An Ontology Alignment Framework for Data-driven Microservices

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

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

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

Eduardo C. Inacio
eduardo.camilo@posgrad.ufsc.br
Dept. of Informatics and Statistics
Federal University of Santa Catarina
Florianópolis, Santa Catarina, Brazil

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

ABSTRACT

The integration of heterogeneous data sources can be done by using data-driven microservices along with Semantic Web technologies. However, it becomes a challenge in cross-domain scenarios, in which data is described by heterogeneous ontologies. This work discusses data integration problems in the context of microservices and proposes an ontology alignment framework aimed at achieving semantic data-driven microservices composition for data integration in multi-ontology scenarios. The proposed framework is evaluated through an experimental design to obtain a suitable statistical analysis of the effects that may be considered for modeling data-driven microservices.

CCS CONCEPTS

• **Information systems → Data exchange**; **Web data description languages**; *Web searching and information discovery*; *Web services*;

KEYWORDS

Semantic Web, Semantic Web Services, Ontology Matching, Microservices

ACM Reference Format:
Ivan Salvadori, Bruno C. N. Oliveira, Alexis Huf, Eduardo C. Inacio, and Frank Siqueira. 2017. An Ontology Alignment Framework for Data-driven Microservices. 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, 9 pages. https://doi.org/10.1145/3151759.3151793

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

ACM ISBN 978-1-4503-5299-4/17/12…$15.00
https://doi.org/10.1145/3151759.3151793

1 INTRODUCTION

Microservices are an emerging implementation approach to SOA (Software Oriented Architecture) that encompasses tenets covering fine-grained interfaces and architectural attributes such as isolated state, loose coupling, and deployment and operation characteristics [19]. The application of the microservices approach implies the implementation of functionality and management of data independently by different services [10]. The application of some tenets introduces fragmentation of data and functionality between different services, which may need to be undone to achieve some of the business-specific goals.

Monolithic applications usually provide several features through a single central point. In this scenario, there is no data integration problem, since all information is managed by a single provider. By adopting the microservices architecture, such information is distributed into several independent small services developed with distinct implementations and entity models. These characteristics pose challenges on how to compose them back together. Semantic Web technologies may be applied to provide machine-readable descriptions of the data managed by a microservice. However, microservices may be semantically described by heterogeneous ontologies, which result in data interpretation problems. In order to integrate the data managed by several microservices, it not only sufficient to know the Web interfaces, but to also understand the meaning of data. When this meaning is described by heterogeneous ontologies it is mandatory to create equivalence relations that map those concepts.

In previous work we investigated approaches regarding microservices composition [13]. However, in a microservices architecture, data itself may be heterogeneous to the point of being described by distinct ontologies, which might jeopardize the composition process. In general, it is not realistic to assume that data provided by services will always be defined by a single ontology [6], specially when they are developed by independent teams to fit distinct applications. The ontology heterogeneity problem can be solved with the alignment of the corresponding classes and properties defined by the multiple ontologies that describe data provided by microservices [12].

There have been some proposed works in the literature to tackle the problem of integrating heterogeneous data exposed through

services interfaces. Trinh et al. [16] present a platform for integrating heterogeneous data sources. Fellah, Malki and Elçi [6] propose a matchmaking algorithm and a partial ontology alignment mechanism for SAWSDL that uses similarity degrees among concepts across ontologies considering syntax, linguistic and structural aspects. Vander et al. [17] tackle the problem of discovering and querying data on the web, employing Linked Data principles. However, none of them specifically address data-driven microservices.

This work discusses the data integration problem associated with the adoption of microservices and presents Alignator, a framework for aligning heterogeneous ontologies used to describe microservices' data. It takes as input sample entities and exploits documented service descriptions to find additional data that could be used in the matching process. These sample entities may be provided by service consumers, developers or reverse proxies, avoiding the need for a complete dump of the data exposed by the service. Based on the fetched data, Alignator employs ontology matching algorithms and output alignment statements. The Alignator framework is evaluated by an experimental design in order to identify factors that affect the performance of the approach. Experimental results showed the effectiveness of our proposal in aligning distinct ontologies of datasets provided by microservices.

The remainder of this paper is organized as follows. Section 2 summarizes the main concepts required for understanding this work. Section 3 describes in detail the definition of the research problem. Section 4 proposes the Alignator framework and Section 5 discusses related work. Section 6 presents the evaluation and describes the execution scenarios as well as the results obtained in experiments. Finally, Section 7 draws the conclusion and presents future work.

2 BACKGROUND

This section introduces the main concepts required to understand the framework described in this paper.

2.1 Microservices

According to Zimmermann [19], microservices do not constitute a new architectural style different from SOA (Software Oriented Architecture). Instead, they can be seen as a particular implementation approach to SOA in terms of service development and deployment. Thus, microservices present an evolutionary and complementary position to develop SOA applications, adhering to seven tenets presented by Zimmerman [19]:

(1) fine-grained interfaces;
(2) business-driven development;
(3) IDEAL (Isolated state, Distribution, Elasticity, Automated management and Loose coupling);
(4) polyglot programming and persistence;
(5) lightweight container deployment;
(6) decentralized continuous delivery; and
(7) DevOps with holistic service monitoring.

Microservices may be implemented by using Web Services and Web APIs, or by using a message broker that supports message queues and allows communication through publisher/subscriber mechanisms. Microservices can also employ Semantic Web Services technology, resulting in semantic microservices. According to McIlraith, Son and Zeng [9] 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, as well as 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 [9].

2.2 Data-driven Microservices

Service computing is a broad concept to describe initiatives that allow companies to expose their core competencies as services [2]. Similarly to how microservices can be seen as a particular approach to implement SOA [19], data-driven microservices represent a particular approach to implement microservices. Two distinct approaches are currently in use for designing microservices: data-driven and action-driven approaches. Data-driven microservices are responsible exclusively for managing the lifecycle of one or more classes of entities, working as a data provider. The management of entities' lifecycle can also be seen as a business goal when it applies business constraints. However, data-driven microservices are limited to manage the information that may be used in a business process instead of implementing such a process. On the other hand, microservices that implement functionalities of a business domain are defined as action-driven microservices. Actions usually take as input properties of entities, perform some business process and may produce an entity as a response.

Among the tenets presented by Zimmerman [19], business-driven development may be considered the key principle for implementing microservices. As discussed by Pautasso et al. [11], microservices should be focused on business goals and how to enable them. Actually, both microservices approaches mentioned above can be adopted to enable business goals. However, it is important to take into consideration that the more specialized a business goal is, less likely will be the reuse of the microservice in other domains. This issue could most directly affect data-driven microservices, in which data constraints regarding a given domain might not fit properly to related domains. Although the adoption of business-driven development helps to identify and conceptualize services, it should affect their reusability as well as their composability. However, generic services may not be the solution when modeling microservices, since they might not provide full capabilities to perform specific business goals with the desired results. With this in mind, developing microservices is about deciding how specific or reusable services should be to tackle business goals.

2.3 Ontology Alignment

The adoption of microservices brings the idea of loose coupling and independent maintenance, which can lead to the design of heterogeneous ontologies describing the same domain.The OWL vocabulary, since its inception, includes predicates to denote equivalence of individuals, classes and properties. However, 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 are 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 [5], 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 [12]. 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 entities and properties of such ontologies. Each correspondence denotes a relation of equivalence, generalization or disjointness between two entities of O_1 and O_2 [12]. Generally, these correspondences are determined by means of a degree of similarity among the entities of ontologies. Euzenat and Shvaiko [5] propose methods for assessing this similarity based on specific features of entities. The basic techniques are classified as follows:

- Name-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 entities, the structure of elements found in 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 PROBLEM DEFINITION

A monolithic application usually provides several features through a single software artifact. When a monolithic application is used to manage the lifecycle of entities, it usually manages several types of entities in a given domain. In this scenario, there is no data integration problem, since all information is managed by a single provider. On the other hand, the adoption of a microservices architecture results in a separation of such features into several independent small services. In this scenario, each microservice manages a subset of entities that may require combining entities provided by other microservices in order to perform the same features that the monolithic application is capable of. One can argue that each microservice should be able to perform a complete business goal; thus, there is no need for composing them. This allegation takes into account only the originally designed perspective. However, when reusing a microservice in a different context, there is no guarantee that a given goal can be fulfilled by only one microservice.

In addition to the single responsibility principle, which leads to fine-grained interfaces, microservices take advantage of independent development, leading to distinct implementations and entity models. These characteristics pose challenges on how to compose them back together to fulfill business goals. One possible solution is adopting Semantic Web technologies to provide machine-readable descriptions of the data managed by a microservice, as well as interaction details. Although the adoption of semantic technologies leverages data integration, the independence of modeling and development could minimize their expected benefits. For instance, microservices may be semantically described by heterogeneous ontologies, which result in data interpretation problems. Ontology alignment techniques can be adopted to tackle this issue, since they aim at figuring out equivalent concepts among different ontologies.

In the context of microservices, ontology alignment has distinct characteristics that differ from the traditional problem, such as: entities are provided through a Web interface and the necessary interaction information may also require alignment before usage. However, the most important difference from the traditional ontology alignment is that equivalence statements are obtained based on entities resulted of the interaction between microservices and their consumers, since it is not possible to directly access a dataset. This work addresses the problem of putting diverse concepts together to create an integrated model that allows the access and understanding of the information provided by several microservices described by heterogeneous ontologies. The integrated model is created by applying ontology matching techniques specifically adapted to the context of data-driven microservices.

4 ALIGNATOR FRAMEWORK

This section presents Alignator[1], a framework for aligning heterogeneous ontologies used to describe the information managed by data-driven microservices.

4.1 Data Intersection

Alignator relies on the existence of intersection between data exposed by different services. Specifically, Alignator exploits the property values shared between entities. As an example, consider the description of a person, exposed by μS_A where each person has a `foaf:name`, and the description in another service (μS_B) where the name attribute is present as `taxpayer:fullName`. If μS_B exposes a query interface for taxpayer name, then a correferent can be found with a `foaf:name` from μS_A. Such type of data sharing is justifiable in microservices, as maintaining links between the data in μS_A and μS_B would incur a certain level of coupling between the interfaces and deployment of the microservices.

In this scenario, the equivalence between `foaf:name` and `taxpayer:fullName` is not known. In fact, it is not possible to differentiate `taxpayer:fullName` from `foaf:name`. Yet, the intersection can still be exploited by blindly obtaining potential related entities from μS_B based on the values of attributes of an entity obtained from μS_A. The set of related entities can be used to feed extensional ontology matchers [12]; thus obtaining, among others, the equivalence between `foaf:name` and `taxpayer:fullName`.

4.2 Alignator Framework Architecture

Alignator aims at finding out alignment triples among several ontologies considering microservice entities described by them. The framework is divided into four main components, as depicted in Figure 1. The first component is the *μService Description Repository*, which stores documents that describe interaction details of registered microservices. The second component is a *μService Entity Loader*, which is capable of obtaining related entities from registered microservices based on a sample entity. The third component, the *Ontology Manager*, is responsible for managing ontologies used

[1] https://github.com/ivansalvadori/alignator-core

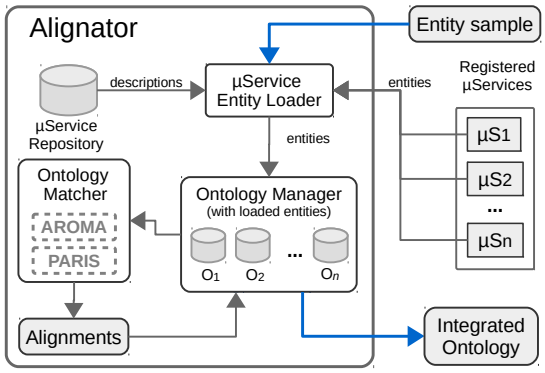


Figure 1: Alignator framework architecture

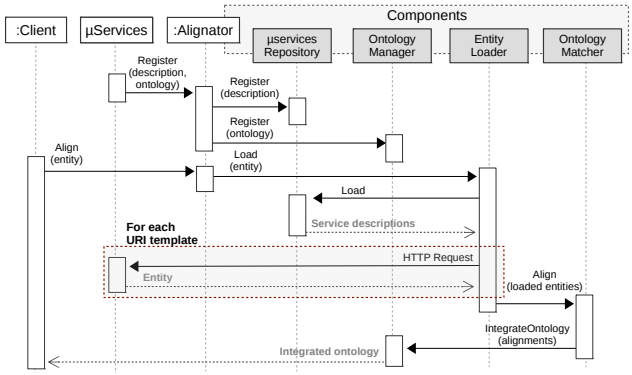


Figure 3: Sequence diagram

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 a data consumer interested in accessing entities semantically described by different ontologies and managed by different microservices. Alignator may also be adopted directly by microservices or middlewares, as previously shown by Salvadori et al. [14].

The µService Repository stores semantic descriptions, which describe the entities a microservice is able to manage and how to obtain them.Alignator adopts Hydra [8] for representing microservice descriptions. As shown in Figure 2, supported classes represent a list of types of entities managed by a microservice. A supported class has a list of URI templates, which describe the necessary information to perform HTTP requests to access entities that can be uniquely identified by parameters. Those parameters are also associated with concepts defined by an ontology, which allows to properly understand the described microservice's inputs.

```
{
  "@context": "http://www.w3.org/ns/hydra/context.jsonld",
  "@id": "http://api.example.com/doc/",
  "@type": "ApiDocumentation",
  "supportedClass": [{
     "@id": "http://ontology#Entity",
     "@type": "IriTemplate",
     "template": "entities{?p}",
     "mapping": [{
        "@type": "IriTemplateMapping",
        "variable": "p",
        "property": "http://ontology#property"
      }
     ]
   }
  ]
}
```

Figure 2: Example of a semantic microservice description

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 ontology matching 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.

The Ontology Manager is responsible for managing all ontologies used by registered microservices. It is also responsible for creating an integrated ontology based on alignments produced by the Ontology Matcher. It holds not only ontological concepts, but also manipulates entities provided by microservices and loaded by the µService Entity Loader, keeping independent repositories for each domain ontology. There is also an entity number control mechanism (ENCM) that allows setting the maximum number of entities in each ontology. It means that when an ontology achieves the threshold, 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. It 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. Typically, ontology matchers are not expected to align simultaneously more than two ontologies. Due to this limitation, ontologies are combined in pairs, resulting in k-combinations defined by $C(O_n, 2)$, where n is the number of ontologies.

Currently, the ontology matcher component can use one of two ontology matchers. One of them is AROMA (Association Rule Ontology Matching Approach) [3]. AROMA employs extensional techniques to analyze the set of instances of entities in order to compute the correspondences and obtain the alignment between different ontologies. The approach used by AROMA allows to match both equivalence relations as well as relations between classes and properties of ontologies.

Another adopted ontology matcher is PARIS (Probabilistic Alignment of Relations, Instances and Schema) [15]. PARIS iterative computing alignments for individuals, properties and classes, which are counted in subsequent iterations. Instance alignment locates a property shared by the ontologies that acts as a highly inverse functional property and for which the two correferent individuals (subjects) share the same object. For subclass relations, the probability that $c_1 \subseteq c_2$ is defined by the number of instances of $c_1 \cap c_2$ in proportion to the number of instances of c_1. Similarly, for subproperties, it assumes that the probability of $p_1 \subseteq p_2$ is related to the number of subject-object pairs occurring with both p_1 and p_2 in proportion to those with p_1.

4.3 Dynamic Perspective

In order to provide a suitable explanation of how Alignator works along with its components and external actors, a sequence diagram is depicted in Figure 3. The starting point is the semantic description registry, where descriptions and ontologies used by microservices are sent to the Alignator framework to indicate that those microservices are participating members. During this process, Alignator registers both the semantic description and the ontology of a microservice separately into the μService Repository and the Ontology Manager, respectively.

The ontology matching process starts when a client sends an sample entity to Alignator, which in its turn asks the Entity Loader to obtain all possible related entities from registered microservices. Firstly, the Entity Loader obtains the semantic descriptions managed by the μService Repository. Then, based on those descriptions, URI templates are processed with all their input parameters replaced by the values extracted from the sample entity. This process may result in several invalid URIs, which will return an HTTP 404 status code. However, any entities that result from this process are forwarded to the Ontology Matcher to perform the matching process and figure out new alignment triples. The resulting alignment triples are sent to the Ontology Manager to be incorporated in the integrated ontology, which contains the concepts regarding all registered ontologies, as well as their equivalent relations.

5 RELATED WORK

Several approaches have been proposed to tackle the problem of integrating heterogeneous data exposed through services interfaces. From one perspective, this problem can be understood as a sub-problem of service composition, in which services act as data providers. On another facet, the problem may be seen as crawling Web Services while applying the Linked Data principles [1].

Trinh et al. [16] 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 instance, widgets are meant to be used directly by end users. They are associated with a semantic model and 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 interaction between widgets, allowing distributed mashups. Widgets are semantically annotated in terms of their inputs and outputs and these annotations can be accessed via SPARQL endpoints. The platform also provides a tolerant search, which makes use of the Alignment API [4] to return widgets that have similar semantic models. However, 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.

Fellah, Malki and Elçi [6] propose a matchmaking algorithm and a partial ontology alignment mechanism for SAWSDL. It uses similarity degrees among concepts across ontologies considering syntax, linguistic and structural aspects. The algorithm takes as input two ontologies and returns the matching degree between their elements. In terms of syntax, it adopts Levenshtein similarity, which measures the similarity of two strings on a scale from 0 to 1. Regarding linguistics, it uses WordNet, which represents a set of synonyms for the English language. With respect to structural aspects, it exploits the hierarchy tree of concepts. This work only considers the semantics used to describe Web Services, nevertheless it does not consider concepts used to describe information. Alignator does not make use of synonyms to create alignments. On the other hand, it considers the content of managed entities, which results in a richer input source. In addition, this related work focuses on SOAP services, whilst Alignator focuses on data-driven microservices.

Vander et al. [17] tackle the problem of discovering and querying data on the Web, employing Linked Data principles [1]. The querying problem is treated as a federated query problem, and data services discover other related data services using their own data as starting point (active discovery) and the requests they receive (reactive discovery). Data services are described by their algorithmically generated data summaries and the client is guided through hypermedia control to perform queries using Triple Pattern Fragments [18]. Issues related to ontology heterogeneity are not addressed. However, clients and servers could be enhanced to reason over OWL alignment triples and Alignator could be fed from data obtained during dataset discovery and query execution.

6 EVALUATION

This section describes the experimental evaluation, as well as the adopted methodology and the obtained results. The objective of this evaluation is to find out which factors influence the alignment strength regarding the adoption of Alignator with both AROMA and PARIS ontology matching algorithms. In order to allow the replication of experimental results, source code and instructions for setting up this experiment are available in a public repository[2].

6.1 Methodology

In order to evaluate ontology matchers, a test bench application and a microservice have been developed, as depicted in Figure 4. A synthetic dataset of 1,000 entities with characteristics to be exploited by the evaluation is managed by both the test bench application and the microservice. Each entity of the dataset contains four properties associated with alphanumeric values and a payload information. Two distinct ontologies were created to describe the entities managed by the test bench application and by the microservice. The test bench application was designed to interact with Alignator by sending each entity of the dataset as an input to the ontology matching process, considering different characteristics applied to the entities provided by the microservice. The main goal is to identify important characteristics that affect the resulting alignments in order to establish guidelines for modeling microservices.

Four factors were considered for this evaluation: A) the number of shared property values, B) the entity's payload size, C) the correspondence level between entities of the two ontologies meant to be aligned and D) the order in which entities are sent to Alignator. These factors were considered because they can be controlled or considered in the microservices' development process in order to

[2] https://drsalvadori.bitbucket.io/Alignator/iiwas2017

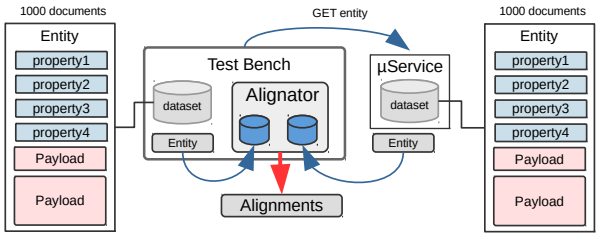


Figure 4: Evaluation scenario

maximize the alignment results. As shown in Table 1, two distinct levels were associated with each factor: a low and a high level. For factor A, previous experiments showed that AROMA requires individuals to share at least two common properties in order to produce property or class alignments that could not be otherwise trivially inferred. While PARIS has no such requirement, the levels of this factor were chosen to accommodate AROMA and avoid distortions in analysis. For factor B, the lower level represents a payload with only one word, which is equivalent to one more property; and the high level is equivalent to a small text property value with 100 words. Factor C is not totally controlled by the developer, however, it is likely to be a known information and may be considered at design time. Finally, factor D exploits the order in which entities are sent to Alignator. Fixed access means that entities are sent to Alignator in the same order throughout the experiment execution whereas random access results in a different order for each observation. Fixed access could be seen as a software agent interaction mode, and random access is closely related to a human interaction mode, in which entities are not accessed following a predefined procedure.

Factors in Table 1 describe data and access pattern. The Ontology Matchers are not considered factors. Instead, each matcher is analyzed independently, aiming to identify how it responds to the scenarios modeled by these factors, coupled with Alignator.

In order to obtain a statistical analysis of effects, a 2^k experimental design [7] was adopted, in which the aforementioned factors and their levels were considered for each of the four evaluated factors. For each ontology matcher algorithm, the experiment was replicated three times to properly account for variability, resulting in 48 observations. The execution order was completely random to make sure that response variables are independently and individually distributed. The experiment was executed using the computational infrastructure provided by the Cloud Computing for Cooperation[3] (C3 Lab), with the following configuration: a dedicated server with 24 Intel Xeon X5690 processors at 3.47GHz frequency and 148 GiB of primary memory, running the CentOS-7 operating system and openjdk version 1.8.0_141 with -Xmx set to 2 GiB for both the microservice and the test bench application.

6.2 Analysis of Effects

The effects of the factors in Table 1 were analyzed with respect to the number of requests necessary until the alignment strength reported by the matcher under analysis reached a satisfactory value.

[3]http://www.c3lab.tk.jku.at

Table 1: Factors and levels

	Factors	-1	+1
A	Shared property values	2	4
B	Payload size	1 word	100 words
C	Entity correspondence level	50%	100%
D	Access order	fixed	random

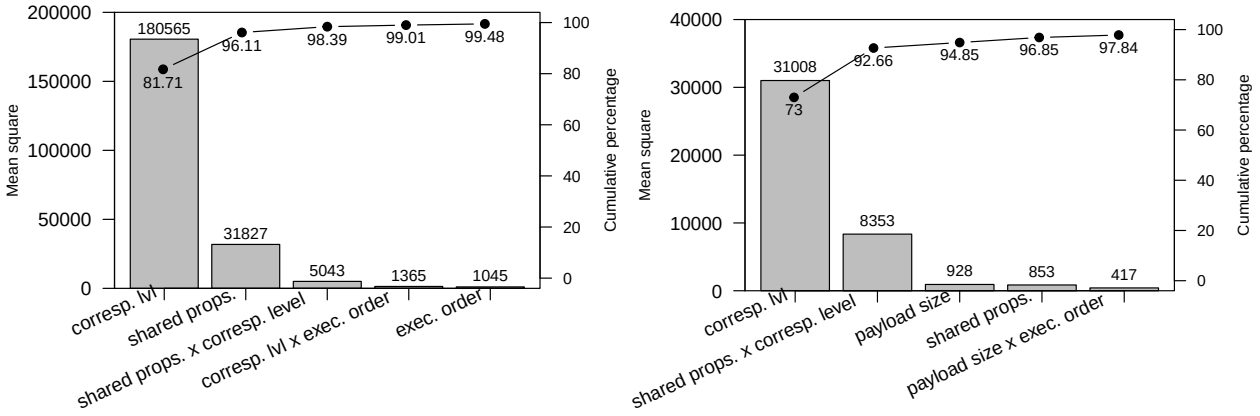
Two response variables were used as possible definitions of satisfactory: $N_{0\%}$ and $N_{1\%}$, where the $N_{\delta\%}$ notation should be read as "the number of requests made by the test bench application (Figure 4) before the reported alignment strength for `ont0:o0p1 owl:equivalentProperty ont1:o1p1` had a value s such that $s - s_{max} \leq \frac{\delta}{100}$, where s_{max} is the maximum strength observed for any of the 1000 requests in the experiment." Informally, $N_{0\%}$ and $N_{1\%}$ represent the number of requests until, respectively, reaching maximum alignment strength and reaching maximum alignment strength with 1% tolerance (as for both matchers $s \in [0, 1]$).

Analysis for $N_{0\%}$. Two linear models yielding $N_{0\%}$ from the evaluated factors were adjusted to the experimental data. One model for the AROMA matcher and another for PARIS. Both homoscedasticity and normality assumptions, required by the adopted analysis method, were met by the models. Further, an adjusted R^2 of 0.9712 was obtained for the model corresponding to AROMA, and an adjusted R^2 of 0.6826 for PARIS. Both R^2, which denotes a good fit and, thereby, supports the conclusions on significance of effects.

Among all considered factors and their pairwise interactions for AROMA, the following presents statistically significant effect at a level of significance of 5% ($\alpha = 0.05$): (C) entity correspondence level, (A) shared property values, (D) access order, (A+C) the interaction between shared property values and entity correspondence level, (C+D) and the interaction between entity correspondence level and access order. For PARIS, the significant factors and pairwise interactions are: C, A, D, A+C, C+D. The payload size presented no statistically significant effect on the response variable as well as other not mentioned interactions for both ontology matchers. Figure 5(a) and Figure 5(b) presents, respectively for AROMA and PARIS, Pareto charts with the ANOVA mean square and the cumulative percentage of variability explained by the statistically significant factors. These charts reveal that for both matchers, factor C (correspondence level) is responsible for almost all variability in $N_{0\%}$. Important differences, revealed by the Pareto charts, are that PARIS is more sensitive to access order (D) and less sensitive to the number of shared properties (since it requires only one shared property between individuals to infer correferences).

The different requirements of AROMA and PARIS, for the number of shared properties, also appears in Figure 6. This chart displays the effects on $N_{0\%}$ when each factor is changed from its low level (-1) to its high level (1). For AROMA, increasing the number of shared properties reduces $N_{0\%}$ from 170 to 118. This effect is not observed for PARIS, which is designed to identify correferent individuals from a single shared property.

Figure 6 reveals that the most significant factor for both matchers is the correspondence level (C). In the case of AROMA, changing



(a) (b)

Figure 5: Pareto charts for the $N_{0\%}$ models: (a) AROMA and (b) PARIS.

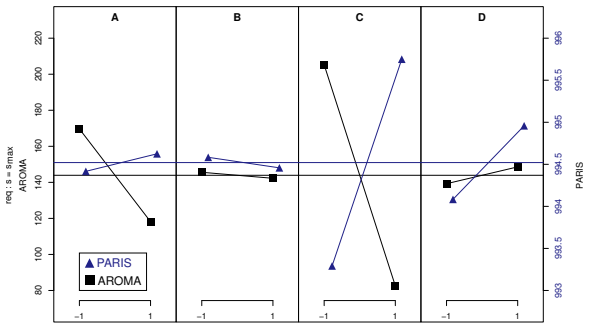


Figure 6: Main Effects plot for response variable $N_{0\%}$ (first request with maximum strength)

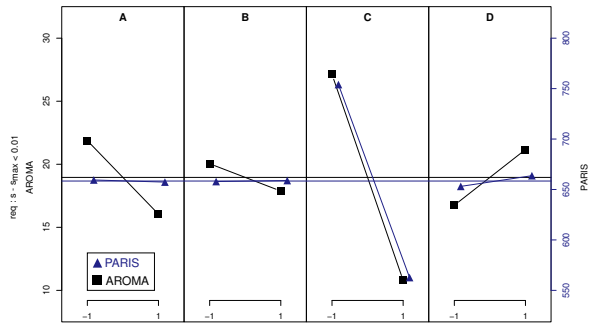


Figure 7: Main Effects plot for response variable $N_{1\%}$ (first request with strength within 1% of maximum)

C from 50% (-1) to 100% (+1), a decrease in $N_{0\%}$ from 205 to 83 is expected. This result is intuitive, since AROMA will always be able to identify correferent individuals, and will have more evidence sooner to infer that `o0:o0p1 owl:equivalentProperty o1:o1p1`. In the case of PARIS, the opposite occurs: $N_{0\%}$ is expected to increase from 993.29 to 995.75 when the correspondence level increases. PARIS identifies only `rdfs:subPropertyOf` ($\subseteq$) relations between properties, which Alignator combines using simple average to infer `owl:equivalentProperty` (=). For these experiments, in general, $Pr(\texttt{o0p1} \subseteq \texttt{o1p1}) \geq Pr(\texttt{o1p1} \subseteq \texttt{o0p1})$ when not all individuals have correspondents (through these properties).

Finally, Figure 6 reveals that the range on which $N_{0\%}$ is affected by any factor is considerably small: from 993 to 996. This range is negligible in practical scenarios and occurs due to the significant slower rate which PARIS increases the strength of its alignments. The reader can refer to Figure 8(a) and Figure 8(c) showing alignment strength versus request number, respectively for AROMA and PARIS. Further details on the evolution of alignment strength is discussed in 6.3.

Analysis for $N_{1\%}$. As previously discussed, while it highlights behavior differences among AROMA and PARIS, $N_{0\%}$ is somewhat disconnected from a practical scenario. Using $N_{1\%}$ offers a counterpoint. Again two linear models were generated, now with an adjusted R^2 of 0.8485 for AROMA and 0.9847 for PARIS.

Among all considered factors, and their pairwise interactions, for AROMA, the following factors presented statistically significant effect at a level of significance of 5% ($\alpha = 0.05$): (C) entity correspondence level, (A) shared property values, (D) access order, (A+C) the interaction between shared property values and entity correspondence level, and (C+D) the interaction between entity correspondence level and access order. For PARIS, the significant factors and pairwise interactions are only factors C and D.

Figure 7 reveals a reverse situation from that observed for $N_{0\%}$: the range in which $N_{1\%}$ is affected is small for AROMA (from 11 to 27) and larger for PARIS. Factor C (correspondence level) remains the factor with largest impact for both aligners, and for PARIS, $N_{1\%}$ is expected to decrease from 563 to 754 when C changes from 50% to 100%.

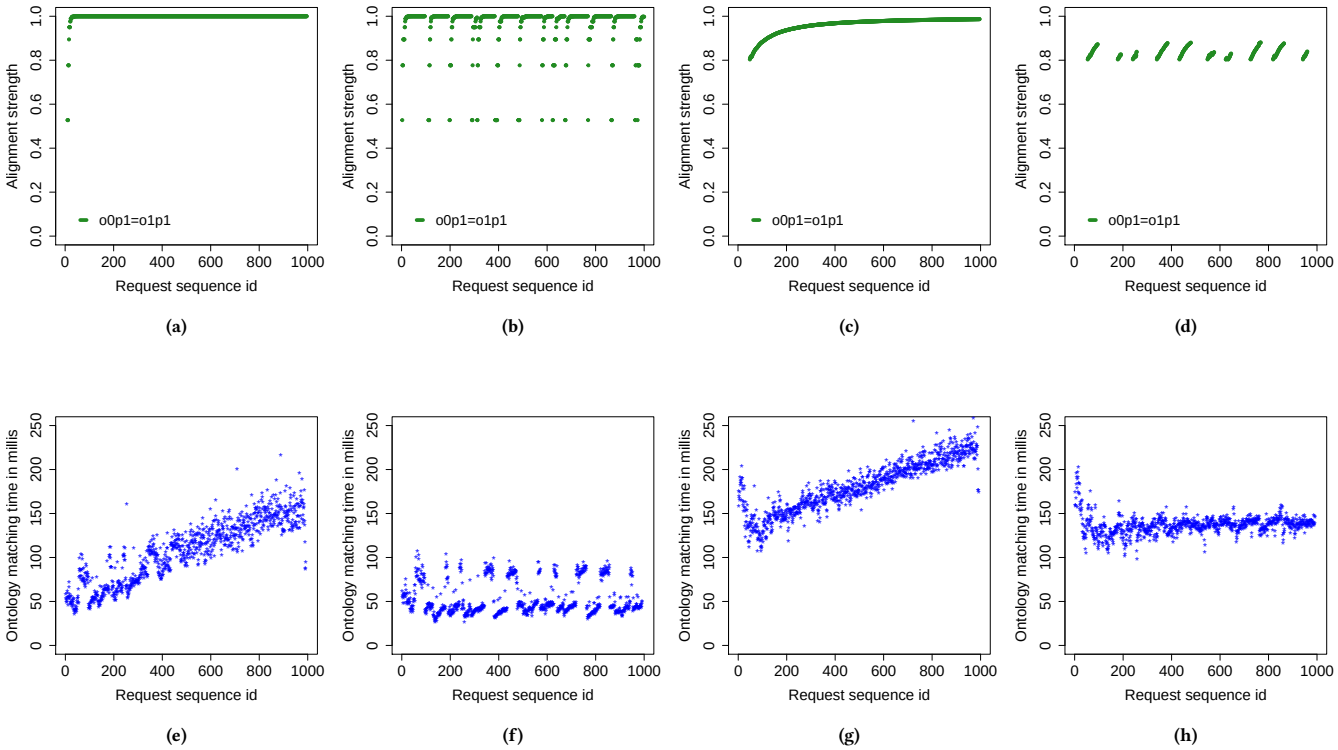


Figure 8: Alignment strength and time of alignment per request; (a) alignment strength without ENCM - AROMA; (b) alignment strength with ENCM=1000 - AROMA; (c) alignment strength without ENCM - PARIS; (d) alignment strength with ENCM=100 - PARIS; (e) alignment time without ENCM - AROMA. (f) alignment time with ENCM=100 - AROMA; (g) alignment time without ENCM - PARIS. (h) alignment time with ENCM=100 - PARIS

Final remarks. Most preceding discussion centered on factors A and C. Factor B (payload size) was only significant in the model for $N_{1\%}$ using AROMA. However, the effect (decrease of $N_{1\%}$ from 20 to 18) is negligible for all practical purposes. Factor D (access order) reveals that the particular ordering of the data in the microservice is, in the four models, above average with respect to reaching high alignment strengths. However, we observe that the effects, even when statistically significant, are small in the light of practical applications. The largest observed effect of D, in absolute values, is in Figure 7 when $N_{1\%}$ increases from 635 to 664 for PARIS when access is switched from fixed to random. The Pareto graphs also show that the contribution of D to variability pales in face of factor C.

In a practical scenario, reaching the maximum alignment strength may not be necessary, as usually the threshold is fixed for the adopted matcher. The response variable $N_{1\%}$ is closer to this situation. AROMA requires fewer requests to reach high alignment strengths, but it is affected by factor A (shared properties), requiring at least 2 shared properties to generate alignments. PARIS, on the other hand, is not affected by A, but presents a slower evolution of the alignment strength. In summary, the choice between AROMA and PARIS depends on (1) whether the individuals have properties acting as composite keys or as a single key; (2) how many requests before alignment strength reaching a threshold are acceptable for the specific application.

6.3 Ontology Matching Time

In addition to the experimental design presented in the last section, an analysis regarding the ontology matching was conducted to study Alignator's behavior. As aforementioned, Alignator relies on the existence of intersection between data exposed by different data-driven microservices. Based on a sample entity, it loads related entities from registered microservices. Hence, the resulting entities are added to each ontology that is aligned by the Ontology Matcher component. The more entities are in an ontology, the higher is the alignment strength. However, accumulating entities into ontologies affects the time to obtain alignments. In order to address this issue, a mechanism to control the number of registered ontologies' entities was developed in the Ontology Manager component, as mentioned in Section 4. By defining a maximum number of entities, the ontology matching time is reduced while the maximum alignment strength is achieved.

Figure 8 shows the alignment strength and ontology matching time per request for each entity of the dataset. In (a) and (c) of

 DOI: 10.1145/3151759.3151793

Figure 8, one can notice that the maximum alignment strength that states `o0p1` described by the ontology `http://ontology0#` is equivalent to `o1p1` described by the ontology `http://ontology1#` is achieved before the 100th request. However, the entity number control mechanism (ENCM) was not enabled. As a consequence of that, the ontology matching time required to align these properties is significantly increased, as can be seen in (e) and (g) of Figure 8. By setting the ENCM to 100, when a given ontology reaches 100 entities, all entities are discarded. As can be seen in (b) and (d) of Figure 8, the alignment strength drops after every 100 requests. It is important to mention that the highest achieved alignment strength is considered despite this drop caused by ENCM. As at any time, at most 100 entities are considered, the ontology matching time per request is significantly reduced, as shown in (d) and (h) of Figure 8. With respect to the alignment strength attained by both ontology matchers, we can see that AROMA reaches a maximum before PARIS, as shown in Figure 8 (a) and (c). The ontology matchers also required different times for performing alignments. For both configurations, with and without ENCM, PARIS spent significantly more time when compared to AROMA. Furthermore, the ontology matching time analysis shows the importance of finding out a suitable number of entities that are required to attain the maximum alignment strength in order to ensure an adequate performance and alignment strength. One can notice that ENCM set to 100 entities was appropriate to AROMA, as shown in Figure 8 (a), which was able to achieve its maximum alignment strength. On the other hand, as can be seen in Figure 8 (e), this configuration was inappropriate to PARIS, resulting in lower alignment scores.

7 CONCLUSION AND FUTURE WORK

This paper discussed the drawbacks regarding data integration caused by the adoption of the microservices architecture. It also proposed Alignator, a framework that aims to create an integrated ontology for semantic data-driven microservices integration in a multi-ontology scenario. Alignator is able to invoke and load entities provided by registered microservices in order to produce alignment statements that represent semantic connections between concepts used to describe microservices data. By considering these alignments, it is possible to improve the integration of data-driven microservices and, moreover, improve the effectiveness of composition approaches based on data linking [13].

The proposed framework was evaluated under an experimental design encompassing four factors. Evaluation results indicated the factors that influence the alignment strength, such as the number of property values shared between related entities, the correspondence level of entities and the order in which entities are loaded. Results obtained by the analysis of effects showed the impact of such factors, which can be considered for modeling microservices. Results also showed the importance of the entity number control mechanism to ensure a suitable ontology matching time.

In future work Alignator will be evaluated with other ontology matchers besides AROMA and PARIS. Thus, it will be possible to perform a comparison among a wide variety of algorithms in order to maximize the quality of alignments. In order to address different characteristics of entities and microservices, Alignator could use more than one ontology matcher at runtime. Thus, a mechanism would be necessary to perform the most appropriate algorithm in the face of different scenarios. Additionally, the maximum number of entities managed by the Ontology Manager component is currently defined by a parameter before runtime. A dynamic mechanism for automatically defining this number at runtime is interesting so that maximum alignment is achieved with the least number of requests, minimizing thereby the ontology matching time.

REFERENCES

[1] Christian Bizer, Tom Heath, and Tim Berners-Lee. 2009. Linked Data - The Story So Far. *Int. Journal on Semantic Web and Information Systems* 5, 3 (jan 2009), 1–22. https://doi.org/10.4018/jswis.2009081901

[2] Glaucia Melissa Medeiros Campos, Nelson Souto Rosa, and Luis Ferreira Pires. 2014. A Survey of Formalization Approaches to Service Composition. In *2014 IEEE International Conference on Services Computing*. IEEE, 179–186. https://doi.org/10.1109/SCC.2014.32

[3] Jérôme David. 2007. Association Rule Ontology Matching Approach. *Int. Journal on Semantic Web and Information Systems* 3, 2 (2007), 27–49. https://doi.org/10.4018/jswis.2007040102

[4] Jérôme David, Jérôme Euzenat, François Scharffe, and Cássia Trojahn dos Santos. 2011. The Alignment API 4.0. *Semant. web* 2, 1 (Jan. 2011), 3–10.

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

[6] 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. https://doi.org/0.5815/ijitcs.2016.06.02

[7] Raj Jain. 1991. *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*. Wiley. 720 pages.

[8] Markus Lanthaler. 2013. Creating 3rd Generation Web APIs with Hydra. In *Proc. of the 22nd International World Wide Web Conference (WWW2013)*. International World Wide Web Conferences Steering Committee, Geneva, Switzerland, 35–37.

[9] S.a. McIlraith, T.C. Son, and Honglei Zeng Honglei Zeng. 2001. Semantic Web Services. *IEEE Intell. Syst.* 16, 2 (2001), 46–53. https://doi.org/10.1109/5254.920599

[10] Sam Newman. 2015. *Building Microservices* (1st ed.). O'Reilly Media, Inc.

[11] Cesare Pautasso, Olaf Zimmermann, Mike Amundsen, James Lewis, and Nicolai Josuttis. 2017. Microservices in Practice, Part 1: Reality Check and Service Design. *IEEE Software* 34, 1 (jan 2017), 91–98. https://doi.org/10.1109/MS.2017.24

[12] Shvaiko Pavel and Jerome Euzenat. 2013. Ontology Matching: State of the Art and Future Challenges. *IEEE Trans. Knowl. Data Eng.* 25, 1 (Jan. 2013), 158–176. https://doi.org/10.1109/TKDE.2011.253

[13] Ivan Salvadori, Alexis Huf, Ronaldo Santos Mello, and Frank Siqueira. 2016. Publishing Linked Data Through Semantic Microservices Composition. In *Proc. of Int. Conf. on Information Integration and Web-based Applications & Services*. ACM. https://doi.org/10.1145/3011141.3011155

[14] 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* (jul 2017), 00–00. https://doi.org/10.1108/IJWIS-04-2017-0029

[15] 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.

[16] Tuan-Dat Trinh, Peter Wetz, Ba-Lam Do, Elmar Kiesling, and A Min Tjoa. 2015. Distributed mashups: a collaborative approach to data integration. *Int. Journal of Web Information Systems* 11, 3 (aug 2015), 370–396. https://doi.org/10.1108/IJWIS-04-2015-0018

[17] Miel Vander Sande, Ruben Verborgh, Anastasia Dimou, Pieter Colpaert, and Erik Mannens. 2016. Hypermedia-Based Discovery for Source Selection Using Low-Cost Linked Data Interfaces. *International Journal on Semantic Web and Information Systems (IJSWIS)* 12, 3 (2016), 79–110.

[18] Ruben Verborgh, Miel Vander Sande, Olaf Hartig, Joachim Van Herwegen, Laurens De Vocht, Ben De Meester, Gerald Haesendonck, and Pieter Colpaert. 2016. Triple Pattern Fragments: A low-cost knowledge graph interface for the Web. *Web Semantics: Science, Services and Agents on the World Wide Web* 37–38 (mar 2016), 184–206. https://doi.org/10.1016/j.websem.2016.03.003

[19] Olaf Zimmermann. 2016. Microservices tenets: Agile approach to service development and deployment. *Computer Science - Research and Development* (nov 2016). https://doi.org/10.1007/s00450-016-0337-0