Publishing Linked Data Through Semantic Microservices Composition

Ivan Salvadori
Graduate Program in Computer Science (PPGCC)
Department of Informatics and Statistics (INE)
Federal University of Santa Catarina (UFSC)
Florianópolis/SC, Brazil 88040-900
ivan.salvadori@posgrad.ufsc.br

Alexis Huf
Graduate Program in Computer Science (PPGCC)
Department of Informatics and Statistics (INE)
Federal University of Santa Catarina (UFSC)
Florianópolis/SC, Brazil 88040-900
alexis.huf@posgrad.ufsc.br

Ronaldo dos Santos Mello
Graduate Program in Computer Science (PPGCC)
P.P. in Methods and Management in Evaluation (PPGMGA)
Department of Informatics and Statistics (INE)
Federal University of Santa Catarina (UFSC)
Florianópolis/SC, Brazil 88040-900
r.mello@ufsc.br

Frank Siqueira
Graduate Program in Computer Science (PPGCC)
Department of Informatics and Statistics (INE)
Federal University of Santa Catarina (UFSC)
Florianópolis/SC, Brazil 88040-900
frank.siqueira@ufsc.br

ABSTRACT

Microservices are replacing monolithic applications by splitting them out into small and independent artifacts that collaborate with one another. Focused on managing highly cohesive information, microservices may be composed to provide richer and linked information. This paper presents a composition method, aimed at composing semantic microservices for achieving data integration based on Linked Data principles. Moreover, the proposed method leverages the independence of development and composability of microservices. This paper also presents a framework and a case study for the proposed method.

CCS Concepts

•**Information systems** → **Mediators and data integration;** *Service discovery and interfaces;*

Keywords

Microservices; Linked Data; Composition

1. INTRODUCTION

The adoption of microservices is in continued expansion, turning a traditional monolithic application into a highly cohesive and loosely coupled set of services. Microservices are designed to provide solutions upon a well-defined domain, resulting in multiple components that communicate and operate together. The adoption of microservices facilitates the deployment and maintenance processes. It 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, there are few proposals that 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, Vries and Schwab [1], Casanova et al. [5] and by Magalhães et al. [16], whilst platform-independent proposals are described by Hu and Jia [12] and by Stoermer, Rassadko and Vaidya [22].

This work presents a composition method for semantic microservices. 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 presents a development framework, which aims to facilitate the adoption of the proposed composition method.

The remainder of this paper is organized as follows: Section 2 summarizes the main concepts required for understanding this work. Section 3 presents the proposed composition method for semantic microservices. The Linkedator framework is presented in Section 4. Section 5 describes related research efforts found in the literature. A case study is presented in Section 6, and the evaluation study is presented in Section 7. Finally, the conclusions and future work are presented in Section 8.

iiWAS '16, Singapore

DOI: http://dx.doi.org/10.1145/3011141.3011155

2. BACKGROUND

2.1 Semantic Web

The Semantic Web [3] 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 [4].

Four principles govern the publishing of Linked Data [10]. 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 resources *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 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 Web Ontology Language (OWL), which allows to perform reasoning and logical decisions.

2.2 Microservices

According to Newman [19], 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 more 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. According to Newman [19], each microservice should be developed and maintained by a single team. This is an important characteristic that makes microservices independent from one another and from developers as well.

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 [21] 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, Son and Zeng [18] 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 [13, 17, 18]. Battle and Benson [2] 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 [8]. 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 matching. Usually, the matching 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 [8]. 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 can be replaced 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 [6, 7, 14]. 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.

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 separated in such way that no client would need to follow links from one microservice to another.

The proposed composition method aims to create links that correspond to object properties on a domain ontology. It uses individual matching techniques considering a formal notion of identity as defined in [8]. 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, 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, working as a representation link enrichment.

3.1 Microservice Architectural Constraints

The composition method assumes a Web-based microservice architecture where 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 of 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 [9] due to the need to interface with legacy systems. Examples of person identifying attributes are U.S. social security number, passport number, user login or ID, e-mail address, etc.

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

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 any REST constraint, 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.

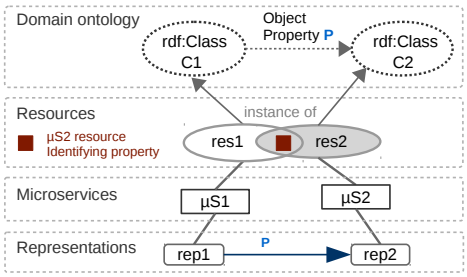


Figure 1: Data linking-based composition method

3.2 Resource Design Constraints

The major resource design constraint is that the domain must be described by an OWL ontology, and the resources managed by microservices must be instances of classes defined in this ontology. Furthermore, the data attributes of the 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 Figure 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 will not store a link to $res2$, but only its identifying data. From the 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, $res1$ must associate internally the identifying data of $res2$ with P to properly determine that $res1$ is related to $res2$ through P. In fact, we model $res1$ to contain, in this case, a blank node as 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,

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

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 [15] 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 the 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.

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 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 the 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.

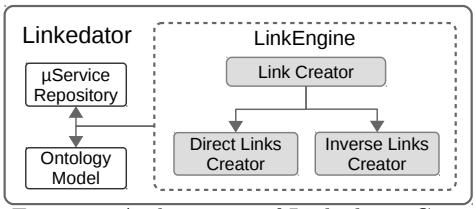


Figure 2: Architecture of Linkedator-Core

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
```

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 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, Linkedator-API loads the ontology file. Then, a microservice should perform a HTTP POST request for registering its description. When a given consumer interacts with a registered 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.

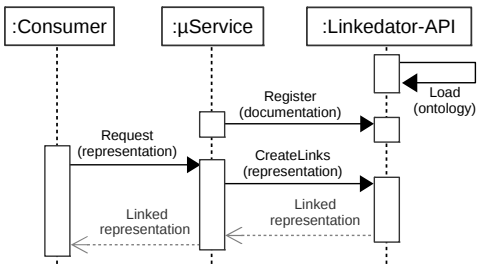


Figure 3: Microservice description registry and link creation processes

The Linkedator-API supports custom configurations, such as link verification and cache. When the link verification is enabled, all links generated by the engine are verified. The verification is performed by executing a HTTP HEAD request to the link, which must result in a HTTP OK response (status code 200), otherwise the link is ignored. This verification is an assurance that the representation will be enriched with only valid links, at least at creation time. When the cache is enabled, 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 by this example is able to manage instances of class *"http://ontology#Person"* defined in the domain ontology. In this example, the defined URI template is used to obtain resource representations by informing a *taxID*. 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 *taxID* is defined by property *"http://ontology#taxID"* described in the domain ontology.

```
{
 "semanticResources": [{
   "entity": "http://ontology#Person",
   "uriTemplates": [{
     "method": "GET", "uri": "person{?taxID}",
     "parameters": { "taxID": "http://ontology#taxID" }
    }
   ]
  }
 ]
}
```

Figure 4: Example of a microservice description

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.

5. 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, Nikolov and Scharffe [8] and Otero-Cerdeira, Rodríguez-Martínez and Gómez-Rodríguez [20] extensively survey work 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 data sources have been considered related work.

These works were divided in two categories: SPARQL-based solutions [1, 5, 16] and independent-platform solutions [11, 22]. This classification criteria is relevant because the majority of microservice implementations do not consider the use of semantic standards, especially regarding the use of triple store as the data source.

The work of Araujo, Vries and Schwab [1], 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 of 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 a 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. [5] 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. [16] 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 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 [12] introduce an approach that combines two methods for entity linkage: semantics-based and similarity-based approaches. The semantics-based approach makes use of equivalence reasoning leveraged by OWL semantics, e.g., *owl:sameAs* and other 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*. On the other hand, this work requires the analysis of all entity values to decide the set of equivalent entities.

FBEM (Feature-Based Entity Matching), proposed by Stoermer, Rassadko and Vaidya [22], 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 the 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 a FBEM 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, like a single identifying property.

6. CASE STUDY

In order to show how the proposed composition method can be applied using Linkedator, this section presents a case study based on criminal, financial and immigration records. Some technical details on the use of Linkedator are also presented in this section. This case study is also adopted for the evaluation process described in Section 7.

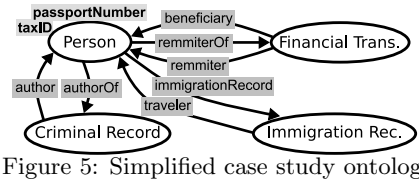


Figure 5: Simplified case study ontology

The domain ontology, summarized in Figure 5, contains four classes: *Person, Financial Transaction, Criminal Record* and *Immigration Record*. To save space, data properties of the classes are shown as light gray rectangles attached to the classes, represented by ellipses; object properties are represented by directed edges. *Person* acts as a central class whose instances may have relations to the other three classes, that themselves have properties in the reverse direction. To avoid coupling, object properties are not stored as links to resources in other microservices, therefore such links are created by the Linkedator framework.

The architecture of the case study is shown in Figure 6. Instances of *Person* are fully handled by µ*Service*1. Instances of the other three classes are handled by different providers, each one represented by a different microservice. From µ*Service*2 to µ*Service*5 manage immigration data from four continents. µ*Service*6 and µ*Service*7 manage criminal records from two providers, FBI and Interpol. Finally, from µ*Service*8 to µ*Service*11 manage financial transactions from the same four continents. All microservices are implemented without knowledge from one another and therefore, the representations produced by them do not include links to related resources on other microservices.

The fact that *taxID* and *passportNumber* are identifying properties of *Person* is not stated on the ontology, but is derived from the URI template descriptions given by the microservices. µ*Service*1 provides templates that given either a *taxID* or a *passportNumber* produce an URI that, if valid, identifies a Person instance. The µ*Service*1's templates allow Linkedator to enrich the representation at Figure 7 (d) with *owl:sameAs* links for the transaction's remitter and beneficiary, as shown in Figure 8.

In the case of *Person* representations, such as the example in Figure 7 (a), no reference remains to related resources. However, since µ*Service*2 to µ*Service*11 provide URI templates that use *taxID* and *passportNumber* as parameters, links for the object properties *immigrationRecord*, *remitterOf* and *authorOf* can be constructed by Linkedator from Figure 7 (a). The result of this enrichment is shown in Figure 9. To save paper space, some links are omitted.

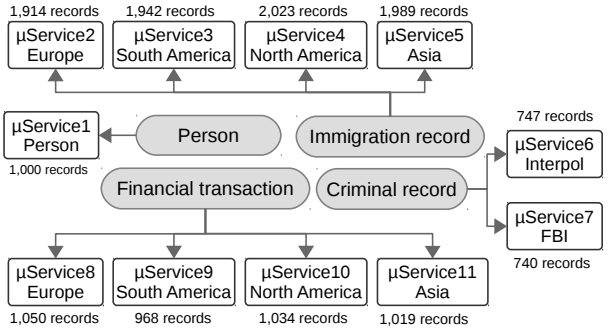


Figure 6: Microservice and data source details - case study

```
{
  "@context": { "ontology": "http://ontology#" },
  "@type": "ontology:Person",
  "ontology:taxID": "733-11716-531-23",
  "ontology:passportNumber": "253-5022-82-43967-856",
  "ontology:firstName": "Nicole",
}
```


(a)


```
{
  "@context": { "ontology": "http://ontology#" },
  "@type": "ontology:ImmigrationRecord",
  "ontology:reportFrom": "ontology:Europe",
  "ontology:traveler": {
    "ontology:passportNumber": "253-5022-82-43967-856"
  },
  "ontology:declaredMoney": "1787.82",
}
```


(b)


```
{
  "@context": { "ontology": "http://ontology#" },
  "@type": "ontology:CriminalRecord",
  "ontology:criminalAgency": "ontology:Interpol",
  "ontology:author": {
    "ontology:taxID": "733-11716-531-23"
  },
  "ontology:registerNumber": "062-11441-05780-76",
  "ontology:crime": "Fraud"
}
```


(c)


```
{
  "@context": { "ontology": "http://ontology#" },
  "@type": "ontology:FinancialTransaction",
  "ontology:transactionFrom": "ontology:NorthAmerica",
  "ontology:transactionID": "581-52207-12414-84",
  "ontology:amount": "17528.46",
  "ontology:remmiter": {
    "ontology:taxID": "733-11716-531-23"
  },
  "ontology:beneficiary": {
    "ontology:taxID": "720-72890-123-53"
  }
}
```


(d)

Figure 7: Examples of representations used in the case study: (a) A *Person*, from µService1; (b) An *Immigration Record*, from µService2; (c) A *Criminal Record*, from µService6; (d) A *Financial Transaction*, from µService10. To save space, non-essential attributes were omitted.

In this case study, data was randomly generated from a predefined schema using the *Mockaroo* tool[4]. A total of 1,000 *Person* records were generated, and related instances of the other three classes were randomly generated and associated to persons using one of the two identifying data properties of *Person*: *taxID* and *passportNumber*. After the generation of a single dataset containing all persons, the data was distributed among the microservices, each microservice managing the instances which corresponded to their definition. For example, µ*Service*5 manages only data of arrivals on Asia Immigration departments.

For each generated *Person* record there was a probability of 25% that the person would have no immigration record. For each person that was selected to have immigration records, a uniformly distributed random number between 1 and 20 of records were generated by randomly selecting the

[4]http://www.mockaroo.com/


```
{
  "@context": { "ontology": "http://ontology#" },
  "@type": "ontology:FinancialTransaction",
  "ontology:transactionFrom": "ontology:NorthAmerica",
  "ontology:transactionID": "581-52207-12414-84",
  "ontology:amount": "17528.46",
  "ontology:remmiter": {
    "ontology:taxID": "733-11716-531-23",
    "@type": "ontology:Person",
    "owl:sameAs": "http://.../person?taxId=733-11716-531-23"
  },
  "ontology:beneficiary": {
   "ontology:taxID": "720-72890-123-53",
    "@type": "ontology:Person",
    "owl:sameAs": "http://.../person?taxId=720-72890-123-53"
  }
}
```


Figure 8: A financial transaction representation after Linkedator enrichment


```
{
  "@context": { "ontology": "http://ontology#" },
  "@type": "ontology:Person",
  "ontology:taxID": "733-11716-531-23",
  "ontology:passportNumber": "253-5022-82-43967-856",
  "ontology:firstName": "Nicole",
  "ontology:immigrationRecord": [{
      "@type": "ontology:ImmigrationRecord",
      "owl:sameAs": "http://.../immigration/253-5022-82-43967-856"
    }, { }, { }, { }
  ],
  "ontology:authorOf": [{
      "@type": "ontology:CriminalRecord",
      "owl:sameAs": "http://.../criminal/733-11716-531-23"
    }, { }, { }, { }
  ],
  "ontology:remmiterOf": [{
      "@type": "ontology:FinancialTransaction",
      "owl:sameAs": "http://.../financial/733-11716-531-23"
    }, { }, { }, { }
  ]
}
```

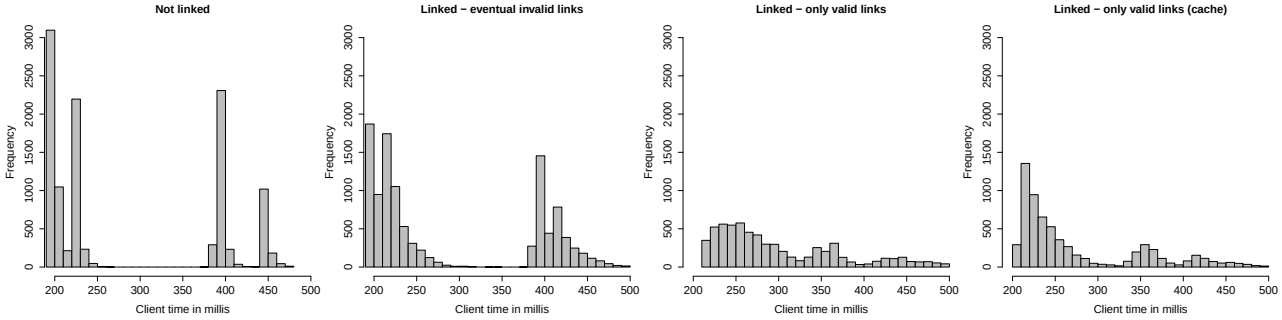

Figure 9: A person representation after Linkedator enrichment

data attributes from lists and predetermined ranges. The same approach was used to generate criminal records and financial transactions. For criminal records, the probability of a person having no record was 50% and the number of records for a person was uniformly distributed between 1 and 5. The probability of a person having no financial transaction was of 25%, while the number of transactions per person was uniformly distributed between 1 and 10.

The result of this process was a single JSON document comprising all persons with nested criminal, financial and immigration records. The instances on this document were distributed across the 11 microservices of Figure 6 according to the criteria associated with each microservice, as previously discussed. After this process, µ*Service*1 outputs *Person* representations such as those in Figure 7 (a) without links to instances of the other classes related to the person. On the other hand, the microservices that provide immigration, financial and criminal records will produce representations that contain blank nodes that include one of the person's identifying data properties. An example can be seen in Figure 7 (b), where the property *traveler* instead of having a link to a *Person*, has a blank node with a *passportNumber*. The blank node represents an anonymous individual, which Linkedator will later identify as being the same individual as a person managed by µ*Service*1.

Table 1: Experiment scenarios' results

Scenario	Linkedator-Core	Linkedator-Jersey	Client	Rep. size	Requests
Not linked	-	-	287.5 ms	891.5 chars	11,000
With invalid links	0.2615 ms	17.02 ms	290.1 ms	1,618.5 chars	11,000
Only valid links	61.505 ms	79.61 ms	304.1 ms	1,501.0 chars	6,484
Only valid links (cached)	38.364 ms	56.834 ms	276.4 ms	1,501.0 chars	6,484



(a) (b) (c) (d)

Figure 10: Client response time distribution (a) Not linked; (b) Linked with occasional invalid links; (c) Linked with only valid links; (d) Linked with only valid links - link validation results cached.

7. EVALUATION

In this section we describe our experimental methodology and 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. In order to allow the replication of experimental results, source code and instructions for setting up this evaluation are available in a public repository[5].

The evaluation is performed through an evaluation question, which aims at finding potential money launderers based on the data managed by microservices that compose the case study. It is important to say that data used in this evaluation is hypothetical and is not related to a real personal record information repository. In order to classify a given person as a potential money launderer, its record must fulfill three characteristics: total amount of financial transactions must be equal or greater than a million dollars, total of declared money on immigration records must be equal or greater than a hundred thousand dollars, and must have a previous criminal record of money laundering.

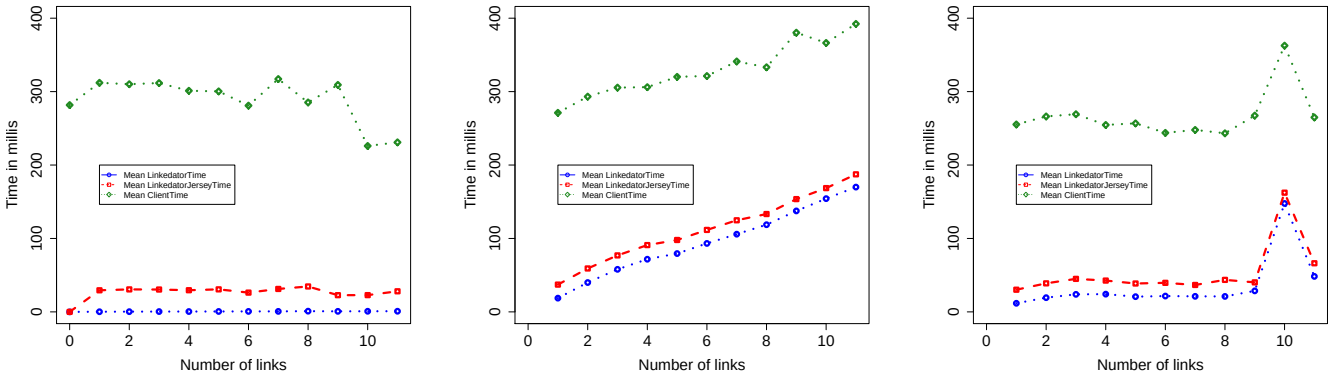
The components of the case study presented in the previous section have been deployed into the Amazon EC2 (Elastic Compute Cloud) environment. Twelve machine instances have been created, a dedicated instance for each microservice, where eleven instances have been used for each microservice implemented in the case study and a dedicated instance for Linkedator-API. 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, running Ubuntu Server 14.04 LTS (HVM) and Oracle Java Development Kit (JDK) 8. The microservices have been configured to use *-Xmx128m -Xms64m*, where flag *Xmx* specifies the maximum memory allocation pool for Java Virtual Machine (JVM), while *Xms* specifies the initial memory allocation pool.

In order to solve the evaluation question, a client application has been developed to interact with microservices. Four different scenarios have been created. In the first scenario there is no microservice composition, therefore the client application must know implementation and deployment details such as URI templates and server addresses. This scenario results in representations not linked, hence the client application must directly perform HTTP requests to each microservice. In the second scenario, microservices are registered in the Linkedator-API and their representations are linked, but with occasional invalid links. In this scenario, the client application only knows the URI to $\mu Service1$ that results in a list of *Person* records enriched with links to other representations. In the third scenario, the Linkedator-API is configured to validate links, which ensures that representations contain only valid links. Finally, in the fourth scenario, the Linkedator-API caches link validation results.

The client application was executed for all scenarios described above and the results are shown in Table 1. This table presents the mean response-time/request for creating links spent in the Linkedator-Core and Linkedator-Jersey as well as the total time perceived by the client. It also presents the mean representation size/request and the number of requests the client application must execute in order to go through all necessary records. One can notice that the link creation process takes a significant amount of time, especially when link validation is enabled. However, the link creation process does not affect the overall client time. On the other hand, the representation size increases 81.547% for the second scenario, which encompasses occasional invalid links, and 68.37% for the third and fourth scenarios, in which there are only valid links.

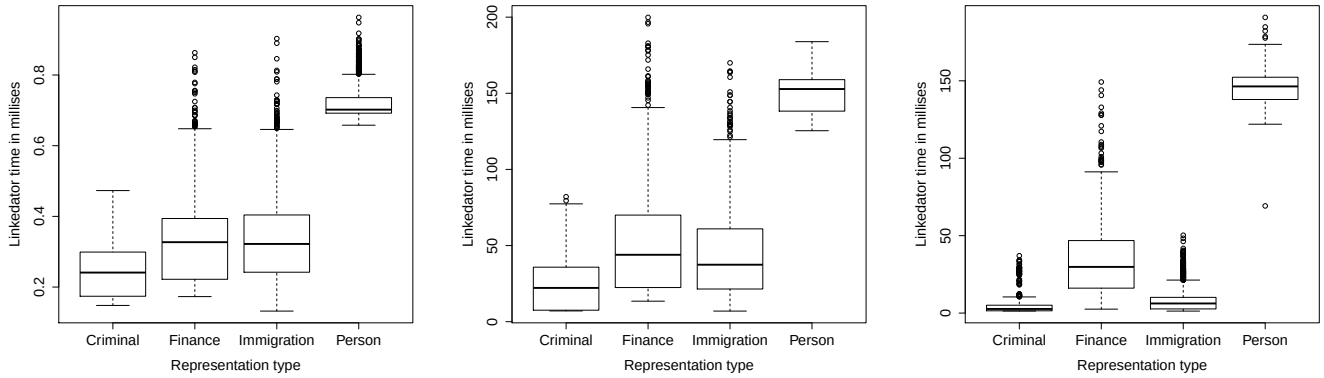
Figure 10 shows how the use of the proposed composition method is perceived by the client with respect to response time distribution. Figure 10 (a) presents the time distribution considering the first scenario (not linked representations). Most requests in this scenario are executed between 200 ms and 250 ms, followed by two spikes in 400 ms and 450 ms. Figure 10 (b) presents the time distribution considering the second scenario (linked representations with eventual invalid links). In this scenario, most requests are

[5]https://drsalvadori.bitbucket.io/Linkedator/iiwas2016



(a) (b) (c)

Figure 11: Impact of number of links on response time: (a) Linked with occasional invalid links; (b) Linked with only valid links; (c) Linked with only valid links - link validation results cached.



(a) (b) (c)

Figure 12: Impact of representation type on response time: (a) Linked with occasional invalid links; (b) Linked with only valid links; (c) Linked with only valid links - link validation results cached.

executed within two ranges, the first one between 200 ms and 300 ms, the second one between 400 ms and 450 ms. Figure 10 (c) presents the distribution of requests executed in the third scenario (representations contain only valid links). One can notice that the link validation process distributes requests more evenly between 200 ms and 300 ms, reducing the spikes observed in previous scenarios. However, the use of cache memory, presented in (d), concentrates the execution of most requests between 200 ms and 250 ms, similar levels achieved in (b).

Figure 11 shows response times considering the number of links evaluated by Linkedator. One can notice that the response time of Linkedator-Jersey is similar to the response time of Linkedator-Core, since all microservices are deployed in the same data center. In (b) due to the need for executing a validation request, the higher the number of links inserted into a representation, the higher the response time. On the other hand, in (c) where cache is enabled, the increase of client response time is not perceived for most of representations. However, there is a spike when representations contain 10 links. The reason for this is directly related to the content of those representations, which fits person records. In the case study, each person record is potentially linked to other 10 representations: four immigration records, four financial transactions and two criminal records. Each of those links is unique, resulting in 10 cache misses given the client behaviour. As a result, person records do not benefit from caching, whereas their linked representations take advantage of reusing links previously validated, as can be seen in Figure 12. Despite the fact that in some cases the cache is not effective, validation of links inside the microservice architecture tends to be faster than validation by clients, which are often positioned at the internet edge, where round-trip time to the microservices is significantly larger.

8. CONCLUSIONS AND FUTURE WORK

This paper proposed a method for composing semantic microservices. In addition, this paper presented Linkedator, a framework that aims to facilitate the adoption of the proposed 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 a set of microservice semantic descriptions. As a result, a given client appli-

cation 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.

By adopting the proposed composition method, splitting up of monolithic applications are facilitated. In this sense, a monolithic application may be split into several microservices and their representations can be linked to one another, reducing the impact of refactoring. However, it demands that client applications be able to deal with linked representations, otherwise refactoring a monolith application will make these clients obsolete.

The evaluation showed that adopting the proposed method by using the Linkedator framework does not influence the overall client response time, considering the case study scenario. However, factors such as link validation, the number of potential links and the use of cache have significant influence on the link creation process. The link validation process has important influence on response time of Linkedator-Core. However, in cases where microservices and the Linkedator-API are deployed in the same infrastructure, it reduces client time, due to link validation within the infrastructure being less expensive than providing occasional invalid links, forcing clients to waste time.

In future work we intend to include similarity-based techniques to identify partial matches between representations. While many works from entity linking include some degree of uncertainty, features of ontologies, such as functional and inverse function properties, may be used to increase the certainty of these works. The possibility of link creation could be maximized associating multiple Linkedator-API nodes to form a federation, in which microservices can delegate validation to other federated nodes whenever appropriate.

9. REFERENCES

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

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

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

[4] C. Bizer, T. Heath, K. Idehen, and T. Berners-Lee. 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, 2008. ACM.

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

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

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

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

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

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

[11] W. Hu, J. Chen, and Y. Qu. 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, 2011. ACM Press.

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

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

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

[15] M. Lanthaler and C. Gütl. 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, 2012. ACM.

[16] R. P. Magalhães, J. M. Monteiro, V. M. P. Vidal, J. A. F. de Macêdo, M. Maia, F. Porto, and M. a. Casanova. 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, 2013.

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

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

[19] S. Newman. *Building Microservices*. O'Reilly Media, Gravenstein Highway North, Sebastopol, CA. USA, 2015.

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

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

[22] H. Stoermer, N. Rassadko, and N. Vaidya. 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, 2010. Springer-Verlag.