Computer Standards & Interfaces 50 (2017) 124–141

Contents lists available at ScienceDirect

Computer Standards & Interfaces

journal homepage: www.elsevier.com/locate/csi

A domain independent readability metric for web service descriptions

Alan De Renzis[a,1,*], Martin Garriga[a,1], Andres Flores[a,1], Alejandra Cechich[a], Cristian Mateos[b,1], Alejandro Zunino[b,1]

[a] *GIISCO Research Group, National University of Comahue, Neuquén, Argentina*
[b] *ISISTAN Research Institute, UNICEN University, Tandíl, Argentina*

ARTICLE INFO

Keywords:
Readability
Web service descriptions
WordNet
Domain independent

ABSTRACT

Web Services are influencing most IT-based industries as the basic building block of business infrastructures. A Web Service has an interface described in a machine-processable format (specifically WSDL). Service providers expose their services by publishing the corresponding WSDL documents. Service consumers can learn about service capability and how to interact with the services. Service descriptions (WSDL documents) should be ideally understood easily by service stakeholders so that the process of consuming services is simplified. In this work we present a practical metric to quantify readability in WSDL documents – attending to their semantics by using WordNet as the underlying concept hierarchy. In addition, we propose a set of best practices to be used during the development of WSDL documents to improve their readability. To validate our proposals, we performed both qualitative and quantitative experiments. A controlled survey with a group of (human) service consumers showed that software engineers required less time and effort to analyze WSDL documents with higher readability values. Other experiment compares readability values of a dataset of real-life WSDL documents from the industry before and after modifying them to adhere to the readability best practices proposed in this paper. We detected a significant readability improvement for WSDL documents written according to the best practices. In another experiment, we applied existing readability metrics for natural language texts detecting their unsuitability to the Web Service context. Lastly, we analyzed the readability best practices identifying their useful applicability to the industry.

1. Introduction

The Service-Oriented Computing (SOC) paradigm has experienced an ever-increasing adoption since it provides support for building distributed, inter-organizational applications in heterogeneous environments [1]. Mostly, the software industry has adopted SOC by using Web Service technologies. A Web Service is a program with a well-defined interface that can be published, located and invoked by using standard Web protocols [2].

Typically, service interfaces and communication protocols are represented in the form of WSDL[2] (Web Services Description Language) documents. WSDL is an XML-based language for specifying service descriptions. A WSDL document is defined by a service provider, optionally published in a service registry, and then used by service consumers both to figure out service capabilities and to establish interaction among consumer applications and the specified Web Service. The goal of WSDL is that service consumers can reason about the functionality of a Web Service without knowing service implementation details and by only looking at its associated WSDL document [3]. Intuitively, this can be achieved provided that the document is readable/understandable, i.e., it does not suffer from specification problems that might obscure the purpose of the service to end users. Moreover, when such documents are published in a registry, services must be discovered prior to be consumed. The process of discovering Web Services requires automatically matching a query against potential candidate services in a registry, and then manually inspecting retrieved candidates to select the desired result. For this process to be effective, service providers must supply meaningful WSDL documents upon developing services [4].

Particularly, from a linguistic point of view, readability is defined as "*the level of ease or difficulty with which text material can be understood by a particular reader who is reading that text for a specific purpose*" [5]. In the context of Web Services, service descriptions (WSDL documents) should be ideally understood easily by service

* Corresponding author.
E-mail address: alanderenzis@fi.uncoma.edu.ar (A. De Renzis).
[1] Also, National Scientific and Technical Research Council (CONICET).
[2] http://www.w3.org/TR/wsdl

http://dx.doi.org/10.1016/j.csi.2016.09.005
Received 14 June 2016; Received in revised form 13 September 2016; Accepted 14 September 2016
Available online 28 September 2016

stakeholders so that the process of consuming services is simplified. Then, producing readable service descriptions can be used as a strategy by service providers to attract service consumers [6]. Current research in the field is indeed compatible with the idea of generating readable service descriptions [7,8], since they address other relevant quality attributes such as maintainability [9], discoverability [10] and adaptability [11].

However, quantifying readability and hence providing a metric broadly applicable for WSDL documents is still a major challenge. Previous works in the field [12,6] require a service domain ontology – structured as a concept hierarchy modeled via languages such as OWL[3]/RDF[4] – to compute readability. Nevertheless, the lack of complete and relevant domain-specific ontologies hinders the applicability of ontology-based approaches in practice [13,14]. Ontologies aim to describe all relevant aspects related to general services which are accessible through a Web Service interface, with the ultimate goal of enabling the (total or partial) automation of the Web Service lifecycle (e.g., discovery, composition, monitoring, etc.) [15]. However, domain-specific ontologies are often avoided in real life applications because they are difficult to specify [16].

Therefore, the main contribution of this paper is to propose a practical metric to quantify readability in WSDL documents, and a set of best practices to improve WSDL readability. To this end, we adapt and extend the semantic readability metric proposed in [12] to focus on WSDL documents by using WordNet [17] as the underlying concept hierarchy. WordNet is a domain-independent lexical database of the English language. Thus, a domain-specific, "heavy" ontology for each domain is not necessary. In addition, we propose a set of best practices that can be used during the development of WSDL documents to improve their readability according to the proposed metric. Following the rationale adopted in previous work [18,3,19], we provide the readability metric along with a best practices catalog for developers to quantify and improve the readability of their WSDL documents. The readability metric is applied over services descriptions, truly accessible to service consumers, and does not consider their implementation code – which is unavailable to third parties.

We performed both qualitative and quantitative experiments to validate our metric and best practices on readability. The first experiment analyzes the applicability/utility potential of readability notions in the industry. For this, we measured the lack of readability best practices in a dataset of industry Web Services crawled from the Mashape.com[5] online service repository. In fact, at the time of writing this article, this is the largest dataset of Web Services available for academic and industry purposes. The second experiment is a controlled survey with a group of (human) service consumers. They were asked to analyze a dataset of WSDL documents to determine whether or not they were readable in general terms, i.e., without considering our metric. The survey results were compared with the readability values obtained by applying the readability metric. The third experiment compares readability values of a dataset of real-life WSDL documents from the industry before and after modifying them to adhere to the readability best practices proposed in this paper. The last experiment compared existing readability metrics for natural language text with regard to our metric. We detected that those text-based metrics do not perform well for the Web Service context.

The rest of this paper is organized as follows. Section 2 introduces basic concepts on Web Services and readability. Section 3 presents the Web Services centered readability metric and the set of best practices proposed in this work. Section 4 presents the experiments to validate our proposals. Section 5 discusses relevant related work. Conclusions and future work are presented afterwards.

[3] Web Ontology Language. https://www.w3.org/2001/sw/wiki/OWL

[4] Resource Description Framework. https://www.w3.org/RDF/

[5] http://mashape.com

2. Background

A service development life-cycle, as any other regular kind of software component, consists of several phases. Within these, the service design phase comprises service interface specification using WSDL. Several important concerns, such as granularity, cohesion, discoverability and reusability, should influence design decisions to result in good service interface designs [20]. Many of the problems related to the efficiency of standard-compliant approaches to service discovery stem from the fact that the WSDL specification is incorrectly or partially exploited by providers [21].

2.1. Web Service Description Language (WSDL)

When employing Web Services, a provider describes each service technical contract, a.k.a. its interface, in WSDL. WSDL is an XML-based language designed for specifying service functionality as a set of abstract operations with inputs and outputs, and to associate binding information so that consumers can invoke the offered operations. WSDL allows providers to describe two parts of a service: what it does (its functionality) and how to invoke it. The first part reveals the service interface that is offered to consumers, while the second part specifies technological aspects, such as transport protocols and network addresses. Consumers use the functional descriptions to match third-party services to their needs, and the technological details to invoke the selected service [22].

With WSDL, service functionality is described as one or more port-types which arrange different operations that exchange input and output messages. Main WSDL elements, such as *port-types*, operations and messages, must be labeled with unique names. Optionally, these WSDL elements might contain documentation as comments. Messages consist of parts that transport data between consumers and providers of services, and vice-versa. Exchanged data is represented using XML according to specific data-type definitions in XML Schema Definition (XSD), a language to define the structure of an XML element. XSD offers constructors to define simple types (e.g., `integer` and `string`), restrictions, and both encapsulation and extension mechanisms to define complex elements. XSD code might be included in a WSDL document using the *types* element, but alternatively it might be put into a separate file and imported from the WSDL document or even other WSDL documents. Fig. 1 shows the structure of a WSDL document that includes the *types* element.

2.2. Plain-text oriented readability

The original readability model from [12], which is based upon "heavy" ontologies, computes two main features of a plain-text document, namely document *scope* and document *cohesion*, according to the presence/absence of domain terms in a document. The vocabulary of the domain is structured as a concept hierarchy, where the more specific the terms, the deeper they appear in the hierarchy. Terms in the document which have a match in the concept hierarchy are regarded as *domain concepts*, otherwise they are *non-domain concepts*. The concept hierarchy that captures the domain terms can be a domain-specific ontology or taxonomy, either defined by domain experts [23] or obtained by exploring specialized Web sites.[6,7] The model also considers *difficulty* (of comprehension) of single words in a document.

The initial feature named *scope* is defined as the coverage of domain-specific concepts in the document. The original model calculates the scope considering the *average depth* of all the concepts in the document. If the *average depth* is high, the document scope will be

[6] http://protegewiki.stanford.edu/wikiIProtegeOntology_Library/–

[7] http://lov.okfn.org/dataset/lovlindex.html

```
<?xml version="1.0"?>

<definitions>
   <types>
      definition of types........
   </types>
   <message>
      definition of a message....
   </message>
   <portType>
      <operation>
         definition of an operation....
      </operation>
   </portType>
   <binding>
      definition of a binding....
   </binding>
   <service>
      definition of a service....
   </service>
</definitions>
```

Fig. 1. WSDL structure.

low. Thus, the document will be less readable. Unfortunately, the original model does not provide any range or threshold to determine “high” and “low” *depth* values.

Then, the *cohesion* feature refers to how much focused is the document on a particular topic. It can be computed as the shortest path among each pair of terms according to the given concept hierarchy. This reflects the semantic relation among the domain terms in the document. According to the model, the more cohesive the domain terms in the document, the more readable the document.

Finally, the *difficulty* feature is related with the comprehension of words in a document. It is calculated by using a traditional readability formula: the *Dale–Chall's readability index* [24]. This formula indicates the percentage of *difficult* words in the document using a growing list of approximately 3000 *familiar* (non-difficult) words.

Formula (1) shows the overall Concept-based Readability Score (CRS), which is calculated upon *scope*, *cohesion* and the *Dale–Chall's readability index* (DaCw):

$$CRS(d) = Scope(d) + Cohesion(d) + DaCw(d)^{-1} Scope(d) = e^{-\left(\sum_{i=1}^{n} depth(c_i)\right)} \quad Cohesion(d) = \frac{\sum_{i,j=1}^{n} sim(c_i, c_j)}{number\ \ Of\ \ Associations}$$

$$DaCw(d) = \% - Of - Difficult - Words \tag{1}$$

For more details of the concept-based readability score we refer interested readers to [12].

Example: To illustrate the original readability model, next we describe a simple example involving a document*d* composed by three words: `compact`, `motorcar` and `truck`. We consider the concept hierarchy presented in Fig. 2.

- *Scope*: The depth of the word `compact` is 3, while the depth of `motorcar` and `truck` is 2. Therefore the scope is: $scope(d) = e^{-(3+2+2)} = 0.000911$.
- *Dale–Chall's Index*: The words `compact` and `truck` do not belong to the Dale–Chall's list, then the percentage of *difficult* words is 66%.
- *Cohesion*: The cohesion value is calculated as follows:

$$cohesion(d) = \frac{\sum_{i,j=1}^{3} sim(c_i, c_j)}{3} = \frac{sim(compact, motorcar) + sim(compact, truck) + sim(motorcar, truck)}{3} = \frac{0.77 + 0.3 + 0.48}{3} = 0.52$$

- Readability value: finally, the readability value is calculated as follows:

$$CRS(d) = 0.000911 + 0.52 + 66^{-1} = 0.536$$

CRS applied to WSDL: An existing approach in the context of readability for Web Services [6] has applied the notions defined in the original model defined by [12] upon WSDL documents in a straightforward way. In this approach, all words included in a WSDL document are analyzed to determine if they are domain terms with respect to a domain-specific ontology previously selected. However, from the analysis of the readability model, we conclude the following issues:

- The original readability model strongly depends on a concept hierarchy (in the form of a domain-specific ontology). However, it is known that the task of building a specific ontology for a domain requires a large effort even by skillful developers [14,13,7]. This hinders the adoption of such readability model in practice.
- The original readability model targeted plain-text documents. Thus, it is unaware of the structural information included in a WSDL document. Structural information involves each comprising part of a WSDL document, namely operations, port types, messages, data type definitions, etc. As this information is known a priori, it is possible and desirable to analyze it when calculating readability. Then, the overall comparison extent is limited to only compare certain subsets of relevant terms.

3. Readability on web services

This section presents the Web Service centered implementation of a readability metric for WSDL documents. This metric allows us to compare WSDL documents according to their readability values. Applied in the context of service discovery and inspection, the readability value enables the (automatic) discovery of services which are more readable, further easing the adoption of Web Services in practice. In addition, we present a set of best practices, which can benefit providers during the development of WSDL documents to offer services with higher readability – according to the proposed metric.

3.1. Web service centered readability

To overcome the issues mentioned in Section 2.2, we decided to

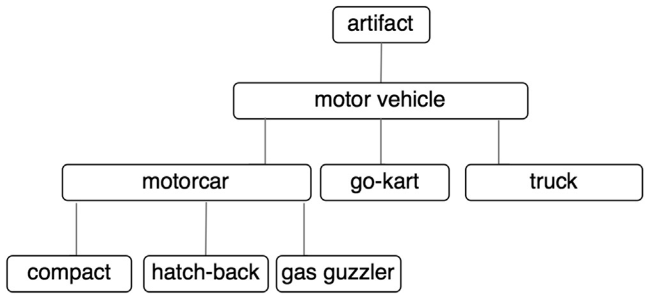


Fig. 2. Vehicles hierarchy.

adapt the original readability model to the context of Web Services, particularly for WSDL documents. In addition, we decided to use WordNet [17] as the underlying concept hierarchy. WordNet is a domain-independent lexical database of the English language that is structured as a lexical hierarchy – a tree with root node {entity}. Terms are grouped in synsets (synonym sets) that represent the same lexical concept. Several relationships connect different synsets, such as hyperonymy/hyponymy, holonymy/meronymy and antonymy. The hierarchy links more general synsets like {`furniture`, `piece_of_furniture`} to increasingly specific ones like {`bed`} and {`bunkbed`}. Thus, the category `furniture` includes `bed`, which in turn includes `bunkbed`; conversely, concepts like `bed` and `bunkbed` make up the category `furniture`. To calculate relationships between concepts, we use the JWNL Java library.[8]

Using WordNet enables to exploit as many information as possible gathered from WSDL documents, without requiring any extra semantic specification for services, such as ontologies, usually unavailable in practice. Also, as WordNet is domain-independent, its concept hierarchy can be used to assess WSDL documents coming from any domain category. In fact, current Web Service repositories such as Mashape.com and ProgrammableWeb.com organize published services in dozens of categories as diverse as Messaging, Advertising, Real Estate, Security and Financial, just to name a few.

In our approach, every term included in a WSDL document is analyzed through WordNet. The original model (Section 2.2) defines domain and non-domain terms. In our Web Service centered model, for a given WSDL document, those terms which belong to the WordNet dictionary are regarded as *existing terms* (equivalent to domain terms in the original model), thus they are used as an input to calculate the readability value; otherwise they are *non-existing terms* (i.e., non-domain terms in the original model) and then they are left out from the readability calculation. It is worth noticing that discarding non-existing terms do not affect the overall readability results, since considering the real-life documents crawled from Mashape.com (which will be also used in the experiments in Section 4), the ratio of existing terms to non-existing terms is 20 to 1. This is because WordNet is domain-independent and contains most words of the English language.

As stated, using WordNet as the underlying concept hierarchy enables considering terms belonging to different domains for the readability calculation. We considered that any domain-specific term that is not recognized by WordNet is a non-existing term.

Each operation in a WSDL document is individually analyzed to calculate its readability value. Then, the average value for all operations in a WSDL document constitutes the overall readability value of the service. This value represents the level of ease or difficulty with which a WSDL document can be understood by a developer. In other words, the difficulty for a human reader to understand the only specification available for a given Web Service.

Fig. 3 depicts the steps for calculating the Web Service centered Readability value, which are detailed in the following sections.

3.1.1. Concept extraction

In the concepts extraction step, all terms from words and identifiers from WSDL documents are pre-processed and identified according to WordNet.

To parse the WSDL documents, we use the Java API of the Membrane SOA model.[9] The extracted terms belong to identifiers included in specifics elements (parts) in WSDL documents: Definition, Data types, Message, Operation, Port type, Port and Service. A list of terms is extracted of each operation part, e.g., a list of message terms, a list of sequence terms, and a list of data type terms, among others.

To extract terms from identifiers, we make use of an algorithm for semantic separation of terms defined in a previous work [25]. Identifiers are normally restricted to a sequence of one or more letters in ASCII code, numeric characters and underscores ("_") or hyphens ("-"). The algorithm supports the rules in Table 1 – i.e., the usual programming naming conventions. A semantic level has been added to be aware of cases that do not follow those conventions. The Term Separation algorithm analyzes the operation identifiers, recognizing potential terms (uppercase sequences and lowercase sequences) as explained in the example below.

Given an identifier, the term separation algorithm first removes all numbers from the identifier. Then, the algorithm analyses each character from detected terms inside the identifier – i.e., uppercase sequences and lowercase sequences. If the algorithm is analyzing a lower case sequence, when an uppercase letter or a special character is detected (as "_" or "-"), the previous lowercase sequence is added to the resulting terms. In the case when an uppercase sequence is being analyzed, when a lowercase letter is detected, two possibilities are considered:

1. The last uppercase letter is the first letter of a term composed by the uppercase letter and the lowercase letters after it.
2. The last uppercase letter is part of a possible acronym or abbreviation composed by the uppercase sequence detected.

Then, WordNet is used to analyze all the potential terms and determine the most adequate term separation:

1. For the first case, if the resulting term belongs to WordNet dictionary, then this is a term and the previous uppercase sequence (without the last uppercase letter) is an acronym or abbreviation.
2. If not, it is the second case, where the uppercase letter belongs to the acronym composed by whole uppercase sequence and the lowercase sequence is other term itself. This situation is considered even when it does not follow the naming convention.

Example: Let be the identifier "`OMSLogin`". This identifier does not strictly follow the Java Bean notation. The first analysis gives an uppercase sequence (`OMSL`), and a lowercase sequence (`ogin`). Then, the sequence `L + ogin=Login` is analyzed with WordNet; this is an existing word in the WordNet dictionary. This determines that `Login` is a term and `OMS` is an acronym (an abbreviation of Online Messenger Service) that is also considered as a term.

All possible terms from a WSDL document are mapped to concepts in the WordNet hierarchy. Those terms that do not belong to the WordNet dictionary are considered as *non-existing* terms. The outcome of this step is a collection including the lists of terms of all WSDL elements for each operation. This structured collection is crucial to calculate the Document Cohesion feature, which is explained latter in this section.

Notice that lemmatization is implicitly performed when analyzing *existing* concepts. Lemmatization (stemming) is a useful technique in natural language processing, used to retrieve the base, stem or root form of a word. WordNet provides a built-in stemmer that is implicitly used by WordNet operations such as getSynonyms() and getHyponyms().

3.1.2. Document scope

The original readability model (Section 2.2) defines document scope feature as the coverage of domain concepts in a document. As the number of domain-specific concepts in a document increases, the readability of the document decreases. However, we have redefined this feature for the context of Web Services, in particular considering WSDL documents:

- Using specific terms in WSDL documents reduces ambiguity about

[8] https://web.stanford.edu/class/cs276a/projects/docs/jwnl/javadoc/
[9] http://www.membrane-soa.org/soa-model-doc/1.4/java-api/parse-wsdl-java-api.htm

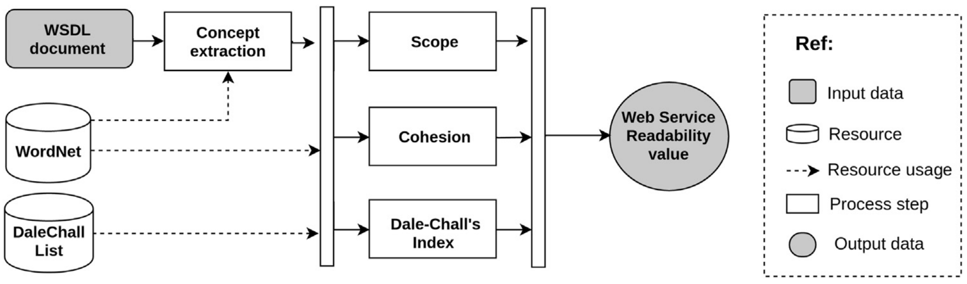


Fig. 3. Steps for calculating the web service centered readability value.

Table 1
Rules for decomposing identifiers.

Notation	Rule	Source	Result
JavaBeans	Splits when changing text case.	`getZipCode`	`get Zip code`
Special tokens	Splits when either "_" or "-" occurs.	`Get_Quote`	`Get Quote`

service functionality and makes them self-explanatory [21]. Thus, we consider that WSDL documents with more domain-specific terms are more readable.

- As discussed in Section 2.2, the approach in [6] applies the readability formula from the original model directly to WSDL documents. However, the obtained values for document scope in their case studies are too small, very close to zero (for example 2.31^{E-92}). Then, this value is insignificant to affect the calculation of the total readability value (as can be seen in Formula (6)).

All in all, we redefined the *scope* as the specificity degree of the concepts in a WSDL document. The deeper the concepts of a document appear in the WordNet hierarchy, the more readable the document is, since the concepts will be more specific and ambiguity will be less likely [26]. The scope of a document *d* is computed according to Formula (2), as the average depth of all concepts in the document divided by the maximum depth of the concept hierarchy – i.e., 16 in the case of WordNet. Scope values range between 0 and 1:

$$Scope(d) = \frac{AverageDepth(d)}{MaxTreeDepth} \tag{2}$$

where *MaxTreeDepth* is the maximum depth of the WordNet hierarchy:

$$AverageDepth(d) = \frac{\sum_{i=1}^{n} depth(t_i)}{n}$$

where $depth(t_i)$ is the depth of concept t_i extracted from the document *d* w.r.t. the WordNet hierarchy and *n* is the total amount of existing terms.

Example: Fig. 4 shows an excerpt of the WordNet hierarchy where, for example, the depth of the concept `enclosure` is 4, and the depth of the concept `birdcage` is 6. These values indicate that the concept `birdcage` is more specific than the concept `enclosure`, and then the use of `birdcage` might prevent a misinterpretation.

3.1.3. Document cohesion

Document Cohesion refers to how much focused a document is on a particular topic. Similarly to document scope, we redefined *cohesion* for the Web Services context. The original readability model analyzes the relations among each pair of concepts included in a plain-text document. Since a WSDL document is a structured document, we decided to exploit such structure. For this, we defined *Document Cohesion* as the average operation cohesion considering each operation included in a WSDL document. Operation cohesion is computed considering the semantic relationship between the existing terms in an operation signature, which is reflected by the shortest path among such terms in the WordNet concept hierarchy. The signature of an operation includes the following parts: *operation*, *input*, *output* and *types*. We do not consider the relationship between terms belonging to a single isolated part. Instead, operation cohesion assesses the relationship between terms from different parts of the operation. For example, terms in a particular *type* definition are not compared to each other. This is because we assume that all terms in the type definition help to describe a domain aspect, but they are not necessarily related to each other.

Operation Cohesion is calculated according to Formula (3), where *n* is the number of terms in a given operation *O*, R_n is the number of relationships between these terms, $length(t_i, t_j)$ is the shortest path between t_i and t_j in the WordNet hierarchy, and *MaxTreeDepth* is the maximum depth of the WordNet hierarchy (16). The *opCohesion* value increases as the *length* between the concepts decreases:

$$opCohesion(O_i) = \frac{\sum_{i,j=1}^{n} sim(t_i, t_j)}{R_n} \tag{3}$$

where $n > 1,\ \ i < j$, and

$$sim(t_i, t_j) = -\log\frac{length(t_i, t_j)}{2 * MaxTreeDepth}$$

Finally, document cohesion is calculated according to Formula (4), where *n* is the number of operations in document *d*, and *w* is the weight given to document cohesion. Essentially, the more cohesive the domain terms in the document, the more readable the document. Thus, in the context of this work, we defined *w*=5 to reflect the importance of cohesion in the readability calculation:

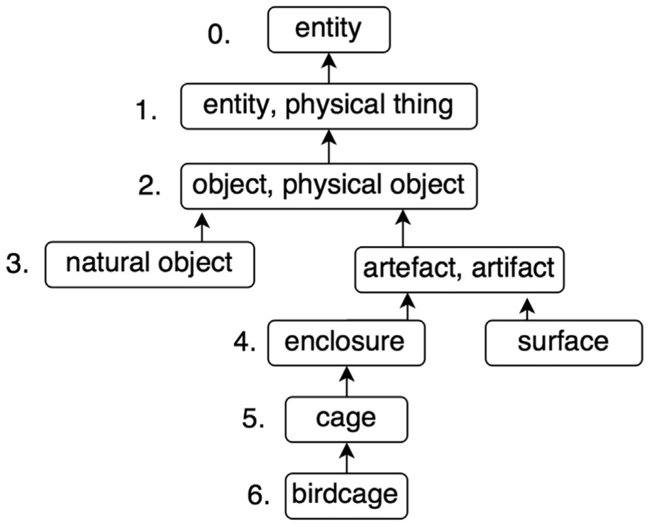


Fig. 4. WordNet hierarchy example.

$$cohesion(d) = w*\frac{\sum_{i=1}^{n} opCohesion(o_i)}{n} \quad (4)$$

Example: Let us consider a WSDL document from the Car Rental domain. Fig. 5 shows an excerpt of the WordNet hierarchy containing concepts from such domain. The *length* between the concepts `compact` and `truck` is 3, and the length between `compact` and `motor vehicle` is 2. These values indicate that `compact` is more similar to `motor vehicle` than to `truck`. Thus, the joint use of the domain concepts `compact` and `motor vehicle` in an operation name or an operation comment, will be more cohesive than the joint use of `compact` and `truck`.

3.1.4. Simplified Dale–Chall's Readability Index

The simplified Dale–Chall's readability index is a formula that considers the percentage of difficult words in a document [24]. The rationale behind this index is that the length of the sentences in a document and the difficulty of words are correlated to the overall difficulty of reading and understanding the document. Since the concept-based readability model measures readability at word level, sentence-level complexity is not applicable and hence only word difficulty is considered. Words in the document are identified as either *familiar* or *unfamiliar* (difficult) words. That is, words are familiar if they can be found in the Dale–Chall's list. Otherwise, words are regarded as unfamiliar or difficult. The Dale–Chall's readability index of a document *d* can be computed according to Formula 5, and ranges from 0 to 1:

$$DaCw(d) = \frac{100 - PDW}{100} \quad (5)$$

where PDW is the percentage of difficult words among the terms extracted from *d*.

3.1.5. Web service readability metric

The Web Service centered readability (WSCR) metric for a WSDL document *d* determines the readability score by considering the three features explained above: the document scope, the document cohesion and the simplified Dale-Chall's index. Readability is computed according to Formula 6. Theoretically, the lowest readability value is *WSCR*=0, which is practically impossible to find in an actual WSDL document (with *Scope*=0, *Cohesion*=0 and *DaCw*=0). The maximum readability value depends on the weight given for cohesion in Formula 4. In the context of this work, the maximum *WSCR=7,* but is also impossible to find in an actual WSDL document (with maximum *Scope*=1, *Cohesion*=5 and *DaCw*=1):

$$WSCR(d) = Scope(d) + Cohesion(d) + DaCw(d) \quad (6)$$

Example: Let us consider a simple WSDL document for the apartment rental domain with one operation, as shown in Fig. 6. The first step for the readability calculation is *concepts extraction*:

- operation terms: [`reserve`, `apartment`, `breakfast`]
- message terms: [`reserve`, `apartment`, `breakfast`, `request`]
- type terms: [`apartment`, `breakfasts`, `meal`, `id`, `date`, `housing`, `description`, `rooms`, `number`]

To calculate the *scope*, it is necessary to calculate the depth of each concept. For example:

$$depth(apartment) = 7$$

$$depth(breakfast) = 8$$

$$depth(room) = 5$$

Considering all the concepts in the operation definition, the total number of terms is 16. Then, the scope is calculated as follows:

$$AverageDepth(d) = \frac{5 + 6 + \cdots}{16} = 7.18$$

$$Scope(d) = \frac{7.18}{MaxTreeDepth} = 0.44$$

being *MaxTreeDepth*=16.

On the other side, the *Simplified Dale–Chall's Readability Index*

$$DaCw(d) = \frac{100 - 18}{100} = 0.82$$

since an 18% of the terms are not included in the familiar words list.

Finally, the *cohesion* is calculated by comparing terms that are not in the same WSDL section. For example, the concept `room` from the list of 'type' terms is compared with the concept `apartment` from the list of 'operation' terms. Then,

$$length(room, apartment) = 4$$

– given by the shortest path between `room` and `apartment` in the WordNet hierarchy. Then,

$$sim(room, apartment) = -\log\frac{4}{2*MaxTreeDepth}$$

where *MaxTreeDepth*=16 is the maximum depth of the WordNet tree. Therefore,

$$sim(room, apartment) = 0.9$$

Considering the concepts `meal` and `breakfast`, $length(meal, breakfast) = 1$ since `meal` is an hyperonym of `breakfast`. Then, $sim(meal, breakfast) = 1.5$.

In the last step, the operation cohesion is calculated as:

$$cohesion(op) = \frac{1.5 + 0.9 + \cdots}{71} = 1.3$$

where 71 is the total number of concept relationships. Recall that the *document cohesion* is the average of operation cohesion values. In this case, it is 1.3 because the WSDL document is composed by only one operation.

Finally the *Web Service centered readability (WSCR)* is calculated as follows:

$$WSCR(d) = Scope(d) + Cohesion(d) + DaCw(d) = 0.44 + 1.3 + 0.82 = 2.56$$

Initially, 2.56 may appear as a low value in readability terms, considering a maximum value of 7. However, to obtain the maximum value, three conditions are necessary:

- All WSDL term depths have to be 16 – i.e., the maximum tree depth of WordNet hierarchy.
- The distances between terms have to be 0 – i.e., equal terms or synonyms.
- All terms have to belong to *Dale–Chall's* list.

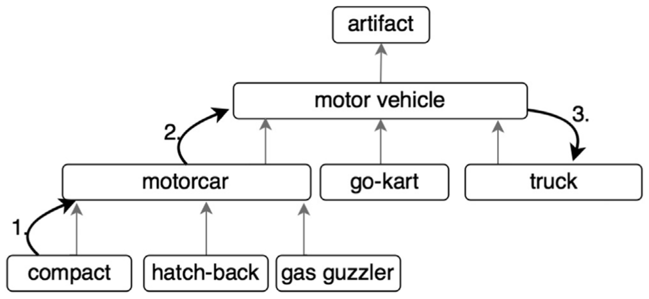
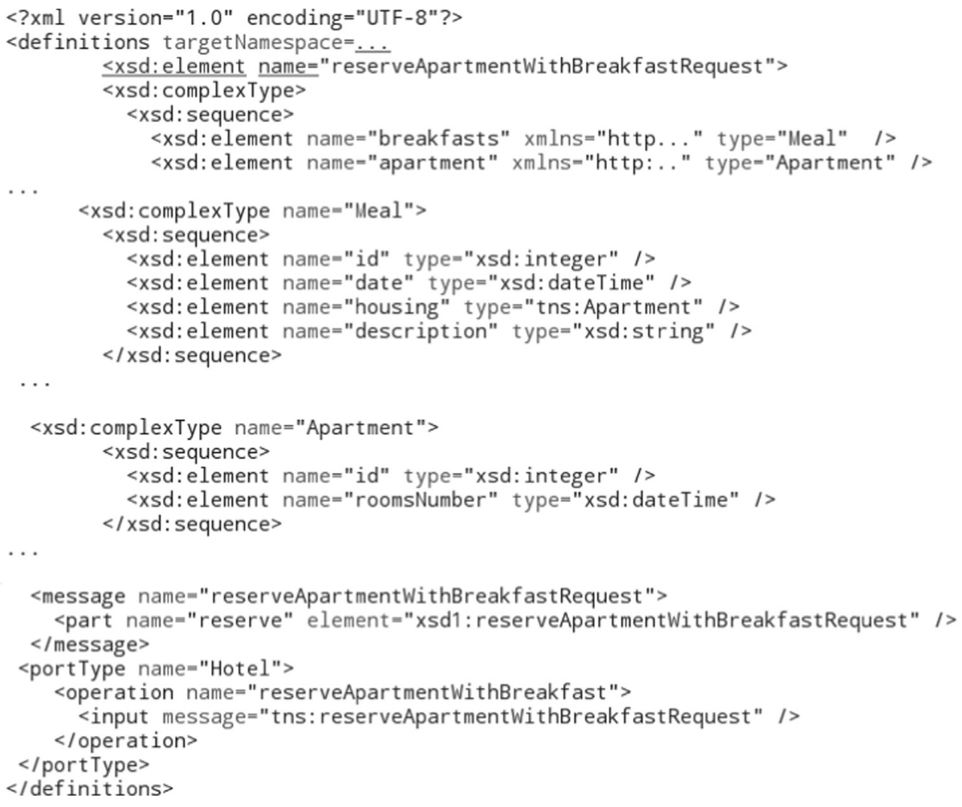


Fig. 5. Length between `compact` and `truck` in the WordNet hierarchy.



```
<?xml version="1.0" encoding="UTF-8"?>
<definitions targetNamespace=...
        <xsd:element name="reserveApartmentWithBreakfastRequest">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="breakfasts" xmlns="http..." type="Meal"  />
            <xsd:element name="apartment" xmlns="http:.." type="Apartment" />
...
      <xsd:complexType name="Meal">
        <xsd:sequence>
          <xsd:element name="id" type="xsd:integer" />
          <xsd:element name="date" type="xsd:dateTime" />
          <xsd:element name="housing" type="tns:Apartment" />
          <xsd:element name="description" type="xsd:string" />
        </xsd:sequence>
 ...

  <xsd:complexType name="Apartment">
        <xsd:sequence>
          <xsd:element name="id" type="xsd:integer" />
          <xsd:element name="roomsNumber" type="xsd:dateTime" />
        </xsd:sequence>
...

  <message name="reserveApartmentWithBreakfastRequest">
    <part name="reserve" element="xsd1:reserveApartmentWithBreakfastRequest" />
  </message>
 <portType name="Hotel">
    <operation name="reserveApartmentWithBreakfast">
      <input message="tns:reserveApartmentWithBreakfastRequest" />
    </operation>
 </portType>
</definitions>
```

Fig. 6. WSDL document for apartment rental domain.

These conditions are not possible in a real-life WSDL document. Therefore, a WSDL document with a readability value of 2.56 is actually readable, but it can certainly be improved.

3.2. Readability best practices

No silver bullet can guarantee that potential consumers of a Web Service will effectively discover, understand and access it [27]. However, the work in [21] has empirically shown that a WSDL document can be improved to simultaneously address these issues, by replacing cryptic WSDL element names with explanatory ones, among other practices for WSDL revision [4].

For this reason, in this paper we address the quality of WSDL documents from the readability perspective. This section presents some guidelines or best practices to improve WSDL documents in terms of readability according to our model. Table 2 presents a list of readability best practices, to improve existing WSDL documents. We acknowledge the following information about each practice:

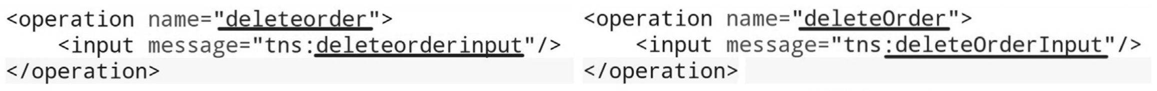
- Practice: a descriptive name for the practice.
- Description: a brief description of the causes that raise problems to be solved by applying the practice.
- Manifestation: A binary rank of the manifestation form of the practice. We refer as *evident* practices those that manifest themselves in the syntax of the WSDL document. On the contrary, *not immediately apparent* practices are those that require not only to analyze the syntax of the document, but also its semantics to detect their manifestation form.

Proper naming conventions: Evidence suggests that using proper naming conventions is a practice that improves readability of WSDL documents [28,29]. Sometimes, programmers tend to use names of data types and variables such as `getcustomer` and `createaccount`, making the code less readable. Also, neglecting camel casing or special tokenization characters (as the ones presented in Table 1) in identifiers with many terms decreases document readability, because it is intuitively more difficult to visually split words upon reading a single term.

Fig. 7 shows an excerpt of a WSDL document where the operation name was refactored to follow naming conventions (i.e., camel case).

Table 2
WSDL readability best practices.

Practice	Description	Manifest
Proper naming conventions	Identifiers do not follow naming conventions: it is convenient to adopt proper naming conventions such as JavaBeans for identifiers	Evident
Descriptive identifiers	Operations, messages and parameters names must reflect their purpose	Not immediately apparent
Specific concepts	Specific concepts reduce the ambiguity, as using identifiers composed by ambiguous or generic terms reduces understandability	Not immediately apparent
Non-cryptic parameter names	Parameter names that describe the conveyed data and use (in/out). The use of parameter names such as `arg1` or `param` or `parameter` is discouraged	Evident
Message names related to operations	Operations and their messages must have related identifiers	Not immediately apparent
Familiar acronyms and abbreviations	Self-explanatory identifiers instead of unfamiliar acronyms and abbreviations	Evident



```
<operation name="deleteorder">
    <input message="tns:deleteorderinput"/>
</operation>
```

(a) Original

```
<operation name="deleteOrder">
    <input message="tns:deleteOrderInput"/>
</operation>
```

(b) Refactored

Fig. 7. Proper naming conventions example.

Descriptive identifiers: Descriptive identifiers consists of using names that accurately represent the functionality of an operation, or the data conveyed by a message or parameter. This practice will help readers to understand what the service offers. For example, let us consider the operation identifier `getApartments` from the Hotel domain that returns all free apartments between a date range. Fig. 8a presents its associated WSDL code. This identifier does not precisely describe the intended functionality, because the same identifier could also describe an operation that returns all apartments in the hotel. In other words, the identifier is ambiguous and too broad in terms of the functionality described. In addition, according to the WSDL part in which the identifier is defined, it should be syntactically correct. For example, the first terms in an operation name should be in the form: <verb > " + " < noun>, since an operation is an action [30]. In the case of a message, a message part, or a data-type, their names should be a noun or a noun phrase because they represent the actual objects involved in the operation.

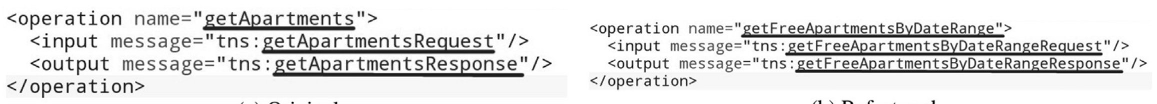
Fig. 8b proposes a modified version of the operation where a more descriptive identifier (`getFreeApartmentsByDateRangeRequest`) is used to describe its functionality.

Specific concepts: Specific concepts further reduce the ambiguity in a WSDL document. It is arguable that a document with specific terms might result more difficult to read. However, with little experience in the service's domain, the specific terms might arguably become familiar. Fig. 9 presents an operation from the Car Rental domain used to rent a compact vehicle. In Fig. 9a the operation name is `bookVehicle`. This identifier describes the operation functionality with an acceptable ambiguity level, and hence does not suffer from the problem listed above. However, since the identifier does not contain specific concepts to fully understand the purpose of the operation, its specificity might be improved by indicating the kind of vehicle to book. Fig. 9b shows a possible rewriting where a more specific concept is used to describe the operation functionality.

Non-cryptic names: Surprisingly, the usage of default parameter names such as `arg_1`, `arg2` or `parameter` is still a common bad practice in service descriptions [10]. This is often a consequence of using automatic tools which do not take this good practice into account [10] to obtain a preliminary WSDL from the service code. This may cause the WSDL document to use meaningless/cryptic identifiers to name port-types, operations and messages. From a semantic perspective, a representative name should at least contain (domain) terms to describe what the corresponding element represents. Thus, meaningless names should be avoided, as this can hinder the actual intention and functioning of the named element. Fig. 10 shows an occurrence of this problem and a possible rewriting. It is important to notice that naming problems are not always solvable, as contextual information may not be enough to refactor the element's name.

Message names related to operation names: From a structural perspective, operations definition in a Web Service description have input/output messages associated. It is expectable that names of both an operation and its messages should be cohesive. Fig. 11 shows an example where the operation name is not related to the message names, and a possible solution. This can lead to miss or overlook the corresponding messages for each operation when exploring the document. In practice, when making the message more cohesive w.r.t. its operation, the best practices described above should be followed to properly name the identifier.

Familiar acronyms and abbreviations: The usage of acronyms or abbreviations created by programmers or service providers is very common. Avoiding unusual acronyms and abbreviations is a good practice because these normally are not familiar for every stakeholder. Fig. 12 shows an example of the supply chain domain where the "OS" acronym stands for "order status". For anyone unrelated with the service domain, this acronym can represent, for example, the noun "operating system". Notice that this practice is related to the "Non-cryptic names" practice, but this latter implies that identifiers appear cryptic to *anyone*. Alternatively, not following the "Familiar acronyms and abbreviations" implies that identifiers appear cryptic/meaningless exclusively to the service consumers who are not familiar with the service domain and its acronyms.



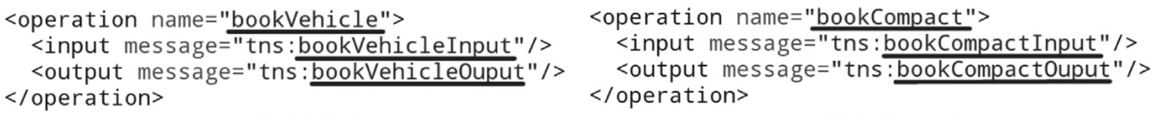
```
<operation name="getApartments">
  <input message="tns:getApartmentsRequest"/>
  <output message="tns:getApartmentsResponse"/>
</operation>
```

(a) Original

```
<operation name="getFreeApartmentsByDateRange">
  <input message="tns:getFreeApartmentsByDateRangeRequest"/>
  <output message="tns:getFreeApartmentsByDateRangeResponse"/>
</operation>
```

(b) Refactored

Fig. 8. Descriptive identifiers example.



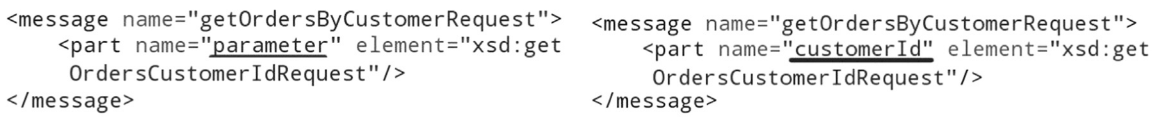
```
<operation name="bookVehicle">
  <input message="tns:bookVehicleInput"/>
  <output message="tns:bookVehicleOuput"/>
</operation>
```

(a) Original

```
<operation name="bookCompact">
  <input message="tns:bookCompactInput"/>
  <output message="tns:bookCompactOuput"/>
</operation>
```

(b) Refactored

Fig. 9. Specific concepts example.



```
<message name="getOrdersByCustomerRequest">
    <part name="parameter" element="xsd:get
     OrdersCustomerIdRequest"/>
</message>
```

(a) Original

```
<message name="getOrdersByCustomerRequest">
    <part name="customerId" element="xsd:get
     OrdersCustomerIdRequest"/>
</message>
```

(b) Refactored

Fig. 10. Non-cryptic parameter names example.

```
<operation name="storeClient">
  <input message="tns:inputMessage"/>
  <output message="tns:outputMessage"/>
</operation>
```

(a) Original

```
<operation name="storeClient">
  <input message="tns:storeClientRequest"/>
  <output message="tns:storeClientResponse"/>
</operation>
```

(b) Refactored

Fig. 11. Message names related to operation names example.

```
<operation name="getOS">
  <input message="tns:getOSRequest"/>
  <output message="tns:getOSResponse"/>
</operation>
```

(a) Original

```
<operation name="getOrderStatus">
  <input message="tns:getOrderStatusRequest"/>
  <output message="tns:getOrderStatusResponse"/>
</operation>
```

(b) Refactored

Fig. 12. Avoid unusual acronyms and abbreviations example.

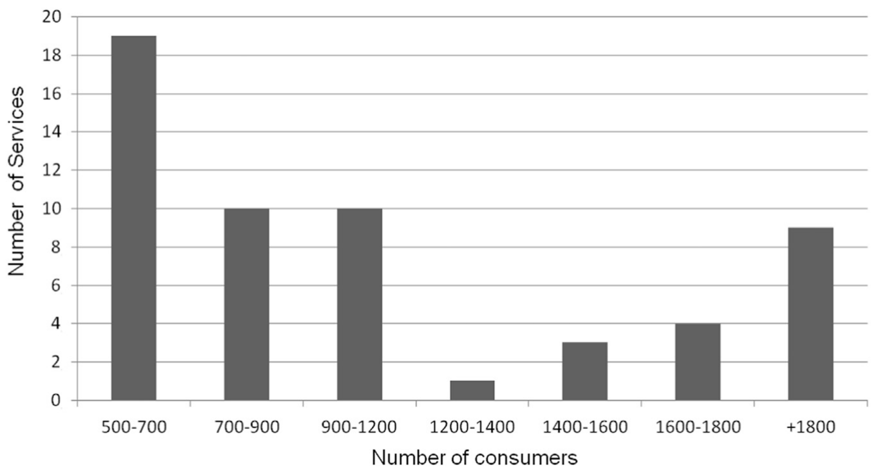


Fig. 13. Number of consumers for the 56 most popular services in the Mashape.com platform.

4. Experiments

In this section we presents both qualitative and quantitative experiments. The first one (Section 4.1) analyzes the applicability/utility of the readability best practices in the industry. The second one (Section 4.2) is a controlled survey with a group of service consumers. The third one (Section 4.3) compares readability values of a dataset of real-life WSDL documents from the industry before and after modifying them to adhere to the readability best practices. The last experiment (Section 4.4) compares the proposed Web Service centered readability metric with other existing readability metrics for plain-text documents.

4.1. *Experiment 1: applicability of best practices in industry*

The goal of this experiment was to analyze the potential applicability/utility of the readability measure in the industry. For this, we measured the absence of readability best practices (as defined in Section 3.2) in a dataset of industry Web Services crawled from the Mashape.com platform.

4.1.1. *Experiment configuration and execution*

To evaluate the best practices application in the industry, we used a dataset of 56 WSDL documents corresponding to real world Web Services extracted from the Mashape.com platform.[10] At the moment of performing this experiment, Mashape.com is used by more than 100,000 developers over the world as a world-class marketplace and repository of Web-accessible APIs, containing approximately 1385 public Web Services. The WSDL documents were generated using an ad hoc Web crawler implemented for this purpose. This crawler explores each available API (service) in the Mashape.com site and automatically generates the corresponding WSDL documents.

We initially considered only the most popular services – i.e., those that were consumed by more than 500 developers, according to the Mashape.com statistics. Thus, we considered a total of 56 services. The most consumed service was selected by 6127 developers. The considered services were consumed 70,570 times by different client-side application developers. Fig. 13 depicts the distribution of the analyzed services according to the number of consumers. By considering the most consumed services, we obtained an attractive panorama about the impact of the readability approach, since more users implies more readers accessing to descriptions.

Finally, an academic group of Web Service experts with knowledge in the readability notions manually inspected each document to identify when a *best practice* was not applied. They did not count how many times each practice was ignored, but rather which practices were ignored in each document.

4.1.2. *Results and discussion*

Fig. 14 shows the percentage of documents that ignore each readability best practice. The results show that the 96% of the service descriptions ignore at least one practice (*overall*). Particularly:

- Naming conventions practice was ignored in the 41% of the analyzed service descriptions.
- Descriptive identifiers practice in the 76% of the analyzed service descriptions.
- Non-cryptic parameter names in the 41% of the analyzed service descriptions.

[10] https://www.mashape.com/

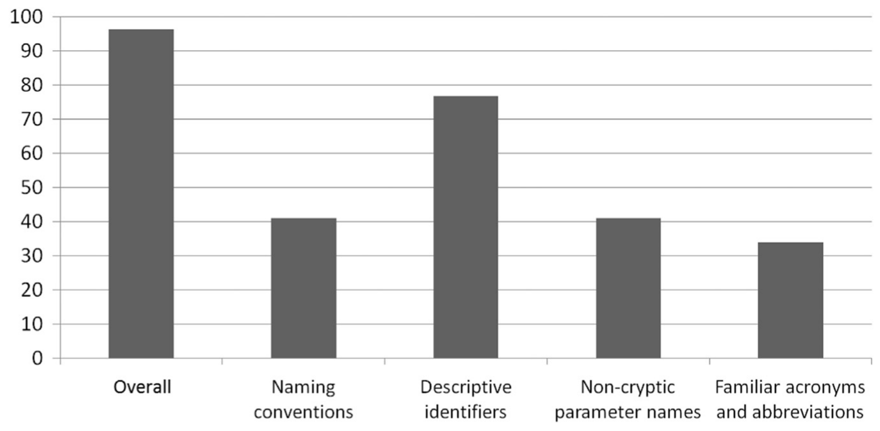


Fig. 14. Percentage of absence of readability best practices in Web Services from the Mashape.com platform.

- Familiar acronyms and abbreviations in the 34% of the analyzed service descriptions.

The results of this experiment show that the most popular (most consumed) services in the Mashape.com platform were developed without awareness of the best practices.

One limitation of this experiment is that, as we mentioned earlier, the WSDL dataset was automatically generated by a crawler using the information extracted from Mashape.com site. In this process, message names were automatically generated using operation names concatenated to "input/output message". Therefore, the best practice "message names related to operation names" was not considered.

4.2. Experiment 2: survey evaluation

We conducted a survey over 6 groups of participants from the IT (Information Technology) sector. Participants included professionals, practitioners, teachers and students. Each participant received WSDL documents from different domains, including WSDLs with either low or high readability values according to our metric. Participants were not concerned with the readability concept prior to take the survey. Then, we measured time and effort required to analyze the functionality contained in the operations defined by the WSDL document. Following we detail the preparation, distribution and results of the experiment.

Hypothesis: Let *n* be WSDL documents that implement similar functionalities. A service consumer will easily analyze (in terms of time and effort) the functionality self-contained in those WSDL documents with higher readability values.

4.2.1. Survey preparation

Dataset: The dataset consisted of 6 original WSDL documents that belong to two domains:

- Domain 1 – eCommerce. Services in this domain provide functionality to manage customers and their purchase orders – i.e., operations to add customers, add orders or associate orders to customers.
- Domain 2 – Hotel. Services in this domain provide functionality to manage hotel bookings – i.e., operations to book a room, associate breakfast to a room or list free rooms.

Each document comprises three operations and their corresponding data types, which are specific to each domain. Moreover, the original WSDL documents presented medium readability values (between 2 and 3). We rewrote each document following the best practices for well-described interfaces introduced in Section 3.2.

After applying the readability best practices, the readability value significantly increased for the rewritten WSDL documents: an increment of 43.6% for the eCommerce domain and 24.96% for the Hotel domain. Therefore, we expect an analogous reduction in time and effort required by participants to understand the more readable versions of the WSDL documents.

Survey form design: For each WSDL document (12 in total) we built a form, containing questions about both the functionality contained in the WSDL document and the required effort to understand such functionality, in terms of the analyzed parts of the WSDL. For each operation in the document, we included two questions:

- Select the correct functionality between three possible options. This question determines if the WSDL is descriptive enough to deduce the operation's functionality. For example, for the operation "*store*" from the eCommerce domain, the options are:
  1. Stores an order corresponding to a Customer
  2. Stores a Customer in the system
  3. Stores the system state when the operation is executed
- Select the required effort in terms of the analyzed elements of the WSDL. Each option refers to how many parts (operation names, data types, parameters, messages and so on) were analyzed to deduce the operation's functionality. Concretely, the three options (for every operation) are:
  1. Operation name and input/output messages only.
  2. Operation name, input/output messages and also parameters.
  3. Operation name, input/output messages, parameters and also data types defined with their attributes.

We also saved the start and finish time when filling a form, to compute the elapsed time participants needed to complete each form. We considered this as an additional indicator of reading effort.

Form distribution: In this survey, 54 people participated as experimental subjects, from which 20 were advanced IT students and 34 were teachers and professionals from the IT sector. We grouped the participants in 6 groups of 9 persons each. Then, we distributed the WSDL documents – as experimental objects – among the participants. We assigned 4 WSDL documents to each group: 2 original WSDL documents (with low readability values) of one domain, and 2 rewritten WSDL documents (with high readability values) of the other domain. Thus, the experimental objects have two possible values: original and rewritten. The goal of this distribution was to make a "blind" survey, where the participants did not know about "good and bad" WSDL documents, in readability terms. In addition, we did not introduce participants into readability concerns nor into good practices to improve readability. Formally, for the experimental design, we followed the guidelines from [31–33]:

- A *balanced factorial* experiment, where the same number *n* of observations is gathered for each experimental subject. In this case,

Table 3
WSDL files used in the experiment.

Domain	WSDL	Form
Customer (original)	CustomerSBO-1	goo.gl/forms/q2MXgTtFOu
	CustomerSBO-2	goo.gl/forms/JQh4yszcOk
	CustomerSBO-3	goo.gl/forms/4I6p5D8AXT
Customer (refactored)	CustomerSBO-1	goo.gl/forms/oJACIS4xas
	CustomerSBO-2	goo.gl/forms/4rkTvGaKjX
	CustomerSBO-3	goo.gl/forms/twlG8WTSGp
Hotel (original)	Hotel-1	goo.gl/forms/ouNwjkAPkM
	Hotel-2	goo.gl/forms/R4SCyPFe9g
	Hotel-3	goo.gl/forms/igcdp9dYwd
Hotel (refactored)	Hotel-1	goo.gl/forms/XTVeNoZhiQ
	Hotel-2	goo.gl/forms/7zAQF0tOI2
	Hotel-3	goo.gl/forms/TeTES3r9Ym

the number of observations is the questions in each form. All participants answered the same number of questions – 6 per WSDL document (24 in total).

- *Confused interaction* in the groups, where only a portion of the WSDL's combinations is assigned to each group [33]. As mentioned earlier, we assigned to each participant 2 original WSDL documents (with low readability values) of one domain, and 2 rewritten WSDL documents (with high readability values) of the other domain. This avoids the experimental threat of the learning effect: when the participant changes from WSDL documents with low readability values to WSDL documents with high readability values, the domain switching prevents comparing with each other. Also, this allows to perform a randomized experiment with a relatively small experimental population, which is not completely randomizable otherwise.

We developed the forms using the Google forms platform, and then we distributed the forms and guidelines by e-mail. Due to the inherent volatility of WSDL services, we hosted the WSDL files in an institutional server from our University to ensure their availability during the conduction of the survey. Table 3 summarizes the list of WSDL files with their domain and the URI of the corresponding form (available for a limited time). A sample form was included in the Appendix. Table 4 summarizes the distribution of forms and WSDL files per group. The WSDL documents were accessible from within the forms during the survey. The complete dataset is available at http://goo.gl/In3PyS.

4.2.2. Results

Two weeks after distributing the forms, we started to collect the results. We measured the results in terms of the following metrics:

Hit rate: This metric collects the answers for the first question about operation's functionality. It assumes a binary value: success (when the operation's functionality is pointed out correctly) or failure (when it is not). Fig. 15a shows the hit rate for the first question about operation functionality:

- The participants selected the correct operation functionality for the 68% of the original WDSL documents.
- The participants selected the correct operation functionality for the 94% of the rewritten WDSL documents.
- For the eCommerce domain, the hit rate was improved by a 32% for rewritten WSDL documents.
- For the Hotel domain, the hit rate was improved by a 14% for rewritten WSDL documents.

Effort: This metric collects the answers for the second question about the effort of analyzing the WSDL file. It is an integer that is mapped to the following qualitative values, according to the selected option for the corresponding question:

1. *Low effort*: The participant only needed to analyze operation name and messages. Effort=1.
2. *Medium effort*: The participant needed to analyze operation name, messages and parameters. Effort=2.
3. *High effort*: The participant needed to analyze operation name, messages, parameters and complex data types with their attributes. Effort=3.

Table 4
Forms distribution per group.

Group	WSDLs	Group	WSDLs
1	Customer-1-O, Customer-2-O Hotel-1-R, Hotel-2-R	4	Customer-1-O, Customer-3-O Hotel-1-R, Hotel-3-R
2	Customer-1-R, Customer-3-R Hotel-1-O, Hotel-3-O	5	Customer-2-O, Customer-3-O Hotel-2-R, Hotel-3-R
3	Customer-1-R, Customer-2-R Hotel-1-O,Hotel-2-O	6	Customer-2-R, Customer-3-R Hotel-2-O, Hotel-3-O

O, Original; *R, Refactored.*

As stated earlier, each possible option of the second question was mapped to a numeric value according to required effort to deduce the operation functionality, ranging from 1 (low effort) to 3 (high effort). Fig. 15b shows the average effort that the participants needed to deduce the operations functionality:

- The average effort for original WSDL documents was 2.65 (medium–high).
- The average effort for refactored WSDL documents was 2.15 (medium).
- The results present an effort reduction of 12% for the eCommerce domain and 32% percent for the Hotel domain.

Time: This represents the elapsed time between starting and submitting each form. It assumes a continuous value using the format MM:SS (minutes and seconds). In that matter, we indicated to the participants that the form should be completed without interruption.

Fig. 15c presents the average time taken to analyze the WSDL documents and answer the forms. The results show that:

- The average time to analyze original WSDL documents was 9:30 min.
- The average time to analyze refactored WSDL documents was 5 min.
- The refactored WSDL documents presented a time reduction of 40% and 55% for eCommerce and Hotel domains, respectively.

4.2.3. Discussion

The experimental results presented in this section validate the experimental hypothesis, as participants required less time and effort to analyze WSDL documents with higher readability values. In particular, we can emphasize the following aspects.

Regarding the relationship between readability values and experimental results, Table 5 shows the readability improvement (percentage) for rewritten and original WSDL documents, and their correlation with the improvements reported in the experiment – in terms of hit rate, effort and time. We observed an overall readability improvement of 34% (considering both domains). This improvement relates to the results observed in hit rate, effort and time – which were improved in 24%, 21% and 46% respectively in the rewritten WSDL documents.

Moreover, the analysis of operations functionality presented a higher hit rate for rewritten WSDL documents. This means that the analysis of more readable documents was easier, having a higher number of positive results. For the Customer domain, the hit rate improvement for refactored documents with regard to original ones

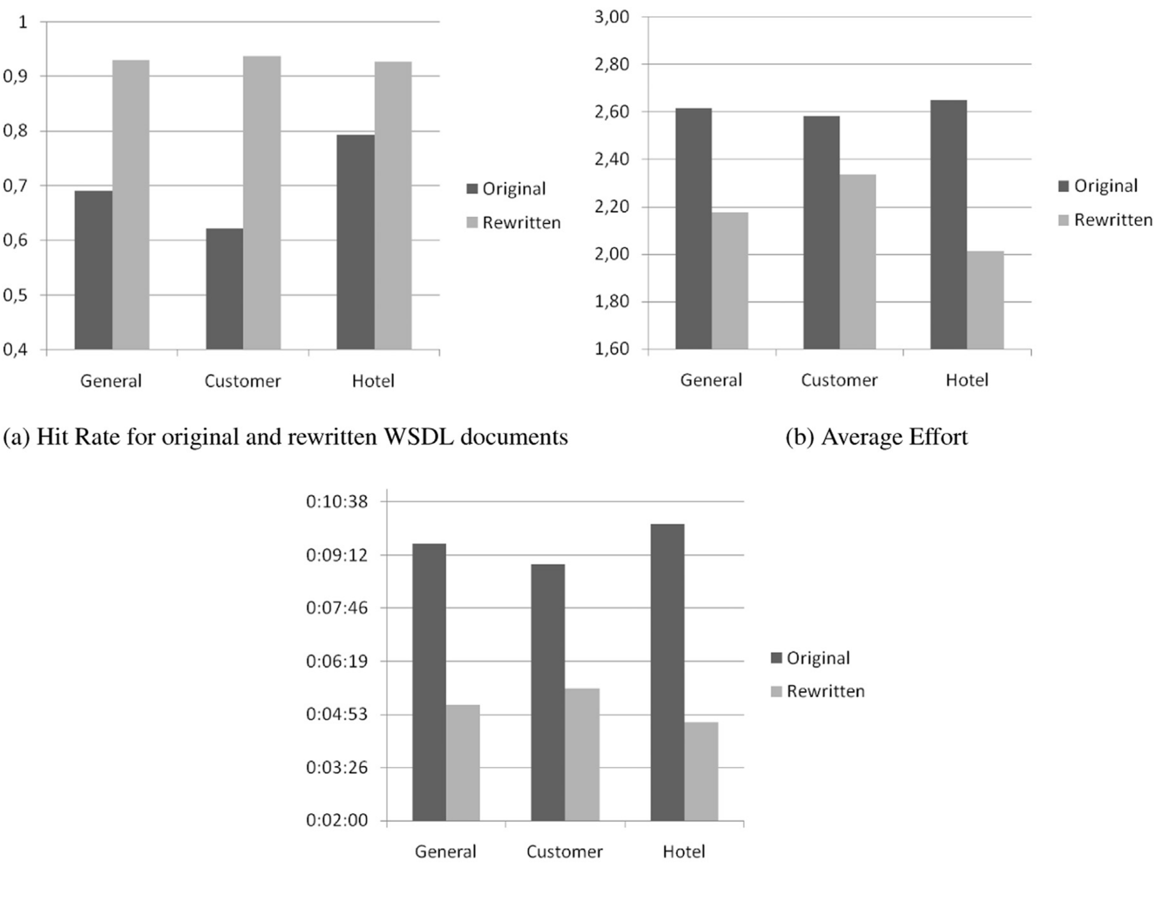


Fig. 15. Survey evaluation results.

Table 5
Readability, hit rate, effort, and time improvement (%) for rewritten WSDL documents.

Domains	Readability (%)	Hit rate (%)	Effort (%)	Time (%)
Overall	34	24	21	46
Customer	44	33	11	38
Hotel	25	14	32	54

was 33%, while for Hotel it was 14%. The slightest improvement for the Hotel domain is because the original documents of such domain presented a higher hit rate. The overall success rate for refactored documents of both domains was 94%.

In the case of average effort to deduce the operations functionality, it was reduced by a 21% overall. However, we expected an even higher reduction for rewritten documents. This is because, in general, service consumers exploit all available information in the documents before selecting a likely functionality. Nevertheless, average effort improvement for rewritten documents was 22%, while the readability values improved around 24 and 43% for the eCommerce and Hotel domain, respectively. Likewise, the required time to analyze the WSDL documents and fill the survey forms was significantly lower (46% overall) for rewritten documents. This suggests that documents with higher readability were simpler to read and analyze.

It is worth mentioning that a possible threat to the validity of an experiment involving people, such as the one performed, is *demand characteristics* [34]. Demand characteristics result from cues in the experimental environment or procedure that lead participants to make inferences about the purpose of the experiment and to respond in accordance with (or in some cases, contrary to) the perceived purpose. Software engineers are inherently problem solvers. When they are told to perform a task, the majority will try to figure out what is expected from them and perform accordingly. *Demand characteristics* influence a participant's perceptions of what is appropriate or expected and, hence, their behavior [33]. In the context of this experiment, the participants may figure out the notion of “good” and “bad” WSDLs (more and less readable documents). Thus, the participants will indicate a lower effort when analyzing readable WSDL documents. Since the experiment adopted *confused interaction* in the groups, the learning effect was reduced and this implies that the deduction was indirectly reduced.

Another experimental threat is the language. Original and rewritten documents were implemented using concepts of the English language. However, even when all participants had some formal English knowledge, they were not native English speakers. Therefore, due to the lack of proficiency in the language, this could generate a slight noise to understand the WSDL documents and complete the survey, specially in terms of time.

4.3. Experiment 3: rewriting evaluation

The goal of this experiment was to measure the Web Service centered readability of WSDL documents in comparative terms. We hypothesize a likely relationship between the Readability value and the application of readability best practices presented in Section 3.2.

4.3.1. Experiment configuration and execution

To evaluate the Web Service centered readability model we used a dataset of 84 WSDL documents extracted from [35]. We have

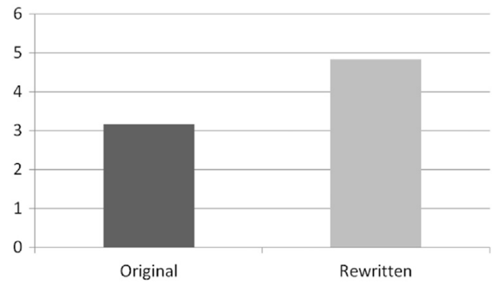


Fig. 16. Average readability values from original and rewritten WSDL documents (higher is better).

implemented a Java application to calculate the readability values for all WSDL documents in the dataset. Then, we applied the readability best practices introduced in Section 3.2 to rewrite a subset of WSDL documents that presented low readability values. Finally, we compared the readability values from the original and refactored documents.

Using as input the WSDL documents in the dataset, we executed the Web Service centered readability procedure. The results consist on the readability value $WSCR(d_i)$ for each WSDL document d_i – as defined in Section 3.1.

After calculating the readability values for the 84 documents, we selected a subset of 34 documents with the lowest readability values to be rewritten. Then remaining documents have a relatively high readability value (greater than 4). It is not worthy to rewrite these documents because only a slight improvement could be obtained – demanding a large effort.

An academic group of Web Service experts with knowledge in the explained readability notions manually rewrote each document (from the selected subset) by applying the best practices presented in Section 3.2. The rewriting criteria were free as it was not prescribed in any guideline. Then, we calculated the readability value for each rewritten WSDL document. Finally, we compared readability values of rewritten documents with the value of their corresponding original documents. Assuming as hypothesis that the rewriting improves the readability, our goal was to quantify these improvements over real-life documents. This provides a precise notion about the potential impact of best practices upon real-life WSDL documents.

4.3.2. Results and discussion

Fig. 16 shows the average readability values from original and rewritten WSDL documents. Original documents presented an average readability value of 3.15 while rewritten documents presented an average value of 4.82. The standard deviations were 0.56 and 0.67 for original and rewritten documents, respectively.

According to the first experiment, it is arguable that documents with higher readability values are easier to read, thus demanding less time and effort to detect key functionality and to be analyzed. The results show an average readability improvement of 52% for rewritten WSDL documents. By drawing a parallel with the previous experiment (survey), the results suggest that rewritten documents would require less time and effort to be analyzed by service consumers.

It is worth noticing that original readability values in the considered dataset are high – around 3.15 – as the WSDL documents are relatively well-written. For example, the terms used in identifiers are well formed and contain existing words, but are not significant enough to infer the corresponding functionality for the operation.

4.4. Experiment 4: comparison with other readability metrics

The goal of the last experiment was to compare the proposed readability metric with other existing readability metrics. For this, we used a third-party readability API published in Mashape.com repository.[11] This API can be used to compute a set of readability metrics for a given textual document. The API currently supports the following metrics: Automated Readability Index, Coleman–Liau Index, Flesch–Kincaid Grade Level, Gunning–Fog Index, SMOG score, and SMOG Index. The service is deployed in the Google App Engine and available at http://ipeirotis.appspot.com/readability/. The code and documentation are available on Github at https://github.com/ipeirotis/ReadabilityMetrics. In this section, we briefly present each readability metric used in the comparison and finally we show the result of this evaluation.

4.4.1. Readability metrics involved in the comparison

The readability metrics offered by the service published in Mashape.com are the following:

Automated readability index: The automated readability index (ARI) is a readability test designed to measure the understandability of a text. It produces an approximate representation of the U.S. grade level needed to comprehend the text. The formula for calculating the ARI is given below, where *characters* is the number of letters and numbers, *words* is the number of spaces, and *sentences* is the number of sentences, manually counted as each text was typed:

$$4.71\left(\frac{characters}{words}\right) + 0.5\left(\frac{words}{sentences}\right) - 21.43$$

Coleman–Liau index: The Coleman–Liau index was designed to be easily calculated mechanically from samples of hard-copy texts. Unlike syllable-based readability indexes, it does not require to analyze the character content of words, only their length in characters. Therefore, it could be used in conjunction with theoretically simple mechanical scanners that would only need to recognize character, word, and sentence boundaries, removing the need for full optical character recognition or manual keypunching. The Coleman–Liau index is calculated with the following formula, where L is the average number of letters per 100 words and S is the average number of sentences per 100 words:

$$CLI - 0.0588{*}L - 0.296{*}S - 15.8$$

Flesch–Kincaid readability: The Flesch–Kincaid readability tests are designed to indicate how difficult is to understand a reading passage in English. There are two tests, the Flesch reading ease, and the Flesch–Kincaid grade level. The “Flesch–Kincaid Grade Level Formula” instead presents a score as a U.S. grade level, making it easier for teachers, parents, librarians, and others to judge the readability level of various books and texts. The grade level is calculated with the following formula:

$$0.39\left(\frac{total\ Words}{total\ Sentences}\right) + 11.8\left(\frac{total\ Sillables}{total\ Words}\right) - 15.59$$

SMOG: The SMOG (Simple Measure of Gobbledygook) grade is a measure of readability that estimates the years of education needed to understand a piece of text. The SMOG grade yields a 0.985 correlation with a standard error of 1.5159 grades with the grades of readers who had 100% comprehension of test materials. Three steps are needed to calculate SMOG: Count a number of sentences (at least 30); then, in these sentences, count the polysyllables (words of 3 or more syllables); and finally calculate the SMOG value using the following formula:

$$grade - 1.0430\sqrt{number\ Of\ Polysyllabes \times \frac{30}{number\ Of\ Sentences}}{*}3.1291$$

Gunning fog index: The index estimates the years of formal education needed to understand a text on a first reading. A fog index of 12 requires the reading level of a U.S. high school senior (around 18

[11] https://market.mashape.com/ipeirotis/readability-metrics/

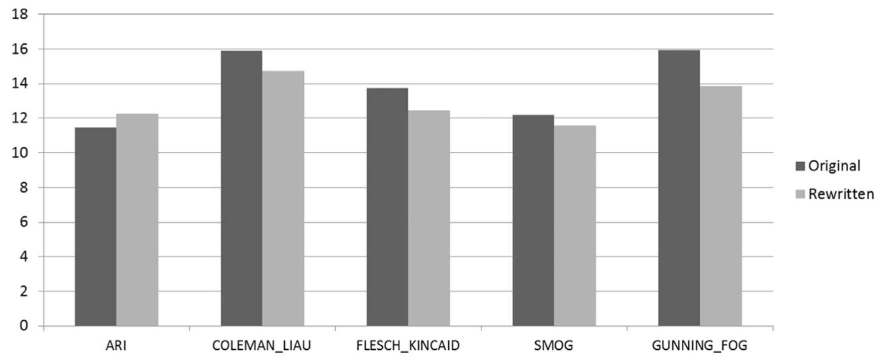


Fig. 17. Average readability values (according to different metrics) for original and rewritten documents.

Table 6
Average readability variations for rewritten WSDL documents.

WSCR	ARI	COLEMAN_LIAU	FLESH_KINCAID	SMOG	GUNNING_FOG
52%	7%	−7%	−9%	−4%	−12%

years old). The fog index is commonly used to confirm that a text can be easily read by the intended audience. The complete formula is:

$$0.4\left[\left(\frac{words}{sentences}\right)+100\left(\frac{complexWords}{words}\right)\right]$$

4.4.2. Experiment configuration

In this experiment we used as dataset the 32 original and 32 rewritten WSDL documents also used in the second experiment (Section 4.3). Then, we generated a representative text in natural language for each WSDL document. For each service operation we generated a sentence with the following structure:

[operation terms] + *"based on"* + *[param1]* + *","* +⋯+ *and* + *[paramN]+". It returns"* + *[returnParam]*+ *"."*

Then, we consumed the service from Mashape.com to calculate the readability metrics using as an input the natural language documents previously generated. Finally we analyzed the obtained results for original and rewritten documents.

4.4.3. Results and discussion

Fig. 17 summarizes the obtained results in two groups: original and rewritten documents. The Y-axis represent the average value obtained for each readability metric.

As expected, the metrics do not present significant changes between original and rewritten documents, since all metrics are based on attributes such as words number, syllables number and sentences. Also, these metrics do not consider the relationship between words, their existence in a dictionary, or their complexity. For service descriptions, analyzing relationships between words is a viable (and desirable) practice because service description documents contain a word set that is smaller than a typical natural language text.

In addition, all metrics provided by the Mashape API do not exploit structural aspects of Web Service descriptions. Then, we conclude that the natural language readability metrics do not perform well for the Web Service context.

Considering Fig. 16 in Experiment 3 and Fig. 17 in this experiment, we could compare our metric and the existing readability metrics previously mentioned. The comparison concerns the readability variations (positives/negatives) between original and rewritten documents. Besides, this is the only way to compare all the metrics since their values are not normalized – which make them not directly comparable. Table 6 shows such variations on readability. We observe a meaningful difference in our metric w.r.t. original documents and those improved following the proposed best practices. Conversely, readability metrics for natural language texts do not present meaningful changes between original and rewritten documents. Besides, values for certain metrics were reduced by the application of best practices, since these metrics are indicators of complexity – i.e., rewritten documents are less complex.

5. Related work

Certainly, our work is related to a number of efforts that can be grouped into two broad classes. On the one hand, there is a substantial amount of research concerning to readability-oriented approaches applied to several areas. On the other hand, several approaches are focused on improving the quality of WSDL documents considering different attributes. In this sense, our approach is related to both kinds of efforts since we share similar goals: proposing a readability metric and improving the quality of WSDL documents.

5.1. Readability-oriented approaches

In [36], authors propose a bilingual (English and Chinese) assessment scheme for Web page and Web site readability based on their textual content, and conduct a series of experiments with real Web data to evaluate the proposal. The authors adopt two classical formulas for readability (one per language), defined in [37,38]. In this approach, the Web site readability is considered as an indicator of the overall difficulty level of a site, considering the readability of all the pages composing the site. Unlike our work, this approach neither includes a semantics basis nor assesses the relations between the words that compose the document – the adopted readability formulas are merely syntactic-based. Then, it would not be possible to calculate how much focused the document is on a particular topic (i.e., cohesion). Also, a document that contains only senseless words could have a good readability value.

The work in [12] (already introduced in Section 2.2) suggests that non-expert users may be interested in searching for domain-specific (e.g., medical and health) information from online resources. However, many users may experience a problem since search results are always a mixture of documents with different levels of readability. With this motivation, authors propose a concept-based model for text readability. In addition to textual content of a document, the model considers how the domain-specific concepts contained in the document affect its readability. Using domain-specific concepts allowed the authors to define a readability formula developed for textual materials. However, as we pointed out in Section 3.1, the lack of complete and relevant domain-specific ontologies hinders the applicability of ontology-based approaches in practice [13,14]. For this reason, we adapted and extended the readability measure by means of WordNet [17] as the underlying concept hierarchy.

In [39], the authors propose a readability metric for Web pages based on link structure. The metric considers the correlation between readability of source and linked pages, assuming that Web pages would typically link to other pages with the same difficulty level. Scores are propagated from good pages in an attempt to separate useful Web pages from Spam. This approach is suitable to measure the readability of Web pages that have little text and to complement readability measures which rely only on textual content. To calculate the readability values for English-written pages, the authors adopted a syntactical formula based on the average number of syllables per word and the average number of words per sentence. As in [36], this readability formula does not consider semantic aspects such as cohesion or word sense.

Lastly, the work in [40], measures document readability at two levels. The first is the surface level readability, that relates to the surface content. It can be assessed by a series of classical readability features. Beyond the surface content, a higher level, namely the topic level readability, reflects whether it is easy for a user to comprehend the hidden topics in documents. Thus, the approach is a topic-based readability model, which can be used to enhance domain-specific information retrieval by considering both the surface and the topic level readability of documents. In our work, we hypothesize that the overall higher taxonomy depths of identified topics in the specific taxonomy would indicate a better document readability. Similarly, the work in [40] also employs the average tree depth of the identified topics to calculate the metric.

5.2. Service descriptions assessment approaches

Several efforts address the problem of the quality of WSDL documents – which should not be confused with considering QoS of the corresponding Web Service. The work in [19] found a high correlation between well-known object-oriented metrics measured directly in the code implementing services, and the occurrences of "anti-patterns" in the corresponding WSDL specifications. Anti-patterns represent a set of indicators of poor quality service interfaces. However, such work is mainly focused in evaluating and improving service discoverability. That is, how easy is to retrieve a service through an automatic mechanism by means of a query. Particularly, service descriptions are assessed from a purely structural perspective. Thus, only morphology of WSDL parts is considered, ignoring their semantic aspects – e.g., meaning of concepts into operation names. In contrast, our main goal is to accomplish service readability/understandability from a dual perspective, including both semantics and human comprehension. Thus, we assess how easy is for a developer to understand the functional capabilities enclosed in a service description. Such capabilities are described by domain concepts in WSDL documents.

The work in [41] measures the impact of naming tendencies on service discovery. The analysis revealed many tendencies, such as *Subsumption relationships*, which consists in naming WSDL parts based on common phrases. For example, using the common phrase "name" to mean "surname", "first_name" or "user_name". The occurrence of many naming tendencies hinders service discoverability as services descriptions become too general. The authors supply a standard-complaint discovery system with heuristics designed for dealing with the identified naming tendencies. As a result, the discovery system achieves better retrieval effectiveness than its original version. Our work could be seen as a complement of such research since readability could improve service discoverability as well, by adding specificity to service descriptions.

The work in [42] extends Web Service data types specifications inside WSDL in order to solve the difficulty for service requester to understand, select and adapt Web Services in the context of a certain application. The proposed model consists of adding descriptions, annotations and constraining facets to the data types, to then generate the corresponding UML class diagram. This process is similar to the application of readability best practices to rewrite WSDL documents.

The work in [43] presents a model of semantic annotations for describing Web Services and an algorithm to discover and compose these Web Services. Semantic annotations are described in OWL-S[12] (Ontology Web Language for Services). In this approach, ontology engineering plays a leading role in adding semantics to service descriptions. However, as mentioned early, it is known that discovering (or even creating) a specific ontology for a given domain requires a large effort even by skillful developers [14,13,7].

The work in [44] presents a metric for understandability of WSDL descriptions called *WSDL Understanding Degree* (WSDLUD). The authors define a criteria tree for each part of WSDL descriptions composed by the following characteristics: (i) Type Understanding Degree, (ii) Message Understanding Degree, (iii) Port Type Understanding Degree, (iv) Binding Understanding Degree and (v) Service Understanding Degree. Each characteristic has an associated sub-criterion tree – e.g., *Type Name Quality* is a sub-criterion from *Type Understanding Degree*. For sub-criteria concerning identifiers like *Type Name Quality*, *Message Name Quality* or *Binding Name Quality* the authors use identifier analysis techniques through the formula *WordsWithMean/NumberOfWords*. *WordsWithMean* is the number of words with meaning included in the identifier (according to a predefined list of words with mean), and *NumberOfWords* is the total number of words included in the identifier. The principal difference with our approach is that the WSDLUD metric does not compare correlation between words that should be related, such as words included in names of messages and their corresponding operations.

Several works have addressed the Web Service anti-patterns detection. The work in [45,46] use knowledge from a base of examples that contains real instances of Web Service anti-patterns. These examples are used to generate a set of Web Service anti-pattern detection rules using two different techniques: a Search-based technique and genetic programming. Additionally, the work in [47] assesses the design and QoS of Service-based Systems (SBSs). Their authors present a framework to automatically specify and detect well-known SOA anti-patterns [48–50] in SBSs. While these approaches adopt a catalog of anti-patterns different from ours, the approach presented in this paper complements and increases the existing tools (e.g., those focused in discoverability) to improve the quality of Web Services from the readability perspective.

Similar to our goal, in [51] the authors present an approach to improve the quality of service descriptions. They propose a solution to describe a complete set of structural constraints for a particular business object in all its use cases. This approach is implemented in XML Schema (XSD) – the *de facto* standard for description of Web Service message structures. Authors define a set of XSD extensions to enable the definition of XSD types and elements to be used in different use cases, each with its own set of specific structural constraints.

6. Conclusions and future work

Broadly, readability is defined as the level of ease or difficulty with which text material can be understood by a particular reader who is reading that text for a specific purpose [5]. In the context of Web Services, service descriptions should be understood with ease so that these descriptions can be used as a strategy by service providing organizations to attract service consumers [6], and to simplify the

[12] https://www.w3.org/Submission/OWL-S/

process of finding and reusing external functionality outsourced by client applications. This latter is due to the fact that service consumption always involves a manual inspection phase after service descriptions have been retrieved from a registry [3].

Motivated by the problems of metrics designed for natural text documents to quantify Web Service readability, this paper presented a readability metric for WSDL documents. In contrast to previous work in the field [12,6], which assume/associate ontologies for representing service descriptions, we propose the use of WordNet as the underlying concept hierarchy, in replacement of a domain-specific ontology for each possible domain. The use of a general, lightweight concept hierarchy makes the approach widely applicable, as meaningful domain-specific ontologies are not usually available in practice [13,14,8]. It is arguable that terms that do not belong to WordNet are left out from the readability calculation in our approach, although they may be significant in certain contexts. However, this does not affect the overall readability calculation, since most terms in (English written) real-life documents belong to WordNet dictionary. For example, in the documents crawled from Mashape.com site, the ratio of WordNet terms to non-WordNet terms is 20 to 1. Moreover, the proposed readability metric is not intended to be used in isolation. As mentioned in Section 5.2, it is complementary to other metrics such as discoverability/understandability metrics [19] or maintainability metrics [9] that measure the complexity of the messages in WSDL documents.

In addition, we proposed a set of readability best practices to be adopted during the development or rewriting of WSDL documents. These practices suggest the adoption of naming conventions, specific concepts, non-cryptic parameter names, message names related with their corresponding operation and familiar acronyms and abbreviations. As reported in the experiments, considering these practices improve overall readability of the resulting service descriptions.

In the first experiment, we analyzed the applicability/utility of the readability metric in the industry. For this, we measured the absence of readability best practices in a large dataset of industry Web Services from the Mashape.com site. The results show that a 96% of the most popular services in this platform were developed without awareness of the best practices, and thus there is an interesting place for improvement of real-world documents.

As qualitative evaluation, we conducted a survey over 6 groups of participants from the IT field (54 persons), where each participant evaluated a set of WSDL documents from different domains, including WSDL documents with either low or high readability values. The experiment has shown that it is easier, in terms of required time and effort, to analyze the self-contained functionality in the WSDL documents with higher readability values. Thus, such documents can be used by service providers to attract potential consumers effectively.

Additionally, in another experiment we calculated readability values for WSDL documents in a public dataset. Then, a subset of documents was rewritten by applying the readability best practices and re-calculated readability values. The readability values from original and rewritten documents were compared, showing that the readability values increased in average 52% for the latter. According to the experiments, it is arguable that documents which acknowledge the readability best practices defined in Section 3.2, not only may have higher readability values but also may reduce the understandability effort by consumers, as shown by the different experiments in Section 4. When faced with little readable documents, consumers were forced to make a (likely) large effort on deducing the functional capabilities of such services [11]. Therefore, the proposed readability model is potentially useful for its application in industry and real-life scenarios of service consumption.

Regrettably, previous work in the field of readability for WSDL documents [6] did not present experimental evaluation to be compared with our results. In the last experiment, we compared our metric with existing readability metrics for plain-text documents using a third-party readability API published in the Mashape.com repository. We conclude that these metrics are too general to be applicable to Web Service descriptions, and thus a specific metric is needed.

As future work we are planning to extend the readability model to support code-first services. Code-first is a popular approach to build Web Services in the industry [16]. With this approach, developers first implement a service and then generate the corresponding WSDL by automatically extracting and deriving the interface from the implemented code. This means that WSDL documents are not directly created by developers but are instead automatically derived via language-dependent tools. In the case of a contract-first approach, WSDL documents are manually developed before service implementation, therefore developers can apply the best practices when crafting the WSDL. Using the code-first approach, developers partially delegate the control of the WSDL generation to such tools. For this reason, we propose to analyze the existing language-dependent tools and propose a readability model extension that considers a code-first service developing approach.

Other future work consists of exploiting and integrating other alternative semantic basis such as DISCO (DIStributed COllocations) [52,25] and NER (Named Entity Recognition) [53] for readability calculation. On the one hand, DISCO is a pre-computed database of collocations and distributionally similar words. The similarity of words is based on the statistical analysis of very large text collections (e.g., Wikipedia), through co-occurrence functions. Thus, DISCO could be an alternative to WordNet as the concept hierarchy that allows calculating document cohesion and scope. On the other hand, NER is a technique to detect and replace *entities* (e.g., names, organizations, acronyms, places, etc.) with their definition, by means of big corpus of annotated text such as Wikidata[13] and Babelnet.[14] We are analyzing how these aspects affect readability. In this context, an important challenge is the disambiguation of overloaded terms and acronyms.

In addition, we are working in algorithms and tooling support to automatize the detection of potential readability improvements and their application. WordNet allows to detect the corresponding part-of-speech (POS) for each word. Thus, it can be assessed, for example, whether the form <Verb> + <Noun> in the first words of an operation name is used or not. In this work, we have presented the minimum set of notions (metrics + practices) that we consider necessary to transfer this metric to the industry. The automatic detection is desirable but not mandatory, and might be further addressed.

Finally, we intend to apply our approach to RESTful Web Services [54], which still present open problems for their description [8]. If we follow the approach proposed by IBM [55], where a REST Web Service is described with WSDL 2.0, then our approach could be directly applied. Otherwise, we will have to adapt our approach to the context of RESTful Web Services.

Acknowledgments

This work is supported through research grants ANPCyT PICT-2012-0045 and SPU-UNCo Service-oriented Reuse (04-F001), and CONICET scholarships. We would like to thank the reviewers for their helpful comments to improve the quality of this paper. Also, we thank to the IT professionals, teachers and students who participated in the survey.

Appendix A. Form example

This WSDL document (link given below) describes a Customer Management service. By analyzing only the self-contained information

[13] http://wikidata.org
[14] http://babelnet.org

in the WSDL (operation names, messages, data types, etc.), select the most adequate answers.

PLEASE COMPLETE THE STARTING TIME BEFORE ANSWERING

WSDL: CustomerSBO-1 http://goo.gl/In3PyS

Starting time (Required): ____:____

- Name and Surname (optional): ____________________________________
- Student Number (only for students): ______________________________
- Knowledge level in the Web Services field (optional):
  1. Basic
  2. Medium
  3. Advanced

- The operation "Store"…
  1. Stores the state of the system when executed
  2. Stores a new Customer in the system
  3. Stores an Order for a given Customer

- The functionality of the "Store" operation is inferable from…
  1. Operation name and input/output messages only
  2. Operation name, input/output messages and parameters
  3. Operation name, input/output messages, parameters and corresponding data types with their attributes

- The operation "addOrd"…
  1. Stores a new Order Type in the system
  2. Stores a new Order for a given Customer
  3. Stores a new Order for the logged Customer

- The functionality of the "addOrd" operation is inferable from…
  1. Operation name and input/output messages only
  2. Operation name, input/output messages and parameters
  3. Operation name, input/output messages, parameters and corresponding data types with their attributes

- The operation "Auth"…
  1. Authenticates the user who is logging in the system
  2. Given an user name and a password, returns a Customer
  3. Given a security code, authenticates an Order

- The functionality of the "Auth" operation is inferable from…
  1. Operation name and input/output messages only
  2. Operation name, input/output messages and parameters
  3. Operation name, input/output messages, parameters and corresponding data types with their attributes

References

[1] J. Erickson, K. Siau, Web service, service-oriented computing, and service-oriented architecture: separating hype from reality, J. Database Manag. 19 (3) (2008) 42–54.

[2] M. Bichler, K. Lin, Service-oriented computing, Computer 39 (3) (2006) 99–101.

[3] J.L. Ordiales Coscia, C. Mateos, M. Crasso, A. Zunino, Refactoring code-first web services for early avoiding WSDL anti-patterns: approach and comprehensive assessment, Sci. Comput. Program. 89 (Part C) (2014) 374–407. http://dx.doi.org/10.1016/j.scico.2014.03.015.

[4] M. Crasso, J.M. Rodriguez, A. Zunino, M. Campo, Revising WSDL documents: why and how, IEEE Internet Comput. 14 (5) (2010) 48–56.

[5] J.J. Pikulski, Readability, Available: ⟨http://www.eduplace.com/state/author/pikulski.pdf⟩, 2012.

[6] P. Sripairojthikoon, T. Senivongse, Concept-based readability of web services descriptions, in: 15th International Conference on Advanced Communication Technology (ICACT), IEEE, New York, United States, 2013, pp. 853–858.

[7] M. Garriga, A. Flores, A. Cechich, A. Zunino, Web services composition mechanisms: a review, IETE Techn. Rev. 32 (5) (2015) 376–383.

[8] M. Garriga, C. Mateos, A. Flores, A. Cechich, A. Zunino, Restful service composition at a glance: a survey, J. Netw. Comput. Appl. 60 (2016) 32–53. http://dx.doi.org/10.1016/j.jnca.2015.11.020.

[9] D. Baski, S. Misra, Metrics suite for maintainability of extensible markup language web services, IET Softw. 5 (3) (2011) 320–341. http://dx.doi.org/10.1049/iet-sen.2010.0089.

[10] C. Mateos, M. Crasso, A. Zunino, J.L. Ordiales Coscia, A stitch in time saves nine: early improving code-first web services discoverability, Int. J. Cooperat. Inf. Syst. 24 (02) (2015) 1550004-1-1550004-38.

[11] M. Garriga, A. Flores, C. Mateos, A. Zunino, A. Cechich, Service selection based on a practical interface assessment scheme, Int. J. Web Grid Serv. 9 (4) (2013) 369–393.

[12] X. Yan, D. Song, X. Li, Concept-based document readability in domain specific information retrieval, in: Proceedings of the 15th ACM International Conference on Information and Knowledge Management, ACM, New York, NY, USA, 2006, pp. 540–549, http://dx.doi.org/10.1145/1183614.1183692.

[13] D. Bouchiha, M. Malki, A. Alghamdi, K. Alnafjan, Semantic web service engineering: annotation based approach, Comput. Inform. 31 (6) (2012) 1575–1595.

[14] M. Crasso, A. Zunino, M. Campo, A survey of approaches to web service discovery in service-oriented architectures, J. Database Manag. 22 (1) (2011) 102–132.

[15] D. Roman, U. Keller, H. Lausen, J. De Bruijn, R. Lara, M. Stollberg, A. Polleres, C. Feier, C. Bussler, D. Fensel, Web service modeling ontology, Appl. Ontol. 1 (2005) 77–106.

[16] C. Mateos, J.M. Rodriguez, A. Zunino, A tool to improve code-first web services discoverability through text mining techniques, Softw.: Pract. Exp. 45 (7) (2015) 925–948.

[17] G. Miller, R. Beckwith, C. Fellbaum, D. Gross, K. Miller, Introduction to wordnet: an on-line lexical database, Int. J. Lexicogr. 3 (4) (1990) 235–244.

[18] J.M. Rodriguez, M. Crasso, C. Mateos, A. Zunino, Best practices for describing, consuming, and discovering web services: a comprehensive toolset, Softw.: Pract. Exp. 43 (6) (2013) 613–639.

[19] C. Mateos, M. Crasso, A. Zunino, J.L. Ordiales Coscia, Detecting WSDL bad practices in code-first web services, Int. J. Web Grid Serv. 7 (4) (2011) 357–387.

[20] M.P. Papazoglou, W.-J. Van Den Heuvel, Service-oriented design and development methodology, Int. J. Web Eng. Technol. 2 (4) (2006) 412–442.

[21] J.M. Rodriguez, M. Crasso, A. Zunino, M. Campo, Improving web service descriptions for effective service discovery, Sci. Comput. Program. 75 (11) (2010) 1001–1021.

[22] J.L. Ordiales Coscia, C. Mateos, M. Crasso, A. Zunino, Anti-pattern free code-first web services for state-of-the-art java wsdl generation tools, Int. J. Web Grid Serv. 9 (2) (2013) 107–126.

[23] A. Buccella, A. Cechich, M. Arias, M. Pol'La, M. del Socorro Doldan, E. Morsan, Towards systematic software reuse of gis: insights from a case study, Comput. Geosci. 54 (2013) 9–20.

[24] J. Chall, E. Dale, Readability Revisited: The new Dale–Chall readability formula, Brookline Books, 1995.

[25] A. De Renzis, M. Garriga, A. Flores, A. Cechich, A. Zunino, Semantic-structural assessment scheme for integrability in service-oriented applications, in: IEEE Latin American Computing Conference (CLEI), 2014, pp. 637–647. http://dx.doi.org/10.1109/CLEI.2014.6965175.

[26] E.N. Zalta, S. Abramsky, Stanford Encyclopedia of Philosophy (2003), 2003.

[27] J. Rodriguez, M. Crasso, C. Mateos, A. Zunino, M. Campo, The EasySOC project: a rich catalog of best practices for developing web service applications, in: IEEE International Conference of the Chilean Computer Science Society (SCCC), 2010, pp. 33–42.

[28] D.M. Jones, Operand names influence operator precedence decisions, in: ACCU Conference, vol. 20, Oxford, UK, 2008, pp. 1–14.

[29] K. Cwalina, B. Abrams, Framework Design Guidelines: Conventions, Idioms, and Patterns for Reusable .net Libraries, Pearson Education, New Jersey, United States, 2008.

[30] R. Chinnici, J.-J. Moreau, A. Ryman, S. Weerawarana, Web Services Description Language (WSDL) 2.0, W3C Recommendation, June 2007, URL ⟨https://www.w3.org/TR/wsdl20/⟩

[31] N. Juristo, A. Moreno, Basics of Software Engineering Experimentation, Springer Publishing Company, New York, United States, 2010.

[32] C. Wohlin, P. Runeson, M. Höst, M.C. Ohlsson, B. Regnell, A. Wesslén, Experimentation in Software Engineering, Springer Science & Business Media, New York, United States, 2012.

[33] R.E. Kirk, Experimental Design, Wiley Online Library, 1982.

[34] M.T. Orne, Demand characteristics and the concept of quasi-controls, in: Artifacts in Behavioral Research, Oxford University Press, Cary, NC, United States, 2009.

[35] A. Heß, E. Johnston, N. Kushmerick, Assam: a tool for semi-automatically annotating semantic web services, in: Proceedings of The Semantic Web Conference (ISWC), Springer, New York, United States, 2004, pp. 320–334.

[36] T.P. Lau, I. King, Bilingual web page and site readability assessment, in: Proceedings of the 15th International Conference on World Wide Web, ACM, New York, United States, 2006, pp. 993–994.

[37] S. Yang, A Readability Formula for Chinese Language, University of Wisconsin-Madison, Madison, 1970.

[38] R. Flesch, A new readability yardstick, J. Appl. Psychol. 32 (3) (1948) 221.

[39] A. Jatowt, K. Akamatsu, N. Pattanasri, K. Tanaka, Towards more readable web: measuring readability of web pages based on link structure, SIGWEB Newslett. Winter (2012). http://dx.doi.org/10.1145/2073242.2073246 4:1–4:7.

[40] W. Zhang, D. Song, P. Zhang, X. Zhao, Y. Hou, A sequential latent topic-based readability model for domain-specific information retrieval, in: Information Retrieval Technology, Springer, New York, United States, 2015, pp. 241–252.

[41] M.B. Blake, M.F. Nowlan, Taming web services from the wild, IEEE Internet Comput. 12 (5) (2008) 62–69.

[42] F. Alshraiedeh, S. Hanna, R. Alazaidah, An approach to extend WSDL-based data types specification to enhance web services understandability, Int. J. Adv. Comput. Sci. Appl. 6 (3) (2015) 88–98.

[43] H.N. Talantikite, D. Aissani, N. Boudjlida, Semantic annotations for web services discovery and composition, Comput. Stand. Interfaces 31 (6) (2009) 1108–1117.

[44] M.M. Berón, H. Bernardis, E.A. Miranda, D.E. Riesco, M.J.V. Pereira, P.R. Henriques, WSDLUD: a metric to measure the understanding degree of WSDL descriptions, in: 4th International Symposium in Languages, Applications and Technologies: Selected Papers, Springer International Publishing, New York, United States, 2015, pp. 91–100.

[45] A. Ouni, M. Kessentini, K. Inoue, M.O. Cinneide, Search-based web service antipatterns detection, IEEE Trans. Serv. Comput., In press, http://dx.doi.org/10.1109/TSC.2015.2502595.

[46] A. Ouni, R. Gaikovina Kula, M. Kessentini, K. Inoue, Web service antipatterns detection using genetic programming, in: Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation, ACM, New York, United States, 2015, pp. 1351–1358.

[47] N. Moha, F. Palma, M. Nayrolles, B.J. Conseil, Y.-G. Guéhéneuc, B. Baudry, J.-M. Jézéquel, Specification and detection of SOA antipatterns, in: International Conference on Service-Oriented Computing ICSOC, Springer, New York, United States, 2012, pp. 1–16.

[48] B. Dudney, S. Asbury, J.K. Krozak, K. Wittkopf, J2EE Antipatterns, John Wiley & Sons, New York, United States, 2003.

[49] J. Král, M. Zemlicka, Crucial service-oriented antipatterns, Int. J. Adv. Softw. 2 (2009) 160–171.

[50] A. Rotem-Gal-Oz, E. Bruno, U. Dahan, SOA Patterns, Manning, New York, United States, 2012.

[51] Ales Frece, Matjaz Juric, Complete and reusable description of message structural constraints in web service interfaces, Comput. Stand. Interfaces 35 (2) (2013) 218–230.

[52] P. Kolb, Experiments on the difference between semantic similarity and relatedness, in: Proceedings of the 17th Nordic Conference on Computational Linguistics NODALIDA, 2009, pp. 81–88.

[53] D. Nadeau, S. Sekine, A survey of named entity recognition and classification, Lingvist. Invest. 30 (1) (2007) 3–26.

[54] R. Fielding, Architectural styles and the design of network-based software architectures (Ph.D. thesis), University of California, CA, USA, 2000.

[55] L. Mandel, Describe REST Web services with WSDL 2.0, IBM Developer Works, 2008, URL ⟨www.ibm.com/developerworks/library/ws-restwsdl/⟩