Towards Automatic Service Level Agreements Information Extraction

Lucia De Marco[1,2], Filomena Ferrucci[1], M-Tahar Kechadi[2], Gennaro Napoli[1] and Pasquale Salza[1]
[1]*Department of Computer Science, University of Salerno, Salerno, Italy*
[2]*School of Computer Science, University College Dublin, Dublin, Ireland*

Keywords: Cloud Computing, Service Level Agreements, Natural Language Processing, Information Extraction.

Abstract: Service Level Agreements (SLAs) are contracts co-signed by an Application Service Provider (ASP) and the end user(s) to regulate the services delivered through the Internet. They contain several clauses establishing for example the level of the services to guarantee, also known as quality of service (QoS) parameters and the penalties to apply in case the requirements are not met during the SLA validity time. SLAs use legal jargon, indeed they have legal validity in case of court litigation between the parties. A dedicated contract management facility should be part of the service provisioning because of the contractual importance and contents. Some work in literature about these facilities rely on a structured language representation of SLAs in order to make them machine-readable. The majority of these languages are the result of private stipulation and not available for public services where SLAs are expressed in common natural language instead. In order to automate the SLAs management, in this paper we present an investigation towards SLAs text recognition. We devised an approach to identify the definitions and the constraints included in the SLAs using different machine learning techniques and provide a preliminary assessment of the approach on a set of 36 publicly available SLA documents.

1 INTRODUCTION

Service Level Agreement (SLA) concerning the provisioning of IT services is defined as a 'formal, negotiated, document in qualitative and quantitative terms detailing the services offered to a customer' (ITIL, 2016).

The necessity of regulating a fair service outsourcing via co-signed SLAs aims to give an explicit understanding to the end users about what a service is, where and when it is delivered and how to use it; also duties and responsibilities of both parties and the possible interaction of a third party are outlined. This is very important in the context of cloud computing to support forensic investigations made especially hard by the extreme flexible nature of the cloud itself.

The main contents of an SLA concern a definition and description of the service, some rules and regulations about its delivery, some performance measures together with possible tolerance intervals about the levels of the services to guarantee and the pricing and penalty measures in case such tolerances are not respected. With the purpose of monitoring service levels a dedicated contract management facility should be a part of the service provisioning.

Such facility can be applied in the context of cloud security and log analysis by verifying if some constraints are violated. One or more violations can be symptoms of a cyber-attack in the service infrastructure. On the other hand a service level violation by the provider side has a billing consequence aiming to refund the consumer for the problem.

In literature (see Section 2) it is possible to find some work concerning these facilities between private parties, where stipulations are expressed in a structured language and thus machine-readable from the origin. Nevertheless, in case of public services the majority of them relies on the presence of a stipulation expressed in common natural language. There is no room for automating the SLAs management in case of public services if these documents are not recognised in the first place.

In this paper we introduce an approach to allow for the automatic extraction of information relevant for monitoring purposes. We employed the tool of information extraction GATE[1] in order to recognise the sentences in SLA documents and extract relevant definitions, values and formulas. We report on a preliminary investigation about the use of the proposed

[1]General Architecture for Text Engineering (GATE), University of Sheffield, https://gate.ac.uk

Marco, L., Ferrucci, F., Kechadi, M-T., Napoli, G. and Salza, P.
Towards Automatic Service Level Agreements Information Extraction.
In *Proceedings of the 6th International Conference on Cloud Computing and Services Science (CLOSER 2016) - Volume 2*, pages 59-66
ISBN: 978-989-758-182-3

approach by analysing the performance of 5 different classification algorithms on a dataset of 36 SLA documents of real public services such as Amazon, Microsoft and Cisco.

The rest of the paper is organised as follows. Related work about SLA monitoring is illustrated in Section 2. Then, an examination of SLA contents is presented in Section 3. The performed preliminary automation of SLAs text recognition is described in Section 4 and its effectiveness is assessed in Section 5. Some conclusions and future work close the paper in Section 6.

2 RELATED WORK

Automating the management of a textual document is a big challenge, specially if this document is a contract with legal validity like SLAs stipulated for cloud services. Several approaches in the literature that exacerbate the automatic management of SLAs and the different modalities to face this issue are discussed in this section.

Some work dedicated to manage SLAs decomposes the issue by monitoring the single constraints included in the contract. For instance, a framework called DESVI (Emeakaroha et al., 2010) proposed by Emeakaroha et al., is dedicated to monitor the performances of low level cloud resources in order to detect if the obtained measures respect the constraints extracted from SLAs with the goal of detecting Quality of Service (QoS) violations. This work has been used by Emeakaroha et al. as a background monitoring platform (Emeakaroha et al., 2012b) to demonstrate its efficiency in monitoring a single cloud data centre. Also Brandic et al. (Brandic et al., 2010) used DESVI as a low level resource value calculator, but the metrics applied on the resources have to output a value required to match with a specified threshold to prevent possible contractual violations. Morshedlou et al. (Morshedlou and Meybodi, 2014) proposed a proactive resources allocation prototype for reducing the negative impact of SLA violations and for improving the level of users satisfaction.

A prototype for an autonomic SLA enhancement is discussed by Maurer et al. (Maurer et al., 2012). It behaves as a resource parameter reconfiguration tool at virtual machine level of cloud infrastructures, with the main advantage of reducing SLA violations and of optimizing resource utilisation. Instead, Emeakaroha et al. (Emeakaroha et al., 2012a) proposed an SLA monitoring and violation detection architecture that plays at a cloud application provisioning level, where some metrics are exploited to monitor at runtime the resources consumption behaviour and the performance of the application itself. Cedillo et al. presented an approach to monitor some cloud services non-functional SLA requirements (Cedillo et al., 2014). A middleware interacting with services and applications at runtime is designed; it analyses the information and provides some reports as soon as an SLA violation is identified. SALMonADA (Muller et al., 2014) is a platform proposed by Muller at. al that utilises a structured language to represent the SLA, which is then automatically monitored to detect whether any violation occurs or not; this detection is performed by implementing a technique based on a constraint satisfaction problem.

One of the reasons to implement a dedicated automatic contractual management system can be the detection of contractual violations to be exploited in many circumstances, such as forensic readiness activities. Forensic readiness (Tan, 2001) is a capability aimed at minimizing the costs of a digital crime investigation by performing some pro-active and preparatory activities in a computing environment. The main aim of our proposal is to contribute towards making automatic information extraction from SLAs as the essential starting point for any cloud forensic readiness tool.

3 SERVICE LEVEL AGREEMENTS

In order to interpret the SLAs written in natural language using business and legal jargon, we started by studying their form and recurring patterns. In this section some definitions and common patterns are described, resulting from a literature study and simple empirical observations.

3.1 Life Cycle

To the best of our knowledge, an SLA follows the life cycle depicted in the UML state chart diagram (Rumbaugh et al., 2004) in Figure 1. an SLA is initially defined via a contractual template, which is customised by the provider depending on some users variation requests on the standard offer. For this reason a negotiation phase happens, where solutions to the change requests are included, together with information about expenses, penalties and reports. The Co-Signed phase determines that both entities agree on the actual contractual contents, then the service provisioning can begin. The SLA has a validity time, that can be either explicitly expressed with start and end dates or with an initial date together with a time interval, both included in the document. Such validity time begins after the

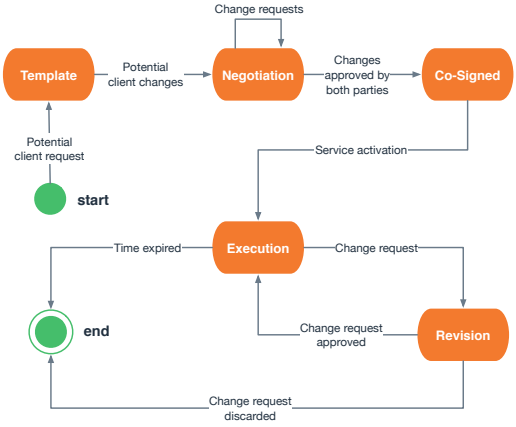

Figure 1: SLA Life Cycle (De Marco et al., 2015).

contract is co-signed by both parties in the Execution phase.

During its life cycle, an SLA can be subject to revisions to resolve some change requests instantiated by either parties. In case no solution is reached, the service provisioning has to be terminated and the SLA is no longer valid.

3.2 General Contents

One of the biggest issues in automatic SLA monitoring is the lack of a standard for representing the contracts. Nevertheless, some efforts were made in order to outline some SLAs commonalties by Baset et al. (Baset, 2012), where the anatomy of such contracts is provided. The authors affirm that an SLA is composed of:

- the provided service levels and the metrics used for guaranteeing them;
- the services time duration and their granularity;
- the billing structure;
- the policy about the level measurement;
- the reporting manners about service guarantee violations.

An SLA is composed of a set of clauses, which describe all the constraints, behaviours and duties of the co-signer parties in order to guarantee the level of the predefined services. For instance, some clauses concern the metrics necessary for measuring the described services level attributes, such as latency or average transmission errors rate.

3.3 Structure

The structure of an SLA may differ from one cloud service provider to another. However, such a contract is composed of several sections. Among these sections, an SLA can be structured as a set of Service Level Objectives (SLOs). In a European Union guideline document (European Commission and DG CONNECT Cloud Select Industry Group, 2014) the described SLOs are catalogued as follows:

- Performance;
- Security;
- Data Management;
- Personal Data Management.

It is reasonable to affirm that an SLA is a set of SLOs, because each of them describes a single parameter without overlapping with the remaining ones. An SLO is composed of a set of sentences, usually more than one. A general SLO describes a constraint about a parameter of a service level included in an SLA, together with the value and a unit measure and/or a percentage value of such a parameter. An SLO in some cases can describe the metrics used by the service provider to calculate the value of the parameter it indicates. Generally, the description is textual, expressed in natural language; in some other cases, beside the textual description, a mathematical formula is textually described. The time interval to consider is also an important feature. Every SLA has a validity time period expressed either explicitly with start and end dates or with a start date and a time period, such as a 'billing month' or a 'solar year'. The SLOs included in the SLA are not necessarily constrained during the same time interval; indeed they can have validity during a different time period, which can even end after the SLA validity time (e.g., the 'backup retention time' SLO can finish after the SLA termination); certainly, no SLO can begin before an SLA starting time.

3.4 SLO Classification

The sentences composing the SLOs can be classified into the following parts:

1. Definition;
2. Value;
3. Not definition.

The assumption made in this taxonomy is that every sentence of an SLO can fall only in one class; a sentence is a sequence of words enclosed in two full stops, where the initial one is discarded. The sentences composing an SLO can represent either a value together with an unit measure or percentage for the constrained parameter or the metric description. Thus, an automatic procedure has to recognise whether a sentence of an SLO is a definition, a numeric value or a text which is not a definition. A subsequent step

of the classification is the identification of mathematical and textual formulas from the detected definition; in this way the class *Definition* can be split in two: Mathematical formula; Textual formula.

The mathematical formula represents a manner in which the detected description can be easily translated into a mathematical formula, namely some strategic words are recognised, such as adding, divided by, rate, ratio, etc.

The textual formula instead describes a definition that outlines a textual description for the metric necessary to the computation of the value of a parameter included in the SLO, which cannot be represented with a mathematical formula. It is necessary to mention that the difference between mathematical and textual formulas is very narrow and a classification algorithm based on training is not enough to distinguish one or another. This reason motivates splitting the class *Definition* into two subclasses, for which another information extraction technique is expected.

4 SLAs TEXT RECOGNITION

In order to design and develop an automatic SLA classifier, some Natural Language Processing (NLP) techniques can be utilised. Such techniques have the aim to elaborate the document containing an SLA and to obtain the information about the SLOs necessary to feed a possible dedicated service monitoring tool.

The principal open-source Information Extraction (IE) (Grishman, 1997) tools are: Apache OpenNLP, OpenCalais, DBpedia and GATE. The principal tasks of OpenNLP are tokenisation, sentence segmentation, part-of-speech tagging, named entity extraction, chunking, parsing and co-reference resolution. All these tasks are present also in OpenCalais and GATE with a more usable graphical interface. OpenCalais can annotate documents with rich semantic meta-data, including entities, events and facts. However, the output of the tool is text enriched by annotations, which are not user-customisable.

We chose to use GATE for the implementation of the automatic classifier for SLAs. GATE performs all the tasks described for the other tools and has the advantage of being customisable; indeed, it allows to create customised types of annotations using a Java Annotation Patterns Engine (JAPE) transducer personalisation; also the annotations that the tool produces can be customised. The most used component from GATE for this automation is the Information Extraction (IE) one. The input to the system is a dataset of SLA documents in many common use formats, such as Microsoft Word and Adobe Portable Document Format (PDF). The output of the classifier is composed of the same documents but with annotations. The annotations are added to some sentences of the documents and they correspond to the identified classes described before (*Definition*, *Value* and *Not definition*).

The implementation of the classifier works following two sequential steps:

1. Classification of the sentences of the document according to the three classes described before;
2. Identification of the mathematical and the textual formulas included in the *Definition* class from the previous step.

4.1 Step 1: Sentences Classification

The ANNIE (A Nearly-New Information Extraction system) plug-in is the principal and most used component of the software GATE. It takes an input document that is annotated as output of the process. Step 1 is composed of a pipe of activities, where each depends on the output of the previous one and gives the input for the subsequent one. Almost each activity of this pipe is responsible for implementing and executing a specific information extraction technique. The whole pipe of phases composing Step 1 is described in the following:

1. 'Document Reset': this phase is responsible for deleting all the annotations already included in the document; it cleans the file so that it is ready for the whole annotation process;
2. 'Tokenisation': the tokeniser component in GATE implements a word segmentation technique. It divides the document in tokens, such as numbers, punctuations, etc. The tokeniser implementation in GATE uses regular expressions to give an initial annotation of tokens. Subsequently, for each token the following features are recognised: the 'string' itself; the 'kind', which is the set the token belongs to, such as word, number, symbol or punctuation; the 'orth', meaning the orthographical structure;
3. 'Gazetteer': this component is responsible for performing a Named Entity Recognition (NER) technique. It identifies the names of the entities based on some lists. Such lists are simple text files with an entry for each line. Each list represents a set of names depending on a domain, such as cities, week days, organisations, etc. An indexation is present to give access to such lists. As default behaviour, the Gazetteer creates a special annotation named 'Lookup' for each entry found in the text. GATE gives the possibility to create a gazetteer with personalised lists;

4. 'Sentence Splitter': this component implements a Sentence Breaking technique. The Sentence Splitter divides the text in sentences and utilises a Gazetteer composed of a list of abbreviations that helps to distinguish the acronyms from the full stops ending the sentences. Every sentence is annotated with a *Sentence* type;
5. 'Part-of-Speech Tagger': the tagger component of GATE implements the part-of-speech tagging technique. Such a component adds to the previously obtained tokens the correct category, which has to be intended as a part-of-speech, namely a verb, a noun, a pronoun, etc. The used lexicon is derived by a training made on a big dataset of articles of *The Wall Street Journal*;
6. 'Semantic Tagger': in the ANNIE component such a tagger is based on the JAPE language, which uses a mixture of Java code and regular expressions. This phase is in charge of producing some specific annotations enriched with additional features (e.g., the annotation *Person* with feature *gender* and values *male* and *female*; the annotation *Location* with feature *locType* and values *region*, *airport*, *city*, *country*, *county*, *province* and *other*). Some values of the features are derived by the lists used by the Gazetteer, but they can also be customised depending on the specific needs. The implemented technique is the Named Entity Recognition (NER);
7. 'Orthographic Co-reference (Ortho Matcher) and Pronominal Co-reference': this component implements a co-reference resolution technique. The Ortho Matcher module allows the adding of relationships to the entities identified by the Semantic Tagger. Such a module assigns a type to the annotations that have not been tagged by the Semantic Tagger. The Pronominal Co-reference module performs a check on the pronouns included in the document and their context in order to obtain the flow of references to the same entity the pronoun is referring to.

After the preprocessing pipe of activities, the machine learning plug-in included in GATE is executed. It is responsible for executing a sentence classification process, aimed to classify the sentences of the document depending on the three classes described in Section 3. In the SLA Classification context several features are added to a customisation of such machine learning component in GATE. The input to this component is the set of sentences obtained by the Sentence Splitter. Then, it has to recognise the 'Class' which is an annotation added to the semantic tagger; the feature of such annotation is *type* and its values are *Definition*, *Value*, *Not definition*.

4.2 Step 2: Formulas Identification

This step is dedicated to the extraction of mathematical formulas and parameter values from the sentences classified as *Definition* and *Value*. The input to this step is the output of Step 1, namely the classified sentences of the document given as input. Then, the sentences annotated as *definition* or *value* are elaborated by a dedicated transducer JAPE file, respectively. The output from this step is an additional feature to the already existing annotations.

The sentences *Definition* are analysed token by token; the current token is matched with a set of mathematical keywords, i.e., adding, subtracting, divided by, rate, etc. At the end of the process two features are extracted: the formula which is the definition of the parameter expressed mathematically; the period, which is the time interval during which the constraint has to be valid.

From the sentences classified as Value some other features are extracted: the numeric value of the parameter; its unit measure; a condition that determines how to compare the actual value; the name of the parameter; the value time, i.e. is the time period during which the constraint has to be valid.

5 ASSESSMENT

The whole process of SLA text recognition is assessed with a preliminary approach in order to validate the behaviour of the proposed GATE customisation, so some experiments are run in order to obtain the necessary information. The experiments utilise a total of 36 SLAs where 27 are from some cloud providers and the remaining 9 are from some SOA-based Web Services. Two different assessments have been performed, each concerns a step of the proposed automation, namely Step 1 and Step 2 described above.

5.1 Step 1: Sentences Classification

The SLA documents have been used to feed the sentence splitter component of the software and then all the identified sentences have been manually annotated with the correct class. The number of sentences among all the 36 SLAs classified as *Definition* is 71, while the *Value* sentences are 39; the total number of identified sentences is 2016, so the sentences classified as *Not definition* and discarded by the assessment is 1906. A leave-one-out cross-validation method has been performed: the dataset is divided in training set and test set for a total of 36 runs. In every run, the training set is composed of 35 SLA documents and

the test set of 1. We run a total of 5 different classification algorithms: Support Vector Machine (SVM), Perceptron Algorithm with Uneven Margins (PAUM), Naive Bayes, K-nearest neighbour (KNN) and C4.5 Decision Tree algorithm.

For the performance analysis, we measured precision, recall and F-measure. Precision is intended as the ratio between the number of true positive results (TP), namely the sentences correctly classified and the sum of true positives and false positives (FP). Recall is calculated as the ratio between the number of true positive results and the sum of true positives and false negatives (FN), namely the sentences classified in another class but actually belonging to it. The F-measure is the harmonic average of precision and recall, calculated as:

$$F\text{-}measure = 2 * \frac{Precision * Recall}{Precision + Recall}$$

Some parameters are set for 3 of the 5 classification algorithms; the 2 left unaltered are Naive Bayes and C4.5.

The parameters set for SVM are cost and value. Cost (C) is the soft margin allowed for the errors and it was set in the range 10^{-3} to 10^{3} in multiples of 10. Value τ represents the irregular margins of the classifier. τ is set to 1 and it represents the standard execution of SVM. When the training set items have a few positive examples and a big number of negative ones, τ is set to a value lower than 1 in order to have a better F-measure; in this experiment the training set items have some positive examples than negatives, so τ varies between 0 and 1 with jumps of 0.25.

In PAUM the number of negative and positive margins can vary. They are set to the values in the following sets: negative margin (n) in -1.5, -1, -0.5, 0, 0.1, 0.5 and 1.0; positive margin (p) in -1, -0.5, 0, 0.1, 0.5, 1, 2, 5, 10 and 50.

In the KNN algorithm the parameter representing the number of neighbours can vary: the default value of neighbours number (k) is set to 1, but the more such a number increases, the more the classification noise is reduced but it lows the precision; in this experiment it varies between 1 and 5.

5.1.1 Results

Figures 2 and 3 show the results of the best classifiers for *Definition* and *Value* classes respectively.

The SVM algorithm on the classes *Definition* and *Value* performs an increase of the values for precision, recall and F-measure coincident with the increase of the C parameter; they become more stable with $C > 1$. The final result is a value for precision, recall and F-measure lower than 60 %: this is caused by some outfit that *Definition* parameters have, not matching with the most common patterns, which play an influence on the classification. The *Value* sentences are not so many, usually one per document, so the values for precision and F- measure are lower than 50 % and the recall is lower than 60 %.

The PAUM algorithm on the class *Definition* performs irregularly when the parameters n and p increase. The best results happen with $n = -0.5$ and $p = 0$, with precision and recall > 50 % and F-measure a bit lower than 50 %. On the class *Value* PAUM performs better with $n = 0.5$ and $p = 1$, with values for precision, recall and F-measure lower than 40 %.

K-NN on the classes *Definition* and *Value* performs worse than SVM and PAUM; it classifies worse when the parameter k increases. The best result on the class *Definition* is obtained with k set to 1, but such a result is worse than SVM and PAUM due to the F-measure value around 40 %, the precision a bit lower than 50 % and a recall lower than 40 %. On the class *Value*, KNN performs the best results with $k = 1$ and becoming stable on $k \leq 4$. The best results output an F-measure, precision and recall values around 30 %.

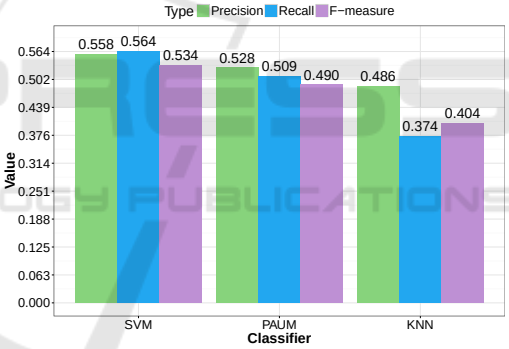

Figure 2: Definition class comparison results.

Thus, in our experiment the classification algorithm that behaves better for the classes *Definition* and *Value* is SVM with the cost C parameter set to 1. It outputs precision and recall values bigger than 55 % on *Definition* and a bit lower than 50 % for *Value*. The classification algorithms without parameters, namely C4.5 and Naive-Bayes have bad behaviours, with precision and recall of 0, indeed they fail to classify the sentences for both *Definition* and *Value* classes.

5.2 Step 2: Formulas Identification

This assessment preliminary phase is dedicated to verify effectiveness of the transducer components customised in GATE and their capability to correctly recognise the mathematical formulas. The utilised dataset is identical to the one of the previous assessment

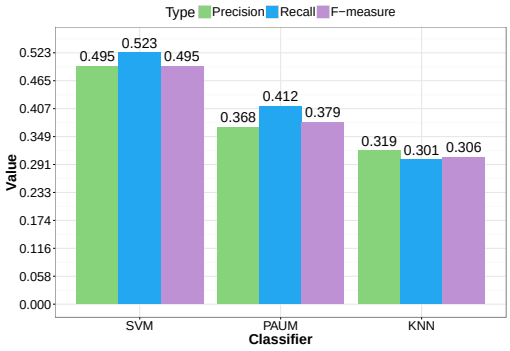

Figure 3: Value class comparison results.

phase, namely 36 SLAs documents. The training set is composed of the whole dataset items; every sentence of the documents is manually annotated with the *Definition*, *Value* and *Not definition* classes. The mathematical formulas of the 36 documents are 25 on the total of 71 *Definitions* annotated. The assessment utilises some distance formulas to calculate the errors: the Levenshtein distance, the Jaccard similarity and the cosine similarity.

The Levenshtein distance represents the minimum number of elementary changes necessary to transform string $\mathcal{A}$ to string $\mathcal{B}$, where $\mathcal{A} \neq \mathcal{B}$. An elementary change can be the cancellation, replacement or insertion of a single character in $\mathcal{A}$.

The Jaccard similarity is mainly utilised to compare similarity and diversity of sample sets. In this case the sample is a string and the single elements are the single words composing it. The value is obtained as a ratio between the difference of the sizes of union and intersection sets of words by the size of the union set.

The similarity cosine is an heuristic technique that measures the similarity of two numeric vectors by calculating their cosine. In textual contexts, the vectors are composed of the frequency of the words considered in the strings to calculate the similarity on. The word frequency is the number of times such a word recurs in a text, so the n^{th} element of the vector represents the frequency of word k, 0 otherwise. The values of the vector elements vary in a range for 0 to $+1$, where $+1$ indicated that the both texts include exactly the same words, but not necessarily in the same order; 0 indicates that both texts have no words in common.

Also a manual semantic analysis is performed on couples of strings, in order to check if they have a different meaning in presence of a similar text.

5.2.1 Results

Figure 4 shows the obtained results.

The Levenshtein distance metric performs a check character by character; the average result is a similarity value of 91 %. The main difference between the oracle and the actual results relies on the presence of a blank space character nearby the parenthesis symbol, which does not alter the strings. In some cases the similarity value is lower, between 70 % and 80 % and this is due to some words being not present in the oracle text, or some numbers being textually-represented and not by digits.

The Jaccard similarity metric gives an average result of 57 %. The reason is due to the checks performed on single words, where a word is a sequence of characters enclosed by blank spaces. In the previous metric the presence of blank spaces was already detected and such a presence consequently contributes to lower the average result of the Jaccard metric because a difference by a final or ending character is interpreted as the presence of two totally different strings, e.g. ‘month)’ and ‘month)’.

The similarity cosine gives an average result of 71 %. It also verifies the similarity of two strings at words level, but considering the word frequency instead of the set of characters composing it.

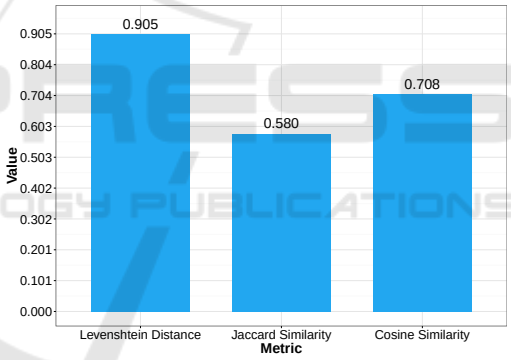

Figure 4: Distance metrics results.

The results of the manual semantic analysis indicates that the implementation output does not differ much from the oracle. Analysing the details of the results, the main differences with the oracle concern again some blank space characters or the representation of numbers with strings instead of digits, which do not represent a semantic difference between the two strings. The manual analysis of the results showed that the actual output is semantically equal to the oracle.

6 CONCLUSIONS AND FUTURE WORK

In this paper a preliminary approach for SLAs text recognition in computing services provisioning is pro-

posed.

Such proposal is then assessed with some classification algorithms (i.e., SVM, PAUM, Naive Bayes, KNN and C4.5) running on a dataset of 36 SLAs publicly accessible from service providers. The results showed that the use of SVM classifier performs better than others, nevertheless they are not so good in terms of the accuracy measures we employed (precision, recall and F-measure). In the future we intend to investigate how to improve the approach possibly using other classifiers.

REFERENCES

Baset, S. A. (2012). Cloud slas: Present and future. *ACM SIGOPS Operating Systems Review*, 46(2):57–66.

Brandic, I., Emeakaroha, V. C., Maurer, M., Dustdar, S., Acs, S., Kertesz, A., and Kecskemeti, G. (2010). Laysi: A layered approach for sla-violation propagation in self-manageable cloud infrastructures. In *IEEE Computer Software and Applications Conference Workshops (COMPSACW)*, pages 365–370.

Cedillo, P., Gonzalez-Huerta, J., Abrahão, S. M., and Insfrán, E. (2014). Towards monitoring cloud services using modelsrun.time. In *Workshop on Modelsrun.time*.

De Marco, L., Ferrucci, F., and Kechadi, M.-T. (2015). Slafm - a service level agreement formal model for cloud computing. In *International Conference on Cloud Computing and Services Science (CLOSER)*.

Emeakaroha, V. C., Calheiros, R. N., Netto, M. A. S., Brandic, I., and De Rose, C. A. (2010). Desvi: An architecture for detecting sla violations in cloud computing infrastructures. In *International ICST Conference on Cloud Computing (CloudComp)*. Citeseer.

Emeakaroha, V. C., Ferreto, T. C., Netto, M. A. S., Brandic, I., and De Rose, C. A. F. (2012a). Casvid: Application level monitoring for sla violation detection in clouds. In *IEEE Annual Computer Software and Applications Conference (COMPSAC)*, pages 499–508.

Emeakaroha, V. C., Netto, M. A. S., Calheiros, R. N., Brandic, I., Buyya, R., and De Rose, C. A. F. (2012b). Towards autonomic detection of sla violations in cloud infrastructures. *Future Generation Computer Systems*, 28(7):1017–1029.

European Commission and DG CONNECT Cloud Select Industry Group (2014). Cloud service level agreement standardisation guidelines.

Grishman, R. (1997). Information extraction: Techniques and challenges. In *Information Extraction: A Multidisciplinary Approach to an Emerging Information Technology*, pages 10–27. Springer.

ITIL (2016). Information technology infrastructure library (itil).

Maurer, M., Brandic, I., and Sakellariou, R. (2012). Self-adaptive and resource-efficient sla enactment for cloud computing infrastructures. In *IEEE International Conference on Cloud Computing (CLOUD)*, pages 368–375. IEEE.

Morshedlou, H. and Meybodi, M. R. (2014). Decreasing impact of sla violations: A proactive resource allocation approach for cloud computing environments. *IEEE Transactions on Cloud Computing*, 2(2):156–167.

Muller, C., Oriol, M., Franch, X., Marco, J., Resinas, M., Ruiz-Cortés, A., and Rodriguez, M. (2014). Comprehensive explanation of sla violations at runtime. *IEEE Transactions on Services Computing*, 7(2):168–183.

Rumbaugh, J., Jacobson, I., and Booch, G. (2004). *The Unified Modeling Language Reference Manual*. Pearson Higher Education.

Tan, J. (2001). Forensic readiness, technical report. Technical report, stake Inc., Cambridge USA.