Improving Entity Linking with Ontology Alignment for Semantic Microservices Composition

Ivan Salvadori[1], Alexis Huf[1], Bruno C. N. Oliveira[1], Ronaldo dos Santos Mello[1,2], and Frank Siqueira[1]

[1]**Graduate Program in Computer Science (PPGCC), Department of Informatics and Statistics (INE), Federal University of Santa Catarina (UFSC), Florianópolis/SC, Brazil**
[2]**P.P. in Methods and Management in Evaluation (PPGMGA)**

Corresponding author: Ivan Salvadori[1]
Email address: ivanlsalvadori@gmail.com

DOI: 10.1108/IJWIS-04-2017-0029
Published in: International Journal of Web Information Systems Volume 13, Issue 3 pp. 302-323, ISSN: 1744-0084

ABSTRACT

Purpose – This paper proposes a method based on Linked Data and Semantic Web principles for composing microservices through data integration. Two frameworks that provide support for the proposed composition method are also described in this paper: Linkedator, which is responsible for connecting entities managed by microservices; and Alignator, which aligns semantic concepts defined by heterogeneous ontologies.

Design/methodology/approach – The proposed method is based on entity linking principles and uses individual matching techniques considering a formal notion of identity. This method imposes two major constraints that must be taken into account by its implementation: architectural constraints and resource design constraints.

Findings – Experiments were performed in a real-world scenario, using public government data. The obtained results show the effectiveness of the proposed methodand that it leverages the independence of development and composability of microservices. Thereby, the data provided by microservices that adopt heterogeneous ontologies can now be linked together.

Research limitations/implications – This work only considers microservices designed as data providers. Microservices designed to execute functionalities in a given application domain are out of the scope of this work.

Originality/value – The proposed composition method exploits the potential data intersection observed in resource-oriented microservice descriptions, providing a navigable view of data provided by a set of interrelated microservices. Furthermore, this study explores the applicability of ontology alignments for composing microservices.

Keywords – Microservices, Linked Data, Service Composition, Ontology Alignment, Semantic Web

Paper type – Research paper

DOI: 10.1108/IJWIS-04-2017-0029

Published in: International Journal of Web Information Systems Volume 13, Issue 3 pp. 302-323, ISSN: 1744-0084

1 INTRODUCTION

The adoption of the microservices architecture for software development is in continued expansion, resulting in the replacement of traditional monolithic applications by collections of highly cohesive and loosely coupled services. Microservices are designed to provide solutions in a well-defined domain, resulting in multiple components that communicate and operate together. The use of microservices facilitates the deployment and maintenance processes, and also makes easier to reach resiliency and scalability requirements. However, it results in a more complex ecosystem, which requires additional communication and cooperation efforts.

Several service composition approaches can be found in the literature. However, few proposals primarily address the microservices architecture. Furthermore, there are few entity-based proposals, i.e., the majority of research works addresses the composition problem assuming action-based implementations. With regard to entity-based implementations, identifying and connecting entities managed by different data providers can be seen as a microservice composition problem. Several proposals that directly address the problem of connecting together different individuals provided by heterogeneous data sources can be found in the literature. Solutions based on SPARQL are described by Araujo et al. (2011), Casanova et al. (2014) and by Magalhães et al. (2013), whilst platform-independent proposals are described by Hu and Jia (2015) and by Stoermer et al. (2010).

This work presents a composition method for semantic microservices that handle different datasets with heterogeneous ontologies. In this method, microservices are modeled as entity providers, in which a given microservice is responsible for managing individuals aligned with a predefined concept in a domain ontology. The proposed method makes use of data linking techniques to find and connect individuals that represent the same real world object, but are managed by different microservices. This work also provides a development framework, named Linkedator, which aims to facilitate the adoption of the proposed composition method.

In general, it is not realistic to assume that data provided by services will always be defined by a single ontology (Fellah et al., 2016), specially when they are developed by independent teams to fit distinct application (sub-)domains. The problem of distinct ontologies can be solved with an alignment between the classes and properties of the heterogeneous ontologies (Pavel and Euzenat, 2013).

This paper provides a complete view of composing heterogeneous microservices in cross-domain environments by extending our previous work (Salvadori et al., 2016), where our proposal was limited to compose microservice relying on the same ontology. In addition, we propose an alignment framework, named Alignator, which aims at identifying alignments of heterogeneous ontologies used to describe information managed by microservices. Furthermore, this work provides a case study along with some evaluations under real-world cross-domain scenarios with public government datasets.

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 the proposed composition method for semantic microservices. The Linkedator framework is detailed in Section 4, while the Alignator framework is presented in Section 5. Section 6 describes related research efforts found in the literature. An evaluation study is presented in Section 7. Finally, conclusions are drawn in Section 8, along with some perspectives for future work on this research subject.

2 BACKGROUND

2.1 Semantic Web

The Semantic Web (Berners-Lee et al., 2001) is an extension of the current World Wide Web, in which data available on the Web is semantically described. The Semantic Web allows not only humans, but also machines to infer the meaning of data published on the Web, and also facilitates data integration and reuse. It is a natural evolution from a Web based on hypertext, targeted at human beings, toward a *Web of Data*, which can be interpreted by machines due to the association of published content with their semantic descriptions. The combination of Web content with semantic descriptions creates an interconnected information structure known as Linked Data (Bizer et al., 2008).

Four principles govern the publishing of Linked Data (Heath and Bizer, 2011). First, Universal Resource Identifiers (URIs) must be used to identify not only resources such as web pages, but also real objects and abstract concepts. Second, these URIs should be dereferenceable, i.e., allow retrieving information about the referenced resource. Third, a single standardized data model should be used: the Resource Description Framework (RDF). Fourth, links should be defined among resources. In RDF, such links are represented by *subject-predicate-object* triples, where the semantics of the link between *subject* and *object* is specified by a *predicate*, in this case known as an object property. These principles allow navigation in the Web of Data, as well as interpretation of data by machines.

Another type of RDF predicate are data properties, which represent resource attributes as literal values. These properties do not specify links, but serve to describe resources. Examples of data properties are the name or the height of a person. In relational or NoSQL databases, attributes are used to identify table rows, documents or nodes. This use allows relating records x and y by referring to the primary key of y as an attribute of x. The equivalent pattern using data properties does not allow navigation in the Web of Data using RDF, as there is no link explicitly represented. Both object and data properties can be defined using the Web Ontology Language (OWL), which allows to perform reasoning and logical decisions.

2.2 Microservices

According to Newman (2015), microservices are small and independent services. However, it is not defined in the literature how small and independent a microservice should be. Microservices are focused on meeting quality of software requirements present in a well-defined domain, which may be divided into multiple bounded contexts. Developing microservices that implement a single bounded context implies on keeping together things that change for the same reason and separating things that change for different reasons, therefore they can be implemented and deployed independently. Once microservices are relatively independent from one another, each one can be developed using the most suitable technology for its given purpose. By splitting up a monolithic application into several microservices, deployment and maintenance are facilitated. It also makes easier to reach resiliency and scalability requirements, since there is no central point of failure and it is possible to scale only the more demanded microservices.

Microservices may be implemented by using Web technology such as Web Services and Web APIs or by using a message broker that supports message queues and allows communication through publisher/-subscriber mechanisms. Richards (2015) classifies as *functional* microservices that implement functional domain requirements, and as *infrastructural* the ones that implement non-functional requirements, such as authentication, monitoring, logging, among others.

Microservices that are developed following the Web Services approach can employ Semantic Web Service technology, resulting in semantic microservices. According to McIlraith et al. (2001) semantic Web Services should expose information about available services, their properties, execution interfaces,

pre- and post-conditions in a machine-readable format. For Web APIs, managed resources, their properties and relationships should be described. To achieve this, Web API descriptions must be enriched by adding a semantic layer, which facilitates the automation process of service discovery, selection and invocation (Islam et al., 2010; Martin-Flatin and Löwe, 2007; McIlraith et al., 2001). Battle and Benson (2008) advocate the adoption of standard semantic technology, such as RDF and endpoint SPARQL, by semantic Web Services.

2.3 Data Linking Principles

Data linking is the task of finding equivalent resources that represent the same real world object (Ferrara et al., 2011). Data linking can be formalized as an operation that takes collections of data as input and produces a set of binary relations between their entities as output. The problem of data linking can be categorized into two main groups: connection of data from heterogeneous sources; and comparison of data for data cleaning, duplicate detection or merge/purge records.

A key requirement to properly produce link relations between entities is to determine the meaning of the match. Usually, it is intended to link together entities that could be considered the same real world object, often expressed using the *owl:sameAs* property. However, the notion of identity can be interpreted among three different meanings: ontological identity, logical identity and formal identity (Ferrara et al., 2011). In the first notion, two different entities with different object descriptions are identified as the same real world object. In the logical identity, two different entities represent the same object when they replace each other in a logical expression without changing the meaning of the expression. Finally, the formal identity is used in cases where each entity of the data source can be uniquely identified by a standard property, such as ISBN for books, DOI for academic papers, email for user accounts, etc.

The problem of data linking is similar to database record linkage and also ontology schema matching, both widely explored in the literature (Elmagarmid et al., 2007; Euzenat and Shvaiko, 2007; Köpcke and Rahm, 2010). Data linking makes use of techniques from these areas, which can be divided into three main categories: value matching, individual matching and dataset matching. The value matching technique applies to linking entities that contain the same property value expressed in different ways. The individual matching technique is used for deciding whether two entities correspond to the same real world object by analyzing their property values. Dataset matching takes into account all entities from two different data sources in order to create an optimal alignment between them.

2.4 Ontology Alignment

Despite best practices for Linked Data representation including the reuse of existing ontologies, there are several similar ontologies that address the same domain. The OWL vocabulary, since its inception, includes predicates to denote equivalence of individuals, classes and properties. However, the OWL equivalence vocabulary is not always used by ontology designers. This becomes a problem when software agents are confronted with data described by a different ontology, and is unable to use it because its semantics may not be understood by the agent even after fetching the ontology. The automatic detection of equivalence relations between heterogeneous ontologies, known as ontology matching (Euzenat and Shvaiko, 2007), aims to solve this problem and is an active research topic.

The matching operation yields an alignment A_1 between two ontologies O_1 and O_2 (Pavel and Euzenat, 2013). To construct the alignment, the following inputs may be used in addition to the ontologies: i) a known alignment A_0, ii) matching parameters (such as weights or thresholds), and iii) external resources. The alignment contains a set of correspondences between classes and properties of such ontologies. Each correspondence denotes a relation of equivalence, generalization or disjointness between two elements of

O_1 and O_2 (Pavel and Euzenat, 2013). Generally, these correspondences are determined by means of a measure of similarity among the elements of ontologies. Euzenat and Shvaiko (2007) propose methods for assessing this similarity based on specific features of elements. The basic techniques are classified as follows:

- Named-based: considers their inputs as strings. This method seeks for similar elements (classes, individuals, relations) based on their names, labels or comments.
- Structure-based: instead of comparing only the names of elements, the structure of the ontologies is also compared, i.e, their taxonomic structure.
- Extensional: analyzes instances of classes. If classes of two ontologies share individuals, the method most likely performs a correct match for these classes.
- Semantic-based technique: is a deductive method that uses the model-theoretic semantics. It often uses a reasoner in order to infer the correspondences.

3 COMPOSITION METHOD

This work proposes a composition method for semantic microservices based on data linking principles. In this sense, microservices work only as entity providers. The proposed composition method exploits the potential data intersection observed in resource-oriented microservice descriptions to create semantic links between resources and therefore provide a navigable view of the whole microservice architecture. A plain microservice architecture requires that either microservices generate links to other microservices, which is a form of coupling, or that the microservices are designed in such way that clients are not required to follow links from one microservice to another.

The proposed composition method aims to create links that correspond to object properties in a domain ontology. It uses individual matching techniques considering a formal notion of identity as defined by Ferrara et al. (2011). It takes as input a set of microservice descriptions, a domain ontology and a representation of the resource that is meant to be enriched with links. Resources are abstractions that represent information handled by microservices. Resources are not directly accessed; instead, they are seen through a representation, which is a snapshot of the state of a resource at a given time. Representations may be available in different formats, such as XML, JSON, HTML, etc. Although the general data linking output results in a collection of mappings between two entities from two data sources, the proposed method is meant to append such links directly to a given representation.

3.1 Microservices Architectural Constraints

The composition method assumes a Web-based microservice architecture, in which each microservice handles a set of resource classes. These resource-oriented microservices must follow three architectural constraints for the composition method to be applied.

The first constraint requires that each microservice provide ways to access their managed resources given some identifying information regarding the resource. Identifying information is widely present in real world resources and it may be necessary even in APIs that fully adopt REST architectural style constraints (Fielding, 2000) due to the need to interface with legacy systems. Examples of person identifying attributes are passport number, social security number, user login, e-mail address, etc.

With regard to the second constraint, microservices must semantically describe managed resources. It is also required to describe how to access entities from identifying data. Finally, these descriptions must be accessible to other components of the microservice architecture.

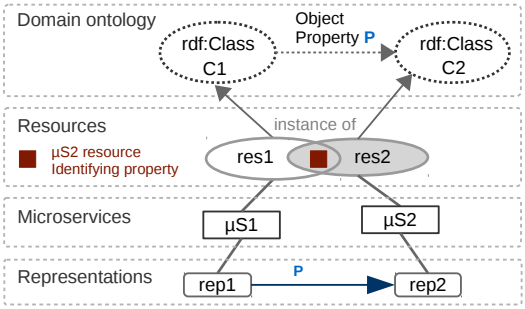

Figure 1. Data linking-based composition method

The third and final constraint is that the representations provided by microservices must allow the inclusion of hyperlinks. However, the microservices themselves are not required to include links in their representations. As a result of this constraint, consumers are able to distinguish between links and literal information.

It is important to notice that the proposed method does not require that microservices follow the REST principles. However, the composition method only creates and appends links to representations, and if a microservice violates REST constraints, these violations are not shadowed by the proposed method, but are exposed to consumers. Furthermore, only microservices capable of dealing with semantically enriched resources and able to provide means to access them are considered semantic microservices.

3.2 Resource Design Constraints

The major resource design constraint is that the resources managed by microservices must be instances of classes in ontologies, and each microservice can only use classes and properties from a single ontology. Furthermore, data attributes of resources must correspond to data properties of the ontology. Likewise, links among representations correspond to OWL object properties. These correspondences are not required for any purpose other than creating links, and are therefore not enforced. Resources may contain data that is not present in the ontology and some object properties might never originate links.

An overview of the composition method is shown in 1. In this example the ontology contains two classes ($C1$, $C2$) and an object property P that has $C1$ as its domain and $C2$ as its range. $C1$ and $C2$ are managed, respectively, by microservices μS1 and μS2. If a particular individual of $C1$, $res1$ is related to an individual of $C2$, $res2$, through property P, μS1 may not store a link to $res2$, but only its identifying data. From data that is actually managed by μS1, it is possible to properly represent the relation between $res1$ and $res2$ through a link in representation $rep1$ referring to representation $rep2$.

In order to avoid coupling between both microservices, μS1 does not store links. Instead, representations are enriched with links by a separate component to satisfy consumers. Thus, μS1 must associate internally the identifying data of $res2$ with P to properly model that $res1$ is related to $res2$ through P. In fact, we model $res1$ to contain, in this case, a blank node as a value of the property P, and then place identifying data of $res2$ in that blank node. This blank node in $res1$ represents $res2$ without linking to $res2$. The composition method identifies this, and automatically adds a link in the blank node to $res2$ to identify that both nodes are in fact the same node.

4 THE LINKEDATOR FRAMEWORK

This section presents Linkedator, a framework for composing semantic microservices in agreement with the composition method described in section 3. Linkedator is divided into 3 components: Core[1], API[2] and Jersey[3]. The first component is responsible for creating links. The second one encapsulates the core functionalities into a Web API. Finally, the third component is a development tool for implementing microservices by using the reference Java technological stack for RESTful Web Services.

4.1 Linkedator-Core

Linkedator-Core is the main component of the framework, responsible for creating links in JSON-LD (Lanthaler and Gütl, 2012) representations. JSON-LD is a representation format based on JSON, which provides support for linked data. The core component is composed by three modules: a service repository, an ontology model and a link engine, as shown by Figure 2. The service repository holds the semantic microservice descriptions, which should describe all the necessary details to interact with their resources. Hence, participating microservices must have their descriptions registered in this module.

The ontology model holds information about semantic classes and properties of resources managed by participating microservices. The link engine module is responsible for analyzing the ontology model, which comprises identifying object properties and creating links between entities provided by registered microservices.

There are two different methods for creating links: direct and inverse. In the direct method, a resource that is about to be linked has blank nodes containing shared information with other resources managed by different microservices. In the inverse method, this representation does not contain such information. Nevertheless, the link engine is capable of creating links to other representations based on the object properties defined in the ontology model.

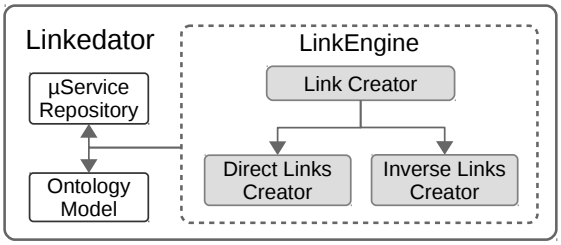

Figure 2. Architecture of Linkedator-Core

Algorithm 1 shows the necessary steps to create direct links and to append them to a given representation. First, the object properties of the informed representation are identified, resulting in an array used to select suitable classes managed by registered microservices (lines 3-6). The selected classes must match with the range of the identified object properties. The next step (line 7) is responsible for finding a suitable URI template that can be filled with data on the representation to identify a resource compatible with the property range. The selected URI template represents a link with variables that could be used to access representations. In line 8, the variables of the selected URI template are replaced with the informed representation data. Finally, the resulting link is associated with the property *owl:sameAs* and appended

[1] https://github.com/ivansalvadori/linkedator
[2] https://github.com/ivansalvadori/linkedator-api
[3] https://github.com/ivansalvadori/linkedator-jersey

Algorithm 1 Direct Links Creation Algorithm

```
1:  procedure CreateDirectLinks
2:      rep = informed representation
3:      objProp = OntologyModel.repObjProp(rep)
4:      for each p ∈ objProp do
5:          classes = μServiceRepo.classes(p.range)
6:          for each c ∈ classes do
7:              t = findUriTemplate(c, rep.objProp)
8:              link = resolveTemplate(c, t, rep)
9:              rep.p.append("owl:sameAs:", link)
10:             rep.append("@type:", e.URI)
11:         end for
12:     end for
13: end procedure
```

Algorithm 2 Inverse Links Creation Algorithm

```
1:  procedure CreateInverseLinks
2:      rep = informed representation
3:      objProp = OntologyModel.objPropByDomain(rep)
4:      for each p ∈ objProp do
5:          classes = μServiceRepo.classes(p.range)
6:          for each c ∈ classes do
7:              t = findUriTemplate(c, rep)
8:              link = resolveTemplate(c, t, rep)
9:              newElement.append("owl:sameAs:", link)
10:             newElement.append("@type:", e.URI)
11:             rep.append(p.uri, newElement)
12:         end for
13:     end for
14: end procedure
```

to the informed representation (line 9). The property *@type* is also appended to the representation, which defines the semantic class of the representation (line 10).

Algorithm 2 describes the necessary steps to create inverse links and to append them to a representation. In this process, object properties that have domain in the informed representation class are selected, which means selecting all object properties that could be part of the informed representation. As the informed representation does not contain the intersection data, it is necessary to create new elements to represent the referenced object. Within these new elements, which represent blank nodes, both the resolved link associated with *owl:sameAs* and the property *@type* are appended.

4.2 Linkedator-API

The Linkedator-API is a component that encapsulates the Linkedator-Core into a Web API meant to be accessible for all the participant microservices. Linkedator-API exposes mainly two functionalities: register a semantic microservice description; and invoke the core component to create and append links to a given representation. It is important to notice that the Linkedator-API works only as a mediator for the Linkedator-Core; the functionalities are actually performed by the core component.

Figure 3 shows a sequence diagram that represents the processes of microservice registration and link creation. Firstly, the Linkedator-API loads the ontology file. Then, a microservice should perform an HTTP POST request for registering its description. When a given consumer interacts with a registered

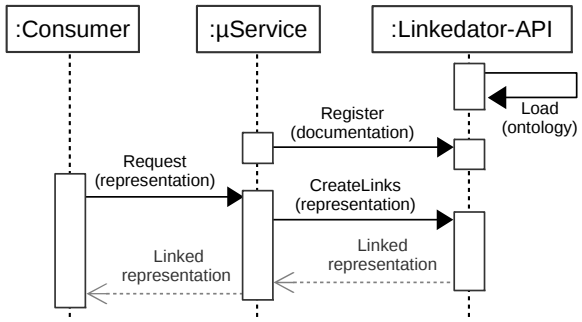

Figure 3. Microservice description registry and link creation processes

microservice, the microservice sends the requested representation to the Linkedator-API for creating all possible links. The Linkedator-API appends the resulting links to the representation and returns it to the microservice, which forwards it to the consumer. The link creation process is transparent to the consumer. Furthermore, new microservices are able to join and register their description at run time, resulting in more data sources and consequently more possibilities for interlinking representations.

The Linkedator-API supports additional configurations, such as link verification and caching. When the link verification is enabled, all links generated by the engine are verified. The verification is performed by executing an HTTP HEAD request to the link, which must result in an HTTP OK response (status code 200), otherwise the link is ignored. This verification is an assurance that the representation will be enriched only with link that are valid at creation time. When caching is enabled, the results of the link validation process are stored in a cache, avoiding to validate a link repeatedly and improving the performance of the composition as a result.

4.3 Linkedator-Jersey

This component is a tool for developing microservices using JAX-RS, the standard technology for developing RESTful Web Services on the Java platform. Its main goal is to automatically create the microservice description by analyzing the annotations used to create endpoints implemented with Jersey – the JAX-RS reference implementation. Figure 4 shows an example of a microservice description. The microservice described in this example is able to manage instances of class *"http://ontology#ClassX"* defined in the domain ontology. Firstly, the path to the OWL ontology file must be informed. Then, the types of entities that the microservice is able to manage are defined as semantic resources. Semantic resources have to define at least one URI template, otherwise links cannot be created to that entity. In this example, the defined URI template is used to obtain resource representations of *ClassX* by informing an entity's property value. A microservice description must semantically define its URI templates, which implies the semantic description of the template variables. In this example, the meaning of variable *y* is defined by property *"http://ontology#propertyY"* described in the domain ontology.

Linkedator-Jersey also facilitates the interaction with the Linkedator-API. By using this component, the microservice description registry is performed automatically when the service starts. The developer only has to configure the address of the Linkedator-API in the configuration file. The second role of the component is to automatically intercept all consumer requests and transparently invoke the Linkedator-API to create links in representations served to consumers.

```
{
 "ontologyFile": "ontology.owl",
 "semanticResources": [{
   "entity": "http://ontology#ClassX",
   "uriTemplates": [{
     "method": "GET",
     "uri": "entity{?y}",
     "parameters": {
       "y": "http://ontology#propertyY" }
    }
   ]
  }
 ]
}
```

Figure 4. Example of a microservice description

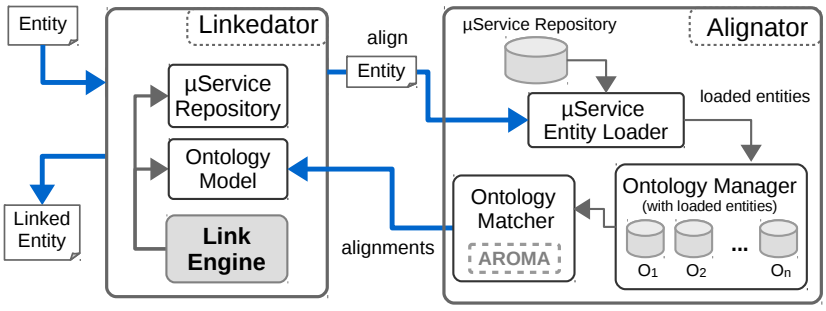

Figure 5. Interaction between Alignator and Linkedator

5 THE ALIGNATOR FRAMEWORK

This section presents Alignator, a framework for aligning heterogeneous ontologies that are used to describe data managed by microservices. Alignator aims at finding out alignment triples among several ontologies considering data entities described by them. Although it was designed as an independent piece of software, it can be used along with the Linkedator framework so as to support the creation of semantic links between resources managed by microservices in cross-domain scenarios.

Alignator is divided into four main components, as depicted in Fig. 5. The first component is a µService Description Repository, which stores documentation that describes interaction details regarding registered microservices. The second component is a µService Entity Loader, which is capable of obtaining related entities from registered microservices based on an entity sample. The third component, the Ontology Manager, is responsible for managing ontologies used by registered microservices and their corresponding loaded entities. Finally, the fourth component is the Ontology Matcher, designed to find equivalent semantic properties and classes. The resulting alignments produced by Alignator are then considered by Linkedator to create links between resources semantically described by different ontologies and managed by different microservices.

The µService Description repository stores semantic descriptions, which describe entities that the microservice is able to manage and the ways to obtain them. Both Linkedator and Alignator have a dedicated µService Description repository. Nonetheless, they may contain different sets of descriptions, since they are independent and used for different reasons. Linkedator uses microservice descriptions to create links between entities, whilst Alignator uses them to invoke microservices in order to load entities.

The µService Entity Loader is responsible for loading related entities from registered microservices. It is able to access the µService Description repository and execute HTTP requests based on URI templates.

This component plays an important role, since the alignment process is better performed when not only ontological concepts are defined, but also when entities are considered. Then, when an entity is loaded, the Ontology Manager is invoked to add the loaded entity into the corresponding ontology as a new individual.

The Ontology Manager is responsible for managing all ontologies sent by registered microservices. It holds not only ontological concepts, but also manipulates entities provided by microservices and loaded by the μService Entity Loader component. Both Linkedator and Alignator manage domain ontologies. However, Linkedator manages only concepts and does not consider entities, as these are not necessary for the link creation process. There is also an Entity Number Control Mechanism (ENCM) designed to prevent an excessive number of entities from impairing the alignment. Hence, when a threshold is reached, the Ontology Manager flushes the entities of the ontology so as to reduce memory consumption.

Finally, the Ontology Matcher is at the heart of Alignator. This component takes a set of ontologies as input and produces alignment statements. The ontologies are described in OWL and the resulting alignments are represented through the use of *owl:equivalentProperty* and *owl:equivalentClass* predicates. Currently, this component uses AROMA (Association Rule Ontology Matching Approach) (David, 2007), which, being an extensional ontology matcher, uses the entities collected by the Entity Loader to produce alignments between different ontologies. The approach used by AROMA allows to match both equivalence relations as well as relations between classes and properties of ontologies. AROMA is only capable of aligning two ontologies. Due to this limitation, all possible combinations of two ontologies are examined, resulting in $O(n^2)$ matcher executions, where n is the number of ontologies. AROMA outputs equivalent properties and classes with a certain alignment strength ranging from 0 to 1, which demands the definition of a threshold. Then, when alignment triples are produced, Linkedator is invoked to store them along with its Ontology Model.

In order to clarify Alignator's functionality, Fig. 6 shows a sequence diagram representing the process of creating alignments along with Linkedator. Firstly, microservices send an HTTP request to register their descriptions and domain ontologies in Linkedator, which also registers them in Alignator´s repository. The link creation process starts when a given consumer performs an HTTP request to Linkedator containing an entity meant to be enriched with links. After adding all possible links, Linkedator forwards the requested entity as input to Alignator so as to load related entities. Then, Alignator invokes all registered microservices according to URI templates, replacing their parameters with values contained in the requested entity. As a result, Alignator loads a set of entities from microservices, which are used as input to find out alignments. All the resulting alignments are intended to be informed to Linkedator, which should consider new alignment statements for forthcoming consumer requests. All links created by considering equivalent properties are associated with the property *rdfs:seeAlso*, which indicates that additional information might be provided by following that link.

6 RELATED WORK

The topic of data linking is directly related to many similar problems, such as data integration, ontology matching, discovery of class correspondence between heterogeneous data sources, among others. A variety of research works take into account these problems to output alignments between different concepts, properties and individuals or even to output the alignment between the taxonomies of two input ontologies. Ferrara et al. (2011) and Otero-Cerdeira et al. (2015) extensively survey works that adopt data linking approaches to evaluate both the lexical and structural similarity of entities. However, only research works that directly address the problem of connecting together different individuals provided by heterogeneous

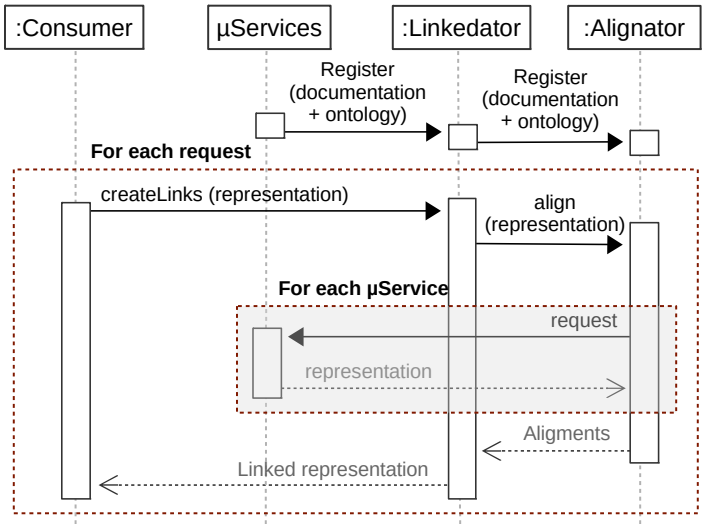

Figure 6. Sequence Diagram of Alignator with Linkedator

data sources have been considered related work.

These works were divided in two categories: SPARQL-based solutions (Araujo et al., 2011; Casanova et al., 2014; Magalhães et al., 2013) and platform-independent solutions (Hu et al., 2011; Stoermer et al., 2010). This classification criteria is relevant because the majority of microservice implementations does not consider the use of semantic technology, especially regarding the use of a triple store as the data source.

The work of Araujo et al. (2011), called *Serimi*, proposes a two phase method to address the instance matching problem. In a first phase, traditional information retrieval strategies are applied to select candidate instances to be linked. This phase aims at finding related instances that have similar labels in two RDF datasets. The resulting instances in a source RDF dataset may contain multiple distinct instances in a target RDF dataset. This problem is addressed by the second phase, which disambiguates those instances by analyzing resource descriptions to identify property sets shared by instances. In the following step, instances of the same class in the source dataset are linked to instances of the same class of interest in the target dataset. Despite the fact that *Serimi* directly addresses the problem of interlinking individuals from different datasets, it can only handle two data sources, which represents a limiting factor to deal with microservices composition. Furthermore, to be able to interlink an instance from the source dataset, it is mandatory to analyze all instances provided by the target dataset, which may be considered unfeasible in some cases. Linkedator does not limit the number of datasets and provides means for interlinking instances without analyzing instance property values of all entities.

Casanova et al. (2014) propose strategies to reduce the overhead of interlinking isolated datasets through *owl:sameAs* links, as well as strategies to improve the support to correctly maintain these links. The proposed strategies take into account the definition of views, represented by a pair $v = (V_F, F)$ where F is a SPARQL query and V_F is the vocabulary of F consisting of a single class and its properties. The authors consider that the responsibility of creating materialized *owl:sameAs* linksets should be shared between data administrators and users. In addition, this work presents a solution for maintaining *owl:sameAs* linksets by reconstructing the links after an update operation. Sharing the responsibilities of creating views for materialized links with users requires technical knowledge and more maintenance effort. In contrast, Linkedator only requires a semantic description of the managed resource classes and

URI templates, which can be automatically registered when Linkedator-Jersey is adopted.

Magalhães et al. (2013) present *Query Evaluation Framework - Linked Data* (QEF-LD), a module capable of performing SPARQL federated queries over distributed linked data sources. This work proposes two architectures for data linking: *Linking Data Mashups* and *Linked Data Mashup Services*. The first one allows consumers to consult several data sources through a single query interface simultaneously. The second one combines Web Services, which manage distinct data sources, by executing a query plan defined in the design step. Both architectures require a SPARQL interface as well as a triple store as data storage solution. However, in scenarios where microservices are able to manage data from a triple store, or when microservices can deal with a SPARQL query regardless their storage solution, the Linking Data Mashups architecture may be applied. Nevertheless, in the second architecture, the limitation of defining a query plan only in design time makes this architecture not suitable for microservices, since they have dynamic behavior and these services may suffer many changes during their lifecycle.

Hu and Jia (2015) introduce an approach that combines two methods for entity linkage: semantics-based and similarity-based approaches. The semantics-based approach leverages equivalence reasoning defined by *owl:sameAs* and other OWL properties, while the similarity-based approach considers shared properties between entities with similar values as evidence that these entities represent the same object. Given an entity (an instance of a class) as input, the proposed approach infers a set of semantically related entities by using functional properties. Then, assuming that related entities share some similar property-value pairs, it expands the search to include entities that match these pairs. This approach combines both semantics-based and similarity-based approaches. On the other hand, Linkedator takes into account only OWL object properties to perform semantic analysis, since the considered microservices do not handle links to external data sources, such as *owl:sameAs*. In contrast, this work requires the analysis of all entity values to decide the set of equivalent entities.

FBEM (Feature-Based Entity Matching), proposed by Stoermer et al. (2010), is an approach for entity resolution that combines probabilistic and ontological methods for deciding whether two records describe the same entity. The FBEM algorithm makes use of string similarity between a selection of values of entities. In order to establish similarity between two entities, a ranked list of candidate entities that match a reference entity is defined. Despite being platform-independent, FBEM requires the modeling of entities by using specific ontology. In contrast, Linkedator does not impose such a modeling, it only requires a domain ontology. Furthermore, FBEM decides whether two entities refer to the same object by applying string similarity techniques. However, such approach may be unsuitable in cases when only few property values are shared among entities provided by heterogeneous data sources, such as a single identifying property.

Trinh et al. (2015) present a platform for integrating heterogeneous data sources. Linked widgets are the main components of the proposed platform. They can be seen as Web Services with some important distinctions. For example, widgets are meant to be used directly by end users, they are always associated with a semantic model and the communication between widgets is bidirectional. Widgets are able to collect data from one or more data sources as well as to process and combine data through enrichment, transformation and aggregation. They are also capable of providing data visualization. The platform includes a communication protocol designed to facilitate interactions among widgets, allowing distributed mashups. Widgets are semantically annotated in terms of their inputs and outputs. These annotations are stored in a repository, which can be accessed via SPARQL endpoints. The platform also provides a tolerant search, which makes use of the Alignment API (David et al., 2011) to return widgets that have similar semantic models. This work addresses a wider range of problems, providing means for data

visualization for end users. On the other hand, it only considers semantic models based on inputs and outputs for creating data integration mashups. Furthermore, the ontology alignment takes only concept definitions into account, unlike Alignator which also considers entities.

7 EVALUATION

According to Tosi and Morasca (2015), researchers mainly keep their efforts in defining new ontologies and tools, instead of the real implementation of Semantic Web Services. As a result, proposals are kept at an abstract level that does not help the wide adoption of these technologies. In order to address this issue, we have conducted a real-world case study, which implements microservices that manage open government data.

7.1 Case Study

The case study is based on open data published by the Brazilian Government Transparency Portal[4], which provides CSV files for downloading information with regard to federal contracts, payments, federal workers, social programs and a variety of other public information. Three different datasets have been chosen for this case study, namely, records of a) federal civil servants, b) payments for the acquisition and contracting of public works as well as government purchases, and c) suppliers that provide goods or services. A total of 690,930; 8,670 and 50,080 records about civil servants, payments and suppliers, respectively, have been used in this case study, all published in January 2017. All the three selected datasets share with one another some information intersection, such as the full name and document number of a person, as well as the identification number and name of suppliers.

As shown in Fig. 7, three microservices have been designed for managing the records of each selected dataset. µService Payments is able to retrieve a payment resolving the URI template `?paymentID`. It is also possible to retrieve a list of all payments through the URI suffix `/list` without informing parameters. Suppliers can be retrieved by resolving the URI template `/?{registerNumber}` and the µService Servants is responsible for providing an interface, through the URI template `?{docID, fullname}`, to retrieve civil servants by their document identification and full name. Fig. 7 also illustrates the information intersection between entities of datasets. On one hand, Civil servants share their document identification and full name with Payments. On the other hand, Suppliers share their register number and name with Payments, although only register number is used by the microservice as the unique identifier.

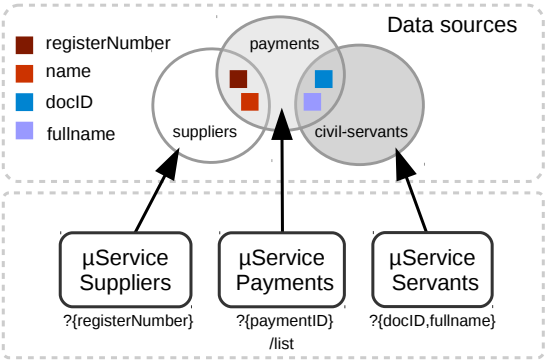

Figure 7. Case study's components

[4] `http://transparencia.gov.br`

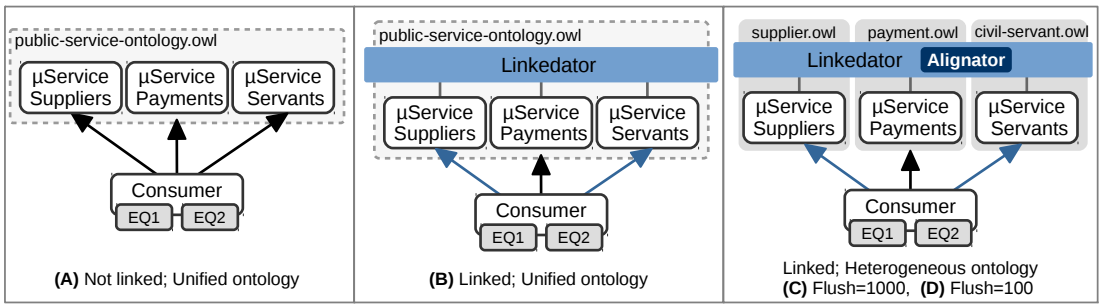

Figure 8. Case study scenarios

7.2 Scenarios

The case study was built and evaluated in four different scenarios, as depicted in Fig. 8:

(A) Not Linked: microservices are deployed without Linkedator.

(B) Linked: all microservices are deployed using the unified ontology *public-service-ontology.owl*. For that reason, the Alignator framework is disabled.

(C) Linked (ENCM=1000): microservices are deployed using their own ontologies. The Alignator framework is enabled with ENCM set to 1000 entities in each ontology.

(D) Linked (ENCM=100): microservices are deployed using their own ontologies. The Alignator framework is enabled with ENCM set to 100 entities in each ontology.

We have also designed a Consumer in order to properly perform the evaluation. The Consumer's implementation relies on two evaluation questions based on the real-world data managed by microservices that compose the case study. The questions aim at finding the federal servants that operate more payments (EQ1) as well as suppliers that received the highest amounts of money from the government (EQ2). The Consumer basically retrieves the list with all payments available in the Payments µService and attempts to follow links to servants and suppliers associated with such payment.

In the first scenario, the Linkedator framework is disabled. Therefore, the Consumer must know implementation and deployment details such as URI templates and server IP addresses of each microservice. As shown in Fig. 8, the second scenario makes use of *public-service-ontology.owl*, which comprises all microservice data concepts in only one ontology. As Linkedator creates links, the Consumer can follow them and retrieve the suppliers and/or servants associated with a payment. The presence of heterogeneous ontologies is evaluated in the third and fourth scenarios, in which each microservice uses its own particular ontology carrying different concepts. The blue arrows depicted in Fig. 8 indicate that HTTP requests are done by the Consumer using the links created by Linkedator. The black arrows represent the requests in which the Consumer previously knows the URIs.

To better understand the desired result of using Linkedator along with Alignator, Fig. 9 shows the expected linked entities. In (a) an example of a *civil-servant* is presented, while (b) and (c) present a *payment* and a *supplier*, respectively. It is worth mentioning that civil-servants and payments share the same value of *servant:registrationNumber* and *payment:executorRegisterNumber* properties, without the information on how to access the related entities. In the same way, payments share some property values (*payment:providerRegisterNumber* and *payment:providerName*) with the suppliers (*supplier:partnerRegistrationID* and *supplier:partnerName*). However, all this information is

```
{ "@context": { "servant": "http://civil-servant.owl#" },
  "@type": "servant:PublicServant",
  "servant:registrationNumber": "1**14",
  "servant:name": "VALMIR M SILVA",
  "servant:professionalPosition": "Administrator",
  "rdfs:seeAlso": ["http://government-payments/1"]
}
```


(a)


```
{ "@context": { "payment": "http://payment.owl#" },
  "@type": "payment:Expense",
  "@id": "http://government-payments/1",
  "payment:expenseValue": 743.73,
  "payment:expenseDate" : 05-12-2016,
  "payment:providedBy": {
    "payment:providerRegisterNumber": "5***8239***156",
    "payment:providerName": "AUTO POSTO ESTONIA 3 LTDA.",
    "@type": "payment:Provider",
    "rdfs:seeAlso": ["http://suppliers/5***8239***156"]
  }
  "payment:executedBy": {
    "payment:executorRegisterNumber": "1**14",
    "payment:executorFullName": "VALMIR M SILVA",
    "@type": "payment:Executor",
    "rdfs:seeAlso": [
      "http://civil-servants/docID=1**14\&name=VALMIR M SILVA"]
    }
}
```


(b)


```
{ "@context": { "supplier": "http://supplier.owl#" },
  "@type": "supplier:Partner",
  "supplier:partnerRegistrationID": "5***8239***156",
  "supplier:partnerName" : "AUTO POSTO ESTONIA 3 LTDA.",
  "supplier:partnerTributationType" : "Comercio varejista...",
  "rdfs:seeAlso": ["http://government-payments/1"]
}
```


(c)

Figure 9. Example of expected entities in JSON-LD

described by different properties. Additionally, the payment entity holds the object property *executedBy*, which can be linked to the corresponding civil-servant. Similarly, the supplier entity holds the object property *payment:providedBy* that corresponds to the supplier. Thus, it is expected that Alignator identifies the following property equivalences: *payment:executorFullName* = *servant:name*, *payment:executorRegisterNumber* = *servant:registrationNumber*, and *payment:providerRegisterNumber* = *supplier:partnerRegistrationID*. As a result, these alignment triples are informed to Linkedator, which creates links associated with the concept *rdfs:seeAlso* to related entities.

All components included in the scenarios have been deployed into the Amazon EC2 (Elastic Compute Cloud) environment. Five dedicated virtual machines have been created: 3 for the microservices of the case study, one for the Linkedator-API (which also runs Alignator as an internal module) and another one for the Consumer. The selected instance configuration was m3.medium, with Intel Xeon E5-2670 v2 (Ivy Bridge) processors running at 2.6 GHz, 3.75 GiB of memory, with Ubuntu Server 14.04 LTS (HVM) and Oracle Java Development Kit 8.

7.3 Experimental Results

The objective of this evaluation is to find out which factors influence the response time regarding the adoption of the proposed composition method and of the Linkedator framework, with (and without) the Alignator framework. In order to allow the replication of experimental results, source code and

Table 1. Experiment scenarios' results - Mean times per request (time in milliseconds)

Scenario	Consumer	Linkedator	Alignator	Total Requests
(A)	4.10	-	-	26,010
(B)	39.83	26.49	-	22,318
(C)	386.06	45.35	363.47	22,297
(D)	139.44	37.95	99.33	22,310

instructions for setting up this evaluation are available in a public repository[5].

The consumer application was executed in all four previously described scenarios, and the results are shown in Table 1. This table presents the mean response-time/request spent for creating links in Linkedator framework, for aligning ontologies in Alignator framework and the time perceived by the Consumer. One can notice that the time perceived by the Consumer is increased in nearly 10 times due to the link creating process, when comparing scenarios (A) with (B). However, the impact of the ontology alignment is even higher, increasing response time in nearly 100 times, as can be seen in (C). On the other hand, as shown in (D), when the entity number control mechanism is properly used, it was possible to reduce the alignment time in 64%.

Table 1 also presents the total number of requests executed by the consumer to obtain all entities to answer the evaluation questions. It is worthwhile to note that the number of requests executed by the Consumer is different in each scenario. In scenario (A), for each record retrieved by the μService Payments, the consumer executed other two requests to μService Suppliers and to μService Servants. Nevertheless, there are payments that do not have correspondence to registered civil servants or suppliers, resulting in invalid links. However, when Linkedator is used, it validates such links and then only valid links are executed. In scenarios (C) and (D), it is necessary to perform alignments to properly create links among heterogeneous ontologies. This process requires a certain number of entities before producing alignments with a specific strength, which causes some link misses.

Figure 10 shows how the use of the proposed composition method is perceived by the client application with respect to response time distribution. Figure 10 (a) presents the time distribution considering the first scenario (not linked representations). Most requests in this scenario are executed between 3 ms and 5 ms. Figure 10 (b) presents the time distribution considering the second scenario (linked entities using unified ontology). In this scenario, most requests are executed within two ranges, the first one between 30 ms and 40 ms, the second one between 50 ms and 60 ms. Figure 10 (c) presents the distribution of requests executed in the third scenario (linked entities using heterogeneous ontology and entity number control set to 1000 entities per ontology). One can notice that the link ontology matching process distributes requests more evenly between 200 ms and 800 ms, reducing the spikes observed in previous scenarios. However, the use of a suitable entity control mechanism, presented in (d), concentrates the execution of most requests between 140 ms and 200 ms.

Figure 11 shows the results regarding two distinct entity number control mechanisms. Graphs (a), (c) and (e) present the alignment strength, number of entities in each ontology and the ontology matching time, respectively, achieved in the first 2000 requests performed by the consumer for scenario (C), while (b), (d) and (f) present the same results considering the first 200 requests for scenario (D). It is important to notice that in both scenarios there were several flushes throughout the execution. In (a), where the ENCM is set to 1000 entities per ontology, flushes are performed around each 250 requests, while in (b), where ENCM is set to 100 entities per ontology, flushes are performed quite more often. Furthermore, the

[5]https://bitbucket.org/salvadori/ijwis2017-case-study/

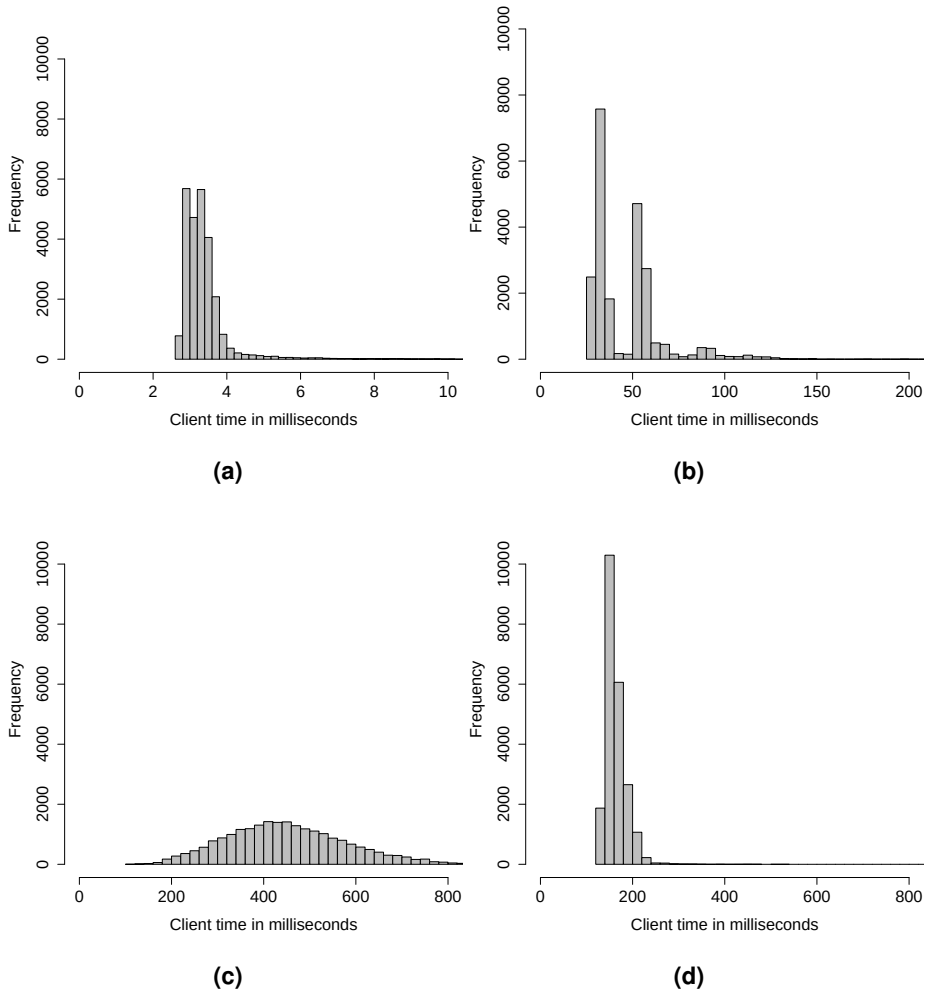

Figure 10. Client response time distribution (a) Not linked; (b) Linked entities using unified ontology; (c) Linked entities using heterogeneous ontology and entity number control set to 1000 entities per ontology; (d) Linked entities using heterogeneous ontology and entity number control set to 100 entities per ontology

alignment strength achieved the maximum value before all sequence of flushes in scenario (a). On the other hand, such alignment strength was not achieved before all flashes in (b). However, Alignator was developed to consider the highest alignment strength achieved in its runtime. Once the maximum value is achieved, lower strength values are not considered.

There is a direct relation between the number of entities considered in the ontology matching process and the time required to produce such alignments. In Figure 11 (e), there were three spikes within the range between the 1st and the 1,000th requests, that correspond to the exact moment in which flushes were performed (c). In contrast, considering scenario (D), in which flushes were performed around each 20 requests (d), the time required to produce alignments was significantly reduced, as can be seen in (f).

Figure 12 (a) shows an analysis of variance regarding the time spent by Linkedator to create links for scenarios (A), (B) and (C). It is possible to observe the impact of ontology alignments in the link creation process. Considering that scenario (B) uses an unified ontology, there is no need for ontology alignment, so there is no effect in the link creation process. However, scenarios (C) and (D) require such alignments.

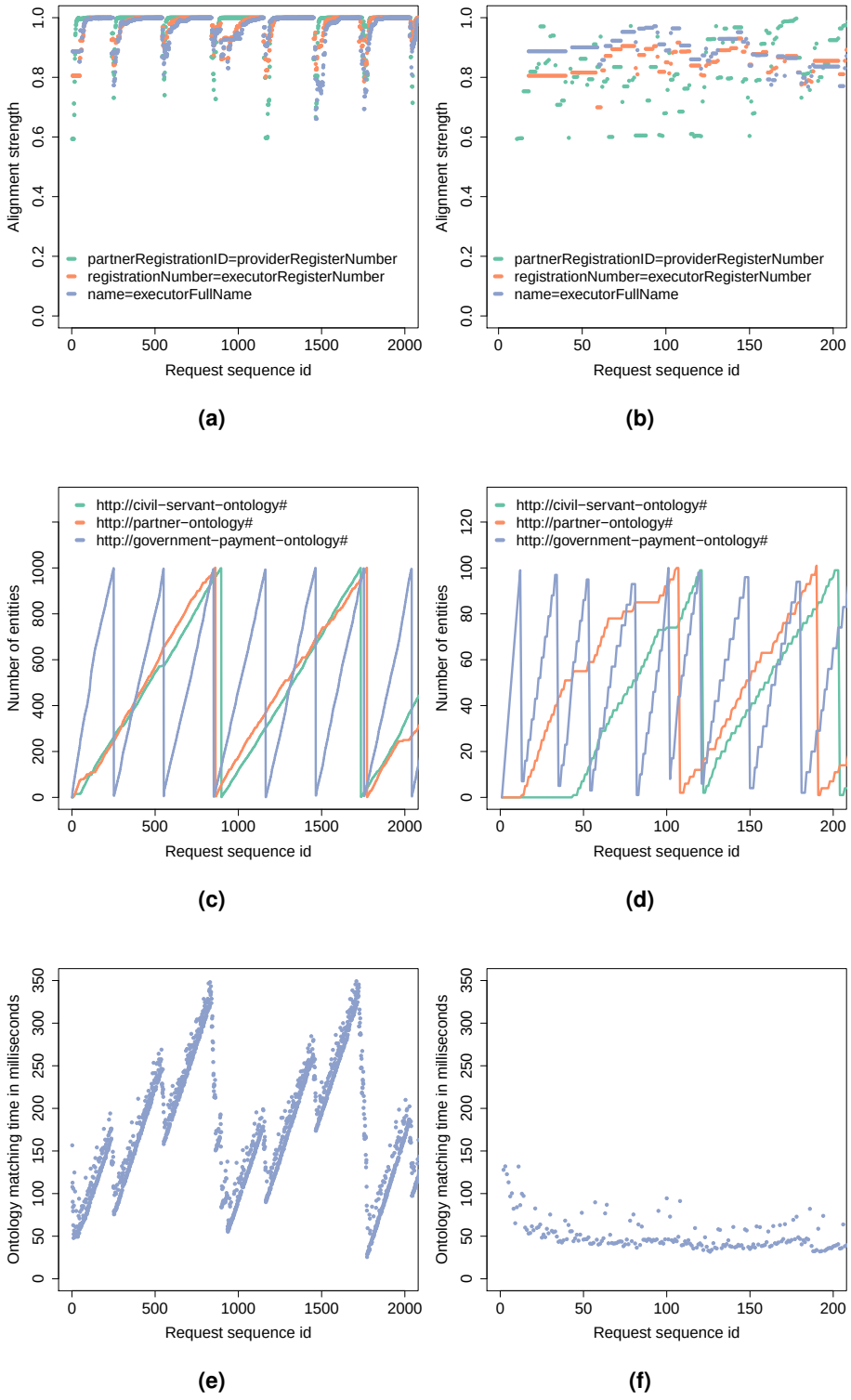

Figure 11. Entity number control mechanism analysis; (a) alignment strength, (c) number of entities in ontologies, (e) matching time for ENCM=1000; (b) alignment strength, (d) number of entities in ontologies, (f) matching time for ENCM=100

One can notice that the required time for creating links in those scenarios had a slight increase. This is due to the fact that equivalent properties and classes are being considered in heterogeneous ontology scenarios. Furthermore, scenario (C) has a significant time variance. This is due to the interaction between Linkedator and Alignator in which Alignator directly writes alignments into Linkedator's ontology model. Such write operation is affected by the number of entities of managed ontologies that Alignator has to deal with.

Figure 12 (b) shows the time required to perform ontology alignments for scenarios (C) and (D). The time spent by the ontology matcher and by Alignator, which encompasses the first one, are shown separately. The first important thing to notice is that the ontology matcher corresponds to 83% of the time spent by Alignator in scenario (C) and 43% in scenario (D). In addition, the time spent by Alignator in scenario (D) was dramatically reduced when compared with scenario (C). This shows the importance of setting up a suitable ENCM to avoid such waste of time, since both scenarios have achieved the same alignment strength value. In addition, scenario (C) has higher time variance due to the larger number of entities which grows in a non-continuous way up to a larger maximum of 3000 entities Figure 11 (c). This non-continuous growth impacts both the variance of alignment time as well as the time required to feed Linkedator with the updated alignments, as can be seen in Figure 11 (e) and Figure 11 (f).

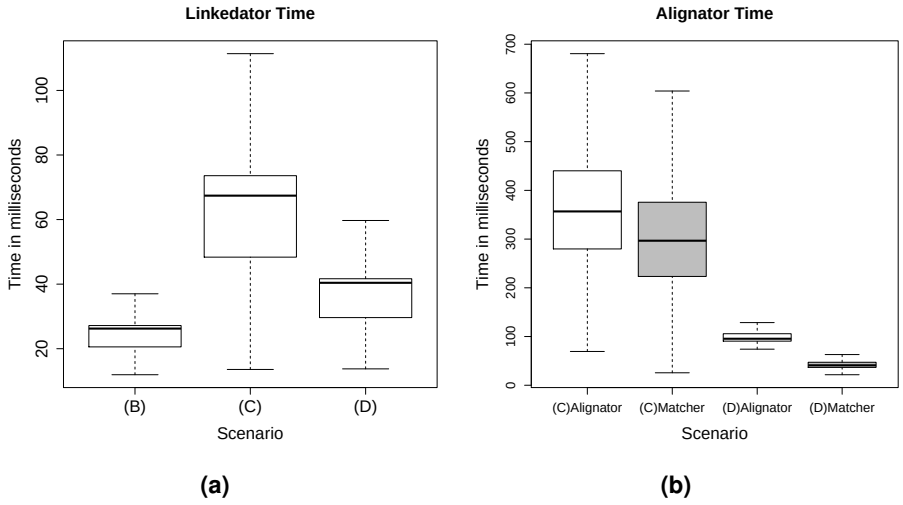

Figure 12. Time variance analysis; (a) Linkedator framework; (b) Alignator framework

8 CONCLUSIONS AND FUTURE WORK

This paper described a method for composing semantic microservices, as well as two frameworks that provide support for executing this method. The proposed method combines semantic standards and data linking techniques for composing microservices modeled as entity providers. Its main goal is to enrich representations with links generated at request time, taking into account a domain ontology and semantic descriptions of microservices. The first framework, named Linkedator, aims to facilitate the adoption of the proposed method. As a result, a given client application capable of dealing with linked representations is able to access more representations without knowing all interaction details. Furthermore, new microservices may join and automatically be accessible for existing clients. Additionally, the Alignator framework aims at identifying alignments of heterogeneous ontologies used by participating microservices. Therefore, it permits to create links between entities, even when microservices handle

heterogeneous ontologies.

Linkedator and Alignator frameworks enable an application architecture to deal with the dynamic nature of microservices, which can be updated or replaced more often than traditional monolithic applications are. Linkedator generates links, both originating and targeting the new or updated microservice, without introducing coupling between services. Alignator allows this to hold even when the microservice in question uses a different ontology with unknown equivalence to those currently used in the architecture. In this sense, microservices can be dynamically removed, updated or incorporated to the architecture without the need of changes to other services, only requiring that they (re-)register with Linkedator.

The evaluation study measured the impact of adopting Linkedator and Alignator frameworks, considering the case study scenario. However, different scenarios that explore some characteristics, such as the adoption of an unified ontology or heterogeneous ontologies were analyzed in order to identify their impact on the link creation process. Scenarios that explore specific characteristics of the adoption of heterogeneous ontologies were also considered. For those scenarios, the evaluation showed the importance of properly defining a threshold to be used by the entity number control mechanism, otherwise the ontology matcher can become very time-consuming or result in alignment strengths that do not achieve the full potential of the Alignator framework.

This work considered microservices designed as data providers. However, not only entities can be linked together, but actions can also be integrated with entities. In future works we intend to consider microservices that implement functionalities of a given business domain to also connect entities with their available actions. In addition, Alignator could be evaluated with other ontology matchers besides AROMA. Thus, it will be possible to perform a comparison between different algorithms in order to maximize the alignment strength. In order to address different characteristics of entities and microservices, Alignator could use more than one ontology matcher. Thus, a mechanism would be necessary to select the most appropriate algorithm in different scenarios.

REFERENCES

Araujo, S., de Vries, A., and Schwabe, D. (2011). SERIMI Results for OAEI 2011. In *Proceedings of the 6th International Conference on Ontology Matching*. CEUR-WS.org.

Battle, R. and Benson, E. (2008). Bridging the semantic Web and Web 2.0 with Representational State Transfer (REST). *Web Semantics*, 6(1):61–69.

Berners-Lee, T., Hendler, J., and Lassila, O. (2001). The Semantic Web. *Scientific American*, 284(5):34–43.

Bizer, C., Heath, T., Idehen, K., and Berners-Lee, T. (2008). Linked data on the web (ldow2008). In *Proceedings of the 17th International Conference on World Wide Web*, WWW '08, pages 1265–1266, New York, NY, USA. ACM.

Casanova, M. A., Vidal, V. M. P., Lopes, G. R., Leme, L. A. P. P., and Ruback, L. (2014). On Materialized sameAs Linksets. In *Database and Expert Systems Applications: 25th International Conference*, pages 377–384. Springer International Publishing.

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

David, J., Euzenat, J., Scharffe, F., and Trojahn dos Santos, C. (2011). The alignment api 4.0. *Semant. web*, 2(1):3–10.

Elmagarmid, A. K., Ipeirotis, P. G., and Verykios, V. S. (2007). Duplicate record detection: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 19(1):1–16.

Euzenat, J. and Shvaiko, P. (2007). *Ontology Matching*. Springer-Verlag New York, Inc., Secaucus, NJ, USA.

Fellah, A., Malki, M., and Elçi, A. (2016). Web services matchmaking based on a partial ontology alignment. *Int. Journal of Information Technology and Computer Science (IJITCS)*, 8(6):9–20.

Ferrara, A., Nikolov, A., and Scharffe, F. (2011). Data Linking for the Semantic Web. *International Journal on Semantic Web and Information Systems*, 7(3):46–76.

Fielding, R. T. (2000). *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, Irvine.

Heath, T. and Bizer, C. (2011). *Linked data: Evolving the web into a global data space*. Morgan & Claypool Publishers.

Hu, W., Chen, J., and Qu, Y. (2011). A self-training approach for resolving object coreference on the semantic web. In *Proceedings of the 20th international conference on World wide web - WWW '11*, page 87, New York, New York, USA. ACM Press.

Hu, W. and Jia, C. (2015). A bootstrapping approach to entity linkage on the Semantic Web. *Web Semantics: Science, Services and Agents on the World Wide Web*, 34:1–12.

Islam, N., Abbasi, A. Z., and Shaikh, Z. a. (2010). Semantic web: Choosing the right methodologies, tools and standards. In *2010 International Conference on Information and Emerging Technologies, ICIET 2010*, pages 1–5. IEEE.

Köpcke, H. and Rahm, E. (2010). Frameworks for entity matching: A comparison. *Data Knowl. Eng.*, 69(2):197–210.

Lanthaler, M. and Gütl, C. (2012). On Using JSON-LD to Create Evolvable RESTful Services. In *Proceedings of the Third International Workshop on RESTful Design*, WS-REST '12, pages 25–32, New York, NY, USA. ACM.

Magalhães, R. P., Monteiro, J. M., Vidal, V. M. P., de Macêdo, J. A. F., Maia, M., Porto, F., and Casanova, M. a. (2013). QEF-LD - A Query Engine for Distributed Query Processing on Linked Data. In *Proceedings of the 15th International Conference on Enterprise Information Systems*, pages 185–192. SciTePress.

Martin-Flatin, J. P. and Löwe, W. (2007). Special Issue on Recent Advances in Web Services. *World Wide Web*, 10(3):205–209.

McIlraith, S., Son, T., and Zeng, H. Z. H. (2001). Semantic Web services. *IEEE Intelligent Systems*, 16(2):46–53.

Newman, S. (2015). *Building Microservices*. O'Reilly Media, Gravenstein Highway North, Sebastopol, CA. USA.

Otero-Cerdeira, L., Rodríguez-Martínez, F. J., and Gómez-Rodríguez, A. (2015). Ontology matching: A literature review. *Expert Systems with Applications*, 42(2):949–971.

Pavel, S. and Euzenat, J. (2013). Ontology matching: State of the art and future challenges. *IEEE Trans. Knowl. Data Eng.*, 25(1):158–176.

Richards, M. (2015). *Microservices vs. Service-Oriented Architecture*. O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472. O'Reilly.

Salvadori, I., Huf, A., Santos Mello, R., and Siqueira, F. (2016). Publishing linked data through semantic microservices composition. In *Proc. of Int. Conf. on Information Integration and Web-based Applications & Services*. ACM.

Stoermer, H., Rassadko, N., and Vaidya, N. (2010). Feature-based entity matching: The fbem model, implementation, evaluation. In *Proceedings of the 22Nd International Conference on Advanced*

Information Systems Engineering, CAiSE'10, pages 180–193, Berlin, Heidelberg. Springer-Verlag.

Tosi, D. and Morasca, S. (2015). Supporting the semi-automatic semantic annotation of web services: A systematic literature review. *Information and Software Technology*, 61:16–32.

Trinh, T.-D., Wetz, P., Do, B.-L., Kiesling, E., and Tjoa, A. M. (2015). Distributed mashups: a collaborative approach to data integration. *Int. Journal of Web Information Systems*, 11(3):370–396.