with `dbpedia:name` and `geo:name`, since there are several places in Brazil that are named after a person. These situations can be dealt with increasing the strength threshold. The higher the threshold, the higher is the precision. Nonetheless, we must pass a balanced value to maintain a suitable recall score, otherwise, the recall tends to decrease. Both recall and precision are also related to the overlapping of data existing between service representations and external sources. In the first requests in scenario with threshold $\alpha = 0.8$, we observed a slight decrease in the recall, since many property values of service did not match those of DBpedia nor Geonames, which leads to a strength $\leqslant \alpha$. Therefore, in scenarios with higher thresholds, the strength tends to balance as new values are passed to each request.

Overall, the evaluation results are promising and show that OntoGenesis can gradually enrich data services with semantics, optimizing the reuse of well-known defined concepts from external data sources. For instance, for life sciences data-services, it is possible to combine specific resources from this domain, such as DrugBank [29] and Bio2RDF [3]; or even use the ScholarlyData[8] dataset for academic services.

5.5 Other Ontology Matchers

Extensional-based techniques can be adapted for service enrichment [23] and, similarly to OntoGenesis, exploit instance data to find ontology alignments. They rely on individuals obtained from data services and from external sources to check if they have some intersecting properties, i.e., properties with matching identifiers or with values in the same domain. This correference or value sharing among properties is used to infer property and class equivalences among different ontologies.

Firstly, we converted our scenario from service semantic enrichment to ontology alignment. The goal is to align the ontology constructed by OntoGenesis (without alignments) against DBpedia ontology, FOAF and Geonames. To this end, two files are generated. The first contains the constructed ontology along with 1021 person instances described in this ontology. The second file contains external ontologies and all instances from the external datasets. We evaluated two extensional matchers, PARIS [25] and AROMA [10], executing 7 rounds in each test. Nevertheless, none were able to produce property alignments. The results were also not favorable performance-wise: on average, AROMA took almost 18 minutes, PARIS took 1 minute 45 seconds and OntoGenesis took only 39 seconds. Figure 7 shows the processing time for both extensional matchers in comparison with OntoGenesis.

The challenge to these otherwise successful ontology matchers is in the absence of correferent and data sharing between the service data and the external data. For instance, we observed that the only complete literal shared between the two service data and external data is "PRETA"[9]. Extending the comparison to Levenshtein with threshold 1, there are 2,324 (0.20%) subjects in the external data that share a single literal value with the service data. However, no such subject shares the value of more than one property with a counterpart in the service dataset, which becomes a challenge for extensional matchers that are based on the similarity of individuals.

[8] Available at: http://www.scholarlydata.org/

[9] Meaning black skin color in Portuguese and also a place in the GeoNames dataset (http://www.geonames.org/3391216)

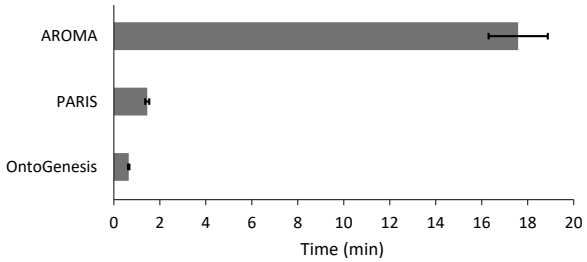

Figure 7: Processing Time Comparison.

6 RELATED WORK

The related work is divided into two major categories. The first category discusses researches on data-based ontology construction and ontology matching approaches. The second category concerns proposals related to the semantic enrichment of services.

Euzenat and Shvaiko [11] identify 4 groups of ontology matching approaches, each with specific challenges in the scenario of semantic enrichment of services. Name-based techniques require ontology elements to be labeled in the same language (which was not the case for the scenario in section 5) and to have elements with similar names. Semantic and structure-based techniques analyze ontology features, e.g., the class hierarchy and relations between classes. Thus, OntoGenesis constructs ontologies with a flat class hierarchy (without subclasses) and the only relations between classes occur through domains and ranges of properties. Finally, although extensional-based techniques can be adapted for service enrichment [23], such techniques present poor performance (as shown in section 5.5) when data obtained through the service interface is unrelated to data from the external sources and when individuals are not the same. In light of such evidence, we argue that extensional ontology matching algorithms present distinct characteristics from the problem we tackle, specifically property matches.

On the other hand, there are few efforts focusing on matching properties in ontologies. Tran et al. [28] introduce a cluster-based similarity aggregation methodology for ontology matching in which they rely on four different similarity measures to align object properties based on their domains and ranges. The authors, however, mention that the evaluation results are not strong enough to distinguish matching and non-matching property names. Zapilko et al. [31] identifies the exact relationship between two objects in large scale linked data, using governmental data. Their goal is to align instances of ontologies separately and to compute the overlap between them in order to improve the matching of object properties. Additionally, the overlap scores use some variations of the Jaccard coefficient. A drawback in the aforementioned approaches is that they are limited to object properties. On the other hand, Nunes et al. [20] use genetic algorithms for complex datatype properties matching. However, their approach also differs from ours in the sense that its goal is the matching of one to many complex relationships in different ontologies. In other words, the authors are interested in mapping properties that are composed of other properties (e.g., mapping "first name" and "last name" to "full name").

Several approaches [7, 9, 19, 22] have been designed aiming to support users in the ontology construction process. However, they all suffer from some shortcoming. First, most of them depend either on very specific or proprietary ontology models, which hinders its wide applicability. Second, they are not fully automatic approaches, since they all assist expert domain users. Finally, traditional ontology construction methods commonly require as input a huge set of unstructured text or Web page data [19], differently from our approach, in which data is provided on demand by services.

Yao et al. [30] introduce a framework for transforming semi-structured Web documents – specifically a set of JSON representations provided by Web services – into a unified ontology. Firstly, the framework parses the JSON and yields RDF triple sets. Meanwhile, multiple independent ontologies are created based on the triple sets and, afterwards, an ontology merging process is performed to achieve one unified ontology model. The resulting ontology must be validated by domain expert users, who can validate and edit the final ontology. Although it demonstrates to be a relevant study related to ontology construction based on JSON documents, the authors do not tackle the automatic semantic enrichment of Web services. Furthermore, the framework only considers JSON as input and outputs a single ontology, while our approach can support different representation formats and derive one domain ontology for each data service. Moreover, we propose a property matching technique in order to optimize the reuse of concepts defined in external resources, whilst they do not use any external source to enhance the constructed ontologies. The paper does not provide the data used in experiments, but based on the presented results we infer that their approach, in the best case, is able to process 2,038 triples/sec. In contrast, OntoGenesis processed approximately 96,300 triples/sec in the scenario described in section 5.

Many researchers have been directing efforts towards automatic/-semi-automatic semantic enrichment of Web services [27]. These works can be divided into two subgroups: service description and service representation enrichment. The first concentrates on adding semantic information to describe services interfaces, while the second aims at enriching service representations with semantic features. Little attention is given to the later group. Most of the proposals found in the literature are interested in enhancing service descriptions with semantics, such as SDWS [6], SWS Editor [14] and ASSARS [16]. In particular to data services, Quarteroni et al. [21] propose a semi-automatic service registration process exploiting existing knowledge bases as well as text processing techniques for semantic annotation and integration of Web data services.

Salvadori et al. [23] propose a composition method that exploits the potential data intersection observed in data-based microservice descriptions in order to create links between semantic resources. The representations provided by microservices are therefore enriched with `owl:sameAs` and `rdfs:seeAlso` links. Complementarily, the authors propose a framework called Alignator, which adopts ontology matching techniques to identify alignments between heterogeneous ontologies that describe microservices. However, this approach only considers services that i) already employ a previously defined domain ontology and ii) provide Linked Data representations as well as semantic descriptions. Therefore, syntactic representations are not supported by this framework.

7 CONCLUSIONS AND FUTURE WORK

In general, data services often provide their data in a (semi-) structured form, but without explicit semantics. Semantic Web technologies allow machine-readable descriptions of data, that ultimately improve reusability and interoperability of data services. This paper presented OntoGenesis, an architecture that aims at constructing domain ontologies for the data provided by data services. Additionally, OntoGenesis improves the constructed ontology through a property matching mechanism which reuses well-known concepts used in external sources. Finally, an adapter plugged into the service transparently enriches the output representation with semantic concepts defined in the domain ontology. Current approaches for semantic enrichment of service focus on generating semantic service descriptions and do not cover automatic semantic enrichment of the data provided by a service. Moreover, approaches for similar problems, such as ontology construction and ontology alignment, in general, do not apply to the scenario where data instance information is provided only through a service interface.

Results show the equivalence strength between properties of the constructed ontology and the external data sources. The precision is tightly related to the strength threshold, so better F-Measure scores are achieved when the threshold is increased (e.g., a threshold of 0.8 yielded around 0.7 of F-measure). In some cases, however, we observed that similar values may also lead to wrong matches. For instance, a person's surname could match with a place name.

Comparing with extensional ontology matchers, OntoGenesis presents better time performance (between 2 and 18 times faster) and is able to find equivalences in scenarios where such aligners would not, even if given a dump of the service data. In comparison with an ontology construction approach [30], OntoGenesis was 47 times faster in terms of triples/sec. Overall, the experiments show that our proposal is a promising approach that can boost the provisioning of richer data by service providers, reducing the efforts involved in the whole process of developing semantic services.

In future work we intend to consider, in the strength equation, the frequency of each term in the data service and in the external data sources so as to minimize false positives. We are also working on a mechanism to identify class equivalence, in addition to the equivalent properties. Besides JSON-LD, a mechanism to provide XML semantically annotated could also be included for compatibility with XML consumers. Furthermore, machine learning techniques can be applied to recognize patterns of property values in order to dynamically yield new heuristic rules.

REFERENCES

[1] Auhood Alfaries. 2010. *Ontology Learning for Semantic Web Services.* Ph.D. Dissertation. Brunel University London. http://dspace.brunel.ac.uk/handle/2438/4667

[2] Sean Bechhofer, Frank Van Harmelen, Jim Hendler, Ian Horrocks, Deborah L McGuinness, Peter F Patel-Schneider, and Lynn Andrea Stein. 2004. OWL Web Ontology Language Reference. *W3C Recommendation* 10 (2004). http://www.w3.org/TR/owl-ref/

[3] François Belleau, Nicole Tourigny, Benjamin Good, and Jean Morissette. 2008. *Bio2RDF : A Semantic Web Atlas of Post Genomic Knowledge about Human and Mouse.* Springer Berlin Heidelberg, Berlin, Heidelberg, 153–160.

[4] Tim Berners-Lee, James Hendler, and Ora Lassila. 2001. The Semantic Web. *Scientific American* 284, 5 (2001), 34–43.

[5] Devis Bianchini, Valeria De Antonellis, and Michele Melchiori. 2015. Developers' Networks Contribution to Web Application Design. In *Proceedings of the 17th International Conference on Information Integration and Web-based Applications & Services (iiWAS '15).* ACM, New York, NY, USA, 55:1–55:10.

[6] Maricela Bravo, José Rodríguez, and Jorge Pascual. 2014. SDWS: Semantic Description of Web Services. *International Journal of Web Services Research* 11, 2 (2014), 1–23.

[7] P. Buitelaar, D. Olejnik, and M. Sintek. 2004. A Protégé Plug-in for Ontology Extraction from Text Based on Linguistic Analysis. In *Proceedings of the 1st European Semantic Web Symposium.* Springer, Berlin, Germany, 31–44.

[8] Michael J Carey, Nicola Onose, and Michalis Petropoulos. 2012. Data Services. *Communications of the ACM* 55, 6 (jun 2012), 86.

[9] Philipp Cimiano and Johanna Völker. 2005. Text2Onto: A Framework for Ontology Learning and Data-driven Change Discovery. In *Proceedings of the 10th International Conference on Natural Language Processing and Information Systems (NLDB'05).* Springer-Verlag, Berlin, Heidelberg, 227–238.

[10] Jérôme David. 2007. Association Rule Ontology Matching Approach. *Int. Journal on Semantic Web and Information Systems* 3, 2 (2007), 27–49.

[11] Jérôme Euzenat and Pavel Shvaiko. 2007. *Ontology Matching.* Springer-Verlag New York, Inc., Secaucus, NJ, USA.

[12] Aissa Fellah, Mimoun Malki, and Atilla Elçi. 2016. Web Services Matchmaking Based on a Partial Ontology Alignment. *Int. Journal of Information Technology and Computer Science (IJITCS)* 8, 6 (June 2016), 9–20.

[13] Markus Lanthaler, Manu Sporny, and Gregg Kellogg. 2014. *JSON-LD 1.0.* W3C Recommendation. W3C. http://www.w3.org/TR/2014/REC-json-ld-20140116/.

[14] Cleber Lira and Paulo Caetano. 2016. REST-Based Semantic Annotation of Web Services. In *Information Technology: New Generations*, Vol. 448. Springer, 269–279.

[15] Hermano Albuquerque Lira, Jose Renato Villela Dantas, Bruno de Azevedo Muniz, Laura Maria Chaves, and Pedro Porfirio Muniz Farias. 2014. Semantic Data Services: An approach to access and manipulate Linked Data. In *XL Latin American Computing Conference (CLEI).* IEEE, 1–12.

[16] Chengduo C.a Luo, Zibin c Zheng, Xiaorui X.d Wu, F.d Fei Yang, and Yao Y.a Zhao. 2016. Automated structural semantic annotation for RESTful services. *International Journal of Web and Grid Services* 12, 1 (2016), 26–41.

[17] Alexander Maedche and Steffen Staab. 2001. Ontology Learning for the Semantic Web. *IEEE Intelligent Systems* 16, 2 (March 2001), 8.

[18] Sheila A. McIlraith, Tran Cao Son, and Honglei Zeng. 2001. Semantic Web Services. *IEEE Intelligent Systems* 16, 2 (2001), 46–53.

[19] Thi Thanh Sang Nguyen and Haiyan Lu. 2016. Domain Ontology Construction Using Web Usage Data. In *AI 2016: Advances in Artificial Intelligence.* Springer International Publishing, 338–344.

[20] Bernardo Pereira Nunes, Alexander Mera, Marco Antônio Casanova, Besnik Fetahu, Luiz André P. Paes Leme, and Stefan Dietze. 2013. *Complex Matching of RDF Datatype Properties.* Springer Berlin Heidelberg, Berlin, Heidelberg, 195–208.

[21] Silvia Quarteroni, Marco Brambilla, and Stefano Ceri. 2013. A bottom-up, knowledge-aware approach to integrating and querying web data services. *ACM Transactions on the Web* 7, 4 (2013), 19:1–19:33.

[22] Sara Salem and Samir AbdelRahman. 2010. A Multiple-domain Ontology Builder. In *Proceedings of the 23rd International Conference on Computational Linguistics.* Association for Computational Linguistics, Stroudsburg, USA, 967–975.

[23] Ivan Luiz Salvadori, Alexis Huf, Bruno C. N. Oliveira, Ronaldo Santos Mello, and Frank Siqueira. 2017. Improving Entity Linking with Ontology Alignment for Semantic Microservices Composition. *International Journal of Web Information Systems* 13 (2017). Issue 3.

[24] Klaus U Schulz and Stoyan Mihov. 2002. Fast string correction with Levenshtein automata. *Int. Journal on Document Analysis and Recognition* 5, 1 (2002), 67–85.

[25] Fabian M Suchanek, Serge Abiteboul, and Pierre Senellart. 2011. Paris: Probabilistic alignment of relations, instances, and schema. *Proceedings of the VLDB Endowment* 5, 3 (2011), 157–168.

[26] Katia Sycara, Massimo Paolucci, Anupriya Ankolekar, and Naveen Srinivasan. 2003. Automated Discovery, Interaction and Composition of Semantic Web Services. *Web Semantics: Science, Services and Agents on the World Wide Web* 1, 1 (2003), 27.

[27] Davide Tosi and Sandro Morasca. 2015. Supporting the semi-automatic semantic annotation of web services: A systematic literature review. *Information and Software Technology* 61 (may 2015), 16–32.

[28] Quang-Vinh Tran, Ryutaro Ichise, and Bao-Quoc Ho. 2011. Cluster-based Similarity Aggregation for Ontology Matching. In *Proceedings of the 6th International Conference on Ontology Matching - Volume 814 (OM'11).* CEUR-WS.org, Aachen, Germany, 142–147.

[29] David S. Wishart, Craig Knox, An Chi Guo, Dean Cheng, Savita Shrivastava, Dan Tzur, Bijaya Gautam, and Murtaza Hassanali. 2008. DrugBank: A knowledge base for drugs, drug actions and drug targets. *Nucleic Acids Research* 36 (2008), D901–D906.

[30] Yuangang Yao, Hui Liu, Jin Yi, Haiqiang Chen, Xianghui Zhao, and Xiaoyu Ma. 2014. An automatic semantic extraction method for web data interchange. *2014 6th Int. Conf. on Computer Science and Information Technology* (2014), 148–152.

[31] Benjamin Zapilko and Brigitte Mathiak. 2014. *Object Property Matching Utilizing the Overlap between Imported Ontologies.* Springer International Publishing, 737–751.