10th Symposium on Languages, Applications and Technologies

SLATE 2021, July 1–2, 2021, Vila do Conde/Póvoa de Varzim, Portugal

Edited by

Ricardo Queirós
Mário Pinto
Alberto Simões
Filipe Portela
Maria João Pereira

OASICS

Editors

Ricardo Queirós
Escola Superior de Media Artes e Design, Politécnico do Porto, Portugal
ricardoqueiros@esmad.ipp.pt

Mário Pinto
Escola Superior de Media Artes e Design, Politécnico do Porto, Portugal
mariopinto@esmad.ipp.pt

Alberto Simões
Instituto Politécnico do Cávado e do Ave, Portugal
asimoes@ipca.pt

Filipe Portela
Universidade do Minho, Portugal
cfp@dsi.uminho.pt

Maria João Pereira
Instituto Politécnico de Bragança, Portugal
mjoao@ipb.pt

ACM Classification 2012
Theory of computation → Formal languages and automata theory; Information systems → Data management systems; Information systems → World Wide Web; Computing methodologies → Natural language processing

ISBN 978-3-95977-202-0

Published online and open access by
Schloss Dagstuhl – Leibniz-Zentrum für Informatik GmbH, Dagstuhl Publishing, Saarbrücken/Wadern, Germany. Online available at https://www.dagstuhl.de/dagpub/978-3-95977-202-0.

Publication date
August, 2021

Bibliographic information published by the Deutsche Nationalbibliothek
The Deutsche Nationalbibliothek lists this publication in the Deutsche Nationalbibliografie; detailed bibliographic data are available in the Internet at https://portal.dnb.de.

License
This work is licensed under a Creative Commons Attribution 4.0 International license (CC-BY 4.0): https://creativecommons.org/licenses/by/4.0/legalcode.
In brief, this license authorizes each and everybody to share (to copy, distribute and transmit) the work under the following conditions, without impairing or restricting the authors' moral rights:
- Attribution: The work must be attributed to its authors.

The copyright is retained by the corresponding authors.

Digital Object Identifier: 10.4230/OASIcs.SLATE.2021.0
ISBN 978-3-95977-202-0 **ISSN 1868-8969** **https://www.dagstuhl.de/oasics**

OASIcs – OpenAccess Series in Informatics

OASIcs is a series of high-quality conference proceedings across all fields in informatics. OASIcs volumes are published according to the principle of Open Access, i.e., they are available online and free of charge.

Editorial Board

- Daniel Cremers (TU München, Germany)
- Barbara Hammer (Universität Bielefeld, Germany)
- Marc Langheinrich (Università della Svizzera Italiana – Lugano, Switzerland)
- Dorothea Wagner (*Editor-in-Chief*, Karlsruher Institut für Technologie, Germany)

ISSN 1868-8969

https://www.dagstuhl.de/oasics

To all whose effort keeps SLATE alive.

Contents

Preface

This book of proceedings compiles the accepted papers for the Tenth edition of the Symposium on Languages, Applications and Technologies, SLATE'2021. It is an honour to organize and host this edition, being a special milestone for the conference.

In the last decade it has shown that there is, in fact, similarities in the techniques used for processing the different types of languages we use daily, ranging from the natural languages used for the communication between humans, the programming languages used to instruct the behavior of computers, and finally the interchange languages, used in the communication between machines.

The Tenth edition bring us some nostalgic, by this way we want to share the conference with all. The conference is held in a hybrid mode, as COVID-19 pandemic looks controlled. We are organizing the conference online and opened the possibility of attending in person at the School of Media Arts and Design, in Vila do Conde/Póvoa de Varzim, which kindly hosts this tenth edition.

In this tenth edition of SLATE, we received several interesting contributions which demonstrate the relevance and interest in the languages processing research and which cover almost equally the three facets of the languages materialized in the three conference tracks.

The tracks are addressing several applications of the languages like data crawlers, sentimental analysis, machine learning applied to languages processing, code generators and parsers.

We want to thank the many people without whom this event would never have been possible: all the Members of the Scientific Program Committee for their valuable effort reviewing the submissions, contributing with corrections and new ideas, and helping on deciding the final list of accepted paper; the members of the Organizing Committee, for the help on dealing with the bureaucratic issues; the invited Speakers (*Ricardo Correia* and *Diana Santos*) for accepting our invitation to share their knowledge; and finally, thank you all the Authors for their contributions to SLATE with their current projects and research problems.

List of Committees

Program Chairs

Ricardo Queirós
ESMAD/P.PORTO, uniMAD &
CRACS/INESC TEC

Mário Pinto
ESMAD/P.PORTO & uniMAD

Alberto Simões
2Ai-School of Technology, IPCA

Filipe Portela
Universidade do Minho

Maria Joao Pereira
CeDRI, Instituto Politécnico de Bragança

Organization Committee

Ricardo Queirós
ESMAD/P.PORTO, uniMAD &
CRACS/INESC TEC

Mário Pinto
ESMAD/P.PORTO & uniMAD

Alberto Simões
2Ai-School of Technology, IPCA

Teresa Terroso
ESMAD/P.PORTO & uniMAD

Jorge Lima
ESMAD/P.PORTO

Scientific Committee

Alberto Simões
Instituto Politécnico do Cávado e Ave

Alda Gancarski
Institut Mines-Telecom

Alexander Paar
Duale Hochschule Schleswig-Holstein

Alexandre Rademaker
IBM Research Brazil

Ana Alves
University of Coimbra

Antonio Leitao
Universidade de Lisboa

António Teixeira
University of Aveiro

António Miguel Cruz
Instituto Politécnico de Viana do Castelo

Arkaitz Zubiaga
Queen Mary University of London

Barrett Bryant
University of North Texas

Bostjan Slivnik
University of Ljubljana

Brett Drury
LIAAD-INESC-TEC

Daniela Da Cruz
Checkmarx

Diana Santos
University of Oslo

Dietmar Seipel
University of Wuerzburg

Dušan Kolář
Brno University of Technology

Fernando Batista
Instituto Universitário de Lisboa

Filipe Portela
University of Minho

Geylani Kardas
Ege Üniversitesi

Hugo Gonçalo Oliveira
University of Coimbra

Irene Rodrigues
Universidade de Évora

Irene Castellón
Universitat de Barcelona

Ivan Luković
University of Belgrade

Jakub Swacha
University of Szczecin

Jan Janousek
Czech Technical University Prague

Jan Kollar
Technická univerzita v Košiciach

Jaroslav Porubän
Technical University of Košice

Jean-Christophe Filliatre
Université Paris-Saclay

João Saraiva
University of Minho

João M. Lourenço
Universidade NOVA de Lisboa

José Carlos Ramalho
University of Minho

José Carlos Paiva
University of Porto

José João Almeida
Universidade do Minho

Jose Luis Sierra
Universidad Complutense de Madrid

José Paulo Leal
University of Porto

Josep Silva
Universitat Politècnica de València

Luis Morgado Da Costa
Nanyang Technological University

Luís Ferreira
Instituto Politécnico do Cávado e Ave

Luísa Coheur
Instituto Superior Técnico

Maria João Pereira
Instituto Politécnico de Bragança

Mario Berón
National University of San Luis

Mário Pinto
ESMAD, Politécnico do Porto

Marjan Mernik
University of Maribor

Mikel Forcada
Universitat d'Alacant

Pablo Gamallo
University of Santiago de Compostela

Pedro Rangel Henriques
University of Minho

Ricardo Queirós
ESMAD, Politécnico do Porto

Ricardo Rodrigues
University of Coimbra

Ricardo Rocha
University of Porto

Salvador Abreu
University of Évora

Simão Melo de Sousa
Universidade da Beira Interior

Tomaz Kosar
University of Maribor

Xavier Gómez Guinovart
Universidade de Vigo

List of Authors

Alberto Simões
2Ai Lab, School of Technology, IPCA,
Barcelos, Portugal
asimoes@ipca.pt

Ana Alves
CISUC, Instituto Politécnico de Coimbra,
Portugal
ana@dei.uc.pt

André Santos
CRACS & INESC Tec LA
University of Porto, Portugal
andrefs@andrefs.com

António Neto
Caixa Mágica Software
Lisboa, Portugal
antonio.neto@caixamagica.pt

António Teixeira
IEETA, DETI
University of Aveiro, Portugal
ajst@ua.pt

Carlos Coutinho
Caixa Mágica Software
ISTAR-IUL, ISCTE
Instituto Universitário de Lisboa
carlos.coutinho@caixamagica.pt

Cátia Tavares
ISCTE, Instituto Universitário de Lisboa
Av. das Forças Armadas, Portugal
catiattavares@gmail.com

Cristiana Esteves Araújo
Centro ALGORITMI
Universidade do Minho, Braga, Portugal
decristianaaraujo@hotmail.com

Daniel Weidner
University of Würzburg
Würzburg, Germany
daniel.weidner@stud-mail.
uni-wuerzburg.de

Dietmar Seipel
University of Würzburg
Würzburg, Germany
dietmar.seipel@uni-wuerzburg.de

Emanuel Matos
IEETA, DETI
Universidade de Aveiro, Portugal
easm@ua.pt

Fernando Batista
INESC-ID & ISCTE-IUL
Instituto Universitário de Lisboa, Portugal
fernando.batista@iscte-iul.pt

Filipa Santos
University of Minho
Braga, Portugal
a83631@alunos.uminho.pt

Hugo Cardoso
University of Minho
Braga, Portugal
a85006@alunos.uminho.pt

Hugo Gonçalo Oliveira
CISUC, DEI
Universidade de Coimbra, Portugal
hroliv@dei.uc.pt

João Coelho
Caixa Mágica Software
Instituto Superior Técnico
Lisboa, Portugal
joao.coelho@caixamagica.pt

João Costa
University of Minho
Braga, Portugal
a84775@alunos.uminho.pt

João Santos
INESC-ID & Instituto Superior Técnico
Lisboa, Portugal
joao.l.santos@tecnico.ulisboa.pt

José Carlos Paiva
CRACS: INESC-Porto LA
DCC - FCUP, Porto, Portugal
jose.c.paiva@inesctec.pt

José Carlos Ramalho
Department of Informatics
University of Minho
Braga, Portugal
jcr@di.uminho.pt

José João Almeida
Centro Algoritmi
Universidade do Minho
Braga, Portugal
jj@di.uminho.pt

José Paulo Leal
CRACS & INESC Tec LA
University of Porto, Portugal
zp@dcc.fc.up.pt

José Santos
CISUC, DEI
Universidade de Coimbra, Portugal
santos@student.dei.uc.pt

Jukka Nurminen
Department of Computer Science
University of Helsinki, Finland
jukka.k.nurminen@helsinki.fi

Leandro Costa
University of Minho
Braga, Portugal
leandro.costa16@hotmail.com

Leonor Llansol
INESC-ID & Instituto Superior Técnico
Lisboa, Portugal
leonor.llansol@tecnico.ulisboa.pt

Luís Duarte
CISUC, DEI
Universidade de Coimbra, Portugal
lduarte@student.dei.uc.pt

Luís Cunha
University of Minho
Braga, Portugal
a83099@alunos.uminho.pt

Luísa Coheur
INESC-ID & Instituto Superior Técnico
Lisboa, Portugal
lcoheur@edu.ulisboa.pt

Manuel Sousa
University of Minho
Braga, Portugal
manuelgcsousa@gmail.com

Marcin Kowiel
F-Secure Corporation, Poland
marcin.kowiel@f-secure.com

Maria João Pereira
Research Centre in Digitalization and
Intelligent Robotics
Instituto Politécnico de Bragança, Portugal
mjoao@ipb.pt

Mariana Gaspar
INESC-ID & Instituto Superior Técnico
Lisboa, Portugal
mariarpx@gmail.com

Mário Rodrigues
IEETA, ESTGA
Universidade de Aveiro, Portugal
mjfr@ua.pt

Miguel Brito
Centro Algoritmi
University of Minho
Guimarães, Portugal
mab@dsi.uminho.pt

Miguel Tavares
Caixa Mágica Software
ULHT - Universidade Lusófona de
Humanidades e Tecnologias
miguel.tavares@caixamagica.pt

Nuno Oliveira
Checkmarx, Braga, Portugal
nuno.oliveira@checkmarx.com

Nuno Ramos Carvalho
Guimarães, Portugal
narcarvalho@gmail.com

Pablo Gamallo
CiTIUS
Universidade de Santiago de Compostela
Galiza, Spain
pablo.gamallo@usc.es

Paulo Martins
University of Minho
Braga, Portugal
paulo.jorge.pm@gmail.com

Pedro Miguel
IEETA, DETI
Universidade de Aveiro, Portugal
pedro.miguel.rafael@gmail.com

Pedro Rangel Henriques
University of Minho
Braga, Portugal
pedrorangelhenriques@gmail.com

Ricardo Pereira
University of Minho
Braga, Portugal
ricardo-97-pereira@hotmail.com

Ricardo Queirós
CRACS: INESC-Porto LA, Porto
uniMAD: ESMAD, Polytechnic of Porto
Porto, Portugal
ricardo.queiros@gmail.com

Ricardo Ribeiro
ISCTE - Instituto Universitário de Lisboa
INESC-ID Lisboa, Portugal
ricardo.ribeiro@iscte-iul.pt

Rui Rodrigues
University of Minho
Braga, Portugal
a74572@alunos.uminho.pt

Salvador Abreu
Nova–Lincs, University of Évora
Portugal
spa@uevora.pt

Tiago Baptista
University of Minho
Braga, Portugal
a75328@alunos.uminho.pt

Tommi Mikkonen
Department of Computer Science
University of Helsinki, Finland
tommi.mikkonen@helsinki.fi

Válter Carvalho
University of Minho
Braga, Portugal
a84464@alunos.uminho.pt

Zafar Hussain
Department of Computer Science
University of Helsinki, Finland
zafar.hussain@helsinki.fi

Natural and Artificial Intelligence; Natural and Artificial Language

Diana Santos
Linguateca
University of Oslo, Norway

Abstract

This text starts by discussing what it means to be intelligent for humans and machines, what is the purpose of language, and how is human language fundamentally different from artificial languages. It presents the issue of values as one inescapable property of human language, and of human categorization in general, after reviewing five distinctive caracteristics of natural language. Then it proceeds to discuss static word embeddings, raising two questions: is the wisdom of the crowd an appropriate justification for using the underlying large text collections? And have the differences between languages been taken into account when intrinsically evaluating Portuguese word embeddings?

2012 ACM Subject Classification Computing methodologies → Machine learning; Computing methodologies → Natural language processing

Keywords and phrases Artificial Intelligence, Natural Language Processing

Digital Object Identifier 10.4230/OASIcs.SLATE.2021.1

Category Invited Talk

Acknowledgements I want to thank Alberto Simões for inviting me to SLATE.

1 Introduction

This invited talk gave me the opportunity to reflect on artificial intelligence (AI) and natural language processing (NLP) after ca. 35 years of having been introduced to both in my student years in the 80'es.

In fact, I have worked for more than 35 years in natural language processsing, and was a student of João Pavão Martins, who belonged to the first generation of artificial intelligence scholars in Portugal. After having launched Linguateca in the end of the 90's, with the goal of fostering R&D in the computational processing of Portuguese, I have taught at the Faculty of Humanities of the University of Oslo for the last ten years.

Therefore I thought it appropriate to share with you some thoughts about natural language and artificial intelligence, widening the scope of influence to philosopy, psychology, statistics, medical history and sociology, while somehow following up a previous discussion of the specificity of natural language as opposed to artificial languages, which I prepared for PROPOR and SBLP in 2006 [17].

2 A note on terminology

SLATE has a very interesting approach to language seen from a computer science view: it divides the field among human and computer languages, or better, in a tripartite way: computer-computer languages, human-computer languages, and human-human languages.

© Diana Santos;
licensed under Creative Commons License CC-BY 4.0

However, there's no denying that *natural language* is by far the most common term when dealing with human language. And, considering the four terms in my title, the most common of all is obviously *artificial intelligence*. Interestingly, the only call in SLATE which mentioned intelligence was CCL, which used instead *computer intelligence*.

Now, the purpose of my title was exactly to point out that, no matter the fact that *natural* and *artificial* are antonyms, natural language does not work as expected, that is, clustering natural intelligence with natural language, and artificial intelligence with artificial languages. Quite the contrary, the pseudo-term *natural intelligence* does not exist (except perhaps as a pun): one always contrasts *artificial intelligence* with *human intelligence*. And in computer science one generally uses just *languages* to refer to programming languages. I would say that *artificial languages* are rather those natural languages invented by people, like Esperanto.

So, and despite the apparent symmetry and plausible analogy among the two pairs, in reality natural language is much more complicated and less predictable. It is the terms *natural language* and *artificial intelligence* which are linked: often natural language processing is considered a part of AI, or even the hallmark of AI – remember Turing's test.

Anyway, terms have a history, and evolve, as language in general does. Therefore they are not defined once and for all and, in fact, they mean very different things forty years ago and now.

Let me then start trying to discuss intelligence.

3 A linguistic inquiry of intelligence

While the concept of intelligence is something of the realm of philosophy or psychology, the linguistic approach to meaning is to investigate how the word is used, and to what it is applied, as Firth aptly expressed in his dictum "a word is defined by the company it keeps".

I will be using in this talk English and Portuguese, so I should hasten to say that the meaning depends on language:

- in English, there is *intelligent* and *clever* and *smart*, *cunny*, etc.
- in Portuguese, there is *inteligente* and *esperto* and *sagaz*, *perspicaz*, *astuto*, etc.

It is rather the clusters represented by these quasi-synonyms that have some sort of translation relation. The particular shades of meaning in each language are language-specific, and virtually impossible to precisely compare.

In any case, and restricting now our look at the words *intelligent* and *inteligente*, it is not hard to see that people use these words in a much broader way than dictionary definitions would foresee. I give just some random examples, to illustrate that its use is very dependent on context, and that intelligence can be assigned to a device or to the person who devised it, and it can even simply indicate a better solution compared to a previous one:

- Another classic example of such parallel communications is the device that we will use as the example in this section, the ***intelligent liquid crystal display (LCD) module***. (*Introduction to Mechatronic Design*)
- newspaper parlance: *sistema de semáforos inteligentes...* (intelligent traffic lights)
- gadgets *smart phones* or techniques *smart queries*, or even environments *casas inteligentes* (intelligent homes)

Anyway, the most important point I would like to make about the concept of intelligence when it appears in *human inteligence* and *artificial intelligence*, is that, again, the meaning of the compounds does not necessarily imply that we are talking about the **same** concept of *intelligence*. One could of course point to the paralellism of human vs. artificial/machine and conclude that the difference is who possesses/displays intelligence. But it is not hard to see that compounds can also completely change the meaning of the noun, like in *human nature* vs. *tropical nature*.

And this reminds me of one of the old lessons of AI: it is not by copying birds that mankind managed to fly. It was by understanding the physical principles birds use.

For most of the activities we use the word *intelligent*, either machines or humans are considered intelligent when they do it, but most often than not, not both. Let me give some examples:

- RoboCup – soccer playing
  While this is a display of intelligent robots, who has ever used the word intelligent when describing how one's child is good at soccer?
- Chess or Go playing
  Human champions of chess or Go are usually considered highly intelligent, but computers have surpassed them by (mainly) the capacity to foresee thousands of outcomes.
- Encyclopaedic answer
  The possibility of knowing a lot of facts in the past was a hallmark of a learned, intelligent person, but the access of a computer to more facts than a human can store has given a computer the lead (cf. the Jeopardy contest)
- Poetry writing
  Poets like Camões or Shakespeare are considered geniuses, so their poetic intelligence is high, although sensibility and way with words are ingredients equally necessary. So far machine poetry is considered funny and sometimes interesting, but not really intelligent.
- Lying to protect other's feelings
  Emotional intelligence like the one displayed by people who choose different versions of a story, or even lie, something that most humans do, is so far outside the realm of computer communication.
- Visually identifying a cancerous tissue
  This is an activity that required specific training for radiologists and other medical staff, and that has been partially taken over by image recognition systems built based on machine learning over large amounts of data. However, an apparently better-than-human system got it totally wrong with given images produced by a different vendor, something that would not fool a human. That forced us to caution "blind" machine learning. See [11] for more information on this.
- Recognizing people in the street
  These capabilities are different from human to human, but may be called social intelligence. The recent surveillance attempts by authoritarian governments using AI systems to identify populations also call into attention this ability.
- Identifying a dialect
  A related ability, that of being able to detect a specific dialect, which is being appropriated by intelligent computers, can also be considered a sign of human intelligence.

The examples could be multiplied at will. Their point is basically that not all these activities are equally praised – and are obviously differently implemented – in humans and in computers.

I would like to here support a new paradigm called hybrid intelligence (HI), beautifully presented by Frank van Harmelen in his 2020 keynote at IC3K, entitled "Hybrid Intelligence: AI systems that collaborate with people, instead of replacing them"[24]. He suggests augmenting human intellect and capabilities instead of replacing them, with the aim of achieving goals unreachable by either humans or machines alone. Van Harmelen argues that AI is unaware of norms and values; reasons; and contexts, and that it is absolutely necessary to have explanations in human society, so that decisions can be disputed. And explanations, he argues, need to be grounded on values, norms, motives, committments and goals.

4 What is intelligence; what is language

I would briefly suggest four criteria for intelligence, namely

- Learning
  One of the most mentioned properties of intelligence is learning. An intelligent person learns from failure. But is learning enough to become intelligent? If we do not know what a machine has learned, we cannot predict, or explain, unexpected failures. If one does not critically choose from whom to learn, one can learn things that are wrong, incorrect, even dangerous if one has no critical sense...
- Knowledgeability
  At least for a person, the more she knows, clearly the more intelligently she can behave. Computers can "know" many more facts than humans, but there are several problems with facts: they depend on theory, they are not consensual, and it requires intelligence to generalize over them. Humans are extremely good at creating generalizations and decide, with incomplete information. Plus, knowing is knowing where to find. Who are the authorities. Who to trust.
- Alternative worlds
  Humans very easily devise alternative worlds: if / as if. We often decide based on imagining different realities, computing consequences that are just thought, even pursuing "impossible" paths. Humans can define intensional concepts, computers so far only extensional ones.
- Context awareness
  An intelligent being/device/system changes behaviour depending on the context, reacting to the environment/situation in a proper way. This is perhaps the property that has been endowed most artificial systems, but to a certain extent only. Human reactions embody assumptions, and beliefs. Humans are able to revise and change their beliefs, and to reason with incomplete knowledge. One of the properties that allow us to do it are emotions [20]. See for example the paradigm of affective computing proposed by Picard [12] to endow machines with some emotions.

But let me point out that, anyway and pace Turing, mimicking a human is different from being a human.

If we look at language – and let me emphasise again that we only talk of *natural language* when we process it with computers, therefore doing AI – the first remarkable thing is that when humans devise languages (programming languages) for computers (which is called language engineering), they endow them with properties very different from those they use in human languages.

In PROPOR 2006 I had a keynote [17] on what distinguished between natural and artificial languages, arguing for the following distinctive properties of atural language:

1. Metaphorical nature
2. Context dependency
3. Reference to implicit knowledge
4. Vagueness
5. Dynamic character (evolution and learnability)

Here I would like to add a sixth characteristic that I believe is extremely important, namely that natural language embodies **values**.

To argue for this, I start by discussing what natural language is for, borrowing heavily from Ellis [2] and Steiner [22]: language is a pre-requisite, or the way, for humans to understand the world (through characterization), language allows one to do things (with others), and language is instrumental to create a shared community, as well as to put others (the onew that do not speak our language) outside.

Human language is a source of power, as sociolinguists and sociologists have argued for a long time.

But another characteristic that is not so much discussed, and which may at first look like a truism, is that it is human-centered, in that the values that constitute it are all relative to Man.

Let me give two examples of natural language concepts that illustrate this. Take first disease. As scholars of medical studies have pointed out, disease is a concept related to humans. As Sedgwick [21] puts it:

> There are no illnesses or diseases in nature. (...) The medical enterprise is from its inception value-loaded; it is not simply an applied biology, but a biology applied in accordance with the dictates of social interest.
> (...) All illness, whether conceived in localized bodily terms or within a larger view of human functioning, expresses both a social value-judgment (contrasting a person's condition with certain understood and accepted norms) and an attempt at explanation (with a view to controlling the disvalued condition).

Only the states which have an undesired effect for the goals Man pursues receive this description. In other words, a disease is something natural that is considered bad for humans (or pets or crops).

Take now the concept of weed: it is a plant that is considered bad for human gardens (or for the concept that humans have of gardens, or of plants in general). We know that this is not a biological property of a plant, it is a value that humans attach to it.

Generalizing, the concept of good or bad is something that pervades our language. Values are essential to communicate among humans. And they are absent from computer languages.

Although we can generalize to language in general (not only human language) that the purpose of language is knowledge representation, and communication, in order to do things with others, and to inform or disinform others, it is only among humans that values are shared and communicated.

As a side remark, and to highlight the importante of values for humans, consider another keynote at SLATE, on "What Programming Language Design Taught Me About Life". In the abstract, Pitman [13] states:

> I came to see languages as much more complex entities than mere functional behavior or stylized syntax. Languages are about community and shared values – and not just the kinds of values that get returned from a function call. The choices a language designer makes will attract certain users and alienate others

So, even in the (in principle, value-free) design of programming languages (to communicate with computers) the issue of values is paramount. (And it is (also?) with humans that language engineering is preoccupied with, not (only?) computers.)

Following the path of looking into other keynotes at SLATE, in last year's on "How Humans Succeed While Failing to Communicate", Graat [6] concludes:

> The task of making a computer understand human communication therefore seems to be the hardest thing to do.

My answer to this is that maybe it is not necessary that computers understand our communication, maybe the right choice is to communicate differently) with them (as we so far have done). And communicate other things.

Because, I would argue that to assign value is something absolutely human: good and bad do not exist in nature or reality.

In order to evaluate, you have to compare with something else. Usually, human judgement.

But – and this is a highly relevant detail – not all judgements are consensual. All of us are aware of ethical paradoxes, different legal opinions, etc.

In fact, and even in a more general sense, cultures have been defined (by Delfim Santos [16]) as different rankings of values.

The bottom line is that human language always includes values, and these values are inherently human.

5 Word embeddings and the wisdom of the crowd

I turn now to one specific technology used in NLP, which one may say has come to dominate NLP in the last years: word embeddings, a form of representing context based on co-occurrence. Based on machine learning over big text collections (the crowd) – see [18] for looking critically at size.

I am obviously not the first one who looks critically at this technology. In fact, an excellent presentation this year by Rada Mihalcea [10] has voiced at least the following concerns: unpredictable, unstable, crowd-dependent, human-ununderstandable, climate-unfriendly, corporation-owned.

My impression is that often one uses word embeddings in a way aptly described by the Portuguese expression *atirar o barro à parede* (let's see if it sticks, if it works), without even providing a rationale for using them.

But another criticism I want to raise here is that people "playing" with word embeddings do not take different languages seriously.

Let us first investigate what underlies the use of word embeddings: the assumption that the larger the set of texts one uses, the more one (system or person) learns – in other words, that quantity leads to quality. This is actually backed by a scientific observation done by statisticians (Galton [3]) more than one hundred years ago, with the name *the wisdom of the crowd*, which states that the median estimate of a group can be more accurate than the estimates of individual experts.

But the application of this "law" has some flaws, as I will proceed to argue.

First, it is not the size itself of the crowd that is the relevant factor: one has to ask the right crowd. What if had asked SLATE's audience two simple questions, one dealing with the meaning of a Norwegian expression, and the other about the family of an Angolan politician? Assuming that no one in the audience was aquainted with the politics of Angola, neither knew Norwegian, no matter how many answers I would get, I would not trust them, compared with those provided by one single Norwegian speaker, or one single Angolan historian.

Second, if one (person) reads a lot of texts, s/he will be able to understand that some texts oppose the others, or are based on others, and s/he will use her or his intelligence to make sense of what s/he read. Not taking everything at face value, or considered equally true or reliable.

Last but not least, recent studies in psychology have shown that social pressure undermines the effect of the wisdom of the crowd. The crowds studied by Galton had been independently asked, and had no idea of what the other respondents had answered.

Lorenz et al. [9] contend, after performing some interesting experiments, that

> [a]lthough groups are initially "wise," knowledge about estimates of others narrows the diversity of opinions to such an extent that it undermines the wisdom of crowd effect in three different ways.
> - The "social influence effect" diminishes the diversity of the crowd without improvements of its collective error.
> - The "range reduction effect" moves the position of the truth to peripheral regions of the range of estimates so that the crowd becomes less reliable in providing expertise for external observers.
> - The "confidence effect" boosts individuals' confidence after convergence of their estimates despite lack of improved accuracy.

So, if we use (static) word embeddings in Portuguese, what is the crowd? The next tables show the size of some of the most common WEs (Table 1)[1], and how many words/entries they share with each other (Table 2).

Table 1 How many words have embeddings; removing words with numbers.

	Size in words	without numbers
nilc	929,606	910,215
nlx	873,910	752,001
pt-lkb	202,001	201,877
cc	2,000,000	1,665,247
base	1,052,405	984,226
lemas	1,613,937	1,374,196

Table 2 Removing words with numbers, how many words are shared.

	nilc	nlx	pt-lkb	cc	todosbase	todoslemas
nilc	–	296,157	75,285	380,252	**536,720**	158,813
nlx	296,157	–	58,716	**596,091**	231,931	304,249
pt-lkb	75,286	58,726	–	70,217	**75,311**	65,900
cc	380,252	**596,091**	70,217	–	365,048	456,284
base	**536,720**	233,795	75,301	281,097	–	314,314
lemas	158,813	304,249	65,900	**390,415**	281,097	–

In Tables 3 and 4 one can appreciate the 15 closest words to the Portuguese word *inteligência* given by some of these models, as an illustration of what they can do, and of what they may not do.

But the question remains: who should answer? Should one use the crowd, that is, a lot of different people who wrote different texts in different contexts and take the average/median? Or should texts about a particular subject be used when one is interested in that particular subject?

[1] See, respectively, [8, 14, 7, 4] for nilc, nlx, cc, and pt-lkb and [19] for base and lemas.

It has been argued by Tshitoyan et al. in an interesting letter to *Nature* [23], that

> models trained on the set of all Wikipedia articles (about ten times more text than our corpus) perform substantially worse on materials science analogies. Contrary to what might seem like the conventional machine learning mantra, throwing more data at the problem is not always the solution. Instead, the quality and domain-specificity of the corpus determine the utility of the embeddings for domain-specific tasks.

Here we are back to one of the most fundamental issues of (at least old) artificial intelligence: aren't we confusing language knowledge with world knowledge? Domain vs. general knowledge?

What do word embeddings encode? World knowledge, or language knowledge? Should they be used for expert tasks, or for general language? AI has always excelled in specific domains, domain expertise, and not in general-purpose reasoning or language.

If we look at the closest words to the Portuguese word *inteligência* given by different ways of employing the several WEs available, we see that they are widely different depending on the method, and also depending on the textual base. (The last line gives the range of similarities.)

Table 3 The closest words in models with 300 dimensions: *lemas* indicates that the word embeddings were created after lemmatizing the corpus, and *lemasmwe* after lemmatizing and connecting multiword expressions, both done by PALAVRAS [1].

base, w2v	lemas, glove	lemasmwe, fasttext	cc, fasttext
intuição	senso	ininteligência	inteligencia
imaginação	habilidade	contra-inteligência	Inteligência
sabedoria	mente	desinteligência	ainteligência
criatividade	criatividade	Inteligência	perspicácia
sagacidade	experiência	inteligência-do-cinema	inteligência.A
habilidade	talento	inteligibilidade	deinteligência
perspicácia	certo	inteligXncia	intelecto
astúcia	capaz	deligência	contra-inteligência
intelecto	imaginação	inteligencia	intelegência
destreza	excelência	inteligência	super-inteligência
0.68-0.57	0.85-0.76	0.95-0.85	0.77-0.62

While it becomes clear to the naked eye that fasttext finds the morphologically closest words, while glove and word2vec the semantically closest, and that the common crawl (cc) embeddings embed a lot of noise (misspellings and tokenization errors), there are all sorts of choices that provide different results. Further examples can be seen in Table 4.

In order to illustrate the same question now done to a specifialized set of texts, namely literary texts in Portuguese, I also present the results in Table 5.

But, of course, we are not going to evaluate a set of embeddings based on one word. There are two standard ways of evaluating (static) embeddings: intrinsic and extrinsic. The intrinsic approaches for Portuguese have been mainly the use of a set of analogies translated from English (the google set), and this is something I would like to fiercely criticize – assuming that word embeddings for Portuguese are conceived as a reflection of the language, and not primarily of the world.

In my opinion, it is wrong to use translated analogies in at least three counts:

Table 4 The closest words in further models with 300 dimensions.

nilc, w2v	nilc, glove	nilc, fasttext	nlx, w2v
inteligencia	habilidade	inteligênciaX	criatividade
inteligências	intuição	geointeligência	intuição
astúcia	criatividade	super-inteligência	imaginação
imint	força	contra-inteligência	sagacidade
criptológica	sabedoria	desinteligência	perspicácia
engenhosidade	senso	ciberinteligência	argúcia
laboriosidade	sensibilidade	foto-inteligência	sensibilidade
sagacidade	imaginação	contrainteligência	testreza
intuição	capacidade	superinteligência	lucidez
imaginação	talento	deligência	inventividade
0.68-0.54	0.54-0.44	0.96-0.88	0.71-0.66

Table 5 The closest words in further models with 300 dimensions, trained on 50 millions of literary text, mainly 19th century.

words, w2v	words, glove	words, fasttext	lemas, w2v	lemas, glove
ciência	ciência	ininteligência	espírito	espírito
compreensão	sensibilidade	desinteligência	intelectual	talento
capacidade	compreensão	Inteligência	talento	'pírito
penetração	espírito	intelligência	inteligente	entendimento
perspicácia	energia	deligência	'pírito	capacidade
espírito	humana	inteligências	ciência	conhecimento
instrução	imaginação	vigência	capacidade	ciência
intelectual	capacidade	desinteligências	compreensão	compreensão
sensibilidade	superior	consCiência	aptidão	prático
concepção	experiência	conciência	faculdade	bastante
0.57-0.47	0.59-0.48	0.97-0.83	0.62-0.50	0.60-0.46

First, because it from the start only caters for what is common in both languages. If there were an interesting feature of Portuguese one wanted to assess that was not present in English, it would never pop up from translation.

Second, because it gives unwarranted importance to things, words and concepts that are key in English but marginal in Portuguese. The most discussed example, that of queen and king, is striking in that respect: no doubt that this is a much more relevant concept in English than in Portuguese. And capitals of US states are certainly part and parcel of the average American's knowledge, but very far away from what one would expect a Portuguese-speaking man in the street to know.

Finally, even the structure of concepts so basic as family are different in the two languages, given that *pais* means both parents and fathers, and *irmãos* can be translated by siblings or brothers. The translation of analogies involving these lexical items will not work, which means that also translation-related phenomena, like errors or translationese, will impact the "Portuguese" analogies.

There is, fortunately, one other set of evaluation data, created from Portuguese lexical resources, TALES [5]. But I hope that the NLP community for Portuguese will join efforts and develop many more sets of Portuguese-inspired evaluation data to evaluate NLP of

Portuguese (not necessarily only or even specifically for word embeddings), because different languages (and cultures) embody different data, categories, and assumptions, and reducing everything to English would amount to an epistemicide of terrible proportions (for the concept of epistemicide, cf. the sociologist Boaventura Sousa Santos [15]).

6 Concluding remarks

To conclude, let me restate my most important messages here:

Values are an extremely important feature of any human language. Human language is fundamentally human centered, on values towards mankind.

Artificial intelligence as a way to replace human intelligence is doomed to failure, because of the absence of values. Hybrid intelligence, namely cooperation between humans and machines, is what we should aim for.

Different (human) languages and different cultures, embodying different values and ways of seeing the world, should not be dismissed as a nuisance. Rather, they are (also) a manifestation of what is human, and what is intelligence.

References

1 Eckhard Bick. PALAVRAS, a Constraint Grammar-based Parsing System for Portuguese. In Tony Berber Sardinha and Thelma de Lurdes São Bento Ferreira, editors, *Working with Portuguese Corpora*, pages 279–302. Bloomsbury Academic, 2014.

2 John M. Ellis. *Language, Thought and Logic.* Northwestern University Press, 1993.

3 Francis Galton. Vox populi. *Nature*, 75(7):450–451, 1907.

4 Hugo Gonçalo Oliveira. Learning Word Embeddings from Portuguese Lexical-Semantic Knowledge Base. In Aline Villavicencio, Viviane P. Moreira, Alberto Abad, Helena de Medeiros Caseli, Pablo Gamallo, Carlos Ramisch, Hugo Gonçalo Oliveira, and Gustavo Henrique Paetzold, editors, *Computational Processing of the Portuguese Language - 13th International Conference, PROPOR 2018, Canela, Brazil, September 24-26, 2018, Proceedings*, pages 265–271. Springer, Cham, 2018.

5 Hugo Gonçalo Oliveira, Tiago Sousa, and Ana Alves. TALES: Test set of Portuguese Lexical-Semantic Relations for Assessing Word Embeddings. In *Proceedings of the Workshop on Hybrid Intelligence for Natural Language Processing Tasks HI4NLP (co-located at ECAI-2020)*, pages 41–47, 2020.

6 Jang Graat. How Humans Succeed While Failing to Communicate, 2020. Abstract of keynote at SLATE'20.

7 Edouard Grave, Piotr Bojanowski, Prakhar Gupt, Armand Joulin, and Tomas Mikolov. Learning Word Vectors for 157 Languages. In *Proceedings of LREC 2018*, pages 3483–3487, 2018.

8 Nathan S. Hartmann, Erick R. Fonseca, Christopher D. Shulby, Marcos V. Treviso, Jéssica S. Rodrigues, and Sandra M. Aluísio. Portuguese Word Embeddings: Evaluating on Word Analogies and Natural Language Tasks. In *Proceedings of Symposium in Information and Human Language Technology, Uberlândia, MG, Brazil, October 2–5, 2017*, 2017.

9 Jan Lorenz, Heiko Rauhutb, Frank Schweitzera, and Dirk Helbing. How social influence can undermine the wisdom of crowd effect. *PNAS - Proceedings of the National Academy of Sciences of the United States of America*, 108(22), 2011. `doi:10.1073/pnas.1008636108`.

10 Rada Mihalcea. The Ups and Downs of Word Embeddings, 2021. Seminar 19 May 2021. URL: `https://www.youtube.com/watch?v=33XtLnPDOC0`.

11 Luke Oakden-Rayner, Jared Dunnmo, Gustavo Carneiro, and Christopher Ré. Hidden stratification causes clinically meaningful failures in machine learning for medical imaging. In *Proceedings ACM Conference on Health Inference Learning*, page 151–159, 2020. `doi:10.1145/3368555.3384468`.

12 Rosalind Picard. *Affective Computing*. MIT Press, 1997.

13 Kent Pitman. What Programming Language Design Taught Me About Life, 2018. Abstract of keynote at SLATE'18.

14 João Rodrigues, António Branco, Steven Neale, and João Silva. LX-DSemVectors: Distributional Semantics Models for the Portuguese Language. In João Silva, Ricardo Ribeiro, Paulo Quaresma, André Adami, and António Branco, editors, *Computational Processing of the Portuguese Language: 12th International Conference, PROPOR 2016, Tomar, Portugal, July 13-15, 2016, Proceedings*, pages 259–270. Springer, 2016.

15 Boaventura de Sousa Santos. *Epistemologies of the South. Justice against Epistemicide*. Paradigm Publishers, 2014.

16 Delfim Santos. Sobre o conceito de cultura. In *Obras Completas, vol. III (1953-1966)*, pages 543–550. Gulbenkian, 2011. 3.a edição, revista e ampliada. Organização e notas: Cristiana de Soveral.

17 Diana Santos. What is natural language? Differences compared to artificial languages, and consequences for natural language processing, 2006. Invited lecture, SBLP2006 and PROPOR'2006, Itatiaia, RJ, Brazil, 15 May 2006. URL: `https://www.linguateca.pt/Diana/download/SantosPalestraSBLPPropor2006.pdf`.

18 Diana Santos. Grandes quantidades de informação: um olhar crítico, 2021. Invited lecture, HD-Rio 2020/2021, 14 April 2021. URL: `https://www.youtube.com/watch?v=Qi-3QzP0NxM0`.

19 Diana Santos. Palavras pulverizadas: algumas experiências, 2021. URL: `https://www.linguateca.pt/Diana/download/valoresPalPulv.pdf`.

20 Diana Santos and Belinda Maia. Language, emotion, and the emotions: A computational introduction. *Language and Linguistics compass*, 12(5), 2018. `doi:10.1111/lnc3.12279`.

21 Peter Sedgwick. Illness – mental and otherwise. *Hastings Center Studies*, 1(3):19–40, 1973.

22 George Steiner. *After Babel: aspects of language and translation*. Oxford University Press, 1992. 1st edition 1975.

23 Vahe Tshitoyan, John Dagdelen, Leigh Weston, Alexander Dunn, Ziqin Ron, Olga Kononova, Kristin A. Persson, Gerbrand Ceder, and Anubhav Jain. Unsupervised word embeddings capture latent knowledge from materials science literature. *Nature*, 571:95—98, 2019.

24 Frank van Harmelen. Hybrid Intelligence: AI systems that collaborate with people, instead of replacing them, 2020. Keynote at IC3K 2020. URL: `http://www.ic3k.org/Documents/Previous_Invited_Speakers/2020/IC3K_2020_KS_3_Presentation.pdf`.

Derzis: A Path Aware Linked Data Crawler

André Fernandes dos Santos
CRACS & INESC Tec LA, Faculty of Sciences, University of Porto, Portugal

José Paulo Leal
CRACS & INESC Tec LA, Faculty of Sciences, University of Porto, Portugal

Abstract

Consuming Semantic Web data presents several challenges, from the number of datasets it is composed of, to the (very) large size of some of those datasets and the uncertain availability of querying endpoints. According to its core principles, accessing linked data can be done simply by dereferencing the IRIs of RDF resources. This is a light alternative both for clients and servers when compared to dataset dumps or SPARQL endpoints. The linked data interface does not support complex querying, but using it recursively may suffice to gather information about RDF resources, or to extract the relevant sub-graph which can then be processed and queried using other methods. We present Derzis[1], an open source semantic web crawler capable of traversing the linked data cloud starting from a set of seed resources. Derzis maintains information about the paths followed while crawling, which allows to define property path-based restrictions to the crawling frontier.

2012 ACM Subject Classification Information systems → Web crawling; Information systems → Structure and multilingual text search

Keywords and phrases Semantic web, linked open data, RDF, crawler

Digital Object Identifier 10.4230/OASIcs.SLATE.2021.2

Supplementary Material *Software (Source Code)*: `https://github.com/andrefs/derzis`
archived at `swh:1:dir:d35f9b43c9b88955c9d6068fcc15e59c4aba816f`

Funding This work is financed by National Funds through the Portuguese funding agency, FCT – Fundação para a Ciência e a Tecnologia, within project UIDB/50014/2020.
André Fernandes dos Santos: Ph. D. Grant SFRH/BD/129225/2017 from Fundação para a Ciência e Tecnologia (FCT), Portugal.

1 Introduction

Two features of the Semantic Web (SW) which make it interesting – being distributed and not needing a globally defined schema – also make accessing it not trivial. Two main questions arise from these features: *how* to access the SW and *what* to access.

The SW is built on top of general purpose technologies and standards such as HTTP and Internationalized Resource Identifiers (IRIs), and a small core of additional specifications, such as RDF(S), OWL and SPARQL. This means that the basic protocols and syntaxes are universal. However, there are several possible approaches when making semantic data available (and, consequently, when consuming semantic data). These usually fall into one of the following categories: (i) downloadable triplesets, (ii) queriable SPARQL endpoints or other SW interfaces, (iii) custom APIs which return RDF data, and (iv) dereferenceable resource IRIs.

The same data is frequently available using multiple methods. The requirements for a Semantic Web application (SWA) will heavily influence which method should be used. Conversely, the methods under which data is available strongly dictate the types of applications which can consume it. A detailed explanation will be provided in the next section.

[1] Available at `https://github.com/andrefs/derzis`.

© André Fernandes dos Santos and José Paulo Leal;
licensed under Creative Commons License CC-BY 4.0

Dereferencing IRIs, also known as the linked data interface, or the *follow-your-nose* approach, can be used in applications which do not require complex querying support, but simply need to fetch information about a given set of resources. It allows fetching data which is up to date and does not require the resources needed to download and process large tripleset dumps. The level of detail can be increased or decreased by adjusting the depth of recursion, or by fine tuning which semantic links should be followed and which IRIs should be dereferenced. It can also be used as as preliminary step on a more complex pipeline, reducing a possibly very large semantic graph (composed of triples originating for many connected triplesets) to a sub-graph containing only the relevant data. This data can then be more easily manipulated and queried.

In this paper we present Derzis, an open source semantic web crawler which, given a set of seed resources, traverses the linked data cloud, following the semantic links extracted from the triples resulting from dereferencing IRIs. Parameters such as maximum depth and property path restrictions can be defined to limit the graph traversal.

This paper is structured as follows. In Section 2 we cover concepts relevant to this work. Section 3 describes and compares other semantic web crawlers and tools following a similar methodology. Our approach is detailed in Section 4, where we provide an example of the crawling algorithm and its formal definition. Section 5 describes the system architecture. Section 6 describes preliminary results obtained using Derzis. Sections 7 and 8 describe, respectively, future developments planned for Derzis and conclusions for this work.

2 Background

The origins of the Semantic Web as a research field can be traced back to the early 2000s, with specifications such as RDF [4] and RDFS [17] being published a little earlier. In a 2001 article in Scientific American [3], Tim Berners-Lee et al. envisioned a web of semantically annotated data, intelligible to machines, in which automated agents could roam autonomously to perform tasks. Despite undeniable contributions to the fields of data sharing, discovery, integration and reuse, the semantic web as a research field has been the subject of much criticism, as compiled by Hogan [13]. The original vision is still far from fulfilled.

The history of this field can be split into three phases [12]: the ontology phase, the linked data phase and the knowledge graphs phase. In the beginning the focus was on developing formal ontologies with the goal of sharing and integrating data. Then the interest shifted to the publication of datasets linked through the reuse of IRIs. More recently, we have seen the industry become interested in semantic technologies, which led to the development of their own knowledge graphs, e.g. Google [22], Microsoft [9], Amazon [16]. In time, more open versions became available, e.g. Wikidata [24] and DBpedia [18]. Nowadays, the semantic web is composed of artifacts inherited from these three phases: domain-specific and general scope ontologies, more shallow vocabularies, linked datasets and knowledge graphs.

Semantic Web Applications can access the semantic web in a number of ways. Choosing one method over another may severely limit what your application can do or how it operates. The same is to say that different applications will have different needs in what regards to consuming semantic data.

One way is to download the knowledge graphs in RDF format in advance. This requires the computational resources to deal with very large files, and the possibility of data becoming eventually outdated. Applications relying on queriable endpoints (usually SPARQL or Triple Pattern Fragments [23]) avoid these shortcomings. However, they become dependent on servers which must be available and capable of responding to potentially complex queries.

Integrating information from different sources may be difficult. Custom APIs are similar, but present one additional disadvantage: they require applications to be even more tightly coupled with them, as the SWA must have built-in the knowledge of how to interact with that non-standard endpoint.

The methods previously described are all better suited for applications where the triplesets to be consumed are known beforehand, either because the triplesets need to be downloaded, their queriable endpoints must be known and integrated with each other, or because the applications need to understand the syntax of custom APIs.

The linked data interface (also known as the *follow-your-nose* approach [25]) allows gathering data about a resource simply by dereferencing (i.e. performing an HTTP request to) its IRI, and parsing the resulting RDF content. This behavior is explicitly stated in the Linked Data principles [2]. This method is light for the servers, as there is no need of software to parse and perform complex queries, and can even be statically served. It is also light for the users as it prevents the need to deal with large files or outdated data, and can reasonably be expected to work more or less uniformly across domains.

A web crawler is a program capable of accessing HTML pages through their URIs, parsing them, extracting hyperlinks and recursively repeating the process [19]. These are typically used by search engines to index web page contents and calculate their relevance. To avoid having crawlers accessing unwanted parts of their sites, domains can indicate which URLs are off-limits by adding directives to their `robots.txt` file according to the Robots Exclusion Protocol [14]. The `Crawl-Delay` directive is commonly used to rate-limit crawlers performing multiple requests. Common crawling policies further specify which pages should be visited, when they should be re-visited, the frequency of the requests and the degree of parallelization of the crawler [5].

An RDF resource is identified by an IRI. An RDF triple specifies a semantic link between two RDF resources (the subject and object of the triple) using another RDF resource (the predicate). Multiple RDF triples make an RDF graph, where the nodes are the subjects and objects of the triples, and the edges are the predicates[2]. According to the Linked Data principles, accessing the IRI of an RDF resource should return an RDF representation of that resource – i.e. triples relevant to that resource's understanding. This usually means triples where the resource appears either as the subject or the object.

A semantic web crawler (SWC) is the application to the semantic web of the web crawler concept. It starts with the IRIs of RDF resources, and *dereferences* them – i.e. fetches the IRIs contents by performing HTTP requests. Instead of a simple HTML page, for each resource it expects in return an RDF document (which can also be an HTML page with RDFa metadata). This document contains triples representing semantic links between the resource and other nodes in the RDF graph. The SWC parses the RDF document and repeats the process for the new resources found. The crawling process can be limited in a number of ways, most commonly by defining which resources should be dereferenced and/or the maximum crawl depth.

[2] Actually, a resource appearing in the predicate position of a triple can be (and frequently is) used as the subject or object of another. This means that the edges of an RDF graph can simultaneously also be nodes.

3 Related Work

There are several semantic web crawlers described in the literature, some of which are publicly available. Their core functionality is similar: following links in the semantic web. However, they differ in more specific features, such as sorting what should be crawled first (e.g. depth-first or breadth-first), white or black-listing resources to be dereferenced, understanding basic RDFS and OWL properties (also referred to as RDFS-Plus [7]) or using domain knowledge to inform the crawl process. Next we present a brief description of notable examples of SWCs. A comparison of their features can be found in Table 1.

Crawler-LD is a semantic web crawler designed to help linked data publishers finding vocabulary terms which can be interlinked with their own data [20]. Given an initial set T of terms, for each Crawler-LD searches the DataHub catalogue of Linked Data triplesets searching terms which are directly or transitively related to it either by sub-classing or using properties such as `owl:sameAs` or `owl:equivalentClass`. Freire and Silva (2020) describe a crawler which is initialized with knowledge of the domain in which it is used [10]. This knowledge is then used to decide which resources and triples to follow and which ones to ignore. Similarly, Bedi et al. (2012) describe a semantic focused crawler which is also enriched with domain knowledge [1]. In this case, semantic distance metrics between the crawled resources and the ones in the domain knowledge are used to prioritize the crawl list. LDSpider is a generic crawler for linked data [15] which is able to handle a large number of formats, perform breadth-first or depth-first crawls, include schema information and limit the set of properties which are followed, or the prefix of resources crawled, or their maximum depth. In KRaider [6], the maximum resource depth cut-off ignores the properties `owl:sameAs`, `owl:equivalentClass` and `owl:equivalentProperty`. Squirrel [21] is a distributed crawler capable of handling RDF formats, compressed files, HTML with RDFa or Microformats, SPARQL endpoints and CKAN addresses. Crawl priority can be defined based on the IP or domain of the resources.

Hartig and Pirrò have proposed a formal foundation to extend to SPARQL that leverages property paths and allows coupling navigation at the data level with navigation on the Web graph [11]. S-Paths [8] is a linked data browser which supports different representations for data. It relies on path characteristics to aggregate data and determine the best view for the results.

Our solution, Derzis, is capable of reducing the size of the graph being crawled without requiring previous domain knowledge. This is achieved by limiting the number of distinct properties in each crawl path. As a result, Derzis focuses on the triples and path expected to contain more relevant information regarding the seed resources.

4 Approach

The goal of this work was the development of a semantic web crawler capable of recursively extracting information from a set of semantic resources. This SWC should allow reducing a large graph to a smaller sub-graph containing the most relevant data regarding the seed resources. The main use case for this is the implementation of graph-based semantic measures. Due to their algorithmic complexity, such measures are impractical to implement in large knowledge graphs, and as such would benefit from the ability of reducing said graphs to more manageable sizes. The base algorithm of this crawler should prove useful in other applications in which there is the need to focus on the relevant parts of a larger graph, e.g. a semantic browser capable of inferring which information should be displayed.

Table 1 Comparison of semantic web crawlers features.

Crawler	Publicly available	Distributed / parallel	Crawl priority	`robots.txt` compliant	RDFS-Plus	Domain knowledge	Crawl limits
Crawler-LD	✘				✔		
Freire and Silva	✘					✔	
Bedi et al.	✘		c			✔	
LDSpider	✔	✔	e	✔	✘	✔	134
KRaider	✘	✔	d		✔	✔	1
Squirrel	✔	✔	ab	✔	✘	✘	12

Crawl priority:
(a) IP
(b) Pay-level domain
(c) Domain knowledge
(d) FIFO
(e) Breadth or depth-first

Crawl limits:
(1) Maximum depth
(2) Black and white lists
(3) Properties followed
(4) Resources prefix

This crawler was developed with the following design goals:

1. recursively dereference RDF graph nodes (subjects and objects),
2. crawl limits based on property path restrictions,
3. parallel and distributed crawling,
4. compliance with `robots.txt` rules.

Next we describe the strategies used to reduce the crawled graph size. Then we present an example of the crawling process with a detailed step-by-step description of the sequence of events that take place. After that we provide a formal definition of the crawling algorithm.

4.1 Reducing the crawled graph

To ensure that priority is given to the relevant parts of the original graph, several strategies are used, which are described next.

Triples where a resource appears as predicate are discarded. Imagine an RDF property `ex:hasColor` which allows specifying the color of a resource, e.g. `ex:Sky ex:hasColor ex:Blue`. If `ex:hasColor` was passed to Derzis as a seed resource, it should find relevant triples which characterize it such as `ex:hasColor rdf:type rdf:Property`, `ex:hasColor rdf:subPropertyOf ex:hasPhysicalProperty` and `ex:colorOf owl:inverseOf ex:hasColor`. These triples are *describing* the property. However, triples such as `ex:Snow ex:hasColor ex:White` or `ex:Grass ex:hasColor ex:Green` are not describing `ex:hasColor`, they are *using* it to describe the other resources. For this reason, when dereferencing a resource, triples where it is used as the predicate are discarded.

Crawling follows only subject and object resources. Imagine an RDF resource `ex:Einstein`. Suppose that, when dereferencing it, we obtain the triples `ex:Einsten ex:wonAward ex:NobelPrize` and `ex:Einstein ex:bornIn ex:Ulm`. Recursively dereferencing the properties `ex:wonAward` and `ex:bornIn` would gather data about these properties. This is arguably less relevant for describing the resource `ex:Einstein` than dereferencing the resources it is related to, in this case, `ex:NobelPrize` and `ex:Ulm`.

Crawling can be limited to paths containing a maximum number of distinct properties. Homogeneous paths between resources are more likely to represent implicit hierarchy or a meaningful chain of connections than heterogeneous ones.

Maximum crawling depth can be defined. Resources too far away from the seed resources are less relevant than those which are closest.

4.2 Crawling example

The IRI `http://dbpedia.org/resource/Mozzarella` is the identifier for Mozzarella cheese on DBpedia. Next we describe step by step an example of the crawling process of Derzis, using this IRI as the only seed. We will limit the crawling to paths with a maximum depth (`maxPathDepth`) of 4 and no more than 2 distinct properties (`maxPathProps`). Figure 1 displays the graph being built by the crawling process. Table 2 presents the triplesets found when dereferencing each resource IRI. For clarity purposes, the triplesets have been truncated and displayed in Turtle format. Additionally, we omit several operations related with job distribution, database housekeeping and request rate-limiting. We also implicitly assume the following prefix definitions:

```
@prefix dbr: <http://dbpedia.org/resource/>
@prefix dbp: <http://dbpedia.org/property/>
@prefix dbo: <http://dbpedia.org/ontology/>
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
@prefix owl: <http://www.w3.org/2002/07/owl#>
```

1. The seed IRI `http://dbpedia.org/resource/Mozzarella` is added to Derzis as a seed Resource. This triggers the creation of a Path, containing one single node (`dbr:Mozzarell`), and no predicates.
2. The seed resource is then dereferenced. All the triples in which `dbr:Mozzarella` is either the subject or object are added to Derzis. The triple `dbr:Calzone dbo:ingredient dbr:Mozzarella` will lead to the extension of the path. This new path will have `dbr:Mozzarella` as its `seed`, `dbr:Calzone` as its `head`, no other nodes and `dbo:ingredient` as the only `predicate`.
3. The process continues with the dereferentiation of `dbr:Calzone`. The resulting triples will give origin to two new paths. One starts in `dbr:Mozzarella` and reaches `dbr:Salami`, with a total length of 3 nodes and a single predicate (`dbo:ingredient`). Another starts also in `dbr:Mozzarella` but leads to `dbr:Pizza`, also with a length of 3 but containing 2 predicates already: `dbo:ingredient` and `dbo:hasVariant`.
4. Dereferencing `dbr:Salami` will extend the previous path leading here, adding one more predicate (`rdf:type`) and one more node (`owl:Thing`). This new path has reached the maximum number of nodes and distinct predicates. It is marked as finished and, consequently, `owl:Thing` is not dereferenced.
5. The path leading to `dbr:Pizza` is already maxed out on the number of predicates. This means that it can only be extended with triples in which the predicate is either `dbo:ingredient` or `dbo:hasVariant`. From the triples obtained from dereferencing `dbr:Pizza`, `dbr:Pizza dbp:country dbr:Italy` and `dbr:Neapolitan_pizza dbp:type dbr:Pizza` are ignored. It is extended to `dbr:Tomato_sauce` and `dbr:Panzerotti`. Both paths have now also reached the maximum length. They are marked as finished, and these resources are not dereferenced.
6. At this point, there are no more active paths. The crawling process finishes and the crawled graph can be exported.

4.3 Crawling algorithm

Given a set R of seed RDF resources, their linked data graph G_R is the graph obtained by collecting all the triples obtained by dereferencing each resource in R, and recursively applying the same to each resource included in those triples. This potentially results on a

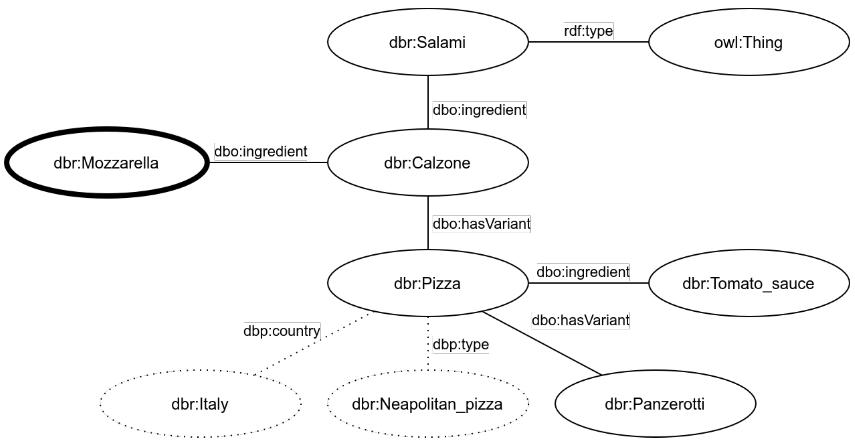


Figure 1 Example of the graph built while crawling.

Table 2 Contents of resources IRIs.

i) `http://dbpedia.org/resource/Mozzarella`	ii) `http://dbpedia.org/resource/Calzone`

`dbr:Calzone dbo:ingredient dbr:Mozzarella .`
`[...]`
	`dbr:Pizza dbo:hasVariant dbr:Calzone .`
	`dbr:Calzone dbo:ingredient dbr:Salami .`
	`[...]`

iii) `http://dbpedia.org/resource/Salami`	**iv) `http://dbpedia.org/resource/Pizza`**

`dbr:Salami rdf:type owl:Thing .`
`[...]`
	`dbr:Pizza dbo:ingredient dbr:Tomato_sauce .`
	`dbr:Pizza dbo:hasVariant dbr:Panzerotti .`
	`dbr:Pizza dbp:country dbr:Italy .`
	`dbr:Neapolitan_pizza dbp:type dbr:Pizza .`
	`[...]`

very large graph (maybe even the whole Linked Data cloud, given its connected nature). We can ignore the properties characterization and classification by dereferencing only the nodes of the graph (subjects and objects of triples). The size of the resulting graph can be further reduced by considering a set of path restrictions PR which determine, for each new node discovered, whether it should be followed (dereferenced) or not. Then $RG_{R,PR}$ is the sub-graph of G_R obtained by traversing it while applying the path restrictions PR. Figure 2 presents a simplified view of the semantic web, with a pair of seed resources (A and B), the linked data graph for A and B (i), the graph reduced by path restrictions (ii), and a disconnected part of the semantic web, unreachable by recursively following links starting from A and B (iii).

Path restrictions require Derzis to maintain information regarding each path being followed during the crawling process: its origin (seed resource), current head (latest resource appended), the total length and total number of distinct properties.

$RG_{R,PR}$ is built iteratively. It is initialized with a set of paths P containing a path for each seed resource. Algorithm 1 presents a simplified pseudocode version of Derzis main behavior (the actual code has small differences mostly regarding optimization of the distributed work). Initially, a path is created for each seed resource. Then, in each iteration, each resource p_{head} which is the head of an active path is dereferenced and marked as *visited*. For each resulting RDF triple, the property and node (either the subject or object, depending on the inverse position occupied by p_{head}) are added to p to create a new path p'. If the path restrictions ($maxPathLength$ and $maxPathProps$) are not met, p' is dropped. Is there is

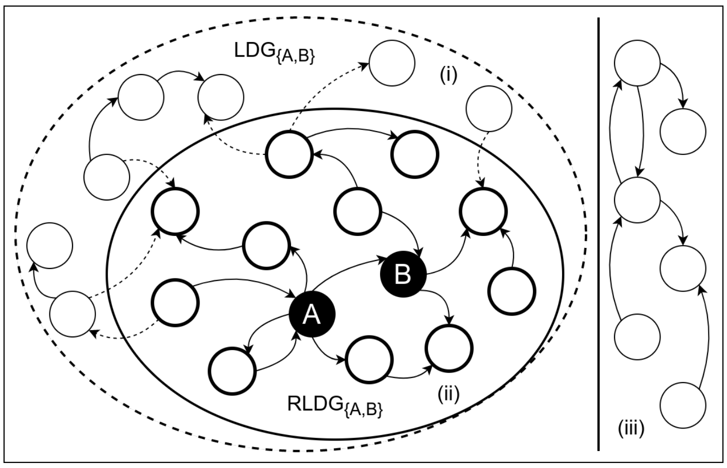


Figure 2 The semantic web and restricted linked data graph of resources A and B.

already a similar path (built with the same properties (or a subset), and with the same or shorter length), p' is dropped. If the resource at the head of the new path p' has already been visited by Derzis, the path is extended using cached information until an unvisited head is reached. The process ends when no more active paths exist. At this point, Derzis can export the restricted graph in RDF format.

Algorithm 1 BuildRG(P).

```
global $maxPathLength, maxPathProps$
global $R_{visited}$

$P' \leftarrow \varnothing$
$R \leftarrow \{p_{head} : p \in P \land p \text{ is active}\}$
if $R = \varnothing$
  then $return$

for each $p_{head} \in R$
do
  $triples \leftarrow dereference(p_{head})$
  $R_{visited} \leftarrow R_{visited} \cup p_{head}$
  for each $node, prop : \{node, prop, p_{head}\} \in triples \lor \{p_{head}, prop, node\} \in triples$
  do
    if $p_{length} \geq maxPathLength \lor prop \notin p \land |p_{props}| \geq maxPathProps)$
      then $break$
    $p' \leftarrow p \cup \{prop, node\}$
    if $\exists q : q_{seed} = p'_{seed} \land q : q_{head} = p'_{head} \land q_{props} \subseteq p'_{props}$
      then $break$
    if $p'_{head} \in R_{visited}$
      then AddExistingHead($p'$)
    $P' \leftarrow P' \cup p'$
    $p \leftarrow finished$
BuildRG($P'$)
```

5 System architecture

The Derzis tool is built around two main components: the Manager and a pool of Workers. Figure 3 provides a visual representation of Derzis architecture. The Manager is responsible for accessing the database (Figure 3.5), decides what should be crawled next and distributes jobs to the workers (Figure 3.2). Workers handle the tasks of retrieving domain `robots.txt` files and resource IRI contents (Figure 3.3). Both components were written in Node.js; they coordinate with each other exchanging messages over Redis, and the data is persisted on a MongoDB instance. The manager, each of the workers, the database and Redis can all be distributed across different machines.

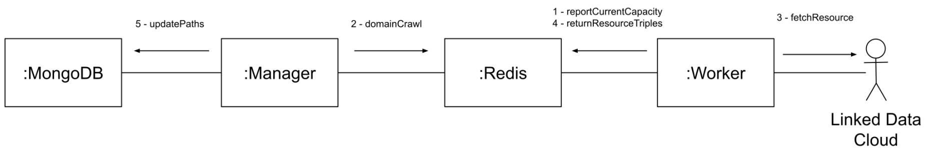


Figure 3 UML Communication diagram of the system architecture.

5.1 Worker

A Worker is an autonomous process which communicates with the Manager over Redis. When it starts, it sends to a configurable channel a message stating the type and number of jobs it is capable of handling (Figure 3.1). This is repeated periodically. Currently these jobs can be either trying to retrieve the `robots.txt` of a domain or dereferencing unvisited resources of a domain. By default, a Worker accepts a maximum of 10 simultaneous `robots.txt` checks and 10 domains with batches of 10 resources each to be dereferenced.

The Worker handles these jobs asynchronously. For each domain, it dereferences each resource sequentially (Figure 3.3), first checking whether the `robots.txt` allows it and what the rate limit is. In the HTTP request that fetches the resource, the Accept header is set to allow several RDF mime types; frequently, however, servers do not comply. In these cases, the Worker checks the `Link` response header in search of alternative URLs (or, if the returned content is HTML, the `<link>` tags). If the Content-Type is still unrecognised an error is raised. The results of each `robots.txt` check and resource dereference are sent back to the Manager also via Redis (Figure 3.4).

5.2 Manager

The Manager starts by posting a message in Redis asking for any Workers listening to report their availability to perform jobs. Anytime it receives an availability message, it retrieves from the database the appropriate number of domains needing their `robots.txt` to be retrieved or resources to be dereferenced. When these jobs are sent to the worker, the Manager marks them with the Worker identifier, making sure that a domain is only worked on by one Worker at a time (preventing multiple concurrent requests to the same domain), and starts a timer for that job. If the Worker does not reply before the timer expires, the domain is reset and made available for other workers.

6 Preliminary results

Derzis is still in early stages of development. For the time being, we do not yet have comparisons of its performance against similar tools or using public benchmarks. Nevertheless, we have already results gathered by running it using a small set of seed resources (5 types of cheese gathered from DBPedia) and combinations of crawling parameters. These were obtained on a laptop with an Intel i7-6500U CPU 2.50GHz and 8GB of memory. The system was configured to spawn 3 workers, each capable of handling 10 `robots.txt` checks and 10 domains with 10 resources each to be dereferenced. Tables 3, 4 and 5 present these results.

Table 3 Total elapsed times for different combinations of *maxPathLength* and *maxPathProps*.

Total elapsed time		Maximum path length	
		2	3
Maximum distinct properties	1	44s	2h32m
	2	–	3h12m

Table 4 Total dereferenced resources for different combinations of *maxPathLength* and *maxPath-Props*.

Total dereferenced resources		Maximum path length	
		2	3
Maximum distinct properties	1	5	860
	2	–	1135

Table 5 Total triples obtained for different combinations of *maxPathLength* and *maxPathProps*.

Total triples		Maximum path length	
		2	3
Maximum distinct properties	1	1.7k	510k
	2	–	744k

The time required to run the crawler in each of these setups has been mostly dictated by DBPedia's `Crawl-Delay` directive, which imposes a 10 second delay between HTTP requests. Around 70% of triples in DBpedia connect resources using the property `dbo:wikiPageWikiLink`, which states that the Wikipedia page of the subject resource links to the Wikipedia page of the object resource. The property `dbo:wikiPageUsesTemplate`, which indicates that a Wikipedia pages has a infobox using a specific template, appears in 13% of the triples. More meaningful properties follow: `dbo:birthPlace` (4%), `dbo:deathPlace` (4%), `rdfs:type` (3%) and `owl:sameAs` (3%). By comparison, in Wikidata, the most frequent property is `http://schema.org/about` (40%) followed by `https://www.wikidata.org/wiki/Property:P31` (a Wikidata custom property similar to `rdf:type`, 3%).

7 Future work

We have planned several additional features for Derzis. Better error handling and other types of crawl limits are some examples. The indexes and queries performed to the MongoDB instance can also be improve. Given the small number of publicly available semantic web crawlers, and the distinct features presented by Derzis, we intend to make it easier to use by other people. Derzis is already open-source. However, usability and documentation needs to be improved. Real-time visualization tools to have an overview of the crawling process would also improve the overall experience.

Another focus of improvement is testing and evaluation. Derzis is currently lacking unit and integration tests. Additionally, real world experiments we have made with it so far are still only a proof of concept. We plan on evaluating Derzis using available benchmarks for SWCs and to perform our own comparisons against other SWCs.

Finally, Derzis was developed with the main motivation of being used in the broader context of implementing graph-based semantic measures. It addresses a need we have of being able to reduce semantic graph sizes keeping only the data relevant to a set of resources. We will continue to develop the larger system and evaluate Derzis in the context of this specific task.

8 Conclusions

The semantic web is a field of research with 20 years of existence. The early promise of a web understood by machines has not yet materialized. The large size of current knowledge graphs, the unreliable availability of queriable endpoints and the lack of universal vocabularies all contribute to this problem.

A semantic web crawler is capable of gathering semantic data using the linked data interface, a method of accessing the semantic web which is lighter for the clients and the servers. There have been several SWCs developed, but only a few are both publicly available and actively maintained. These tend to have in common a core set of features, such as limiting the depth of crawling, distributed architectures and complying with the Robots Exclusion Protocol. Other features such as requiring previous domain-knowledge, other types of crawl limits, customizable crawl priority or understanding RDFS and OWL basic constructs are present in specific crawlers.

Derzis is open source and available at `https://github.com/andrefs/derzis`. It presents as a distinctive feature the ability to reduce the crawled graph size. By keeping paths information during the crawling process, Derzis is able to limit the crawling to paths with a maximum number of distinct properties. This means that it can focus on more homogeneous paths, which provide more relevant information than heterogeneous ones. Additionally, Derzis does not dereference predicate resources, which usually provide information more relevant to the predicate than to the other elements of the triples.

References

1 Punam Bedi, Anjali Thukral, Hema Banati, Abhishek Behl, and Varun Mendiratta. A multi-threaded semantic focused crawler. *Journal of Computer Science and Technology*, 27(6):1233–1242, 2012.

2 Tim Berners-Lee. Linked Data – Design Issues, 2006. URL: `http://www.w3.org/DesignIssues/LinkedData.html`.

3 Tim Berners-Lee, James Hendler, and Ora Lassila. The Semantic Web. *Scientific american*, 284(5):34–43, 2001.

4 Dan Brickley, Ramanathan V Guha, and Andrew Layman. Resource Description Framework (RDF) Schemas, 1998. URL: `https://www.w3.org/TR/1998/WD-rdf-schema-19980409/`.

5 Carlos Castillo. Effective web crawling. *SIGIR Forum*, 39(1):55–56, 2005.

6 Giuseppe Cota, Fabrizio Riguzzi, Riccardo Zese, Evelina Lamma, et al. KRaider: a Crawler for Linked Data. In *34th Italian Conference on Computational Logic*, volume 2396, pages 202–216. CEUR-WS. org, 2019.

7 Fabien Gandon Dean Allemang, Jim Hendler. RDFS-Plus. In *Semantic Web for the Working Ontologist: Effective Modeling for Linked Data, RDFS, and OWL. Chapter 7*, 2020. doi:10.1145/3382097.3382107.

8 Marie Destandau, Caroline Appert, and Emmanuel Pietriga. S-Paths: Set-based visual exploration of linked data driven by semantic paths. *Semantic Web*, 12(1):99–116, 2020. doi:10.3233/SW-200383.

9 Michael Färber. The Microsoft Academic Knowledge Graph: A Linked Data Source with 8 Billion Triples of Scholarly Data. In *Proceedings of the 18th International Semantic Web Conference*, ISWC'19, pages 113–129, 2019. doi:10.1007/978-3-030-30796-7_8.

10 Nuno Freire and Mário J Silva. Domain-Focused Linked Data Crawling Driven by a Semantically Defined Frontier. In *International Conference on Asian Digital Libraries*, pages 340–348. Springer, 2020.

11 Olaf Hartig and Giuseppe Pirrò. A context-based semantics for SPARQL property paths over the web. In *European semantic web conference*, pages 71–87. Springer, 2015.

12 Pascal Hitzler. A review of the semantic web field. *Communications of the ACM*, 64(2):76–83, 2021.

13 Aidan Hogan. The Semantic Web: Two decades on. *Semantic Web*, 11(1):169–185, 2020. doi:10.3233/SW-190387.

14 Gary Illyes, Henner Zeller, Lizzi Harvey, and Martijn Koster. Robots Exclusion Protocol. URL: https://tools.ietf.org/html/draft-koster-rep-04#section-2.5.

15 Robert Isele, Jürgen Umbrich, Christian Bizer, and Andreas Harth. LDspider: An open-source crawling framework for the Web of Linked Data. In *Proceedings of the 2010 International Conference on Posters & Demonstrations Track*, volume 658, pages 29–32. Citeseer, 2010.

16 Arun Krishnan. Making search easier, 2018. URL: https://www.aboutamazon.com/news/innovation-at-amazon/making-search-easier.

17 Ora Lassila and Ralph R. Swick. Resource Description Framework (RDF) Model and Syntax specification, 1998.

18 Jens Lehmann, Robert Isele, Max Jakob, Anja Jentzsch, Dimitris Kontokostas, Pablo N Mendes, Sebastian Hellmann, Mohamed Morsey, Patrick Van Kleef, Sören Auer, et al. Dbpedia–a large-scale, multilingual knowledge base extracted from wikipedia. *Semantic web*, 6(2):167–195, 2015.

19 Marc Najork. Web crawler architecture, 2009.

20 A Gomes Raphael do Vale, Marco A Casanova, Giseli Rabello Lopes, and Luiz André P Paes Leme. CRAWLER-LD: a multilevel metadata focused crawler framework for linked data. In *International Conference on Enterprise Information Systems*, pages 302–319. Springer, 2014.

21 Michael Röder, Geraldo de Souza Jr, and Axel-Cyrille Ngonga Ngomo. Squirrel–Crawling RDF Knowledge Graphs on the Web. In *International Semantic Web Conference*, pages 34–47. Springer, 2020.

22 Amit Singhal. Introducing the knowledge graph: things, not strings. *Official google blog*, 5:16, 2012.

23 Ruben Verborgh, Miel Vander Sande, Olaf Hartig, Joachim Van Herwegen, Laurens De Vocht, Ben De Meester, Gerald Haesendonck, and Pieter Colpaert. Triple Pattern Fragments: a low-cost knowledge graph interface for the Web. *Journal of Web Semantics*, 37:184–206, 2016.

24 Denny Vrandečić and Markus Krötzsch. Wikidata: a free collaborative knowledgebase. *Communications of the ACM*, 57(10):78–85, 2014.

25 Liyang Yu. Follow your nose: a basic semantic web agent. In *A Developer's Guide to the Semantic Web*, pages 533–557. Springer, 2011.

Major Minors – Ontological Representation of Minorities by Newspapers

Paulo Jorge Pereira Martins
University of Minho, Braga, Portugal

Leandro José Abreu Dias Costa
University of Minho, Braga, Portugal

José Carlos Ramalho
Department of Informatics, University of Minho, Braga, Portugal

Abstract

The stigma associated with certain minorities has changed throughout the years, yet there's no central data repository that enables a concrete tracking of this representation. Published articles on renowned newspapers are a way of determining the public perception on this subject, mainly digital newspapers, being it through the media representation (text and photo illustrations) or user comments. The present paper seeks to showcase a project that attempts to fulfill that shortage of data by providing a repository in the form of an ontology: RDF triplestores composing a semantic database (W3C standards for Semantic Web). This open-source project aims to be a research tool for mapping and studying the representation of minority groups in a Portuguese journalistic context over the course of two decades.

2012 ACM Subject Classification Information systems → Web Ontology Language (OWL); Information systems → Ontologies; Computing methodologies → Ontology engineering; Information systems → Graph-based database models

Keywords and phrases RDF, OWL, Ontologies, Knowledge Representation, Minorities

Digital Object Identifier 10.4230/OASIcs.SLATE.2021.3

Supplementary Material
Software (Source Code): `https://github.com/Paulo-Jorge-PM/crawler-majorminors`
archived at `swh:1:dir:595c4e87b736bf4c8b3ae59d50f79e4c921cc4c8`
Dataset: `https://github.com/Paulo-Jorge-PM/datasets-majorminors`
archived at `swh:1:dir:053b6e2c5e2cd710fcb8e769db46479d836965aa`
Software (Source Code): `https://github.com/Paulo-Jorge-PM/ontology-assembler-majorminors`; archived at `swh:1:dir:49e46f6b20ac99a03db72f9121ceb2760606c8c7`
Software (Source Code): `https://github.com/leandrocosta16/scraper-MajorMinors`
archived at `swh:1:dir:7396d982c983ecdd8d8e1c2ffd169fba37bd9ad4`

1 Introduction

Tim Berners-Lee, director of the World Wide Web Consortium, established the term Semantic Web and promotes the concept of converting the Web into a big collection of semantic databases: "People keep asking what Web 3.0 is. I think maybe when you've got an overlay of scalable vector graphics – everything rippling and folding and looking misty-on Web 2.0 and access to a semantic Web integrated across a huge space of data, you'll have access to an unbelievable data resource" [9].

"Web 1.0 started as a Read only medium; the next version Web 2.0 established itself as a Read/Write medium. Now the currently evolving version of web, Web 3.0, is said to be a technologically advanced medium which allows the users to Read/Write/Execute and also allows the machines to carry out some of the thinking, so far, expected only from the humans" [8], introducing the concept of Semantic Web and Linked Open Data.

© Paulo Jorge Pereira Martins, Leandro José Abreu Dias Costa, and José Carlos Ramalho;
licensed under Creative Commons License CC-BY 4.0
10th Symposium on Languages, Applications and Technologies (SLATE 2021).
Editors: Ricardo Queirós, Mário Pinto, Alberto Simões, Filipe Portela, and Maria João Pereira; Article No. 3; pp. 3:1–3:13
OpenAccess Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Ontologies, the backbone of the Semantic Web 3.0, which contain the vocabulary, semantic relationships, and simple rules of inference and logic for a specific domain, are accessed by software agents. These agents locate and combine data from many sources to deliver meaningful information to the user [5].

This project explores this concept, trying to contribute to the Semantic Web initiative, by building a web of relational semantic data related to a subject, instead of a traditional web of documents. Our subject of study is the media representation of minorities in Portuguese newspapers. For this study 8 minority groups were focused (based on the research fields of the teams associated with this project): “refugees”, “women”, “Africans”, “Asians”, “homosexuals”, “migrants”, “gypsies” and “animals”. By “minorities” we refer to macro groups with some kind of social stigma and/or struggle for equality/rights, in a broad sense. In the future we would like to add more specific minority groups according to the scientific community feedback.

Figure 1 Minorities extracted.

Survey data is a common way of studying the public perception of minorities, but “they are only conducted sporadically and they seldom use a comparable set of questions, limiting their usefulness for comparing across time, space or minority group” [2]. With this project, we propose to use ontologies built with the content of digital newspapers, from 1996 to 2019, to analyze the evolution of the representation of identity groups. Ontologies can be a powerful research tool because not only they store a corpus, but they can also be enriched with immense semantic information and relationships with additional entities.

An ontology of press clippings from Portuguese newspapers was built with references to minorities and dozens of defined entities, resulting in a corpus of articles, comments, and images featuring semantic relationships. The corpus covers the first two decades of the 21st century. The outcome of this project is an ontological database, that is, a tree of semantic relations and hierarchical references, which aims to contribute to a portrait of the representation of minorities by Portuguese media. Currently, our main corpus is the “Público” newspaper (digital version) from 1996 to 2018 (and the first months of 2019). This data was crawled through the “Arquivo.pt” national archive which collects the history of Portuguese World Wide Web (“.pt” domains). This specific newspaper was used due to its popularity and the fact that it was one of the first journalistic sources to have a digital presence in Portugal.

One of the main goals of the Semantic Web initiative (Web 3.0) is to achieve a general Natural-language Interface, but it suffers from inconsistency, vastness, and vagueness [4]. Linked Open Data attempts to make this ideal come true, but it is still a work in progress. An endeavor to contribute to this ideal was conducted, following the W3C recommendations and most recent standards, intuitive interfaces were built in order to achieve an accessible product. Mainly two ontologies were developed with inferences, using OWL (RDF, Turtle, etc.) to store the data in a GraphDB database, SPARQL to query it and a VUE interface generated dynamically and having multiple endpoints to interact with it.

The ontology was built and enriched by automatically identifying thousands of entities (public figures, political parties, brands, religions, etc.) in different parts of the content (title, body, preview, tags, etc.) as well as a network of relations between them. Besides the newspaper articles themselves, two additional corpus were built: one with the photos used to illustrate the articles and another with the user comments inside the articles.

This data repository, which aims to compile and map the representation of minorities in the Portuguese press using ontologies, was specially designed to serve as study and research material for different research groups from different areas, such as Computer Engineering, Social Sciences, Linguistics, Humanities, Statistics, etc. Some research groups at the University of Minho are working on specific questions on the topics covered, such as, NLP and analysis of the evolution of written styles by the Portuguese cyber users during the last 20 years (user comments); gender studies (articles related to "women" and "sexuality" representations); post-colonial studies ("racial" representation); G2i is studying the portrait and image representation (photos used to illustrate the articles reflect the evolution of media representation of some minorities during the last 20 years); etc.

We invite the scientific community to freely conduct research and new approaches to this data. Every output was released as open-source.

Like Gillespie said, in her study related to media, minorities, and cultural change, the "globalization of communications and cultures articulates new kinds of temporal and spatial relations, it transforms the modes of identification available within societies. Media are being used by productive consumers to maintain and strengthen boundaries, but also to create new, shared spaces in which syncretic cultural forms, such as 'new ethnicities' can emerge. These processes are uneven and their consequences unforeseeable, though likely to be weighty. But they cannot be examined in the abstract or at a distance" [3].

This project tries to bring factual data to this subject, attempting to become a useful research tool. Its main roadmap can be summarized in Figure 2.

In Section 2 a preview of the project outputs and objectives will be summarized. The remaining sections will cover particular aspects concerning the implementation and development stages, namely the Section 3 will focus the data mining steps, the Section 4 will discuss the data treatment, and the Section 5 will present the ontology results and implementation. Finally, Section 6 will discuss the results and future objectives.

2 Project outcomes

From this project resulted a website with different services, two ontologies (ontology a) all the newspaper corpus; ontology b) only the corpus referring minorities) and a public API (GraphDB / SPARQL). Namely, these are some of the main endpoints/URLs:

- **Website:** `http://minors.ilch.uminho.pt`
- **GraphDB (SPARQL public API):** `http://sparql.ilch.uminho.pt`
- **Visual interface:** `http://minors.ilch.uminho.pt/search`
- **SPARQL search:** `http://minors.ilch.uminho.pt/sparql`

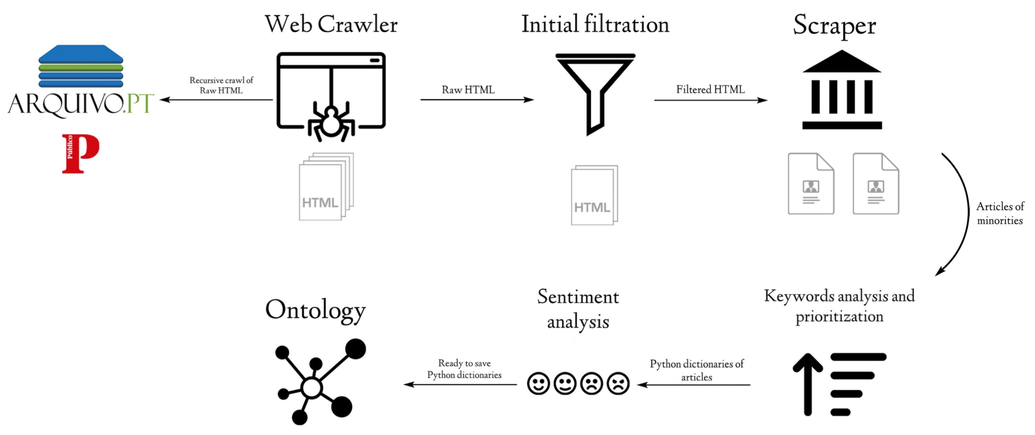

Web Crawler
Initial filtration
Scraper
ARQUIVO.PT
Recursive crawl of Raw HTML
Raw HTML
Filtered HTML
HTML
HTML
Articles of minorities
Ontology
Sentiment analysis
Keywords analysis and prioritization
Ready to save Python dictionaries
Python dictionaries of articles

Figure 2 Data collection and processing flow of events.

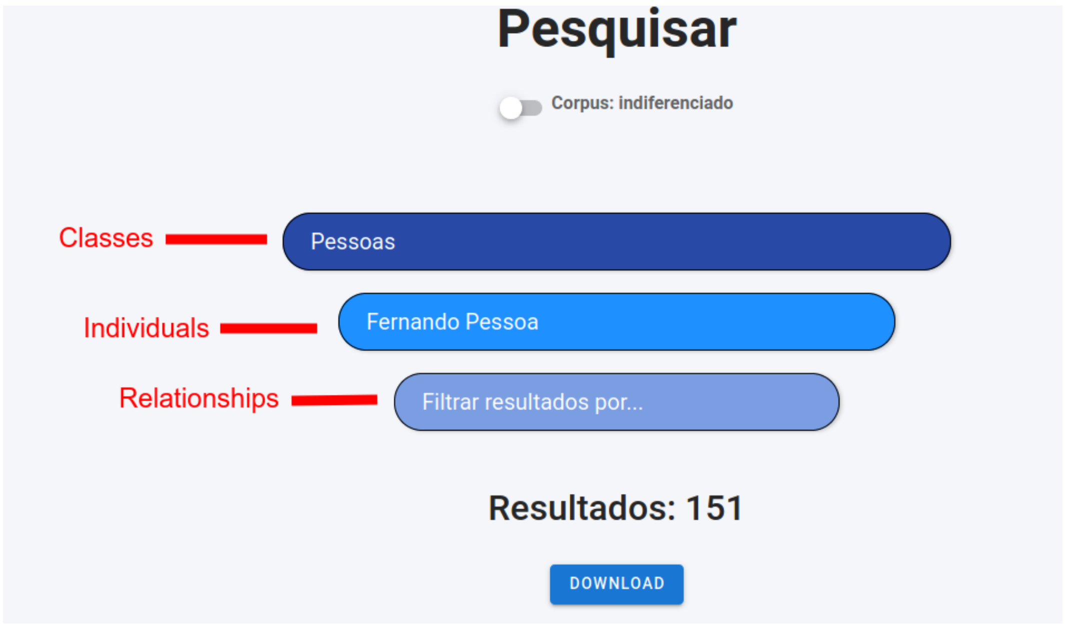
- **Articles:** `http://minors.ilch.uminho.pt/articles`
- **Images:** `http://minors.ilch.uminho.pt/images`
- **Comments:** `http://minors.ilch.uminho.pt/comments`
- **WebVOWL:** `http://minors.ilch.uminho.pt/ontology`

The main objective of this project is to provide a powerful research tool for the scientific community. The project compiles and makes complex data publicly available, which would otherwise be inaccessible to the average user due to the technical complexity involved. In addition to the project website, which counts with reactive search interfaces, and repositories, a SPARQL endpoint that allows complex queries and direct download of large amounts of data is provided.

In a first instance, one can simply navigate the compiled galleries of articles, images, and commentaries in the main website. Its cached data (JSON), previously extracted through the query interfaces, with every possible result for each class, ordered by minority and priority (scoring system).

With the reactive user interface ("Visual navigation") it is possible to filter and extract more complex results: the interface dynamically builds the SPARQL queries and requests the data. One can easily search and download more complex data without technical knowledge of SPARQL or Ontologies, incorporating three possible levels of data filtering: classes, individuals, and some relationships (Figure 3).

But the potential of the platform shines when dealing directly with SPARQL. As an example, with a simple SPARQL query, it is possible to obtain data (XML, JSON, or CSV) with a set of complex and extensive criteria and deep semantic relationships, for example: "display all images that appear associated with articles published in the first quarter of 2016, but only those which are related to "women" minorities and refers in the main text the public figure "António Costa", with the profession of politician, but only if it also includes the keywords "ban" and "abortion" in the body of the text, in addition to mentioning cities in Portugal, excluding the Algarve (etc.)" – the query could be extended to infinity. In a traditional database, with simple text matches, this ease in manipulating data and entity relations, depth, and richness of results would not be possible with the same degree of precision.



Pesquisar

Corpus: indiferenciado

Pessoas

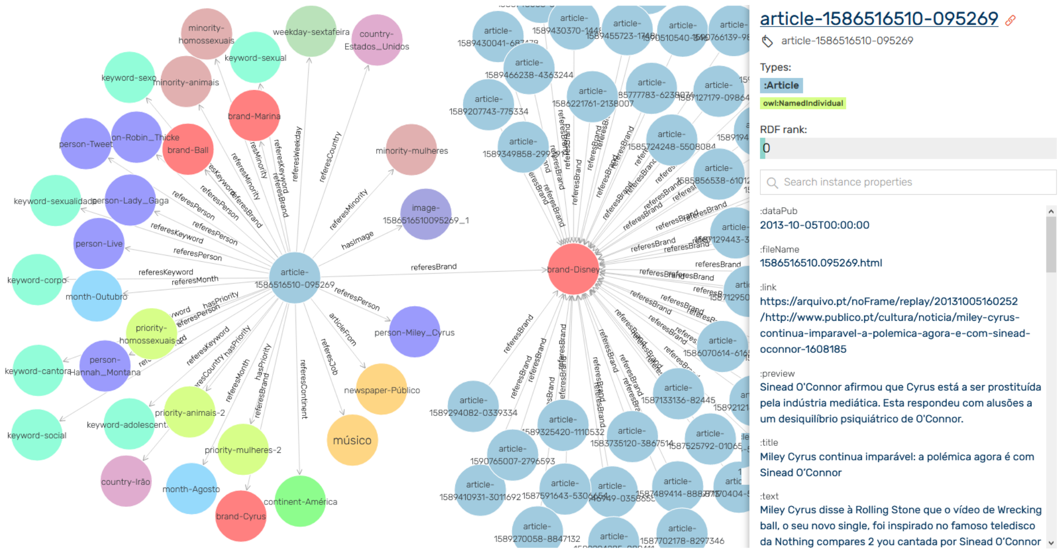
Fernando Pessoa

Filtrar resultados por...

Resultados: 151

Figure 3 Visual navigation.

Figure 4 shows a sample of the ontology, with one of the many articles resulting from a query that extracted articles related to a specific minority but also mentioning a particular company brand (“Disney”).



article-1586516510-095269

article-1586516510-095269

Types:
:Article
owl:NamedIndividual

RDF rank:
0

Search instance properties

:dataPub
2013-10-05T00:00:00

:fileName
1586516510.095269.html

:link
https://arquivo.pt/noFrame/replay/20131005160252/http://www.publico.pt/cultura/noticia/miley-cyrus-continua-imparavel-a-polemica-agora-e-com-sinead-oconnor-1608185

:preview
Sinead O'Connor afirmou que Cyrus está a ser prostituída pela indústria mediática. Esta respondeu com alusões a um desiquilíbrio psiquiátrico de O'Connor.

:title
Miley Cyrus continua imparável: a polémica agora é com Sinead O'Connor

:text
Miley Cyrus disse à Rolling Stone que o vídeo de Wrecking ball, o seu novo single, foi inspirado no famoso teledisco da Nothing compares 2 you cantada por Sinead O'Connor

Figure 4 Example of queried data and relationships extracted from the ontology (only two classes are visually expanded: an Article and the mentioned brand “Disney”).

3 Crawling Phase – Data Extraction

As a starting point, a mean was needed to obtain the raw data that would compose the ontological database. As mentioned before, the reference platform for data extraction was "Arquivo.pt". For that purpose, an open-source crawler for data-mining[1] was built.

"Arquivo.pt", the public repository used for data acquisition, provides a public API for filtering and data search (exact text matches) in their repository, but after some tests, we found it very inefficient for our objective. For example, it wasn't possible to simply search for "women minorities" or the archive would only return a very limited few pages with this exact same match, ignoring the ones with, for example, "feminist protest for the right to abortion" or "young girl was killed by her boyfriend". There was a need to develop a specific algorithm to find pages referring to semantic associations with minorities. But first, full access to the source code of each archived web page was needed in order to be able to interact with the HTML structure and optimize search criteria (title, description, body, tags, etc.). We opted to build semantic datasets, with entities related to different subjects, and searching for them in specific parts of the documents, scoring the results according to an algorithm aimed at establishing a hierarchy of relevant associations useful to identify relevant results and navigate them, for this reason, it was important to have a local corpus of the source-code.

For this reason, a personalized crawler was developed from the ground up, downloading all the "Público" newspaper web pages source-code directly from the "Arquivo.pt", crawling their repository itself directly by building a multidimensional array of inner links and recursively extracting new ones. We started the crawling process from different versions of the "publico.pt" homepage using different snapshots from the archive (we selected the last snapshot of each year and/or each website's new rebranding first edition). For each collected web page, its source HTML code was extracted and stored, and through the use of regular expressions, XPATH, "selenium" and the Python library "requests"[2], URLs found inside the HTML code were recursively collected and processed, avoiding any possibility of blind areas.

The crawler was developed using Python, for which there are multiple auxiliary similar projects (e.g.: Scrappy[3]). However, a custom web crawler was developed from scratch. This decision was based on the freedom intended in some aspects, which is maximized when custom code is used instead of libraries. Thus, full customization on the gathering process and data manipulation were achieved. The crawler is composed of a recursive algorithm that dynamically collects pages and linked pages of a specific pre-defined customizable domain. As mentioned, the used domain was the "Público" newspaper but the algorithm was developed to operate with any defined domain, so other newspapers can be used in the future. Moreover, since millions of pages may be collected from a specific domain, a save logs system was introduced so whenever the crawler stops, the collection can be resumed at any time.

Initially, we ended up with 6 million HTML files, but the majority were repeated content. The reason for this is related to the snapshot system of "Arquivo.pt", we extracted annual snapshots identified by timestamps ending up repeating past content for each one (on purpose, with the objective of covering non-archived versions from different snapshots). "Público" newspaper had half a dozen of different website remakes and more than one domain in its

[1] Crawler's GitHub: `https://github.com/Paulo-Jorge-PM/crawler-majorminors`
[2] `https://docs.python-requests.org/en/master/`
[3] `https://scrapy.org/`

history, this also increased the number of the repeated snapshots. A filtering step was needed to eliminate duplicated content. First, we eliminated non-relevant files (such as marketing campaigns, technical error pages,etc.), selecting only files possibly with articles. In a second step, we used two criteria for isolating individual non duplicated articles: filtered HTML files by their individual unique URIs (but because of the different domains used by the newspaper in the past this was not enough) and later filtered these by individual non repeated titles inside the articles.

After this initial filtering process we ended up with 94.309 individual pages URIs and 48.949 individual articles filtered by duplicated titles (see Table 1). We should add that we ended up with some discrepancies: some years had more extracted content than others (see Figure 5 for total results referring minorities). For example, the newspaper's first year is 1996, but "Arquivo.pt" only has a snapshot of half a dozen pages from that time. Also, during its first years, the newspaper had not much digital content created. This low volume of articles continued until 2004. After this period the "Público" newspaper started many rebranding processes with its digital image and website template, including different domains ("publico.pt", "publico.clix.pt", etc.). This resulted in different HTML source codes, some of them not fully accessible, others dynamically generated and harder to crawl or not collected by the snapshot. Also, the newspaper has a subscription system, blocking access to non-paying users, which nowadays it gives full access to the "Arquivo.pt", but in past years many snapshots instead of the full content showed the blocked access message or cut text. These, and other variables, resulted in different numbers of collected articles for different years and for this reason, the number of minority references found for each year should be treated with caution. Also, the comment system changed many times and limited the archiving, for example after 2010 a login system blocked the "Arquivo.pt" snapshots and we could not retrieve the most recent user comments.

Table 1 Crawled output.

Type	Number
Newspapers articles (individual URIs)	94.309
Newspapers articles (non repeating titles)	48.949
Articles referring minorities (non repeating)	11.496
User comments related with minorities	3.348
Newspapers illustrations related with minorities	9.658

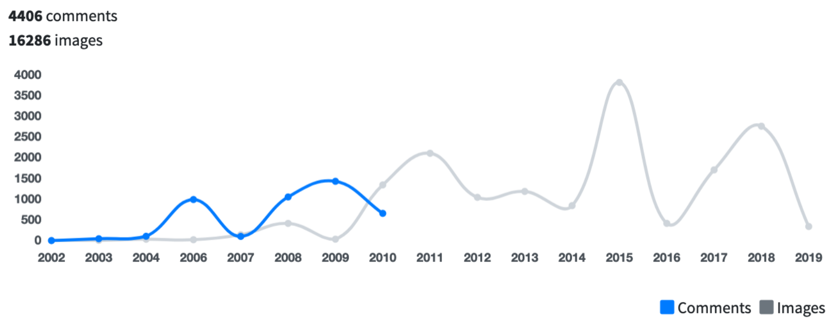


Figure 5 Total number of images and comments filtered per year.

Listing 1 Time intervals of structural changes in Público's articles.

```
end_date_publico2001_2002 = datetime.datetime(2003, 2, 14)
end_date_publico2003_2004 = datetime.datetime(2005, 2, 22)
end_date_publico2005_2006_2007_Jun = datetime.datetime(2007, 5, 18)
end_date_publico2007Jun_Nov = datetime.datetime(2007, 11, 30)
end_date_publico2007Nov_2008_2009Set = datetime.datetime(2009, 9, 30)
end_date_publico2009Set_16Nov2012 = datetime.datetime(2012, 11, 17)
end_date_publico16Nov2012_Out2016 = datetime.datetime(2016, 11, 21)
```

4 Scraping Phase – Data Mining

Once the raw HTML pages were collected by the Crawler, further filtration proceeded. For that purpose we built an open-source scraper for data-extraction[4].

Beforehand, in order to perform the filtering, Python dictionaries were created containing a collection of keywords often associated with the minority groups within the defined scope. These keywords were the result of a study of word occurrences and textual patterns in minority-related articles, which can assist with the identification of a possible minority reference.

In an initial phase, before any cleaning and segregation of elements were preceded, every HTML file which did not contain any of the selected keywords within its structure were excluded. This initial filtration avoided unnecessary post-processing of dispensable pages.

Following the removal of unwanted pages, it was noticed that a considerable part of the collected pages were not article pages, being most of them landing pages, which are also part of the "Público" newspaper domain. The "Arquivo.pt" APIs[1] from which pages were collected, does not feature specific page types filtering, resulting in a mixture of multiple different types of pages. However, the only intended pages were articles, which had a specific pattern of HTML elements. By checking for article-exclusive patterns of HTML elements it was possible to perform another filtration process.

As a result of both filtration processes, all the HTML files left were articles that mentioned minorities-specific keywords. This initial filtering took place before any intensive scraping due to the posterior use of complex regular expressions which are CPU intensive tasks.

Finally, the main scraping algorithm was executed. This algorithm was developed by analyzing the structure evolution of the articles pages from Público. Their structure changed periodically, and a different use case was added to the algorithm for each iteration of the structural changes.

Before entering the main scraping algorithm the date of the page was verified against the defined time intervals (see Listing 1) which then invokes the corresponding function. Each one of the functions start by looking for specific HTML elements, using a combination of the beautifulsoup [5] library and regular expressions. Depending on the date of the article, several elements were collected and stored in separate variables. The elements were Main Title, SubTitle, Preview, Corpus, Tags, Comments and Image URLs.

Once all elements were segregated, each one of them go through a prioritization algorithm. This algorithm made use of the keyword dictionaries and assigned a priority value to the article, depending on the element where the keywords were found. Priority values were assigned as in Table 2.

[4] Scraper's GitHub: `https://github.com/leandrocosta16/scraper-MajorMinors`
[5] `https://pypi.org/project/beautifulsoup4/`

Table 2 Priority points associated with each article elements.

Elements	Priority points
Main tile	7
Tags/Topics	5
Literal mention of the minority name	4
Subtitle/Headline	3
Preview	3
Corpus	2

Firstly, minorities are detected individually for each collected element of the article. Once the minorities are identified for a given element, a data structure containing the identified minorities is iterated and for each one of them the associated element priority value is added cumulatively. Moreover, if a given article references two or more different minority groups it will assign a priority value for each one of them separately.

Although the same list of keywords is used to classify the priority rate, there are some which directly relate to a given entity (e.g.: the literal name of a minority), and those must hold a higher priority value. For this reason, a separate list containing top priority keywords was used and articles which mention them would acquire extra priority points.

This points-based prioritization system provided a way of ordering articles by relevance, with further filtering options in perspective. Higher priority articles are likely to be highly related to the minority in question.

5 Ontology

The main outcomes of this project are the developed ontologies and interfaces to interact with them.

"An ontology is a formal description of knowledge as a set of concepts within a domain and the relationships that hold between them. To enable such a description, we need to formally specify components such as individuals (instances of objects), classes, attributes, and relations as well as restrictions, rules, and axioms. As a result, ontologies do not only introduce a shareable and reusable knowledge representation but can also add new knowledge about the domain" [6]. For example, the YAGO (Yet Another Great Ontology) [7] is one of the largest ontologies in the Linked Open Data cloud, being in itself a knowledge base. This project also aims in being a smaller knowledge base focused on a particular domain of study.

The developed interfaces to interact with the resulted ontologies can be divided in three layers: a basic UI gallery through the main website; a reactive interface that generates SPARQL queries automatically in three layers of filtration (general classes, specific individuals and basic relationships); and a complete SPARQL API and/or visual navigation graph (through GraphDB).

This project was built upon W3C official recommendations: RDF, OWL, and SPARQL. RDF represents information using semantic triples: subject, predicate, and object. We opted for the Turtle syntax for this representation. For storing and managing our triplestores we opted for the GraphDB database[6]. From their official documentation: "GraphDB is an enterprise-ready Semantic Graph Database, compliant with W3C Standards. Semantic

[6] GraphDb: `https://graphdb.ontotext.com/`

graph databases (also called RDF triplestores) provide the core infrastructure for solutions where modeling agility, data integration, relationship exploration, and cross-enterprise data publishing and consumption are important".

GraphDB has free and commercial versions. We implemented the first one, but it is limited by a maximum of two queries in parallel. This added to the fact that our project runs on an older limited server makes the performance not ideal for dealing with many concurrent connections. We created a caching system for improving this limitation, but even so, performance is not our target at the moment, due to the lack of budget, our main focus was accessibility.

In Figure 6 we can see the architecture and inner sequence of the blocks that compose this platform.

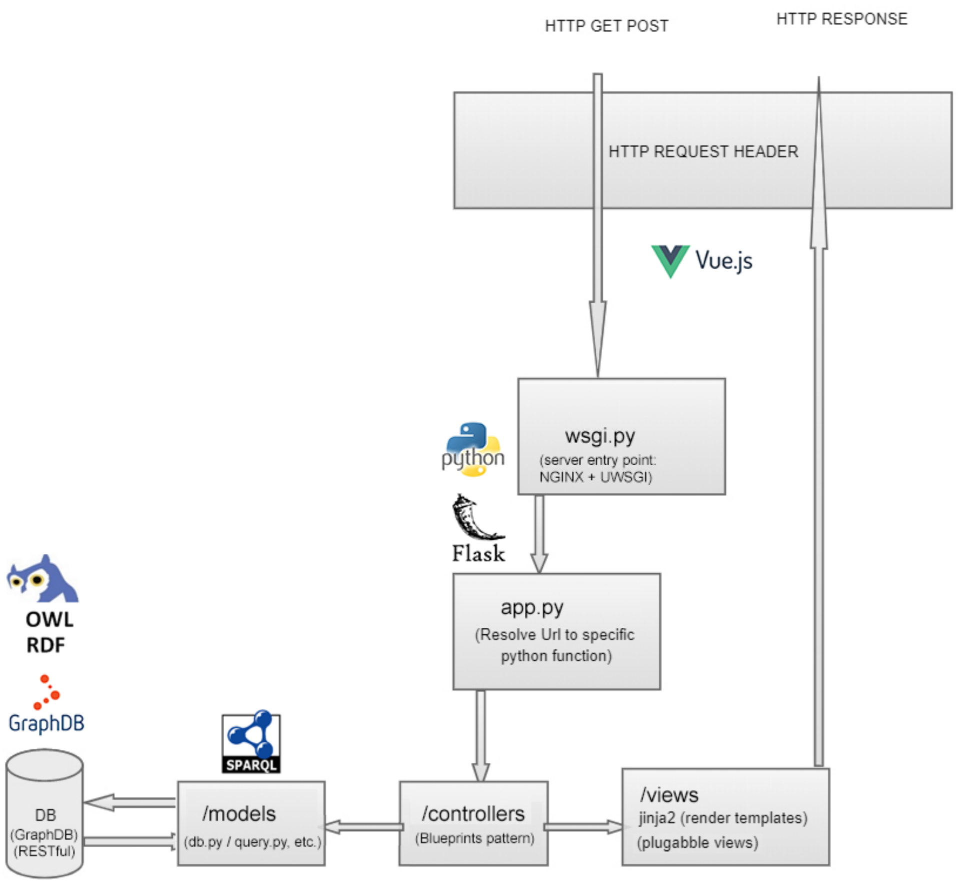


Figure 6 Website: sequence diagram.

Beforehand, preceding the ontology development, datasets were created for each individual set of entities that we aimed to search and build relationships within the articles (see Table 3). These entities (political parties, public figures, brands, etc.) were presented in the previous sections. These datasets were extracted using crawlers, Regular Expressions, XPATH (etc.) from dozens of sources, and Linked Open Data. They are open source and documented in GitHub[7].

[7] Dataset's GitHub: `https://github.com/Paulo-Jorge-PM/datasets-majorminors`

Table 3 Entities extracted.

Type	No. of References
Public Figures	32.648
Political Parties	8.525
Minorities	11.530
Entities	11.473
Brands	46.629
Religions	260
Sports	4.333
Ethnicities	1.345
Car brands	969
Animals	6.199
TV Channels	3.799
Continents	7.119
Countries	28.794
Cities	28.933
Other Places	1.067
Newspapers defined tags	10.186
Weekdays	20.575
Months	24.113
Minorities related keywords	36.897

These entities datasets were used to build the final ontology and the scraper provided inner metadata related to each article (sections, structure, minorities references, etc.). From this point, it was time to build a tree of relationships between these articles (and their images and comments) and the multiple entities referenced in their texts.

For this purpose we built an open-source ontology assembler, that dynamically searched and compared the dataset files with the article's metadata (using Regular Expressions, ontology skeletons, and structured dictionaries) to build the final ontology triplestores and automatically expand the graph tree. It is open-source and its code can be found in GitHub[8].

In Figure 7 a summary of the main classes defined for the ontology is represented and Table 4 is a summary of the ontology outputs. At `http://minors.ilch.uminho.pt/ontology` it is possible to find a more detailed visualization converted to WebVowl with the classes and their properties.

6 Conclusions and Future Work

The intent behind the development of this project is to provide a powerful research tool for the scientific community. The project compiles and makes complex data publicly available, which would normally be inaccessible to the average user due to the technical barrier and complexity overheads.

One of the goals of the Semantic Web is to achieve a general natural-language interface that is both intuitive and universal, fed by a global network of semantic databases. W3C recommended standards like Resource Description Framework (RDF), SPARQL, XML, Web

[8] Ontology assembler: `https://github.com/Paulo-Jorge-PM/ontology-assembler-majorminors`

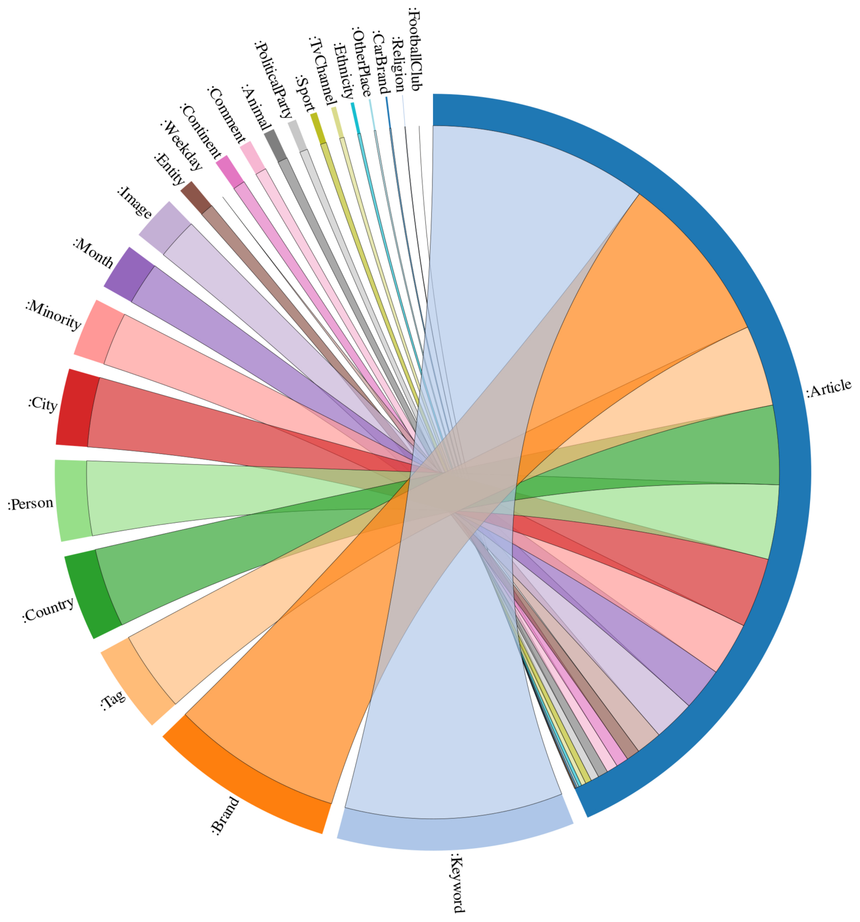


Figure 7 Some of the classes defined in the ontology.

Table 4 Ontology output.

Type	Number
Ontology (All) triples	5.178.169
Infered (All) triples	2.084.435
Ontology (All) size	650,3 Mb
Ontology (Minorities) triples	1.110.411
Infered (Minorities) triples	453.330
Ontology (Minorities) size	125,7 Mb

Ontology Language (OWL) (etc.) have been expanding this area and introducing the concept of Web 3.0, however, the holy grail is still far from being reached: it suffers from inconsistency, vastness and vagueness [4].

This bridge between Semantic Web and Natural-language User Interfaces is one of the main challenges for its growth. With this project, we hope to invite new users to explore the Semantic Web potentialities.

The project currently is in a preliminary stage, with collected data from only one newspaper as a source of data. For future work, the ideal would be to incorporate more sources, extracting data from other main Portuguese newspapers besides the "Público". This requires time and resources, but the platform and tools were built already with this in mind, the ontology structure already incorporates a class for different types of newspapers. In fact, this project was built with flexibility in mind, one could expand it to new languages, countries, or even themes (instead of "minorities" one could clone a similar semantic database and interfaces for extraction of other fields of study).

We invite research groups to freely conduct research upon this tool, they are open-source. On the main website we plan on disclosing research outputs related to the project for future reference, we gladly invite new additions.

References

1 arquivo. pwa-technologies, April 2021. [Online; accessed 19. Apr. 2021]. URL: `https://github.com/arquivo/pwa-technologies/wiki`.

2 Erik Bleich, Hannah Stonebraker, Hasher Nisar, and Rana Abdelhamid. Media portrayals of minorities: Muslims in british newspaper headlines, 2001–2012. *Journal of Ethnic and Migration Studies*, 41(6):942–962, 2015.

3 Marie Gillespie. *Television, ethnicity and cultural change*. Psychology Press, 1995.

4 Catherine C. Marshall and Frank M. Shipman. Which semantic web? *The fourteenth ACM conference*, page 57, 2003. `doi:10.1145/900062.900063`.

5 Robin D Morris. Web 3.0: Implications for online learning. *TechTrends*, 55(1):42–46, 2011.

6 OntoText. What are Ontologies? Ontotext Fundamentals Series, April 2021. [Online; accessed 12. Apr. 2021]. URL: `https://www.ontotext.com/knowledgehub/fundamentals/what-are-ontologies`.

7 Thomas Pellissier-Tanon, Gerhard Weikum, and Fabian Suchanek. Yago 4: A reason-able knowledge base. *ESWC 2020*, 2020. URL: `http://yago-knowledge.org`.

8 Rajiv and Manohar Lal. Web 3.0 in Education. *International Journal of Information Technology*, 3(2):973–5658, 2011.

9 Victoria Shannon. A "more revolutionary" Web - The New York Times, 2006. URL: `https://www.nytimes.com/2006/05/23/technology/23iht-web.html`.

Lyntax – A grammar-Based Tool for Linguistics

Manuel Gouveia Carneiro de Sousa
University of Minho, Braga, Portugal

Maria João Varanda Pereira[1]
Research Centre in Digitalization and Intelligent Robotics,
Polythechnic Insitute of Bragança, Portugal

Pedro Rangel Henriques
University of Minho, Braga, Portugal

Abstract

This paper is focused on using the formalism of attribute grammars to create a tool that allows Linguistic teachers to construct automatically their own processors totally adapted to each linguistic exercise. The system developed, named **Lyntax**, is a compiler for a domain specific language which intends to enable the teacher to specify different kinds of sentence structures, and then, ask the student to test his own sentences against those structures. The processor **Lyntax** validates the grammar (DSL program) written by the teacher, generating a processor every time the student defines a new sentence. For that ANTLR is used in both steps, generating not only the specialized processor but also the visualization of the syntax tree for analysis purposes. An interface that supports the specification of the language was built, also allowing the use of the processor and the generation of the specific grammar, abstracting the user of any calculations.

2012 ACM Subject Classification Human-centered computing → Visualization systems and tools; Social and professional topics → Computer science education; Software and its engineering → Source code generation

Keywords and phrases Attribute Grammars, Linguistic Rules, Pedagogical Linguistic Tools

Digital Object Identifier 10.4230/OASIcs.SLATE.2021.4

Funding This work has been supported by FCT – Fundação para a Ciência e Tecnologia within the Projects Scopes: UIDB/05757/2020 and UIDB/00319/2020.

1 Introduction

Attribute Grammars are a way of specifying syntax and semantics to describe formal languages [3] and were first developed by the computer scientist Donald Knuth in order to formalize the semantics of a context-free language [7]. They were created and are still used for language developing, compiler generation, algorithm design and so on [8]. Due to its similarity with natural language grammars, it seems natural to explore the possibility to use them for linguistics purposes [4]. Using attribute grammars, it is possible to specify the way sentences are correctly written. For instance, using *synthesized attributes*, it is possible to represent the gender of an adjective, while *inherited attributes* can be used to associate the appropriate meaning to a preposition, depending on the context of the sentence [5]. In this way, linguistic rules can be modelled with an attribute grammar, and when a sentence is provided, it is possible to validate its syntax and contextual semantics, adverting for any errors that may be encountered [2].

Applying attribute grammars to model the syntax and semantics of natural languages is a technique that has already been practiced, but it demands knowledge in grammar engineering in order to translate natural languages properties to attribute grammar rules [3]. In spite of

[1] to mark corresponding author

© Manuel Gouveia Carneiro de Sousa, Maria João Varanda Pereira, and Pedro Rangel Henriques; licensed under Creative Commons License CC-BY 4.0

the existing tools, they are not so easily available and straightforward for those who do not have programming and computation proficiency – in this specific case, linguists. There are tools available that use languages with logic components to define linguistic rules but they are not so easy to use. It is easier to rapidly grasp the concepts that are involved in this domain an create a domain specific language to allow the definition of linguistic rules in a higher level of abstraction.

So, the main proposal is to define a new DSL (Domain Specific Language) with a simpler notation, making it easy to understand and learn and to rapidly use. The main focus is to keep the syntax as simple and concise as possible, avoiding complex (or not so common) symbols. This allows the specification of linguistic rules to be done in a much natural manner. Also, a visually appealing user interface was created, granting the user the possibility of analysing the generated syntax-tree.

Moreover, since the teacher specification is agnostic it allows the student to map his sentence written in Portuguese or other natural language to the structure defined by the teacher. That's why the processor is regenerated for each student sentence.

Besides this introduction, this paper is divided in 5 sections: one focused on the state-of-the-art, another one to introduce **Lyntax**, a fourth one explaining two case studies and a last one before conclusion for implementation details.

2 State-of-the-art

As the project here reported aims at developing a tool which allows the definition of linguistic rules in a computational style, it is obvious that such a system can be used to improve the teaching of linguistics in a classroom context. In this section, it will be discussed available tools that can also be used for similar purposes, that deserve to be studied and referenced.

PAG is a tool that was created with the purpose of helping two distinct groups of students from *Universidad Complutense de Madrid*. One of those groups involves computer science students, attending a compiler construction course, and the other group involves linguistic students, from a class on computational linguistics. Teachers from both classes used the same methodology to teach their classes, and noticed that it wasn't good enough for the students to master all the concepts: On one hand, they would have computer science skilled students, with great aptitude to produce solutions, but leaving aside the respective specifications, which lead to poor and inaccurate formal specifications, but on the other hand, linguistic students produced good formal specifications, as they are proficient with the natural language, but lack computer science skills to well transpose all the knowledge into computacional models [6]. The result was an environment based in attribute grammars that allows the specification of those same grammars using a language close to Prolog. The main goal was to embed Prolog into the language and maintain all the familiar basic notation, since both groups of students were already familiar with the Prolog and attribute grammar syntax and notation. Through rapid prototyping, which PAG makes use of, it is possible to obtain a functional processor at a embryonic state of the problem [6]. With this, computer science students can obtain results quite early, allowing them to apply more time into formal specifications. Moreoever, as the complexity of the syntax is reduced, this allows for a better and easier learning experience for students which have less aptitude for the solution codification or programming in general, which is the case for linguistic students. Overall, PAG solved the problem that was purposed in an effective way. Despite that, and giving the respective credit to those who built the tool, the fact is that Prolog can still be quite difficult to grasp for some people, and a challenge when it comes to learn it. The usage of a specification language that closely resembles the natural one, could be a great addition.

CONSTRUCTOR [1] is a Natural Language Interface that accepts and processes English sentences, using them as instructions for plane geometry constructing. Those instructions are then transformed into the respective graphical representation. The idea is that these sentences are issued as commands which represent steps for the creation of a geometrical construction. CONSTRUCTOR analyses the issued input, translates it into a semantic representation and, based on that semantic representation, builds a visual construction. Furthermore, CONSTRUCTOR keeps track of the sequence of inputs issued by the user, which results in a more controlled construction process, while giving the user feedback within each step. The purpose of exploring a tool of this type is not directly related to linguistics nor linguistic rules training. The relevance of this reference is related to the use of attribute grammars with natural language processing, which techniques could be helpful when tacking a specific problem of this kind.

Overall, both tools, despite their differences, share the use of attribute grammars as a basis for a particular system, or even to simplify their use through various techniques. On account of that, it is compelling to create a friendly way to specify and analyse different linguistic rules and their examples, which intends to be the purpose of **Lyntax**.

3 Lyntax: a new proposal to specify linguistic rules

This section presents **Lyntax** as a DSL based solution to describe and test linguistic rules in natural language sentences.

In order to specify all kinds of possible sentences/rules, the idea of creating a "meta-language" emerged. This "meta-language" will be used by the teacher to specify the rules for sentence construction. These rules will be written (in a single file) according to the following structure, that is divided into three main components:

1. **STRUCTURE** – the block where the teacher specifies the structure of the sentences to be tested.
2. **ERRORS/RULES** – list of conditions that should be tested over the sentences. Using ERRORS, if the conditions are matched, an error will be thrown. On the other hand, using RULES, the conditions expressed need to be true to accept the sentence as a valid input.
3. **INPUT** – this block corresponds to the sentence (the lexical part) that the student wants to test. This will be written by the student and then automatically joined with the specification previously described by the teacher.

As can be seen in Figure 1, this file will then be processed by an ANTLR (ANother Tool for Language Recognition – a Compiler Generator) that will work on the information that was written, and then generate a grammar (also specified in ANTLR syntax). This grammar corresponds to the translation of the "meta-language" into ANTLR instructions. Afterwards, the generated grammar will be used to create a validator of sentences; the student can write his sentences (the INPUT that is used to generate the processor and the student's sentence represented in the Figure 1 are the same) and use that validator to obtain confirmation about the sentence correctness. A new processor will be generated for each sentence the student wants to test. The results would be the validation of the given sentences and a tree for a better visualization of the input structure.

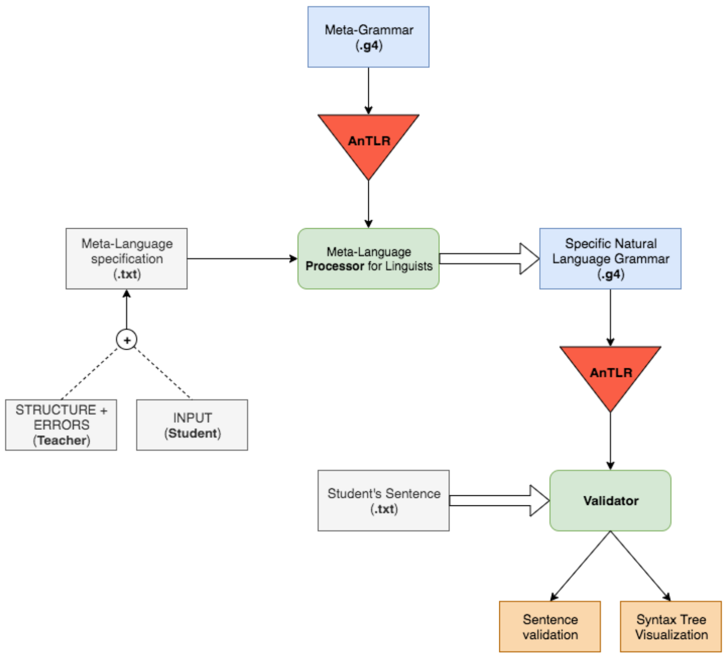

Meta-Grammar (.g4)
AnTLR
Meta-Language specification (.txt)
Meta-Language Processor for Linguists
Specific Natural Language Grammar (.g4)
+
STRUCTURE + ERRORS (Teacher)
INPUT (Student)
AnTLR
Student's Sentence (.txt)
Validator
Sentence validation
Syntax Tree Visualization

Figure 1 System architecture.

3.1 Meta-Language

As it was stated in the introduction of this paper, the main project goal is to create a DSL that must be easy to learn and to grasp. With this in mind, the structure described is composed of three parts, where two of them will be written by the teacher, and the third one is intended to be written by the student; the three parts will be concatenated into a single file.

3.1.1 Domain Specific Meta-Grammar

The created DSL joins the teacher's specification with the student's sentence and its structure is described next.

Listing 1 Processor production.

```
processor : structure errors input
;
```

Firstly, the teacher specification is divided into **structure** and **errors** blocks. The **structure** block is divided into the main **parts** of the sentence. Each part has an **element** containing the information about a certain component.

Listing 2 DSL structure/part/element productions.

```
structure : 'STRUCTURE: ' (part)+ ;

part : 'part' '[' element ']' ;

element : '(' WORD ( '|' WORD )* ( ',' attributes )?
    ( ',' subparts )? ')' ('?')? ;
```

As seen in Listing 2, the **element** is composed of the name of the component, a possible set of **attributes** and possible **subparts**.

Listing 3 DSL attributes/subparts productions.

```
attributes : 'attributes' '{' WORD ( ',' WORD )* '}'
;

subparts : 'subparts' '[' element ( ',' element ) ']'
;
```

The **subparts** production allows *injecting* more elements inside a single component. So, one component may be composed of several other components. As shown in Listing 3, the **subparts** production is a list of one or more elements.

Secondly, the teacher can define a list of restrictions to be applied to each attribute defined in the previous structure. A sentence will be valid if it follows the specified structure and if it obeys to the specified conditions.

Listing 4 DSL errors/conditions productions.

```
errors : ('RULES'|'ERRORS') ':' ( condition ';' )+
;

condition : assignment ( ('AND'|'OR') assignment )*
;
```

The **errors** production will have two meanings“RULES’, then the conditions defined by the teacher need to be checked in order for a sentence to be correct; on the other hand, if the keyword is “ERRORS”, then if the conditions are matched, the sentence is not considered correct within that structure. As shown in Listing 5, the **condition** production is composed of a set of assignments that can be joined using the logical operators “AND” and “OR”. Each condition intends to create logical evaluations for the various attributes defined. Each assignment is composed of expressions.

Listing 5 DSL assignment/expression production.

```
assignment
    : expression ('='|'!=') expression
    | expression ('='!'!=') '"' WORD '"'
;

expression : WORD ( '.' WORD )* '->' WORD
;
```

Each **expression** is composed of the path to a certain attribute, to a value or to other expression. If, for instance, the teacher says that an attribute is equal to some value, then the student can not use other value to that attribute – this would result in an error.

Finally, the **input** block (Listing 6), which corresponds to the students specification. This was treated as a different and separate DSL, as its main purpose was to identify the lexical parts of the sentence written by the student, allowing for a correct and non-subjective parsing of each word in the sentence.

Listing 6 DSL input production.

```
input : 'INPUT:' phrase
;

phrase : ( '-' parts )+
;
```

Each phrase is composed of one or more **parts**, each of them holding various **blocks** (Listing 7), where all the information is stored. Inside, the name of the components and their required attributes must be specified. It is also important to notice that a correct path must be specified by the student. If the student specifies a component that is not declared in the structure defined previously by the teacher, then an error should be thrown.

Listing 7 DSL parts/block/content productions.

```
parts : '(' block ( ',' block )* ')'
;

block : WORD content
;

content : (slice)? (attrs)? (parts)?
;
```

The student can specify the **slice** of the sentence that corresponds to the component that is being declared, and a set of associated attributes (**attrs**). Furthermore, it is possible to continue to define more **parts** within one part, just like the teacher's DSL **subparts**.

Listing 8 DSL slice/attrs/evaluations/eval productions.

```
slice : ':' '``' (WORD)+ '"'
;

attrs : '[' evaluations ']'
;

evaluations : eval ( ',' eval )*
;

eval : WORD '=' '``' WORD '"'
;
```

Inside the **slice** production shown in Listing 8, a list of words can be written. These are the words that will then be used to build the lexical part of the generated grammar. Also, when specifying attributes, the student must assign a value for each attribute that will then be used to validate each component of the sentence.

4 Case Studies

In order to better explain the architecture proposed in the previous section, some concrete examples will be presented in this section. The main idea is to show the specifications used by the teacher and by the student and how the generated processor verifies the correcteness of the student sentences.

4.1 Case Study 1

The example showed in Listing 9 contains a structure that is composed of two main parts: a subject (Sujeito) and a predicate (Predicado). The subject is then subdivided into a possible determiner (Determinante) and a noun (Nome), which are then matched with a word (the lexical part identified by the student). The predicate is composed of a verb and a complement that is directly related to the verb. This complement (Complemento_Direto) is then composed of a possible determiner (Determinante) and a mandatory noun (Nome).

Listing 9 Example of a possible sentence structure defined by the teacher.

```
STRUCTURE:
    part[(
        Sujeito,
        attributes{tipo},
        subparts[
            (Determinante)?,
            (Nome)
        ]
    )]

    part[(
        Predicado,
        subparts[
            (Verbo, attributes{tipo}),
            (Complemento_Direto, subparts[(Determinante)?, (Nome)]),
        ]
    )]

ERRORS:
    Sujeito->tipo = "animado"
        AND Predicado.Verbo->tipo = "inanimado";
    Sujeito->tipo = "inanimado"
        AND Predicado.Verbo->tipo = "animado";
```

In this particular example, both the subject and the verb from the predicate have an attribute named `tipo` which purpose is to check if the component is animated or inanimated. By analysing the ERRORS block (Listing 9), it can be seen that the value of the attribute `tipo` must be the same for both components, otherwise an error should be pointed out.

In this case, the sentence:

"O Carlos teme a sinceridade."

which is in fact a valid sentence, as the name "Carlos" and the verb "teme" are both of the type animated.

Listing 10 Example of a correct student's sentence.

```
INPUT:
    - (Sujeito: "O␣Carlos" [tipo = "animado"]
        (Determinante: "O", Nome: "Carlos"))
    - (Predicado: "teme␣a␣sinceridade"
        (Verbo: "teme" [tipo = "animado"],
         Complemento_Direto: "a␣sinceridade"
            (Determinante: "a", Nome: "sinceridade")))
```

An example of an incorrect input sentence can be seen in Listing 11.

Listing 11 Example of an incorrect student's sentence.

```
INPUT:
    - (Sujeito: "O␣Carlos" [tipo = "animado"]
        (Determinante: "O"))
    - (Predicado: "teme␣a␣sinceridade"
        (Verbo: "teme" [tipo = "animado"],
         Complemento_Direto: "a␣sinceridade"
            (Determinante: "a", Nome: "sinceridade")))
```

In this structure the `Nome` component is missing, an error message identifying the missing component will be printed to the output, as shown in Listing 12.

Listing 12 Example error message of missing component.

```
ERROR: (INPUT)
- The mandatory component 'Nome' has not been defined.
```

4.2 Case Study 2

If, for instance, the main goal of the teacher is to test different attributes despite of the components of a sentence, a simple structure can be defined for that purpose. The following structure and rules aim at testing the gender conformance between two components, and this can be done with very simple sentences.

Listing 13 Example of an arbitrary sentence structure.

```
STRUCTURE:
    part(
        Frase,
        subparts[
            (Determinante, attributes{genero}),
            (Nome, attributes{genero}),
            (Verbo)
        ]
    )

ERRORS:
    Frase.Determinante->genero = "masculino"
    AND
    Frase.Nome->genero = "feminino";

    Frase.Determinante->genero = "feminino"
    AND
    Frase.Nome->genero = "masculino";

    Frase.Determinante->genero != "masculino"
    AND
    Frase.Determinante->genero != "feminino";

    Frase.Nome->genero != "masculino"
    AND
    Frase.Nome->genero != "feminino";
```

Based on the rules written, we can see that the gender must be equal, or the sentence is invalid. Furthermore, the rules ensure that the gender can only be male or female ("masculino" and "feminino" respectively) in order to be a valid sentence. In Listing 14 an example of a possible valid sentence is presented.

Listing 14 Example of an arbitrary sentence input.

```
INPUT:
    - (Frase: "A␣Olinda␣come"
        (Determinante: "A" [genero = "feminino"],
         Nome: "Olinda" [genero = "feminino"],
         Verbo: "come"))
```

A possible mistake that could be done in this particular example is a missing the specification of the attribute attribute `genero` for one of the components as can be seen in Listing 15.

Listing 15 Example of an incorrect arbitrary sentence input.

```
INPUT:
    - (Frase: "A␣Olinda␣come"
        (Determinante: "A" [genero = "feminino"],
         Nome: "Olinda",
         Verbo: "come"))
```

Using the input written above (Listing 15), the error message of Listing 16 is displayed to the user.

Listing 16 Example error message of missing attributes.

```
ERROR: (INPUT)
- There are attributes related to the component 'Nome'
  that were not defined.
```

5 Lyntax: Development

This section will present the development process of the system. Firstly, *OpenJDK* (Open Java Development Kit) was the *Java* platform chosen for that. Secondly, in order to process, execute or translate DSL programs, *ANTLR* **4.8** was used. Lastly, using the *Apache NetBeans*[2] (version **10**) IDE (Integrated Development Environment), it was possible to build the user interface that composes the system.

5.1 Meta-Grammar Processor

At first, the DSL grammar defined in 3.1.1 is used by *ANTLR* to generate a processor. This processor takes the teacher + student specification and constructs an *ANTLR* specification that will be used to generate a specific processor for each student exercise. This new processor is generated to be used by the student to verify if his sentences are correctly following the structure defined by the teacher. As consequence of that some errors like the ones presented in Listing 12 or Listing 16 may occur.

[2] `https://netbeans.apache.org/`

Listing 17 Processor rule from the meta-grammar.

```
processor
@init {
    /* Main data structure. */
    List<RoseTree> struct = new ArrayList<>();

    (...)
}
    : structure[struct]
      errors[struct]
      input[struct]
    {
        (...)
    }
```

The structure presented in Listing 17 is responsible for storing all the information that is being parsed from the file given as input (the meta-language file).

The principle of having a tree as the main data structure falls into the need of maintaining a valid path. For example, if the teacher says that the structure will have a component ***A***, and this component has two children, ***B*** and ***C***, then the paths ***A***→***B*** and ***A***→***C*** should be stored. In this particular problem, it is required to have a tree that within each node has a list of children with an arbitrary size of ***N***. This type of structure is denominated as Rose Tree, which is a prevelant structure within the functional programming community. It is a multi-way tree, with an unbounded number of branches per node.

Listing 18 RoseTree class.

```
class RoseTree {
    String chosenValue;
    String path;
    boolean visited;
    boolean required;
    Map<String, String> attributes;
    Set<String> optionValues;
    List<String> lexical_part;
    List<RoseTree> children;

    (...)
}
```

When in the main production (*processor*), a list of *Rose Trees* is initialized, with each tree of the list corresponding to the main components of the sentence. This structure would travel along the parsing tree, to first be populated with information and then serving as the main source of validation and checking.

On the first block (STRUCTURE) there are not many calculations happening within the productions. The main task is to simply validate the syntax and extract data to be stored in the *Rose Tree*. For each node, it is stored the name of the component, if it is required to be declared or not, a group of attributes (could be non-existent), a lexical part (if it is the case), and finally a list of nodes, referred as the children.

After the parsing of the structure, there is a list of conditions named ERRORS that need to be validated and converted into *Java* syntax – this conversion would then be injected on the main rule of the generated grammar. These logical expressions are based on the attributes of each component and their relations. For example, if the teacher says that a component ***A*** has an attribute named ***a***, and this attribute is required to have value ***x***, if the student assigns it a value of ***z***, then an error should appear. All these conditions can be combined with the logical operands "AND" or "OR". The way a logical expression is parsed is based on the path specified by the teacher when accessing the attribute. Using the example before, a component ***A*** with a child ***B***, with ***B*** having an attribute ***x***, in order to access it, the syntax should be

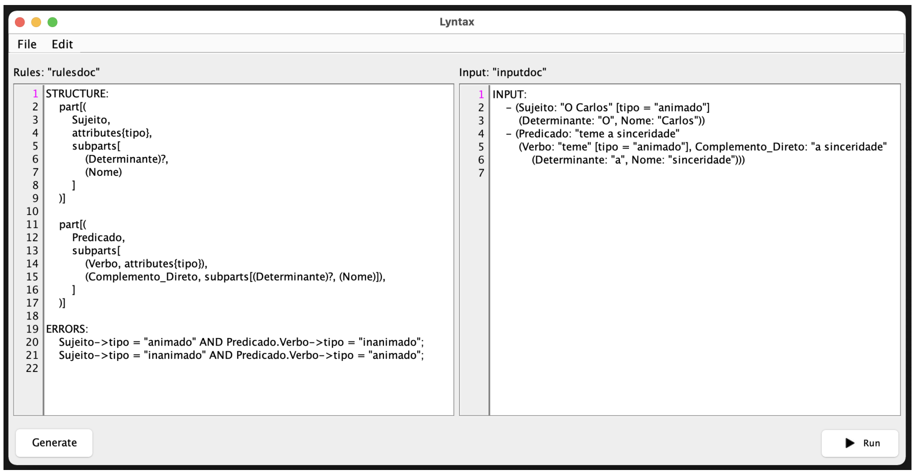
$$A.B \rightarrow x$$

as the full path is required. This is done in order to calculate the correct path and avoid ambiguity between attributes. Over the parsing of these rules, the path is being validated, and in case of any error, the user is notified.

Finally, the last block corresponds to the input that was written by the student. The goal is to validate the components that were defined, and match them with the structure created by the teacher. Again, the RoseTree was used as a way to check if the student's components and paths were valid. The task of the student was to "parse" his sentence and divide it by components, identifying the lexical segments and storing them within a node of the *Rose Tree*. At last, the main rule of the Meta-Grammar makes use of a generator to generate all the rules for the Specific Natural Language Grammar. Within this generator, the various *Rose Tree's* are passed as an argument and then traversed recursively.

5.2 Interface

As stated in the introduction of this paper, after the creation of a system capable of testing various sentences, the goal was to build a user interface that allowed for an easier and simpler use of the system, without the need of using the command line tool built, which takes care of runtime compilations. The interface was built using ***Swing***, a GUI widget toolkit for *Java*. *Swing* has a lot of sophisticated GUI components available for use, allowing the developer to focus on pure functionality (Figure 2). Furthermore, using the *Apache NetBeans* IDE for *Java*, it was possible to use a GUI builder for manipulating *Swing* components, by dragging and dropping them to a canvas – this would generate the specific *Java* code for each component.



Rules: "rulesdoc"

```
STRUCTURE:
    part[(
        Sujeito,
        attributes{tipo},
        subparts[
            (Determinante)?,
            (Nome)
        ]
    )]

    part[(
        Predicado,
        subparts[
            (Verbo, attributes{tipo}),
            (Complemento_Direto, subparts[(Determinante)?, (Nome)]),
        ]
    )]

ERRORS:
    Sujeito->tipo = "animado" AND Predicado.Verbo->tipo = "inanimado";
    Sujeito->tipo = "inanimado" AND Predicado.Verbo->tipo = "animado";
```

Input: "inputdoc"

```
INPUT:
    - (Sujeito: "O Carlos" [tipo = "animado"]
        (Determinante: "O", Nome: "Carlos"))
    - (Predicado: "teme a sinceridade"
        (Verbo: "teme" [tipo = "animado"], Complemento_Direto: "a sinceridade"
            (Determinante: "a", Nome: "sinceridade")))
```

Figure 2 Lyntax user interface.

After the specification of the rules (in the left side) and input (in the right side), the user can generate the Specific Natural Language Grammar to be able to create the Sentence Validator, using the "Generator" button. The text within the two text boxes is concatenated, and given as input for the MetaGrammar processor. All these operations are done in background, following the same order as the instructions showed above. If all goes well, the user should have prompted a message saying that the Grammar was successfully generated (Figure 3) – it is now possible to test the sentence.

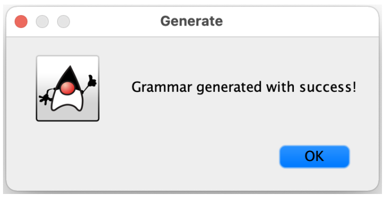


Figure 3 Grammar generation success message.

At last, by clicking the `Run` button, the validator is created, and the sentence passed as input. If no errors occur during this process, the user should see the sentence syntax tree as shown in Figure 4.

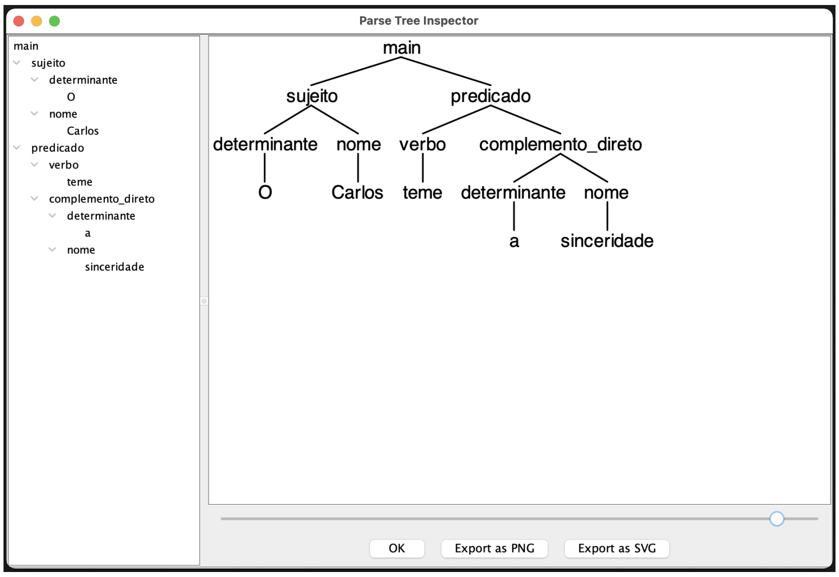


Figure 4 Example of a generated syntax tree within TestRig.

6 Conclusion

The definition of a new well-rounded DSL was done in order to allow common users to define various kinds of sentence structures and linguistic rules. Using AG's, the processing and recognition of that DSL enabled for various types of computations to occur. Using ANTLR, we were able to generate a parser for this DSL that would perform many validations and generate a specific grammar for the sentence that is intended to be tested. Once again, using ANTLR, it is possible to generate a parser from the newly obtained grammar, allowing for the verification of the sentence and the visualization of the respective syntax tree.

The tool designed to provide these functionalities was named **Lyntax**. **Lyntax** is the name of the tool that merges the Meta-Language processor and validator with a user interface that allows for the specification of the language, such as an editor. This interface grants the user the possibility of generating the specific grammar and its respective parser at a high-level, with just the click of a button. The abstraction offered by **Lyntax** allows the user to focus only on the definition and testing of linguistic rules, which is, actually, his main concern.

One very important task that is still to be done is the conduction of tests with the final users – this could be both students from secondary schools or university. The main objective would be to see how the students would react and embrace the tool and its functionalities. For that, the users would be given a set of exercises to be executed using **Lyntax** and a survey with various questions that would try to capture their experience, but also query them about the usefulness of the tool and how it could help or enhance their linguistics studies within the classroom.

Moreover, the tool will be tested for both Portuguese and Spanish exercises in order to prove the versatility of the implemented approach and find commonalities between the defined rules.

References

1 Zoltán Alexin. Constructor : a natural language interface based on attribute grammar. *Acta Cybernetica*, 9(3):247–255, January 1990. URL: `https://cyber.bibl.u-szeged.hu/index.php/actcybern/article/view/3371`.

2 Patrícia Amorim Barros, Maria João Varanda Pereira, and Pedro Rangel Henriques. Applying attribute grammars to teach linguistic rules. In *6th Symposium on Languages, Applications and Technologies (SLATE 2017)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

3 Rahmatullah Hafiz. *Executable attribute grammars for modular and efficient natural language processing*. PhD thesis, University of Windsor, Canada, 2011.

4 Petra Horáková and Juan Pedro Cabanilles Gomar. La concordancia nominal de género en las oraciones atributivas del español: una descripción formal con gramáticas de atributos. *Entrepalavras*, 4(1):118–136, 2014.

5 Donald E. Knuth. The genesis of attribute grammars. In P. Deransart and M. Jourdan, editors, *Attribute Grammars and their Applications*, pages 1–12, Berlin, Heidelberg, 1990. Springer Berlin Heidelberg.

6 José Luis Sierra and Alfredo Fernández-Valmayor. A prolog framework for the rapid prototyping of language processors with attribute grammars. *Electronic Notes in Theoretical Computer Science*, 164(2):19–36, 2006.

7 Kenneth Slonneger and Barry L Kurtz. *Formal syntax and semantics of programming languages*, volume 340. Addison-Wesley Reading, 1995.

8 Krishnaprasad Thirunarayan. Attribute grammars and their applications. In P. Deransart and M. Jourdan, editors, *Attribute Grammars and their Applications*, Berlin, Heidelberg, 1990. Springer Berlin Heidelberg.

Programming Exercises Interoperability: The Case of a Non-Picky Consumer

Ricardo Queirós
CRACS – INESC-Porto LA, Portugal
uniMAD – ESMAD, Polytechnic Institute of Porto, Portugal

José Carlos Paiva
CRACS – INESC-Porto LA, Portugal
DCC – FCUP, Porto, Portugal

José Paulo Leal
CRACS – INESC-Porto LA, Portugal
DCC – FCUP, Porto, Portugal

Abstract

Problem-solving is considered one of the most important skills to retain in the coming decades for building a modern and proactive society. In this realm, computer programming learning is vital to enrich those skills. Practicing in this area boils down to solve programming exercises. In order to foster this practice, it is necessary to provide students with the best of the breed automated tools and a good set of exercises in a fair quantity covering the curricula of a typical programming course. Despite the increasing appearance of automated tools such as program evaluators, gamification engines and sophisticated web environments, access to exercises remains problematic. In fact, although the existence of several code repositories (most for feed computer programming contests), the majority of them store the exercises in proprietary formats and without any access facilities hindering their use. This leaves no other option to teachers but to manually create programming exercises which is time-consuming and error prone, or simply, reuse the same exercises, from previous years, which is considered as a detrimental and limiting approach to enhance multi-faceted and creative programmers.

The article surveys the current interoperability efforts on programming exercises, more precisely, in terms of serialization formats and communication protocols. This study will sustain the selection of an API to feed a code playground called LearnJS with random programming exercises.

2012 ACM Subject Classification Applied computing → Computer-managed instruction; Applied computing → Interactive learning environments; Applied computing → E-learning

Keywords and phrases programming exercises format, interoperability, automated assessment, learning programming

Digital Object Identifier 10.4230/OASIcs.SLATE.2021.5

Funding This paper is based on the work done within the Framework for Gamified Programming Education (FGPE) Plus: Learning tools interoperability for gamified programming education project supported by the European Union's Erasmus Plus programme (agreement no. 2020-1-PL01-KA226-HE-095786).

1 Introduction

The need for educational resources repositories has been growing in the last years since more instructors are eager to create and use digital content and more of it is available. This growth led many to neglect interoperability issues which are crucial to share resources and to reuse them on different domains [11].

© Ricardo Queirós, José Carlos Paiva, and José Paulo Leal;
licensed under Creative Commons License CC-BY 4.0

One of these domains is computer programming. In this realm, the increasing popularity of programming contests worldwide resulted in the creation of several contest management systems fed by code exercise repositories. At the same time, Computer Science courses use programming exercises to encourage the practice of programming. Thus, the interoperability between these types of systems is becoming, in the last decade, a topic of interest in the scientific community [12]. In order to address these interoperability issues, several programming exercise formats and API were developed to expose these resources in a standard fashion.

This article explores the current state of programming exercise interoperability by surveying the most popular programming exercise serialization formats and API for exercise consumption. This study will sustain the selection of an API based on its simplicity and flexibility, to be consumed by a code playground called LearnJS.

The remainder of this paper is organized as follows. Section 2 surveys syntactic interoperability on programming exercises based on how data is serialized and how it is communicated between systems. In Section 3, one of the APIs previously studied is used to fed a code playground called LearnJS. Finally, Section 4 summarizes the main contributions of this research and presents some plans for future work.

2 Programming Exercises

Computer programming is a complex field [2]. One of the most efficient ways to learn to program is through solving exercises. For the practice to be effective, there must be some mechanism that provides feedback on the quality of the learner's resolution. Nowadays, most programming courses typically have large classes and, thus, it is difficult for the teacher to provide timely and consistent feedback [1].

In this context, a set of automated tools has emerged that promote the edition, test, and execution of programs and delivers fast and expressive feedback based on program evaluators. Moreover, several systems combine gamification elements to promote extrinsic and intrinsic motivation to foster students' engagement and loyalty [3]. Despite these advances, the scarcity of programming exercises is still a problem. Although there are several code repositories [11], they do not provide any kind of API hindering its automatic consumption. In addition, those who provide these API return exercises in disparate formats, which leads to the need to use converters to harmonize formats. With this scarcity of exercises and given the difficulty of creating promptly good exercises, teachers often reuse exercises from previous years, which limits creativity.

In the next subsections two surveys on programming exercises interoperability are presented based on data and communication facets.

2.1 Formats

Nowadays, there is a plethora of exercise formats [9]. Most of them were created to formally represent exercises for computer programming contests and are often stored in contest management systems or code repositories.

CATS [1] is a format for programming assignments. The format encoded in XML describes the exercise metadata and a set of additional resources such as statement, tests, test programs, etc. All these files are wrapped up in a ZIP file to facilitate deployment.

[1] `http://imcs.dvgu.ru/cats/docs/format.html`

Freeproblemset (FPS) [2] is a transport file format for the storage of all information about a programming exercise. It aims to provide free exercises sets for managers of ACM/ICPC Online Judges by transporting data from one judge to another. The format uses XML to formalize the description of a programming exercise. It includes information on the exercise itself, test data, special judge data (optional) and solutions (optional).

Mooshak Exchange Format (MEF) is the internal format of Mooshak [3] defined as a system for managing programming contests on the Web [5]. Mooshak is being used in several Universities worldwide to support learning activity. In the competitive context, it was used as the official evaluation system for the IEEE programming contests for some years. MEF includes a XML manifest file referring several types of resources such as problem statements (e.g. PDF, HTML), image files, input/output test files, correctors (static and dynamic) and solution programs. The manifest also allows the inclusion of feedback and points associated to each test.

Peach Exchange Format (PEF) is a specific format for programming task packages used in Peach [4], a system for the presentation, collection, storage, management and evaluation (automated and/or manual) of assignments [14].

YAPExIL [7] is a language for describing programming exercise packages, which builds on top of the XML dialect PExIL [10]. Comparatively, YAPExIL (1) is formalized through a JSON Schema rather than an XML Schema, (2) replaces the logic for automatic test generation with a script provided by the author, and (3) adds several assets that underpin different types of programming exercises. Its JSON Schema can be divided into four separate facets: metadata, containing simple values providing information about the exercise; presentation, including components that are presented to either the student or the teacher (e.g., problem statement and instructions); assessment, involving what enters in the evaluation phase (e.g., tests, solutions, and correctors); and tools, containing any additional tools that complement the exercise (e.g., test generators).

In the last years, several approaches appeared to evaluate the expressiveness of programming exercises formats [4]. One of the most notable approaches is the model proposed by Verhoeff where he describes conceptually the notion of a task package as an unit for collecting, storing, archiving, and exchanging all information concerning with a programming task. The choice of the Verhoeff model over the alternatives is due to its more comprehensive coverage of the required features. This model organizes the programming exercise data in five facets: (1) Textual information – programming task human-readable, (2) Data files – source files and test data, (3) Configuration and recommendation parameters – resource limits, (4) Tools – generic and task-specific tools, and (5) Metadata – data to foster the exercises discovery among systems. Table 1 presents a comparative study of all the referred formats based on the Verhoeff model.

This study confirms the diversity of programming exercises formats highlighting both their differences and their similar features. This heterogeneity hinders the interoperability among the typical systems found on the automatic evaluation of exercises. Rather than attempting to harmonize the various specifications, or working on specific data ingestion approaches, a pragmatic solution could be, for instance, to provide a service for exercises formats conversion [9] based on a pivotal format in which the conversion is based.

[2] `http://code.google.com/p/freeproblemset/`
[3] `https://mooshak2.dcc.fc.up.pt/`
[4] `http://peach3.nl`

Table 1 Comparison of programming exercise formats.

Category	Feature	CATS	FPS	MEF	PEF	PExIL	YAPExIL
Textual	Multilingual			X	X	X	X
	HTML format	X	X	X	X	X	X
	LaTeX format			X			
	Image	X	X	X	X	X	X
	Attach files	X			X		X
	Description	X	X	X	X	X	X
	Grading						X
	Samples					X	X
Data files	Solution	X	X	X	X	X	X
	Skeleton						X
	Multi-language	X	X		X	X	X
	Tests	X	X	X	X	X	X
	Test-groups	X			X	X	X
	Sample tests		X			X	X
	Grading	X		X	X		X
	Feedback			X		X	X
Configuration Recommendation	Compiler						X
	Executor						X
	Memory limit	X	X		X		X
	Size limit						X
	Time limit	X	X				X
	Code lines						X
Tools	Compiler				X	X	X
	Test generator	X				X	X
	Feedback generator			X		X	X
	Skeleton generator					X	X
	Checker	X			X	X	X
	Corrector			X		X	X
	Library	X		X	X		X
Metadata	Exercise	X	X	X	X	X	X
	Author	X			X	X	X
	Event		X		X	X	X
	Keywords			X	X	X	X
	License					X	X
	Platform				X		X
	Management				X		X

2.2 API

There are several code repositories [11]. Despite its existence, few offer interoperability features such as standard formats for their exercises and APIs to foster its reuse in an automated fashion. The most notable APIs for computer programming exercises consumption are CodeHarbor[5], CrimsonHex, FGPE AuthorKit[6], ICPC[7], and Sphere Engine-[8]

[5] `https://codeharbor.openhpi.de/`
[6] `https://python.usz.edu.pl/authorkit/ui`
[7] `https://clics.ecs.baylor.edu/index.php/Contest/API`
[8] `https://docs.sphere-engine.com/problems/api/quickstart`

FGPE Authorkit [6] is a Web programming exercises management tool that was created as part of the Erasmus+ Project entitled Framework for Gamified Programming Education (FGPE). The Authorkit aims to foster the creation of gamified programming exercises. The Web application allow users to prepare gamified programming exercises divided into two facets: to create the content of exercises and to assign a gamification layer. The former allows to specify all the components of a programming exercise (e.g. problem statement or input/output test files). The latter allows the definition of game-based data such as rewards, rules, leaderboards and challenges. Both data types are stored in two dedicated formats – Yet Another Programming Exercises Interoperability Language (YAPExIL) and Gamified Education Interoperability Language (GEdIL) [13]. The Authorkit expose its functions, with a dedicated API, allowing users to import content from popular non-gamified exercise formats, and export it or share it with other peers, internally or via a GitHub repository where all exercise data is synchronized.

Sphere Engine is a set of API that enable creating coding assessment solutions and online code execution environments. It is composed by two API: compilers [9] and problems [10]. The former allows users to execute computer programs in a secure run-time environment and receive feedback on the resolution. The latter allows users to create, edit, and manage programming problems. In order to use the API one should have an API token. After authenticating into the Sphere Engine client panel, it is possible to get a token in the API Tokens section. The main features of the API are create, edit, and manage programming exercises, define test cases, and import/export exercises. Using the export endpoint, clients have access to all information about an exercise including a manifest file called config.xml with references for all the assets wrapped in a ZIP file.

ICPC API is a well-known API for accessing information provided by a Contest Control System or Contest Data Server. This API is meant to be useful, not only at the ICPC World Finals, but more generally in any ICPC-style contest setup such as an external scoreboard or a contest analysis software. The API makes available several objects as JSON elements such as exercises, teams and members, submissions, runs, awards, contest state and scoreboards.

CodeHarbor is a repository system which allows to share, rate, and discuss auto-gradeable programming exercises. The system enables instructors to exchange exercises through the proFormA XML format across diverse code assessment systems. Despite all these features, most of them are made through an user interface, and the API is only available for grading scenarios.

crimsonHex [11] repository aims to store programming exercises as learning objects. It provides a set of operations based in the IMS DRI specification and exposed through a REST API. The main functions are (1) the Register/Reserve function to book a new resource identifier from the repository, (2) the Submit/Store function push an exercise to the repository, (3) the Search/Expose function enables external systems to query the repository using the XQuery language, (4) the Report/Store function associates a usage report with an existing exercise, and (5) the Alert/Expose function notifies users of changes in the state of the repository using an RSS feed.

The comparison of API can be made through several approaches. Despite the plethora of options the most popular are: architectural styles, domain coverage, and analytics. The former explores architectural key aspects of the API such as design patterns, communication protocols and encoding types. The second focuses on the coverage rate of the API in relation

[9] `https://docs.sphere-engine.com/compilers/overview`
[10] `https://docs.sphere-engine.com/problems/overview`

to the features of the system. The latter focuses on engineering metrics such as performance and uptime, but also customer and product metrics such as engagement, retention, and developer conversion. In this study, the first two approaches will be used.

The three key aspects to be aware in the architectural styles facet are the (1) design philosophy/pattern (e.g., RESTful vs GraphQL), (2) communication protocol (e.g., HTTP vs WebSockets), and (3) encoding (e.g., human-readable text such as JSON vs Binary formats like ProtoBuf). Often, these three different aspects can be mixed together. For instance, one can use RESTful API over WebSockets but using a Binary Protocol (e.g. MessagePack).

Regarding the domain coverage of the API in the programming exercises domain, the most important features are the management of exercises such as create, read, update, delete, import/export exercises; the submission/grading features; the gamification features such as challenges and scoreboards and, finally, the user management features such as assigning user to teams.

Table 2 presents the comparison of the previous API based on two criteria: architectural styles and domain functions.

Table 2 Comparison of API from code repositories.

Category	Facet	CodeHarbor	crimsonHex	FGPE	ICPC	Sphere
Architectural Style	Design pattern	REST	SOAP REST	REST GRAPHQL	REST	REST
	Communication protocol	HTTP	HTTP	HTTP	HTTP	HTTP
	Encoding	XML	XML	JSON	JSON	JSON
Functions	CRUD	NO	YES	YES	YES	YES
	Import/Export	NO	YES	YES	YES	YES
	Submit/Grade	NO	NO	NO	NO	YES
	Game elements	NO	NO	YES	YES	NO
	Groups & Users	NO	NO	YES	YES	YES

Based on this study, one can conclude that, currently, the most popular approach to create web API is through RESTful API using the JSON format on the HTTP protocol. The coverage of the API is done essentially in the exercises management CRUD and import/export features.

3 Use case: LearnJS

This section presents the basic steps for the use of the FGPE Authorkit API by a code playground called LearnJS [8] defined as Web playground which enables anyone to practice the JavaScript language. More specifically, the playground allow users to solve programming exercises of several types (e.g. blank sheet, skeleton, find the bug, and quizzes) and to receive prompt feedback on their resolutions. In LearnJS, students can see videos or PDF of specific topics and solve exercises related with those topics with automatic feedback on their resolutions. Currently, the playground has two main components: (1) an Editor which allows students to code their solutions in an interactive environment and (2) an evaluator which evaluates the student's code based on static and dynamic analyzers. At this moment, a simple running prototype [11] (version 0.7.7) is available.

[11] https://rqueiros.github.io/learnjs

The use of the AuthorKit API by LearnJS will allow the support of a new type of exercise called random that, after selection, will make the playground use the API to fetch a random exercise from the Authorkit repository. The first step towards the integration is to register in the AuthorKit with the name, email, and password data in the request body. If the registration is successful, the next step is to log in using the registration data. This action will return a token that should be used in any subsequent requests.

In the AuthorKit, exercises are assigned to projects that act as containers. For that reason, the next step is to get a list of projects. Each project has a set of metadata such as an identifier, a title, a status, the number of contributors, and exercises. With a project identifier, it is possible to get all its exercises. An exercise is composed of several properties such as an identifier, a title, a type, keywords, a difficulty level, and a status. To obtain assigned assets such as the statement or tests it is necessary to explicitly append in the correspondent endpoint the type of assets to obtain.

Table 3 presents the endpoints used for this integration (base URL: `http://fgpe.dcc.fc.up.pt/api`.

Table 3 Endpoints of FGPE AuthorKit API used in the integration.

Functions	Method	REST API
Register	POST	/auth/register
Login	POST	/auth/login
GetProjects	GET	/projects
GetExercises	GET	/exercises?page=1&limit=6
GetExercise	GET	/exercises/:id
GetExerciseAssets	GET	/exercises/:id?join=statements&join=tests
ExportExercise	GET	/exercises/:id/export

Note that in order to effectively get the assets you should always use the suffix /contents in the endpoints. For instance, for each test, the contents of input and output are in /tests/:id/input/contents and /tests/:id/output/contents. The same logic can be used to fetch the statement of the exercise as shown in Listing 1.

Listing 1 Inject an exercise statement in the LearnJS GUI.

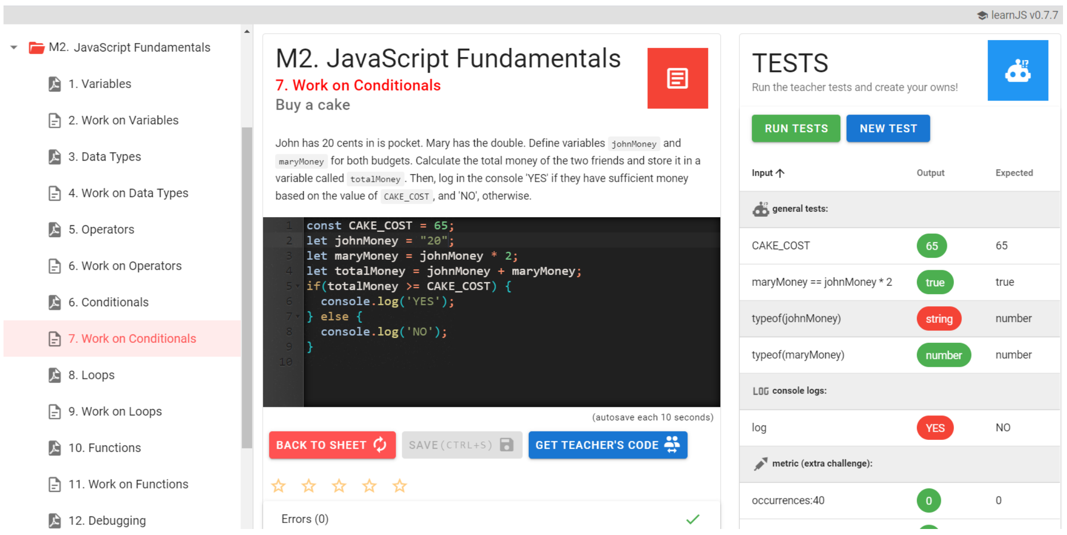
```
...
// Get statement ID
const resp = await fetch(`${baseUrl}/exercises/${id}?join=statements`)
const result = await resp.json();
const sId = result.statements[0].id

// Get statement (base64 string)
const statement = await fetch(`${baseUrl}/statements/${sId}/contents`)
const statementBase64 = await statement.text();

// Document fragment creation
let fragment = document.createDocumentFragment();
fragment.appendChild(window.atob(statementBase64))

// Injection of the statement in the playground UI through DOM
document.querySelector('#sDiv').innerHTML = fragment.body.innerHTML
...
```

The final result is shown in Figure 1 where the statement of the exercise appears in the top panel of the screen of the LearnJS GUI.



M2. JavaScript Fundamentals

7. Work on Conditionals

Buy a cake

John has 20 cents in is pocket. Mary has the double. Define variables `johnMoney` and `maryMoney` for both budgets. Calculate the total money of the two friends and store it in a variable called `totalMoney`. Then, log in the console 'YES' if they have sufficient money based on the value of `CAKE_COST`, and 'NO', otherwise.

```
const CAKE_COST = 65;
let johnMoney = "20";
let maryMoney = johnMoney * 2;
let totalMoney = johnMoney + maryMoney;
if(totalMoney >= CAKE_COST) {
  console.log('YES');
} else {
  console.log('NO');
}
```

(autosave each 10 seconds)

☆ ☆ ☆ ☆ ☆

Errors (0) ✓

TESTS

Run the teacher tests and create your owns!

Input ↑	Output	Expected
general tests:		
CAKE_COST	65	65
maryMoney == johnMoney * 2	true	true
typeof(johnMoney)	string	number
typeof(maryMoney)	number	number
LOG console logs:		
log	YES	NO
metric (extra challenge):		
occurrences:40	0	0

Figure 1 LearnJS playground GUI.

4 Conclusion

This article surveys the current state of programming exercises interoperability at two levels: data and communication. In the former, several exercise formats were identified and compared using the Verhoeff model. In the latter, several API have been studied based on their architectural styles and covered functions.

The ultimate goal of the survey is to support the best decisions to do in the process of integrating a new exercise type in the LearnJS playground. This integration consists of fetching exercises from the FGPE AuthorKit API and transforming them from their original format – YAPExIL – to the LearnJS internal format. As future work, the goal is to benefit from the gamified layer of the FGPE AuthorKit to gamify the LearnJS playground using the same API.

References

1 Kirsti M. Ala-Mutka. A survey of automated assessment approaches for programming assignments. *Computer Science Education*, 15(2):83–102, 2005. doi:10.1080/08993400500150747.

2 Yorah Bosse and Marco Aurélio Gerosa. Why is programming so difficult to learn? patterns of difficulties related to programming learning mid-stage. *SIGSOFT Softw. Eng. Notes*, 41(6):1–6, 2017. doi:10.1145/3011286.3011301.

3 Patrick Buckley and Elaine Doyle. Gamification and student motivation. *Interactive Learning Environments*, 24(6):1162–1175, 2016. doi:10.1080/10494820.2014.964263.

4 Stephen H. Edwards, Jürgen Börstler, Lillian N. Cassel, Mark S. Hall, and Joseph Hollingsworth. Developing a common format for sharing programming assignments. *SIGCSE Bull.*, 40(4):167–182, 2008. doi:10.1145/1473195.1473240.

5 José Paulo Leal and Fernando Silva. Mooshak: a web-based multi-site programming contest system. *Software: Practice and Experience*, 33(6):567–581, 2003. doi:10.1002/spe.522.

6 José Carlos Paiva, Ricardo Queirós, José Paulo Leal, and Jakub Swacha. Fgpe authorkit – a tool for authoring gamified programming educational content. In *Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education*, ITiCSE '20, page 564, New York, NY, USA, 2020. Association for Computing Machinery. doi:10.1145/3341525.3393978.

7 José Carlos Paiva, Ricardo Queirós, José Paulo Leal, and Jakub Swacha. Yet Another Programming Exercises Interoperability Language (Short Paper). In Alberto Simões, Pedro Rangel Henriques, and Ricardo Queirós, editors, *9th Symposium on Languages, Applications and Technologies (SLATE 2020)*, volume 83 of *OpenAccess Series in Informatics (OASIcs)*, pages 14:1–14:8, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. `doi:10.4230/OASIcs.SLATE.2020.14`.

8 Ricardo Queirós. LearnJS - A JavaScript Learning Playground (Short Paper). In Pedro Rangel Henriques, José Paulo Leal, António Menezes Leitão, and Xavier Gómez Guinovart, editors, *7th Symposium on Languages, Applications and Technologies (SLATE 2018)*, volume 62 of *OpenAccess Series in Informatics (OASIcs)*, pages 2:1–2:9, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. `doi:10.4230/OASIcs.SLATE.2018.2`.

9 Ricardo Queiros and Jose Paulo Leal. Babelo—an extensible converter of programming exercises formats. *IEEE Trans. Learn. Technol.*, 6(1):38–45, January 2013. `doi:10.1109/TLT.2012.21`.

10 Ricardo Queiros and José Leal. Pexil - programming exercises interoperability language. In *title = "XATA 2011: XML: associated technologies and applications"*, January 2011.

11 Ricardo Queirós and José Paulo Leal. crimsonhex: a learning objects repository for programming exercises. *Software: Practice and Experience*, 43(8):911–935, 2013. `doi:10.1002/spe.2132`.

12 Alberto Simões and Ricardo Queirós. On the Nature of Programming Exercises. In Ricardo Queirós, Filipe Portela, Mário Pinto, and Alberto Simões, editors, *First International Computer Programming Education Conference (ICPEC 2020)*, volume 81 of *OpenAccess Series in Informatics (OASIcs)*, pages 24:1–24:9, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. `doi:10.4230/OASIcs.ICPEC.2020.24`.

13 Jakub Swacha, José Carlos Paiva, José Paulo Leal, Ricardo Queirós, Raffaele Montella, and Sokol Kosta. Gedil—gamified education interoperability language. *Information*, 11(6), 2020. `doi:10.3390/info11060287`.

14 Tom Verhoeff. Programming task packages: Peach exchange format. *Olympiads in Informatics*, 2:192–207, January 2008.

DataGen: JSON/XML Dataset Generator

Filipa Alves dos Santos
University of Minho, Braga, Portugal

Hugo André Coelho Cardoso
University of Minho, Braga, Portugal

João da Cunha e Costa
University of Minho, Braga, Portugal

Válter Ferreira Picas Carvalho
University of Minho, Braga, Portugal

José Carlos Ramalho
Department of Informatics, University of Minho, Braga, Portugal

Abstract

In this document we describe the steps towards DataGen implementation.

DataGen is a versatile and powerful tool that allows for quick prototyping and testing of software applications, since currently too few solutions offer both the complexity and scalability necessary to generate adequate datasets in order to feed a data API or a more complex APP enabling those applications testing with appropriate data volume and data complexity.

DataGen core is a Domain Specific Language (DSL) that was created to specify datasets. This language suffered several updates: repeating fields (with no limit), fuzzy fields (statistically generated), lists, highorder functions over lists, custom made transformation functions. The final result is a complex algebra that allows the generation of very complex datasets coping with very complex requirements. Throughout the paper we will give several examples of the possibilities.

After generating a dataset DataGen gives the user the possibility to generate a RESTFull data API with that dataset, creating a running prototype.

This solution has already been used in real life cases, described with more detail throughout the paper, in which it was able to create the intended datasets successfully. These allowed the application's performance to be tested and for the right adjustments to be made.

The tool is currently being deployed for general use.

2012 ACM Subject Classification Software and its engineering → Domain specific languages; Theory of computation → Grammars and context-free languages; Information systems → Open source software

Keywords and phrases JSON, XML, Data Generation, Open Source, REST API, Strapi, JavaScript, Node.js, Vue.js, Scalability, Fault Tolerance, Dataset, DSL, PEG.js, MongoDB

Digital Object Identifier 10.4230/OASIcs.SLATE.2021.6

1 Introduction

Every application and software developed should be thoroughly tested before release, in order to determine the system's ability to withstand realistic amounts of data and traffic, and that implies the usage of representative datasets that fit its use cases. The creation of said datasets is a laborious and drawn out process, as it implies firstly generating the test data in some way, in a file format compatible with the system. As it stands, there are currently no efficient, intuitive and scalable tools to do this and so, developers often end up having to create test records either by hand, which is incredibly inefficient and time-consuming, or by using existing tools with some clever tactics to manage their shortcomings. As a matter of fact, many projects are not able to progress to the development stage due to the lack of adequate and sufficient data [3].

© Filipa Alves dos Santos, Hugo André Coelho Cardoso, João da Cunha e Costa, Válter Ferreira Picas Carvalho, and José Carlos Ramalho;
licensed under Creative Commons License CC-BY 4.0

Even with a reliable generation tool, the user might want to have control over the data in the resulting records, being able to observe and manipulate it freely, through CRUD requests, and adapting it however they want. Though there are products capable of doing this – for example the package json-server, which creates a full fake REST API –, its usage entails the user manually starting the application every time they want to edit the dataset's contents and ensuring the data's format is compliant with the software they're using, which ends up requiring a lot of extra effort and setup on the user's side.

As such, the team came up with the idea of coupling the generation process and the REST API, in a way that allows the user to automatically generate data compatible with an integrated RESTful API – for which the software Strapi was chosen, as will be explained in more detail in Section 3.5 –, allowing the user to load the records into an API server and perform CRUD operations over it, either through the user interface or via HTTP requests.

This paper will cover the development of the resulting application, DataGen – a more versatile and powerful tool for data generation, according to the user's specifications, and subsequent handling –, seeking to explain the decisions that were taken for its architecture, as well as all the implemented functionalities.

2 Related Work

User privacy concerns [13] have been a major topic of discussion over the last decades, which lead to the creation of strict regulations regarding the way sensitive data should be handled, such as the EU's General Directive on Data Protection GDPR [2]. These regulations keep entities from disclosing private and sensitive information, which in turn hurts new growing ideas and projects that would require access to similar data. As it stands, not only is the lack of available data a big problem in this context, as are the data sharing agreements themselves, as their ratification tends to take, on average, a year and a half [10], which proves to be fatal for many small and upcoming companies.

To circumvent these concerns, organisations have been increasingly adopting synthetic data generation [15], an approach that was originally suggested by Rubin in 1993 [1] in order to create realistic data from existing models without compromising user privacy. The prospect of being able to create viable information that does not relate to actual individuals is a very enticing solution for the privacy conundrum. If perfected, it could potentially provide the capacity of generating sizeable amounts of data according to any use cases, bypassing the need to access real users' information. As such, this approach has been increasingly adopted and put to the test, in order to measure its efficiency with real-life cases [5, 18, 17].

Dataset generation has become a requisite in many development projets. However, the majority of developed tools produce datasets for specific contexts like intrusion detection systems in IoT [4], crops and weed detection [6], vehicular social networks based on floating car data [12], 5G channel and context metrics [16], GitHub projects [9], to cite a few. Many others exist in different contexts like medicine, bioinformatics, weather forecasts, color constancy, market analysis, etc.

Most of the researched tools are domain specific. The team's goal is to create a generic tool but powerful enough to generate datasets for several and different contexts and with many levels of complexity. There are some tools available, many online, but they cover the generation of simple datasets, many times flat datasets.

The main source of inspiration for this project was an already existing web application called "JSON Generator", developed by Vazha Omanashvili [11], as it is the most interesting dataset generation tool that was found. It features a DSL (Domain Specific Language) that's parsed and converted into a JSON dataset, allowing the user to generate an array of objects that follow some predefined structure, specified in the DSL.

In terms of utility, it is a very powerful tool that can generate very complex data structures for any application with relatively little effort. However, it had some shortcomings which initially inspired the team to develop an alternative solution that attempts to address them.

Specifically, these shortcomings are:

1. Limited size for the “repeat” statement (100 total). Arguably, the size of the dataset itself is one of the most important features to take into account. For applications on a larger scale, having a small amount of array elements does not allow for more realistic tests, as very few, if any, that are developed in a production environment use as little as 100 entries created by the aforementioned directive.
2. It only generates JSON. Despite being one of the most popular file formats, there are many others that could be useful to the user as they might want the data to be in a different format for their application without converting the JSON themselves, such as XML (another open text format [19]). This allows for further flexibility and expands the possible use cases that it provides.
3. Does not generate a RESTful API for the dataset. Many users may optionally want their newly generated dataset hosted and exposed by a RESTful API for direct usage in their web applications, or to download a custom one created specifically for their dataset for later deployment on a platform of their choosing.
4. Does not take into account fuzzy generation. Some elements of a dataset may not be deterministic and are represented by probabilities. For example, there may a field that exists only if another property has a specific value and the application should be able to take that into account.
5. It does not have much data available. For instance, the user might want to use names from a list of famous people for their dataset, as it allows for a more realistic generation and consistency, which this tool currently does not provide.
6. It is not multilingual. The data that this application uses is only available in English, it would be more user-friendly to give them the choice to use their preferred language for the dataset instead of forcing it to just one.
7. Does not take into account integration on applications. The generation and download of a dataset requires the consumer to use the website’s interface – this is not ideal as many applications may want to use HTTP requests to automate this process for internal usage.
8. Does not support functional programming features. Functions such as “map”, “reduce” and “filter” that are staple in the functional paradigm due to their simplicity and effectiveness are not present in this application. This would allow the user to chain statements and transform fields to the result they want, granting the application the ability to deal with more complex use cases.

With clear goals and an initial application to take inspiration from, the team proceeded to the development stage by deciding on its architecture (i.e. programming languages, frameworks, external tools, etc), which will be explained in the following sections.

3 DataGen Development

Building the application from the ground up requires a divide and conquer approach, since having a monolithic single server architecture will lead to a less stable user experience, due to the lack of scalability.

Having the application compartmentalized in multiple servers, each with their specific function, allows for a much more sensible and balanced architecture since it leaves room for the possibility of individually upgrading each of them, leading to a much higher scalability and fault tolerance, as the failure of one component does not compromise the functionality of the entire application, allowing for quicker and easier routine maintenance.

The following subsections will explain how the current architecture was achieved and the technological decisions behind each of the components.

3.1 Architecture

The picture below shows the general scope of the architecture, albeit simplified. This architecture allows for major upgrades, such as load balancers on both the back and front-end since they are both **stateless** (the JWT approach allows the servers to not maintain sessions with each user's state) and using MongoDB as a distributed database – sharded cluster.

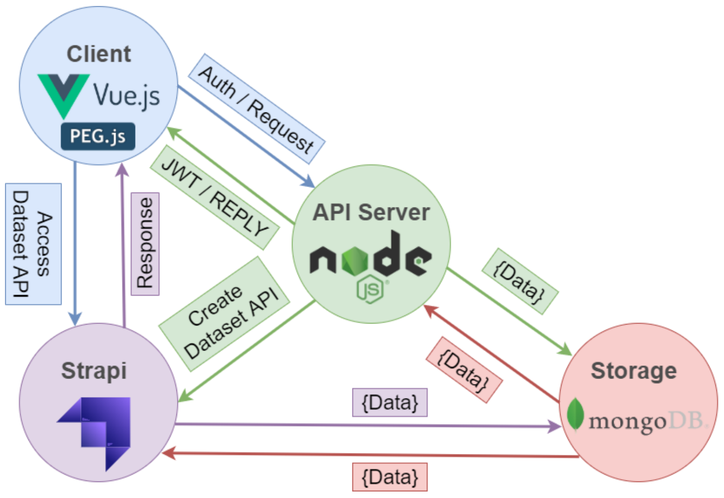


Figure 1 Current architecture of DataGen.

3.2 Front-End

The first component needed for the application is the front-end server, which is responsible for building and showing the website's interface to the user, making it the entry point to the application and its scripted behaviour.

3.2.1 Grammar

The application uses a PEG.js grammar-based parser [8] [14] to process the user's input and generate the intended dataset. The aforementioned grammar defines a domain-specific language (DSL), with JSON-like syntax, providing many features that allow for the generation of complex and diverse datasets. These features include relational and logic capabilities, providing means for the datasets to satisfy several forms of constraints – which push towards the use of some declarative framework for this specification –, as well as functional capabilities, allowing for easier management and processing of certain pr of the datasets.

The first and most fundamental of said features is the JSON-similar syntax – the user can specify key-value properties, where the value may take any of the basic JSON types and data structures, from integers to objects and arrays. The user may also nest these to create a structure with any depth they may want.

```
name: {
  first: ["John", "William"],
  last: "Doe"
},
age: 21
```

To specify the size of the dataset, or a nested array, there is the **repeat** statement, where the user indicates the structure that they want replicated (which may range from a primitive JSON type to a complex object), as well as the number of copies, or range of numbers.

```
names: [ 'repeat(150,200)': {
    first: '{{firstName()}}',
    last: '{{surname()}}'
} ]
```

They may also specify as many collections as they want in a single model (collections are the key-value properties on the uppermost level of the model) and the application will return the resulting dataset in json-server syntax – an object with a property for each collection. During the parsing of the input, the application recursively builds both the final dataset and the Strapi model for the specified structure, concurrently, in order to allow for posterior integration in a RESTful API.

```
{
  names: [ 'repeat(10)': '{{fullName()}}' ],
  animals: [ 'repeat(20)': '{{animal()}}' ]
}
```

To define the value of a property, the user may also use interpolation. To access an interpolation function, it must be encased in double curly braces. There are two types of interpolation functions:

- functions that generate spontaneous values during execution, according to the user's instructions – for example, there is a random integer generation function where the user must at least provide a range of values for the intended result:

  ```
  id: '{{objectId()}}',
  int: '{{integer(50,100)}}',
  random: '{{random(23, "hello", [1,2,3], true)}}'
  ```

- functions that return random values from a group of datasets incorporated in the application behind an API, where each dataset has information of a given category, for example names and political parties.

  ```
  name: '{{fullName()}}',
  party: '{{political_party()}}'
  ```

These interpolation functions may also be interwoven together and with normal strings to generate more structured strings, such as addresses. Some of them take arguments, in which case the user can either manually introduce the value or reference another property defined above, through a variable **this**, allowing them to establish relations between properties.

```
parish: '{{pt_parish()}}',
district: '{{pt_district("parish", this.parish)}}',
address: 'St. {{fullName()}}, {{pt_city("district", this.district)}}'
```

In regard to the API of datasets, the team did an extensive search for useful datasets, used the well-structured ones it found and salvaged whatever information it could from others that were maybe less organized, processing this information to remove errors and normalize its content, before joining it with other data of the same topic to create bigger, more complete datasets for the user to use.

The team also created some original datasets by hand, for topics deemed appropriate, and manually introduced bilingual support, for both portuguese and english, in all datasets made available in the application, in order to let the user choose whichever language suits best their goal. To indicate their language of choice, the user's model must start with the following syntax:

```
<!LANGUAGE pt> or <!LANGUAGE en>
```

Currently, DataGen has support datasets of all the following categories: actors, animals, brands, buzzwords, capitals, cities, car brands, continents, countries, cultural landmarks, governmental entities, hackers, job titles, months, musicians, names, nationalities, political parties, portuguese businessmen, portuguese districts, cities, counties and parishes, portuguese public entities, portuguese politicians, portuguese public figures, portuguese top 100 celebrities, religions, soccer clubs, soccer players, sports, top 100 celebrities, weekdays, writers.

The grammar also makes available a feature named **unique()**, to which the user may provide an interpolation function, or a string interpolated with such a function, as an argument. **unique** guarantees that the interpolation functions on which it is applied always return unique values. This is especially relevant when it comes to interpolation functions that fetch random data from the support datasets inside a **repeat** statement, as there is no guarantee that there won't be duplicates among the fetched records and the user might not want that.

As such, **unique** only has any effect when applied on dataset interpolation functions or with **random** (which can be observed in one of the examples from last page). As long as it's one of those (possibly interpolated with normal strings) and there are sufficient distinct entries for the entire **repeat** statement, this tool guarantees that all objects in the resulting dataset will have a different value in the property in question. If the user uses a string with more than one interpolation function, there is also no effect – there may be repeated combinations in the end.

Below are two examples: the first one depicts the correct usage of the **unique** feature; the second shows instances of a wrong approach (not enough distinct values for the repeat statement; not one of the supported interpolation functions; more than one interpolation function) that will either not work or not assure any mutually exclusive guarantee for the resulting values:

```
[ 'repeat(6)': {
  continent: unique('{{continent()}}'),
  country: unique('Country: {{country()}}'),
  random: unique('{{random(1,2,3,4,5,6)}}')
} ]
```

```
[ 'repeat(10)': {
  continent: unique('{{continent()}}'),
  int: unique('{{integer(5,20)}}'),
  random: unique('{{firstName()}} {{surname()}}')
} ]
```

Back to the properties of the model, the user may also use JavaScript functions to define their value. There are two types of functions: signed functions, where the name of the method corresponds to the key of the property, while the result of the body of the function translates to its value, and anonymous arrow functions, which are used to indicate solely the value of a property (the key needs to be precised separately beforehand).

```
name: "Oliver",
email(gen) {
  var i = gen.integer(1,30);
  return `${this.name}.${gen.surname()}${i}@gmail.com`.toLowerCase();
},
probability: gen => { return Math.random() * 100; }
```

Inside these functions, the user is free to write JavaScript code that will be later executed to determine the value of the property. This way, more complex algorithms may be incorporated into the construction logic of the dataset, allowing for a more nuanced and versatile generation tool. Given that the user has access to the whole Javascript syntax, they may make use of relational and logical operators to elaborate conditions on the intended data, as well as functional methods (for example "map" or "filter", which Javascript implements).

Inside these blocks of code, the user has full access to any property declared above in the DSL model, through the variable **this**, as well as any interpolation function available in the parser, through a **gen** variable – whenever using a function, the user must declare this argument in its signature, which they may then use inside to run said interpolation functions. All of this can be observed in the example above.

The grammar also allows fuzzy generation of properties, i.e. constraining the existence of certain properties based on logical conditions or probabilities. As of now, the grammar has four different tools for this purpose:

- **missing/having** statements – as an argument, they receive the probability of the properties inside (not) existing in the final dataset; this probability is calculated for each element, originating a dataset where some elements have said properties and others don't;

  ```
  missing(50) { prop1: 1, prop2: 2 },
  having(80) { prop3: 3 }
  ```

- **if... else if... else** statements – these work just as in any programming language: the user can use relational operators and other conditional statements to create conditions and string multiple of them together with the help of logical operators. The final object will have the properties specified in the first condition that evaluates to true (or eventually none of them, if all conditions prove to be false). In these conditions, similar to the functions, the user has unrestricted access to all properties declared above in the DSL model, as well as the interpolation functions, which gives them the ability to causally relate different properties;

```
type: '{{random("A","B","C")}}',
if (this.type == "A") { A: "type is A" }
else if (this.type == "B") { B: "type is B" }
else { C: "type is C" }
```

- the **or** statement – the grammar makes available this logical operator for quick prototyping of mutually exclusive properties, where only one will be selected for each object (note that it doesn't make sense to create an **and** statement, since that translates to simply listing the wanted properties normally in the DSL model);

```
or() {
  prop1: 1,
  prop2: 2,
  prop3: 3
}
```

- the **at_least** statement – inside this, the user writes a set of properties and gives the statement an integer argument specifying the minimum number of those properties that must be present in the final object. The parser selects that number of properties randomly from the given set.

```
at_least(2) {
  prop1: 1,
  prop2: 2,
  prop3: 3
}
```

Finally, the grammar also provides an implementation of the fundamental functional programming features – **map**, **filter** and **reduce**. The user may chain together one or several of these functions with an array value (from any of the various array-creating features made available). Shorthand syntax is not allowed, so curly braces must always be opened for the code block. Aside from that, these features work exactly like the native Javascript implementations: the user may either declare the function inside or use anonymous syntax for the variables; they may declare only the current value or any of the additional, albeit less commonly used, variables. Examples of several possible usages of these features can be observed below:

```
map: range(5).map(value => { return value+1 }),
filter: [0,1,2].filter(function(value, index) {return [0,1,2][index]>0}),
reduce: range(5).reduce((accum, value, index, array) => {
                                          return accum + array[index] }),
combined: range(5).map((value) => { return value+3 })
                  .filter(x => { return x >=  5})
                  .map(x => { return x*2 }).reduce((a,c) => {return a+c})
```

3.3 Interoperability

After processing the model, the parser generates an intermediary data structure with the final dataset, which can then be translated to either JSON or XML, according to the user's preference. The parser also generates another data structure with the corresponding Strapi model, for possible later integration in its RESTful API.

Note that the application's purpose is to create datasets according to the user's instructions, in either JSON or XML. Although the model specification may involve Javascript code, under the form of functions or conditions, as explained in the previous subsection, this does not correlate to the target application whatsoever. DataGen merely generates test datasets – it can be used for any kind of application that accepts data in JSON/XML format, whether it be an application written in Javascript, Python, C++ or some other language.

3.3.1 Client-Side Generation

This project was developed with the intent to be a web application, more specifically a website with user-friendly features. A server-sided approach would imply parsing the DSL models on the back-end server, which wouldn't be sensible as the created PEG.js parser doesn't require access to any private services (i.e. databases) hidden by the back-end itself.

Therefore, a client-sided approach makes the most sense for this application in particular, freeing resources (mainly the CPU and memory modules) from the back-end server and shifting the computation of the generated dataset to the client in their browser, using the back-end as an API server.

There are many frameworks aimed at client-sided browser experiences, however, it was decided that Vue.js would be the most adequate for this application. It allows for reactive two-way data binding – connection between model data updates and the view (UI) which creates a more user-friendly experience, since it shows changes on the DOM as they occur, instead of forcing a reload on the page (as in the traditional served-sided approach). Other reasons such as flexibility – on the components' styling and scripted behaviour – and performance – it's more efficient than React and Angular – were also a deciding factor on picking this specific framework.

After deciding which framework to use, the team started developing the interface itself, which currently has the following features:

- Authentication – it uses JWT (JSON Web Tokens) that are sent in the "Authorization" header on every HTTP request that needs to be validated by the back-end (i.e. accesses restricted user data);
- Generation and download of the dataset and/or its API – as previously mentioned, it uses a PEG.js parser for converting the DSL into JSON or XML, as well as Strapi for the API (which will be explained in Section 3.5);
- Saving DSL models – authenticated users may save their created models and optionally make them available for others to use, being able to do CRUD operations on those they own;
- Documentation – since the DSL has a defined structure, it is essential that the user has access to these guidelines at any time;
- Team description – the user may want to contact the team directly so there is a dedicated page for them to find all of this information and a brief description of the team.

The front-end needs to access persistent information such as user data and saved models which is accessible through the back-end's RESTful API, viewed in more detail in Section 3.4.

3.4 Back-End

The application needs a back-end server for multiple purposes, namely storing data, authenticating users and generating the API for the datasets.

None of the above require intensive computational power for each request sent by the user, which was one of the main reasons why the team chose a Node.js server – it is built to deal with any incoming request that is not CPU intensive due to its single-threaded, event-driven, non-blocking IO model – and because it is scalable, has good performance in terms of speed, and has a wide selection of packages (available on the **npm** registry).

For storing all the data that needs to be persistent, the back-end server accesses a MongoDB server instance, which was chosen due to its scalability (the data is not coupled relationally, which means that each document may be in a different node instance without any conflicts since they are self-contained) and direct compatibility with Node.js since they both accept JSON documents.

Currently the application uses three collections on the MongoDB database:

- **users** – stores all user specific data, which is their name, email, password (hashed), and the dates of register and most recent access;
- **models** – stores all DSL models and whom (user) they belong to, their visibility (public or private), title, description and register date;
- **blacklist** – stores users' JWT tokens after they log-out and their expiry date so that they are automatically removed from the database after they expire.

Authenticating the user allows them to access their saved DSL models and doing CRUD operations on them. Due to its Node.js integration, a JWT (JSON Web Token) approach was the chosen strategy to authenticate a user – after they submit their credentials, the system compares them to the ones saved on the database and if they match, they are given the token in the HTTP response which they need to use for any further request that accesses critical information for that same user – which expire after a short time for precaution and safety. After they log-out, the same token is added to a blacklist to provide extra security since it does not allow for a user that got access to another user's JWT (if they log-out before the short expiration date) to submit requests signed with it.

Generating the API is a more complex process and it has a dedicated Section (3.5) which explains the steps that are followed in order to obtain a fully functional REST API for any generated dataset.

3.5 Strapi API

DataGen also provides another important functionality which is generating a data API from the dataset previously created. It's a useful feature since a lot of users may want to perform CRUD operations on the data they requested or even utilize the API themselves for further work.

The tool chosen to create this API was Strapi [20], one of the best content management systems currently. Strapi automatically generates a REST API and allows multiple APIs to run simultaneously. It's also very simple to configure and it supports different database systems like PostgreSQL, MySQL, SQLite and MongoDB, being that the latter was the one used in this project. JSON-server was also considered as a tool but lacked scalability, as it only allows a single API to run at a time, which wouldn't be ideal at all. However, Strapi also had its own challenges, like the difficult way in which it stores structured data (an array, for example) and how data is imported. All of these obstacles were surpassed and will be explained further in the paper.

The process of building the API begins within the grammar section of the project, since the Strapi model is written in a recursive way, at the same time the dataset itself is being built. This strategy was a big time save in terms of the program's execution. For example, anytime an array is encountered, because Strapi doesn't have its own type to represent it, a component is created with the array's elements and a reference to that component is written in the model itself. This recursive process keeps on going with this same logic until it reaches the root, which corresponds to the collection.

After the model is created, this data is redirected to an auxiliary application that processes it and rearranges the data to be in the usual Strapi format. The data consists in the finished model and also an array filled with all the components created. At this point, the user can download a zipped version of the API model, if they so intend, and easily run it on their personal device.

Furthermore, DataGen populates the newly created API with the generated dataset through a number of POST operations. Because of Strapi's lack of methods of importing whole files as data, this cycle of POST requests was the solution found to provide a temporary populated API REST, with all the standard HTTP methods functional.

4 Results

One of the priorities during development was to test DataGen with real cases from early on, in order to not only validate the concept and its operability, but also to observe what kind of restrictions and requests were more frequent in the creation of test datasets, as a means to gather reliable external feedback on useful capabilities that should be implemented.

The team made contact with other parties and received requests to create test datasets for real systems, using DataGen, which involved the usage of complicated generation logic with many restrictions. These opportunities helped further DataGen's growth, as new ideas arised from the analysis of the requirements and were brought to life in the application, as well as proved the application's credibility, given that the results obtained were adequate and very positive.

In the interest of keeping this paper concise, it will be shown only the most important parts of one of the most complex of these application cases, that of elimination records [7].

Elimination records are a structure that must be created and carefully filled out in order to safely eliminate documentation that reaches the end of its administrative conservation deadline. This is an increasingly important tool nowadays, since most public information has shifted to being stored in digital format and the correct method for storing such data is often not followed, which increases the risk of long-term information loss. In order to correct this, the deletion of outdated documentation is just as important as the storage of new one.

The generation of these documents implies a complex logic, with many properties that directly relate between themselves according to their values and others whose value must belong to a very rigid group of possibilities. Each record has a legitimation source, whose type can take one of five different string values. According to the record's source type, its funds (public entities) vary from a single entity, in some cases, to an array of several:

```
legitimationSource: {
  type: '{{random("PGD/LC", "TS/LC", "PGD", "RADA", "RADA/CLAV")}}',
  funds(gen) {
    if (["PGD/LC","TS/LC","PGD"].includes(this.legitimationSource.type))
      return [gen.pt_entity()]
    else {
      var arr = [], i
      for (i=0; i < gen.integer(1,5); i++) arr.push(gen.pt_entity())
      return arr
    }
  }
```

Moving on, each record has an array of classes. In case the legitimation source's type is "PGD/LC" or "TS/LC", each class has a code; else, it has either a code, a reference or both. The class code itself can be composed by 3 or 4 levels, given that each level follows its own categorization:

```
classes: [ 'repeat(2,5)': {
  if (["PGD/LC","TS/LC"].includes(this.legitimationSource.type)) {
    code(gen) {
      var level1 = (...) //abbreviated
      var level2 = gen.random(10,20,30,40,50)
      var level3 = gen.integer(1,999,3)
      var level4 = gen.random("01","02")

      var class = level1 + '.' + level2 + '.' + level3
      if (Math.random() > 0.5) class += '.' + level4
      return class
    }
 }
  else {
    at_least(1) {
      code(gen) { (...) //equal to the function above },
      reference: '{{random(1,2,3,55,56)}}'
    }
  }
} ]
```

There are also year properties that must belong to the last 100 years:

```
yearStart: '{{integer(1921,2021)}}',
yearEnd(gen) {
  var year = gen.integer(1921,2021)
  while (year < this.yearStart) year = gen.integer(1921,2021)
  return year
}
```

Finally, there are two related fields, number of aggregations and the corresponding list, where the size of the list must correspond to the number indicated:

```
numberAggregations: '{{integer(1,50)}}',
aggregations: [ 'repeat(this.numberAggregations)': {
  code: '{{pt_entity_abbr()}} - {{integer(1,200)}}',
  title: '{{lorem(3,"words")}}',
  year: '{{integer(1921,2021)}}',
  if (["PGD/LC","TS/LC"].includes(this.legitimationSource.type)) {
      interventionNature: '{{random("PARTICIPANT","OWNER")}}'
  }
} ]
```

5 Conclusion

Along the paper it was discussed the development of a multilingual data generator, with built-in REST API integration. The intent behind this project was to create a versatile and powerful tool that would allow for quick prototyping and testing of software applications, a very important and common subject that seems to go surprisingly unnoticed, despite its vast relevance.

Be it either small-scale projects of university students or big, complex company software, every application should be thoroughly tested along its development, which requires the leading team to create realistic data in order to populate the system. Even today, this process is very poorly optimized, which often leads either to very time-consuming manual generation or, even worse, to a scarce and inneficient testing of the system, with few records, possibly leading to wrongful conclusions, unnoticed bugs and dangerous bottlenecks.

As such, DataGen emerges as a quick and easy to use application that allows the user to swiftly prototype a data model according to their use cases and put their system to practice with a newly-generated dataset, with accurate and realistic values, automating the generation process and facilitating the user's role in it, ultimately enhancing the user's experience and allowing more time and resources to go towards the project itself.

DataGen was thoroughly experimented with real-life cases and proved to be capable of creating complex and sizeable datasets for third party applications. The product will very soon be put in a production environment and made available for the general public, after a laborious and successful development phase.

5.1 Future work

This platform will be open-source and its contents will be uploaded to GitHub. The next step for the application itself is putting it in a production environment, to be made available for anyone that may want to use it.

As for the grammar, the team intends to develop an user-friendly personalized grammar checker that analyses the user's DSL model and, in the presence of errors, communicates what they are and how to fix them, in a simple and clear way, in order to enhance the user's experience and make the application easier to use.

Extensive documentation on all the functionalities provided is also under development, along with examples on how to use them, in order to guide the user since the application uses a DSL. Without it, the user may be overwhelmed by the amount of features they must learn by themselves.

References

1 D.b. statistical disclosure limitation, 1993.

2 General data protection regulation, 2018. URL: `https://gdpr-info.eu/`.

3 Artificial intelligence in health care: Benefits and challenges of machine learning in drug development (staa)-policy briefs & reports-epta network, 2020. URL: `https://eptanetwork.org/database/policy-briefs-reports/1898-artificial-intelligence-in-health-care-benefits-and-challenges-of-machine-learning-in-drug-development-staa`.

4 Yahya Al-Hadhrami and Farookh Khadeer Hussain. Real time dataset generation framework for intrusion detection systems in iot. *Future Generation Computer Systems*, 108:414–423, 2020. `doi:10.1016/j.future.2020.02.051`.

5 Anat Reiner Benaim, Ronit Almog, Yuri Gorelik, Irit Hochberg, Laila Nassar, Tanya Mashiach, Mogher Khamaisi, Yael Lurie, Zaher S Azzam, Johad Khoury, Daniel Kurnik, and Rafael Beyar. Analyzing medical research results based on synthetic data and their relation to real data results: Systematic comparison from five observational studies. *JMIR Med Inform*, 2015.

6 Maurilio Di Cicco, Ciro Potena, Giorgio Grisetti, and Alberto Pretto. Automatic model based dataset generation for fast and accurate crop and weeds detection. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5188–5195, 2017. doi:10.1109/IROS.2017.8206408.

7 Elimination records. https://clav.dglab.gov.pt/autosEliminacaoInfo/. Accessed: 2020-05-02.

8 Bryan Ford. Parsing Expression Grammars: A Recognition-Based Syntactic Foundation. In *Proceedings of the 31st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, 2004. Accessed: 2021-04-20. URL: https://bford.info/pub/lang/peg.pdf.

9 Georgios Gousios. The ghtorent dataset and tool suite. In *2013 10th Working Conference on Mining Software Repositories (MSR)*, pages 233–236, 2013. doi:10.1109/MSR.2013.6624034.

10 Bill Howe, Julia Stoyanovich, Haoyue Ping, Bernease Herman, and Matt Gee. Synthetic data for social good, 2017.

11 JSON Generator. https://next.json-generator.com/4kaddUyG9/. Accessed: 2020-05-04.

12 Xiangjie Kong, Feng Xia, Zhaolong Ning, Azizur Rahim, Yinqiong Cai, Zhiqiang Gao, and Jianhua Ma. Mobility dataset generation for vehicular social networks based on floating car data. *IEEE Transactions on Vehicular Technology*, 67(5):3874–3886, 2018. doi:10.1109/TVT.2017.2788441.

13 Menno Mostert, Annelien L Bredenoord, Monique Biesaart, and Johannes Delden. Big data in medical research and eu data protection law: Challenges to the consent or anonymise approach. *Eur J Hum Genet.*, 24(7):956–60, 2016. doi:10.1038/ejhg.2015.239.

14 PegJS. https://pegjs.org/. Accessed: 2021-04-20.

15 Haoyue Ping, Julia Stoyanovich, and Bill Howe. Datasynthetizer: Privacy-preserving synthetic datasets. In *Proceedings of SSDBM '17*, 2017. doi:10.1145/3085504.3091117.

16 Darijo Raca, Dylan Leahy, Cormac J. Sreenan, and Jason J. Quinlan. Beyond throughput, the next generation: A 5g dataset with channel and context metrics. In *Proceedings of the 11th ACM Multimedia Systems Conference*, MMSys '20, page 303–308, New York, NY, USA, 2020. Association for Computing Machinery. doi:10.1145/3339825.3394938.

17 Debbie Rankin, Michaela Black, Raymond Bond, Jonathan Wallace, Maurice Mulvenna, and Gorka Epelde. Reliability of supervised machine learning using synthetic data in health care: Model to preserve privacy for data sharing. *JMIR Med Inform.*, 2020.

18 Anat Reiner Benaim, Ronit Almog, Yuri Gorelik, Irit Hochberg, Laila Nassar, Tanya Mashiach, Mogher Khamaisi, Yael Lurie, Zaher S Azzam, Johad Khoury, Daniel Kurnik, and Rafael Beyar. Analyzing medical research results based on synthetic data and their relation to real data results: Systematic comparison from five observational studies. *JMIR Med Inform*, 8(2):e16492, Feb 2020. doi:10.2196/16492.

19 Regulamento nacional de interoperabilidade digital (RNID). https://dre.pt/application/file/a/114461891. Accessed: 2020-04-21.

20 Design APIs fast, manage content easily. https://strapi.io/. Accessed: 2020-04-21.

MUAHAH: Taking the Most out of Simple Conversational Agents

Leonor Llansol
INESC-ID & Instituto Superior Técnico, Porto, Portugal

João Santos
INESC-ID & Instituto Superior Técnico, Porto, Portugal

Luís Duarte
CISUC, DEI, Universidade de Coimbra, Portugal

José Santos
CISUC, DEI, Universidade de Coimbra, Portugal

Mariana Gaspar
INESC-ID & Instituto Superior Técnico, Porto, Portugal

Ana Alves
CISUC & Instituto Politécnico de Coimbra, Portugal

Hugo Gonçalo Oliveira
CISUC, DEI, Universidade de Coimbra, Portugal

Luísa Coheur[a]
INESC-ID & Instituto Superior Técnico, Porto, Portugal

[a] Corresponding author

Abstract

Dialog engines based on multi-agent architectures usually select a single agent, deemed to be the most suitable for a given scenario or for responding to a specific request, and disregard the answers from all of the other available agents. In this work, we present a multi-agent plug-and-play architecture that: (i) enables the integration of different agents; (ii) includes a decision maker module, responsible for selecting a suitable answer out of the responses of different agents. As usual, a single agent can be chosen to provide the final answer, but the latter can also be obtained from the responses of several agents, according to a voting scheme. We also describe three case studies in which we test several agents and decision making strategies; and show how new agents and a new decision strategy can be easily plugged in and take advantage of this platform in different ways. Experimentation also confirms that considering several agents contributes to better responses.

2012 ACM Subject Classification Computing methodologies → Natural language processing; Computing methodologies → Multi-agent systems; Computing methodologies → Artificial intelligence; Information systems → Information retrieval

Keywords and phrases Dialog systems, question answering, information retrieval, multi-agent

Digital Object Identifier 10.4230/OASIcs.SLATE.2021.7

Supplementary Material *Software (Source Code)*: `https://github.com/leonorllansol/muahah`
Software (Source Code): `https://github.com/NLP-CISUC/muahah`

Funding This work was partially supported by: the project Flowance (POCI-01-0247-FEDER-047022), co-financed by the European Regional Development Fund (FEDER), through Portugal 2020 (PT2020), and by the Competitiveness and Internationalization Operational Programme (COMPETE 2020); the demonstration project AIA, "Apoio Inteligente a empreendedores (chatbots)", funded by the Fundação para a Ciência e Tecnologia (FCT), through the INCoDe 2030 initiative; and by national funds through FCT, within the scope of the project CISUC (UID/CEC/00326/2020) and by European Social Fund, through the Regional Operational Program Centro 2020.

© Leonor Llansol, João Santos, Luís Duarte, José Santos, Mariana Gaspar, Ana Alves, Hugo Gonçalo Oliveira, and Luísa Coheur;
licensed under Creative Commons License CC-BY 4.0

1 Introduction

Recent technology advances have brought many frameworks for building conversational agents. Examples are Rasa[1] or Dialogflow[2] and they "are often designed with the tacit assumption that at any time, there is only one agent and one human" [11]. Behind such systems, there is usually a set of modules, specialized in different topics or types of dialog. Thus, each system is developed with a distinct task in mind, and, after understanding the user intentions, the most appropriate module is triggered, while the remaining are set aside. Yet, we can be faced with a scenario where (i) a new agent has to cover different domains or types of dialog, for which there is no training data nor resources for manually defining relevant entities and user intents; (ii) a set of independent agents is available, but they are only expert on more specific domains, possibly overlapping, or follow different techniques, and, again, there is no data to train a model for deciding which agent provides the answer to each user request.

In this paper, we tackle the aforementioned issue, and assume that *all agents can potentially answer all questions*. With this in mind, we implemented a new platform, in Python: the Multi-Agents Hand-in-Hand platform (dubbed MUAHAH), enables the integration of distinct agents, expert on different domains and/or with different levels of complexity, as well as decision strategies, possibly taking all the agents' answers into account. We present MUAHAH and evaluate it in three case studies. In the first two, we consider: (i) a panoply of retrieval-based agents, specifically, agents that get their answers from a given knowledge base with trigger/answer pairs, in which trigger is a user request (e.g., a question) and answer is the response to that request; (ii) two decision strategies, namely *Simple Majority*, which implements a majority vote between all the agents and picks the most voted answer, and *Priority System*, which prioritizes the answer of an agent over the others. Here, combining different agents indeed leads to more plausible answers for complex and out-of-domain requests. In the third case study, we integrated new agents in MUAHAH, based on retrieval and embedding methods, as well as a new decision strategy, *Borda Count*, to confirm that one can easily rely on MUAHAH for developing new dialog systems, and that combining ranks of answers by different agents leads to more accurate systems. Still, it should be clear that, although we have implemented several agents, we are not proposing a new way of creating them, but a plug-and-play framework to integrate previously created agents. In addition, we are not claiming a particular way of taking advantage of these agents, but, again, a framework that enables to customise how the responses of several agents are combined.

The paper is organized as follows: Section 2 briefly covers related work on chatbot development and architecture; Section 3 describes MUAHAH, the proposed platform; Sections 4, 5 and 6 describe the three case studies; finally, Section 7 discusses the main conclusions and directions for future work.

2 Related Work

Since the early rule-based systems such as the famous ELIZA [24] to the recent end-to-end dialog systems [25, 26, 22], conversational agents can be found in many different flavours. Also, some agents are task-oriented (e.g., Max [18] was a guide in the Heinz Nixdorf Museums Forum), while others target to engage in general conversations (e.g., SSS [1] is a retrieval-

[1] `https://rasa.com`
[2] `https://dialogflow.com`

based agent with a knowledge base of subtitles). As previously stated, we do not intend to contribute with an architecture for building agents, but with a platform that allows their integration.

We define *agent* as any piece of software that, upon receiving a user request, delivers one or more responses. In many systems, these agents are the modules that retrieve an answer about a specific topic. For instance, Gunrock [3], the winner of the 2018 Amazon Alexa Prize, maps user's input into one of 11 possible topic dialog modules (e.g., movies, books and animals). Here, we do not assume that an agent is specialized in a certain topic, but that we want a simple way to put all these systems together. A similar framework is MACA [23] that touches a number of common points with our work. However, MACA proposes several integrating processes, as, for instance, the re-usability of slots across different tasks. Here, even though the same knowledge sources can be shared between agents, each agent is independent of each other. Also, our agents are *slaves* of a coordinator, as there is no interaction between them.

An important module common to many chatbot architectures is the Dialog Manager, responsible for keeping track of the dialog state and using it to select a response [14]. Gunrock has a Dialog Manager that classifies the user's intent, and, based on this intent, selects the most appropriate module to forward the user request. Sounding Board [8], the winner of the 2017 Amazon Alexa Prize, uses a Dialog Manager that contains a set of "miniskills" for handling different topics, where only one miniskill is selected per turn, based in the topic of the user request. Both the aforementioned systems classify the intent and topic of the user request for selecting a single proper module that will handle it. NPCEditor's [14] Dialog Manager takes a list of responses and uses the dialog state to choose one of them. Here, we do not have a Dialog Manager, but: an Agent Manager, responsible for forwarding the user request to all the active agents and to send their answers to a coordinator; and a Decision Maker, that sends the agents' answers to a set of available decision methods and returns their answers to the coordinator. Unlike most Dialog Managers, these two modules are not trained.

Still on multi-agent scenarios, many other research questions are currently under study. For instance, Divekar et al. [5] target to determine which agent is being addressed, and Eisman et al. [6] train a multi-agent dialog model via reinforcement learning. In the latter scenario, agents learn with each other by interacting in natural language. Here, we assume that the agents are already trained or rely on unsupervised techniques.

3 Proposed Architecture

In this section, we describe the architecture of MUAHAH, depicted in Figure 1. The *Coordinator* is a central module that acts as an interface between the user, the *Agent Manager* and the *Decision Maker*.

The *Agent Manager* aims at providing an interaction point between MUAHAH and the provided agents. It is responsible for launching the agents, and for guaranteeing that the communication between them and the *Coordinator* is done correctly. When MUAHAH is booted up, the *Agent Manager* locates all the available agents through their configuration files and integrates an instance of each into the system. Upon receiving a request from the *Coordinator*, the *Agent Manager* sends it to the integrated agents, which provide their answers. Such answers are then sent back to the *Coordinator*, which finally sends them to the *Decision Maker*. Note that each agent is only responsible for outputting (at least) a response for the given request.

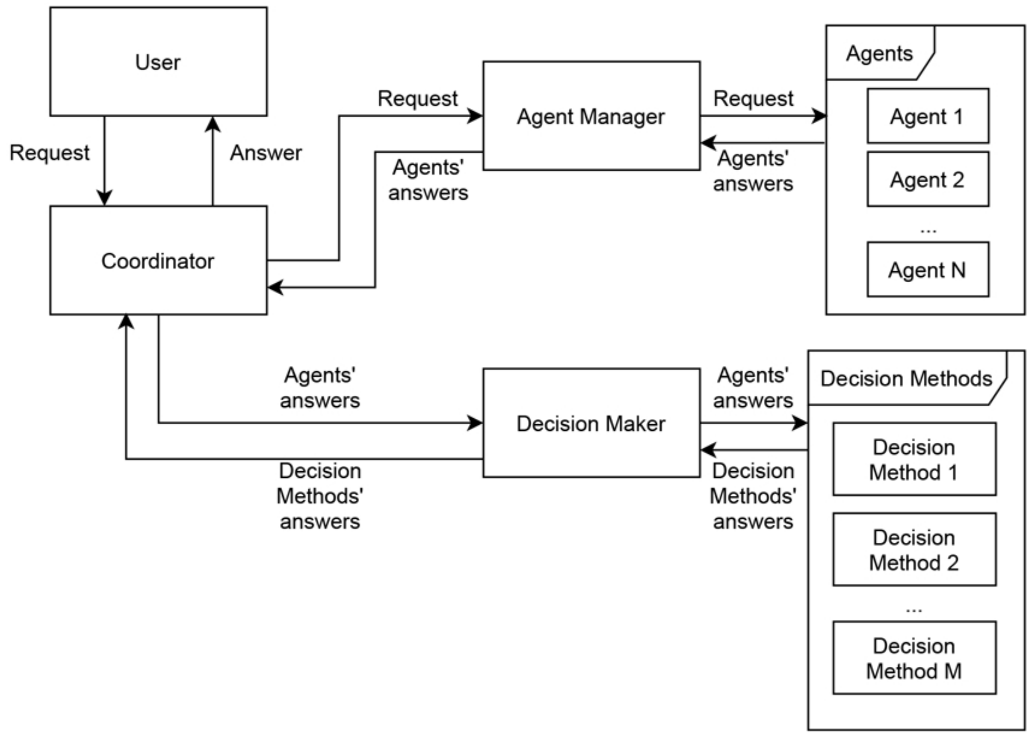


Figure 1 MUAHAH architecture.

The *Decision Maker* makes its decision based in one of the implemented decision strategies. These can consider different items, such as the answers of the agents, the agent who gave each answer, and the user query. But new decision strategies can be added to MUAHAH[3]. Each decision strategy has a weight, defined in the system's configuration file. The *Decision Maker* returns a set of answers, one given by each *Decision Method* to the *Coordinator*, which uses those weights for selecting the best answer and finally return it to the user. Originally, the *Decision Maker* includes two strategies: *Simple Majority* and *Priority System*. Yet, new methods can be integrated, as in our third case study (Section 6).

With *Simple Majority*, the *Decision Maker* will deliver the most frequent answer between the set of answers by the agents. In a scenario where a certain agent's answer would be accurate, even if it did not always deliver an answer to the given query, it would make sense for that agent to be prioritized. For that purpose, we developed the *Priority System*, a decision strategy that allows the developer to explicitly set priorities for the available agents. Therefore, when evaluating the answer each agent delivers to a given user request, MUAHAH verifies whether the prioritized agent was able to deliver a response to the user request: if so, that answer is deemed to be a plausible one and is returned to the user; if the prioritized agent is not able to answer the user request, the system considers the answers by each of the remaining agents and selects the final answer through *Simple Majority*, as described earlier.

[3] To do so, an abstract class, `Decision Method`, has an abstract method, `getAnswer`, that takes a set of answers and delivers the best, based on its heuristic. Hence, to add a new decision strategy, only a source file is needed, with the class that extends `Decision Method` and implements `getAnswer`. It is also necessary to add the new strategy's name and its weight to the system's configuration file.

Finally, in order to build a new agent for MUAHAH, the developer needs to customize a configuration file, which allows the agent to be called by the *Agent Manager*; it also allows the developer to set configurable parameters without directly interacting with the source files. In addition, the agent's source files must contain a class that extends the abstract class (`Agent`) and implements its own `requestAnswer` method, which takes a user request and returns a list of answers. Finally, since an agent can be active or inactive, it is also necessary to edit the system's configuration file, adding the name of the agent and whether it is active or not.

For a better understanding, we refer to the working examples in Sections 4 and 5.

4 Case study A: using the Simple Majority

We first evaluate the Simple Majority strategy in MUAHAH. Here, the Subtle corpus [2] was used as the knowledge base of nine agents. Subtle[4] contains about 3 million interactions from movie subtitles, i.e., trigger/answer pairs like "Trigger: I like the sea., Answer: me too". Due to the large number of included interactions, Subtle was indexed in a search engine, thus enabling that only a small set of interactions is retrieved as response candidates. In this case, 20 interactions are retrieved from Lucene[5]. Then, our agents compare the user request both with the trigger and with the answer, as both might be useful for finding an adequate answer. The difference between the agents is the algorithm for computing sentence similarity, always in the 0 to 1 range, which is either *Cosine* [20], *Jaccard* [13] or *Edit Distance* [15], and the weight given to the similarity between requests, triggers and their response. For such, three different agents, that extend the abstract class `Agent`, are added to our system: `Cosine-Agent`, `Jaccard-Agent` and `EditDistance-Agent`, each implementing the *requestAnswer* method according to their algorithm. Moreover, three agents were created of each type, with different similarity weights. Table 1 summarises the nine agents. For instance, agent C2 uses the cosine similarity and gives more weight to the similarity between the user request and the trigger than between the user request and the answer, in a relation of 75 to 25.

Table 1 Agent distribution according to the similarity computation and weights.

RequestSim/ResponseSim	**50/50**	**75/25**	**100/0**
Cosine	C1	C2	C3
Edit Distance	E1	E2	E3
Jaccard	J1	J2	J3

For each agent, sentences were lower-cased. Then, for computing the *Cosine Similarity*, punctuation was removed, stopwords were kept, and words were not stemmed. For the *Edit Distance*, both punctuation and stopwords were kept. Finally, for the *Jaccard Similarity*, both punctuation and stopwords were removed.

To test this configuration, we created a set of 100 simple questions (e.g., "What's your name?", "How are you?") and a set of 100 more complex questions (e.g., "What's your opinion on Linux-based platforms?", "If the world ended tomorrow, what would you do today?"). Then, we run both the resulting system and compared it to Say Something Smart (SSS) [16],

[4] A smaller version was used to reduce index creation time and is available from `https://github.com/leonorllansol/muahah/blob/master/corpora/1million.txt`

[5] `https://lucene.apache.org`

which has a single retrieval agent. For this, four annotators were given a sample of 25 simple and 25 complex questions with the answers by both systems, randomly selected, and asked to score each of the responses between 1 and 4, based on whether the answer:

4: Was plausible without additional context;
3: Was plausible in a specific context, or did not actively answer the query but maintained its context;
2: Actively changed the context (e.g., an answer that delivers a question to the user which does not match the initial query's topic), or contained Structural issues (even if its content fits in the context of the question);
1: Had no plausibility value.

For illustrative purposes, one response of each kind for the question "What is the price of olive oil in 7-Eleven?" would be:

4: "In 7-Eleven, the olive oil costs 3.00€."
3: "He knows what is the price."
2: "Olive oil."
1: "Eleven."

The mean and mode of all answer scores were computed for each system. Also, following the evaluation of other engines, such as AliMe [19], we considered that, to be discerned as acceptable, a given answer would need an average score of at least 2.75 between the four annotations (corresponding, e.g., to the case where three annotators score it 3 and the last one 2). Table 2 summarizes the obtained results.

Table 2 SSS system against our system when answering basic questions.

	Simple questions			**Complex questions**		
	Mean Score	**Answers $\geq$ 2.75**	**Approval**	**Mean Score**	**Answers $\geq$ 2.75**	**Approval**
SSS	2.68	23 / 50	46%	2.26	11 / 50	22%
MUAHAH	2.6	24 / 50	48%	2.6	22 / 50	44%

This instance of MUAHAH keeps up with SSS for basic questions and outperforms it with complex questions. We should add that, in what concerns simple questions, scores assigned to SSS were more polarized, with a special focus on scores of 2 and 4 to its answers, while scores of MUAHAH answers were more evenly distributed. With complex questions, 2 was the most common score for both systems, with a stronger preponderance for SSS to deliver implausible answers. This can be explained by the fact that SSS relies on a single agent to decide on all its answers, while the multi-agent system considers what possible answers are more common, from multiple points of view.

To better understand the use of the *Simple Majority* strategy, we present a running example, using the aforementioned agents, `Cosine-Agent`, `Jaccard-Agent` and `EditDistance-Agent`, and the Subtle corpus.

1. The user poses the query q to the system: "Quando começa o Verão?" (When does summer start?)
2. The *Coordinator* forwards q to the *Agent Manager*.
3. The *Agent Manager* instantiates the active agents: `Cosine-Agent`, `Jaccard-Agent` and `EditDistance-Agent`, and forwards q to them.
4. Each agent selects a response, based on the sentence similarity between the request and the retrieved candidates. Their selected responses are returned to the *Agent Manager*:

- A_{Cos}: "Assim que o senhor e a chuva desaparecerem." (As soon as you and the rain disappear.)
- A_{Jac}: "Assim que o senhor e a chuva desaparecerem." (As soon as you and the rain disappear.)
- A_{ED}: "Não te ouvimos." (We can't hear you.)

5. The *Agent Manager* sends the agents' answers to the *Coordinator*.
6. The *Coordinator* forwards the agents' answers, [A_{Cos}, A_{Jac}, A_{ED}], to the *Decision Maker*.
7. The *Decision Maker* sends the agents' answers to the active *Decision Method*: *Simple Majority*, with a weight of 1.
8. *Simple Majority* returns the more common answer, A_{Cos} and A_{Jac}, to the *Decision Maker*.
9. The *Decision Maker* returns the method's answer to the *Coordinator*.
10. The *Coordinator* returns the answer to the user: "Assim que o senhor e a chuva desaparecerem." (As soon as you and the rain disappear.)

5 Case study B: Priority System

The *Priority System* is a decision strategy that gives priority to a certain agent if that agent can deliver a response. To test this strategy, we use Edgar Smith [9], another retrieval-based conversational agent, this time on a specific domain. Edgar answers questions about the Monserrate palace, in Sintra, based on a corpus[6] of 1,179 interactions. Besides question/answer pairs about the palace, Edgar answers some out-of-domain (chit-chat) questions such as "What's your name?" or "How old are you?". However, Edgar will not be able to answer most out-of-domain questions. This is a significant issue, as users tend to get more engaged with conversational platforms when answers to chit-chat queries are delivered.

To address the aforementioned issue, we set up MUAHAH to use Edgar as a Priority Agent, with the nine agents described in case study A (Section 4) used as Edgar's back-up, in case Edgar is not able to provide an answer. If not backed-up, when Edgar's top-scored answer has a similarity score below a given threshold, defined in its configuration file, it says "I do not understand your question". We use again the *Jaccard Similarity* to compare the user request with the questions in Edgar's knowledge base, a threshold of 0.35, and the previous *Simple Majority* decision strategy for choosing an answer when Edgar is not able to provide one.

We created two sets of 100 questions each. The first with questions about the palace and the second with other kind of requests, such as personal and trivia questions (e.g., "What is your name?", "Do you like to sing?"). Similarly to case study A, four annotators were given a sample of 50 questions and answers from each set, and were, once again, asked to score each response between 1 and 4, according to the earlier described criteria. Table 3 summarizes the results.

As expected, in what concerns questions about the palace, Edgar achieved the best performance, but this MUAHAH configuration managed to keep up with minor accuracy costs, and both systems had most of their answers scored 4. On the other hand, regarding out-of-domain questions, MUAHAH beat Edgar by a comfortable margin.

To better understand the use of the *Priority System* strategy, we present a running example, using as prioritized agent the aforementioned Edgar, and also the remaining agents `Cosine-Agent`, `Jaccard-Agent` and `EditDistance-Agent` as backup. Note that Edgar has its own corpus, while the other agents use the Subtle corpus.

[6] Available from `https://github.com/leonorllansol/muahah/blob/master/corpora/edgar/edgar.txt`

Table 3 Edgar evaluated against our system.

	Questions about the palace			Out-of-domain Questions		
	Mean Score	Answers $\geq$ 2.75	Approval	Mean Score	Answers $\geq$ 2.75	Approval
Edgar	3.015	31 / 50	62%	2.37	18 / 50	36%
MUAHAH	2.87	29 / 50	58%	2.625	22 / 50	44%

1. The user poses query q to the system: "Quando foi construído o Palácio de Monserrate?" (When was the Palace of Monserrate built?)
2. The *Coordinator* forwards q to the *Agent Manager*.
3. The *Agent Manager* instantiates the active agents: Edgar, `Cosine-Agent`, `Jaccard-Agent` and `EditDistance-Agent`, and forwards q to them.
4. Each agent selects a response, based on the sentence similarity between the request and the retrieved candidates. Their selected responses are returned to the *Agent Manager*:
   - A_{Cos}: "Quando lhe telefonei a meio da noite." (When I called you in the middle of the night.)
   - A_{Edgar}: "O palácio foi construído entre 1858 e 1864, sobre a estrutura de um outro palácio." (The palace was built between 1858 and 1864, on the structure of another palace.)
   - A_{Jac}: "Meu Deus!" (My God!)
   - A_{ED}: "Quando lhe telefonei a meio da noite." (When I called you in the middle of the night.)
5. The *Agent Manager* sends the agents' answers to the *Coordinator*.
6. The *Coordinator* forwards the agents' answers, $[A_{Cos}, A_{Edgar}, A_{Jac}, A_{ED}]$, to the *Decision Maker*.
7. The *Decision Maker* sends the agents' answers to the active *Decision Method*: *Priority System*, with a weight of 1.
8. *Priority System* returns the answer given by the prioritized agent Edgar, A_{Edgar}, to the *Decision Maker*.
9. The *Decision Maker* returns the method's answer to the *Coordinator*.
10. The *Coordinator* returns the answer to the user: "O palácio foi construído entre 1858 e 1864, sobre a estrutura de um outro palácio." (The palace was built between 1858 and 1864, on the structure of another palace.)

For queries out of Edgar's domain of expertise (e.g., "Quando começa o Verão?"), Edgar is expected to given a low similarity score to the top-ranked answer. If this score is below the set threshold, Edgar will return the default "no answer" message, which results in the application of *Simple Majority* to the remaining agents.

6 Case study C: Integrating new Agents and new Decision Strategy

The final case study was significantly different and tackled the integration of new agents and of a new decision strategy in MUAHAH. Briefly, we integrated three retrieval-based agents that, given a request, in natural language, retrieve the responses for similar requests in a knowledge base (KB). Each agent follows a different matching technique: (i) Agent-BM25 is based on the search engine Whoosh[7], used for indexing the KB and retrieving suitable candidates

[7] `https://whoosh.readthedocs.io/`

with the BM25 ranking function; (ii) Agent-W2V relies on a pre-trained word2vec [17] model for encoding the requests in a numeric vector, i.e., requests are represented by the average embedding vector of their words, and candidate requests in the KB are ranked according to their cosine similarity with the user request; (iii) Agent-BERT relies on a pre-trained BERT [4] model for encoding the requests in a numeric vector (i.e., the [CLS] embedding of the second to last layer), and ranks the candidate requests in the KB according to their cosine with the user request.

For integrating each agent, a sub-directory with the name of the agent was created under the directory managed by the Agent Manager. In this directory, a script with the directory name was added, implementing the `requestAnswer` method, where the retrieval logic resides. In this case, it has a string parameter with the request and returns a list of answers ranked according to their matching logic. In the same directory, a `config.xml` file was created with the name of the main class. Finally, in the main MUAHAH directory, the name of the agent was added to the main `config.xml` and set to active. We also inactivated all agents except the new three.

The integrated decision strategy was the Borda Count voting [7], which scores each candidate response according to their rank by different agents. This is different from the previous decision strategies, which rely exclusively on the first answer given by each agent. For this reason, Borda Count expects that the `requestAnswer` method of the agents returns a list of results. The higher the rank, the higher the score. More precisely, for each agent, the score of each candidate on a rank will be equal to the number of considered positions minus its position in the rank, with the latter starting in 0. For instance, if we consider the top-5 candidates, the first candidate gets 5 points and the fifth gets 1. The selected response will then be the one with the highest final score, obtained by summing all of its partial scores, in all the considered lists, in this case, retrieved by each search strategy.

For integrating Borda Count, a script was created with its name and added to the directory managed by the Decision Maker module. This script is based on a class that extends the `DecisionMethod` class and implements the method `getAnswer` with the logic underlying Borda Count.

To test the integration, we used the three agents for answering questions about entrepreneurship and economic activity in Portugal, for which there is a corpus of 855 Frequently Asked Questions (FAQs), in Portuguese, and their variations [10][8], i.e., paraphrases or related questions using other words, possibly omitting information, that simulate user requests. Some of the questions are quite complex, so our agents are not expected to answer every request successfully, especially considering that they rely on simple matching techniques and were not trained for the target domain. Examples of questions include "*É obrigatório declarar o exercício de uma atividade económica junto da Autoridade Tributária e Aduaneira?*" (Is it mandatory to declare the exercise of an economic activity with the Tax and Customs Authority?) or "*Qual o número máximo de crianças permitido por Ama?*" (What is the maximum number of children allowed by a nanny?), with variations like "*Como devo informar a autoridade tributária sobre a minha atividade*" and "*Quantas crianças uma ama pode acolher?*".

We note the natural language of the requests is irrelevant to MUAHAH. When necessary, specific natural language processing, namely language-specific resources and tools, have to be included in the agents. For instance, in this case study, where the corpus is in Portuguese,

[8] Available from `https://github.com/NLP-CISUC/AIA-BDE/`

Agent-W2V used a word2vec-CBOW model pre-trained for Portuguese, with 300-sized vectors available from the NILC embeddings [12] and Agent-BERT used BERTimbau [21] large, a pre-trained BERT model for Portuguese that encodes sentences in 1,024-sized vectors. Whoosh was used with no language-specific analyser.

The goal of this case study was twofold: (i) to test whether the agents, now in MUAHAH, could effectively respond to user requests, here simulated when matching variations with questions in the KB; (ii) to compare the performance of using all three agents, alone and in parallel, i.e., combined with SimpleMajority or Borda Count for selecting the most suitable question. The corpus contains variations produced by different methods, but we focused on two handcrafted types, namely: (i) VUC, 816, written by the creators of the corpus; (ii) VMT, 168, by contributors of the Mechanical Turk crowdsourcing platform. For each variation, agents ranked each of the 855 FAQs according to their suitability. Despite the availability of the agents described in Sections 4 and 5, we note that these three agents were the only effectively used for this purpose.

Since, for this case study, there were previously-created variations, simulating user requests, each one with a known answer, evaluation could be automatized. Table 4 has the accuracy of each agent and decision strategy, corresponding to the proportion of variations for which the correct question was ranked first, meaning that the correct answer would be given. Performances are modest, but they improve when the three agents are combined, either with Borda Count or SimpleMajority. This shows that, besides enabling the combination of different types of dialog or domains in the same platform (Sections 4 and 5), combining different agents for the same domain also leads to a more accurate system. The more complementary the answers of the agents are, the better, which is why we opted for significantly different retrieval techniques.

Table 4 Accuracy of integrated agents, alone and combined, when matching question variations with the correct questions.

	Agent-BM25	**Agent-W2V**	**Agent-BERT**	**SimpleMajority**	**Borda**
VMT	57.1%	65.5%	67.9%	74.4%	72.6%
VUC	55.9%	54.0%	56.7%	61.2%	63.0%

7 Conclusions and Future Work

We presented MUAHAH, a multi-agent platform for dialog systems that allows to easily integrate different agents and, in order to get suitable responses, different decision-making strategies that might consider the answers of all active agents. Such strategies are extremely useful when no data is available for training a model that decides on the best agent for each request. Instead, when targeting a specific domain or different types of dialog, they can often take the most out of an ensemble of simple agents, without requiring additional training or manual intent definition.

Some retrieval-based agents, such as those used in the case studies A and B, are already included in MUAHAH, for different purposes. The same for simple decision strategies. These include those of the first two presented case studies where, according to human users, different combinations of agents resulted in better responses. We further showed that MUAHAH is flexible enough for integrating additional agents and decision strategies, and thus adaptable to different domains. For this reason, we believe that it is a suitable alternative when developing new dialog systems. We thus make MUAHAH and its source code available to all of those interested: MUAHAH is available from `https://github.com/leonorllansol/muahah`, and its fork including the new agents and new decision strategy, resulting from case study C, is at `https://github.com/NLP-CISUC/muahah`.

A critical aspect of MUAHAH are the decision strategies, where there is plenty of work to be done. For instance, the aforementioned Gunrock [3] takes into account a fact/opinion interleaving strategy, which is certainly interesting to explore. In particular, we believe that the system's results would greatly improve if it took user feedback into account, which would enable to learn specific weights for each agent, instead of a flat priority to a single (or to multiple) agents. Towards a more natural dialog, a module for dealing with context would also improve conversations. Such a model could keep track of concepts and entities referred and remember them in future interactions. This would enable the chatbots do deal with anaphora, and could also be exploited by pro-active chatbots.

References

1 David Ameixa, Luisa Coheur, Pedro Fialho, and Paulo Quaresma. Luke, I am your father: dealing with out-of-domain requests by using movies subtitles. In *Proceedings of the 14th International Conference on Intelligent Virtual Agents (IVA'14)*, LNCS/LNAI, Boston, 2014. Springer-Verlag.

2 David Ameixa, Luísa Coheur, and Rua Alves Redol. From subtitles to human interactions: introducing the subtle corpus. Technical report, Tech. rep., INESC-ID (November 2014), 2013.

3 Chun-Yen Chen, Dian Yu, Weiming Wen, Yi Mang Yang, Jiaping Zhang, Mingyang Zhou, Kevin Jesse, Austin Chau, Antara Bhowmick, Shreenath Iyer, Giritheja Sreenivasulu, Runxiang Cheng, Ashwin Bhandare, and Zhou Yu. Gunrock: Building a human-like social bot by leveraging large scale real user data, 2018.

4 Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1*, pages 4171–4186, Minneapolis, Minnesota, 2019. ACL. `doi:10.18653/v1/N19-1423`.

5 Rahul R. Divekar, Xiangyang Mou, Lisha Chen, Maíra Gatti de Bayser, Melina Alberio Guerra, and Hui Su. Embodied conversational ai agents in a multi-modal multi-agent competitive dialogue. In *Proceedings of 28th International Joint Conference on Artificial Intelligence, IJCAI-19*, pages 6512–6514. International Joint Conferences on Artificial Intelligence Organization, 2019.

6 Eduardo M. Eisman, María Navarro, and Juan Luis Castro. A multi-agent conversational system with heterogeneous data sources access. *Expert Syst. Appl.*, 53(C):172–191, July 2016.

7 Peter Emerson. The original Borda count and partial voting. *Social Choice and Welfare*, 40(2):353–358, February 2013.

8 Hao Fang, Hao Cheng, Maarten Sap, Elizabeth Clark, Ari Holtzman, Yejin Choi, Noah A. Smith, and Mari Ostendorf. Sounding board: A user-centric and content-driven social chatbot. *arXiv preprint arXiv:1804.10202*, 2018.

9 Pedro Fialho, Luísa Coheur, Sérgio Curto, Pedro Cláudio, Ângela Costa, Alberto Abad, Hugo Meinedo, and Isabel Trancoso. Meet EDGAR, a tutoring agent at MONSERRATE. In *Proceedings of 51st Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 61–66, Sofia, Bulgaria, 2013. ACL.

10 Hugo Gonçalo Oliveira, João Ferreira, José Santos, Pedro Fialho, Ricardo Rodrigues, Luísa Coheur, and Ana Alves. AIA-BDE: A corpus of FAQs in portuguese and their variations. In *Proceedings of 12th International Conference on Language Resources and Evaluation*, LREC 2020, pages 5442–5449, Marseille, France, 2020. ELRA.

11 T. K. Harris, S. Banerjee, A. I. Rudnicky, J. Sison, K. Bodine, and A. W. Black. A research platform for multi-agent dialogue dynamics. In *RO-MAN 2004. 13th IEEE International Workshop on Robot and Human Interactive Communication (IEEE Catalog No.04TH8759)*, pages 497–502, 2004.

12 Nathan S. Hartmann, Erick R. Fonseca, Christopher D. Shulby, Marcos V. Treviso, Jéssica S. Rodrigues, and Sandra M. Aluísio. Portuguese word embeddings: Evaluating on word analogies and natural language tasks. In *Proceedings of 11th Brazilian Symposium in Information and Human Language Technology (STIL 2017)*, 2017.

13 Paul Jaccard. The Distribution of the Flora in the Alpine Zone. *New Phytologist*, 11(2):37–50, 1912.

14 Anton Leuski and David Traum. NPCEditor: A tool for building question-answering characters. In *Proceedings of 7th International Conference on Language Resources and Evaluation (LREC'10)*, Valletta, Malta, 2010. ELRA.

15 Vladimir Iosifovich Levenshtein. Binary Codes Capable of Correcting Deletions, Insertions and Reversals. *Soviet Physics Doklady*, 10:707, 1966.

16 Daniel Magarreiro, Luisa Coheur, and Francisco S. Melo. Using subtitles to deal with out-of-domain interactions. In *DialWatt – the 18th workshop on the semantics and pragmatics of dialogue*, SemDial Workshop Series, Edinburgh, 2014. Springer-Verlag.

17 Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. In *Proceedings of the Workshop track of ICLR*, 2013.

18 Thies Pfeiffer, Christian Liguda, Ipke Wachsmuth, and Stefan Stein. Living with a virtual agent: Seven years with an embodied conversational agent at the heinz nixdorf museumsforum. In *Proceedings of the International Conference Re-Thinking Technology in Museums 2011 - Emerging Experiences*, pages 121–131. thinkk creative & the University of Limerick, 2011.

19 Minghui Qiu, Feng-Lin Li, Siyu Wang, Xing Gao, Yan Chen, Weipeng Zhao, Haiqing Chen, Jun Huang, and Wei Chu. AliMe chat: A sequence to sequence and rerank based chatbot engine. In *Proceedings of 55th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 498–503, Vancouver, Canada, 2017. ACL.

20 Amit Singhal. Modern information retrieval: A brief overview. *IEEE Data Engineering Bulletin*, 24, January 2001.

21 Fábio Souza, Rodrigo Nogueira, and Roberto Lotufo. BERTimbau: Pretrained BERT models for Brazilian Portuguese. In *Proceedings of the Brazilian Conference on Intelligent Systems (BRACIS 2020)*, volume 12319 of *LNCS*, pages 403–417. Springer, 2020.

22 Zhiliang Tian, Rui Yan, Lili Mou, Yiping Song, Yansong Feng, and Dongyan Zhao. How to make context more useful? an empirical study on context-aware neural conversational models. In *Proceedings of 55th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 231–236, Vancouver, Canada, July 2017. ACL.

23 Hoai Phuoc Truong, Prasanna Parthasarathi, and Joelle Pineau. MACA: A modular architecture for conversational agents. In *Proceedings of 18th Annual SIGdial Meeting on Discourse and Dialogue*, pages 93–102, Saarbrücken, Germany, August 2017. ACL.

24 Joseph Weizenbaum. ELIZA – a computer program for the study of natural language communication between man and machine. *Communications of the ACM*, 9:36–45, 1966.

25 Jun Yin, Xin Jiang, Zhengdong Lu, Lifeng Shang, Hang Li, and Xiaoming Li. Neural generative question answering. In *International Joint Conference on Artificial Intelligence IJCAI*, pages 2972–2978, New York, 2016. IJCAI/AAAI Press.

26 Tiancheng Zhao and Maxine Eskenazi. Towards end-to-end learning for dialog state tracking and management using deep reinforcement learning. In *Proceedings of 17th Annual Meeting of the Special Interest Group on Discourse and Dialogue*, pages 1–10, Los Angeles, 2016. ACL.

NER in Archival Finding Aids

Luís Filipe Costa Cunha
University of Minho, Braga, Portugal

José Carlos Ramalho
Department of Informatics, University of Minho, Braga, Portugal

Abstract

At the moment, the vast majority of Portuguese archives with an online presence use a software solution to manage their finding aids: e.g. *Digitarq* or *Archeevo*.

Most of these finding aids are written in natural language without any annotation that would enable a machine to identify named entities, geographical locations or even some dates. That would allow the machine to create smart browsing tools on top of those record contents like entity linking and record linking.

In this work we have created a set of datasets to train Machine Learning algorithms to find those named entities and geographical locations. After training several algorithms we tested them in several datasets and registered their precision and accuracy.

These results enabled us to achieve some conclusions about what kind of precision we can achieve with this approach in this context and what to do with the results: do we have enough precision and accuracy to create toponymic and anthroponomic indexes for archival finding aids? Is this approach suitable in this context? These are some of the questions we intend to answer along this paper.

2012 ACM Subject Classification Computing methodologies → Natural language processing; Computing methodologies → Machine learning; Computing methodologies → Maximum entropy modeling; Computing methodologies → Neural networks; Information systems → Digital libraries and archives

Keywords and phrases Named Entity Recognition, Archival Descriptions, Machine Learning, Deep Learning

Digital Object Identifier 10.4230/OASIcs.SLATE.2021.8

1 Introduction

Throughout the history of Portugal, there was a need to create an archive where information about the kingdom was recorded.

In 1378, during the reign of D. Fernando, the first known Portuguese certificate was issued by *Torre do Tombo* (TT), an institution over 600 years old that is still the largest Portuguese archive, storing a great part of Portuguese historical and administrative records. As time passed, the volume of information contained in national archives has considerably increased, and today there are hundreds of archives spread across the country. Most of these archives have information from the public administration containing records from the 20th century onwards, however, Portugal has three archives with historical information, the *Arquivo Nacional da Torre do Tombo*, the *Arquivo Distrital da cidade de Braga* and the *Arquivo Distrital da cidade de Coimbra* which record various events throughout the history of the country.

The city of Braga was for many years the administrative capital of northern Portugal and of Galicia. In antiquity, most of the records were made by the clergy social class. Even today, the the church's strong influence in the district of Braga is visible, something that influenced the abundance and variety of historical document fonds present in the archive of this district.

At the moment, many of these archival documents are already available to the public in digital format, so it is now intended to interpret their content from a semantic point of view, i.e., to classify and extract different types of Named Entities (NE) present in a

© Luís Filipe Costa Cunha and José Carlos Ramalho;
licensed under Creative Commons License CC-BY 4.0

given fond. Therefore, this paper proposes the use of entity recognition in natural language, using Machine Learning (ML), a well-known and widely used technique in Natural Language Processing (NLP). In this way, several ML algorithms will be presented, with the intent of generating different results and conclude which algorithm best suits the domain and the problem in question.

2 Related Work

The amount of historical information available in Portuguese archives is increasing, making the exploration of this data complex. Thus, the use of the available computational power is not something new for professional historians. In fact, there are several tools that have been developed over time that assist in the archival data processing.

An example of this is the *HITEX* [17] project, developed by the *Arquivo Distrital de Braga* between 1989 and 1991. This project consisted of semantic model development for the archive historical data, something quite ambitious for that time. Despite this, during its development, it ended up converging to an archival transcription support system, which allowed the transcription of natural text and the annotation by hand of Named Entities enabling the creation of chronological, toponymic and anthroponomic indexes.

Another problem associated with this type of documents was its structure's lack of standardisation. This made it difficult to share information between the archival community both nationally and internationally. To promote interoperability, in Portugal, guidelines for the archival description have been created that describe rules for standardising the archival descriptions [22]. The purpose of these standards is to create a working tool to be used by the Portuguese archivist community in creating descriptions of the documentation and its entity producer, thus promoting organisation, consistency and ensuring that the created descriptions are in accordance with the associated domain's international standards. In addition, the adoption of these guidelines makes it possible to simplify the research or information exchange process, whether at the national or international level.

3 Named Entity Recognition

One of the objectives of NLP is the classification and extraction of certain entities in textual documents. It is easy to understand that entities such as people's names, organisations, places or dates translate into crucial information about certain contexts because this type of data can be used for various purposes, making this practice very popular. Therefore a new NLP subfield rises, Named Entity Recognition (NER).

To be able to recognise entities in texts, two different approaches were taken [10]. Initially, very specific regular expressions were coded to filter various entity types. This mechanism had good results in certain cases where there was an in-depth knowledge of the domain in which this method was intended to be applied. However, this is not always the case, for example, such an approach was considered not to be very dynamic due to the fact that it is necessary to rewrite a large part of the code if one wants to change the domain language. Furthermore, the existence of ambiguity between entities makes them hard to classify, like the names of people, places, etc.

Alternatively, statistical classifiers are used. This method consists of using ML models (this paper will deal with supervised ML models [23]) in order to try to predict whether a certain word sequence represents an entity.

This approach has some advantages over the previous one, such as being able to be used for different languages without having to change a lot of code, the model can be trained with different parameters adjusting to different contexts, an annotated dataset is generated that can be reused for other purposes, etc. In fact, today there are several already trained ML models capable of identifying and classifying various entities, however, the available models are generic, which means that entity prediction for more specific contexts will return results below expectations.

In spite of being much more dynamic than the previous approach, the use of this type of model leads to some work for the experimenter. Thus, it is necessary for the experimenter to write down an annotated training dataset to prepare the model. Despite being tedious work, it has a low complexity level and therefore does not require great specialisation.

4 OpenNLP

One of the tools chosen for this paper was *Apache OpenNLP*, a machine learning-based toolkit implemented in Java, developed by *Apache*. Essentially and as its name implies, its purpose is the processing of natural language through the use of ML algorithms having a wide range of features, such as tokenization, sentence segmentation, part-of-speech tagging, named entity extraction, chunking, parsing, and co-reference resolution [18].

In this paper, the features associated with NER will be addressed, which depend on the tokenization task. At this time, *Apache OpenNLP* provides models for various tasks in several different languages such as English, Spanish, Danish, and some more, however, there is no pre-processed model of NER for the Portuguese language provided by this tool. Available Portuguese annotated datasets like HAREM [6] and SIGARRA [19] were used to train NER models, however, the obtained results were below expectations. In this way, it will be necessary to train one model from scratch.

To understand how *OpenNLP* works, it is necessary to investigate what kind of ML algorithms it uses. In this case, the base algorithm used is Maximum Entropy (MaxEnt).

4.1 Maximum Entropy

We borrow the concept of entropy from physics (thermodynamics) to apply it to various areas of computer science like the *Information Theory* or even classification algorithms, such as MaxEnt where entropy represents the level of uncertainty.

According to [7] in the *Information Theory*, the occurrence of a given event with a low probability of occurring translates into more information than the occurrence of an event with a high probability of occurring. On the other hand, there is *Information Entropy*, which, in *Information Theory* corresponds to the measure of uncertainty, i.e., the average quantity of information required to represent an event drawn from the probability distribution for a random variable. The entropy takes a low value when the probability of certainty for some event is high and it takes a high value when all events are equally likely.

> *"Information entropy is a measure of the lack of structure or detail in the probability distribution describing your knowledge."* [Jaynes, E. T. 1982]

Maximum Entropy Models are statistical models that maximize the entropy of a probabilistic distribution subjected to an N number of constraints. These types of models reveal good results when used to model real-world problems considered hard to model. Usually, they are used on the prediction of high dimensional data, in other words, when there is a much greater number of possible combinations than the amount of available data.

The principle behind this algorithm is that the distribution with the most uncertainty, that is compatible with the context domain, should be chosen. To do so, it is necessary to create several features which represent the information known about the domain. In fact, these features represent restrictions of the model which help the classification of the intended target. After generating the features, it is then necessary to maximize the entropy of all models that satisfy these restrictions. By doing so, we are preventing our model from having features that are not justified by empirical evidence, preserving as much uncertainty as possible. [14]

"Ignorance is preferable to error and he is less remote from the truth who believes nothing than he who believes what is wrong." [Thomas Jefferson (1781)]

4.2 Features

As previously stated, a feature is a way in which known information about the context is passed to the model as constraints, i.e., evidence or hints that make the model correctly classify certain specific cases.

Mathematically speaking, it can be represented as a binary function that for some given $x \in X$, that represents the class of the entities we are trying to predict, and $y \in Y$, that represents the possible contexts that we are observing, it returns the corresponding boolean value.

$$f : X \times Y \longrightarrow \{0,1\}$$

All features correspond to functions with this signature, however, as already mentioned, a feature represents a constraint, which means that the experimenter must choose the type of information each feature adds to the model, for example:

$$f(a,b) = \begin{cases} 1 & \text{if a = Local and } \textit{checkLocation(b)} = \textit{true} \\ \\ 0 & \text{otherwise.} \end{cases}$$

$$\textit{checkLocation}(b) = \begin{cases} 1 & \text{if previous word in } b \text{ is “em” and current word starts} \\ & \text{with capital letter.} \\ \\ 0 & \text{otherwise.} \end{cases}$$

In this case, this feature helps the model to classify "`Local`" (place) type entities. In the Portuguese language, when a word beginning with a capital letter is anticipated by the word "em", there is a high probability that this word is a place (e.g., em Braga, em França).

These features are usually context-dependent, i.e., they must be created according to the problem to be modeled. There is often an interdependence between them, making it necessary to iterate over these features so that the decision to be made, in a given iteration, takes into account previous decisions. For example, in the presence of a proper name, it is normal to have a sequence of words that begin with a capital letter. When the first word of the sequence is classified as a person's name, it is very likely that the following words, which start with a capital letter, are also part of that name, so the classifications or decisions made previously are taken into account in current decisions.

As stated in [21], this behaviour about overlapping features is what makes the MaxEnt model really distinguish itself from other models, in the first place, because it is possible to add information already known through features, but also, by letting these features overlap in order to try to predict the best possible results.

4.3 Entropy Maximisation

Following the MaxEnt algorithm, the optimal solution to this classification problem is the most uncertain distribution subject to the defined constraints. The idea behind this is to choose the model that makes the fewer implicit assumptions as possible. Thus, after defining all constraints considered relevant to the context domain, the next step is to maximise the entropy of the model.

To do so, the function of Information Entropy is used.

$$H(X) = -\sum p(a,b) \log p(a,b)$$

The maximum entropy function is a convex function, which means that the value of the weighted average of two points is greater than the value of the function in this set of points. Thus, the sum of the entropy function is also convex. A constraint on this function creates a linear subspace that corresponds to a surface that is also convex and, therefore, has only a global maximum [13].

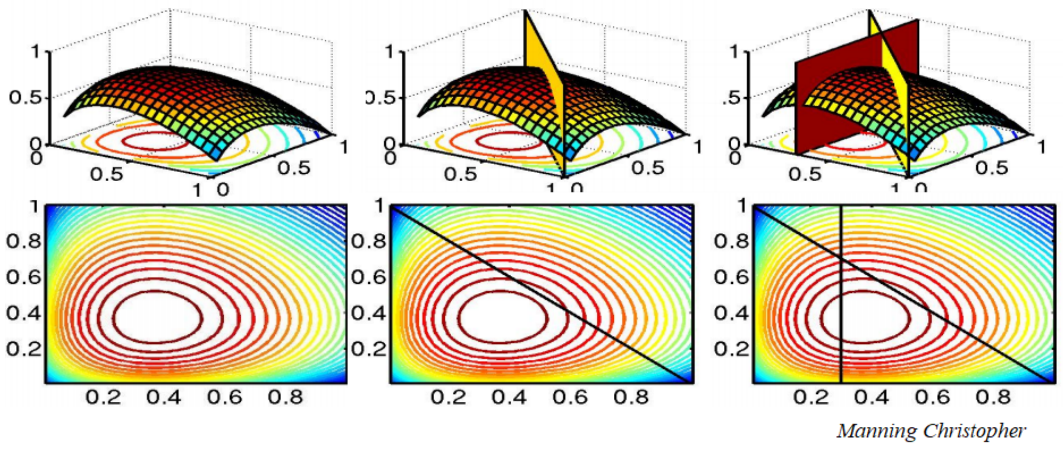


Figure 1 Entropy function subject to restrictions.

As explained in [3], in order to maximise the entropy of the model subject to a limited number of features, it is needed to solve a constrained optimisation problem. In other words, a problem with low complexity can be solved analytically, however, when the number of constraints increases and they overlap with each other, it is not possible to find a general solution analytically. This problem is then solved using Lagrange multipliers by forming a Lagrangian function. An example of this resolution can be found in [15].

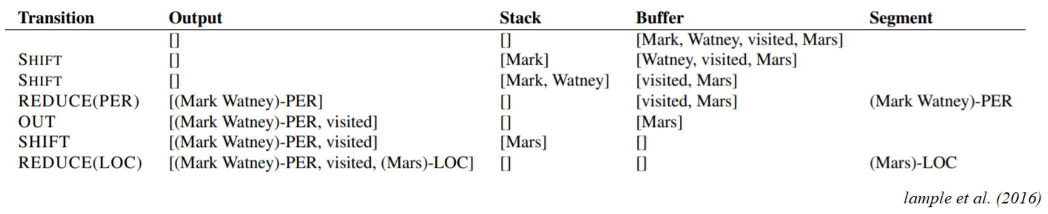
5 spaCy

Another tool that was used in this work is *spaCy*, an open-source library for advanced natural language processing, belonging to the company Explosion, founded by the creators of *spaCy*.

Again, this library offers several features associated with NLP, however, only those relevant to NER will be addressed. Despite having several similarities to *OpenNLP*, *spaCy* presents a very different approach to entity recognition. In addition, the support provided by its creators, documentation and information available about this software is much more accessible. This tool provides several base models of different languages such as Chinese, Danish, Dutch, English, French, Portuguese, etc., which is an advantage over the previous tool. It was implemented in the python programming language and published under the *MIT license.* In paper, the approach taken by *spaCy* regarding entity recognition [24] will be presented.

5.1 Transition Based NER

Most NER frameworks generally use a tagging system, which in practice translates into attaching a tag to each word of interest in the document to further classify it. Instead of using this type of structure, *spaCy* uses a different mechanism to deal with this problem, a transition-based approach.



Transition	Output	Stack	Buffer	Segment
	[]	[]	[Mark, Watney, visited, Mars]	
SHIFT	[]	[Mark]	[Watney, visited, Mars]	
SHIFT	[]	[Mark, Watney]	[visited, Mars]	
REDUCE(PER)	[(Mark Watney)-PER]	[]	[visited, Mars]	(Mark Watney)-PER
OUT	[(Mark Watney)-PER, visited]	[]	[Mars]	
SHIFT	[(Mark Watney)-PER, visited]	[Mars]	[]	
REDUCE(LOC)	[(Mark Watney)-PER, visited, (Mars)-LOC]	[]	[]	(Mars)-LOC

lample et al. (2016)

Figure 2 Example of Transition Based sequence applied on NER.

Analogous to a state machine, this approach is based a set of actions that the model can take in order to make the state machine transit into different states or configurations. The model always takes into account the first word in the buffer and then decides what action it should take. For example, in the Figure 2 we can see that state changes as actions are taken. The challenge of this system is in the prediction of the actions or transitions. In order to address this problem *spaCy* presents a new *Deep Learning* framework [25].

5.2 Deep Learning framework for NLP

In order to predict the actions to be taken in the transition-based model, a statistical model based on Neural Networks is used. The idea starts by finding representations for all words in a given document. After that, it is necessary to contextualise these words in the document, recalculating their representation value. Then the model comes up with a summary vector that represents all the information needed to help predict the target value of the word. From this vector, it is then possible to predict the next valid transition.

To structure this model, the deep learning framework *Embed, Encode, Attend, Predict* divides this whole process into four distinct components, in order to simplify its understanding [25].

5.2.1 Embed

The first task of this approach is *Embed.* This task consists of calculating embeddings using a word identifier, in order to generate vectors for each word in the document.

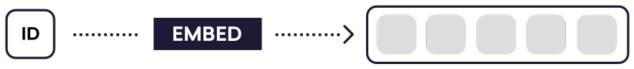


Figure 3 Embed process.

In fact, the objective of this stage is to generate different representations for words with different semantic meanings through multidimensional vectors. These vectors allow the use of a "hypothesis distribution" so that words that refer to the same entity will have a similar distribution value. This type of mechanism allows the model to be able to associate similar words semantically, even without completely knowing its definition or characteristics, i.e., it does not need to know the word's meaning, but it knows that some words are related in some manner, taking in account the surrounding word vectors. For example, to find out if a particular word refers to a student, words like `human` or `rational being` do not have much impact on entity recognition, however, words such as `study`, `book`, `class` or even `school` are usually related and used near the word `student`. This way the model knows that words like `student`, `pupil`, `finalist` are quite similar in distribution.

This type of mechanism makes the model less limited to the text that was annotated, which translates into a great capacity for learning.

5.2.2 Encode

The encode task aims to transform the vectors previously created in the embedding phase, which are context-independent, and take into account the context in which they are found, thus providing a matrix of context-sensitive vectors.

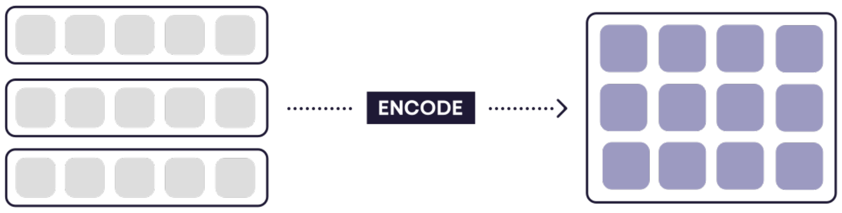


Figure 4 Encode process.

To make a vector, created from a word, context-dependent, it is necessary to look at the sentence in which this word belongs. For this, in NLP the most common way that several articles address this problem is the use of a Bidirectional Long Short Term memory (BI-LSTM) [11] which takes the whole sentence into account.

However, to determine the context of a vector in the sentence, *spaCy* uses a different method, Convolutional Neural Network (CNN).

This approach only uses four words on either side of a token instead of the whole phrase. Basically, the idea behind this approach is that when using the whole sentence to determine the context of a certain token, the best results are not always obtained, i.e., this practice can make the model show difficulties in knowing if certain context should be associated with the token, making it difficult to discern what matters from what does not. This approach can also provoke the model to over-fit the data, making the model sensitive to things it shouldn't. In this way, *spaCy*'s developer believes that in the vast majority of cases, a small window of words is sufficient to accurately represent the context of a token.

In addition to this, with this type of Neural Network, it is possible to create a decaying effect, to define the level of importance that a given context has on the vector of a word, which is not possible with the previously mentioned method (BI-LSTM).

Finally, it is interesting to note that CNNs have a lower computational cost compared to BI-LSTMs due to the fact that they take advantage of parallelism for each of its layers, managing to use the resources of GPU efficiently [26].

5.2.3 Attend

The third task of this framework is Attend, which consists of taking the matrix built previously and selecting all the necessary information to help the model with the prediction task.

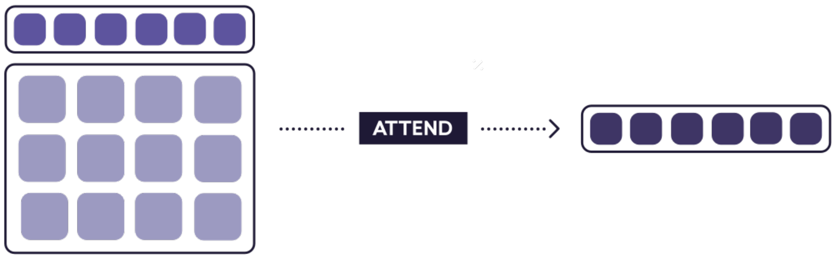


Figure 5 Attend process.

It is at this stage that the model takes into account the defined features. These have a great influence on the way vectors are created, so *spaCy* allows defining them arbitrarily. This type of mechanism ensures dynamism and versatility to the system because depending on the context, it may be necessary to tune the model by modifying the set of features that are being used. At the moment, *spaCy*'s features are defined in the following two steps. Select the first word that is in the buffer, the words that are immediate to the left and right of that word and the last entities previously classified by the model (as the model reads the document from left to right, it is not possible to take into account the entities to the right of the word, because they have not yet been recognised by the model).

After selecting the desired information, a vector that represents problem-specific information is generated and is ready to be used in the prediction phase.

5.2.4 Predict

Finally, there is the last task of this framework, the Prediction.

Figure 6 Predict process.

As its name implies, this step is based on the prediction of the target value. After all the words are turned into vectors (*Embbed*), the vectors are contextualised with the document (*Encode*) and the feature defined are taken into account (*Attend*), the system is ready to make the prediction. This prediction is made using a simple multi-layer perceptron which will return the action probabilities. Then, it is necessary to validate these actions and finally choose the action according to the algorithm's confidence. Finally, this process is iterated through a cycle until the document is finished.

It is important to emphasise that all stages of this framework are pre-computed, that is, they occur outside the cycle, so when the model goes through the document, fewer computations are needed.

6 TensorFlow BI-LSTM-CRF

The *Tensorflow* library, developed by *Google*, presents a vast set of ML features, usually associated with neural networks, which allow to develop and train models in a similar way to the learning method of the human mind. It is an Open Source library, published under *Apache 2.0 license*. By using this library, it is intended to implement a *Deep Learning* model capable of performing NER on archival documents.

Thus, it is necessary to create a system capable of processing the input data, i.e., tokenize the documents and generate word embeddings in order to use them in a specific ML architecture capable of solving sequence tagging problems.

6.1 Recurrent Neural Network

One of the first approaches associated with Deep Learning in NLP was to use Recurrent Neural Networks [8].

In fact, RNNs are famous for obtaining good results on sequential data, which makes them the perfect algorithm for analysing natural text. Despite this, the research community quickly encountered problems associated with this method. First of all, this type of neural network is unidirectional, i.e., it would only take into account the context of the sentence that is before a given token. It is easy to understand that this is a problem when trying to identify a token's entity, because, in order to accurately classify the token, the context of the word's neighbourhood, whether refers to the past or future input, must be taken into account.

In addition, this type of neural network has difficulties in preserving Long Term Dependencies due to the phenomenon of Vanishing Gradient. As the name implies, an RNN has difficulties in preserving contexts observed throughout the sentence, so if there are clues at the beginning of a long sentence, which could help to identify the entity of a token, found at the end of that sentence, they will not be taken into account, which leads to a poor classification.

6.2 Long Short Term Memory

In order to combat RNNs problems, LSTMs are introduced. Basically, LSTMs are RNNs with a memory component added to them in order to create an RNN with Long Term Memory capable of preserving Long Term Dependencies. The introduction of this cell of memory keeps a state that is being updated along the RNN chain. The update of this state is done through gates [16], input, output and forget that regulate the information that must be updated at each timestep, the information that must be passed to the next cell and the information that must be forgotten, respectively. While this state is being updated by the cells of the network, a notion of context is generated for each token, making the RNN capable of taking into account Long Term dependencies.

Using this new method, the Long Term Dependencies problem is solved. However, an LSTM remains unidirectional, so the information that follows a token will not be considered in its classification.

6.3 Bidirectional Long Short Term Memory

A BI-LSTM consists of using two LSTMs, one that will analyse the document in the forward direction, the other backward. The idea behind this architecture is to have an LSTM responsible for capturing the previous context (forward layer), but also, use another LSTM to exploit future context as well (backward layer). In this way, there is information of the entire document associated with each token, contextualising it, something that did not happen with a simple RNN or LSTM. This is only possible due to the ability to preserve Long Term Dependencies resulting from the use of the memory cell otherwise, even if the model analysed the whole sentence it would not be able to maintain all dependencies.

However, although these approaches have better results than a simple RNN, it is important to note that they need more computational resources as it is now necessary to process the memory component in a bidirectional structure.

6.4 BI-LSTM-CRF

For some time, BI-LSTM alone reigned, obtaining state of the art results in several NLP tasks until, in 2015, a new article [9] came out that introduced a new model, BI-LSTM-CRF.

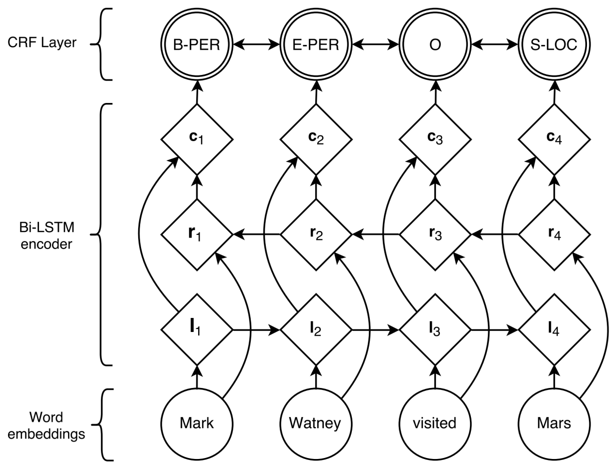


Figure 7 Bidirectional Long Short Term Memory Conditional Random Field.

This architecture consists of using a BI-LSTM and adding a Conditional Random Field (CRF) network to it. This new component receives the consecutive tagging outputs from the BI-LSTM and is responsible for decoding the best tagging sequence, boosting the tagging accuracy [12]. In fact, knowing the relationships between the different labels that one wants to identify in the document, can help the model to classify, with greater accuracy, the best chain of labels. For example, using the BIO format to annotate the phrase "`João Sousa Batista plays Tetris`" we have the following result `[B-Person, I-Person, I-Person, O, O]`. When writing down the name of the person "`João Sousa Batista`" it does not make sense to have the sequence `[B-Person, I-Date, B-Person]`, so this new component comes in a way to help validate the output tag sequence.

The use of BI-LSTM-CRF has already been tested in NLP and has demonstrated state-of-the-art results in various tasks. In this way, this work will use the *TensorFlow* library to implement a solution with this architecture, testing it in archival documents.

7 DataSets

The data used to test the algorithms referred in this article correspond to datasets from two national archives, the Arquivo Distrital de Braga [1] and the Arquivo Regional e Biblioteca Pública da Madeira [2] where the following NE types were extracted: Person, Profession or Title, Place, Date and Organisation.

Firstly, there is a dataset of a fond that shows a pioneering period in computing history, between 1959 and 1998. This fond (PT/UM-ADB/ASS/IFIP), produced by the International Federation for Information Processing (IFIP), contains a section corresponding to the Technical Committee 2, which has a subsection corresponding to Working Group 2.1, that is composed of several series where different archival descriptions are organised, for example, correspondences, meeting Dossiers, news from newspapers, etc.

Secondly, there are two datasets corresponding to a series (PT/UM-ADB/DIO/MAB/006), from the archival fond Mitra Arquiepiscopal de Braga, which contains genre inquiries. The archival descriptions in this series contain witnesses' inquiries to prove applicants' affiliation, reputation, good name or "blood purity". One of the datasets has a very standardised structure, while the other contains a lot of natural text elements.

Thirdly, there is a historical dataset corresponding to the fond (PT/UM-ADB/FAM/ACA) of the Arquivo da Casa do Avelar (ACA) which depicts the family history of Jácome de Vasconcelos, knight and servant of King D. João I. This family settled in Braga around the years 1396 and 1398 with a total of 19 generational lines, up to the present time [1]. This fond is composed of subfonds and subsubfonds that contain records associated with members of this family with a patrimonial, genealogical and personal domain.

Fourth, there is the dataset of the Familia Araújo de Azevedo fond (FAA), also known as Arquivo do Conde da Barca. This archive, produced from 1489 to 1879 by Araújo de Azevedo's family, who settled in Ponte da Barca and Arcos de Valdevez (district of Viana do Castelo) at the end of the 14th century, contains records predominantly associated with foreign policy and diplomacy across borders. This fond is composed of several subfonds composed of archival descriptions with information from members of the FAA family, such as requirements, letters, royal ordinances, etc.

Fifth, there is a dataset that characterizes the streets of Braga in the year 1750. This corpus contains elements that characterize the history, architecture and urbanism of each artery in the city, which help to understand the main lines of its evolution.

Finally, two datasets from the Arquivo Regional e Biblioteca Pública da Madeira were used which correspond to two archival fonds, more precisely to Paróquia do Jardim do Mar and Paróquia do Curral das Freiras, both parishes from Madeira archipelago. These fonds consists of three series each, which represent registrations of weddings, baptisms and deaths. Each series consists of files, that correspond to the year of each record and finally, each file has a set of pieces with archival descriptions.

In total, the annotated corpora contain 164478 tokens that make up 6302 phrases. All the annotated corpus are available to the public in [4]. The distribution of entities is presented in the Table 1.

Table 1 Number of annotated entities per corpus.

Corpus	Person	Place	Date	Professions or Title	Organization	Total
IFIP	1503	325	100	40	318	2286
Familia Araújo de Azevedo	369	450	118	428	94	1459
Arquivo da Casa Avelar	465	239	141	118	91	1054
Inquirições de Genere 1	2002	3713	121	0	0	5836
Inquirições de Genere 2	692	10	54	0	0	756
Jardim do Mar	2393	574	1762	1	2	4732
Curral das Freiras	8729	0	0	0	0	8729
Ruas de Braga	1126	1293	684	391	338	3832
Total	17279	6604	2980	978	843	28684

8 Data Processing

In order to perform entity recognition with *OpenNLP*, *spaCy* and *TensorFlow* in these datasets, it is necessary to train different models so that they learn to find Named Entities in different contexts accurately.

For that, it is necessary to have annotated text that represents each dataset domain. In order to do so, a shuffle of each dataset was performed proceeded with the annotation of a significant fraction of each of them. This shuffle allows the data selection to be impartial, making it a more representative sample of each context domain. During the annotation process, several techniques were used, such as regular expressions, manual annotation and even the use of a statistical model proceeded by correction of the output by the annotator. To facilitate this process, a simple javascript program was created that allows to annotate texts in the browser with a simple keypress.

After the annotation process, the annotated datasets were divided into two parts, 70% of each was used to train the model while the remaining 30% was reserved for validation. It is important to note that the three tools used to implement the NER algorithm use different input data formats. Thus, in this stage, parsers were implemented to convert datasets into three different formats. After that, the tokenization process is initiated. Both *spaCy* and *OpenNLP* have their own Portuguese optimised tokenizer, however, *TensorFlow* does not implement this tool out of the box. In this way, several tokenizers were experimented such as using the Keras tokenizer API, the use of regular expressions and finally *spaCy*'s tokenizer. In this case, *spaCy*'s tokenizer showed better results due to the fact it is optimised for the language in question.

With the datasets processed we feed them into the ML algorithms in order to train the NER models. In this process, individual optimisations are performed for each tool, such as defining the hyperparameters, tunning the models in order to generate the best results.

9 Results

The metrics used to measure NER models' performance are Recall, Precision and F1-score since the accuracy metric does not satisfy the needs of this NLP area [5].

Looking at the Table 2, we can conclude that the created NER models were able to successfully classify most of the intended entities. It appears that in most cases, the BI-LSTM-CRF model generated with *TensorFlow* obtains the best results with an F1-score between 86,32% and 100%, followed by *spaCy* with an F1-score between 70,09% and 100%, and finally *OpenNLP* with an F1-score between 62,67 and 100%. As we can see with these results, the introduction of Deep Learning on NER reveals significant advances in this field.

It is important to note that only one model was created for datasets with high proximity in the context domain. For *OpenNLP*, when using the corpus Genere Inquiries 2 to validate the model trained on the Genere Inquiries 1 dataset, the results obtained were lower (62,67% F1-score) in comparison to the other tools (87,78% and 98,78% F1-score). In this case, it turns out that deep learning has demonstrated a greater capacity for transfer learning.

Finally, analysing the Table 2 we see that the FAA dataset is the one in which the models presented the lowest results. One reason for this is that it contains very long sentences. In fact, as previously seen, a Bi-LSTM-CRF is prepared to deal with Long Term Dependencies which makes it present better results than the other tools (86.32% F1-score).

Table 2 Named Entity Recognition results.

Corpus	Model	Tool	Precision(%)	Recall(%)	F1-Score(%)
IFIP	IFIP	OpenNLP	87,08	82,61	84,79
		spaCy	88,16	89,90	89,02
		TensorFlow	96,12	98,67	97,00
Familia Araújo de Azevedo	Familia Araújo de Azevedo	OpenNLP	72,56	72,30	72,43
		spaCy	74,41	72,82	74,09
		TensorFlow	88,98	87,28	86,32
Arquivo da Casa Avelar	Arquivo da Casa Avelar	OpenNLP	80,15	79,85	80,00
		spaCy	87,82	87,18	87,50
		TensorFlow	89,25	93,25	90,63
Inquirições de Genere 1	Inquirições de Genere 1	OpenNLP	99,93	98,87	99,90
		spaCy	97,35	95,08	96,20
		TensorFlow	100	100	100
Inquirições de Genere 2	Inquirições de Genere 1	OpenNLP	63,17	62,17	62,67
		spaCy	89,66	85,98	87,78
		TensorFlow	98,86	98,95	98,78
Jadim do Mar	Jardim do Mar	OpenNLP	100	99,86	99,93
		spaCy	100	100	100
		TensorFlow	100	100	100
Curral das Freiras	Jardim do Mar	OpenNLP	93,37	99,84	96,50
		spaCy	99,97	99,90	99,93
		TensorFlow	100	100	100

After obtaining the above results, an attempt to create a generalised model with annotations of all datasets was made.

Annotating a fraction of a dataset where this technology is to be applied is not always practical, so it would be interesting to create a generalised model capable of adapting to new contexts of similar nature. On the other hand, it is important that this new model doesn't obtain worse results in the already observed datasets, due to its degree of generalisation.

In this way, the generalised model was trained with 70% of each dataset to be later validated with the remaining 30% of each one. This procedure was repeated for the three tools, obtaining the following results.

Table 3 Generalised NER model validation results.

Corpus	Tool	Precision(%)	Recall(%)	F1-Score(%)
IFIP	OpenNLP	89.43	83.60	86.41
	spaCy	86.99	88.71	87.84
	TensorFlow	92.84	96.85	94.08
Família Araújo Azevedo	OpenNLP	81.94	63.67	71.66
	spaCy	75.19	76.78	75.98
	TensorFlow	78.22	82.47	78.89
Arquivo da Casa Avelar	OpenNLP	88.84	81.68	85.11
	spaCy	87.18	87.18	87.18
	TensorFlow	86.83	92.21	87.99
Inquirições de Genere 1	OpenNLP	99.60	99.53	99.57
	spaCy	98.31	96.74	97.52
	TensorFlow	100	100	100
Inquirições de Genere 2	OpenNLP	74.70	65.61	69,80
	spaCy	79.96	92.21	87.26
	TensorFlow	93.70	98.34	94,82
Jardim do Mar	OpenNLP	99.71	99.71	99.71
	spaCy	99.15	100	99.57
	TensorFlow	100	99.60	99.72
Curral das Freiras	OpenNLP	93.49	99.69	96.49
	spaCy	99.98	99.90	99.94
	TensorFlow	100	100	100

As can be seen, the results obtained by this model are similar to the previous ones, so we can say that the NER performance has not decreased. To measure his performance in a different context, a new corpus with brief notes of the streets of Braga (1750), was annotated and then, after processing it, it was used as validation generating the following results:

Table 4 Generalised NER model validation results on Ruas de Braga corpus.

Corpus	Tool	Precision(%)	Recall(%)	F1-Score(%)
Ruas de Braga	OpenNLP	73.09	61.09	66.55
	spaCy	75.39	62.62	68.42
	TensorFlow	50.50	58.80	53.00

The results obtained are lower than intended. In fact, in addition to the model not having been trained with any part of this dataset, it contains a lot of Organization and Profession type entities. As can be seen in Table 1, the model was trained with few instances of this type, thus the entity recognition may prove challenging.

On the other hand, it appears that the model generated with BI-LSTM-CRF obtained worse results. One of the reasons for this is the fact that this model has a vocabulary reduced to his training data. In fact, both deep learning frameworks presented in this paper represent words through word embeddings, however, *spaCy* uses pre-trained word embeddings from a Portuguese corpus named *Bosque* [20] which makes it have a much larger vocabulary. In this way, *spaCy*'s model can assign semantic meaning to words that were not present in his training which becomes a valuable tool when evaluating corpus from different contexts than the one it was trained on.

10 Conclusion

The archival finding aids used in this paper contain very specific structure and context, which means that available generic NER models may present results that are lower than intended. In addition, there is the language barrier, that is, the amount of Portuguese annotated data, available to train this type of model, is limited.

Despite this, in this paper, it was demonstrated that by training our own models with data that coincides with those we intend to perform NER, it is possible to obtain satisfactory results. In fact, with the advances in the use of Deep Learning in this area of NLP, F1-score values above 86% were achieved in almost all of the used datasets.

Thus, observing the obtained results, it is considered that the use of ML algorithms to perform entity recognition in archival documents, is suitable and with this approach, we can extract information that allows us to create different navigation mechanisms and create relations between information records.

Future work

One way to improve the results presented in this article would be to increase the amount of annotated data. Passing more information to the models' training makes them able to process a greater variety of data making them more generic. On the other hand, it would be interesting to explore new technologies that aim to address this problem, for example, the attention mechanism [27].

Lastly, entity linking could be performed in order to make it possible to browse between different archival documents but related by some entity. It would also be interesting to use the extracted entities to create toponymic and anthroponomic indexes to understand the impact that this tool could have on browsing archival finding aids.

References

1 Archeevo Arquivo Distrital de Braga. Bem-vindo ao arquivo distrital de braga. Accessed in 10-03-2021. URL: `http://pesquisa.adb.uminho.pt/`.

2 Archeevo Arquivo Regional e Biblioteca Pública da Madeira. Accessed in 10-03-2021. URL: `https://arquivo-abm.madeira.gov.pt/`.

3 Adam L. Berger, Vincent J. Della Pietra, and Stephen A. Della Pietra. A Maximum Entropy Approach to Natural Language Processing. *Computational Linguistics*, 1996.

4 Luís Filipe Costa Cunha and José Carlos Ramalho. URL: `http://ner.epl.di.uminho.pt/`.

5 Leon Derczynski. Complementarity, F-score, and NLP evaluation. In *Proceedings of the 10th International Conference on Language Resources and Evaluation, LREC 2016*, 2016.

6 Cláudia Freitas, Cristina Mota, Diana Santos, Hugo Gonçalo Oliveira, and Paula Carvalho. Second HAREM: Advancing the state of the art of named entity recognition in Portuguese. In *Proceedings of the Seventh International Conference on Language Resources and Evaluation (LREC'10)*, Valletta, Malta, 2010. European Language Resources Association (ELRA). URL: `http://www.lrec-conf.org/proceedings/lrec2010/pdf/412_Paper.pdf`.

7 Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning (Adaptive Computation and Machine Learning series)*. The MIT Press, November 2016. URL: `https://www.xarg.org/ref/a/0262035618/`.

8 Alex Graves, Abdel Rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*, 2013. `doi:10.1109/ICASSP.2013.6638947`.

9 Zhiheng Huang, Wei Xu, and Kai Yu. Bidirectional lstm-crf models for sequence tagging, 2015. `arXiv:1508.01991`.

10 Grant S. Ingersoll, Thomas S. Morton, and Andrew L. Farris. *Taming text: how to find, organize, and manipulate it.* Manning, Shelter Island, 2013. OCLC: ocn772977853.

11 Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, and Chris Dyer. Neural architectures for named entity recognition. In *2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL HLT 2016 - Proceedings of the Conference*, 2016. doi:10.18653/v1/n16-1030.

12 Xuezhe Ma and Eduard Hovy. End-to-end sequence labeling via bi-directional LSTM-CNNs-CRF. In *54th Annual Meeting of the Association for Computational Linguistics, ACL 2016 - Long Papers*, 2016. doi:10.18653/v1/p16-1101.

13 Christopher Manning. Maxentmodels and discriminative estimation. URL: https://web.stanford.edu/class/cs124/lec/Maximum_Entropy_Classifiers.pdf.

14 OpenNLP Maxent. The maximum entropy framework, 2008. Accessed in 24-09-2020. URL: http://maxent.sourceforge.net/about.html.

15 Mike Morais. Neu 560: Statistical modeling and analysis of neural data: Lecture 8: Informationtheory and maximum entropy, 2018. Accessed in 20-10-2020. URL: http://pillowlab.princeton.edu/teaching/statneuro2018/slides/notes08_infotheory.pdf.

16 Christopher Olah. Understanding lstm networks, August 2015. Accessed on March 10, 2021. URL: http://colah.github.io/posts/2015-08-Understanding-LSTMs/.

17 José Nuno Oliveira. Hitex: Um sistema em desenvolvimento para historiadores e arquivistas. *Forum*, 1992.

18 Apache OpenNLP. Welcome to apache opennlp, 2017. Accessed in 18-10-2020. URL: https://opennlp.apache.org/.

19 André Ricardo Oliveira Pires. Named entity extraction from portuguese web text. Master's thesis, Faculdade de Engenharia da Universidade do Porto, 2017.

20 Alexandre Rademaker, Fabricio Chalub, Livy Real, Cláudia Freitas, Eckhard Bick, and Valeria de Paiva. Universal Dependencies for Portuguese. In *Proceedings of the Fourth International Conference on Dependency Linguistics (Depling 2017)*, 2017.

21 Adwait Ratnaparkhi. *Maximum entropy models for natural language ambiguity resolution.* PhD thesis, University of Pennsylva, 1998.

22 Ana Maria Rodrigues, Catarina Guimarães, Francisco Barbedo, Glória Santos, Lucília Runa, and Pedro Penteado. Orientações para a descrição arquivística, May 2011. URL: https://act.fct.pt/wp-content/uploads/2014/05/ODA-3%C2%AA-vers%C3%A3o.pdf.

23 Satoshi Sekine and Elisabete Ranchhod. *Named Entities: Recognition, classification and use.* John Benjamins Publishing Company, July 2009.

24 spaCy. spacy 101: Everything you need to know · spacy usage documentation. Accessed in 07-01-2021. URL: https://spacy.io/usage/spacy-101.

25 spaCy. Model architecture, 2017. Accessed in 14-01-2021. URL: https://spacy.io/models.

26 Emma Strubell, Patrick Verga, David Belanger, and Andrew McCallum. Fast and accurate entity recognition with iterated dilated convolutions. *CoRR*, 2017. doi:10.18653/v1/d17-1283.

27 Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in Neural Information Processing Systems*, 2017-December, 2017.

Mooshak's Diet Update: Introducing YAPExIL Format to Mooshak

José Carlos Paiva
CRACS – INESC-Porto LA, Portugal
DCC – FCUP, Porto, Portugal

Ricardo Queirós
CRACS – INESC-Porto LA, Portugal
uniMAD – ESMAD, Polytechnic Institute of Porto, Portugal

José Paulo Leal
CRACS – INESC-Porto LA, Portugal
DCC – FCUP, Porto, Portugal

Abstract

Practice is pivotal in learning programming. As many other automated assessment tools for programming assignments, Mooshak has been adopted by numerous educational practitioners to support them in delivering timely and accurate feedback to students during exercise solving. These tools specialize in the delivery and assessment of blank-sheet coding questions. However, the different phases of a student's learning path may demand distinct types of exercises (e.g., bug fix and block sorting) to foster new competencies such as debugging programs and understanding unknown source code or, otherwise, to break the routine and keep engagement. Recently, a format for describing programming exercises – YAPExIL –, supporting different types of activities, has been introduced. Unfortunately, no automated assessment tool yet supports this novel format. This paper describes a JavaScript library to transform YAPExIL packages into Mooshak problem packages (i.e., MEF format), keeping support for all exercise types. Moreover, its integration in an exercise authoring tool is described.

2012 ACM Subject Classification Applied computing → Computer-managed instruction; Applied computing → Interactive learning environments; Applied computing → E-learning

Keywords and phrases programming exercises format, interoperability, automated assessment, learning programming

Digital Object Identifier 10.4230/OASIcs.SLATE.2021.9

Category Short Paper

Supplementary Material *Software (Source Code)*: `https://github.com/FGPE-Erasmus/yapexil-mef-converter`; archived at `swh:1:dir:017c9838b4dfb8d45092d31e55804671f3449de2`

Funding This paper is based on the work done within the Framework for Gamified Programming Education (FGPE) Plus: Learning tools interoperability for gamified programming education project supported by the European Union's Erasmus Plus programme (agreement no. 2020-1-PL01-KA226-HE-095786).

1 Introduction

Learning programming demands a tremendous amount of practice. Practice in this domain consists of developing a program that attempts to solve a problem, receive feedback on it, and use this feedback to further improve the solution until it meets all requirements. As the number of students enrolled in programming courses grows, educational practitioners rapidly started to search for tools such as contest management systems, evaluation engines, and online judges to support them in delivering timely and accurate feedback to students during exercise solving.

© José Carlos Paiva, Ricardo Queirós, and José Paulo Leal;
licensed under Creative Commons License CC-BY 4.0

There is a panoply of different tools providing automated assessment of programming assignments, but the vast majority targets a single type of activity by design: blank-sheet programming exercises, where the student is challenged to code a solution from scratch to a presented problem statement. One of such tools is Mooshak, an open-source web-based system for managing programming contests, which includes automatic judging of submitted programs. Even though Mooshak handles custom static and dynamic analysis scripts and, thus, enables the assessment of non-traditional programming exercises, its format – the Mooshak Exchange Format (MEF) – does not forecast support for other types of programming activities.

Only recently, to the best of the authors' knowledge, an open format for describing programming exercises that explores the development of new competencies through different types of activities, such as understanding unknown source code and debugging, has been published – the Yet Another Programming Exercises Interoperability Language (YAPExIL). YAPExIL ships with direct support for seven types of activities, namely blank-sheet, solution improvement, bug fix, gap filling, block sorting, and spot the bug, to be applied at different phases of a student's learning path or solely to dissipate the monotony associated with solving exercises of the same type.

However, YAPExIL is a completely decoupled format, i.e., it is neither associated nor targets any specific tool. Hence, due to its recency, there is still no automated assessment tool that can work with problem packages adhering to the YAPExIL format. This paper introduces a JavaScript library to transform YAPExIL packages into problem packages adhering to the MEF format, ready to be consumed by Mooshak. This conversion has minimal loss of the encoded content in YAPExIL, maintaining support for all its types of programming exercises.

The remainder of this paper is organized as follows. Section 2 surveys the mentioned formats, highlighting both their differences and similar features. In Section 3, the YAPExIL to MEF conversion JavaScript library is introduced. Then, Section 4 demonstrates how the different exercise types can be described in MEF. Finally, Section 5 summarizes the main contributions of this research and presents some plans for future work.

2 Related Work

This section describes the programming exercise formats previously mentioned, namely MEF [1] and YAPExIL [3]. According to [3], which follows an extended Verhoeff model [5] to evaluate expressiveness, these are the two most expressive programming exercise formats (excluding PExIL [4], which is a subset of YAPExIL).

2.1 Mooshak Exchange Format (MEF)

Mooshak [1] is a web-based contest management system originally developed to manage programming contests over the Internet. Like many other systems of similar categories, it features its own internal format to describe programming exercises called **M**ooshak **E**xchange **F**ormat (MEF). A package adhering to MEF includes an XML manifest file in the root, containing metadata and references to other types of resources present in the package such as problem statements (e.g., PDF and HTML), images, input/output test files, static and dynamic correctors, and solutions. The manifest file can also hold points (i.e., the weight of the test within the overall problem grade) and feedback messages for each test case.

Due to its wide adoption in educational contexts to assist teachers in delivering timely and accurate feedback on programming assignments, version 2 of Mooshak heads most efforts towards its improvement for educational contexts. In particular, it extends MEF to support code skeletons for students to start from as well as to give the possibility to have multiple solutions, public and private tests, and different editors for separate kinds of exercises (e.g., code and diagram).

2.2 Yet Another Programming Exercises Interoperability Language (YAPExIL)

Yet **A**nother **P**rogramming **Ex**ercises **I**nteroperability **L**anguage (YAPExIL) [3] is a language for describing programming exercise packages, which builds on top of the XML dialect PExIL (**P**rogramming **Ex**ercises **I**nteroperability **L**anguage) [4]. Comparatively, YAPExIL (1) is formalized through a JSON Schema rather than an XML Schema, (2) replaces the complex logic for automatic test generation with a script provided by the author, and (3) adds several assets that underpin different types of programming exercises. Its JSON Schema can be divided into four separate facets: **metadata**, containing simple values providing information about the exercise; **presentation**, including components that are presented to either the student or the teacher (e.g., problem statement and instructions); **assessment**, involving what enters in the evaluation phase (e.g., tests, solutions, and correctors); and **tools**, containing any additional tools that complement the exercise (e.g., test generators).

The two central goals of YAPExIL are (1) to fully cover Verhoeff's expressiveness model [5]; (2) to support the definition of different types of programming exercises beyond the traditional ones. Both of these goals have been achieved and proven [3]. In particular, it includes support for seven different types of programming exercises:

- `BLANK_SHEET`, which challenges the student to write her solution to a given problem statement starting from a blank sheet;
- `EXTENSION`, that asks the student to complete a partially finished solution source code, without modifying the provided parts;
- `IMPROVEMENT`, which provides a correct initial source code that does not yet fulfil all the goals set in the exercise specification (e.g., develop a solution with less than 2 loops), so the student has to modify it to solve the exercise;
- `BUG_FIX`, that challenges the student to fix a solution source code in which a few bugs have been planted;
- `FILL_IN_GAPS`, which provides code with missing parts and asks students to fill them with the right code;
- `SPOT_BUG`, which asks students to merely indicate the position of the bugs in a solution source code with a few bugs;
- `SORT_BLOCKS`, that asks students to sort the broken and shuffled blocks of code of a correct solution.

3 YAPExIL-to-MEF Converter

The conversion from YAPExIL to MEF has been developed as a JavaScript library[1], using version ES2017. The library exports two API methods for its clients, namely

`yapexil2mef(pathToZip, outputPath = 'output.zip', options = {})`, that reads a ZIP archive with a YAPExIL programming exercise located at `pathToZip` and writes a new ZIP archive at `outputPath` with a MEF programming exercise.

[1] `https://github.com/FGPE-Erasmus/yapexil-mef-converter`

`yapexil2mefStream(zipStream, outputStream, options = {})`, which receives an input stream `zipStream` from a ZIP archive containing a YAPExIL programming exercise and writes a MEF programming exercise to `outputStream`.

Both methods receive an `options` parameter containing a list of files to ignore and supported image and statement extensions.

Following YAPExIL facet structure, the actual conversion can be seen as a four-step process, including metadata, presentation, evaluation, and tools steps. In the metadata step, only the `title`, `module`, `type`, and `difficulty` map into MEF format, while `author`, `status`, `keywords`, `event`, and `platform` are left out (except if platform matches `-timeout N`, then `N` is set as the default problem timeout). The `title` is set as the `Name` of the MEF programming exercise. Moreover, it is also part of the `Title` field, which consists of a concatenation of the `module` with the `title` of the YAPExIL exercise. The `difficulty` maps almost directly to MEF's `Difficulty` as both are 5-level scales (i.e., from `beginner`, `easy`, `average`, `hard`, and `master` to `VERY_EASY`, `EASY`, `MEDIUM`, `DIFFICULT`, and `VERY_DIFFICULT`, respectively).

The `type` of exercise requires some extra work, as it is an important field in respect to the support for different types of exercises and has no related property in the target format. While the `BLANK_SHEET`, `EXTENSION`, `IMPROVEMENT`, and `BUG_FIX` exercises can be solved using a common code editor and only differ on the type of code skeletons provided by the author, the `FILL_IN_GAPS`, `SPOT_BUG`, and `SORT_BLOCKS` exercises may require appropriate user interface elements to be solved. Hence, from the evaluation engine perspective, the latter need to keep some sort of indicator about the kind of editor to display in the user interface, whereas the former uses the default editor. As MEF supports code, diagram, and quiz questions, there is already the property `Editor_kind` with a similar purpose, but a different set of possible values: `CODE`, `DIAGRAM`, and `QUIZ`. This set has been extended with three new values, namely `FILL_IN_GAPS`, `SPOT_BUG`, and `SORT_BLOCKS`, to maintain support for all exercise types. Table 1 summarizes the conversion rules applied in the metadata step.

Table 1 Conversion from YAPExIL metadata facet to MEF.

YAPExIL	MEF
title	Name; Title
module	Title
author	-
difficulty	Difficulty (map values)
status	-
type	Editor_kind (extend and map values)
keywords	-
event	-
platform	-

In the presentation step, the instruction files, Markdown and additional problem statements, and non-image embeddables are dropped out. Table 2 presents the conversion of YAPExIL presentation facet to MEF. A single problem statement, either HTML or text, goes to the `Description` field, plus a PDF statement which is saved in `PDF` field. Hence, at most two problem statements are kept. All embeddables with a supported image extension are stored as type `Image`. Finally, each `skeleton` is directly persisted in the equivalent type `Skeleton`, unless a `Template` for the same file extension exists in the evaluation facet.

Table 2 Conversion from YAPExIL presentation facet to MEF.

Element	YAPExIL	MEF
Instruction	M	-
Statement	M	1 (Description) + 1 (PDF)
Embeddable	M	M (only images)
Skeleton	M	M

In that case, `templates` are applied to solutions during the conversion, as they are not supported in Mooshak. The same happens with `solutions`, otherwise each `solution` is directly mapped into `Solution`. For both kinds of correctors, static and dynamic, the commands are merged into a large corrector and persisted into MEF, whereas the `library` files are ignored. In regards to tests, YAPExIL follows a tree structure, where intermediate nodes depict `testsets` and leaf nodes represent `tests`. In MEF, this tree is flattened into a list of `Test`. Table 3 outlines the key points of the conversion from the YAPExIL evaluation facet to MEF.

Table 3 Conversion from YAPExIL evaluation facet to MEF.

Element	YAPExIL	MEF
Library	M	-
DynamicCorrector	M	M (merged)
StaticCorrector	M	M (merged)
Solution	M	M
Template	M	applied to each Solution and Skeleton
TestSet	M	flattens to a list of Test
Test	M	M

The tools facet of YAPExIL is completely dropped out as its integration would require profound changes in MEF and, specifically, in Mooshak 2 with little gain.

4 Validation

The primary goal of the presented conversion is that the resulting MEF package can fully describe the same 7 types of programming exercises supported in YAPExIL. Traditional programming exercises only require a statement, tests, and a solution and, thus, support is kept. In addition to a `BLANK_SHEET` exercise, `BUG_FIX`, `EXTENSION`, `SPOT_BUG`, and `FILL_IN_GAPS` exercises require a skeleton, which is also part of MEF. An `IMPROVEMENT` exercise has the same requirements from the previous ones, but needs to evaluate other program metrics besides correctness, using either static or dynamic correctors. From Section 3, we see that corrector scripts are merged into a single script and, thus, there is no loss of value. Finally, the `SORT_BLOCKS` type has the particularity of requiring multiple skeletons (the blocks), which finds no barrier in MEF.

Besides the capability of representing each of the exercises' components, the user interface delivering the different types of exercises to the student must adapt according to the type of exercise. To this end, the format needs to keep either the type of exercise or an adequate editor to display. This is the role of `Editor_kind` of MEF, which has been extended with the values that require adaptations in the user interface (i.e., `FILL_IN_GAPS`, `SPOT_BUG`, and `SORT_BLOCKS`) beyond `CODE`.

Furthermore, an experiment to assess the library performance and functional correctness has been conducted with an example programming exercise package in YAPExIL format for each type of activity. All activities are variants, dependent on its type, of the same problem which challenges students to write a Python function that computes the derivative of $f(x) = x^b$ at point $x = m$, given b and m. The tests were developed using Jest, a JavaScript library for creating, running, and structuring tests for any project built using JavaScript or TypeScript. A total of 1000 runs has been performed on the test machine (Ubuntu 18.04.5 LTS 64 bits with i7-5820K CPU @ 3.30GHz x 12 and 32GB RAM) under the same conditions and their execution times registered separately for each type of activity. The conversion includes reading the archive from disk and write the new archive to disk. Figure 1 presents the a boxplot of the performance measurements. All tests were correct from the functional perspective.

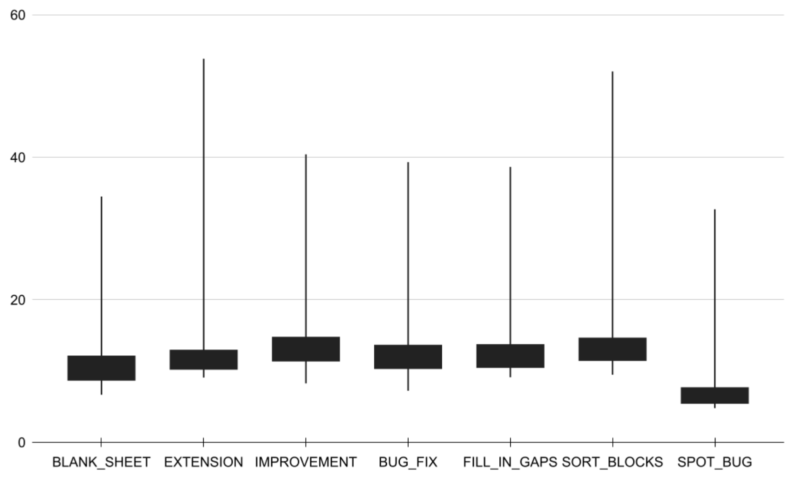


Figure 1 Boxplot of the performance measurements of the conversion for each type of activity (1000 runs). Times are in milliseconds.

On the one hand, the conversion of `SPOT_BUG` is the fastest as it contains only one test file with the line and column location of the bug, whereas all the others have six test cases. On the other hand, `IMPROVEMENT` and `SORT_BLOCKS` are the more time-consuming. The former for adding a static corrector to do the additional checks, the latter for having three skeletons rather than one.

5 Conclusions

Learning how to program relies on practice. Practice programming boils down to develop solutions to described problems, receive feedback on them, and improve them until they meet all requirements. Providing accurate and timely feedback to large classes is not a feasible task for teachers, who often resort to automated assessment tools to mitigate this issue. However, each tool typically adheres to its own programming exercise format and has no compatibility with other formats, precluding the share and reuse of exercises among institutions. Furthermore, the exercises proposed are always homogeneous, only varying the problem statement and requirements, i.e., the student is challenged to code a solution to a given problem statement, starting from an empty file.

This paper presents a JavaScript library to convert programming exercise packages adhering to YAPExIL, a JSON format supporting support for seven types of programming activities (blank-sheet, solution improvement, bug fix, gap filling, block sorting, and spot the bug), into packages complying with MEF, the internal format of Mooshak. The resulting package has no loss regarding the capability to describe each type of activity, requiring a minimal change to MEF (more options in an enumeration). Moreover, the library is already integrated into a collaborative web programming exercises editor, developed as open-source software [2], to allow exporting programming exercises directly as MEF packages, as presented in Figure 2.

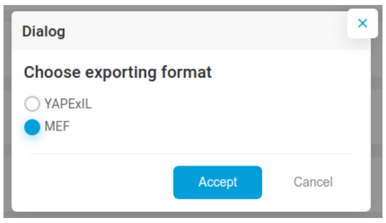


Figure 2 Export exercise dialog in FGPE AuthorKit.

The work described in this paper is part of the Framework for Gamified Programming Education (FGPE) project. This project includes the creation of about 500 programming exercises, most of which are already available online and are now supported by Mooshak 2. The consecutive next steps are to finish the gamified learning environment presenting such exercises to students.

References

1 José Paulo Leal and Fernando Silva. Mooshak: A web-based multi-site programming contest system. *Software: Practice and Experience*, 33(6):567–581, 2003.

2 José Carlos Paiva, Ricardo Queirós, José Paulo Leal, and Jakub Swacha. Fgpe authorkit – a tool for authoring gamified programming educational content. In *Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education*, ITiCSE '20, page 564, New York, NY, USA, 2020. Association for Computing Machinery. `doi:10.1145/3341525.3393978`.

3 José Carlos Paiva, Ricardo Queirós, José Paulo Leal, and Jakub Swacha. Yet Another Programming Exercises Interoperability Language (Short Paper). In Alberto Simões, Pedro Rangel Henriques, and Ricardo Queirós, editors, *9th Symposium on Languages, Applications and Technologies (SLATE 2020)*, volume 83 of *OpenAccess Series in Informatics (OASIcs)*, pages 14:1–14:8, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. `doi:10.4230/OASIcs.SLATE.2020.14`.

4 Ricardo Queirós and José Paulo Leal. Making programming exercises interoperable with PExIL. In José Carlos Ramalho, Alberto Simões, and Ricardo Queirós, editors, *Innovations in XML Applications and Metadata Management*, pages 38–56. IGI Global, 2013. `doi:10.4018/978-1-4666-2669-0.ch003`.

5 Tom Verhoeff. Programming task packages: Peach exchange format. *International Journal Olympiads In Informatics*, 2:192–207, 2008.

LeMe–PT: A Medical Package Leaflet Corpus for Portuguese

Alberto Simões
2Ai, School of Technology, IPCA, Barcelos, Portugal

Pablo Gamallo
Centro Singular de Investigación en Tecnoloxías Intelixentes (CiTIUS),
University of Santiago de Compostela, A Coruña, Spain

Abstract

The current trend on natural language processing is the use of machine learning. This is being done on every field, from summarization to machine translation. For these techniques to be applied, resources are needed, namely quality corpora. While there are large quantities of corpora for the Portuguese language, there is the lack of technical and focused corpora. Therefore, in this article we present a new corpus, built from drug package leaflets. We describe its structure and contents, and discuss possible exploration directions.

2012 ACM Subject Classification Computing methodologies → Information extraction; Computing methodologies → Language resources

Keywords and phrases drug corpora, information extractiom, word embeddings

Digital Object Identifier 10.4230/OASIcs.SLATE.2021.10

Supplementary Material *Dataset*: `https://github.com/ambs/LeMe`

Funding This project was partly funded by the project “NORTE-01-0145-FEDER-000045,” supported by Northern Portugal Regional Operational Programme (Norte2020), under the Portugal 2020 Partnership Agreement, through the European Regional Development Fund (FEDER), by Portuguese national funds (PIDDAC), through the FCT – Fundação para a Ciência e Tecnologia and FCT/MCTES under the scope of the project “UIDB/05549/2020”, and through the IACOBUS program, managed by GNP and AECT. In addition, it has received financial support from DOMINO project (PGC2018-102041-B-I00, MCIU/AEI/FEDER, UE), eRisk project (RTI2018-093336-B-C21), the Consellería de Cultura, Educación e Ordenación Universitaria (accreditation 2016-2019, ED431G/08, Groups of Reference: ED431C 2020/21, and ERDF 2014-2020: Call ED431G 2019/04) and the European Regional Development Fund (ERDF).

1 Introduction

Drug Package Leaflets (DPL), also known as Patient Information Leaflets (PIL), are documents that contain valuable information for patients about the characteristics of medicines. Each DPL provides useful information about a drug, mainly stating the active substance that constitutes the drug, listing side effects, describing interactions with other drugs, and describing the drug’s safety and efficacy, among other information.

In Portugal, DPLs are publicly accessible on the web, through the Portuguese National Authority of Medicines and Health Products website (Infarmed). This includes the documentation for all drugs currently accepted in the country, as well as some others that were previously approved but were late removed from the market.

Given the free nature of these documents and their relevant terminological content from different perspectives (drugs, illnesses, secondary effects), make this information highly valuable for Natural Language Processing (NLP), to be applied in different tasks, as information extraction, question answering solutions or machine translation, just to mention a few.

© Alberto Simões and Pablo Gamallo;
licensed under Creative Commons License CC-BY 4.0
10th Symposium on Languages, Applications and Technologies (SLATE 2021).
Editors: Ricardo Queirós, Mário Pinto, Alberto Simões, Filipe Portela, and Maria João Pereira; Article No. 10; pp. 10:1–10:10
OpenAccess Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

This contribution describes the creation of the LeMe–PT Corpus (Leaflets of Medicines), a corpus comprising more than a thousand of DPLs, and a set of experiments, including relation extraction and word similarity, performed to evaluate the relevancy of the corpus contents. In the next section related projects and works are introduced. Section 3 describes the corpus construction and its contents, and Section 4 presents some experiments for information extraction, and word embeddings creation and validation. Finally, we conclude with some future directions for the corpus use.

2 Similar Resources

In this section, we focus on describing some related work, including both other works where DPLs were used for corpus building, as well as other medical corpora, that were used for linguistic analysis and information extraction.

The EasyLecto system [17] aims to simplify DPLs by replacing the technical terms describing adverse drug reactions with synonyms that are easier to understand for the patients. EasyLecto simplifies the text of DPLs so as to improve their readability and understandability, and thereby allowing patients to use medicines correctly, increasing, therefore, their safety. This work is based in the Spanish version of the MedLinePlus Corpus.[1] The authors designed a web crawler to scrape and download all pages of the MedLinePlus web containing information on drugs and related diseases and, finally, stored the content into JavaScript Object Notation (JSON) documents. In order to evaluate the quality of their system, 306 DPLs were selected and manually annotated with all adverse drug reactions appearing in each document. The main problem of this approach is that it relies on external terminologies to provide synonyms. To overcome this limitation, the same authors described a new method in a more recent work [16], based on word embedding, to identify the more colloquial synonym for a technical term.

A similar work on readability of DPLs was previously reported [10]. In this work, the authors built a medical thesaurus of technical terms appearing in these documents, aligning them with a colloquial equivalent, easier to understand for the patients, in order to substitute the technical names by their colloquial equivalents, and making the DPLs easier to read and understand.

There are also corpus-based studies focused on the analysis of linguistic phenomena in DPLs. In [22], the author aimed at identifying keywords and frequent word sequences (recurrent n-grams) that are typical of this type of text. It was used a corpus with 463 DPLs and 146 summaries of drug characteristics written in English. Similar corpus-driven studies for Polish [23] and Russian [24] were reported. In [14], the building of a corpus comprised of medicine package leaflets medicines is described as well as its use as a teaching resource for Spanish–French translation in the medical domain.

There is very little literature on information extraction from DPLs, [1] is a Master's thesis work that describes a system to automatically extract entities and their relationships from Portuguese leaflets, with the aim to get information on dosage, adverse reactions, and so on. This work applies Named Entity Recognition (NER) and Relation Extraction (RE) techniques on text, to populate a medical ontology. Unfortunately, neither the corpus, software or the extraction results are freely available.

[1] See `https://www.nlm.nih.gov/medlineplus/spanish/`.

We should also highlight works on information extraction from medical corpora, not necessarily from DPLs. In [15] the authors describe a method to extract adverse drug reactions and drug indications in Spanish from social media (online Spanish health-forum), in order to build a database with this kind of information. The authors claim that health-related social media might be an interesting source of data to find new adverse drug reactions.

Still in Spanish, [21] aims to develop tools and resources for the analysis of medical reports and the extraction of information (entities and terms) from clinical documents. Regarding semantic information, the work reported in [19] describes the steps to create in-domain medical word embeddings for the Spanish using FastText and both the ScioELO database and Wikipedia Health as source of information [12].

For Portuguese, there is work on extracting information from medical reports [3]. The main issue with this kind of system is the availability of such corpora, as the data protections laws require the anonymization of the documents and, even after that process, hospitals would not allow the public release of such a corpus.

Finally, we should point out that there are very few corpora based on compiling leaflets which are available for free exploitation. One of the few examples is the Patient Information Leaflet (PIL) corpus, which consists of 471 English documents in Microsoft Word and HTML formats[2].

In Portuguese, the *Prontuário Terapêutico Online* is a website that allows several types of search on the Infarmed database of drug leaflets[3]. This site also includes some extra information that is not present directly in the leaflets, but was added to help doctors on their drug prescription.

It is also relevant to mention the availability of generic medical corpora. One of the most known is the European Medicines Agency (EMA) Parallel Corpora, available from the OPUS project [20]. From this corpus a set of related projects were developed, as a Romanian corpus [8], or the organization of information extraction tasks under the Cross Language Evaluation Forum (CLEF), as described in [6] and [9].

3 The Corpus

Our corpus was built with drug package leaflets obtained from the Infarmed[4] website[5] Given the interactive process required to download these documents, the DPLs download was performed manually, using as seed a list of drug active substances. Each active substance was searched in the website, and a random drug package was chosen (it was not given any priority to generic or original drugs). When different pharmaceutical forms were available, the less common was chosen.

On some situations, the document linked from the Infarmed website is available at the European Medicines Agency website. In these situations, the documents include a full report on the drug, performed tests, effects, and so on. At the end, in an appendix, these reports include a copy of the DPL[6]. So, in these situations, the document was truncated to include only this specific appendix.

[2] See `http://mcs.open.ac.uk/nlg/old_projects/pills/corpus/PIL/`.
[3] Available at `https://app10.infarmed.pt/prontuario/`.
[4] Infarmed, available at `http://infarmed.pt`, is the Portuguese National Authority of Medicines and Health Products.
[5] Note that these documents are copyrighted by the respective pharmaceutical company. We are just easing the access to these documents in a textual format.
[6] Some of these reports include different DPLs copies, accordingly with the various drug dosages available in the market.

The corpus include 1191 different package leaflets, referring to 1191 different active substances (some leaflets refer to compound active substances). The majority of these documents are divided into five to seven different sections. The most common are:

- what is the drug application, including sometimes its type;
- precautions the patient should take before using the drug;
- the usual dosage, depending the illness, age and other patient characteristics;
- the possible secondary effects and/or interactions with other drugs;
- how to store the package and other less relevant information.

For the documents obtained from the European Medicines Agency, they were automatically cleaned, removing the introductory report. Some still include different variants of the instructions, that will require manual cleanup. At the current version (v1.0), the corpus comprises about 3 000 000 tokens, from which about 2 650 000 are words, accounting to over 30 000 different word forms.

The corpus is available in a text file for each specific active substance and it is minimally annotated with XML-like tags:

- Title tags, dividing the different sections of the document. Most documents include only the five or six sections. A few do not follow this specific structure, and have more than ten sections.
- Item and Sub-item tags, annotating all lists automatically detected in the document.

We intend that new versions of the corpus include further annotations, namely on illnesses, drugs, secondary effects, and other relevant information. The next section describes the first steps towards the inclusion of this kind of information in the corpus.

4 Experiments

In this section we present some experiments performed with this corpus, presenting some directions for information extraction.

4.1 Regular-Expression based Information Extraction

One first experiment was performed to extract information about what is each substance. For that, the first section of each document was processed, trying to find two different kinds of relations:

- Hyponymy: referring to the medicine type. Examples of detected types are presented on Table 1. This relation was obtained for 1058 different substances. The extraction of this information is performed by the use of the following regular expressions:

```
que é \s+ uma? \s+ ([^.]+)
que é pertence.*? \s+ (?:por|d[eao]s?) \s+ ([^.]+)
```

  Note that these two regular expressions are applied in context, meaning they will be only activated in the proper section of the document.
- Condition or illness the medicine is adequate for, as shown in Table 2. This relation was obtained for 979 different substances. Follows a pair of examples of the different regular expressions used to extract this information:

```
tratamento \s+ ((?:\S+ \s+)?) d[aoe]s? \s+ ([^.]+)
(?:indicado|usado|utilizado) \s+ (?:para|n[ao]s?) \s+ ([^.]+)
```

These relations can be extracted with reasonable recall and high precision as the vocabulary used in this kind of document is quite controlled and the syntactic structures are recurrent. This can be comparable to the language used by lexicographers [18]. For instance, the relations mentioned above are extracted using six simple regular expressions. However, in some cases, this technique is extracting large sentences which should be reduced and simplified. Combining these simple text-mining techniques with some basic natural language processing techniques would allow for more compact extractions and higher quality data.

Given the amount of different possibilities to make the results better, at the current stage it was not performed any evaluation on precision or recall for these methods. Nevertheless, the manual annotation for some of these properties is planned, so that the corpus can also be used as an information retrieval test set.

Table 1 Examples of hyponymy relations extracted from LeMe–PT.

zolmitriptano *medicamentos denominados de triptanos*
zolpidem *medicamento de administração oral* | *medicamentos ansiolíticos, sedativos e hipnóticos*
valproato semisódico *anticonvulsivante*
valaciclovir *medicamentos designados de antivirais*
toxina botulínica A *relaxante muscular utilizado para tratar várias condições no corpo humano*
tramadol + dexcetoprofeno *analgésico da classe dos anti-inflamatórios não esteróides*

Table 2 Examples of conditions or illnesses extracted from LeMe–PT.

zolmitriptano *depressão* | *tratar as dores na enxaqueca*
zotepina *esquizofrenia, que tem sintomas como ver, ouvir ou sentir coisas que não existem*
tansulosina *sintomas do trato urinário inferior causados por um aumento da próstata*
tapentadol *dor crónica intensa em adultos*
tribenosido + lidocaina *hemorroidas externas e internas*

4.2 Words proximity using Word Embeddings

Some experiments were performed using Word2Vec [11]. More precisely, the corpus was pre-processed with the `word2phrases` script [13], which is shipped with the `word2vec` code, to create multi-word expressions and extracted word embeddings with the `word2vec` program, by training both a continuous bag of word model (CBOW) and a Skip-gram model, for a window size of 10 words, 15 iterations, and 300 dimension vectors.

Table 3 presents proximity terms for a set of words. For the first example, *alprazolam*, the list includes mostly other soothing drugs. For the second, *palpitações* [palpitations] the results are different kinds of heart rates dysfunctions. Finally, in the third column, *sonolência* [somnolence], the proximity terms are related to mental status, from soothing to tremors.

Table 3 Proximity terms obtained by Word2Vec (CBOW model).

alprazolam		palpitações		sonolência	
triazolam	0.843 623	aceleração	0.872 615	tonturas	0.803 442
diazepam	0.775 389	batimento cardíaco	0.857 808	sedação	0.801 227
alfentanilo	0.768 538	palpitações cardíacas	0.856 427	letargia	0.784 619
temazepam	0.762 558	batimento cardíaco acelerado	0.856 367	confusão mental	0.781 599
amissulprida	0.753 863	taquicardia	0.855 876	vertigens	0.761 623
midazolam	0.742 607	ritmo cardíaco lento	0.851 237	ataxia	0.747 047
tranquilizante	0.733 400	frequência cardíaca lenta	0.847 746	tremores	0.742 320
clonazepam	0.716 460	batimento cardíaco rápido	0.845 664	tremor	0.742 008
sedativo	0.716 204	acelerado	0.842 884	nervosismo	0.739 591
brotizolam	0.716 096	ritmo cardíaco rápido	0.838 991	coordenação	0.737 030

4.3 Word Embeddings Evaluation

In order to perform a basic evaluation task on the quality of the word embeddings generated from LeMe–PT corpus, an intrinsic evaluation was built by making use of a specific word similarity task, namely the outlier detection task [2, 5]. The objective is to test the capability of the embeddings to generate homogeneous semantic clusters. It consists of identifying the word that does not belong to a semantic class. For instance, given the set of words

$$S = \{lemon, orange, pear, apple, bike\},$$

the goal is to identify the word *bike* as an outlier of the class of fruits. One of the advantages of this task is that it has high inter-annotator agreement as it is easy to identify outliers when semantic classes are clearly defined.

To evaluate our embeddings model with the outlier detection task, five medical classes were built. Each one consists of eight words belonging to a specific class and eight outliers which do not belong to that class. The five classes are *analgesics*, *antidepressants*, *autoimmune diseases*, *respiratory diseases* and *pharmaceuticals*. The first four were elaborated by consulting specialized medical websites and the last one by choosing the first 8 drugs (in alphabetical order) from Infarmed. To give an example, Table 4 depicts the elaborated class of *antidepressants*.

As in this example, the five classes and their corresponding sets of outliers are unambiguous, thus there is no fuzzy boundary between class elements and outliers.

The outlier metric is based on a specific clustering method, called *compactness score*. Given a set of word elements $C = \{e_1, e_2, \ldots, e_n, e_{n+1}\}$, where $e_1, e_2, \ldots, e_n$ belongs to the same semantic class and e_{n+1} is the outlier, the compactness score compact(e) of an element $e \in C$ is defined as follows:

$$\text{compact}(e) = \frac{1}{n} \sum_{\substack{e_i \in C \\ e \neq e_i}} \text{sim}(e, e_i) \tag{1}$$

Table 4 Class of antidepressants and set of outliers.

Antidepressants	Outliers
imipramina	abiraterona
clomipramina	serotonina
amitriptilina	insónia
desipramina	tremores
nortriptilina	paracetamol
fluoxetina	doença
paroxetina	farmácia
citalopram	carro

An outlier e is detected if the value of compact(e) is lower than the compact$(e_i), \forall e_i \in C$, the scores of the words belonging to the class C. Two specific evaluation metrics are used: *accuracy* measures how many outliers were correctly detected by the system divided by the number of total tests. In [2], the authors also define *Outlier Position Percentage* (OPP) which takes into account the position of the outlier in the list of $n+1$ elements ranked by the compactness score, which ranges from 0 to n (position 0 indicates the lowest overall score among all elements in C, and position n indicates the highest overall score).

To compare the embeddings generated from LeMe–PT corpus with other models generated from generic Portuguese corpora, we also applied the outlier task to three pre-trained Embeddings from Inter-institutional Center for Computational Linguistics (NILC) [7]: NILC-Word2Vec, NILC-FastText, and NILC-Glove.[7]

The three models contain 300 dimensions. They were generated from a vast corpus with 1 395 926 282 word tokens, i.e. about 600 times larger than LeMe–PT. NILC-Word2Vec and NILC-FastText were trained with both the Skip-Gram and and Continuous Bag-Of-Words (CBOW) algorithms.

Table 5 shows the *accuracy* and *OPP* scores obtained with the compared embedding models with both Skip-Gram and CBOW algorithms (except for Glove as this is not applicable). The CBOW model derived from LeMe–PT clearly outperforms the other generic CBOW models. Concerning Skip-Gram, the differences are not so clear, although our model achieves the highest OPP value. These results suggest that the corpus built from DPLs is coherent and capable of generating competitive semantic models that are well adapted to the specific domain of medical leaflets.

5 Conclusions

In this document we present LeMe–PT, a corpus on drug package leaflets. It is freely available for download through a GitHub repository[8], and includes a larger number of documents in comparison with similar projects.

The documents are minimally annotated with a clear structure, allowing the extraction of information from different sections. Given its compilation process was manual and comprehensive, it includes drug leaflets from all medicine areas, and for every drug active substance currently in the Portuguese market[9].

[7] Available at `http://nilc.icmc.usp.br/nilc/index.php/repositorio-de-word-embeddings-do-nilc`.
[8] Please visit `https://github.com/ambs/LeMe`.
[9] About twenty active substances included only one specific package, which did not have its leaflet available.

Table 5 Outlier Position Percentage (OPP) and Accuracy of several embedding models on the outlier detection dataset with 5 classes.

Model	OPP		Accuracy	
	skip-gram	cbow	skip-gram	cbow
LeMe–PT (word2vec)	**87.81**	**96.88**	62.50	**87.5**
NILC-word2vec	83.44	80.00	62.50	50.00
NILC-fasttext	83.44	78.75	62.50	42.50
NILC-glove	83.12		57.5	

While some first experiments on the relevance of the corpus were performed, there are different directions one can explore in the future:

- Further annotation of the documents, namely on illnesses and secondary effects. This process can be bootstrapped automatically, but a throughout manual validation would be imperative. This would allow the use of the corpus for the evaluation of information extraction tools.
- Combine the simple text-mining techniques with basic natural language processing techniques, in order to obtain higher quality data.
- Apply Named Entity Recognition (NER) techniques to extract relevant entities, and use these entities with Open Information Extraction techniques [4] to improve recall and extract more types of relationships between the different kind of entities present in these documents.
- The availability of these documents should be cross-country, meaning the possibility to create parallel or, at least, comparable corpora with a high degree of terminological information.
- Given the relevance of the SNOMED Clinical Terms[10] ontology, it would be interesting to perform a concept alignment with this structure, which could result on a bootstrap mechanism for a Portuguese subset of SNOMED CT.

References

1 Bruno Lage Aguiar. *Information extraction from medication leaflets.* PhD thesis, Faculdade de Engenharia, Universidade do Porto, Porto, Portugal, 2010.

2 José Camacho-Collados and Roberto Navigli. Find the word that does not belong: A framework for an intrinsic evaluation of word vector representations. In *Proceedings of the ACL Workshop on Evaluating Vector Space Representations for NLP*, pages 43–50, Berlin, Germany, 2016.

3 Liliana Ferreira, António Teixeira, and João Paulo Silva Cunha. Medical information extraction in european portuguese. In *Handbook of Research on ICTs for Human-Centered Healthcare and Social Care Services*, pages 607–626. IGI Global, 2013. doi:10.4018/978-1-4666-3986-7.ch032.

4 Pablo Gamallo. An Overview of Open Information Extraction (Invited talk). In Maria João Varanda Pereira, José Paulo Leal, and Alberto Simões, editors, *3rd Symposium on Languages, Applications and Technologies*, volume 38 of *OpenAccess Series in Informatics (OASIcs)*, pages 13–16, Dagstuhl, Germany, 2014. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/OASIcs.SLATE.2014.13.

[10] See https://snomed.org.

5 Pablo Gamallo. Evaluation of Distributional Models with the Outlier Detection Task. In Pedro Rangel Henriques, José Paulo Leal, António Menezes Leitão, and Xavier Gómez Guinovart, editors, *7th Symposium on Languages, Applications and Technologies (SLATE 2018)*, volume 62 of *OpenAccess Series in Informatics (OASIcs)*, pages 13:1–13:8, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. `doi:10.4230/OASIcs.SLATE.2018.13`.

6 Lorraine Goeuriot, Liadh Kelly, Hanna Suominen, Leif Hanlen, Aurélie Névéol, Cyril Grouin, João Palotti, and Guido Zuccon. Overview of the clef ehealth evaluation lab 2015. In Josanne Mothe, Jacques Savoy, Jaap Kamps, Karen Pinel-Sauvagnat, Gareth Jones, Eric San Juan, Linda Capellato, and Nicola Ferro, editors, *Experimental IR Meets Multilinguality, Multimodality, and Interaction*, pages 429–443, Cham, 2015. Springer International Publishing.

7 Nathan Hartmann, Erick Fonseca, Christopher Shulby, Marcos Treviso, Jessica Rodrigues, and Sandra Aluisio. Portuguese word embeddings: Evaluating on word analogies and natural language tasks, 2017. `arXiv:1708.06025`.

8 Radu Ion, Elena Irimia, Dan Ştefănescu, and Dan Tufiș. ROMBAC: The Romanian balanced annotated corpus. In *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC'12)*, pages 339–344, Istanbul, Turkey, 2012. European Language Resources Association (ELRA). URL: `http://www.lrec-conf.org/proceedings/lrec2012/pdf/218_Paper.pdf`.

9 Liadh Kelly, Lorraine Goeuriot, Hanna Suominen, Aurélie Névéol, João Palotti, and Guido Zuccon. Overview of the CLEF eHealth evaluation lab 2016. In *Lecture Notes in Computer Science*, pages 255–266. Springer International Publishing, 2016. `doi:10.1007/978-3-319-44564-9_24`.

10 Fabian Merges and Madjid Fathi. Restructuring medical package leaflets to improve knowledge transfer. In *IKE: proceedings of the 2011 international conference on information & knowledge engineering*, Las Vegas, Nevada, 2011.

11 Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space, 2013. cite arxiv:1301.3781. URL: `http://arxiv.org/abs/1301.3781`.

12 Tomas Mikolov, Edouard Grave, Piotr Bojanowski, Christian Puhrsch, and Armand Joulin. Advances in pre-training distributed word representations. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*, Miyazaki, Japan, May 2018. European Language Resources Association (ELRA). URL: `https://www.aclweb.org/anthology/L18-1008`.

13 Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. In *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'13, page 3111–3119, Red Hook, NY, USA, 2013. Curran Associates Inc.

14 Manuel Cristóbal Rodríguez Martínez and Emilio Ortega Arjonilla. El corpus de prospectos farmacéuticos como recurso didáctico en el aula de traducción especializada francés-español. In Vicent Montalt, Karen Zethsen, and Wioleta Karwacka, editors, *Current challenges and emerging trends in medical translation. MonTI 10*, pages 117–140. Universidad de Alicante, 2018.

15 Isabel Segura-Bedmar, Santiago de la Peña González, and Paloma Martínez. Extracting drug indications and adverse drug reactions from Spanish health social media. In *Proceedings of BioNLP 2014*, pages 98–106, Baltimore, Maryland, June 2014. Association for Computational Linguistics. `doi:10.3115/v1/W14-3415`.

16 Isabel Segura-Bedmar and Paloma Martínez. Simplifying drug package leaflets written in spanish by using word embedding. *Biomedical Semantics*, 8 (45), 2017. `doi:10.1186/s13326-017-0156-7`.

17 Isabel Segura-Bedmar, Luis Núñez-Gómez, Paloma Martínez, and M. Quiroz. Simplifying drug package leaflets. In *SMBM*, 2016.

18 Alberto Simões, Álvaro Iriarte, and José João Almeida. Dicionário-aberto – a source of resources for the portuguese language processing. *Computational Processing of the Portuguese Language, Lecture Notes for Artificial Intelligence*, 7243:121–127, 2012. doi:10.1007/978-3-642-28885-2_14.

19 Felipe Soares, Marta Villegas, Aitor Gonzalez-Agirre, Martin Krallinger, and Jordi Armengol-Estapé. Medical word embeddings for Spanish: Development and evaluation. In *Proceedings of the 2nd Clinical Natural Language Processing Workshop*, pages 124–133, Minneapolis, Minnesota, USA, 2019. Association for Computational Linguistics. doi:10.18653/v1/W19-1916.

20 Jörg Tiedemann. Parallel data, tools and interfaces in opus. In Nicoletta Calzolari (Conference Chair), Khalid Choukri, Thierry Declerck, Mehmet Ugur Dogan, Bente Maegaard, Joseph Mariani, Jan Odijk, and Stelios Piperidis, editors, *Proceedings of the Eight International Conference on Language Resources and Evaluation (LREC'12)*, Istanbul, Turkey, May 2012. European Language Resources Association (ELRA).

21 Pilar López Úbeda. Reconocimiento de entidades en informes médicos en español. In *Proceedings of Doctoral Symposium of the 33rd Conference of the Spanish Society for Natural*, 2018.

22 Łukasz Grabowski. Register variation across english pharmaceutical texts: A corpus-driven study of keywords, lexical bundles and phrase frames in patient information leaflets and summaries of product characteristics. *Procedia - Social and Behavioral Sciences*, 95:391–401, 2013. doi:10.1016/j.sbspro.2013.10.661.

23 Łukasz Grabowski. On lexical bundles in polish patient information leaflets: A corpus-driven study. *Studies in Polish Linguistics*, 9(1), 2014.

24 Łukasz Grabowski. Distinctive lexical patterns in russian patient information leaflets: a corpus-driven study. *Russian Journal of Linguistics*, 23(3):659–680, 2019. doi:10.22363/2312-9182-2019-23-3-659-680.

Towards Automatic Creation of Annotations to Foster Development of Named Entity Recognizers

Emanuel Matos
IEETA, DETI, University of Aveiro, Aveiro, Portugal

Mário Rodrigues
IEETA, ESTGA, University of Aveiro, Aveiro, Portugal

Pedro Miguel
IEETA, DETI, University of Aveiro, Aveiro, Portugal

António Teixeira
IEETA, DETI, University of Aveiro, Aveiro, Portugal

Abstract

Named Entity Recognition (NER) is an essential step for many natural language processing tasks, including Information Extraction. Despite recent advances, particularly using deep learning techniques, the creation of accurate named entity recognizers continues a complex task, highly dependent on annotated data availability. To foster existence of NER systems for new domains it is crucial to obtain the required large volumes of annotated data with low or no manual labor. In this paper it is proposed a system to create the annotated data automatically, by resorting to a set of existing NERs and information sources (DBpedia). The approach was tested with documents of the Tourism domain. Distinct methods were applied for deciding the final named entities and respective tags. The results show that this approach can increase the confidence on annotations and/or augment the number of categories possible to annotate. This paper also presents examples of new NERs that can be rapidly created with the obtained annotated data. The annotated data, combined with the possibility to apply both the ensemble of NER systems and the new Gazetteer-based NERs to large corpora, create the necessary conditions to explore the recent neural deep learning state-of-art approaches to NER (ex: BERT) in domains with scarce or nonexistent data for training.

2012 ACM Subject Classification Information systems → Specialized information retrieval; Applied computing → Computers in other domains

Keywords and phrases Named Entity Recognition (NER), Automatic Annotation, Gazetteers, Tourism, Portuguese

Digital Object Identifier 10.4230/OASIcs.SLATE.2021.11

1 Introduction

Named entities are single-word or multi-word expressions that refer to specific individuals, such as people and organizations, or denote other concrete information such as postal and e-mail addresses or dates. They are expressed using specific patterns (ex: addresses, dates, e-mails) or composed by a sequence of nouns referring a single entity as, for instance, "António Guterres" or "The Secretary-General of the United Nations" which, in 2021, refer to the same person.

Automatic recognition of these named entities is the objective of Named Entity Recognition (NER) [20], a key step used in several natural language processing (NLP) tasks. Its goal is to identify single-word or multi-word expressions in texts and classify them using a set of categories that usually include names of that usually include names of organizations and people, locations, and time references. Due to this need to both identify and classify, some approaches split the task in two parts: first, detection of named entities; second, Named Entity Classification (NEC). The complete set of categories depends on the type of

© Emanuel Matos, Mário Rodrigues, Pedro Miguel, and António Teixeira;
licensed under Creative Commons License CC-BY 4.0

information to be detected. For instance, if you need to extract information about restaurants, the set of categories may include organizations (the name of the company), types of food, locations, service hours, among other.

The range of NLP tasks that benefit from NER include Information Extraction (IE) [29, 19], Business Intelligence [31], Forensics [2], Medicine [30] and question answering systems [6]. In general, detecting named entities prevents later steps of processing to break multi-word entities in segments that can lose the original meaning. In IE, the correct recognition of named entities is crucial, for example, to be able to detect in documents which entities are the subjects and objects. For example, a text: "Researcher Emanuel Matos at the University of Aveiro", a typical IE processing sequence is called entity detection (in this case "Emanuel Matos" and "Universidade de Aveiro"), classification (PERSON and ORGANIZATION, respectively), and extraction of relationship(s) between the identified entities.

Despite their potential and relevance for NLP, the performance of out-of-the-box NER systems is not always adequate for specific domains, preventing its wider adoption. Most systems provide methods to train and adapt them to concrete applications but this adaptation is usually limited to developers with in-depth knowledge of the area and/or having access to large amounts of annotated texts. To obtain or create annotated data that associates words or sequences of words to a type of entity is essential and in general a major problem. Another challenge is to train existing systems using these data. Not just expressing data in the format(s) accepted by existing systems but also doing so in a way non-developers feel comfortable with.

Aiming to mitigate the problems highlighted above, the following main objectives were adopted for the work presented in this paper:

1. Develop processes to simplify the creation of NER systems for new domains, starting by the creation of the needed annotated data;
2. Make NER deployment as easy as possible in order to used by non specialists, contributing to breaking existing usage barriers thus fostering wider adoption of such systems.

After this initial section establishing motivation, problem(s), relevance and main objectives, the rest of the paper is structured as follows: Section 2 presents a brief view of current approaches to NER, NER systems for Portuguese, as well as information regarding sets of categories considered in previous work; Section 3 presents the proposed solution, involving an ensemble of NERs; initial representative results are presented in Section 4; Section 5 concludes the paper with some discussion of the results and pointing to several lines of research for the future.

2 Related Work

As many other natural language processing tasks, Named Entity Recognition (NER) can be achieved either using rule based or data driven (machine learning) approaches. Some approaches are better adjusted to a specific scenario depending on the entities to detect. To detect entities with well-defined and mandatory surface patterns, such as e-mail addresses, postal codes, telephone numbers, among other, rule-based approaches allow to code compact patterns that are quite effective to detect (almost) all entities. To detect entities whose patterns may differ in format (capital letters) without impeding human recognition of the entity, such as names of people, streets, organizations, etc., data-based approaches are best in handling these more flexible patterns, we can recognize more easily the surrounding contexts checking the highest frequency at which these entities are likely to appear. In software applications, the named entity task often uses a mix of both approaches to detect entities.

2.1 Current Approaches to NER

Considering the mentioned two main types, the most relevant approaches are:

Rule-based NER – covering both systems based in patterns and in lists (the so-called Gazetteers).

Surface Patterns – Surface patterns NER are usually implemented using regular expressions, or regex for shorthand, which are sequences of characters that specify search patterns. They are easy to implement but are quite sensitive to errors as they can break patterns and thus cause detection failures. Also, improvements come with increasing complexity as rules can interact, and their order can be meaningful, which makes more difficult to manage the rule set [9]. One of the main challenges of creating handcrafted rules is that it can be very time consuming to compile a comprehensive set of rules when target entities do not have well-defined and mandatory surface patterns. It can also be difficult to port the solution to other application contexts. For entities with well-defined surface patterns this approach is often easy to implement and provides reliable results.

Gazetteers – A gazetteer is a geographical dictionary or directory. In NLP context, the term gazetteer was further extended and now means a list of items that often include organizations, people names, alongside geographical entities such as cities or landmarks. Approaches using gazetteers can be as simple as matching candidate portions of text against the lists and having the decision just based on the existence in the list. More sophisticated methods of using gazetteers include using them as triggers in which a keyword can be used to find an entity (for example, Ms. can be used to identify that the next statement is a person), This approach is easy and can get rather good results. Unfortunately, the creation and maintenance of the lists can be a hard and tedious process and it also has problems with ambiguity (ex: gate can be an object or a name of a person depending on the context).

NERs using Machine learning – Machine learning methods are more flexible to adapt to distinct contexts provided that exists enough data about the target context. Diverse machine learning methods have been applied to NER. They can be categorized in three main branches that have distinct needs of training data: (1) supervised learning, (2) unsupervised learning and (3) reinforcement learning.

The supervised learning methods use a training set (a corpus) that was already manually labeled by experts. The unsupervised learning method consumes an untrained data set and extract patterns from it, contrary to the supervised method this one doesn't need a labeled training set. The reinforcement learning method uses agents to learn polices [12] that can be used to label an untrained data set. These agents are trained using a reward system. Machine learning methods that were successfully applied to NER over the years include:

Hidden Markov Model (HMM) – HMM is a statistical Markov model in which the system being modeled is assumed to be a Markov process with unobservable, hidden, states. There is another process, visible, whose behavior depends on the underlying hidden process. The goal is to learn about the hidden process by observing the visible one. NER is considered as a classification problem where the named entity class is the hidden part and the textual form the visible one. The goal is to decide which word is part of some name or not part of any name, and only one label can be assigned to a word in each context. Therefore, the model assigns to every word either one of the desired classes or a label representing none of the desired classes.

Support Vector Machines (SVM) – SVM are supervised learning models with associated learning algorithms that analyze data for classification and regression analysis. Support vectors are data points that are closer to the decision boundaries that help classify the data points. In NER, SVM classifies each word, using its context, to one of the classes that represent region information and named entity's semantic class.

Conditional Random Field (CRF) – In CRF, the prediction implements dependencies between predictions. In NER, a linear chain CRF the decision if a word belongs to a named entity class, or not, depends only the word itself and on the information of the previous word. The CRF approach utilizes conditional probability to train untrained data using a trained data set.

Recursive Neural networks – Most neural-based models for NER are based on some sort of Long Short Term Memory (LSTM). LSTM are recursive neural networks in which the hidden layer updates are replaced by purpose-built memory cells. As a result, they may be better at finding and exploiting long range dependencies in data [16, 18]. Bidirectional LSTM are amongst the best performers and in these, word and character embeddings are passed through a left-to-right LSTM and a right-to-left LSTM. The outputs are combined to produce a single output layer. In the simplest method, this layer can then be directly passed onto a softmax that creates a probability distribution over all NER tags, and the most likely tag is chosen.

Transfer learning – Transfer learning reuses pre-trained models in order to perform a different task. It's very popular as it makes possible training deep neural networks with small amounts of data. In NER it was used, for example, to develop NERs for novel types of entities [28].

Transformers – Transformers [34], introduced in 2017, are a deep learning model based on the attention mechanism designed to handle sequential input data, such as natural language. Unlike RNNs, Transformers do not require data to be processed in order allowing much more parallelization and, because of that, training with huge datasets. This created the conditions for the development of pre-trained systems such as BERT (Bidirectional Encoder Representations from Transformers) and GPT (Generative Pre-trained Transformer) [24]. Transformers demonstrated their superior efficiency in the recognition of named entities and in a variety of other classification tasks. A variety of state-of-the-art NER systems were developed adopting BERT for different domains and languages (ex: [32]).

2.2 Examples of Entities

Examples of entities considered over the years are presented in Table 1. The number of entities was quite reduced in initial datasets, only 4 in CONLL 2003. Despite increase the set continues to be limited, in most cases, to no more than 10.

2.3 NER systems for Portuguese

Several systems can detect and tag Named Entities in texts in Portuguese, being particularly relevant the following:

Linguakit [11] – A Natural Language Processing tool containing several NLP modules, developed by ProLNat@GE Group[1], CiTIUS, University of Santiago de Compostela. It can process 4 languages: Portuguese, English, Spanish, and Galician. One of its modules

[1] `http://gramatica.usc.es/pln/`

Table 1 Entities integrating a representative selection of datasets.

Dataset	N	Entities
CADEC	5	Adverse Drug Reaction (ADR), Disease, Drug, Finding, Symptom
CONLL 2003	4	LOC (location), ORG (organization), PER (person), MISC (miscellaneous)
i2b2 Challenges	16	Username, City, Patient, ZIP, Doctor, Country, Hospital, Profession, Phone, State, Street, Medical Record, Date, Organization, Age, IDnum
MITRestaurant	8	Amenity, Cuisine, Dish, Hours, Location, Price, Rating, Restaurant Name
MUC-6	6	PER (person), LOC (location), ORG (organization), PCT (percentage), MON (month), DAT (date)
NIST-IEER	10	MUC + TIM (time), DUR (duration), CAR (cardinality), MEA (measure)
GUM	11	Person, Place, Organization, Quantity, Time, Event, Abstract, Substance, Object, Animal, Plant
NAACL 2019	6	Organization, Person, Location, Geopolitical, Facility, Vehicles
re3d	10	Document Reference, Location, Military Platform, Money, Nationality, Organisation, Person, Quantity, Temporal, Weapon

is a NER tagger that classifies entities into four classes: person, organization, local or miscellaneous. The system employs lists of known entities (gazetteers) and a set of rules that allow disambiguating entities that appear in more than one list (which can be, for example, person or place).

FreeLing [23] - An open-source language analysis that, besides NER, it also implements tokenization, MSD-tagging, syntactic parsing, and lemmatization. The supported languages are Catalan, English, Galician, Italian, Portuguese, and Welsh.

NLPyPort [10] – A Python development focused on Portuguese based on NLTK. Adopting pre-existing resources and their adaptations, it provided better performance than the existing Python alternatives in tokenization, PoS Labeling, stemming and NER.

StanfordNLP [27] – Also Python based, it contains useful tools to: convert a string containing natural language text into lists of phrases and words; perform morphological analysis; obtain syntactic dependence structure (for more than 70 languages, using the Universal Dependencies formalism); perform constituent analysis and co-reference resolution, detect, and classify Named Entities. In addition, it can call the CoreNLP Java package.

AllenNLP [13] – A multilingual deep learning Python library for NLP developed by the Allen Institute for Artificial Intelligence, one of the leading research organizations of Artificial Intelligence. Using AllenNLP to develop a model is much easier than building a model by PyTorch from scratch. Not only it provides easier development but also supports the management of the experiments and its evaluation after development. AllenNLP has the feature to focus on research development. More specifically, it is possible to prototype the model quickly and makes easier to manage the experiments with a lot of different parameters. In Allennlp, NER predictions are based in pretrained models. There are many types of pretrained [2].

Representative recent examples of NER for Portuguese are briefly presented in Table 2.

[2] In `https://github.com/allenai/allennlp-hub/blob/master/allennlp_hub/pretrained/allennlp_pretrained.py` some pretrained models can be found.

Table 2 Recent representative Work in NER for Portuguese.

Ref.	Language	Domain	Technics
[25]	Multilingual (inc. Portuguese)	Webpages, Newspaper, Several genres	HMM, CRF
[22]	Brazilian Portuguese	Legal	LSTM-CRF
[26]	Portuguese	General	CRF+LG
[21]	European Portuguese	Clinical	BiLSTM-CRF
[8]	European Portuguese	Sensitive Data	Rule-based, CRF, Random Fields and BiLSTM
[32]	Portuguese	HAREM Golden collection	BERT, CRF

In [25], the authors applied multiple techniques to different datasets to create an "out-of-box" comparisons. Results revealed Stanford CoreNLP [F-measure=56.10%] as the best system and NTLK [F-measure=30.97%] as the worst.

The LeNER-Br system [22], presented in 2018, was developed for Brazilian legal documents The Paramopama training data set was used to train LSTM-CRF models with F1 scores of 97.04% and 88.82% for Legislation and judicial entities. According to the authors, the results showed the feasibility of the NERs for judicial applications.

Aiming to recognize named entities in many textual genres, including genres that differ from those for which you were trained, Pirovani and coworkers [26], in 2019, adopted a hybrid technique combining Conditional Random Fields with a Local Grammar (CRF+LG), that they adapted to various textual genres in Portuguese, according to the task of Recognition of Named Entities in Portugal at IberLEF 2019.

In 2019, Lopes and coworkers [21], addressed NER for Clinical data in Portuguese with BiLSTMs and word embeddings, the state-of-the-art model at the time obtaining F1-scores slightly higher than 80% and equivalent results for both Precision and Recall. The data set was pre-processed by NLPPort [10] and processed by BiLSTM-CRF and CRF for comparison. BiLSTM was superior in all comparisons for "In-Domain" models.

In a work published in 2020, NER was applied to sensitive data Discovery in Portuguese [8], being used in the process of protecting sensitive data. A component was developed to extract and classify sensitive data, from unstructured text information in European Portuguese combining several techniques (lexical rules, machine learning algorithms and neural networks). The rule-based approaches were used for a set of specific classes (ex: NumIdentificacaoCivil). For the other classes of entities, CRF, Random Fields and BiLSTM were used. The datasets used for training and testing were HAREM Golden Collection, SIGARRA News Corpus and DataSense NER Corpus. This validation was carried out with the project's stakeholders. Although the global results with the use of lexicon-based models were inferior to the current state of the art, for the TEMPO and VALOR entities the results were superior to those obtained with other methodologies.

A first use of BERT in NER for Portuguese appeared in 2020 [32]. In this work, Portuguese BERT models were trained and a BERT-CRF architecture was employed, combining transfer capabilities of BERT with the structured CRF forecasts. Pre-training of BERT used brWac corpus, which contains 2.68 billion tokens from 3.53 million documents and is the largest Portuguese open corpus to date. Training of the NER model was done with First HAREM. Tests with MiniHAREM dataset surpassed the previous state art (BiLSTM-CRF+FlairBBP), despite being trained with much less data.

From the selected representatives of recent developments of NER for Portuguese it is clear that: (1) the target domains are quite diverse, being different for all the selected references; (2) the set of techniques applied is also diverse, often being adopted Machine

Learning methods and tools, including more recent ones such as LSTM and BERT; (3) NER for Portuguese continues to be a relevant and active area, with developments aligned with state-of-the-art evolution; (4) there are signs of expansion of application areas/domains.

3 Proposed solution – NER without annotated data using an Ensemble of NERs

To provide annotated corpora for development of new NER systems we propose to explore the combined use of several existing NER systems and resources capable of providing information regarding entities and their types, for example DBpedia [3, 4].

3.1 Overview of the system

The complete system derives directly of the main idea of the proposal – explore an ensemble of NERs – and consists of 4 phases, as represented in Fig. 1, that will be described in the following sections.

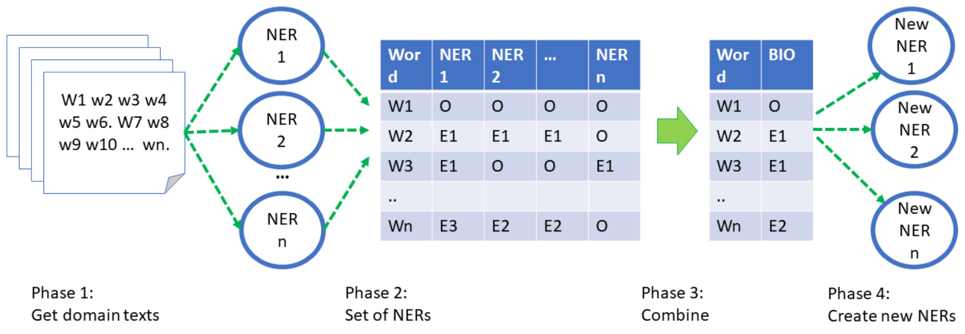


Figure 1 Overview of the proposed process for automatic tagging of named entities by using and Ensemble of NERs, showing its 4 phases.

3.2 Phase 1: Obtaining documents for the domain

For an initial proof-of-concept related to tourism were selected. A set of sources were selected manually, and the text scrapped using the Scrapy library for retrieving documents and BeautifulSoup library for getting document data. One of the main sources selected was Wikivoyage [35]. Based on the tool's ease of use concept, the manual extraction option, without advanced features, was implemented to adjust expectations regarding the development of the work and evaluate the capture properties of possible entities without advanced techniques. The option for Tourism domain resulted from our previous work in this domain [33], selected by the high potential of automatic information extraction to provide relevant information to several domain stakeholders (e.g., hotel managers).

3.3 Phase 2: Application of an Ensemble of NERs

From the NERs capable of processing texts in Portuguese, a subset was selected covering the main types. As representative of rule based and gazetteer approaches was selected Linguakit [11]. Allen NLP [13] was selected as a representative of a machine learning approach. As 3rd NER of the ensemble a system based in DBpedia was selected.

For integration of Linguakit in the system a simple Python wrapper script was created to invoke the Perl implementation and load and process its output.

A Python script was also created for the NER based in Allen NLP, but in this case using allennlp Python library [1] `Predictor` class. Publicly available modules were used [3].

The DBpedia-based NER was developed by the authors in Python using the SPARQL-Wrapper library [17] to make SPARQL queries to DBpedia endpoint. The overall process is presented as pseudo-code in Algorithm 1.

Algorithm 1 DBpedia-based NER.

```
input  : text file, maxl =max length of word sequences
output : two column dataframe with word and tag
1  read cache from file;
2  /* read text, tokenize and create dataframe with 2 columns (1st with
      words, 2nd with tag initialized to "O") and words list           */
3  dataframe , words_list ← init_from_file(filename);
4  for $l \leftarrow 1$ to maxl do
5      for $pos \leftarrow 0$ to $len$(words_list) $- l + 1$ do
6          word_seq ← words_list [pos ]+ .. + words_list [pos +l-1] ;
7          dbpedia_result =query_DBpedia (word_seq) ;
8          selected_result = postprocess (dbpedia_result) ;
9          tag =get_super_class (selected_result) ;
10         add tag to dataframe [pos ];
11         add (word_seq, tag) to cache;
12     end
13 end
14 save dataframe to CSV file;
15 save cache to JSON file;
```

All 3 NERs save their output as CSV files to allow access to all intermediate processing results and simple communication with the following phases.

3.4 Phase 3: Combination of the outputs of the Ensemble of NERs

The Phase 3 goal is to create new tag candidates from the tags produced by the 3 NERs. For the initial proof-of-concept version, two decision strategies were implemented: **Winner Take All** (WTA) which assigns to each word the most common tag, augmenting the confidence in the tags; **Entity detection** (ENTITY), that tags words as only Entity or not, producing annotated data useful to train entity detectors (not including classification of the entity).

3.5 Phase 4: Exploration of the results

The obtained annotated data from previous phase can be explored in several ways: analyzed in terms of agreement among NERs; evaluated against manually annotated data; Named Entities annotated extracted and used to create Gazetteer-based NERs; used to supervised training of data-driven machine learning NERs.

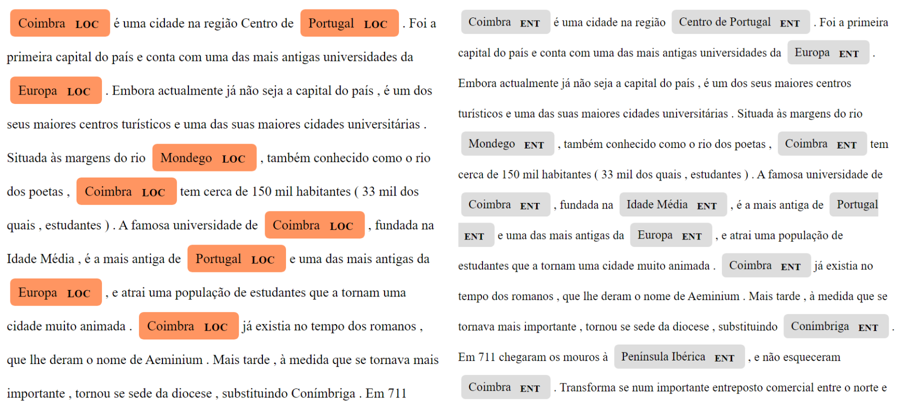
[3] `https://storage.googleapis.com/allennlp-public-models/ner-model-2020.02.10.tar.gz`

4 Results

In this section are reported the initial results obtained with the proposed system, starting with representative examples of obtained annotations based on combining the output of the 3 NERs.

4.1 Examples of obtained annotations

Illustrations of the obtained results with the ENTITY and WTA strategies for a text example are presented in Fig. 2.



Coimbra LOC é uma cidade na região Centro de Portugal LOC . Foi a primeira capital do país e conta com uma das mais antigas universidades da Europa LOC . Embora actualmente já não seja a capital do país , é um dos seus maiores centros turísticos e uma das suas maiores cidades universitárias . Situada às margens do rio Mondego LOC , também conhecido como o rio dos poetas , Coimbra LOC tem cerca de 150 mil habitantes (33 mil dos quais , estudantes) . A famosa universidade de Coimbra LOC , fundada na Idade Média , é a mais antiga de Portugal LOC e uma das mais antigas da Europa LOC , e atrai uma população de estudantes que a tornam uma cidade muito animada . Coimbra LOC já existia no tempo dos romanos , que lhe deram o nome de Aeminium . Mais tarde , à medida que se tornava mais importante , tornou se sede da diocese , substituindo Conímbriga . Em 711

Coimbra ENT é uma cidade na região Centro de Portugal ENT . Foi a primeira capital do país e conta com uma das mais antigas universidades da Europa ENT . Embora actualmente já não seja a capital do país , é um dos seus maiores centros turísticos e uma das suas maiores cidades universitárias . Situada às margens do rio Mondego ENT , também conhecido como o rio dos poetas , Coimbra ENT tem cerca de 150 mil habitantes (33 mil dos quais , estudantes) . A famosa universidade de Coimbra ENT , fundada na Idade Média ENT , é a mais antiga de Portugal ENT e uma das mais antigas da Europa ENT , e atrai uma população de estudantes que a tornam uma cidade muito animada . Coimbra ENT já existia no tempo dos romanos , que lhe deram o nome de Aeminium . Mais tarde , à medida que se tornava mais importante , tornou se sede da diocese , substituindo Conímbriga ENT . Em 711 chegaram os mouros à Península Ibérica ENT , e não esqueceram Coimbra ENT . Transforma se num importante entreposto comercial entre o norte e

Figure 2 NER Tagging of text from `https://pt.wikivoyage.org/wiki/Coimbra`. WTA results at left, ENTITY results at right.

The results obtained with a text example are presented in Fig. 2 for the strategies ENTITY and WTA. By combining the entities marked by the 3 annotators (at bottom right) it is possible to detect entities such as "Centro de Portugal" or "Península Ibérica". WTA strategy achieves a lower recall caused by Linguakit and Allen NERs (almost) only detecting PERSON and LOCATION, with large disagreements among them. AllenNLP seems to favor PERSON and classifies, for instance, "Praça Barão da Batalha" as PERSON while Linguakit correctly classifies it as LOCATION. The major contributor for higher recall using the ENTITY strategy is the DBPedia-based NER, frequently contributing with the tags necessary to mark text segments as ENTITY.

4.2 NERs output analysis

The outputs of the 3 NERs were compared with the outputs of the Ensemble of NERs (WTA and ENTITY) for a better understanding on their behavior, and thus complementing the qualitative analysis of the annotation results. The analysis started by looking both confusion matrices and computing a simple similarity measure (Jaccard similarity calculated not considering the cases where "O" BIO tag was assigned by the NERs being compared). Examples of confusion matrices obtained for the same text used in Fig. 2 are presented in Fig. 3, and the results for Jaccard similarity in a set of 39 texts are presented in Table 3.

The confusion matrices and the Jaccard similarity show that: (1) the 3 systems produce quite dissimilar results, being the higher Jaccard similarity only 0.422 (for Linguakit versus DBPedia-based NERs); (2) many of the entities marked as LOCATION by Linguakit and DBPedia are tagged as PERSON by AllenNLP; (3) DBPedia NER detects a richer set of

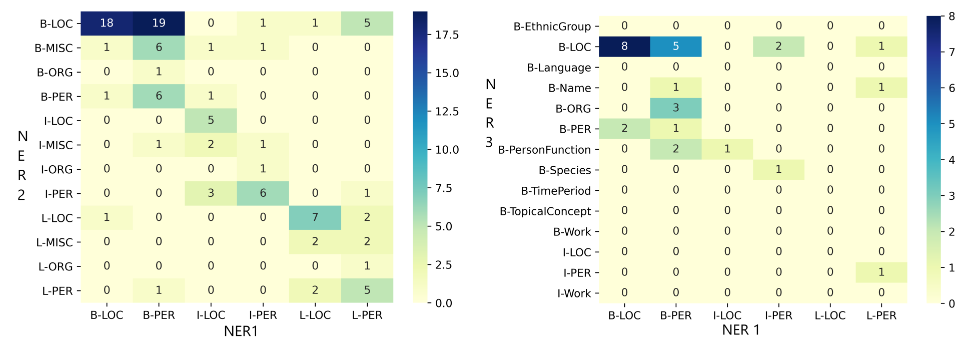


Figure 3 Confusion matrices for the tags assigned by the 3 NERs to the text `https://pt.wikivoyage.org/wiki/Amarante`: At left, AllenNLP (NER1) vs Linguakit (NER2), right, AllenNLP vs DBpedia NER (NER3).

Table 3 Jaccard similarity among NER tags for a set of 30 Wikivoyage texts. Individual results for a sample of the texts is presented as well the average (M) and standard deviation for the complete set of texts (at the bottom of the table).

	NER1				NER2			NER3		WTA
	NER2	NER3	ENT	WTA	NER3	ENT	WTA	ENT	WTA	ENT
Coimbra	0.223	0.180	0.613	0.429	0.387	0.709	0.705	0.256	0.690	0.397
Luanda	0.480	0.696	0.667	0.804	0.517	0.753	0.830	0.333	0.792	0.580
Roma	0.318	0.667	0.705	0.667	1.000	0.795	0.733	0.034	1.000	0.341
M	0.346	0.336	0.632	0.579	0.422	0.631	0.649	0.431	0.750	0.512
STD	0.201	0.214	0.105	0.162	0.219	0.193	0.146	0.253	0.170	0.140

entities than the other two; (4) WTA tags are more similar to the DBPedia-based NER (M=0.75), demonstrating its usefulness; (5) the maximum Jaccard similarity for ENTITY is just 0.63, for the comparisons with AllenNLP and Linguakit NERs, which can be interpreted as the contribution of the several NERs to ENTITY annotations.

4.3 New NERs

As stressed before, to train data-driven NERs such as the Hybrid LSTM-CRF are necessary substantial amounts of tagged data. To contribute to the existence of the needed data it is crucial to tag as much text as possible and as fast as possible. For that, two new NER taggers were created to profit from the combined results of the ensemble of NERs: one is a fast implementation of a Gazetteer and the other one resorts to Finite State Transducers (FSTs).

The first, a **simple Gazetteer NER**, was developed by the authors using Bloom Filters [5] to store information about the entities. Separate filters were used for each number of words in an entity, i.e., single word entities in one filter, 2-word entities in another, etc.

The second annotator, an FST-based NER, was implemented using the library `pynini` [14] and adapting one of the examples in [15]. Briefly, the process consists of:

1. Construct transducers which insert the left and right tags.
2. Build a substitution transducer by concatenating these transducers to the left and right of the deterministic Finite State Automata (FSA) we are trying to match. Its definition needs: the left context for the substitution; the right context; and an FSA representing the alphabet of characters we expect to find in the input.

3. Create the substitution transducer which passes through any part of a string which does not match.
4. Apply to a string. The simplest way to do so is to compose a string, apply it on the transducer, and then convert the resulting path back into a string with:
   `output = pynini.compose(input_string, rewrite).string()`

All word sequences marked as ENTITY were extracted from a corpus and used as lists. As an initial experiment, the corpus consisted of the 36 texts from Wikivoyage used to calculate the Jaccard similarities. The process resulted in a list with 379 entries, 176 with 1 word, 83 of 2 words, 59 of 3 words, etc. Both Gazetteer-based NERs were tested in texts outside the corpus used to extract the list. Fig. 4 presents the results obtained for a text sample.

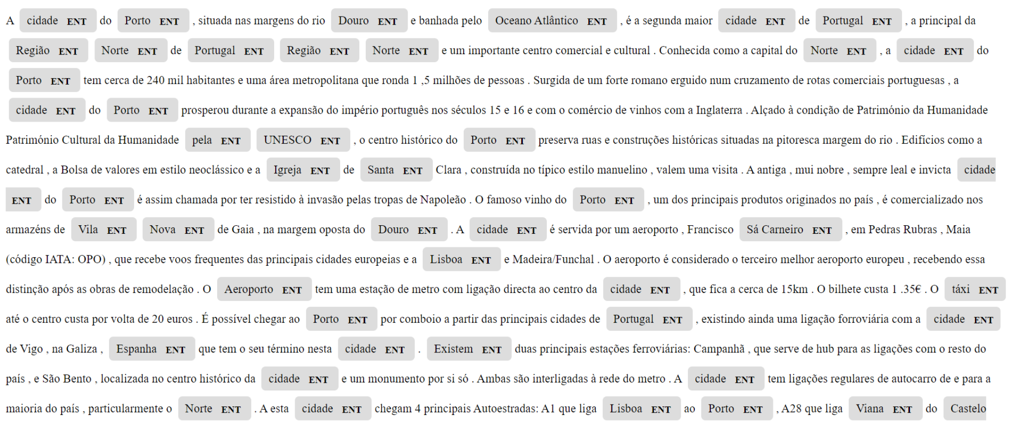

A cidade ENT do Porto ENT , situada nas margens do rio Douro ENT e banhada pelo Oceano Atlântico ENT , é a segunda maior cidade ENT de Portugal ENT , a principal da Região ENT Norte ENT de Portugal ENT Região ENT Norte ENT e um importante centro comercial e cultural . Conhecida como a capital do Norte ENT , a cidade ENT do Porto ENT tem cerca de 240 mil habitantes e uma área metropolitana que ronda 1 ,5 milhões de pessoas . Surgida de um forte romano erguido num cruzamento de rotas comerciais portuguesas , a cidade ENT do Porto ENT prosperou durante a expansão do império português nos séculos 15 e 16 e com o comércio de vinhos com a Inglaterra . Alçado à condição de Património da Humanidade Património Cultural da Humanidade pela ENT UNESCO ENT , o centro histórico do Porto ENT preserva ruas e construções históricas situadas na pitoresca margem do rio . Edifícios como a catedral , a Bolsa de valores em estilo neoclássico e a Igreja ENT de Santa ENT Clara , construída no típico estilo manuelino , valem uma visita . A antiga , mui nobre , sempre leal e invicta cidade ENT do Porto ENT é assim chamada por ter resistido à invasão pelas tropas de Napoleão . O famoso vinho do Porto ENT , um dos principais produtos originados no país , é comercializado nos armazéns de Vila ENT Nova ENT de Gaia , na margem oposta do Douro ENT . A cidade ENT é servida por um aeroporto , Francisco Sá Carneiro ENT , em Pedras Rubras , Maia (código IATA: OPO) , que recebe voos frequentes das principais cidades europeias e a Lisboa ENT e Madeira/Funchal . O aeroporto é considerado o terceiro melhor aeroporto europeu , recebendo essa distinção após as obras de remodelação . O Aeroporto ENT tem uma estação de metro com ligação directa ao centro da cidade ENT , que fica a cerca de 15km . O bilhete custa 1 .35€ . O táxi ENT até o centro custa por volta de 20 euros . É possível chegar ao Porto ENT por comboio a partir das principais cidades de Portugal ENT , existindo ainda uma ligação forroviária com a cidade ENT de Vigo , na Galiza , Espanha ENT que tem o seu término nesta cidade ENT . Existem ENT duas principais estações ferroviárias: Campanhã , que serve de hub para as ligações com o resto do país , e São Bento , localizada no centro histórico da cidade ENT e um monumento por si só . Ambas são interligadas à rede do metro . A cidade ENT tem ligações regulares de autocarro de e para a maioria do país , particularmente o Norte ENT . A esta cidade ENT chegam 4 principais Autoestradas: A1 que liga Lisboa ENT ao Porto ENT , A28 que liga Viana ENT do Castelo

Figure 4 Example of text (about Porto) annotated with the Gazetteer-based NER created with lists derived from the automatic annotations obtained with the proposed system.

The Figure shows clearly that the Gazetteer-based NER could recognize several named entities. A major problem is clear, the approach is not capable of handling continuous or close parts of an entity (ex: "cidade do Porto") and fails to recognize several entities (ex: "Vigo").

5 Conclusion

Aiming at making possible and simple creation of new NERs for domains without annotated data available, the problem of automatic annotation of Named Entities is addressed in this paper, being proposed the combined used of an ensemble of NER systems. The emphasis was placed on creating annotated data based on several NERs. A first proof-of-concept of the proposed method was implemented with 2 existing NER systems (AllenNLP and Linguakit) and a DBpedia-based NER developed by the authors.

The main contributions of this paper are: (1) the method based on an ensemble of NERs and combination of their outputs; (2) start of a new annotated corpus for experiments in NER for Tourism domain; (3) fast new Gazetteer-based NERs.

The initial results revealed that combining the outputs of the NERs ensemble has potential to both extend the set of entities (limited in systems as AllenNLP) and contribute to reduce the difficulties distinguishing some entities (ex. AllenNLP and Linguakit present high degree of disagreement for PERSON and LOCATION).

With the combination strategy providing a 1-class annotation, designated as ENTITY, the proposed system could detect an interesting set of Named Entities in texts related to Tourism that were used to create initial versions of new Gazetteer-based NERs. The other combination strategy, WTA, is useful to create higher confidence annotations for a set of entities, such as those handled by Linguakit.

Future work

As the presented work is both a first step and an initial proof-of-concept, the future work is rich and covering distinct lines of research, the most relevant being:

- **Improve the NERs integrating the Ensemble:** Improve the NERs integrating the ensemble: Particularly improve the performance of DBpedia-based NER, both in augmenting its speed (ex: with a local Virtuoso server) and improving the decision of Tags to keep from the DBpedia query results.
- **Addition of NERs to the Ensemble:** Add other NERs to the ensemble: For example, NERs specialized in using other sources of information (ex: YAGO and Wikipedia) and ontologies.
- **Use tagged data to train NERs:** train systems based on BERT [7] for instance, both for ENTITY/NO-ENTITY tagging and to classify with tags adequate to the domain.
- **Improve the process of NERs output combination**.
- **Test the system in new domains**.
- **Integrate the newly obtained NERs in an Information Extraction pipeline**.

References

1 Allen NLP - An Apache 2.0 NLP research library, built on PyTorch, for developing state-of-the-art deep learning models on a wide variety of linguistic tasks. URL: `https://github.com/allenai/allennlp`.

2 Flora Amato, Giovanni Cozzolino, Vincenzo Moscato, and Francesco Moscato. Analyse digital forensic evidences through a semantic-based methodology and NLP techniques. *Future Generation Computer Systems*, 98:297–307, 2019.

3 Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. Dbpedia: A nucleus for a web of open data. In *The semantic web*. Springer, 2007.

4 Christian Bizer, Jens Lehmann, Georgi Kobilarov, Sören Auer, Christian Becker, Richard Cyganiak, and Sebastian Hellmann. Dbpedia-a crystallization point for the web of data. *Journal of web semantics*, 7(3):154–165, 2009.

5 Burton H Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.

6 Hiral Desai, Mohammed Firdos Alam Sheikh, and Satyendra K Sharma. Multi-purposed question answer generator with natural language processing. In *Emerging Trends in Expert Applications and Security*, pages 139–145. Springer, 2019.

7 Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint*, 2018. `arXiv:1810.04805`.

8 Mariana Dias, João Boné, João C Ferreira, Ricardo Ribeiro, and Rui Maia. Named entity recognition for sensitive data discovery in portuguese. *Applied Sciences*, 10(7):2303, 2020.

9 Tobias Ek, Camilla Kirkegaard, Håkan Jonsson, and Pierre Nugues. Named entity recognition for short text messages. *Procedia - Social and Behavioral Sciences*, 27:178–187, 2011.

10 João Ferreira, Hugo Gonçalo Oliveira, and Ricardo Rodrigues. Improving NLTK for processing portuguese. In *8th Symposium on Languages, Applications and Technologies (SLATE)*, 2019.

11 Pablo Gamallo and Marcos Garcia. Linguakit: uma ferramenta multilingue para a análise linguística e a extração de informação. *Linguamática*, 9(1):19–28, 2017.

12 Pablo Gamallo, Marcos Garcia, César Piñeiro, Rodrigo Martínez-Castaño, and Juan Pichel. Linguakit: A big data-based multilingual tool for linguistic analysis and information extraction. In *2018 Fifth International Conference on Social Networks Analysis, Management and Security (SNAMS)*, 2018. `doi:10.1109/SNAMS.2018.8554689`.

13 Matt Gardner, Joel Grus, Mark Neumann, Oyvind Tafjord, Pradeep Dasigi, Nelson Liu, Matthew Peters, Michael Schmitz, and Luke Zettlemoyer. Allennlp: A deep semantic natural language processing platform, 2018. `arXiv:1803.07640`.

14 Kyle Gorman. Pynini: A python library for weighted finite-state grammar compilation. In *Proceedings of the SIGFSM Workshop on Statistical NLP and Weighted Automata*, pages 75–80, 2016.

15 Kyle Gorman and Richard Sproat. How to get superior text processing in python with pynini, o'reilly ideas blog, 2016. accessed 22/04/2021. URL: `https://www.oreilly.com/content/how-to-get-superior-text-processing-in-python-with-pynini/`.

16 Alex Graves and Jürgen Schmidhuber. Framewise phoneme classification with bidirectional lstm and other neural network architectures. *Neural networks*, 18(5-6):602–610, 2005.

17 Ivan Herman, Sergio Fernández, Carlos Tejo Alonso, and Alexey Zakhlestin. Sparql endpoint interface to python. URL: `https://sparqlwrapper.readthedocs.io/en/latest/main.html`.

18 Zhiheng Huang, Wei Xu, and Kai Yu. Bidirectional LSTM-CRF models for sequence tagging. *arXiv preprint*, 2015. `arXiv:1508.01991`.

19 Daniel Jurafsky and James H. Martin. Information extraction. In *Speech and Language Processing*, chapter 17. (3rd ed. draft), 2020.

20 Daniel Jurafsky and James H. Martin. Sequence labeling for parts of speech and named entities. In *Speech and Language Processing*, chapter 8. (3rd ed. draft), 2020.

21 Fábio Lopes, César Teixeira, and Hugo Gonçalo Oliveira. Contributions to clinical named entity recognition in portuguese. In *Proc. 18th BioNLP Workshop and Shared Task*, 2019.

22 Pedro H. Luz de Araujo, Teófilo E. de Campos, Renato R. R. de Oliveira, Matheus Stauffer, Samuel Couto, and Paulo Bermejo. LeNER-Br: a dataset for named entity recognition in Brazilian legal text. In *PROPOR*, LNCS. Springer, 2018.

23 Lluís Padró. Analizadores Multilingües en Freeling. *Linguamática*, 3(2):13–20, 2011. URL: `https://linguamatica.com/index.php/linguamatica/article/view/115`.

24 A. Patel and A.U. Arasanipalai. *Applied Natural Language Processing in the Enterprise: Teaching Machines to Read, Write, and Understand*. O'Reilly Media, Incorporated, 2021.

25 André Pires, José Devezas, and Sérgio Nunes. Benchmarking named entity recognition tools for portuguese. *Proceedings of the Ninth INForum: Simpósio de Informática*, pages 111–121, 2017.

26 Juliana PC Pirovani, James Alves, Marcos Spalenza, Wesley Silva, Cristiano da Silveira Colombo, and Elias Oliveira. Adapting NER (CRF+ LG) for many textual genres. In *IberLEF@ SEPLN*, pages 421–433, 2019.

27 Peng Qi, Timothy Dozat, Yuhao Zhang, and Christopher D. Manning. Universal dependency parsing from scratch. In *Proc. of the CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 160–170, Brussels, Belgium, 2018.

28 Lizhen Qu, Gabriela Ferraro, Liyuan Zhou, Weiwei Hou, and Timothy Baldwin. Named entity recognition for novel types by transfer learning, 2016. `arXiv:1610.09914`.

29 Mário Rodrigues and António Teixeira. *Advanced applications of natural language processing for performing information extraction*. Springer, 2015.

30 Antonio Moreno Sandoval, Julia Díaz, Leonardo Campillos Llanos, and Teófilo Redondo. Biomedical term extraction: NLP techniques in computational medicine. *IJIMAI*, 5(4), 2019.

31 K. Sintoris and K. Vergidis. Extracting business process models using natural language processing (nlp) techniques. In *Proc. Conf, on Business Informatics (CBI)*, pages 135–139, 2017.

32 Fábio Souza, Rodrigo Nogueira, and Roberto Lotufo. Portuguese Named Entity Recognition using BERT-CRF, 2020. `arXiv:1909.10649`.

33 António Teixeira, Pedro Miguel, Mário Rodrigues, José Casimiro Pereira, and Marlene Amorim. From web to persons - providing useful information on hotels combining information extraction and natural language generation. In *Proc. IberSpeech*, Lisbon, 2016.

34 Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017. `arXiv:1706.03762`.

35 Wikivoyage. URL: `https://pt.wikivoyage.org/`.

Semantic Search of Mobile Applications Using Word Embeddings

João Coelho
Caixa Mágica Software, Lisbon, Portugal
Instituto Superior Técnico, Lisbon, Portugal

António Neto
Caixa Mágica Software, Lisbon, Portugal
University Institute of Lisbon, Portugal

Miguel Tavares
Caixa Mágica Software, Lisbon, Portugal
Lusophone University of Humanities and Technologies, Lisbon, Portugal

Carlos Coutinho
Caixa Mágica Software, Lisbon, Portugal
ISTAR-IUL, University Institute of Lisbon, Portugal

Ricardo Ribeiro
University Institute of Lisbon, Portugal
INESC-ID Lisbon, Portugal

Fernando Batista
University Institute of Lisbon, Portugal
INESC-ID Lisbon, Portugal

Abstract

This paper proposes a set of approaches for the semantic search of mobile applications, based on their name and on the unstructured textual information contained in their description. The proposed approaches make use of word-level, character-level, and contextual word-embeddings that have been trained or fine-tuned using a dataset of about 500 thousand mobile apps, collected in the scope of this work. The proposed approaches have been evaluated using a public dataset that includes information about 43 thousand applications, and 56 manually annotated non-exact queries. Our results show that both character-level embeddings trained on our data, and fine-tuned RoBERTa models surpass the performance of the other existing retrieval strategies reported in the literature.

2012 ACM Subject Classification Information systems → Retrieval models and ranking; Information systems → Document representation; Information systems → Language models; Information systems → Search engine indexing; Information systems → Similarity measures; Computing methodologies → Machine learning

Keywords and phrases Semantic Search, Word Embeddings, Elasticsearch, Mobile Applications

Digital Object Identifier 10.4230/OASIcs.SLATE.2021.12

Funding This work was supported by PT2020 project number 39703 (AppRecommender) and by national funds through FCT – Fundação para a Ciência e a Tecnologia with reference UIDB/50021/2020.

1 Introduction

The penetration of mobile devices in society has led most companies to see them as indispensable means for being in close contact with their customers. According to [8], Google Play Store has 2.6 million mobile apps available, Apple's iOS App Store has 1.85 million, and

© João Coelho, António Neto, Miguel Tavares, Carlos Coutinho, Ricardo Ribeiro, and Fernando Batista;
licensed under Creative Commons License CC-BY 4.0

Aptoide has about 1 million[1], which creates an extremely tough competition between apps. In terms of app downloads, in 2019 there were 204 billion app downloads, a number that has been increasing over the years. However, many of these downloads consist of multiple attempts to find the right app. Many downloaded apps are never used, and in 77% of cases, apps are not used again within 72 hours of installation. This shows a big misalignment between the supply of apps by the app stores (distribution services) and the demand for them by the consumers (discovery). Furthermore, around 2019, 52% of apps were discovered by word-of-mouth between acquaintances, friends or family, and only 40% were discovered by searching app stores. These inefficiencies make app discovery and distribution a considerable and extremely relevant challenge, since they take place in a market of massive penetration in societies and seriously affect the relationship between companies and consumers.

Based on this problem, the strategic objective of the AppRecommender project is to investigate and develop technologies capable of offering the right app, to the right customer, at the right time, by proposing a semantic search engine. The goal is to optimize current app distribution and discovery services and, inherently, to promote closer ties between companies and their target customers. The impact will be to increase user simplicity, efficiency and satisfaction in the discovery of apps, by optimizing the alignment between their needs, characteristics and context with the apps offered by the app store. As for developers or companies promoting mobile apps, the impact is on the level of increased proximity to target consumers, and optimizing their commercial success. The work here described was developed in the scope of the AppRecommender project.

This paper describes a dataset containing information about 500 thousand mobile apps, collected in the scope of this work, and proposes a set of approaches for semantic search, using the unstructured textual information contained in their name and description. The proposed approaches make use of word-level, character-level, and contextual word-embeddings that we have trained or fine-tuned with our available data. In order to evaluate the proposed approaches, we have used the public dataset described in [14], which includes information about 43,041 mobile applications and 56 non-exact queries previously annotated. Our results show that both character-level embeddings trained on our data, and fine-tuned RoBERTa models surpass the performance of the other existing retrieval strategies reported in the literature for this database.

This paper is organized as follows: Section 2 presents an overview of the related literature, focusing on existing work on semantic retrieval. Section 3 presents the data used in our experiments, which consists of the data that we have collected to train or fine-tune our models, and also the dataset that we have used for evaluation. Section 4 describes our set of approaches. Section 5 describes the conducted experiments and the achieved results. Finally, Section 6 summarizes the most relevant contributions, and presents the future work.

2 Related Work

As previously mentioned, there is a big misalignment between the supply of apps by the app stores and the demand for them by the consumers. In fact, this happens even considering that mobile app stores have search engines that allow users to find apps according to a provided query.

In general, although difficult to assess as they are proprietary, commercial app stores use keyword-matching approaches and are based in search engines, such as Lucene [4], or in open-source solutions, such as Solr [20] and Elasticsearch [1], built on top of Lucene.

[1] `https://pt.aptoide.com/company/about-us`

In order to improve search results, several strategies have been explored. For example, Mobiwalla [5, 6], a search engine based on Lucene, uses natural language processing techniques such as stemming and lemmatization to create multiple versions of the original query and synonyms or hyponyms for generalization. Another strategy is to use topic information. Topic modeling captures the main themes of a collection of documents, thus improving search on that collection [2]. Zhuo et al. [24] enrich queries and apps representations with topics (based on the titles and descriptions of the applications) and tags (a filtered combination of human labels and tags obtained by crawling the web and usage data regarding each application) to improve semantic matching. Park et al. [14] also explore topic information by jointly modeling apps descriptions and reviews and generating apps representations based on their descriptions, using this topic model. In a subsequent work, Park et al. [13] explore social media data to model the implicit intention of a user query. They create a parallel corpus containing aligned text spans that associate explicit intentions to the corresponding implicit intentions (*[I want pizza]$_{explicit}$ because [I'm hungry]$_{implicit}$*). They use this data to infer the intention associated with a query and then use a relevance model to find the apps that match this intention. Ribeiro et al. [18] also use topic models to improve the semantic matching between the query and the apps. However, the focus is on how topic models (applied to the description of the apps) can be used to infer keywords to expand the queries sent to standard search engines.

Closer to our work, and given the successful use of word embeddings in several speech and language processing problems [7], recent work on retrieval tasks also focused in exploring these representations. For example, Yao et al. [22] train user specific word embeddings (using her/his personal data) and use them to compute representations of user queries and documents. The matching between queries and documents is done using a neural matching model. This was experimented in search logs. Samarawickrama et al. [19] also explore embeddings for searching in Twitter. They also train user specific word embeddings and use them for query expansion. Yates et al. [23] survey methods for text ranking (i.e., score a collection of textual documents with respect to a query) leveraging neural language models. They distinguish between dual-encoders and cross-encoders. The former encode queries and documents independently, performing better temporally, while the latter encode concatenations of queries and documents, generally obtaining better results, but not suitable to search over large collections, given its computational cost.

3 Data

The dataset used in the scope of this work was built from scratch, by scrapping Aptoide's API, and is publicly available[2] for reproducible purposes. A first endpoint was used to extract general data about applications, including the title, Aptoide identifier, added date, update date, and a set of statistics. The Aptoide identifier was then used to query two other endpoints for application-specific information. The second endpoint contains information regarding the developer, required permissions and the description. The third endpoint contains information about the categories associated with the application.

The first endpoint returned information about 499 thousand applications. For those, the second and third endpoints were queried, returning information for 436,969 of them. The observed discrepancy in values occurred, not only due to missing information on the API, but also due to privatization and/or discontinuation of applications.

[2] `https://apprecommender.caixamagica.pt/wordpress/resources/`

The relevancy statistics in our dataset include the number of downloads, the average rating, and the total number of ratings of each of application. An initial analysis revealed that the vast majority of the applications were not of much value. This is supported by the high number of applications with very few downloads (Figure 1), and by the high number of applications that have never been rated (414,053). For the applications that were rated at least once, the average rating distribution is depicted in Figure 2. Nonetheless, we considered the average rating not to be a very descriptive measure, due to an average number of total ratings of approximately 4.

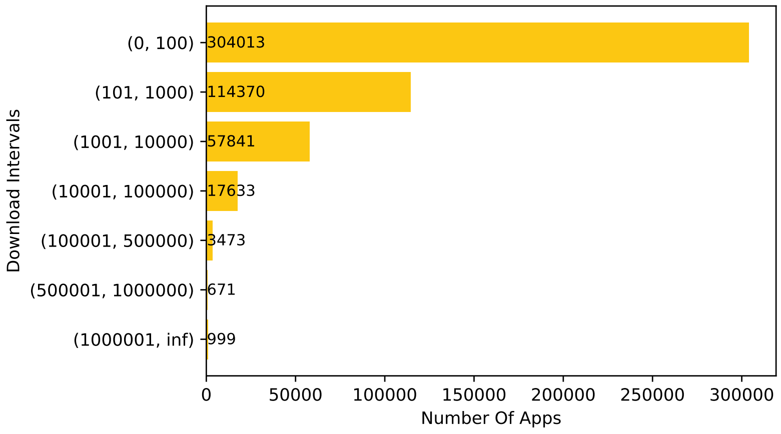


Figure 1 Number of applications within a interval of download values.

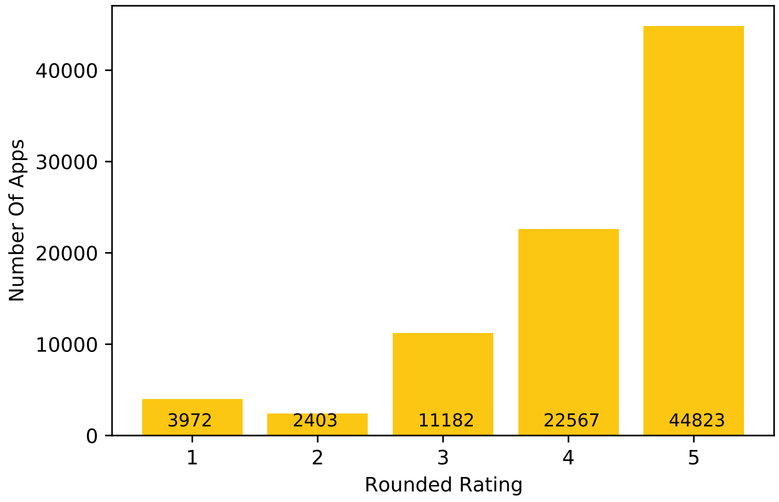


Figure 2 Rounded average rating distribution.

This way, we hypothesised that searching the whole dataset may not be ideal due to the irrelevant nature of the majority of the applications. As such, we derived a subset of relevant-only applications to search upon, based on four heuristics:

1. The top-5000 downloaded applications which were updated in the last 2 years;
2. The top-5000 downloaded applications which were updated in the last 6 months;
3. The top-1000 rated applications, with at least 200 rates and 1000 downloads;
4. The top-750 with more rates, added in the last 3 months, with at least 1000 downloads.

The objective was to consider applications that are widely used, not leaving out recent ones that are being updated constantly. The applications were also filtered to consider only those with descriptions in English. Since the information about the language is not included in our dataset, we used PyCLD3 (Python bindings for Google's CLD3 language detection model) to automatically detect it.

Overall, after removing duplicates, we ended up with 5,819 relevant applications. For each one of those applications, their name, description, downloads, average rating, total rating, added date and last update date were indexed in Elasticsearch.

The remaining non-relevant applications were stored in raw text files, since their textual data (name and description) constitute relevant resources for training our language models.

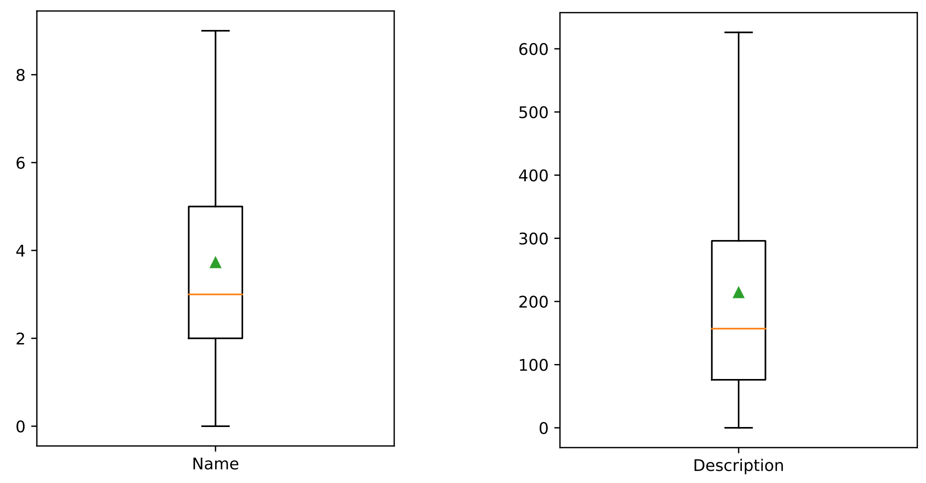


Figure 3 Distribution of the number of tokens for application's names (left) and application's descriptions (right). Green triangle represents the mean value, whereas the yellow line represents the median.

Figure 3 shows some basic statistics about the number of words that constitute both the application name and description, after removing some of the most salient outliers for visualization purposes. The figure reveals that the name of an application usually contains more than 2 tokens, and that it's description ranges from less than 100 tokens to more than 300 tokens. The following text is an example of an application description containing 111 words and multiple sentences.

Don't Starve: Pocket Edition, brings the hit PC game enjoyed by over 6 million players to Android.
Now you can experience the uncompromising wilderness survival game full of science and magic on the go! Play as Wilson, an intrepid Gentleman Scientist who has been trapped and transported to a mysterious wilderness world. Wilson must learn to exploit his environment and its inhabitants if he ever hopes to escape and find his way back home.
Enter a strange and unexplored world full of strange creatures, dangers, and surprises. Gather resources to craft items and structures that match your survival style. Play your way as you unravel the mysteries of this strange land.

4 Approach

The main objective of our proposal was to compare semantic with lexical search in the context of mobile applications. Sections 4.2 to 4.3 introduce the models used to do so. For this, we consider both word-level and character-level word embeddings, to access which works better in this domain. Then, we compare both to contextual embeddings, generated through Transformer-based neural language models. Section 4.4 describes the indexing and searching mechanisms.

4.1 GloVe Word Embeddings

GloVe [15] is a model for unsupervised generation of static word embeddings. Alike Word2Vec [12], it uses local context window methods, but combines it with word-word correlation matrix factorization. We used a pre-trained model on textual data from Wikipedia and Gigatext, with 100-dimensional vectors.

To generate the embeddings for application's names, the strings are lower-cased and out-of-vocabulary words are removed. For descriptions, besides lower-casing and out-of-vocabulary word removal, stop-words words are removed. The models are used to generate word-level embeddings, and the average of the vectors is used as a final sentence embedding.

4.2 FastText Word Embeddings

FastText [3] follows an approach similar to Word2Vec [12], but each word is represented as a bag of character n-grams. This way, character-level embeddings are considered, instead of word-level representations.

In preliminary tests, we compared a CBOW [12] fastText model (FT1), pre-trained on English Common Crawl considering word 5-grams, to the aforementioned pretrained GloVe model. As the results favored the character-level embeddings, we trained a CBOW fastText model (FT2) from scratch using out textual data (see Section 3).

Since these models are quite big, we reduced the dimension of the vectors from 300 to 100, using the dimension reduction tool provided by fastText python library.

For both FT1 and FT2, the process to generate embeddings is the same, which is the default behaviour of fastText's embedding generation tool, with minor changes. For application's names, the strings are lower-cased. For descriptions, besides lower-casing, stop-words are also removed. The nature of these embeddings (character-level) allow for out-of-vocabulary words to be included. The models are used to generate word-level embeddings, which are then normalized by their L2 norm. The average of the vectors with positive L2 norm is taken as a final sentence embedding.

4.3 RoBERTa Contextual Embeddings

The main limitation of the previous models is that a word always has the same vector, despite the context of its usage. As such, we consider the usage of Contextual Embeddings, which are context-dependent representations that capture the use of words across multiple scenarios [10]. We explore the usage of a fine-tuned neural language model as a dual-encoder in the context of a mobile application search engine, since good results have been reported in other retrieval contexts [9, 16, 17], and dual-encoders make it possible to pre-compute and index representations for the application's textual data.

We start by fine-tuning RoBERTa$_{\text{base}}$ [11] on a masked language modelling task, using the Huggingface Transformers library [21]. We split our textual data (see Section 3) into train and test sets (90% and 10%, respectively). The text in the train set is processed into sets of 512 tokens by RoBERTa's original tokenizer, with a masking probability of 15%. The model was trained during 1 epoch with a batch size of 4, using a cross-entropy loss function. The test set was used to evaluate the fine-tuned model, which achieved a perplexity of 4.54 on a mask prediction task.

We further trained the model on a semantic similarity task, using the Sentence Transformers library [17]. For this, we synthetically build queries by concatenating an application's name with its categories. Then, each query is associated with the description of the application. This way, we build a collection of (query, relevant description) pairs. Within a training batch of size 8, triplets are built from the pairs, in the form (query, relevant description, fake description). The relevant description of a given query is used as the fake descriptions for the others, which allows for $8 \times 7 = 56$ training examples per batch. The model is used to generate representations for the query and the descriptions, and scores are obtained from the cosine similarity between the representations of the query and the descriptions. The objective is to minimize the negative log-likelihood of softmaxed scores. The whole textual dataset is used to train, except for 500 random applications which were used to evaluate. For those applications, queries were built as previously described, and a description corpus was built from their descriptions. The model was used as a dual-encoder to rank the 500 descriptions for each query. Given that each query only has one relevant description, we computed the Precision@1, the Recall@10, and the MRR@10. The results were 0.8795, 0.9732, and 0.9167, respectively.

Given a sentence as input, the final model (henceforth RoBERTapp) is used to generate representations through mean pooling of the word token embeddings of the last hidden-layer.

4.4 Indexing and Searching

The previous models were used to generate representations for the name and description of applications to be indexed (see Section 3). The representations were pre-computed, and stored along with the name and description of each application in an ElasticSearch index. The name and description are used to perform the classic lexical search. We built an interface to allow us to query this index, where we can choose which model to use, and which fields to consider (i.e., name, description, or both).

Searching with the lexical model uses the standard ElasticSearch analyser to process the query, which was also used to process the indexed textual data. Given a query q, the Lucene Scoring Function is used to compute the scores over the chosen fields, combining them if more than one:

$$\text{score}(q,a) = \sum_{f \in a} \text{LSF}(q,f) \ , \tag{1}$$

$$\text{LSF}(q,f) = \frac{1}{\sqrt{\sum_{t \in q} \text{idf}(t)^2}} \times \frac{|q \cap f|}{|q|} \times \sum_{t \in q} \left(\frac{\text{tf}(t,f) \cdot \text{idf}(t)^2 \cdot w_f}{\sqrt{|f|}} \right) \ , \tag{2}$$

where f are application a's textual fields (in our case, name and/or description), t are the query's tokens, tf is the term-frequency, and idf is the inverse document frequency.

Searching with the vector-based models leverages ElasticSearch's built-in cosine similarity function. Representations for queries are generated at run-time, using the chosen model. Given a query q and an application a, the score is computed as follows:

$$\text{score}(q, a) = \alpha\, \text{sim}(\text{M}(q), \text{M}(a_n)) + \beta\, \text{sim}(\text{M}(q), \text{M}(a_d)) \;, \tag{3}$$

where M is the model encoding function, a_n is an application's name, a_d is an application's description, sim is the cosine similarity function, and α, β are combination weights. Note that $\text{M}(a_n)$ and $\text{M}(a_d)$ are already indexed, only $\text{M}(q)$ is generated at run-time.

Ultimately, given a query and a model, the scores are computed and the top-N scoring applications are retrieved.

5 Experiments and Results

Since our dataset does not include any queries and relevance judgments, we evaluate our models on the data provided by Park et al. [14], before manually analysing the results of the models on our indexed data.

This dataset contains information about 43,041 mobile applications including name and description. The dataset also features 56 non-exact queries (i.e., queries with a meaningful semantic context, instead of an application's name). For each one of the queries, 81 applications are labelled with a relevancy score of 0 (not relevant), 1 (somewhat relevant), or 2 (very relevant). These scores were manually annotated.

The authors used the Normalized Discounted Cumulative Gain as the evaluation metric, which takes the full order of the item list and graded relevances into account:

$$\text{DCG@}k = \text{R}(1) + \sum_{i=2}^{k} \frac{\text{R}(i)}{\log_2(i)} \;, \tag{4}$$

$$\text{NDCG@}k = \frac{\text{DCG@}k}{\text{IDCG@}k} \;, \tag{5}$$

where $\text{R}(i)$ is a function that returns the relevance value of the passage at rank i. The index of the passage up to which the ranking is considered is represented by k. The DCG is normalized with the ideal DCG (IDCG), i.e., the DCG of a perfectly sorted result.

Table 1 shows the results for the evaluation of our models, which was conducted under the same conditions as reported by Park et al. [14], i.e., ranking the 81 labeled applications for each one of the 56 queries.

For comparison, Table 2 shows results achieved by Google Play and LBDM reported by Park et al. [14], and results achieved by lexical models leveraging topic modelling techniques, reported by Ribeiro et al. [18], for the same scenario.

The results show that the pre-trained models (GloVe and FT1) performed worse than previous approaches. On the other hand, FT2 and the RoBERTapp models surpass previous approaches. We can also conclude that searching considering the name and description is, overall, the most advantageous combination. In this scenario, RoBERTapp achieved the best results for NDCG@{3,25}, and the best scores for NDCG@{5,10} were achieved by FT2.

RoBERTapp is the best model when dealing with descriptions only, perhaps due to the second fine-tuning task, which consisted in scoring descriptions based on a synthetic query. Conversely, FT2 is superior to RoBERTapp when searching with name only.

Since models FT2 and RoBERTapp performed well, we manually evaluated them on our relevant application index, considering names and descriptions. We query the index as described in Section 4.4. As expected, when the query corresponds or is close to an

Table 1 NDCG@{3,5,10,25} for the multiple models, considering application's name and description (N+D), only name (N), and only description (D).

	NDCG@3	**NDCG@5**	**NDCG@10**	**NDCG@25**
GloVe (N + D)	0.527	0.523	0.522	0.538
GloVe (N)	0.523	0.514	0.512	0.529
GloVe (D)	0.504	0.491	0.489	0.508
FT1 (N + D)	0.540	0.521	0.532	0.543
FT1 (N)	0.512	0.507	0.513	0.529
FT1 (D)	0.462	0.466	0.461	0.476
FT2 (N + D)	0.587	**0.589**	**0.582**	0.600
FT2 (N)	0.595	0.582	0.571	0.582
FT2 (D)	0.519	0.529	0.519	0.545
RoBERTapp (N + D)	**0.616**	0.587	0.581	**0.605**
RoBERTapp (N)	0.582	0.570	0.568	0.590
RoBERTapp (D)	0.585	0.581	0.577	0.585

Table 2 NDCG@{3,5,10,25} achieved by Google Play and LBDM [14], and achieved by lexical models considering applications' description and topics [18].

	NDCG@3	**NDCG@5**	**NDCG@10**	**NDCG@25**
LBDM	0.584	0.563	0.543	0.565
Google Play	0.589	0.575	0.568	0.566
BM25F	0.574	0.542	0.527	0.544
ElasticSearch	0.552	0.532	0.504	0.519

application name, none of the models had problems retrieving the correct applications. Since our objective is to enrich a search engine with semantic capabilities, we further tested with non-exact queries. Table 3 shows the results for three example queries. Note that this search was conducted in the setup described in Section 4.4, considering only the embeddings generated by the models. This is, no other information regarding relevance (e.g., downloads, ratings, etc...) was considered, so as to better access the usefulness of semantic models.

The semantic models were able to retrieve relevant applications which fit the scope of the non-exact queries. For example, searching for "social network" with RoBERTapp returned the most widely-used social networks. One can argue that FT2 worked better than RoBERTapp for the query "airline tickets", since the latter returned a game as the top-result. Still, overall, both models provide more appropriate results than the lexical model.

6 Conclusions and Future Work

This paper proposes a set of approaches for semantic search of mobile applications, which use clues contained in the name and textual descriptions of these applications for selecting the most relevant ones for a given query. Our approaches are based on word-level (GloVe), character-level (fastText), and on contextual (RoBERTa) word-embeddings. We have described the process of collecting a dataset, containing information about mobile apps, that was further described and used for training and fine-tuning our models. The proposed approaches have been evaluated using a publicly available dataset of mobile applications, and the results achieved show that both character-level embeddings, trained on our data, and fine-tuned RoBERTa models, fine-tuned also using our data, when applied in an unsupervised way,

Table 3 Comparison between the lexical model and the semantic models (FT2, RoBERTapp) for non-exact queries, considering name and description. The ordered top-5 results are shown.

Query: "social network"

Lexical	**RoBERTapp**	**FT2**
Hornet - Social Network	Facebook	Peeks Social
BandLab – Music Studio & Social Network	Instagram	Network Browser
Pi Network	Twitter	Hornet - Social Network
Network Browser	Internet	Cartoon Network App
Peeks Social	Facebook Viewer	Air-Share

Query: "food at home"

Lexical	**RoBERTapp**	**FT2**
Domino's Pizza - Online Food Delivery App	EatSure - Food Delivery | Order Food Now!	foodpanda - Local Food & Grocery Delivery
Mixer – Interactive Streaming	foodpanda: Fastest food delivery, amazing offers	EatSure - Food Delivery | Order Food Now!
Trendyol - Online Shopping	foodpanda - Local Food & Grocery Delivery	Swiggy Food Order & Delivery
DoorDash - Food Delivery	DoorDash - Food Delivery	Zomato - Online Food Delivery & Restaurant Reviews
foodpanda - Local Food & Grocery Delivery	Toca Kitchen Sushi Restaurant	Glovo: Order Anything. Food Delivery and Much More

Query: "airline tickets"

Lexical	**RoBERTapp**	**FT2**
Privat24	Airline Commander - A real flight experience	MakeMyTrip-Flights Hotels Cabs
Trip.com: Flights, Hotels, Train	Southwest Airlines	Cleartrip - Flights, Hotels, Train Booking
OpenSooq	American Airlines	ebookers - Hotel, Flight, Car Hires
Flüge.de	Flightradar24 Flight Tracker	Goibibo - Hotel Car Flight
KAYAK flights, hotels & car hire	MakeMyTrip-Flights Hotels Cabs	Trip.com: Flights, Hotels, Train

surpass the performance of the other existing retrieval strategies reported in the literature. We further confirmed that the proposed semantic-related models capture the scope of non-exact queries, which lexical models struggle to do, by manually searching over our relevant applications index.

In the near future, we are planning to improve our RoBERTa model by considering additional unsupervised/semi-supervised training tasks. The reported experiments use only the similarity between the query and each one of the candidate applications. We are also planning to create a multi-criteria retrieval system that takes into account other relevant information, such as the number of downloads, rating and the fact that a given application was updated recently.

References

1 S. Banon. Elasticsearch, 2010. URL: `https://www.elastic.co/`.

2 David M. Blei. Probabilistic Topic Models. *Commun. ACM*, 55(4):77–84, 2012. `doi:10.1145/2133806.2133826`.

3 Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomás Mikolov. Enriching Word Vectors with Subword Information. *Trans. Assoc. Comput. Linguistics*, 5:135–146, 2017. URL: `https://transacl.org/ojs/index.php/tacl/article/view/999`.

4 D. Cutting. Apache Lucene, 1999. URL: `https://lucene.apache.org/`.

5 Anindya Datta, Kaushik Dutta, Sangar Kajanan, and Nargin Pervin. Mobilewalla: A Mobile Application Search Engine. In Joy Ying Zhang, Jarek Wilkiewicz, and Ani Nahapetian, editors, *Mobile Computing, Applications, and Services*, pages 172–187, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

6 Anindya Datta, Sangaralingam Kajanan, and Nargis Pervin. A Mobile App Search Engine. *Mobile Networks and Applications*, 18, 2013.

7 Sahar Ghannay, Benoit Favre, Yannick Estève, and Nathalie Camelin. Word Embedding Evaluation and Combination. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)*, pages 300–305, Portorož, Slovenia, 2016. European Language Resources Association (ELRA). URL: `https://www.aclweb.org/anthology/L16-1046`.

8 Mansoor Iqbal. App download and usage statistics (2020). web page, October 2020. URL: `https://www.businessofapps.com/data/app-statistics/`.

9 Omar Khattab and Matei Zaharia. ColBERT: Efficient and effective passage search via contextualized late interaction over BERT. In Jimmy Huang, Yi Chang, Xueqi Cheng, Jaap Kamps, Vanessa Murdock, Ji-Rong Wen, and Yiqun Liu, editors, *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval, SIGIR 2020, Virtual Event, China, July 25-30, 2020*, pages 39–48. ACM, 2020. `doi:10.1145/3397271.3401075`.

10 Qi Liu, Matt J. Kusner, and P. Blunsom. A Survey on Contextual Embeddings. *ArXiv*, abs/2003.07278, 2020. `arXiv:2003.07278`.

11 Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. RoBERTa: A robustly optimized BERT pretraining approach. *CoRR*, abs/1907.11692, 2019. `arXiv:1907.11692`.

12 Tomas Mikolov, G.s Corrado, Kai Chen, and Jeffrey Dean. Efficient Estimation of Word Representations in Vector Space. In *Proceedings of the International Conference on Learning Representations*, 2013.

13 Dae Hoon Park, Yi Fang, Mengwen Liu, and ChengXiang Zhai. Mobile App Retrieval for Social Media Users via Inference of Implicit Intent in Social Media Text. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, CIKM '16, page 959–968, New York, NY, USA, 2016. Association for Computing Machinery. `doi:10.1145/2983323.2983843`.

14 Dae Hoon Park, Mengwen Liu, ChengXiang Zhai, and Haohong Wang. Leveraging User Reviews to Improve Accuracy for Mobile App Retrieval. In Ricardo Baeza-Yates, Mounia Lalmas, Alistair Moffat, and Berthier A. Ribeiro-Neto, editors, *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval, Santiago, Chile, August 9-13, 2015*, pages 533–542. ACM, 2015. `doi:10.1145/2766462.2767759`.

15 Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global Vectors for Word Representation. In Alessandro Moschitti, Bo Pang, and Walter Daelemans, editors, *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 1532–1543. ACL, 2014. `doi:10.3115/v1/d14-1162`.

16 Yingqi Qu, Yuchen Ding, Jing Liu, Kai Liu, Ruiyang Ren, Xin Zhao, Daxiang Dong, Hua Wu, and Haifeng Wang. RocketQA: An Optimized Training Approach to Dense Passage Retrieval for Open-Domain Question Answering. *CoRR*, abs/2010.08191, 2020. `arXiv:2010.08191`.

17 Nils Reimers and Iryna Gurevych. Sentence-BERT: Sentence embeddings using siamese BERT-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, November 2019. arXiv:1908.10084.

18 Eugénio Ribeiro, Ricardo Ribeiro, Fernando Batista, and João Oliveira. Using Topic Information to Improve Non-exact Keyword-Based Search for Mobile Applications. In Marie-Jeanne Lesot, Susana Vieira, Marek Z. Reformat, João Paulo Carvalho, Anna Wilbik, Bernadette Bouchon-Meunier, and Ronald R. Yager, editors, *Information Processing and Management of Uncertainty in Knowledge-Based Systems*, pages 373–386, Cham, 2020. Springer International Publishing.

19 Sameendra Samarawickrama, Shanika Karunasekera, Aaron Harwood, and Ramamohanarao Kotagiri. Search Result Personalization in Twitter Using Neural Word Embeddings. In Ladjel Bellatreche and Sharma Chakravarthy, editors, *Big Data Analytics and Knowledge Discovery*, pages 244–258, Cham, 2017. Springer International Publishing.

20 Y. Seeley. Apache Solr, 2004. URL: https://lucene.apache.org/solr/.

21 Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. Transformers: State-of-the-Art Natural Language Processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online, 2020. Association for Computational Linguistics. URL: https://www.aclweb.org/anthology/2020.emnlp-demos.6.

22 Jing Yao, Zhicheng Dou, and Ji-Rong Wen. Employing Personal Word Embeddings for Personalized Search. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '20, page 1359–1368, New York, NY, USA, 2020. Association for Computing Machinery. doi:10.1145/3397271.3401153.

23 Andrew Yates, Rodrigo Nogueira, and Jimmy Lin. Pretrained transformers for text ranking: BERT and beyond. In Liane Lewin-Eytan, David Carmel, Elad Yom-Tov, Eugene Agichtein, and Evgeniy Gabrilovich, editors, *WSDM '21, The Fourteenth ACM International Conference on Web Search and Data Mining, Virtual Event, Israel, March 8-12, 2021*, pages 1154–1156. ACM, 2021. doi:10.1145/3437963.3441667.

24 Juchao Zhuo, Zeqian Huang, Yunfeng Liu, Zhanhui Kang, Xun Cao, Mingzhi Li, and Long Jin. Semantic Matching in APP Search. In *Proceedings of the Eighth ACM International Conference on Web Search and Data Mining*, WSDM '15, page 209–210, New York, NY, USA, 2015. Association for Computing Machinery. doi:10.1145/2684822.2697046.

Command Similarity Measurement Using NLP

Zafar Hussain
Department of Computer Science, University of Helsinki, Finland

Jukka K. Nurminen
Department of Computer Science, University of Helsinki, Finland

Tommi Mikkonen
Department of Computer Science, University of Helsinki, Finland

Marcin Kowiel
F-Secure Corporation, Poland

Abstract

Process invocations happen with almost every activity on a computer. To distinguish user input and potentially malicious activities, we need to better understand program invocations caused by commands. To achieve this, one must understand commands' objectives, possible parameters, and valid syntax. In this work, we collected commands' data by scrapping commands' manual pages, including command description, syntax, and parameters. Then, we measured command similarity using two of these – description and parameters – based on commands' natural language documentation. We used Term Frequency-Inverse Document Frequency (TFIDF) of a word to compare the commands, followed by measuring cosine similarity to find a similarity of commands' description. For parameters, after measuring TFIDF and cosine similarity, the Hungarian method is applied to solve the assignment of different parameters' combinations. Finally, commands are clustered based on their similarity scores. The results show that these methods have efficiently clustered the commands in smaller groups (commands with aliases or close counterparts), and in a bigger group (commands belonging to a larger set of related commands, e.g., *bitsadmin* for Windows and *systemd* for Linux). To validate the clustering results, we applied topic modeling on the commands' data, which confirms that 84% of the Windows commands and 98% ofthe Linux commands are clustered correctly.

2012 ACM Subject Classification Computing methodologies → Natural language processing

Keywords and phrases Natural Language Processing, NLP, Windows Commands, Linux Commands, Textual Similarity, Command Term Frequency, Inverse Document Frequency, TFIDF, Cosine Similarity, Linear Sum Assignment, Command Clustering

Digital Object Identifier 10.4230/OASIcs.SLATE.2021.13

1 Introduction

While users dominantly operate their computers with graphical user interfaces, the operating system converts the mouse clicks to process invocations and other low-level commands. These commands can be stored in textual form to logfiles and resemble the command line (CLI) commands, which users type to command prompts in terminal windows. The stored command history is useful for diagnostics, learning the prevailing commands, user profiling, and various other purposes.

In this work, we aim to help cybersecurity specialists to detect similar commands via textual analysis of commands' man-pages. In particular, we want to understand when two commands are doing the same thing. This is a basic building block of a more complex system to detect anomalous commands. It is important as attackers may try to hide their operation by using unusual commands. To detect any unusual command, it should be compared with other prevalent commands. A user might use alternate commands (aliases) to perform same task. These alternate commands can be valid but because of the different command names,

© Zafar Hussain, Jukka K. Nurminen, Tommi Mikkonen, and Marcin Kowiel;
licensed under Creative Commons License CC-BY 4.0

there are possibilities that they will not be associated with any prevalent command or a group of usual commands. To handle this situation we want to create a system, which will be used as reference, to learn about the unusual command. When an unusual command is encountered, the reference system will be checked, and if there are alternate commands for this unusual command, this command should be marked as safe. One example of this scenario is `del` command. If `del` is encountered as the unusual command, by referring the system we can learn that `del` is similar to `erase` command (which has already been used), so `del` should be marked as safe. In case the new command does not have any similar command, and cannot be found in the command history (logfiles), then the cybersecurity specialists will require further actions. Therefore, we develop command similarity measures, which indicate how strongly two commands resemble each other.

An obvious way to measure command similarity is to compare two command strings with each other. However, it is a poor approach if two commands, such as `del` and `erase`, look very different but do the same thing. Therefore, instead of comparing explicit command strings, we compare command descriptions, available in the documentation manuals. For this, we use natural language processing (NLP) technology. In this way, we develop measures that better detect the semantic similarity of commands than operating only at the command syntax level. Besides studying the command description we also study the command parameter descriptions because they have a major impact on what the command will ultimately do.

The essence of our work is that instead of comparing the raw commands, we compare their documentations. We start from Windows and Linux manual pages, and extract command syntax, description, and parameter information. We then use NLP techniques to analyze the texts. Notice that instead of users explicitly typing the commands, many are generated by a Graphical User Interface (GUI), scheduled tasks, or system activities, which result in new process invocation with its command-line arguments. The key contributions of this paper are (i) showing how NLP techniques are useful in evaluating command similarity; (ii) presenting results of Windows and Linux command similarities with statistics and examples; and (iii) clustering commands based on their similarity.

2 Background and Related Work

A program can be started by another program via operating system API directly or indirectly. A user can invoke a program by interacting with GUI or from a command-line prompt. Programs are also invoked by scheduled tasks and other system activities. Commands have a general form of `param0(command) param1 param2 param3 ... paramN`, where `param0` is typically a name or path of the program to be invoked and `param1` to `paramN` are command parameters (also called flags or switches). Moreover, parameters can be comprised of two or more parts that are logically separated. As the syntax of parameters is not standardized, developers may end up using inconsistent conventions. Concerning command similarity detection, several approaches have been proposed, including (i) measures based on command history, which tries to predict what the user is likely to type next [12, 5]; (ii) converting program fragments to natural language and using NLP for a different kind of processing of these, e.g. for similarity detection of code fragments [19]; and (iii) string similarity comparison either at a character or at token level [6]. There are some tools and methodologies related to the Command-Line parsing. These include NL2Bash[15], a parsing method to map English sentences to Bash commands; ExplainShell[1] to see the help text that matches each argument

[1] `https://explainshell.com/`

of a given command; a documentation browser[2], which finds the relevant docs by parsing the given code and connecting parts of it to their documentation; and a fuzzy hashing method [11], which computes context triggered piecewise hashes(CTPH) to match inputs with sequences of identical bytes in the same order. These programs and methodologies can be used to understand the commands, their explanations, and structure.

However, little research is available in the exact domain of our work. There are papers on the comparison of CLI commands and GUI [21], aiming to find out the strengths/weaknesses of these systems, on the use and importance of CLIs, Adaptive CLIs [4, 8], and Statistical approaches to create profiles from CLIs [9]. The closest work to ours is in the field of linguistics, where the focus is on syntax and semantics of commands. For instance, [14, Chapter 5] provides a linguistic sketch of the Unix language family, which introduces how the Unix commands are written, conjoined, and embedded. However, it does not study the similarity between commands. Another closely related field is code similarity, where interest is in differentiating between representational and behavioral similarities [10]. For instance, text-based approaches seek similarities at characters level and token-based approaches perform token-based filtering and detect renaming of variables, classes, and so on. By defining behavioral similarity with respect to input/output, code fragments can be run to find the similarities, and deep learning can be used to find similarities among code [19]. As discussed earlier, our research goal is to reach a better understanding of the commands' similarities. One way to find similarities among commands is to compare the text by extracting the vector representations and finding the similarities of these vectors to detect similar commands. However, our approach of moving from command to corresponding natural language description and performing comparison at that level has not been studied before.

3 Applying NLP mechanisms

Data Collection

We collected Windows data by web scrapping the official documentation[3] of Windows commands. For web scraping, a Python library Beautiful Soup [17] is used. We selected four parameters for each command, which are the command name, command description, command syntax, and command parameters. We selected a total of 685 commands.

For Linux, no official command documentation was found, but open source web projects for Linux manual pages are available. One of these projects is "The Linux man-pages project"[4], which contains documentation of commands from man pages 1 to man pages 8. We only used man pages 1 and man pages 8, which are for "User commands" and "Superuser and system administration commands" respectively. Four parameters were selected for Linux commands as well, including the command name, its description, synopsis, and parameters. With Beautiful Soup, we collected 2247 Linux commands.

[2] `http://showthedocs.com/`
[3] `https://docs.microsoft.com/en-us/Windows-server/administration/Windows-commands/Windows-commands`
[4] `https://www.kernel.org/doc/man-pages/`

Data Wrangling

Since some of the Windows commands had only the parameter "/?", which could be problematic when comparing with each other, this parameter is excluded from the dataset. The rationale for excluding this parameter is that if two commands with different objectives have only the "?" parameter while comparing them will result in high similarity, which would affect the results. Similarly, "-help", "?", and "version" parameters were also excluded from Linux commands, as there were some commands with only these parameters. All other parameters are kept as they were for further utilization.

Similarity Measures

Several methods are considered to calculate the similarities between commands, such as Word2vec [16] and GloVe [18]. Since Word2vec and GloVe are trained on general English (news, Wikipedia) corpus we did not find embedding useful for the CLI manuals comparison. Since commands data does not have too much variation in the topics, applying any pre-trained model would not be a plausible solution. Generally, pre-trained models work well for document similarities when the topics are of different domains, for example, science and religion, politics and technology, economics and sports, etc. This data is just about commands' description, which with minor variations, contain similar vocabulary and are of the same domain.

TFIDF

A simple yet powerful method to calculate document similarity is to use a document vector based on the Term-Frequency Inverse-Document-Frequency (TFIDF) on the bag of words, with cosine similarity. Generally, in text mining, documents are represented as vectors where the elements in the vectors reflect the frequency of terms in documents [20]. Each word in a document has two types of weights. Local weights are normally expressed as Term Frequencies (TF), and global weights are stated as Inverse Document (IDF) Frequency [20]. TF represents the number of times a word occurred in a document, whereas, IDF gives the weight of a term. Multiplying TF and IDF result in the TFIDF score of a word in a document. The higher the score, the more relevant that word is in that particular document. TFIDF often produces higher scores for words related to the topical signal of a text and lower scores for the words with high frequency, it is well suited for tasks that involve textual similarity.

Cosine Similarity

Cosine Similarity produces a metric, which indicates how related are two documents by looking at the angle between two documents instead of magnitude [7]. Using cosine measure on the top of TFIDF vectors gives a similarity score of two text documents.

Linear Sum Assignment

To compare one command's parameters with another command's, an $n_1 \times n_2$ matrix is created, where n_1 are the total parameters of command 1 and n_2 are the total parameters of command 2. This matrix shows the similarity score of parameters of one command with the parameters of another. Since the objective of this analysis is to find out the similarity score of one command with another based on their parameters, it is required to define a single similarity score based on $n_1 \times n_2$ parameter similarity matrix. To create such matrix, the Hungarian method [13] is used. The method solves the Linear Sum Assignment

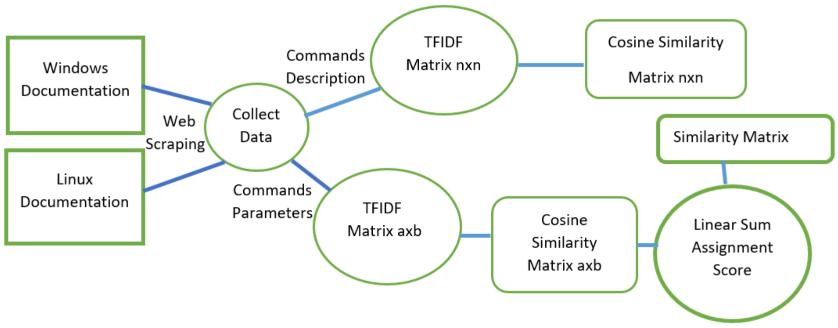


Figure 1 A Step-wise Overview of the Mechanism.

Problem (LSAP), one of the most famous problems in linear programming and combinatorial optimization [3] Given an $n \times n$ cost matrix C, each $C_{i,j}$ is the cost of matching vertex i of the first partite set and vertex j of the second set [3]. The objective is to match each row to a different column so that the sum of the corresponding entries is minimized. In other words, we select n elements of C so that there is exactly one element in each row and one in each column and the sum of the corresponding costs is minimum [3]. The formulation of this algorithm is $min \Sigma_i \Sigma_j C_{i,j} X_{i,j}$, where X is a boolean matrix with $X_{i,j}$=1 iff row i is assigned to column j. Tuning this formula, as shown below, by multiplying the similarity matrix with -1 and normalizing it with the minimum number of parameters among two commands, returns a value that indicates how similar two commands are based on their parameters.

$$\frac{-min \Sigma_i \Sigma_j C_{i,j} X_{i,j}}{min(n_1, n_2)} \tag{1}$$

Command similarities based on the subset of data

As the data contains descriptions and parameters, we use both as a basis for comparison. The rationale for two different comparisons is to analyze which commands have similar objectives. We believe that comparing descriptions of programs results a group of programs performing similar actions, like data compression, file manipulation, or network tools. Another point of interest is comparing commands parameters, to consider which of the commands have common flags.

Similarities based on command description. First, commands are compared based on their description. A TFIDF matrix is generated after removing the stopwords, or frequently used words that do not carry any thematic meaning, like the, is, at, or an. After creating the TFIDF matrix, cosine similarity is calculated for all the commands. The generated $n \times n$ matrix, with diagonal values as 1, gives cosine similarities of each commands' description with other commands.

Similarities based on command parameters. For the comparison of the parameters, first, the TFIDF matrix of commands is generated after removing the stopwords. This matrix is of $n_1 \times n_2$ shape, where n_1 are the total number of parameters of command 1 and n_2 are the parameters of command 2. To calculate a single similarity score, the LSAP method is applied on top of this TFIDF matrix. This resulted in an $n \times n$ matrix where n is the total number of commands.

Similarities based on command description and parameters combined. Finally, for each command, parameters' text is appended at the end of description text but the parameters' names are excluded from this textual analysis. Following the same procedure, the TFIDF matrix is created for all n commands. Using the matrix, the cosine similarity of the commands' overall text is calculated. This resulted in an $n \times n$ matrix, which shows the commands' overall similarity.

4 Results

After creating the $n \times n$ matrix of command similarities, Windows and Linux commands are visualized in the separate graphs. As discussed above, these similarities are based on description, parameters, and the overall (description + parameters) commands textual data. The starting point of our analysis is to find out the ratios of commands with high similarities and low similarities. Commands with high similarities are those that have at least one other command with a similarity score of 0.75 or more. The rationale for selecting 0.75 as a threshold is that it will give us commands which have 3/4th of similar text, which in general is a good enough threshold of comparing two texts. For example in Windows commands, based on the commands' description, `md` has highest similarity score (equal to 1) with `mkdir`, whereas `sxstrace` have highest similarity with `pathping` with a similarity score 0.088. So, `md` falls under the high similarity tag (sim >=0.75) and `sxstrace` falls under the low similarity tag (sim <0.75).

One reason for the commands falling under the low similarity tag is the very short description. It is also worth mentioning that changing the threshold values, will affect the results. The result shows that commands that fall under the low similarity tag can be assumed as the isolated commands, as they are not related to other commands and generally they have no alternative commands either. In contrast, commands that fall under the high similarity tag are the ones either from the same domain such as windows' `bitsadmin*` or have the alternative commands such as `del` and `erase`.

When compared based on their descriptions, most commands fall in the low ranges (sim less than 0.75). These commands do not show high similarity with other commands. These commands need more attention when working with them, as any analysis for the commands with high similarity such as `md` and `mkdir` can be easily applied and interpreted, as they have a similarity score of 1 based on their description. For example, a program supposed to learn the behavior of `md`, will easily learn the behavior of `mkdir` also.

Table 1 shows that Windows commands, when compared based on their description only, have a small percentage of high similarity commands. 79.1% of the commands are in the low similarity ranges, which indicates that based on the description, Windows commands are spaced out from each other. But when these commands are compared based on their parameters, there are almost 41% of the commands with high similarity. On verifying this result, it appears that some commands are from a bigger group which have similar parameters set with each other, for example, `manage bde*` and `logman*`. When Windows commands' descriptions are extended with parameters, the result shows that 72% of the commands fall in the low similarity ranges and a mere 28% of the commands have one or more similar commands.

Comparing Linux command similarity percentage shows almost the same results as in Windows commands similarities for description and overall. Around 74% of the commands based on their descriptions fall in the low similarity ranges, whereas parameters similarities show that 31.5% of the commands have similar parameters. If we compare parameters

Table 1 Percentage of Windows and Linux Commands in Different Similarity Ranges.

Sim. ranges	Descr.	Params	Overall
Sim >=0.75 Windows	20.9%	40.7%	27.9%
Sim >=0.75 Linux	26.3%	31.5%	29.1%
Sim <0.75 Windows	79.1%	59.3%	72.1%
Sim <0.75 Linux	73.7%	68.5%	70.9%

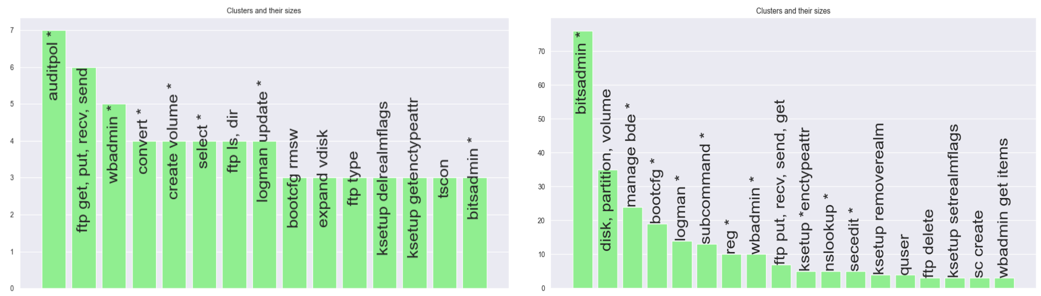
similarities result of Windows and Linux, there is a difference of almost 10%. One reason for this difference could be a high number of parameters for Linux commands which separate the commands quite well. Just like Windows commands, there are some groups of Linux commands also, such as `lttng*`, `pmda*`, and `systemd*`, which share almost the same parameters within their group, but other than these groups, parameters are quite different from each other. Overall, Linux commands show that 29% of them have one or more similar commands, whereas almost 71% of them fall in the low similarity ranges.

4.1 Windows Commands

As discussed earlier, Windows commands are compared based on the commands' description, parameters, and overall. Here we are sharing only the clusters of highly similar commands (in which we are interested) based on these three scenarios. The clusters are created by selecting a command and comparing their similarity score with other commands. If any other command has a similarity score of 0.75 or more with the selected command, an edge is created between the two commands. Applying this method for all the commands results in a graph where commands form communities (for the consistent connotations, we call them clusters). Some commands form a bigger cluster as it has already been discussed and some create a cluster of just two or three commands. Since showing hundreds of clustered commands in one graph reduces the readability, we selected the method of showing the count of commands in each cluster. Figure 2a shows the clusters of commands with high similarities based on the commands' descriptions. The clusters with at least three commands are selected to show the group of commands which share a similar description. The first and the biggest cluster in this analysis is made of commands such as `auditpool backup`, `auditpool get`, `auditpool restore` etc., whereas the second cluster is made of `ftp get`, `ftp send`, `ftp recv` etc. The interesting factor is to note the cluster seven which contains the commands such as `ftp ls`, `ftp dir`. This cluster is separated from cluster 2 which, though contains `ftp` commands, are doing a different job as compared to the commands in cluster six. This verifies that a simple but powerful approach, such as TFIDF along with Cosine Similarity measure, can successfully separate the commands based on their description.

Figure 2b shows the clusters of commands with high similarities based on the commands' parameters. The biggest cluster which has 76 commands in it, is of the `bistadmin*` commands which are used to create, download or upload jobs, and to monitor their progress. On manually verifying the Windows' manual pages, it proves that the `bitsadmin*` commands share the same parameters. The second cluster contains the commands such as `create partition`, `attributes disk`, `online volume` etc. The next clusters are made of commands of their groups, such as `manage bde*` makes a cluster of their own, whereas `bootcfg*` creates a separate cluster. Similarly, `logman*` commands have a different cluster, whereas `reg*` commands fall in a separate cluster.

The third scenario is the extension of the commands' description with their parameters. The clusters with at least 3 commands are selected only to see the groups of commands. Figure 3 shows that the biggest cluster is of `bitsamdin*` commands. By comparing the



(a) Description Similarities. **(b)** Parameters Similarities.

Figure 2 Windows commands clusters.

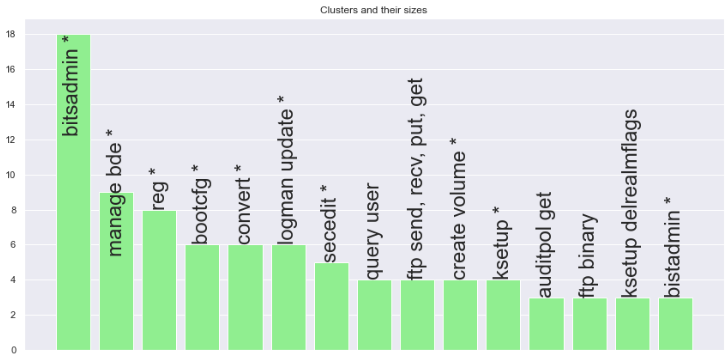
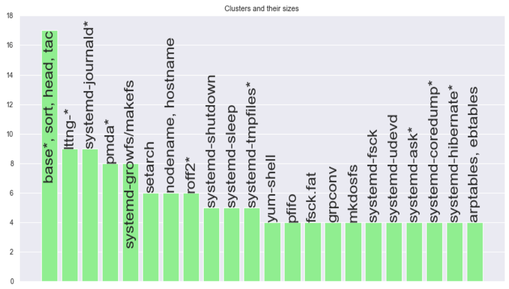


Figure 3 Windows commands clusters overall similarities.

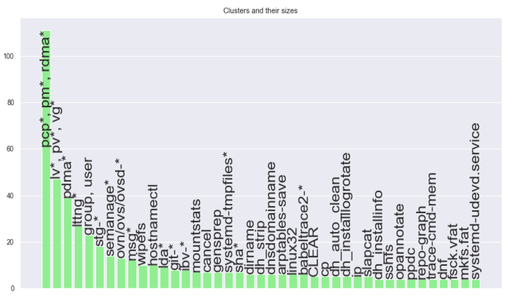
size of this cluster with the Figure 2b which is of windows' parameters similarities, it is worth noticing that comparing the `bitsamdin*` commands' complete textual-data, reduces the cluster size. A total of 76 `bitsamdin*` commands are highly similar when compared based on their parameters, but when we compare the complete text, the size of the cluster reduces to 18. This indicates that overall 18 `bitsamdin*` commands are similar to each other, but 76 `bitsamdin*` commands share the same parameters. These separate analyses help us distinguish the commands sharing the same parameters, the commands sharing the same description, and the similar commands overall. The rest of the clusters are formed by `manage bde*` commands, `reg*` commands, `bootcfg*` commands and so on.

4.2 Linux Commands

Following the same processes of calculating commands' similarities, Linux commands are compared. In the first scenario, commands' description are compared as shown in the Figure 4a. Only the clusters with at least 4 commands are considered for Linux analysis. The biggest cluster in this scenario is based on 17 different commands, such as `base32`, `base64`, `head`, `expand`, `sort`, `shuf` etc, which are sharing the same description. The `lttng*` commands such as `lttng-save`, `lttng-load`, `lttng-start`, `lttng-stop` etc. form a cluster of their own, whereas `pmda*` commands such as `pmdakernel`, `pmdalinux` etc. constitute a separate cluster. The interesting point in this scenario is the `systemd*` commands which are forming multiple clusters based on their jobs specifications. In the second scenario, commands' parameters are studied and their similarities are calculated.



(a) Description Similarities.



(b) Parameters similarities.

Figure 4 Linux commands clusters.

The commands with highly similar parameters are clustered together as shown in Figure 4b. A total of 111 commands which are from different groups, but share the same parameters, are making the biggest cluster in this scenario. The commands such as `pmlogsummary`, `pcp-vmstat`, `pmevent`, `pmdumplog`, and `pmstat` are examples of this cluster. A point worth mentioning here is that all 111 commands in this cluster do not necessarily need to be connected, as one command in this cluster can have a similarity of exact 1 with another command and at the same time a similarity of 0.8 with a third command. Now the second and third command can be disconnected from each other but they all are in the same cluster. As it was in Windows commands' parameters similarities, here also commands from the same group are making clusters of their own such as `stg*`, `msg*`, and `git*` commands.

The final scenario is the Linux commands' overall comparison. Figure 5 shows that the biggest cluster consists of 15 commands. It includes commands `lvmdiskscan`, `lvscan`, `pvresize`, `pvchange`,and `vgmerge` etc. These are all from Linux Superuser and system administration commands (i.e. man pages 8). The second cluster contains the commands such as `pcp2csv`, `pcp2json`, and `pcp2xml` etc. Similarly, `lda*` commands, `pmda*` commands, `git*` commands, and `yum*` are all in their own separate clusters. These results show that the combination of TFIDF and Cosine Similarity can separate the text quite well.

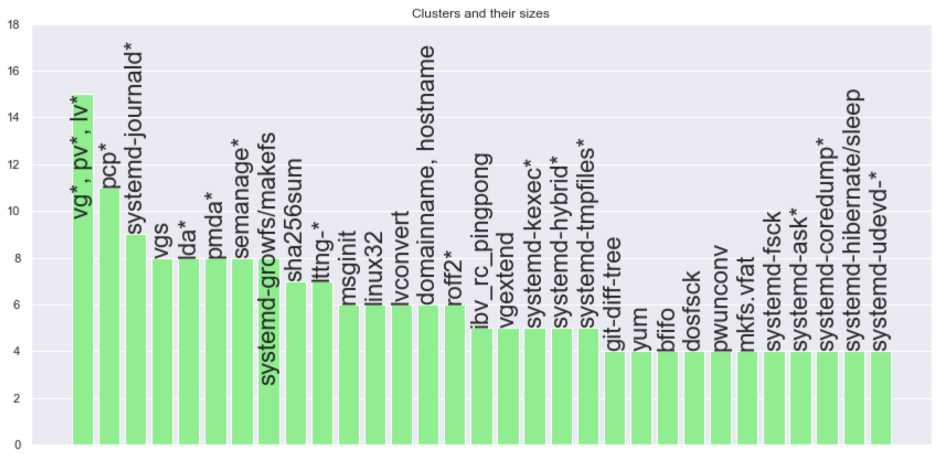


Figure 5 Linux commands clusters overall similarity.

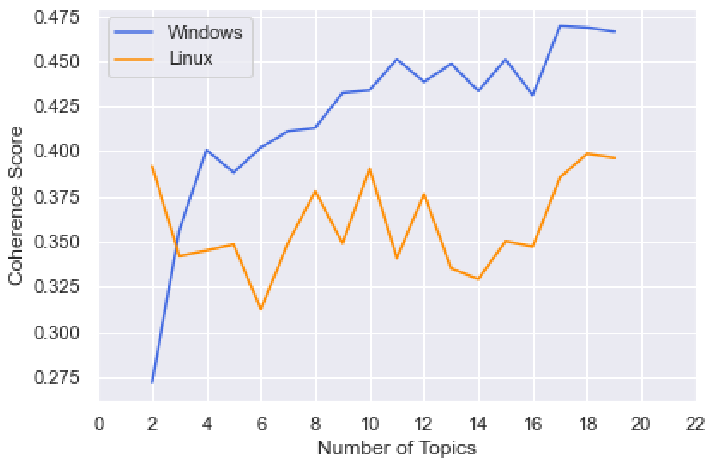


Figure 6 Number of Topics vs Coherence Score.

These results show that most of the Windows and Linux commands are not easily separable from their description as more than 60% of the commands fall between the similarity score of 0.25 and 0.75. The commands at the extreme ends are easily separable as either they have highly similar commands or they are unique. When commands are compared based on their parameters, the middle range ratios decreased for both Windows and Linux commands, indicating that commands (or group of commands) are more easily distinguishable from each other based on their parameters.

4.3 Clusters Validation With Topic Modeling

A topic model is a type of statistical model for discovering the abstract "topics" that frequently occur together in a collection of documents [1, 2]. Topic modeling is a frequently used text-mining tool for the discovery of hidden semantic structures in a text body [2].

To validate these clustering results, we applied topic modeling on Windows and Linux commands' descriptions, separately. We executed the model for various number of topics, starting from 2 up to 20. To find the optimal number of topics, we compared the coherence score against the specified number of topics. Figure 6 shows the number of topics on x-axis and coherence score on y-axis. The result indicates that the best coherence score for Windows and Linux can be achieved by setting the number of topics to 18. The topic modeling for Linux shows that the best coherence score can be achieved by setting the number of topic as 4 or 5, but with 2247 Linux commands, setting a low value as the number of topics, will not give us any useful insights. The coherence score is 0.465 for Windows OS commands, and 0.408 for Linux OS commands. Theoretically, coherence score of close to 1 is considered as optimal, but depending on the problem in hand and available textual data, optimal coherence score varies. In our case, we have a very specific vocabulary, which indicates the low diversity in the vocabulary. With this vocabulary, a coherence score of more than 0.4, is considered good, which is also verified in the Table 2, where four different commands (which belong to the same family) are given the same topic. Therefore, we used 18 as the optimal number of topics for both Windows and Linux commands.

For validation, we selected those commands which have one or more similar commands based on their descriptions. We passed each of these command's description to the model to get a topic number. For example, in Figure 2a, four `create volume *` commands are

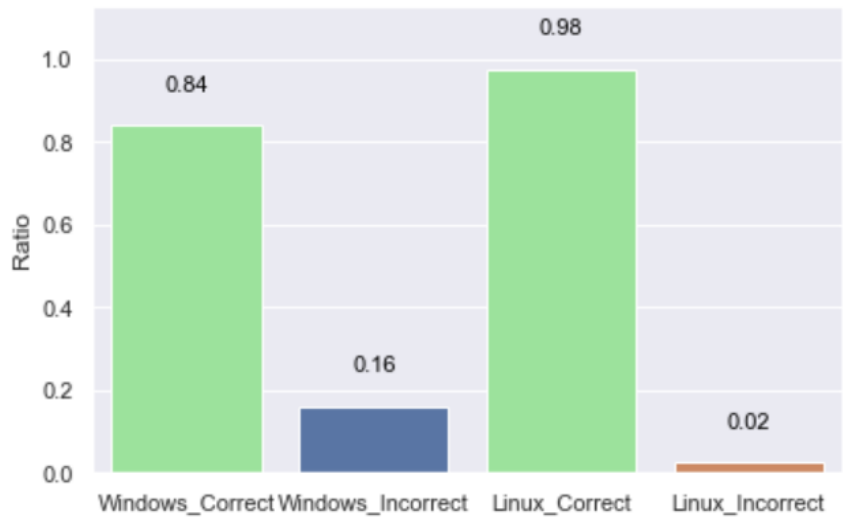


Figure 7 Ratio of Windows & Linux Commands Clusters Validated by Topic Modeling.

making a cluster of their own. We passed the description of these commands to our model, and the model suggested that these commands belong to Topic 1. Table 2 shows `create volume *` commands' description and the topic number our model is suggesting. By looking at the topics under Topic 1, we found the following terms: "`disk`", "`volume`", "`path`", "`dynamic`", "`load`", "`metadatum`", "`writer`", "`restore`", "`datum`", "`save`". Looking at the `create volume *` commands descriptions and the topics under Topic 1, we can say that our model is doing a fairly good job.

Table 2 Validating Topic Modeling for Windows Commands (create volume *).

Command	Description	Topic
create volume simple	Creates a simple volume on the specified dynamic disk. After you create the volume, the focus automatically shifts to the new volume.	1
create volume raid	Creates a RAID-5 volume using three or more specified dynamic disks. After you create the volume, the focus automatically shifts to the new volume.	1
create volume stripe	Creates a striped volume using two or more specified dynamic disks. After you create the volume, the focus automatically shifts to the new volume.	1
create volume mirror	Creates a volume mirror by using the two specified dynamic disks. After the volume has been created, the focus automatically shifts to the new volume.	1

Similarly, we validated clustering results of Windows and Linux commands based on their descriptions. Figure 7 shows that 84% of Windows commands and 98% of Linux commands which were in same clusters based on their description also share the same topics. The reason Linux commands show such a high ratio is because there are more than one variant of same command found in the documentation, like `clear` and `CLEAR` commands. We wanted to keep them in their original form as they both are used commonly. These results validate the clusters we created with TFIDF and cosine similarity.

4.4 Testing of Command Similarities

Once the similarity results are validated by the topic modeling, we created an ad-hoc scenario where we receive commands and compare them explicitly. For each pair of commands, if the command names are not same, we look in the $n \times n$ matrix of similarities we created in Section 3. A threshold value of 0.8 is specified, and if two different commands have a similarity score of more than 0.8, we classify them as “Similar”. Table 3 shows seven examples of Windows commands, where *Command_1* and *Command_2* are the two commands with different names, *Score* is the similarity score of two commands, and *Result* indicates whether two commands are similar or not. The names of the commands are bold in the table. It can be seen from the table that “query user” and “quser” are the different commands but performing the same task. Their similarity score is 0.989 which indicates that these two commands are aliases and a user can use them alternatively. Similarly, “ftp put” and “ftp send” are performing the same task. By looking in the similarity matrix, we found that their textual similarity is 0.9, which indicates that these two commands are also aliases and can be used alternatively. These are just seven examples, but there are hundreds of Windows and Linux commands which are aliases and should be considered as same commands when comparing them. Without the similarity matrix we created, any system measuring the similarities of commands will either classify the commands and their aliases as “Not-Similar” because of different commands names, or the system will need to be fed manually with all the commands and their aliases, which does not seem an optimal way of doing it. Official documentation[5] of Windows commands can be referred to manually confirm the commands (shown in the Table 3) and their textual descriptions.

Table 3 Testing Windows Commands’ Similarities.

Command_1	Command_2	Score	Result
C:\>**query user** user /server:Server64	C:\>**quser** user /server:Server64	0.989	Similar
cd Windows	**chdir** Windows	0.85	Similar
ftp> send d:\web2\script.py	**ftp> put** d:\web1\script.txt	0.9	Similar
del C:\examples\MyFile.txt	**erase** C:\examples\MyFile.txt	0.88	Similar
mkdir C:\test\temp	**md** C:\test \data	0.99	Similar
change port com12=com1	**chgport** com12=com1	0.884	Similar
ftp>get remote_file.py [local_file.py]	**ftp>recv** remote_file.py [local_file.py]	0.882	Similar

5 Discussion

Most computer vulnerabilities can be exploited in a variety of ways. Attackers may use a specific exploit, a misconfiguration in one of the system components, or a secret pathway to gain entry to a computer system. In most of these attacks, hackers run malicious programs through command line commands. One way to detect a hacker or malicious activities on a machine is by looking at sequences of program invocations and their parameters. The detection can be based on a combination of many different methods, from rule engines, the prevalence of the program’s invocation, reputation scores, to more advanced machine learning methods. The threat hunters focus on two tasks: first to detect all hacking or malicious activities as quickly as possible and second reduce the number of false positives to a bare minimum.

[5] `https://docs.microsoft.com/en-us/Windows-server/administration/Windows-commands/Windows-commands`

This research can help both of the mentioned scenarios. First of all the web-scraped program manuals will provide a good context database for common programs invocations on Linux or Windows operating system. Secondly, we were able to find program aliases and cluster program families, with the use of a simple and robust method. One could calculate the prevalence of the program clusters instead of individual programs to limit down the number of false-positive detection. Furthermore, the obtained clusters can be used as a validation benchmark for more sophisticated, unsupervised methods, that try to find similar programs based on program command parameters only. Defining similar programs manually would be a very tedious task due to the large number of common programs mentioned in the manuals, and an even bigger set of proprietary software used in the wild.

By enriching the program invocation commands with external sources of data not only provides us the necessary context of the programs but also helps us in understanding the user's intentions and behavior, which is an important factor to detect malicious activities.

This research work resulted in creating a reference database of the Windows and Linux commands. Primarily, this reference database will be used by cyber security specialists to learn about the commands and their aliases, but it can be used for any other use case or research work where the objective is to find prevalent commands and learn the common parameters among them.

6 Conclusion

This research work studies Windows and Linux commands. It helps us in finding the clusters of commands (small and larger groups), the ratios of commands which are isolated or highly similar to other commands, and to reach a better understanding of command similarity than what is possible by simply comparing the command strings as such. These results can be the basic building block of many applications and machine learning models in the cybersecurity domain. These results can also be useful for user profiling, predicting a set of parameters for different commands, and learning the commands and their set of parameters that invoke different programs.

References

1 Rubayyi Alghamdi and Khalid Alfalqi. A survey of topic modeling in text mining. *International Journal of Advanced Computer Science and Applications*, 6, January 2015. `doi:10.14569/IJACSA.2015.060121`.

2 David M. Blei, Andrew Y. Ng, and Michael I. Jordan. Latent dirichlet allocation. *J. Mach. Learn. Res.*, 3(null):993–1022, 2003.

3 Rainer Burkard, Mauro Dell'Amico, and Silvano Martello. *Assignment Problems*. Society for Industrial and Applied Mathematics, 2012.

4 Brian D. Davison and H. Hirsh. Toward an adaptive command line interface. In *HCI*, 1997.

5 Brian D. Davison and H. Hirsh. Predicting sequences of user actions. In *AAAI/ICML 1998 Workshop on Predicting the Future: AI Approaches to Time-Series Analysis*, 1998.

6 Najlah Gali, Radu Mariescu-Istodor, Damien Hostettler, and Pasi Fränti. Framework for syntactic string similarity measures. *Expert Systems with Applications*, 129:169–185, 2019. `doi:10.1016/j.eswa.2019.03.048`.

7 Jiawei Han, Micheline Kamber, and Jian Pei. 2 - getting to know your data. In *Data Mining (Third Edition)*, The Morgan Kaufmann Series in Data Management Systems, pages 39–82. Elsevier, third edition edition, 2012.

8 Haym Hirsh and Brian Davison. Adaptive unix command-line assistant. *Proceedings of the International Conference on Autonomous Agents*, October 1998. `doi:10.1145/267658.267827`.

9 José Iglesias, Agapito Ledezma Espino, and Araceli Sanchis de Miguel. Creating user profiles from a command-line interface: A statistical approach. In *International Conference on User Modeling, Adaptation, and Personalization*, volume 5535, pages 90–101. Springer, 2009.

10 E. Juergens, F. Deissenboeck, and B. Hummel. Code similarities beyond copy paste. In *2010 14th European Conference on Software Maintenance and Reengineering*, pages 78–87, 2010. doi:10.1109/CSMR.2010.33.

11 Jesse Kornblum. Identifying almost identical files using context triggered piecewise hashing. *Digital Investigation*, 3:91–97, 2006. The Proceedings of the 6th Annual Digital Forensic Research Workshop (DFRWS '06). doi:10.1016/j.diin.2006.06.015.

12 Benjamin Korvemaker and Russ Greiner. Predicting unix command lines: Adjusting to user patterns. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence*, page 230–235, 2000.

13 H. W. Kuhn. The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2(1-2):83–97, 1955. doi:10.1002/nav.3800020109.

14 J. Lawler and H.A. Dry. *Using Computers in Linguistics: A Practical Guide.* Routledge, 1998.

15 Xi Victoria Lin, Chenglong Wang, Luke Zettlemoyer, and Michael D. Ernst. Nl2bash: A corpus and semantic parser for natural language interface to the linux operating system. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation LREC 2018, Miyazaki (Japan), 7-12 May, 2018.*, 2018.

16 Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. *CoRR*, 2013. arXiv:1310.4546.

17 Vineeth G Nair. *Getting Started with Beautiful Soup.* Packt Publishing Ltd, 2014.

18 Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014. URL: http://www.aclweb.org/anthology/D14-1162.

19 M. Tufano, C. Watson, G. Bavota, M. Di Penta, M. White, and D. Poshyvanyk. Deep learning similarities from different representations of source code. In *2018 IEEE/ACM 15th International Conference on Mining Software Repositories (MSR)*, pages 542–553, 2018.

20 M. Umadevi. Document comparison based on tf-idf metric. In *International Research Journal of Engineering and Technology (IRJET)*, volume 7(02), 2020.

21 Antony Unwin and Hofmann Heike. Gui and command-line - conflict or synergy? In *Proceedings of the 31st Symposium on the Interface: models, predictions, and computing*, 2000.

Using Machine Learning for Vulnerability Detection and Classification

Tiago Baptista
Centro Algoritmi, Departamento de Informática, University of Minho, Braga, Portugal

Nuno Oliveira
Checkmarx, Braga, Portugal

Pedro Rangel Henriques
Centro Algoritmi, Departamento de Informática, University of Minho, Braga, Portugal

Abstract

The work described in this paper aims at developing a machine learning based tool for automatic identification of vulnerabilities on programs (source, high level code), that uses an abstract syntax tree representation. It is based on *FastScan*, using *code2seq* approach. *Fastscan* is a recently developed system aimed capable of detecting vulnerabilities in source code using machine learning techniques. Nevertheless, *FastScan* is not able of identifying the vulnerability type. In the presented work the main goal is to go further and develop a method to identify specific types of vulnerabilities. As will be shown, the goal will be achieved by optimizing the model's hyperparameters, changing the method of preprocessing the input data and developing an architecture that brings together multiple models to predict different specific vulnerabilities. The preliminary results obtained from the training stage, are very promising. The best *f1* metric obtained is 93% resulting in a precision of 90% and accuracy of 85%, according to the performed tests and regarding a trained model to predict vulnerabilities of the injection type.

2012 ACM Subject Classification Security and privacy → Vulnerability scanners; Computing methodologies → Machine learning

Keywords and phrases Vulnerability Detection, Source Code Analysis, Machine Learning

Digital Object Identifier 10.4230/OASIcs.SLATE.2021.14

Funding This work has been supported by FCT – Fundação para a Ciência e Tecnologia within the R&D Units Project Scope: UIDB/00319/2020.

Acknowledgements Special thanks to Search-ON2: Revitalization of HPC infrastructure of UMinho, (NORTE-07-0162-FEDER-000086), co-funded by the North Portugal Regional Operational Programme (ON.2-O Novo Norte), under the National Strategic Reference Framework (NSRF), through the European Regional Development Fund (ERDF).

1 Introduction

Nowadays, information systems are a part of almost every aspect in life and furthermore, almost every company is dependent on the liability, safety and security of a software application. So, it is essential to have the capability to identify and correct pieces of code that contain known vulnerabilities in order to prevent the software from being compromised.

Cybernetic attacks are a constant and present a real threat to companies and people in general, since nowadays almost every device has an Internet connection. As a consequence, devices are exposed to external threats that try to exploit vulnerabilities.

A vulnerability is a flaw or weakness in a system design or implementation (the way the algorithms are coded in the chosen programming languages) that could be exploited to violate the system security policy [13]. There are many types of vulnerabilities and many approaches to detect such flaws and perform security tests. All the known approaches present

© Tiago Baptista, Nuno Oliveira, and Pedro Rangel Henriques;
licensed under Creative Commons License CC-BY 4.0

pros and cons but none of them stands as a perfect solution. Static analysis is one of the approaches and it can be defined as the analysis of a software without its execution. Static Application Security Testing (SAST) is an application security testing methodology that allows detecting vulnerabilities at the early stages of software development. It is adopted by many companies, such as Checkmarx[1]. These methodologies have many strengths such as the ability to find vulnerabilities without the need to compile or run code, offering support for different programming languages and being able to easily identify common vulnerabilities and errors like Structured Query Language (SQL) injections and buffer overflows. Despite this, there are still problems with the referred approach, mainly in the production of a great number of false positives, in the lack of identification of the vulnerability type and even performance issues.

There are many tools that implement the concept of static analysis and apply it to vulnerabilities detection. Some tools rely on only lexical analysis like *FlawFinder*[2] [10] but have the tendency to output many false positives because they do not take into account the semantic [4]. Other tools like *CxSAST*, Checkmarx SAST tool[3], overcome this lack by using the Abstract Syntax Tree (AST) of the program being evaluated. In this context, another challenge is to create and apply the same tool to different languages, so that one can clearly identify vulnerabilities with high accuracy and have good performance with big inputs.

To overcome the flaws identified in the *SAST* approach, decreasing the number of false positives and the processing time, a new approach came to the researchers mind: to integrate machine learning techniques. The idea can be realized by altering and tuning open source projects, namely *code2vec* and *code2seq* in order to try to identify accurately and efficiently than other tools like *CxSAST* [9]. *code2vec* main idea is to represent a code snippet as a single fixed-length code vector, in order to predict semantic properties of the snippet. On the other hand *code2seq* represents a code snippet as a set of compositional paths and uses attention mechanisms to select the relevant paths while decoding [1, 2]. The resultant approaches relying on *code2seq* and *code2vec* are called *FastScan* and were not one hundred percent success but opened the path to further investigation [9].

It is clear that a good analysis tool can help spot and eradicate vulnerabilities, furthermore, it is becoming a part of the development process. But, there is still room for improvement and all the research work done in this area can be of uttermost relevance for the industry.

With all of the previous in consideration the main contributions expected from the research project that will be discussed along the paper are:

- **The improvement of the *FastScan* approach**:
  - Find the best hyperparameters for each case study;
  - Use these hyperparameters to improve evaluation metrics for the models (precision, recall and *f1*);
- **The development a specific model for each type of vulnerability**, that is capable of identifying if a code snippet has a vulnerability or not (Boolean model) of a given type;
- **The design of a proper architecture to develop a general model** capable of identifying the occurrence of vulnerabilities and their type, given a code snippet;

Since it has different objectives and it is an evolution of *FastScan*, from now on, when referring the work, it will be called ***new FastScan***.

[1] `https://www.checkmarx.com/`
[2] `https://dwheeler.com/flawfinder/`
[3] `https://www.checkmarx.com/`

This paper is divided in seven sections, in the first section will be presented the context and motivation, in the second section the current state of the art in vulnerability detection, the third section will detail the proposed approach, including architecture, the fourth section with more details about the developed work, the fifth section refers to the specification for the first phase (that will be presented in Section 3), then section six will approach the results and the last section with conclusions and future work.

2 Vulnerability Detection

The main spark that ignited the work described in this paper, and the work from specialists around the world, is software vulnerability. It is necessary to understand the concept of vulnerability and other main concepts related to them in order to understand all the methodologies presented next. A vulnerability is defined as a flaw or weakness in a system design, implementation, or operation and management that could be exploited to violate the system's security policy [12]. These flaws might be system design, development, or operation errors and its exploit can lead to harmful outcomes. This perspective of harm or loss of information is normally called risk. Nowadays, these concepts have become common in any software development process because with the presence of electronic devices running software in almost every area of our society, there is the need to eliminate, or at least minimize, the occurrence of situations that can risk the security of data or operations.

There are different types and classifications of vulnerabilities, provided by different contributors. They expose discovered vulnerabilities, exploits, and solutions in online databases[4]. Even though there are many sources and knowledge about vulnerabilities, exploits still occur.

Open Web Application Security Project (OWASP)[5] foundation, that works towards a more secure software development process and applicationss, has a list of the top ten security risks on applications. Since one of the objectives of this paper work is to correctly identify types of vulnerabilities using machine learning models, first it is necessary to know them and understand how they use security breaches to harm systems.

Since this top ten is intended to identify the most serious security risks on web applications, and it is important to know them in-depth in order to know how to identify them, manually or automatically, and even more realise what consequences can they bring to the system. Since the results presented are related to a specific vulnerability it is important to know it more deeply.

2.1 Injection attack

An injection attack refers to an attack where untrusted data is supplied as input to a program. This input is then processed and changes the application's expected behaviour. Normally, this vulnerability is related to insufficient user input validation. Since this is a well known and one of the oldest exploits, that has some automatic tools in order to exploit without having much knowledge, makes this one of the most common and dangerous vulnerability. There are many types of injections, namely Code injection, Email Header Injection, Host Header Injection, Lightweight Directory Access Protocol (LDAP) Injection, SQL injection among others.

[4] `https://cve.mitre.org/`
[5] `https://owasp.org/www-project-top-ten/`

Next is shown an example for SQL injection, where an attacker sends `OR 1=1--` instead of a valid id, as shown in Figure 1. Without an input validation, this query will return all data contained in the table in this case the table `accounts`.

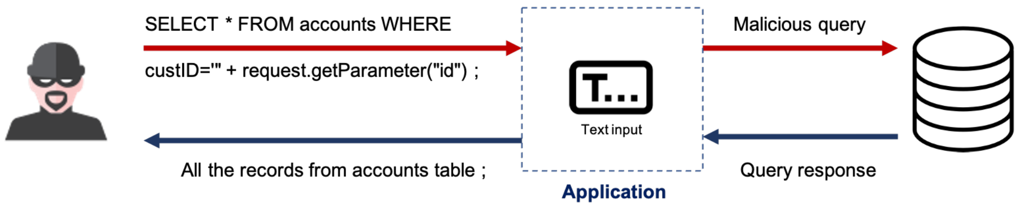

SELECT * FROM accounts WHERE
custID='" + request.getParameter("id") ;
All the records from accounts table ;
T...
Text input
Application
Malicious query
Query response

Figure 1 Schema explaining injection.

In many of these affected software development projects, there is no application of formal and systematic methods to execute source code auditing and testing. Auditing can be defined as the process of analyzing application code (in source or binary form) to uncover vulnerabilities that attackers might exploit [7]. This process is important because allows scanning the source code covering all the paths that normal testing might not cover in order to try to detect possible vulnerabilities.

2.2 Detection techniques

Having in mind the big issue in software development presented before, there are many solutions that try to eliminate it by making an in-depth code auditing. Starting with the concept of defensive programming, that refers to the practice of coding having in mind possible security problems that can occur [5]. This is clearly a good practice but one that does not solve the vulnerability related problems. Even good and experienced programmer cannot know how to prevent all the exploits created for certain Application Programming Interface (API), library or other. Also, languages by construction were not build thinking on the way an attacker would take advantage of certain nuances. Taking as an example, the buffer overflow exploits, one the most common problem reported in vulnerabilities databases. This can occur due to mishandling of inputs but in its core is allowed by the construction of the language. Has it is easily confirmed by C or C++ that do not provide any built-in protection against accessing or overwriting data in memory [6]. Since good intentions and practices are not enough to solve this problem, there was the need to apply and develop other techniques that will be briefly presented next:

- **Static analysis for security testing (SAST):** Static analysis methods are a resource for identifying logical, safety and also security vulnerabilities in software systems. They are normally automatic analysis tools and intend to avoid manual code inspections in order to save time and avoid the investment of resources in manual tasks that could be fruitless [11].
- **Dynamic analysis for security testing (DAST):** In order to find vulnerabilities, testing seems to be the easiest path to follow, and that is what DAST stands for. In DAST, a program is executed in real-time in order to test and evaluate it. In DAST the tested software is seen as a black box, and the tools or person performing the tests only interact with the application or software as users that have no knowledge of its internal operations.
- **Interactive analysis for security testing (IAST):** Contrarily to DAST, IAST aims to find vulnerabilities by analysing software from within such as SAST. But contrarily to SAST and similarly to DAST, IAST executes the analysis with the software running.

IAST uses different instruments to monitor a running application in order to gather information about the internal processes. These tools try to mitigate SAST's and DAST's limitations, namely, identify the specific place where a bug/vulnerability is located. By running from the software's inside has leverage over DAST and allows testers to define what paths or functionalities to cover, this can lead to miss-handled bugs/vulnerabilities but if well thought can lead to gains in time and work efficiency contrarily to SAST that has full coverage over the software.

- **Machine Learning in vulnerability identification:** Artificial intelligence and more specifically machine learning and deep learning systems are being used to automate many different tasks and in different areas, with success. For example in image recognition, disease behaviour prediction, traffic prediction, virtual assistant, among others. The area of software security is not an exception, as referred on the previous sections, current vulnerability identification tools and in general, all security tools have flaws and many rely on a great amount of manpower and are very time-consuming. In order to try to tackle such limitations and with the rise of many machine learning applications, there were many investigation focused on applying such techniques in the software security area. There are many examples of such applications, but for this paper purposes, the next sections focus on the application of machine learning and deep learning for vulnerability identification [3].

3 Proposed approach

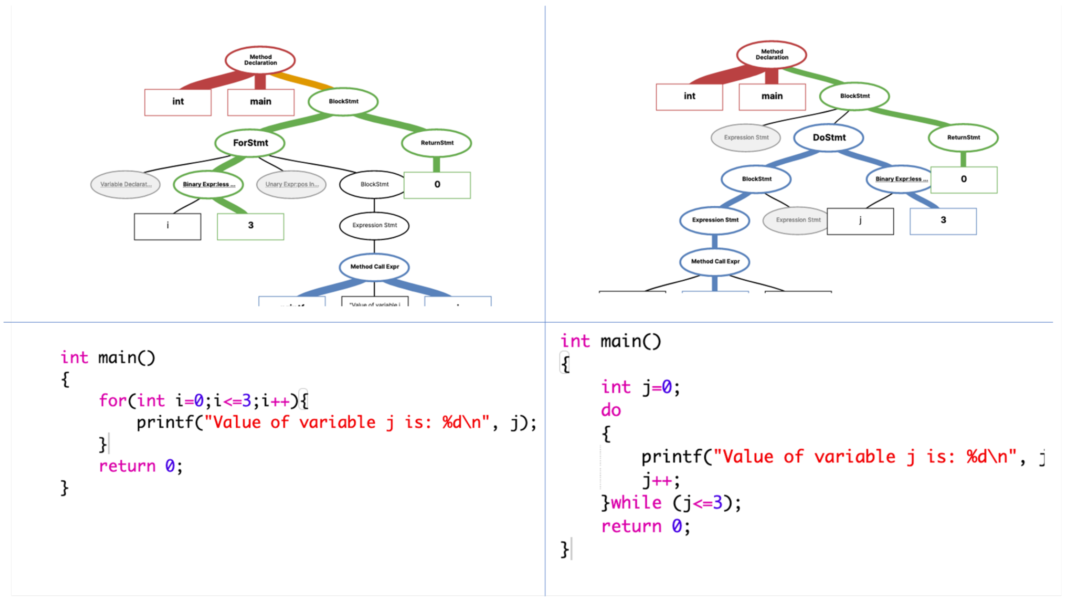
In this section, it is detailed the proposed approach in order to fulfil the objectives detailed in Section 1. It will be presented the system architecture and a high-level representation of the system flow of data in order to better perceive all the involved processes.

3.1 System Architecture

In order to understand why *code2seq* was chosen in *Fastscan* and remained in *new FastScan* it is first necessary to understand *code2seq* structure and applications. *code2seq* creates a representation of source code using AST and then uses it to infer properties. An AST represents a source code snippet in a given language and grammar. The leaves of the tree are called terminals and the non-leafs are called non-terminals. It represents all the variables declarations, operator, conditions and assignments. In order to represent code in this structures it is necessary to create sequences of terminal and non terminal nodes and consider all possible paths between nodes.

This representation has some significant advantages over the use of simple code tokenisation, when compared in terms of code comparison. Namely when trying to find two methods that have the same functionality but different implementations. Having the AST enables a better comparison, since both functions paths will be similar, as represented in Figure 2 . So functions will have different token representations but similar path representation only differing in the *Block* statement.

In a simplified overview *code2seq* uses encoder-decoder architecture that reads the AST paths instead of the tokens. In the encoding end, there is the creation of vectors for each path using a bi-directional Long short-term memory (LSTM) and the extraction of tokens, where the AST terminal nodes are transformed into tokens and these tokens are split into subtokens (for example, an ArrayList is transformed into the tokens Array and List). In the end a decoder uses attention models to select relevant paths.



```
int main()
{
    for(int i=0;i<=3;i++){
        printf("Value of variable j is: %d\n", j);
    }
    return 0;
}
```

```
int main()
{
    int j=0;
    do
    {
        printf("Value of variable j is: %d\n", j
        j++;
    }while (j<=3);
    return 0;
}
```

Figure 2 Clarification *code2seq* AST representation.

To accomplish the development of a specific model for each type of vulnerability, that is capable to identify if code snippet has a vulnerability or not (Boolean model), it is proposed to follow the architecture and flow presented in Figure 3. This approach is a refinement of Ferreira's work with *code2seq* named *FastScan* [9].

In the first phase, it is essential to use *CxSAST* in order to obtain the input for the preprocessing – since it converts several languages into a generic internal representation, allowing to create a tool language independent. Since the focus is on creating a model capable of identifying certain vulnerabilities then it is necessary to filter *CxSAST* output in order to correctly train the model. Then use *code2seq* normal pipeline in order to obtain a model able to identify a specific vulnerability. The goal is to have at least a model for each of *OWASP* top 10 vulnerabilities.

Having completed the first phase, it is necessary to develop a general model capable of identifying if there are vulnerabilities and it's type, given a code snippet. In order to complete this objective it is proposed to follow the architecture and flow presented in Figure 4. In order to take advantage of the work done in the first approach it is proposed to use the previously trained models and combine them. This combination might be done using ensemble technique or only by running *code2seq* testing phase in parallel.

In order to obtain the best performance wise results, it is expected to test the developed pipeline using the most powerful hardware resources available at the University of Minho's cluster[6].

[6] http://www4.di.uminho.pt/search/pt/equipamento.htm

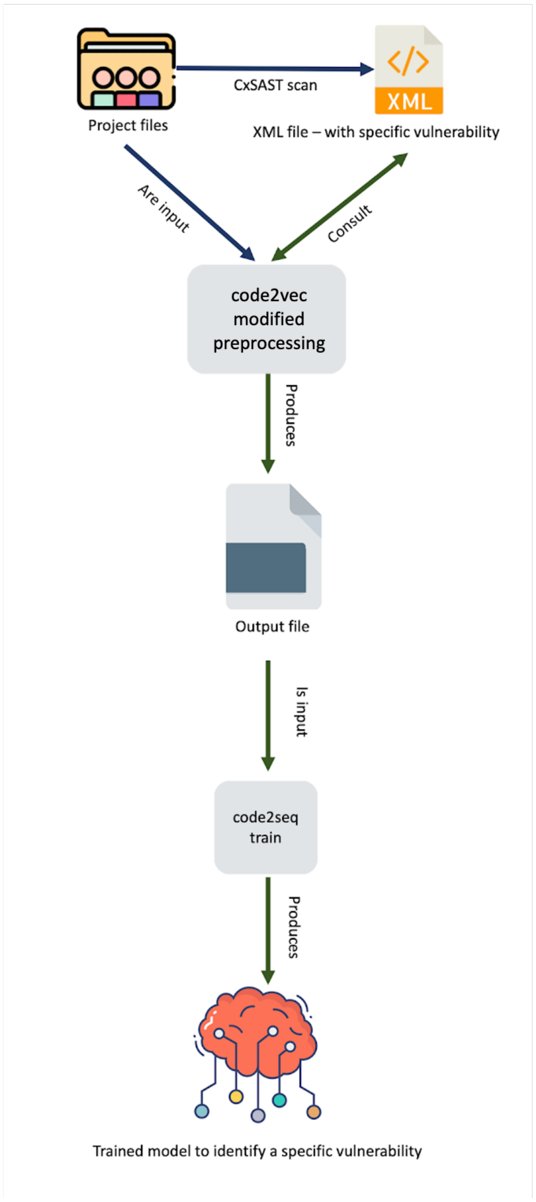

Project files
CxSAST scan
XML
XML file – with specific vulnerability
Are input
Consult
code2vec modified preprocessing
Produces
Output file
Is input
code2seq train
Produces
Trained model to identify a specific vulnerability

Figure 3 First phase architecture.

4 Development

The work that led to this paper, *FastScan*, had promising results and showed the potential on using machine learning for detecting vulnerabilities in source code having as a base the open source project *code2seq*. Nevertheless, it left open for investigation some issues, namely the ones that are a part of this paper objectives. To tune *code2seq* parameters and to develop a way to identify specific types of vulnerabilities in source code.

To address this issues, the first phase began with the the *FastScan* code as the base. On which it was build a solution that it is able to produce a model capable of identifying if there are vulnerabilities of a specific type given a source code input. The conceptual approach is the same as in *FastScan* but with the modifications to direct the prediction to a specific type of vulnerability and it was done the hyperparameters optimization, applied to the input dataset, in order to improve the model accuracy.

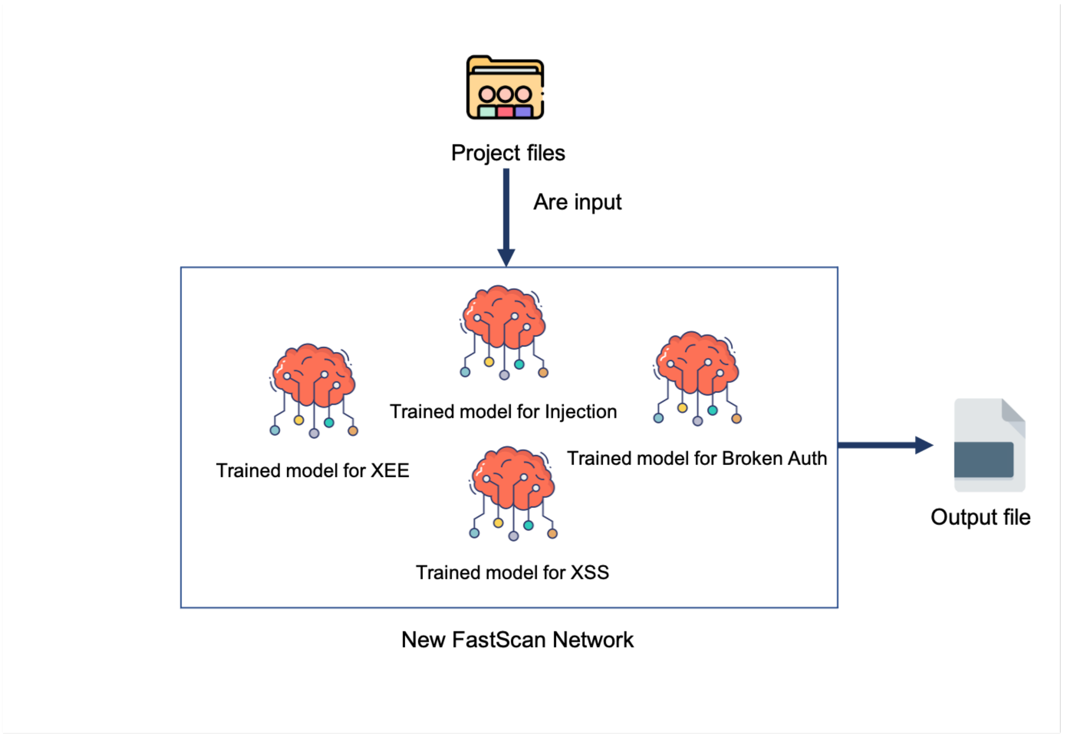


Figure 4 Second phase architecture.

With all this in mind, this section will describe the different phases that were explored during the presented work. The first section will describe the datasets, then the hardware and technical details on which the experiments were made and the following sections will detail the work done in the first phase. Explaining the steps in each one, the challenges, experiments and the obtained results for further analysis in the next section.

4.1 Datasets

The datasets take a major part in this project, because all the developed work has no utility unless there is enough good data to train the models.

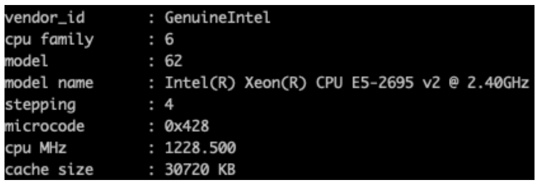
The first dataset which, from now on, will be referenced as *dt01*. *dt01* is composed by **43** different projects, there is the original source code and for each project there is an XML file with detected vulnerabilities. This XML file was provided by *Checkmarx* and it is the output of their static analysis tool *CxSAST*. With one important detail, the output was validated by humans which means that there are no false positives.

4.2 Hardware and technical details

Since in *FastScan*, the low computational power was identified has a major barrier and setback to the developed work, since preprocessing and training tasks were slow and implied much waiting time, it became clear that the use of a regular personal computer was not enough to achieve the desired results in the experiments of this paper [9].

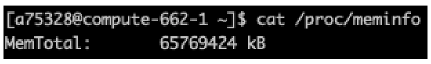
This barrier was overcame with the use of the University of Minho cluster[7]. More specifically using a node with the following specifications:

- **CPU as seen in Figure 5:**
  - Intel® Xeon® Processor E5-2695 v2[8];
  - twelve cores ;
  - twenty four threads.
- **RAM:** Sixty four GB as seen in Figure 6 ;
- **GPU:** Two NVIDIA® TESLA® K20M[9].



```
vendor_id       : GenuineIntel
cpu family      : 6
model           : 62
model name      : Intel(R) Xeon(R) CPU E5-2695 v2 @ 2.40GHz
stepping        : 4
microcode       : 0x428
cpu MHz         : 1228.500
cache size      : 30720 KB
```

Figure 5 CPU details.



```
[a75328@compute-662-1 ~]$ cat /proc/meminfo
MemTotal:       65769424 kB
```

Figure 6 Memory details.

There were problems installing all the needed dependencies to run *new FastScan* natively, because of the security constraints in the cluster system. Namely, there were no root permissions, so the installation of dependencies such as *Java*, python and its libraries were a great technical obstacle. In order to overcome this, it was used docker and all its potentialities, since docker presented a good solution because it is consistent and makes all this installation process easy by using a base image with *Java* and adding all the dependencies needed without further concerns.

It is important to notice that this solution was followed after the confirmation that there was no significant overhead or performance loss introduced by the use of docker and that it was a better solution than a virtual machine [8].

5 First Phase

In this section is explained the first phase referred previously in Section 3.1. This phase intends to receive as input a certain dataset containing *Java* projects and an Extended Markup Language (XML) file from CxSAST containing identified vulnerabilities to output a trained model for a specific and predetermined vulnerability .

5.1 Data filtering

This step was of great importance and it was not present in *FastScan*. Since the objective of new *Fastscan's* the first phase is to create a model capable of detecting a specific vulnerability, it is firstly needed to filter the original dataset input data by vulnerability type. So that it is possible to train a model to predict only a specific type of vulnerability.

[7] `http://search6.di.uminho.pt`
[8] `https://ark.intel.com/content/www/us/en/ark/products/75281/intel-xeon-processor-e5-2695-v2-30m-cache-2-40-ghz.html`
[9] `https://www.techpowerup.com/gpu-specs/tesla-k20m.c2029`

Before knowing how it was achieved, it is important to understand the two different components from the dataset:

- Source code files: This are the original files from several different projects;
- XML files: This are the files generated by the *CxSAST* from *Checkmarx*. There is one file for each project, on which are registered the vulnerabilities detected by the tool.

This filtering process was developed using *python* and aims to filter the XML files of each project, in order to keep only the ones that refer to the vulnerability that is desired to train a model. In order to achieve this, it was developed a python program that is able to read and parse the XML (provided by *CxSAST* – and it is built to be applied to its specific structure) and filter the vulnerabilities entries by the vulnerability name. It is not a literal search, it can be send arguments to the program with the range of words/strings to be searched in the vulnerability name field of the XML entries. In the end, the output is a set of XML files only with the entries for a specific vulnerability and ready to be ingested in the next step – Preprocessing.

5.2 Preprocessing

This step was not changed from the *FastScan*, but it is different from the original *code2seq*. The preprocessing (which is performed on the train and test data) returns a file with the dataset labeled and the *AST* that represents the input dataset. The source code files and XML files are parsed , in the preprocessing, which is built in *Java*. There is the creation of the *AST* from the input of the source code files and this was not modified from the original *code2seq*. But there is another step required to obtain the label for each method, the parsing of the XML files.

There is also performed the parsing from the *XML* file (a new step added in *New FastScan*). In this second it is relevant to refer that it is stored the vulnerabilities registered in the file, namely the name of the vulnerability, the start and end line and column in the file and also the filename and path within the project.

After the parsing stage, for each method (that resulted from the source code parsing) there the verification if it occurs in the register of vulnerabilities obtained by the *XML* parsing, the search is achieved by comparing the filename and path within the project which is stored in the result of the *XML* and also the source code parsing. Then it is registered the presence or not of vulnerabilities, with a *boolean*.

This combination of the parsing of the *XML* (with the results of the *CxSAST*) and the original source code leads to the output of the preprocessing, that is constituted by a text file that has a line for each method, on which the first element is the label indicating the presence of vulnerabilities followed by the *ast* paths.

On the original *code2seq* preprocessing, in the output file the first entry was the function's name – the first entry is the label that will be used in the training phase. But *New Fastscan* preprocessing was modified in order to make the first entry of each line a boolean that indicates the presence or not of a vulnerability in the method. This change was enough to modify the prediction of the model obtained in the next training phase, since the training is now gonna be done with the boolean has the prediction label instead of the previous string that was the method name.

5.3 Hyperparameter optimization

This step was important to tackle a problem detected in *FastScan*, without the optimization of the hyperparameters the final model performance might not be the best possible [9].

In order to understand the following section, firstly it is important to know the notions of accuracy, precision, recall and *f1*.

- Accuracy is the most known performance measure and it is the result of a ratio between correctly predicted observation to the total observations. But it can be misleading if there are not the same number of false positives and false negatives, so it is important to explore other metrics;
- Precision refers to the ratio between correctly predicted positive observations and the total predicted positive observations. This metric answers the question of from the methods labeled as having a vulnerability, how much of them actually had one? A high precision means a low occurrence of false positives;
- Recall is often called sensitivity, it answers the question of, from the methods that really had a vulnerability how many of them were labeled?;
- The *f1* metric is the weighted average between precision and recall, this is the most balanced meausure of all the preseted since takes false positives and false negatives into account. This metric is usefull when there is uneven distribution in the dataset, in the case, where there is not the a close number between vulnerable and not vulnerable entries.

To do so, it was used *wandb*[10] – It is a python library, used to monitor, compare, optimize and visualize machine learning experiments. It can be integrated with many frameworks including *tensorflow* (the python library used in the training) and it was of great importance in the development of this work. It implements different search methods in order to obtain the best parameters to increase a specific evaluation metric. This library allowed to solve the hyperparameter tuning which is a very complicated problem that normally requires experience in the field and complicated algorithms [14].

This library allowed the optimization of the models hyperparameters for a specific dataset by using *sweeps* that allow to find a set of hyperparameters with optimal performance and apply different methods to do so, such as grid, random and Bayesian[11]. It also allows to visualize performance and time metrics from the experiments, that can be consulted in Section 6.

The chosen method was the Bayesian method because it guaranteed the best possible combination without compromising time and performance. There were other methods such as random search where the parameters are being chosen randomly from a specific range – this method could be even faster but it would not guarantee the best hyperparameters since it does not cover all the combinations and it does not have a method to improve its results. Other method could be the grid search, in this method all the possible hyperparameter combinations are tested, in this case it would be of great time and processing cost given the number of hyperparameters.

The library applies the Bayesian method by building a probabilistic model that maps the value of each hyperparameters with the values to optimize – that could be accuracy, f1 and precision. Each hyperparameter value used in the algorithm iteration and the obtained results are taken into account into the next iterations. This way the search for the best hyperparameters is faster because it is being guided by the previous experiences.

[10] `https://wandb.ai/site`
[11] `https://docs.wandb.ai/guides/sweeps`

5.4 Train

This step referees to the effective training and production of the models. This is performed the same way has in the origial *code2seq*, relying on the developed approach that uses tensorflow in order to perform all the process.

6 Testing and Results

In this section will be presented the results obtained so far. The presented results were obtained using the dataset previously referred. The dataset was split in the following way:

- 75% to train = 32 projects;
- 15% to test = 7 projects;
- 10% to validate = 4 projects.

Regarding the data filtering step, it was applied the filter of injection, in order to filter the results for this specific vulnerability so that it is possible to train the model only to identify injection vulnerabilities.

Then the preprocessing was applied to the the source code and the filtered XML file obtaining the preprocessed input to train and test the final model. The total number of examples (paths) obtained from the preprocessing from the dataset was 191813.

The next phase was the hyperparameter optimization, firstly it was attempted to run the script to obtain the best parameters that maximize precision. After analysing the results in Table 1 it was observed that it was possible to obtain a really high precision but at the cost of very low recall. A model with high precision but low recall returns a low count of results, but most of the predicted labels are correct when compared to the training labels on the other hand a model with high recall but low precision returns a high count of results, but most of the predicted labels are incorrect when compared to the training labels. After this consideration it was clear that a vulnerability prediction system must have a balance between both because it is important to correctly identify a vulnerability but it is also important not to overlook one.

This balance can be obtained by using the *f1* metric, this metric is a weighted mean of the precision and recall and being so if in the optimization is focused in optimize the *f1* value then it is guaranteed the balance it is searched between precision and recall as seen in Equation 1.

$$f1 = 2 * precision * recall/precision + recall \tag{1}$$

So the optimization was done trying to maximize the *f1* metric. This was applied through the use of *wandb* sweeps, where it was defined that the value to optimize was the f1 as referred in Section 5.3.

After having the sweep results, a model was trained using the best hyperparameters that can be consulted in Table 1.

The final model for injection using the *dt01* dataset had 85% of accuracy, 90% of precision, 97% of recall leading to an *f1* of 93% and it was achieved in the first epoch as seen in Figure 7. This was the chosen, according to the criteria explained before, because it had the best score regarding the f1 *metric*. After this the model training leads to the increase of precision, as expected in a training process, but with the lowering of recall values and therefore a lower *f1* value.

Table 1 Best Hyperparameter for *dt01*.

BATCH_SIZE	BEAM_WIDTH	BIRNN	CSV_BUFFER_SIZE
127	0	true	104857600
DATA_NUM_CONTEXTS	DECODER_SIZE	EMBEDDINGS_DROPOUT_KEEP_PROB	EMBEDDINGS_SIZE
0	302	0.4162172547338106	193
MAX_CONTEXTS	MAX_NAME_PARTS	MAX_PATH_LENGTH	MAX_TARGET_PARTS
234	8	10	6
NUM_DECODER_LAYERS	NUM_EPOCHS	PATIENCE	RANDOM_CONTEXTS
1	3000	4	true
READER_NUM_PARALLEL_BATCHES	RELEASE	RNN_DROPOUT_KEEP_PROB	RNN_SIZE
1	false	0.7488847205115016	256
SAVE_EVERY_EPOCHS	SHUFFLE_BUFFER_SIZE	SUBTOKENS_VOCAB_MAX_SIZE	TEST_BATCH_SIZE
1	10000	151701	256
TARGET_VOCAB_MAX_SIZE			
27000			

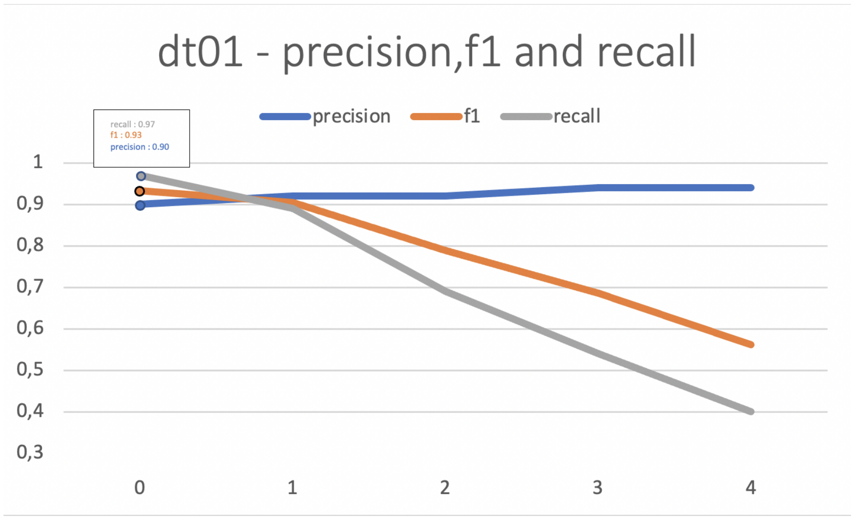


Figure 7 Model evaluation.

7 Conclusion and Future Work

This section is intended to close the paper, summarising the outcomes reached so far. The first section contains the context on vulnerability detection, the motivation and objectives of the project. The second section is a literature review on vulnerability detection. The outcomes of the reported stage provided the foundations for the proposed approach. The third section presents and discusses the our working proposal. The fourth section explains the development and includes the presentation of the dataset used for training as well as describes the hardware details. The fifth section discusses the implementation. Finally the sixth section analyzes the training results obtained when testing models.

Taking into account the results from this first experiment, it becomes clear that the hyperparameter optimization has improved the results in the increase the precision and the other metrics. Also the train only for a specific vulnerability might as well have an influence since the train for a more strict purpose is more effective, namely in this case. While *Fastscan* attempts to predict the presence of many types of vulnerabilities, *new Fastscan* aims at creating models to predict a single type of vulnerability, gathering the parts into a global analyzer in a final system.

Applying this model as a scanner that verifies projects before it goes through a *SAST*, other tool or manual verification could represent a great improve in terms of spent time, processing and manual work since it could eliminate the projects without vulnerabilities from further scanning. Comparing to the traditional tools this approach requires less results verification and promises better accuracy.

Summing up, the concept of vulnerability was presented and explained, the different methods to identify them were discussed. Moreover, the improved approach and its first promising results were described and analyzed. Also, it is important to highlight the effort that done in the adaptation and installation of algorithms and programs to run the system in parallel platform offered by the SEARCH cluster available in the Informatic Department of University of Minho to be possible to train the models in acceptable time.

The next step planned to carry on the project is the training of models for other vulnerability types in order to accomplish the second objective, as listed in the Section 1. After that, it will be necessary to implement the architecture presented in Section 3.1, in order to obtain a system capable of predicting the presence of vulnerabilities of different types.

It also is important to investigate if there is the possibility to reduce the computational cost with the mix of traditional and machine learning techniques.

References

1 Uri Alon, Shaked Brody, Omer Levy, and Eran Yahav. code2seq: Generating sequences from structured representations of code. *arXiv preprint*, 2018. `arXiv:1808.01400`.

2 Uri Alon, Meital Zilberstein, Omer Levy, and Eran Yahav. code2vec: Learning distributed representations of code. *Proceedings of the ACM on Programming Languages*, 3(POPL):1–29, 2019.

3 Philip K Chan and Richard P Lippmann. Machine learning for computer security. *Journal of Machine Learning Research*, 7(Dec):2669–2672, 2006.

4 Brian Chess and Gary McGraw. Static analysis for security. *IEEE security & privacy*, 2(6):76–79, 2004.

5 Brian Chess and Jacob West. *Secure programming with static analysis.* Pearson Education, 2007.

6 Crispan Cowan, Calton Pu, Dave Maier, Jonathan Walpole, Peat Bakke, Steve Beattie, Aaron Grier, Perry Wagle, Qian Zhang, and Heather Hinton. Stackguard: Automatic adaptive detection and prevention of buffer-overflow attacks. In *USENIX security symposium*, volume 98, pages 63–78. San Antonio, TX, 1998.

7 Mark Dowd, John McDonald, and Justin Schuh. *The art of software security assessment: Identifying and preventing software vulnerabilities.* Pearson Education, 2006.

8 Wes Felter, Alexandre Ferreira, Ram Rajamony, and Juan Rubio. An updated performance comparison of virtual machines and linux containers. In *2015 IEEE international symposium on performance analysis of systems and software (ISPASS)*, pages 171–172. IEEE, 2015.

9 Samuel Gonçalves Ferreira. Vulnerabilities fast scan - tackling sast performance issues with machine learning. Master's thesis, University of Minho, 2019.

10 Rahma Mahmood and Qusay H Mahmoud. Evaluation of static analysis tools for finding vulnerabilities in Java and C/C++ source code. *arXiv preprint*, 2018. `arXiv:1805.09040`.

11 Marco Pistoia, Satish Chandra, Stephen J Fink, and Eran Yahav. A survey of static analysis methods for identifying security vulnerabilities in software systems. *IBM Systems Journal*, 46(2):265–288, 2007.

12 R. W. Shirey. Internet security glossary, version 2. *RFC*, 4949:1–365, 2007.

13 Robert W. Shirey. Internet security glossary, version 2. *RFC*, 4949:1–365, 2007. `doi:10.17487/RFC4949`.

14 Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. *arXiv preprint*, 2012. `arXiv:1206.2944`.

NetLangEd, A Web Editor to Support Online Comment Annotation

Rui Rodrigues
Departamento de Informática, University of Minho, Braga, Portugal

Cristiana Araújo
Centro Algoritmi, Departamento de Informática, University of Minho, Braga, Portugal

Pedro Rangel Henriques
Centro Algoritmi, Departamento de Informática, University of Minho, Braga, Portugal

Abstract

This paper focuses on the scientific areas of Digital Humanities, Social Networks and Inappropriate Social Discourse. The main objective of this research project is the development of an editor that allows researchers in the human and social sciences or psychologists to add their reflections or ideas out coming from reading and analyzing posts and comments of an online corpus [1]. In the present context, the editor is being integrated with the analysis tools available in the NetLang platform. NetLangEd, in addition to allowing the three basic operations of adding, editing and removing annotations, will also offer mechanisms to manage, organize, view and locate annotations, all of which will be performed in an easy, fast and user-friendly way.

2012 ACM Subject Classification Human-centered computing → Interactive systems and tools; Software and its engineering → Application specific development environments

Keywords and phrases Online Annotation tool, Document Markup System, Text Editor, Discourse Analysis

Digital Object Identifier 10.4230/OASIcs.SLATE.2021.15

Funding This work has been supported by FCT – Fundação para a Ciência e Tecnologia within the R&D Units Project Scope: PTDC/LLT-LIN/29304/2017.

1 Introduction

This paper discusses a tool developed in the context of the project *NetLang* – The Language of Cyberbullying: Forms and Mechanisms of Online Prejudice and Discrimination in Annotated Comparable Corpora of Portuguese and English. In the last decades, we have witnessed an exponential growth of the Internet, more specifically in the way we communicate with each other, that brought many great things to our society from breaking distances between people to giving voice to those who didn't have one, allowing them to report the many injustices happening in the modern days. However it also empowered anti-social behaviors like online harassment, cyberbullying and hate speech. This type of communication its usually hostile and malicious, expressing discrimination, intimidation and disapproval towards certain characteristics like sex, race, religion, ethnicity, colour, national origin, disability, or sexual orientation of a person or a group of people. The objective of this kind of speech is to injure, harass and degrade the targeted person or group in order to dehumanize them [1, 2, 5, 9, 12, 13]. In order to solve this problem, many big companies made available several options to address this type of speech like flagging, reporting, counter-speaking or simply censor certain words that are commonly used in this type of speech. However these strategies do not always work,

[1] Consisting of texts extracted from various sources of Social Computer Mediated Communication (comment boards of news sites and social networks) which are then converted to a specific *JSON* format.

© Rui Rodrigues, Cristiana Araújo, and Pedro Rangel Henriques;
licensed under Creative Commons License CC-BY 4.0

generating many cases of outrage like the case of 2013 where several pages were found in Facebook with hateful content towards women like *Violently raping your friend just for laughs* and *Kicking your girlfriend in the fanny because she won't make you a sandwich.* In a matter of weeks a petition was created that aggregated 200,000 signatures and many important companies either removed their ads or threatened removing them from Facebook [10]. In that context, the platform *NetLang* is being developed to support researchers in the human and social sciences or psychologists collecting and making available for exploration a corpus of posts and comments that express this kind of hateful language. To improve the usability of the platform it was decided to develop a tool, like an editor, that would allow the researchers to write their thoughts on specific parts of the comments creating annotations and saving them so they can continue the analysis later. Our initial survey found several tools that offer this type of functionality like *Word*[2], *Adobe Reader*[3], *doccano*[4], *Hypothes.is*[5] and *Genius Web Annotator*[6], differentiating among them in the way the note is created, the method used to save/display them, the text formatting options available, etc.

After studying the problem through the exploration of other works and solutions related to the topic, two objectives were established for the work here reported: to develop an editor that allows users to annotate the text that they are analyzing; and finally to carry out usability, conformance, and performance tests with the developed editor and real end users. As final result, it is expected to have an editor that has a simple and easy to use interface. This project will be very useful for its researchers since it will allow them to have the content under study and their notes located on the same platform, ensuring better organization and accessibility.

The paper is organized as follows. Section 2 will cover topics such as the history of annotations, how people annotate on paper and on the web, annotation functionality and types, advantages and disadvantages of digital annotations and an analysis of the existing solutions. In Section 3 the system requirements will be listed and the system architecture will also be discussed and sketched using block diagrams. Section 4 describes how the editor was developed, showing some screenshots of the final result. Finally, in Section 5 an analysis of the results obtained will be made and conclusions will also be drawn regarding the work done and the future work.

2 Document Annotation, state-of-the-art

In this section the state of the art regarding annotation will be portrayed. Its initial part will be used to describe the evolution of the annotations over the years. Then, studies conducted by several researchers will be analyzed in order to understand the behavior of readers when annotating on paper and online. Based on these studies and other articles, the purposes and types of annotations will be described in general followed by the listing of the advantages and disadvantages of digital annotation compared to physical annotation. After that, some tools that allow the annotation of digital texts will be listed and compared taking into consideration some dimensions that are considered to be important for the development of *NetLangEd*.

[2] Available at `https://www.groovypost.com/howto/annotate-in-word/`, accessed in December 2020
[3] Available at `https://helpx.adobe.com/acrobat/using/commenting-pdfs.html`, accessed in December 2020
[4] Available at `https://doccano.github.io/doccano/`, accessed in December 2020
[5] Available at `https://web.hypothes.is/help/annotation-basics/`, accessed in December 2020
[6] Available at `https://genius.com/web-annotator`, accessed in December 2020

History of annotations

The usage of text annotations became a prominent activity around 1000 AD in Talmudic commentaries and Arabic rhetorics treaties[7]. It was then used in the Medieval ages to discuss, critique and learn from annotations created from previous readers who also read from the same manuscript. There were also situations where at the time the manuscripts were being copied, their annotations were included in the copy [14]. However, the emergence of the printing press has made this use of annotations obsolete due to having facilitated the circulation of information and the ability to purchase individual copies of text [14]. Nowadays annotation is an activity that is mostly done in private corresponding to the reader's interaction with the text being read. Computer-based technologies also provide many solutions for both individual and shared annotations, allowing to apply this method to online and offline digital documents [14].

How people annotate on paper and on the web

In order to understand the behavior of readers when annotating on paper and on the web, the results described in the papers [11, 7, 3, 8] were studied, obtaining several conclusions for both cases.

In the case of paper annotation, it was verified that highlights were the most used type of annotation and the most common purpose was to remember, thus being able to relate these two since highlights are normally used to help in the memorization process and to make it easier to find the important parts of the text in a later reading. Another reason for this predominant use of highlights is because they allow the reader to stay focused on the task of reading since it is a method of quick execution compared to the other ones. Still regarding the highlights, another important characteristic of these are their colors, which may have additional meanings and can facilitate detection. In one of the analyzed experiments, it was also found that the purpose of reading greatly influences the way in which readers annotate their documents, namely in the types of annotations they use. A final observation is that the annotations with text that are shared are written in a more explicit way so that other readers understand them more easily.

Regarding annotation on the web, it is difficult to determine which types of annotations are most used since it depends on the features that the systems provide. That being said, in all cases, readers were careful to place the notes as close as possible to the respective parts of the document to which they referred. Another interesting observation is that, in cases where it was possible to highlight, this tended to be the most common choice, confirming the popularity of this type of annotation. In this context, it was discovered that they had the purpose of signaling parts of the text that were not understood, that they wanted to remember or because they were important. As in the paper case, here the colors of the highlights are also of great importance, allowing readers to better structure their annotations. Regarding the notes that were made with the purpose of being shared, it was also possible to verify the same situation that was described in the paper case. These are more developed and explicit than the private notes, which are more short and ambiguous. Through this, it is possible to conclude that the readers when writing private notes are only concerned with their significance to themselves while in the case of shared notes they are written so that the other readers have no problems in understanding them.

[7] Available at `https://en.wikipedia.org/wiki/Text_annotation`, accessed in December 2020

Functionality of annotations

There are several benefits that are obtained through private and public annotations. Taking into account how people annotate on paper and on the web, as explained in the previous section, and based on the work done by Jindia and Chawla [6] it was possible to identify the following ones: facilitate the current or future reading process; facilitate the later writing process; understand the insights of another reader; provide feedback to the text writer or other readers; help with the memorization and recall process; draw attention to certain parts of the text that are considered to be important for future reference or reading and correct a specific part of the text.

Types of annotations

Annotations can take different forms where some of them assume a textual representation and others consist of graphic effects. Thus, through the same sources of knowledge used in the previous subsection, the following types of annotation were identified:

- **Mark:** The method of marking an important word/phrase through visual effects. These can be highlights, underlines, strikeouts, figures, etc.
- **Paraphrase:** It consists of reproducing in a simpler and more accessible way the central ideas of the original text, without changing its meaning.
- **Comment:** This type is based on the formulation of comments to specific parts of the text that are directed to the writer or self-directed. These may be of agreement/disagreement, questions, responses, connection to ideas from other texts, personal experience, adding explanation, etc.

Note that both the *Paraphrase* and *Comment* type can be combined with the *Mark* type in order to be able to contextualize them in the text.

Advantages and Disadvantages of Digital Annotations

In this subsection we will discuss the advantages and disadvantages of using digital annotations in comparison to paper-based annotations, since one of the main reasons that led to the development of this work was the fact that the users of the platform *NetLang* were using paper to write their comments/analysis of the text being read. To point out pros and cons of digital annotation, the considerations previously mentioned and the work of Glover, Xu and Hardaker [4] were taken into account.

The **advantages** that have been identified are as follows: better organization due to not having the notes spread over several sheets; adding annotations wont damage the original text; easier to change the content of annotations; allows the removal of annotations without leaving marks in the document; it allows you to write notes with more extensive content without having to worry about the space they will occupy and easier to locate annotations on the document through mechanisms.

The **disadvantages** identified were the following ones: it is not possible to directly manipulate the document requiring to follow a specific process that is more complex than simply using the pencil to write and draw on the paper; the forms that digital annotations can take are limited to those offered by the tool used; although a simple click of a button is enough to share digital annotations with other users; the circulation of a physical document will always be the most intuitive way to carry out this process and requires the user to learn how to use the tool.

Existing Tools

To conclude this section, the documentation of several tools that have annotation mechanisms was analyzed. Of the several that were found, the most popular or that have the most interesting characteristics are the following ones: *doccano*[8], *Word*[9], *Adobe Reader*[10], *Weava*[11], *LINER*[12], *Diigo*[13], *Hypothes.is*[14] and *Kami*[15]. The *A.nnotate*[16] tool was also considered, but since it is not as popular as the tools mentioned before it will not be covered here as well as the tools *Inception*[17], *Prodigy*[18], *UBIAI*[19] and *LabelStudio*[20] since they are very similar to the *doccano* tool. In order to be able to compare them, some points that were considered to be important for the editor were stipulated, being the following: *Annotation of web pages*, *Free*, *Multiple forms of annotation*, *Text formatting*, *Search facilities* and *Export and import*. After defining everything, the tools were compared in Table 1.

Table 1 Tool comparison.

Tool	Annotation of web pages	Free	Multiple forms of annotation	Text formatting	Search facilities	Export and import
doccano	No	Yes	Limited	No	No	Yes
Word	No	No	No	Yes	Limited	No
Adobe Reader	No	Yes	Yes	Limited	Yes	Yes
Weava	Yes	Limited	No	No	Yes	No
LINER	Yes	Limited	No	No	Yes	No
Diigo	Yes	Limited	Limited	No	Yes	No
Hypothes.is	Yes	Yes	No	Yes	Yes	No
Kami	No	Limited	Yes	Yes	Limited	No

Through Table 1 it is possible to conclude that, although there is no perfect tool, the most complete is the *Adobe Reader* since it is the one that satisfies most of the requirements in a satisfactory way. However, it does not fulfill the most important requirement for this work, this being the ability to annotate web pages. Thus, if we look at the tools that fulfill this requirement, the best one would be *Hypothes.is* due to fulfilling the greatest number of the remaining requirements. Another factor that should be mentioned is that this tool is the only one of the four web based tools that have been analyzed here that does not restrict some of its functions through free and paid accounts. The only negative aspects of this tool is that it only allows the use of highlights, to which it is possible to associate comments, as a form of annotation and doesn't have any mechanism that allows to export the work done so that it can be imported later to continue it.

[8] Available at `https://github.com/doccano/doccano`, accessed in April 2021
[9] Available at `https://www.microsoft.com/en-us/microsoft-365/word`, accessed in April 2021
[10] Available at `https://get.adobe.com/br/reader/`, accessed in April 2021
[11] Available at `https://www.weavatools.com/`, accessed in April 2021
[12] Available at `https://getliner.com/`, accessed in April 2021
[13] Available at `https://www.diigo.com/`, accessed in April 2021
[14] Available at `https://web.hypothes.is/`, accessed in April 2021
[15] Available at `https://www.kamiapp.com/`, accessed in June 2021
[16] Available at `http://a.nnotate.com/`, accessed in June 2021
[17] Available at `https://inception-project.github.io/`, accessed in June 2021
[18] Available at `https://prodi.gy/demo`, accessed in June 2021
[19] Available at `https://ubiai.tools/`, accessed in June 2021
[20] Available at `https://labelstud.io/`, accessed in June 2021

3 NetLangEd, the proposed architecture

In this section, the work proposal will be described in detail starting with a list that contains the features that the editor to be developed must have followed by the diagrams of the system architecture. Based on the information collected during the literature review phase to understand the *State of the Art*, discussed in Section 2, the features that the editor should present will now be defined.

The editor will have the following **Functional Properties**:

- **Marks on the text:** The part of the text that the user notes should be marked in some way.
- **Highlight color options:** The user must be able to customize the color of the highlight.
- **Tooltips in annotated text:** When hovering the cursor over the annotated text, a tooltip containing at least the respective comment must be presented.
- **Text formatting:** The user must be able to format the comment text in different ways, such as changing the font size, changing the font family, creating lists, etc.
- **Annotated comments list:** The user must be able to see all comments on the annotations made in the document he is analyzing.
- **Removal:** There should be options that allow the removal of annotations and their comments.
- **Editing:** As with removals, there should also be options that allow editing annotation comments.
- **Location discovery:** There must be a mechanism that allows a quick location of the annotation of the respective comment in the document.
- **Search:** The user should be able to search a word/phrase in the comments of the annotations.
- **Filter:** The user must be able to filter comments according to various criteria.
- **Import:** The user must be able to import from his computer the annotations and their respective comments that were previously made in the document that is currently being analyzed.
- **Export:** The user must be able to export to his computer the annotations and their respective comments that were made in the document.
- **Clear the document:** The user must be able to remove all annotations in the document at once.

The **non-functional properties** that the editor will possess are as follows:

- **Simple and clear interface:** The interface should not take up too much space so as not to distract the user and its content must be simple and explicit so that the user does not feel confused when using it.
- **Simple and clear functionalities:** The functionalities must be easy to understand and to execute so that the user does not have difficulties in using the tool.
- **Quick add functionality:** This functionality should be quick to perform, being done in the smallest number of steps possible.
- **Quick edit functionality:** Editing annotation comments should be possible both in the annotation comments list and in the document, thus allowing the user to remove them in any situation. This functionality should also be quick to perform, being done in the smallest number of steps possible.
- **Quick removal functionality:** As in the case of editing, the option to remove comments from annotations should be possible to execute both in the comment list of annotations and in the document, thus allowing the user to remove them in any situation. This functionality should also be quick to perform, being done in the smallest number of steps possible.

- **Annotation overlay:** The parts of the text where annotations overlap should be properly treated so that the annotations involved can be easily distinguished.
- **Annotation comment representation:** An annotation comment must be presented in the same way both in the annotation comment list and in the document's tooltips.

After the identification of the functional and non-functional requirements for the comments annotation editor under discussion, its architecture is sketched.

3.1 System Architecture

In this subsection, several diagrams will be presented that will allow to better understand the architecture of the system. Thus, Figure 1 presents the architecture of *NetLangEd* and its integration with *NetLang* platform.

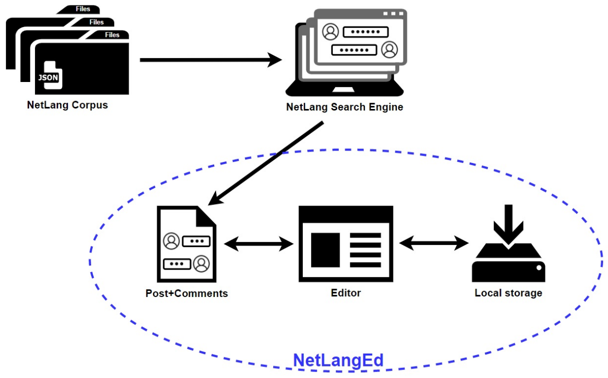


Figure 1 *NetLangEd* Architecture and Integration with *NetLang* platform.

Observing Figure 1 it is possible to see that the editor will be accessed from the pages that contain the posts and comments that are stored in the *NetLang* repository. Another aspect that can be seen in Figure 1 is the possibility of exporting and importing the work done, these being done to and from the user's computer respectively. Finally, Figure 2 presents how the functionalities are organized in the system and how they can be accessed.

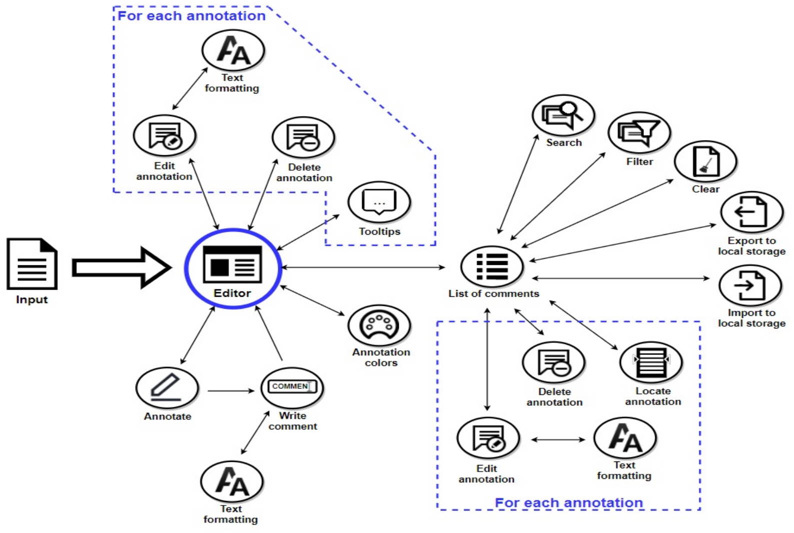


Figure 2 *NetLangEd* usage diagram.

As can be noticed in Figure 2, some of them (add, edit and remove annotations, choose the color of the annotations and tooltips) can be executed directly on the text; the remaining are executed on the menu.

4 NetLangEd, development

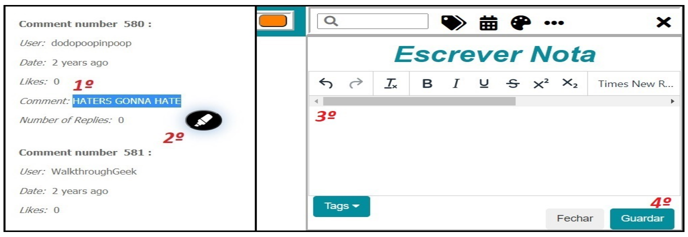
In this section it will be described how each of the requirements that were listed in Section 3 has been achieved and how the integration process with the *NetLang* platform was carried out.

4.1 Functional properties

The functional requirements that were implemented will be discussed in this subsection.

Marks on the text

This requirement, that consists of allowing users to annotate HTML text without affecting its original formatting, was the most important objective to be achieved in this project. Several solutions were tested, with the vast majority of them failing with some tags such as lists, links, etc or simply making it difficult to add, edit and remove annotations. Taking these problems into account, the only solution that avoided most of them was to surround each word, with the exception of HTML tags, with a *span* tag, after which it was decided to surround each character in order to be able to annotate any part of the words instead of being forced to annotate it entirely. These *span* tags have a unique identifier in order to be able to change their properties more easily, and for that it was necessary to develop a function that returned the identifiers of the *spans* that are present in the selection made by the user. It should be noted that this approach is not without problems, being dependent on the text to be annotated to remain unchanged so that the identifiers remain constant and other problems that arose during the integration phase with the *NetLang* platform that will be discussed in Subsection 4.3. Having said that, the adding process begins with the user selecting the part of the text that he wants to annotate followed by clicking on the pop-up that appears after completing the selection, in order to confirm the intention to annotate the selected part of the text. After that, the addition box will appear in the sidebar in which the user can enter the comment, with the possibility of associating tags to it, completing the process using the button that allows to save the comment. This whole process can be seen in Figure 3.


Comment number 580 :
User: dodopoopinpoop
Date: 2 years ago
Likes: 0
Comment: HATERS GONNA HATE
Number of Replies: 0

Comment number 581 :
User: WalkthroughGeek
Date: 2 years ago
Likes: 0

Figure 3 Text annotation functionality.

Highlight color options

This feature was implemented as a palette with the color options that are available (see Figure 4). Initially it was possible to pick any color, however this method would complicate not only the process of filtering the annotations by their color because it would lead to too many options but it would also complicate the process of picking a color since the user would have to either find the desired color or would have to save the color he picked for later use.

Tooltips in annotated text

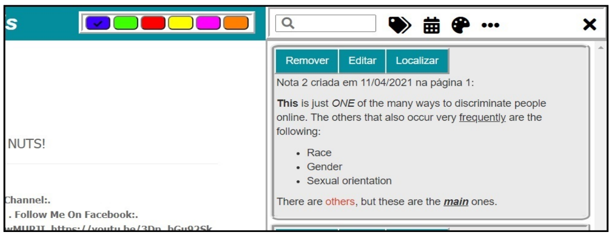
This feature can be seen whenever the user places the mouse cursor on top of an annotation presenting the result visible in Figure 8b. In addition to containing the comment that is associated with the annotation, the number of the note is also shown to make it possible to identify them more easily. However, there are some restrictions regarding the display of comments in the tooltips. In order to make sure that the tooltip is always visible, it can only be shown up to six lines of the comment. In cases where there is more content to show, an ellipsis is shown.

Text formatting

Concerning text formatting, there are many options that can be included, as *undo*, *redo*, *clear formatting*, *bold*, *italic*, *underline*, *strikethrough*, *superscript*, *subscript*, *fonts*, *font sizes*, *numbered list*, *bullet list*, *text color*, *background color*. From a technical point of view all of them can be included. However only fifteen (as can be seen in Figure 3) were implemented because the others were difficult to display in the tooltips.

Annotated comments list

As can be seen in Figure 4, all the annotations that were made on the document that is currently being analysed will be listed on a sidebar. Another important aspect of this sidebar was its simplicity, clarity and convenience. To this end, it was decided to keep at the top of it all the options that are used on a recurring basis separated, like the filters that will be talked about later, and group in one option all the options that wont be use as regularly, like exporting, importing, etc. Another decision that was made was to place all the options that can be made on the comments that are present in the sidebar on top of each one of them.



Nota 2 criada em 11/04/2021 na página 1:

This is just *ONE* of the many ways to discriminate people online. The others that also occur very frequently are the following:

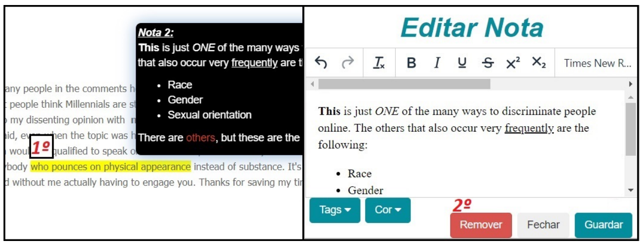
- Race
- Gender
- Sexual orientation

There are others, but these are the ***main*** ones.

Figure 4 List of comments in the sidebar.

Removal

This requirement has been implemented and can be executed in two ways, where one is performed by clicking on the desired annotation, that is present in the document, showing the edition box that has the button that allows its removal and the other way is through the removal button that is located over the annotations present in the sidebar. Both alternatives can be seen in Figures 5a and 5b.


Nota 2:
This is just *ONE* of the many ways that also occur very frequently are t

- Race
- Gender
- Sexual orientation

There are others, but these are the

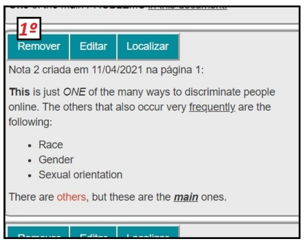
any people in the comments h
people think Millennials are s
my dissenting opinion with
id, even when the topic was
would 1º ualified to speak o
body who pounces on physical appearance instead of substance. It's
d without me actually having to engage you. Thanks for saving my ti

Editar Nota

This is just *ONE* of the many ways to discriminate people online. The others that also occur very frequently are the following:

- Race
- Gender

(a) From the document.



Nota 2 criada em 11/04/2021 na página 1:

This is just *ONE* of the many ways to discriminate people online. The others that also occur very frequently are the following:

- Race
- Gender
- Sexual orientation

There are others, but these are the ***main*** ones.

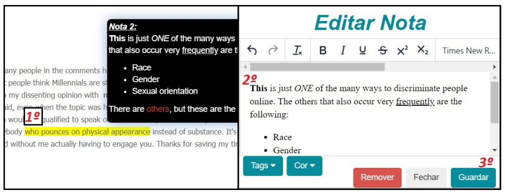
(b) From the sidebar.

Figure 5 Annotation removal functionality.

Note that, in both ways, the text box will be precariously filled with the current comment of the respective annotation so that the user can make the decision to remove the annotation more clearly.

Editing

As in the previous *functional property*, the editing process can also be done in two ways. The first consists of clicking on the annotation, present in the document, whose comment is to be edited and the second way is through the button that allows editing that is located on top of each of the annotations present in the sidebar. In both cases the result of the clicks is the display of the edit box where the user can edit the comment, as well as the tags and color associated with it, concluding the process by clicking in the button to save the comment. Both ways can be seen in Figures 6a and 6b.


Nota 2:
This is just *ONE* of the many ways that also occur very frequently are t

- Race
- Gender
- Sexual orientation

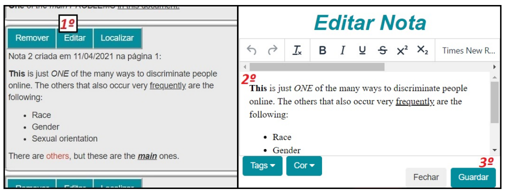
There are others, but these are the

Editar Nota

This is just *ONE* of the many ways to discriminate people online. The others that also occur very frequently are the following:

- Race
- Gender

(a) From the document.


Nota 2 criada em 11/04/2021 na página 1:

This is just *ONE* of the many ways to discriminate people online. The others that also occur very frequently are the following:

- Race
- Gender
- Sexual orientation

There are others, but these are the ***main*** ones.

Editar Nota

This is just *ONE* of the many ways to discriminate people online. The others that also occur very frequently are the following:

- Race
- Gender

(b) From the sidebar.

Figure 6 Annotation editing functionality.

It should be noted that, in this case, the text box will also be precariously filled with the current comment of the respective annotation so that the user does not have to rewrite the entire text if he only wants to make some changes to it.

Location discovery

The purpose of this requirement was to help the user to easily locate the comment in the document, being implemented through the button at the top of each of the comments present in the sidebar visible in Figure 4. Initially, this feature only scrolled the page to the place where the annotation was located, making it the only visible annotation in order to be more easily detectable. However, after making the decision to display the documents in pages, due to the problems that were detected in the process of integrating the work with the *NetLang* platform and which will be discussed later, this feature now has the ability to switch to the page where the annotation is located in case it was not done on the page that is currently open.

Search

This requirement is present at the top of the sidebar that can be seen in Figure 4. It is important to mention that only the annotations whose comments contain exactly what is entered by the user will be filtered, that is, the search takes into account the small and capital letters, blank spaces, etc. It is also important to note that the result of applying the filter is visible both in the document as in the sidebar.

Filter

This requirement was implemented through the three buttons that are to the right of the search bar present at the top of the sidebar visible in Figure 4, having buttons that allow filtering the annotations by tags, date and color. In addition to these three types of filters, others were discussed, however these seemed the most useful. Note that these three filters can be used together, including with the search functionality. That said, it was necessary to make another decision on how these filters will act together, that is, if it is enough that the annotations comply with one of the filters or if they have to comply with all of them. After some reflection and taking into account that the purpose of filtering is to specify as much as possible a characteristic of something, it was decided that the annotations have to comply with all the filters to be shown. Another characteristic that is worth mentioning is that, as in the previous requirement, the result of applying the filters is visible both in the document as in the sidebar. Regarding the management of the tags, this will be done through the button that allows filtering by tags, through which it will be possible to open the pop-up visible in Figure 7 where the user will be able to create, edit and remove tags.

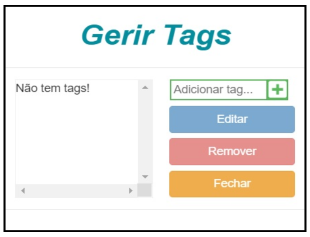


Figure 7 Tags manager.

Import

This requirement was implemented as a feature that is present in the dropdown that results from clicking on the button that is represented by the three dots visible in Figure 4. While the basic functionality of this requirement was achieved, it was attempted to apply a security mechanism to prevent the user from being able to write malicious HTML code in the part of the comments that would later be run when loading the file. Although it works and prevents the import of a file that has been changed, the user can access the “keyword” and replicate the hash process in order to overcome this barrier. However, it can be said that at least this process cannot be carried out in such an easy way.

Export

This functionality also appears in the drop-down that results from clicking on the button that is represented by the three dots visible in Figure 4. Export option writes in a text file the necessary information so that the user can restart the work where it was left in a next section. Note that the user must click on the *save button* to save the current status of the work done, before exporting it. Otherwise, the work will be lost.

Clear the document

This feature is also present in the drop-down that results from clicking on the button that is represented by the three dots visible in Figure 4. This functionality simply removes all annotations made in the document as well as in the sidebar.

4.2 Non-functional properties

In this subsection, it will be explained how the non-functional requirements were implemented.

Simple and clear interface

This requirement was achieved through several variables that were taken into account in the development of the interface. The first one is related to the way in which the buttons were placed at the top of the sidebar, which were explicitly placed those that are more likely to be used more frequently and those that are used less frequently were grouped in the option characterized by the three dots, as can be seen in Figure 4. Another important aspect is related to the comments of the annotations present in the sidebar. These were embedded inside compartments to better distinguish where a comment starts and ends. Another important decision was to place the operations that can be executed on these comments directly on them in order to be clear to the user to know which comment is applying the operations on. All these decisions can also be seen in Figure 4. The placement of the color palette was also subject of reflection. It was initially planed to be placed in the sidebar, however since it is a tool that the user may want to use at any time, it was decided to keep it fixed next to the sidebar as shown in Figure 4. Regarding the boxes that appear to edit or add a comment, it initially appeared in the center of the screen with the capability to be moved to any other part of the screen. However, since this task of always moving the box is boring and as it is preferable to always be able to see the part of the text that is being commented for contextualization, it was decided that in both cases these boxes would be displayed in the sidebar, as can be seen through Figure 3. Finally, the last point that was considered was the sidebar itself. It was decided to allow it to be hidden or expanded since when it was open it could distract the user. Another detail is related to the fact that it pushes the text that is being annotated to the side instead of overlapping it, thus not covering the text, allowing the user to continue to annotate even with it expanded. This decision can be seen in Figure 4.

Simple and clear functionalities

Most features have a word on their button that clearly describes their purpose, as for example, in the case of Figure 4. There are some exceptions, such as the functionalities at the top of the sidebar that have icons as shown in Figure 4. However, care was taken to use icons that represent the respective functionalities. Thus, only the operations that are executed from the document, these being the tooltips, removal and editing, are not so obvious to execute. However, in this case it was decided to sacrifice this aspect a little in order to benefit other requirements that will still be discussed here.

Quick add functionality

This functionality was implemented being only necessary to perform four actions (select text, click on pop-up, write comment and save comment) as can be seen in Figure 3. There is a way to reduce the number of necessary steps to three, which consisted in removing the pop-up part and display the addition box after completing the selection, however this method could create complications in cases where the user accidentally selects something that he didn't want to annotate. Note that this process may require a greater number of steps if the user wants to associate tags to the annotation, however it is not a mandatory step in the process.

Quick edit functionality

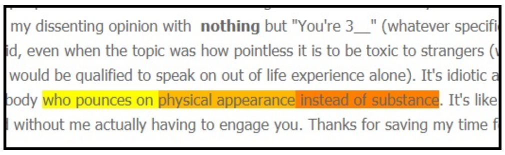
To fulfill this requirement, it was important to make sure that the editing functionality could be performed both in the document and in the sidebar. In this case, both situations only require three actions (in the document it is necessary to click on the annotation, edit and save changes and in the case of the menu it will be necessary to click on the edit, edit and save changes button) to complete the process, both of which are visible in Figures 6a and 6b, without having found any way to reduce this number. Note that, as in the case of addition, this process may require a greater number of steps if the user wants to edit the tags or the color that are associated with the annotation, however these are not mandatory steps in the process.

Quick removal functionality

This requirement, just like in the previous one, was achieved by not only allowing the user to use the removal functionality on the document and on the sidebar but also execute it with the shortest number of clicks. Regarding this last aspect, in the document its only required two steps (click on the annotation and click on the remove button) and on the list of comments its required only one (click the remove button), both of this cases being visible in Figures 5a and 5b. In the case of executing this functionality in the document, it is possible to be done in just one click by right-clicking on the annotation to be removed, however this method could lead to situations where the user could unintentionally remove an annotation.

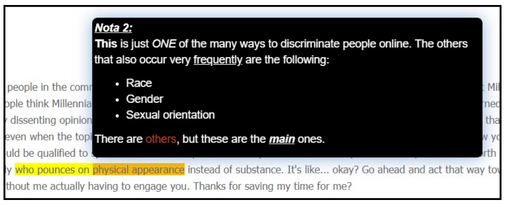
Annotation overlay

This feature was implemented using a color effect, as can be seen in Figure 8a. Observe that the color effect is present in the part of the text in which the overlap has occurred, which will have a color that will be the result of combining the color of the last annotation made on that part of the text with the color of the new annotation also made on that part of the text. When there are a greater number of overlaps, placing the cursor over an annotation it will only show that specific annotation, as visible in Figure 8b, returning to normal as soon as the mouse is moved out of it.



my dissenting opinion with **nothing** but "You're 3__" (whatever specifi
id, even when the topic was how pointless it is to be toxic to strangers (
would be qualified to speak on out of life experience alone). It's idiotic a
body who pounces on physical appearance instead of substance. It's like
without me actually having to engage you. Thanks for saving my time f

(a) Without effect.



Nota 2:
This is just *ONE* of the many ways to discriminate people online. The others that also occur very frequently are the following:

- Race
- Gender
- Sexual orientation

There are others, but these are the ***main*** ones.

people in the comn
ople think Millennia
dissenting opinion
even when the topi
uld be qualified to
y who pounces on physical appearance instead of substance. It's like... okay? Go ahead and act that way to
thout me actually having to engage you. Thanks for saving my time for me?

(b) With effect.

Figure 8 Example of an annotation overlay.

Annotation comment representation

This feature can be seen in Figures 8b and 4, where it is possible to see that the annotation content and its formatting are displayed in the same way in both cases.

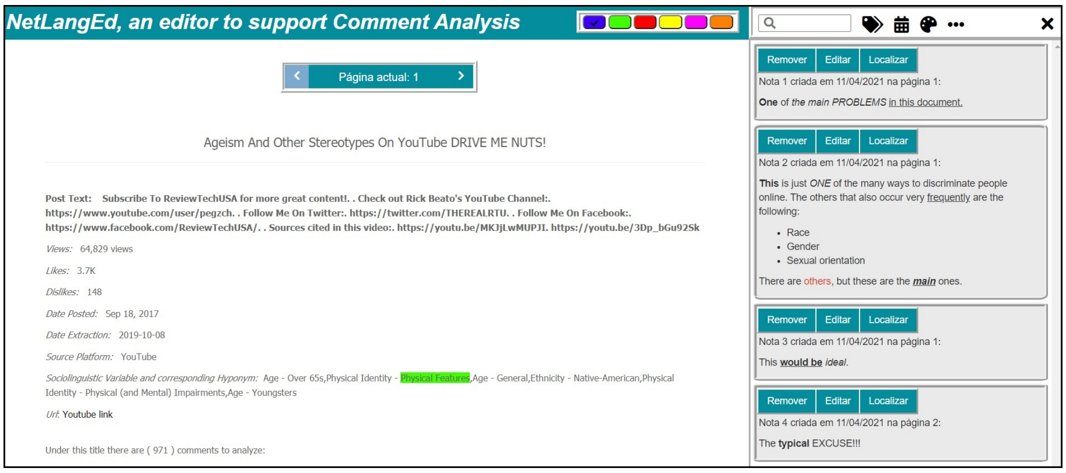
4.3 Integration with the NetLang project

To wrap up this section, where the tool development was discussed, the problems that were encountered during the integration process with the *NetLang* existing platform will be identified and the solutions found will be also presented. The first problem was related to the fact that there are very long texts on which it would be applied the process of placing *span* tags in all of its characters, causing the browser to jam and not load the page. The solution to this problem was to load part of the document instead of loading it all at once and to load more whenever the user pressed a button. Although the previous solution eliminated the problem of not being able to load the page, it did not avoid the second problem that occurred when the page size started to become very large, causing the editor to function slowly. In view of this, the final solution was to load the document as pages that would not exceed a maximum limit, where the user could navigate backwards or forwards one page at a time or even directly open a specific page.

After overcoming all this problem, the planned tool is operational and can be accessed at `http://bit.ly/NetLangEd`. Figure 9 is a screenshot that illustrates a working session. The image shown exhibits the text window on the left with an annotation in green and with the page browser at the top, the menu on the right side with the filters and other options at the top and with all the comments of the annotations made in the document listed below it and with the color palette to the left of the menu.

5 Conclusion

Along this paper it was presented the evolution that occurred in the use of annotations, from a strict use for sharing knowledge to a primarily personal use. Through the analysis of the studies carried out by several authors it was possible to observe details of the readers' behavior when annotating on paper and online, through which were identified not only the types of annotations used but also their purposes. It was also possible to enumerate some advantages and disadvantages that online annotations have compared to paper annotations, allowing to conclude that there are two main factors that determine the preference of its use. The first is related to the original format of the document to be annotated where in the case of being paper it will be very unlikely that it will be digitized to be digitally annotated and the second factor is related to the functionalities that the annotation system provides where in the case of not covering the readers' main needs of annotation will lead them to prefer to print the document and annotate it on paper. In order to understand the characteristics that



Ageism And Other Stereotypes On YouTube DRIVE ME NUTS!

Post Text: **Subscribe To ReviewTechUSA for more great content!. . Check out Rick Beato's YouTube Channel:. https://www.youtube.com/user/pegzch. . Follow Me On Twitter:. https://twitter.com/THEREALRTU. . Follow Me On Facebook:. https://www.facebook.com/ReviewTechUSA/. . Sources cited in this video:. https://youtu.be/MKJjLwMUPJI. https://youtu.be/3Dp_bGu92Sk**

Views: 64,829 views

Likes: 3.7K

Dislikes: 148

Date Posted: Sep 18, 2017

Date Extraction: 2019-10-08

Source Platform: YouTube

Sociolinguistic Variable and corresponding Hyponym: Age - Over 65s,Physical Identity - Physical Features,Age - General,Ethnicity - Native-American,Physical Identity - Physical (and Mental) Impairments,Age - Youngsters

Url: Youtube link

Under this title there are (971) comments to analyze:

Nota 1 criada em 11/04/2021 na página 1:

One of *the main PROBLEMS* in this document.

Nota 2 criada em 11/04/2021 na página 1:

This is just *ONE* of the many ways to discriminate people online. The others that also occur very frequently are the following:

- Race
- Gender
- Sexual orientation

There are others, but these are the ***main*** ones.

Nota 3 criada em 11/04/2021 na página 1:

This **would be** *ideal*.

Nota 4 criada em 11/04/2021 na página 2:

The **typical** EXCUSE!!!

Figure 9 Example of a work session.

a web annotator should have, several existing solutions have been analyzed, being possible to observe some characteristics that can be considered essential and others that are either not very useful or simply do not fit the objectives that are intends to achieve with this work. When comparing these solutions it was possible to conclude that, although none was perfect, the best would be *Hypothes.is* since, of the tools that allow the annotation of web pages, it is the one that fulfills the greatest number of the remaining requirements. That being said, the analysis that was carried out in all of them served as a basis to create a list of requirements that the editor to develop must fulfill by which the diagrams of the system architecture were developed. After finishing the development of the editor, it is possible to affirm that all the requirements that were stipulated here were fulfilled. To the best of our knowledge, *NetLangEd* is the first annotation editor integrated with a corpus browser. However, there are some aspects that can be improved or even added in the future, such as the aesthetic component of the editor, allowing filters to act on the form of a conjunction and developing a more efficient way to make annotations so that it is possible to load to the document in its entirety instead of loading it in pages. Another future work will be to conduct tests with users in order to obtain feedback.

References

1 Raphael Cohen-Almagor. Countering hate on the internet. *Annual Review of Law and Ethics*, 2014.

2 Karmen Erjavec. "You don't understand, this is a new war!" Analysis of hate speech in news web sites' comments. *Mass Communication and Society*, 2012.

3 Nor Fariza, Hazita Azman, and Afendi Hamat. Investigating students' use of online annotation tool in an online reading environment. *3L: Language, Linguistics, Literature*, 19:87–101, 2013.

4 Ian Glover, Zhijie Xu, and Glenn Hardaker. Online annotation – research and practices. *Computers & Education*, 49(4):1308–1320, 2007.

5 David John Harvey. Dangerous speech – some legislative proposals. *SSRN Electronic Journal*, 2019.

6 Anjali Jindia and Sonal Chawla. Annotations in e-learning. *International Journal of Computers & Technology*, 12:3852–3859, 2013.

7 Ricardo Kawase, Eelco Herder, and Wolfgang Nejdl. A comparison of paper-based and online annotations in the workplace. In *Learning in the Synergy of Multiple Disciplines. EC-TEL 2009. Lecture Notes in Computer Science*, pages 240–253, 2009.

8 Jingyan Lu and Liping Deng. Examining students' use of online annotation tools in support of argumentative reading. *Australasian Journal of Educational Technology*, 2013.

9 Silvana Neshkovska and Zorica Trajkova. The essentials of hate speech. *International Journal of Education TEACHER*, 2017.

10 Chikashi Nobata, Joel Tetreault, Achint Thomas, Yashar Mehdad, and Yi Chang. Abusive language detection in online user content. *Proceedings of the 25th International Conference on World Wide Web – WWW '16*, 2016.

11 Ilia A. Ovsiannikov, Michael A. Arbib, and Thomas H. McNeill. Annotation technology. *International Journal of Human-Computer Studies*, 50(4):329–362, 1999.

12 Alexandra A. Siegel. *Social Media and Democracy: The State of the Field, Prospects for Reform*, pages 56–88. Cambridge University Press, 2020.

13 Stefanie Ullmann and Marcus Tomalin. Quarantining online hate speech: technical and ethical perspectives. *Ethics and Information Technology*, 2019.

14 Joanna Wolfe and CM Neuwirth. From the margins to the center - the future of annotation. *Journal of Business and Technical Communication*, 2001.

Intelligent Query Answering with Contextual Knowledge for Relational Databases

Dietmar Seipel
Department of Computer Science, Universität Würzburg, Germany

Daniel Weidner
Department of Computer Science, Universität Würzburg, Germany

Salvador Abreu
Nova–Lincs, University of Évora, Portugal

Abstract

We are proposing a *keyword–based query interface* for knowledge bases – including relational or deductive databases – based on contextual background knowledge such as suitable *join conditions* or synonyms. Join conditions could be extracted from existing referential integrity (foreign key) constaints of the database schema. They could also be learned from other, previous database queries, if the database schema does not contain foreign key constraints.

Given a textual representation – a word list – of a query to a relational database, one may parse the list into a structured term. The intelligent and cooperative part of our approach is to *hypothesize* the semantics of the word list and to find suitable *links* between the concepts mentioned in the query using contextual knowledge, more precisely *join conditions* between the database tables.

We use a knowledge–based parser based on an extension of Definite Clause Grammars (Dcg) that are interweaved with calls to the database schema to suitably *annotate* the tokens as table names, table attributes, attribute values or relationships linking tables. Our tool DdQl yields the possible queries in a special domain specific rule language that extends Datalog, from which the user can choose one.

2012 ACM Subject Classification Information systems → Data management systems

Keywords and phrases Knowledge Bases, Natural Language Interface, Logic Programming, Definite Clause Grammars, Referential Integrity Constraints

Digital Object Identifier 10.4230/OASIcs.SLATE.2021.16

Supplementary Material *Software (Source Code)*: `http://www1.pub.informatik.uni-wuerzburg.de/databases/research.html` [19]

1 Introduction

The growing wave of *digitization*, which the smart world of the future is facing, could be met by concepts from *artificial intelligence (AI)*. The field of AI can be divided into symbolic and subsymbolic approaches, e. g., [11,15]. *Symbolic or knowledge–based* approaches model central cognitive abilities of humans like *logic*, deduction and planning in computers – mathematically exact operations can be defined. *Subsymbolic or statistical* approaches try to learn a model of a process (e. g., an optimal action of a robot or the classification of sensor data) from the data.

Current knowledge–based information systems are increasingly becoming hybrid, including different formalisms for knowledge representation. In this paper, we use concepts from AI and logic programming for answering non–expert queries to hybrid knowledge bases. Still, the most frequent fromalism is relational databases, but it would be very interesting to include rule–bases, ontologies and Xml databases as well.

It is becoming popular to consider natural language queries [1]. In a simple form, this concept is well–known from *keyword–based queries* in search engines like Google. It can be very helpful for users who are not so familiar with the database schema, and for users on

© Dietmar Seipel, Daniel Weidner, and Salvador Abreu;
licensed under Creative Commons License CC-BY 4.0

mobile devices, where it is difficult to enter complex–structured queries. For a preceeding speech–to–text transformation, currently subsymbolic approches, e.g. voice/speech assitants such as the commercial systems Siri, Alexa, or Dragon NaturallySpeaking or the publicly available tools Mozilla Common Voice/Deep Spech [12] are popular. In this paper, the complicated step of assigning a suitable semantics – i.e. of compling textual keyword–based queries to correct complex–structured knowledge base queries, e.g. in SQL or Datalog– is done using a symbolic, declarative knowledge–based approach with techniques from logic programming and deductive databases [3, 6, 7].

The rest of this paper is structured as follows: Section 2 gives an overview on database query languages and intelligent query answering. Section 3 presents our running example of a database schema, a database instance (tables) and a set of relational database queries; we sketch some possible ways of deriving suitable join conditions. Section 4 describes our new system and langauge DDQL for answering keyword–based queries using technology from logic programming and deductive databases; we use our running example database. Finally, Section 5 concludes with a summary.

2 Database Query Languages and Intelligent Query Answering

Natural language interfaces (NLI) are considered a useful end–user facing query language for knowledge bases, see Affolter et al. [1] and Damljanovic et al. [8]. This holds especially true for complex databases and knowledge bases, where the intricacies of both the information schema and the technicalities of the query language – SQL most of the time – put the task of issuing useful queries well beyond the skill of most prospective, non–technical users. NLIs can usually be catgorized into keyword–, pattern–, parsing–, and grammar–based systems. Recent case studies are also reported by Stockinger [23] who argues that the trend of building NLI databases is stimulated by the recent success stories of artificial intelligence and in particular deep learning. An important keyword–based system is SODA [2]. Li and Jagadish [14] hold that NLIs are superior to other approaches to ease database querying, such as keyword search or visual query–building. They present the parsing–based systems NaLIR and NaLIX.

The main gripe with a natural language interface is that it's inherently difficult to verify reliably: an ambiguous sentence might be incorrectly parsed and its meaning evaluated, without the end user ever becoming aware of the situation. As a consequence, much effort has been placed into devising user–friendly ways of removing the ambiguity and translating the query to a semantically equivalent one in the native database query language. In practice, this entails presenting alternatives to the user and asking him to decide; the process may be iterated.

Doing so with SQL as the target seems a natural choice, but hits many difficulties arising from the language's many quirks. This situation is exacerbated when one must present the query interpretation back to the user. Relying on a more abstract query language, such as a first order predicate logic–based one, turns out to be both easier and more effective, especially as the reflection of the user's utterance interpretation will be presented in a form which is closer to its presumed grammatical structure and therefore easier to recognize and understand.

Besides convenience in presentation, relying on a logic representation for the queries and schema has several enabling benefits: a major one is that it provides a unifying framework for heterogeneous sources of information, such as SQL databases but also deductive databases, XML databases, ontologies queried in SPARQL or RDF datasets [22]. This topic has been amply covered in the literature, see for example [16] for an overview on logic in deductive databases.

Interpreting a natural language sentence as a database query entails attempting to do several queries, ranging over the schema but also the data and even the query history. Contextual speech recognition is a very hard problem, which can be eased if one manages to make use of background knowledge. The inherent ambiguity in the task of parsing and tagging a sentence in natural language can be mitigated and complemented with concurrent knowledge base queries: domain knowledge may be used to constrain the admissible interpretations as well as to provide useful annotations. Having a logic–based framework also makes it easy to provide views, which may be further used in interpreting natural language queries. The logic dialect needs not be full first–order logic, as Datalog is sufficient to express queries originally formulated in simple natural language.

3 Relational Database Queries

It is difficult for database users to have to remember the strucuture of the database (the database schema) and the correct writing of the terms (table names and attributes) and the values in the tables. Nevertheless, they have a good notion of the queries that they would like to ask. One could, e.g., imagine the following database queries:

$\mathcal{Q}_1$ *Give me the salary of Borg.*
$\mathcal{Q}_2$ *Who is the father of Alice* ?
$\mathcal{Q}_3$ *What is the salary of Research* ?
$\mathcal{Q}_4$ *Give me the sum of the salaries of the departments by name.*
$\mathcal{Q}_5$ *Give me the supervisor name of an employee by name.*

The database user does not say that *salary* is an attribute of a database table or that *Borg* is a value of another attribute. Moreover, there could be slight spellings mistakes.

We are proposing an intelligent expert tool for query answering based on the deductive database system DDbase [21] of the declarative programming toolkit Declare [19]. We have developed a module DdQl of DDbase, that can first parse the textual representation of the query using Declare's extended definite clause grammars (Edcg) [18] in Prolog based on the background knowledge of the database schema and the database, then hypothesize the intended semantics of the query using expert knowledge, and finally present possible queries and answers, so that the user can select one. In future extensions, it might be possible to define Datalog–like rules in natural language.

3.1 The Database Schema

We use the relational database Company from [9] in an extensive case study for exemplifying our approach. The database schema in Figure 1 contains 6 entity/relationship types and 8 referential integrity constraints between them (links given by arrows). Some corresponding database tables will be given in the following Section 3.2. The query compilation in Section 4.2 will extract undirected connected subgraphs from the corresponding link graph in Figure 2.

The relationship types from the corresponding ER diagram of [9] are represented in the database schema as follows:

(a) in the table Employee, the 1:n relationship types Works_For and Supervision from the ER diagram are integrated as foreign keys Dno and SuperSsn (the Ssn of the supersisor), respectively;
(b) the manager of a department is given by the attribute MgrSsn in Department;
(c) the table Works_On gives the employees working on a project, and the attribute Dnum in Project gives the responsible department of a project.

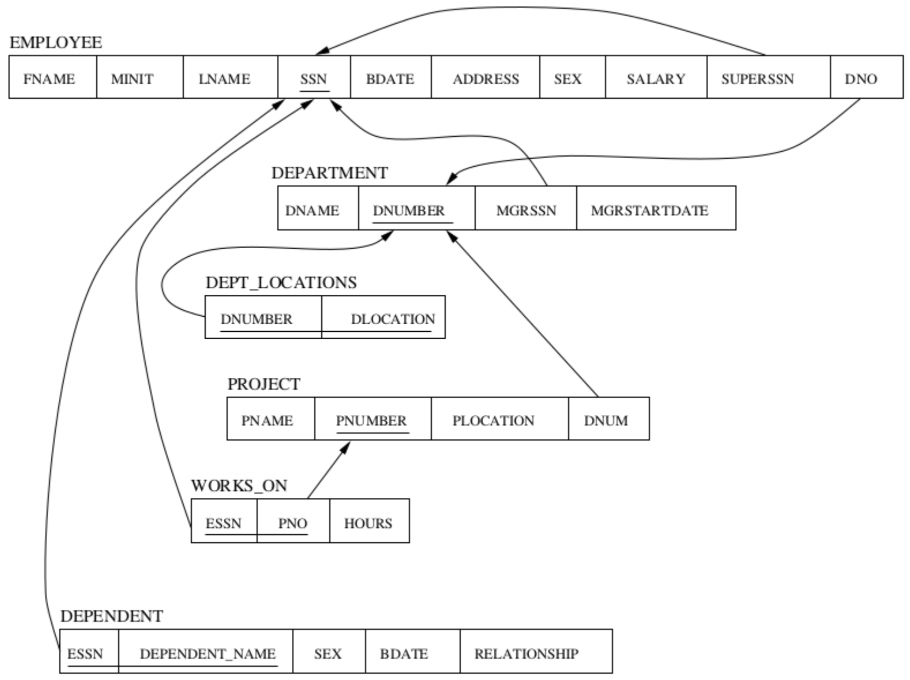


Figure 1 Referential Integrity Constraints for the Relational Database COMPANY.

Functionalities and existency constraints require: every employee works for exactly one department; every department must have exactly one manager; an employee can manage at most one department; every employee must work for at least one project; and every project must have exactly one responsible department. All constraints of the database schema can be used for optimizing queries.

3.2 Database Tables

In the following, we will use a slightly restricted version of the database, where some entity types and attributes are not present or renamed.

EMPLOYEE								
FNAME	LNAME	SSN	BDATE	ADDRESS	SEX	SALARY	SUPERSSN	DNO
John	Smith	4444	1955-01-09	731 Fondren, Houston	M	30000	2222	5
Franklin	Wong	2222	1945-12-08	638 Voss, Houston	M	40000	1111	5
Alicia	Zelaya	7777	1958-07-19	3321 Castle, Spring	F	25000	3333	4
Jennifer	Wallace	3333	1931-06-20	291 Berry, Bellaire	F	43000	1111	4
...	...	...	...	...	...	...	...	...
James	Borg	1111	1927-11-10	450 Stone, Houston	M	55000	NULL	1

The departments and their managers are given by the table DEPARTMENT with the primary key DNO (DNUMBER in Figure 1). The 1:1 relationship type MANAGES is integrated as a foreign key MGRSSN together with the describing attribute MGRSTARTDATE, the start date of its manager. The multi–valued attribute Locations of the entity type DEPARTMENT yields a separate table DEPT_LOCATIONS, which we do not consider here. The table WORKS_ON shows the HOURS that the employees work on the projects.

Department			
Dname	Dno	MgrSsn	MgrStartDate
Headquarters	1	1111	1971-06-19
Administration	4	3333	1985-01-01
Research	5	2222	1978-05-22

Works_On		
ESsn	Pno	Hours
1111	20	NULL
2222	2	10.0
2222	3	10.0
3333	30	20.0
...	...	...

The 1:n relationship type Controls between Department and Project is integrated as the foreign key Dnum in Project. Every project is located at one of the locations of its controlling department.

Project			
Pname	Pnumber	Plocation	Dnum
ProductX	1	Bellaire	5
Reorganization	20	Houston	1
...	...	...	...
Newbenefits	30	Stafford	4

In a deductive database variant, the rows of the relational tables would be represented by Datalog facts.

3.3 Datalog–Like Rules

In our system DdQl, the relational or deductive database can be enriched with further Datalog–like rules in field notation, or knowledge from ontologies in Rdf, Owl, or Swrl. Likewise, some background knowledge from the database schema can be represented in a Datalog–like manner. For the foreign–key constraint from Employee to Department, a Datalog–like rule with field notation can be generated in DDbase which links the social security number of an employee with the number of his or her department:

```
works_for(Employee, Department) :-
   employee:['SSN':Employee, 'DNO':Department].
```

From an ontology, it could be known that every employee is human. This would be expressed by the following Datalog–like rule with field notation, since Ssn is the primary key of Employee:

```
human(Employee) :-
   employee:['SSN':Employee].
```

The user of the database would like to ask queries in *Google style*, i.e. without precisely remembering the database schema and the links between the tables by referential integrity constraints. Here, the background knowledge given by the Datalog–like rules can be used.

3.4 Inference of Join Conditions

Suitable join conditions can be inferred from the foreign key constraints given in the database schema. The schema of the database `company` contained many foreign key constraints. For many other databases, no foreign key constraints are given. But join conditions can be inferred from previous SQL queries in the log file: e.g., a join condition can be assumed, if the primary key of a table (all attributes of the primary key) is joined with some attributes of another table.

In Declare, the schema of a table can be extracted automatically from a running relational MySQL database system and presented in XML to the user and analysed with Prolog:

```
<table name="employee">
   <attribute name="SSN" type="varchar(9)" is_nullable="NO"/>
   <attribute name="SALARY" ...>
   <attribute name="DNO" ...>
   ...
   <primary_key> <attribute name="SSN"/> </primary_key>
   <foreign_key>
      <attribute name="SUPERSSN"/>
      <references table="employee">
         <attribute name="SSN"/> </references> </foreign_key>
   <foreign_key>
      <attribute name="DNO"/>
      <references table="department">
         <attribute name="DNUMBER"/> </references> </foreign_key>
</table>
```

Currently, this XML representation is derived using ODBC in Declare, and join conditions are extracted in DDQL. If the second foreign key constraint was not given in the schema, we may still infer a corresponding join condition from the following SQL statement occuring in a query log file:

```
use company;
select employee.DNO, employee.SSN, employee.SALARY
from employee, department
where department.DNAME = 'Research'
and employee.DNO = department.DNUMBER;
```

Declare provides a tool named SQUASH [5] to parse SQL statements to Prolog terms, which may be mapped to XML. SQUASH proposes a domain specific language SQUASHML for SQL statements; this can be further processed in Declare to infer the join conditions. In a simplified version, the SQL statement above looks as follows:

```
<select>
   <select_list>
      <object table="employee" column="DNO"/>
      <object table="employee" column="SSN"/>
      <object table="employee" column="SALARY"/>
   </select_list_element>
   <from_list>
      <object table="employee"/>
      <object table="department"/>
   </from_list>
   <where_list>
      <condition junctor="and">
         <comparison operator="=">
            <object table="department" column="DNAME"/>
            <object value="Research"/>
         </comparison>
      </condition>
   </where_list>
</select>
```

4 The Declarative Database Query System and Language DDQL

In this section, we will present the new declarative database query system and language DdQl, which is based on concepts from logic programming. The knowledge-based compilation of keyword–queries to Datalog is done in three steps. In experiments with the `company` database, useful queries were generated; if a database does not contain referential integrity constraints, then we will need query logs for deriving suitable join conditions.

4.1 Syntax and Evaluation Using Logic Programming Concepts

The declarative programming toolkit Declare [19] and its deductive database system DDbase [21] already have functionality for evaluating database queries formulated using extensions of Datalog. Even hybrid queries including different knowledge representation formalisms are possible in DDbase. E.g., relational databases can be accessed using Sql queries and Odbc; for Xml processing, a query, transformation and update language FNQuery is given in [20].

We assume that the reader has some basic knowledge about logic programming [3, 7] as well as relational [9] and deductive databases [16]. DDbase allows for rules of the form A :- $L_1, \ldots, L_m$, where the head A is a logical atom and the body is a conjunction (the comma "," denoting the conjunction "$\wedge$") of literals L_i, $1 \leq i \leq m$. The literals can be logical atoms $L_i = B_i$, default negated literals $L_i = \mathit{not}\,(A_i)$, or aggregation literals $L_i =$ `ddbase_aggregate`$(X, (A_1, \ldots, A_n), Xs)$ over logical atoms A_i. In the domain specific language of DDbase, rules can have ordinary logical atoms – in Prolog notation – or *field notation* atoms $p : [a_1 : t_1, \ldots, a_k : t_k]$, where p is a predicate symbol, $a_1, \ldots, a_k$ are field names, and $t_1, \ldots, t_k$ are corresponding terms, which amounts to non–positional arguments in Prolog terms. The rules must fullfil the *safety condition* that variables in atoms A_i within default negated literals must be bound by preceeding ordinary atoms or aggregation literals in the same rule body. It is not the intent of this paper to formally define the semantics of DDbase. The next subsections will focus on the knowledge–based compilation of queries and give some intuitive examples without default negation. Only one ($\mathcal{Q}_4$) contains an aggregation literal

```
ddbase_aggregate( [C, sum(D)],
   ( employee(_, _, _, _, _, _, _, D, _, E),
     department(C, E, _, _) ), Xs ),
member([A, B], Xs).
```

In analogy to Sql and extending Prolog's predicate `findall/3`, this groups instantiations of the variable `C` with the sum (which is an aggregation function) of the corresponding instantiations of the variable `D` and returns pairs. Here, the template is `X = [C, sum(D)]` and pairs `[A, B]` are selected from the result `Xs`.

DDbase programs are *evaluated bottom–up* with *stratified fixpoint* computation like Datalog and they can – possibly – be compiled to Sql queries; often DDbase programs can also be evaluated top–down like in Prolog. For the evaluation in logic programming, the field notation atoms are compiled to ordinary, logical Datalog atoms based on background knowledge about the database schema. The ordinary Datalog rules can be compiled to Sql with DDbase, if there is no default negation – and for stratified default negation or aggregation. A stratified evaluation requires that none of the embedded atoms A_i is mutualle recursive with the head atom A. Then, the program is split into strata, such that default negated literals or aggregation literals refer to lower strata; the strata are evaluated successively, and the output of a lower stratum is fed into the strata above. For non–stratified default negation, DDbase could use *answer set* solvers, cf. [4, 13], if there are no aggregation literals.

4.2 Knowledge–Based Compilation of Queries

DdQl compiles a NL query $\mathcal{Q}_N$ to a Datalog query $\mathcal{Q}_D$ in three steps:

$\mathcal{Q}_N \to \mathcal{Q}_A \to \mathcal{Q}_F \to \mathcal{Q}_D$.

It shows them in a command line interface; the result tables are shown in a graphical interface.

Annotation of Key Words ($\mathcal{Q}_A$)

First, using Definite Clause Grammars (Dcgs, see, e.g., [3, 7]), an annotated query $\mathcal{Q}_A$ is generated. Using, e.g., the following grammar rule in the extended Dcg formalism introduced in [18], also the resulting parse tree can be computed:

```
query ==> aggregation, of, attribute, of, table.
```

The Dcg rules are fully interleaved with database access operations of DDbase using Odbc. E.g., the derivations of `attribute` and `table` can result in Odbc calls, or – for potential speed–up – calls to a cached collection of facts previously extracted from the database. Some words of the query – such as "`of`" and "`the`" – are ignored.

DdQl generates the annotations one after the other on *backtracking*, starting with the most likely annotations. E.g., the query $\mathcal{Q}_1$ with the key words "`salary, of, Borg`" is first annotated to the following query $\mathcal{Q}_A$:

```
salary=company/employee/attribute
'Borg'=company/employee/row(@LNAME)
```

The keyword "`of`" is ignored. In DdQl, it can be detected easily from the database schema that `salary` is an attribute of the relation `employee`. The location of `'Borg'` has to be done based on the contents of the database, which is more expensive. But this can be done depending on the context of the table `employee`; it turns out that it is the value of the attribute `LNAME`. After the first annotation has been done and the first solution to the query has been produced, DdQl uses backtracking to generate further annotations and solutions. Of course, then DdQl will also search for `'Borg'` in other tables of the database.

Generation of Field Notation ($\mathcal{Q}_F$) and Datalog ($\mathcal{Q}_D$)

Then, the compilation of the annotated queries $\mathcal{Q}_A$ to Sql or Datalog is done using technology from DDbase based on the database schema. As an intermediate representation, conjunctive queries $\mathcal{Q}_F$ in Datalog are generated with atoms in field notation. The conjunctive queries are then refined and optimized to ordinary Datalog queries $\mathcal{Q}_D$ using background knowledge from the database schema or Datalog–likes rules. $\mathcal{Q}_D$ could be evaluated on a deductive variant of the relational database or a deductive database; an Sql variant of $\mathcal{Q}_D$ can be evaluated on the relational database.

In the following, we will show and explain the intermediate queries $\mathcal{Q}_F$ and $\mathcal{Q}_D$ for a few example queries to the database `company` – and we skip the annotated queries $\mathcal{Q}_A$.

The relevant links between the concepts mentioned in a user query might be an undirected tree, or even a cyclic graph. E.g., if the user asks for the salary of all employees working in a department that controlls the project `'ProductX'` and is located in `'Houston'`, then the red link tree in Figure 2 (arrows starting with a bullet) – which is an abstraction of Figure 1 – might be used. The direction of the labelled referential integrity constraints Manages and Controls is due to the fact that their corresponding relationships are integrated as the attributes MgrSsn and Dnum into Department and Project, respectively.

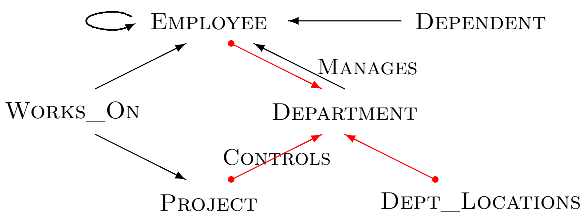


Figure 2 Link Graph for the Referential Integrity Constraints.

4.3 Example Queries

In the following, we will explain a spectrum of example queries to show different features of DDQL.

Selection Queries $\mathcal{Q}_1$, $\mathcal{Q}_2$, and $\mathcal{Q}_3$

The two selection queries $\mathcal{Q}_1$ and $\mathcal{Q}_2$ are relatively simple to solve, since they rely on only one table, namely EMPLOYEE and DEPENDENT, respectively (the latter table is not detailed in this paper, but the schema in Figure 1 contains it). The selection query $\mathcal{Q}_3$ results in the following NL query $\mathcal{Q}_N$:

```
salary, of, 'Research'
```

The database user might not know that SALARY is an attribute of the table EMPLOYEE, and that `Research` is the value of the attribute DNAME of the table DEPARTMENT. The query is annotated, and a list $\mathcal{Q}_F$ of query atoms in field notation is generated:

```
employee:['SSN':B, 'SALARY':C],
employee:['SSN':B, 'DNO':A],
department:['DNUMBER':A, 'DNAME':'Research'].
```

The variables `A`, `B`, and `C` are automatically generated by the system. The second atom links the first and the third; here, the WORKS_FOR relationship is integrated in EMPLOYEE using its last argument DNO. It would also be conceivable to link EMPLOYEE and DEPARTMENT by an atom associated with the MANAGES relationship to return the salary of the manager of the research department, or even by a longer path using the relationships WORKS_ON and CONTROLS to return the salary of all employees working for a project that is controlled by the research department. The former, alternative path through MANAGES might be offered to the user, but the latter, the longer path through WORKS_ON and CONTROLS seems unlikely.

The list $\mathcal{Q}_F$ could be optimized using the database schema; the two `employee` atoms could be combined into a single one, since they share the variable `B` for the key SSN: the result is the atom

```
employee:['SSN':B, 'SALARY':C, 'DNO':A].
```

Thus, from $\mathcal{Q}_F$, a Datalog query $\mathcal{Q}_D$ is generated:

```
select(A, B, C) :-
   employee(_, _, _, B, _, _, _, C, _, A),
   department('Research', A, _, _).
```

From $\mathcal{Q}_D$, an SQL query is compiled, which could be presented to the user together with its resulting table:

```
use company;
select employee.DNO, employee.SSN, employee.SALARY
from employee, department
where department.DNAME = 'Research'
and employee.DNO = department.DNUMBER;
```

The – intermediate – Datalog query $\mathcal{Q}_D$ can be used for evaluation or clarification. If the rule system of the user is complicated or refers to further Datalog rules, ontologies or XML tables, then $\mathcal{Q}_D$ cannot be translated to SQL and has to be evaluated in DDbase directly.

In general, the situation can be more complicated, as we can have multiple occurrences of the same table and we need aliases, e.g. for $\mathcal{Q}_5$. Moreover, the linking atoms can be ambiguous and we may need aggregations.

Aggregation Query $\mathcal{Q}_4$

Consider the following NL query $\mathcal{Q}_N$:

```
sum, of, salary, of, department, name
```

The database user might want to know the sum of the salaries of the employees grouped by the names of their departments. The query is annotated, and a list $\mathcal{Q}_F$ of query atoms in field notation is generated:

```
aggregation:[C, sum(D)],
employee:['SSN':F, 'SALARY':D],
employee:['SSN':F, 'DNO':E],
department:['DNUMBER':E, 'DNAME':C].
```

The system found out that the word `name` in the query refers to the attribute DNAME of DEPARTMENT. The third atom links the second and the fourth; here, the WORKS_FOR relationship is integrated in EMPLOYEE using its last argument DNO. The aggregation is encoded as a special atom. This list could be optimized using the database schema; the two `employee` atoms could be combined into a single one, since they share the value `F` for the key SSN, namely to the atom `employee:['SSN':F ,'SALARY':D ,'DNO':E]`. Thus, from $\mathcal{Q}_F$, a Datalog query $\mathcal{Q}_D$ is generated:

```
select(A, B) :-
   ddbase_aggregate( [C, sum(D)],
     ( employee(_, _, _, F, _, _, _, D, _, E),
       department(C, E, _, _) ),
     Xs ),
   member([A, B], Xs).
```

`ddbase_aggregate/2` is a meta–predicate provided with DDbase. From its result `Xs`, pairs `[A, B]` are selected using `member/2`. Here, this gets compiled to an aggregation with a `Group By` clause in SQL. From $\mathcal{Q}_D$, an SQL query is compiled, which can be presented to the user together with its resulting table:

```
use company;
select department.DNAME, sum(employee.SALARY)
from employee, department
where employee.DNO = department.DNUMBER
group by department.DNAME;
```

Moreover, it is possible in DDbase to use more general, user–defined aggregation functions. E.g., for `list(D)` (instead of `sum(D)`), $\mathcal{Q}_D$ would be directly evaluated in DDbase to produce a *non–first normal form (NFNF, NF^2)* relation showing a list of the salaries grouped by the departments `C`, which is not possible in SQL:

NF^2 Relation	
DNO	SALARIES
1	'NULL'
4	25000, 43000, 25000
5	30000, 40000, 38000, 25000

Query $\mathcal{Q}_5$ with Aliases

The following NL query $\mathcal{Q}_N$ needs multiple occurrences of the same table and aliases:

```
name, of, supervisor, of, exmployee, name
```

The database user might want to know the names of the supervisors of the employees. The query is annotated, and a list $\mathcal{Q}_F$ of query atoms in field notation is generated:

```
employee:['SSN':D, 'LNAME':B],
employee:['SSN':D, 'SUPERSSN':C],
employee:['SSN':C, 'LNAME':A].
```

The second atom links the first and the third. The optimizer could combine the first two atoms, since they agree on the key `SSN`, to the atom

```
employee:['SSN':D, 'LNAME':B, 'SUPERSSN':C].
```

Notice, that the third atom cannot be merged since it differs on `SSN`. Thus, from $\mathcal{Q}_F$, a Datalog query $\mathcal{Q}_D$ is generated:

```
select(A, B) :-
   employee(_, _, B, D, _, _, _, _, C, _),
   employee(_, _, A, C, _, _, _, _, _, _).
```

Note that the SSNs of the supervisor (`C`) and the employee (`D`) are not part of the result. From $\mathcal{Q}_D$, an SQL query is compiled, which could be presented to the user together with its resulting table:

```
use company;
select employee__1.LNAME, employee__2.LNAME
from employee employee__1, employee employee__2
where employee__1.SUPERSSN = employee__2.SSN;
```

4.4 General Aspects of DDQL

The DdQl approach – exemplified in the case study – can be applied to knowledge databases – e.g. relational or deductive databases. The following aspects have to be taken into account.

Links Based on Query Order

For successor atoms of the annotated query $\mathcal{Q}_F$, links have to be found, which might not be unique or obvious. In fact, also for the queries $\mathcal{Q}_3$ and $\mathcal{Q}_4$ the links based on Works_For were not unique, but they were taken to follow the direction from Employee to Department in Figure 1. If Department were to be mentioned before Employee in the query, then it would be more likely that the user–intended semantics would be based on Manages.

Multiple Links

The atoms in $\mathcal{Q}_F$ might not appear consecutively in the graph of Figure 1. There might be several linking trees, or the atoms could be on a cycle. DdQl is collecting *heuristics* for finding suitable links. For $\mathcal{Q}_3$ and $\mathcal{Q}_4$ these links were affected by the order of the words in the query.

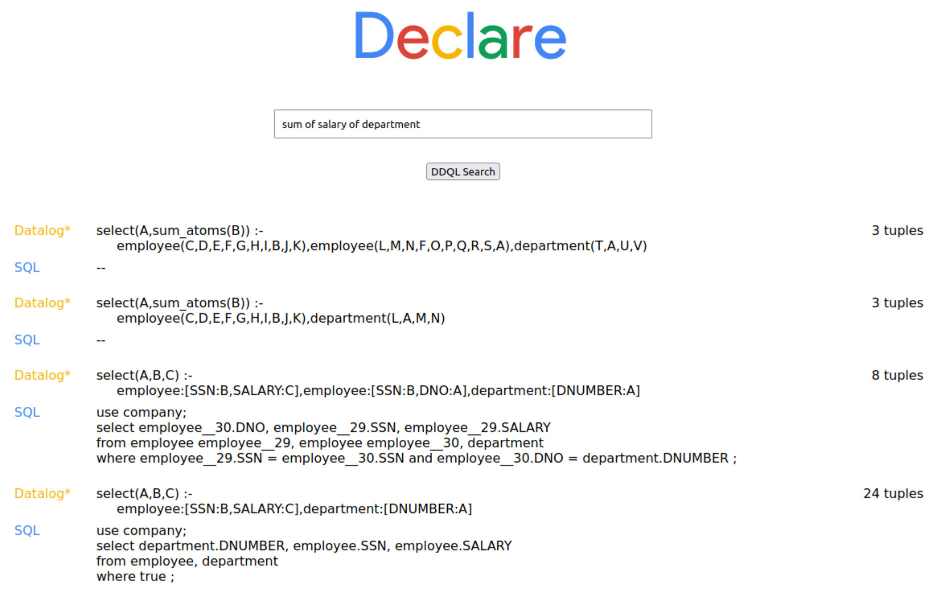
However, if we were to ask about Employee and Pname, then there would be two equally reasonable links, namely through Works_On and through Department (which could itself be supported by two different foreign–key constraints). These two cases can be expressed by the following Prolog clauses:

```
select(Lname, Pname) :-
   employee(_, _, Lname, Ssn, _, _, _, _, _, _),
   works_on(Ssn, Pno, _),
   project(Pname, Pno, _, _).
select(Lname, Pname) :-
   employee(_, _, Lname, _, _, _, _, _, _, Dnum),
   department(_, Dnum, _, _),
   project(Pname, _, _, Dnum).
```

In the second query, the linking atom `department(_, Dnum, _, _)`, which is produced because of the foreign key constraints between Employee and Department and between Department and Project, could be redundant. It makes a difference when there is no department with the number `Dnum` referenced by Employee and Project. In that case, the user has to decide about whether he wants to include this atom or not. If the query were to contain a keyword that is similar to a link, then the link would be preferred. In general, all corresponding queries can be presented to the user, for a choice to be made.

Similarities and Subsumptions

So far, we have not yet discussed similarities or subsumptions between words from the query and the terms used in the database. Here, ontologies or concepts from linguistics can be applied. A simple case would be the *Levenshtein Distance* (Edit Distance) between words or conversions between singular and plural. More complicated cases could be handled by background knowledge such as translations between languages.



```
Datalog*   select(A,sum_atoms(B)) :-                                                      3 tuples
               employee(C,D,E,F,G,H,I,B,J,K),employee(L,M,N,F,O,P,Q,R,S,A),department(T,A,U,V)
SQL        --

Datalog*   select(A,sum_atoms(B)) :-                                                      3 tuples
               employee(C,D,E,F,G,H,I,B,J,K),department(L,A,M,N)
SQL        --

Datalog*   select(A,B,C) :-                                                               8 tuples
               employee:[SSN:B,SALARY:C],employee:[SSN:B,DNO:A],department:[DNUMBER:A]
SQL        use company;
           select employee__30.DNO, employee__29.SSN, employee__29.SALARY
           from employee employee__29, employee employee__30, department
           where employee__29.SSN = employee__30.SSN and employee__30.DNO = department.DNUMBER ;

Datalog*   select(A,B,C) :-                                                              24 tuples
               employee:[SSN:B,SALARY:C],department:[DNUMBER:A]
SQL        use company;
           select department.DNUMBER, employee.SSN, employee.SALARY
           from employee, department
           where true ;
```

Figure 3 Graphical User Interface of DDQL: Variant of Query $\mathcal{Q}_4$.

Parsing Sentences and Rules

With a powerful speech–to–text component, we could aim to *parse* more complex structures for sentences. Then we could enable the user to define subqueries producing views. With DDQL, it is already possible to define additional Datalog predicates with rules to be used – like SQL views – in further queries. Also, sometimes referential integrity constraints can be inferred for these new predicates.

4.5 The Graphical User Interface

A prototype of the graphical user interface (GUI) of DDQL is shown in Figure 3. After entering the keywords separated by blanks, a list of possible search queries in Datalog* is generated. The generated queries might be further optimized. Currently, a corresponding SQL query is shown, if there are no aggregation functions. Obviously, for user–defined aggregation functions, only the Datalog* variant is possible. Figure 3 also shows the number of result tuples, such that the user can choose a possible alternative answer. A ranking of the results would be possible based on these numbers, the contextual knowledge, and the query history – but so far we have not finalized that.

4.6 Benchmark – Database Schemas and Queries

We have investigated the database schemas of a collection of relational databases. The tuples `[DB, T, A, F]` listed below show the following:

(a) `DB` is the name of the database,
(b) `T` is the number of tables in the database,
(c) `A` is the average number of attributes per table in the database,
(d) `F` is the average number of foreign keys per table in the database.

Most databases did not provide foreign key constraints. Only the database `company` provided foreign key constraints, namely 1.33 on average per table.

```
Tuples = [
   [alignment, 5, 3.4, 0], [company, 6, 4.67, 1.33],
   [evu, 20, 5.7, 0], [morphemes, 1, 7, 0], [selli, 1, 7, 0],
   [stock, 6, 3.33, 0], [wm_2002, 9, 7.33, 0] ]
```

I.e., for most databases DdQl has to rely on the query log files with previous queries, which we can parse to Xml using our tool Squash, to learn about possible join conditions.

In a benchmark with many queries to the relational database `company` – which has 1.33 foreign key constraints per table on average – about 4 to 5 different alternative suggestions were generated on average per query. The answer intended by the user was always one of them. For the single query

```
salary, of, exmployee, '1111'
```

many (i.e. 20) different alternative suggestions were generated. We are currently trying to reduce this number for similar cases. Without this query, only 3.5 alternative suggestions were generated on average.

5 Conclusions

In this paper, we have shown how queries to relational databases can be answered in keyword–based natural language interfaces using intelligent, cooperative techniques based on logic programming and deductive databases.

The concepts mentioned in the query are linked based on contextual background knowledge, mainly from the database schema. Also Datalog–like rules can be added as background knowledge, e.g. for deductive databases – without a database schema. In the deductive database DDbase, the knowledge could also be hybrid – including ontological, linked data (Rdf, Owl), Swrl knowledge and semi–structured Xml documents – and hybrid queries could become possible in DdQl.

Various styles or patterns of the database schema design can significantly influence the level of application of our approach. E.g., under the *universal relation scheme assumption* (URSA) [10] we could simply use natural join queries.

Currently, we are containerizing Declare – inluding DdQl – in a docker image. In the future, we are planning to add a voice recognition part, especially for using DdQl on mobile devices. Also the knowledge acquisition could be done with a voice assistant based on a domain–specific language, see [17].

We are constantly incorporating further aspects of AI into DdQl. In the future, also concepts from subsymbolic AI will be investigated and included where useful. For instance, by looking at the user behaviour from previous queries, we may derive heuristics for finding intended queries (e.g. linking atoms) using some form of *machine learning*.

References

1 Katrin Affolter, Kurt Stockinger, and Abraham Bernstein. A Comparative Survey of Recent Nlis for Databases. *VLDB J.*, 28(5):793–819, 2019. doi:10.1007/s00778-019-00567-8.

2 Lukas Blunschi, Claudio Jossen, Donald Kossmann, Magdalini Mori, and Kurt Stockinger. Soda: Generating Sql for Business Users. *Proc. VLDB Endowment*, 5(10):932–943, 2012. doi:10.14778/2336664.2336667.

3 Ivan Bratko. *Prolog Programming for Artificial Intelligence*. Addison–Wesley Longman, 4th edition, 2011.

4 Gerhard Brewka, Thomas Eiter, and Mirek Truszczynski. Answer Set Programming at a Glance. *Communications of the ACM*, 54(12):92–103, 2011.

5 Andreas Böhm, Dietmar Seipel, Albert Sickmann, and Matthias Wetzka. Squash: A Tool for Designing, Analyzing and Refactoring Relational Database Applications. In *Proc. International Conference on Applications of Declarative Programming and Knowledge Management (INAP)*, pages 82–98, 2007.

6 Ceri, Stefano and Gottlob, Georg and Tanca, Laetitia. *Logic Programming and Databases*. Springer, 1990.

7 William Clocksin and Christopher S. Mellish. *Programming in Prolog*. Springer Science & Business Media, 2003.

8 Danica Damljanovic, Milan Agatonovic, and Hamish Cunningham. Nlis to Ontologies: Combining Syntactic Analysis and Ontology–based Lookup through the User Interaction. In *Extended Semantic Web Conf.*, pages 106–120. Springer, 2010.

9 Ramez Elmasri and Shamkant B. Navathe. *Fundamentals of Database Systems, 3rd Edition*. Addison–Wesley Longman, 2000.

10 Hector Garcia-Molina, Jeffrey Ullman, and Jennifer Widom. *Database Systems: The Complete Book*, volume 638. Pearson Prentice Hall, 2009.

11 Ben Goertzel. Perception Processing for General Intelligence: Bridging the Symbolic/Subsymbolic Gap. In *International Conference on Artificial General Intelligence*, pages 79–88. Springer, 2012.

12 Awni Hannun, Carl Case, Jared Casper, Bryan Catanzaro, Greg Diamos, Erich Elsen, Ryan Prenger, Sanjeev Satheesh, Shubho Sengupta, Adam Coates, et al. Deep Speech: Scaling up End–to–End Speech Recognition. *arXiv preprint*, 2014. `arXiv:1412.5567`.

13 Nicola Leone, Gerald Pfeifer, Wolfgang Faber, Thomas Eiter, Georg Gottlob, Simona Perri, and Francesco Scarcello. The dlv System for Knowledge Representation and Reasoning. *ACM Transactions on Computational Logic*, 7(3):499–562, 2006.

14 Fei Li and H. V. Jagadish. Understanding Natural Language Queries over Relational Databases. *SIGMOD Rec.*, 45(1):6–13, 2016. `doi:10.1145/2949741.2949744`.

15 Clayton McMillan, Michael C Mozer, and Paul Smolensky. Rule Induction through Integrated Symbolic and Subsymbolic Processing. In *Advances in Neural Information Processing Systems*, volume 4, pages 969–976, 1992.

16 Jack Minker, Dietmar Seipel, and Carlo Zaniolo. Logic and Databases: A History of Deductive Databases. In Jörg H. Siekmann, editor, *Computational Logic*, volume 9 of *Handbook of the History of Logic*, pages 571–627. Elsevier, 2014. `doi:10.1016/B978-0-444-51624-4.50013-7`.

17 Falco Nogatz, Julia Kübert, Dietmar Seipel, and Salvador Abreu. Alexa, How Can I Reason with Prolog? (Short Paper). In *Proc. 8th Symposium on Languages, Applications and Technologies (SLATE 2019)*, 2019.

18 Christian Schneiker, Dietmar Seipel, Werner Wegstein, and Klaus Prätor. Declarative Parsing and Annotation of Electronic Dictionaries. In *Proc. 6th International Workshop on Natural Language Processing and Cognitive Science (NLPCS 2009)*, 2009.

19 Dietmar Seipel. Declare – A Declarative Toolkit for Knowledge–Based Systems and Logic Programming. URL: `http://www1.pub.informatik.uni-wuerzburg.de/databases/research.html`.

20 Dietmar Seipel. Processing Xml–Documents in Prolog. In *Workshop on Logic Programming (WLP 2002)*, 2002.

21 Dietmar Seipel, Rüdiger von der Weth, Salvador Abreu, Falco Nogatz, and Alexander Werner. Declarative Rules for Annotated Expert Knowledge in Change Management. In *Proc. 5th Symposium on Languages, Applications and Technologies (SLATE 2016)*, 2016.

22 Steffen Staab and Rudi Studer, editors. *Handbook on Ontologies*, International Handbooks on Information Systems. Springer, 2009. `doi:10.1007/978-3-540-92673-3`.

23 Kurt Stockinger. The Rise of Natural Language Interfaces to Databases. ACM SIGMOD Blog, URL: `https://blog.zhaw.ch/datascience/the-rise-of-natural-language-interfaces-to-databases/`, 2019.

Sentiment Analysis of Portuguese Economic News

Cátia Tavares
Instituto Universitário de Lisboa (ISCTE-IUL), Lisboa, Portugal

Ricardo Ribeiro
Instituto Universitário de Lisboa (ISCTE-IUL), Lisboa, Portugal
INESC-ID, Lisbon, Portugal

Fernando Batista
Instituto Universitário de Lisboa (ISCTE-IUL), Lisboa, Portugal
INESC-ID, Lisbon, Portugal

Abstract

This paper proposes a rule-based method for automatic polarity detection over economic news texts, which proved suitable for detecting the sentiment in Portuguese economic news. The data used in our experiments consists of 400 manually annotated sentences extracted from economic news, used for evaluation, and about 90 thousand Portuguese economic news, extracted from two well-known Portuguese newspapers, covering the period from 2010 to 2020, that have been used for training our systems. In order to perform sentiment analysis of economic news, we have also tested the adaptation of existing pre-trained modules, and also performed experiments with a set of Machine Learning approaches, and self-training. Experimental results show that our rule-based approach, that uses manually written rules related to the economic context, achieves the best results for automatically detecting the polarity of economic news, largely surpassing the other approaches.

2012 ACM Subject Classification Computing methodologies → Machine learning; Information systems → Language models; Computing methodologies → Natural language processing; Applied computing → Economics; Computing methodologies → Language resources; Computing methodologies → Artificial intelligence; General and reference → Measurement

Keywords and phrases Sentiment Analysis, Economic News, Portuguese Language

Digital Object Identifier 10.4230/OASIcs.SLATE.2021.17

Funding This work was supported by national funds through FCT, Fundação para a Ciência e a Tecnologia, under project UIDB/50021/2020.

1 Introduction

Economic data and economic indicators are an important resource to reveal the true picture of an economy's condition. They allow us to understand the state of the economy and to determine our investment and consumption decisions. Also, forecasters and policy makers need information about how economy stands to take appropriate responses to their decisions. To give response to that and given the fact that economic indicators have usually a lag in their publication, having mostly a monthly and quarterly frequency, we should take advantage of the increasingly digital world that we have and transform the exponential amount of information available in an opportunity. Taking into consideration that news are the main form of transmission of information about the present, they can generate changes in the expectations of their readers. If the news are positive, the expectations of economic agents will also be positive and, consequently, the sentiment about the future of the economy as well. Otherwise, mistrust will be generated about the situation of the economy, which may have repercussions on economic agents investment and consumption actions [22].

Currently, a large amount of information is shared in news sites, blogs, and social networks. If processed timely and adequately, it can help to obtain key insights about the economic situation in almost real time. Sentiment analysis, also known as opinion mining, is the task

© Cátia Tavares, Ricardo Ribeiro, and Fernando Batista;
licensed under Creative Commons License CC-BY 4.0

of finding the opinion of an author of a text concerning its content [9]. In general, it consists of identifying the polarity, positive or negative, of a text span (e.g., a document or a sentence) that contains "explicit opinions, beliefs, and views about specific entities" (a subjective text span).

Considering that the available linguistic resources for the Portuguese language are scarce, sentiment analysis of texts in Portuguese is still an active field of research, especially considering documents in specific domains, such as the economic domain [23]. In this work, we explore three approaches to perform sentiment analysis of Portuguese economic news. Our baseline approach consists of translating Portuguese economic news data into English, and then apply well-known and widely used sentiment resources for English, such as VADER [15] and TextBlob[1]. In the second approach, we manually created a set of rules based on the economic context, and used a rule-based approach. Finally, we trained different machine learning models, in order to try to improve our results even further. The performance of each one of the previous approaches was evaluated using a manually labeled dataset, containing 400 economic sentences, created in the scope of this work.

The rest of this document is organized as follows: Section 2 presents an overview on the related literature; Section 2.1 overviews the different strategies commonly used to perform sentiment analysis, while Section 2.2 focus on sentiment analysis in the economic context; Section 3 presents a description of our news data in Section 3.1 and the description of our Golden Data manually annotated in Section 3.2; Section 4 presents the proposed pipeline of our work; Section 5 describes the experiments performed with each one of our adopted approaches, namely, the translation-based approach (Section 5.1), the rule-based approach (Section 5.2), and the Machine Learning approach (Section 5.3), with Section 5.4 presenting a summary of the results attained; finally, Section 6 presents the major conclusions, and recommendations for future work.

2 Related Work

This section describes some relevant literature. We start by addressing sentiment analysis in general, and then we focus on its application in the economic domain.

2.1 Sentiment Analysis

As previously mentioned, sentiment analysis focuses on the analysis of users' expressions, classifying them according to the polarity. Data from different types of sources such as blogs, news, and social media, the use of different languages, non-standard words and the use of emojis and other symbols led to approaches with distinct complexity levels.

Sentiment analysis has gained an important role in the analysis and understanding of consumer communication in the media, allowing to provide key information about the public opinion on several subjects [31]. In that sense, in recent years, in addition to the more traditional focus on data from different news services, research in the field of sentiment analysis has been carried out in several domains, focusing on the analysis of data from social networks such as Twitter [12]. The difference between these two types of textual data is that in the latter the opinion is generally clear, objective and is well defined in the text, while the first may cover several domains and may consist of more subjective texts and descriptions of complex and context-based events [3].

[1] `https://textblob.readthedocs.io`

There are several techniques for sentiment analysis. There are approaches based on lexicons, which consist of predefined collections/dictionaries of terms and the associated sentiment/emotion; approaches based on Machine Learning (ML); and, even hybrid approaches, in which both the previous approaches are combined. ML techniques can also be divided into two groups, supervised techniques and unsupervised techniques. In the first case, the data must be labeled, which is not the case with unsupervised techniques [18, 7].

Sentiment dictionaries are dictionaries where each word is associated with and opinion/polarity (positive, negative or neutral) and are very useful resources to classify the sentiment polarity. There are many sentiment dictionaries based on the English lexicon, however, for the Portuguese language these resources are scarce. The literature about sentiment analysis focusing on the English language is vast but the linguistic resources available for sentiment analysis in Portuguese and other languages are still limited. Several studies adopt an approach based on the translation of the original data to English and after that an English sentiment analysis tool is applied. However, translation errors and language specific information can have a significant impact on the final result [23].

In the specific case of Portuguese, well-know sentiment analysis tools, like VADER, TextBlob, or Stanza [24], do not work. VADER combines a lexicon and a rule-based approach for sentiment analysis. VADER original experiments were performed only on English data. TextBlob is a Python library that provides several natural language processing modules, including one for sentiment analysis. TextBlob includes two sentiment analysis approaches, a rule-based model and a supervised ML model, based on a Naïve Bayes classifier. As provided, it only deals with the English language. Stanza toolkit also uses a ML model for sentiment analysis, in this case based on a Convolutional Neural Network classifier. Stanza is also a Python library and has models for English, Chinese, and German.

The ML approaches rely on ML algorithms to determine the sentiment as a text classification problem. Given a phrase/instance of unknown class, the model predicts the label/class to which it belongs. Supervised methods require the existence of labels in the training data, and example of supervised classifiers are Support Vector Machine (SVM) and decision trees [18, 29]. The unsupervised methods do not require the existence of labels and have been subject of a lot of research.

Depending on the approach used to perform SA, before classifying the sentiment it is necessary to extract and select the features of the text. Feature extraction consists of performing a transformation to the original features and generate more significant ones, aiming to reduce complexity and provide simpler representations to the data [16, 18]. During the feature extraction process, useful features are identified and extracted, and analysis can also be done to understand which features increase accuracy the most. To help weight features, measures such as TD-IDF (term frequency-inverse document frequency) are used. After extracting the features, the sentiment classification is done.

Ahmed and Ahmed [2] used an approach based on lexicons to classify data from news. Firstly, they used text-preprocessing techniques, such as punctuation removal, “stop-words” removal and stemming. After reducing the derived words, they computed the polarity using TF-IDF measure to identify the most frequent words and to be able to assign them sentiment scores through dictionaries such as SentiWordNet. Finally, the news polarity was determined as positive, negative or neutral, by the sentiment average of the total news words.

Mohamed [19] evaluated several algorithms to perform sentiment analysis based, concluding that SVM outperforms other methods such as Naïve Bayes and decision trees. However, each SA technique will have different performance and results depending on the data in which it is applied.

2.2 News, Sentiment Analysis and the Economy

The relationship between economic news, the economy and public perception, and opinion has been a subject of research for a while. The news have influence on the evaluations and opinions of economic agents about the economy. When they are negative, public opinion about future economic conditions is unfavorable and pessimism about the economy is generated [8, 13]. The importance of public opinion is due to the fact that changes in expectations about the economic future can be a source of economic fluctuations [11].

In order to understand the current state of the economy, high-frequency information is needed quickly and in real time [26]. This way, economic agents can use a multitude of high-frequency information in order to guide their actions, including news from the media [28].

There are several studies that focus on understanding the behavior of financial markets and stock values based on economic news [30], some of them are shown in Table 1. Mining the news plays an important role in designing strategies to predict market behavior and, based on events and news items, it is possible to predict market prices [17]. When there is pessimism in the media, patterns of falling stock prices and short-term returns are expected, concluding that the news information is useful for making predictions of market return and risk [27, 6].

In relation to the foreign exchange market, text mining is also a promising way to predict exchange rate movements based on the economic events present in the news, bringing benefits to investors and risk managers [20].

According to Huang et al. [14], traditional economic indicators based on surveys have been replaced by techniques for extracting sentiments from news texts and central bank statements, through the application of machine learning and other computational techniques. News-based sentiment indicators make it possible to predict periods of financial crisis, serving as early indicators of them. Nyman et al. [21], showed that periods of financial crisis can be detected in the news, being preceded by sentiments of anxiety. From the Bank of England news and publications data, they were able to obtain information about episodes of risk and market volatility.

Aguilar et al. [1], when trying to monitor economic activity in Spain by building a sentiment indicator based on the news, found that the developed indicator has advantages over the indicator based on surveys in GDP forecasting and in forecasting the crisis related to COVID-19. Also according to Fraiberger et al. [10], the sentiment indicator based on the sentiment present in the news gives a direct and real-time view of the aggregate sentiment of the current and future state of the economy, correctly portraying fluctuations in GDP, which allows policy makers to react more efficiently to economic conditions.

3 Data

This work uses a large dataset of economic news for training our models, and a dataset of labeled news that serves as reference for our evaluation experiments. This section presents the details about the two datasets.

3.1 Portuguese Economic News

The data used for training our models consists of economic news produced between January 1st, 2010 to December 31st, 2020, covering an 11 years time-span (132 months). The corpus was extracted from online news published by *Expresso* and *Público*, two well-known Portuguese newspapers. For each article, we have extracted the date, the headline and the

Table 1 Sentiment analysis in economic context.

Ref.	Goals and Results	Data	Techniques	Metrics
[14]	Use news data to predict periods of financial crisis ——— News sentiment index contains useful information to predict financial crises and market risk	Financial Times News	– Word vector representation – Semantic clustering – Sentiment of each cluster	Precision Recall F-score
[21]	Use text analysis to extract statistics about the economy, predicting important events and systemic risk ——— Strong correlation with financial market events, such as structural breaks, and with other market measures such as sentiment, confidence, market volatility and systemic risk	– Comments about the bank of England market – Financial market research reports – Economic news	– Word count – Loughran and McDonald sentiment dictionary	Granger causality p-value
[5]	Create indicator to predict the state and evolution of the economy in France (GDP) ——— Media are a promising tool for economic analysis and have made it possible to forecast French GDP	Le Monde News	– Construction of a sentiment dictionary – Logistic regression	RMSFE
[10]	Create sentiment index to predict economic fluctuations ——— Index gives a direct and real-time view of the aggregate sentiment of the current and future state of the economy, correctly portraying GDP fluctuations	Economic News	Loughran and McDonald sentiment dictionary (economy) and Young and Soroka dictionary (economy and politics)	Granger causality p-value RMSE
[1]	Create sentiment indicator to monitor economic activity in Spain in real time ——— Correlation of the indicator developed with the Economic Sentiment Indicator (ESI) of 0.8. Better performance than ESI in forecasting GDP and the economic crisis related to COVID-19. Better GDP forecast when we use the indicator developed compared to the ESI.	Economic News	Count words related to improvements and economic downturns	RMSE

Table 2 Analysis of the number of words in the headlines and text descriptions.

	Headline	Description
Average length	9.08	15.31
Median length	9	15
Maximum length	31	235

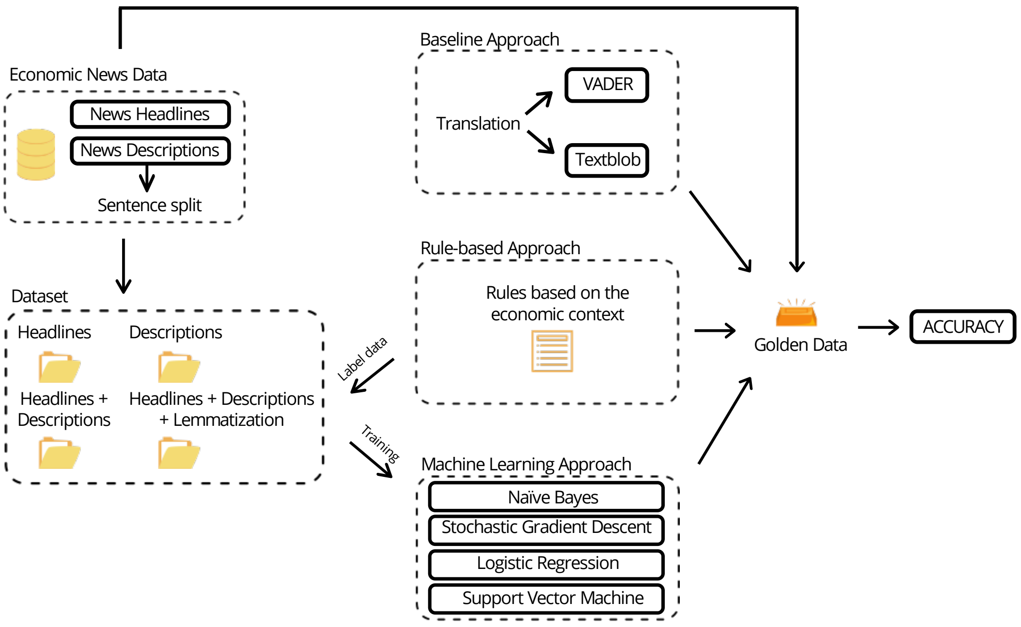


Figure 1 Automatic classification of economic news pipeline.

corresponding description, when available. This accounts to over 90,000 economic news headlines, 62,326 of them also complemented with a textual description, which can contain one or more sentences. Table 2 shows some statistics about the number of words present in each one of the fields for each article.

3.2 Manually Annotated Data

In order to evaluate our approaches, we have collected a sample of 400 sentences from recent economic news, and manually classified them with one of three possible labels, according to its corresponding polarity: Negative, Neutral, and Positive.

4 Pipeline

In order to automatically classify the sentiment of Portuguese economic news, we have adopted the following strategy, represented in Figure 1, that consists of a data collection stage, an automatic classification stage, and the evaluation. We have started by collecting the data from online newspapers, as described in Section 3. Each one of the news stories was processed in order to extract the corresponding date, title (headline), and description. Additionally to collecting the data from 2010 to 2020, we also have selected 400 sentences, extracted from most recent news, that were manually annotated with the purpose of evaluating the approaches under study.

In terms of available natural language processing tools, the Portuguese language may be considered a low-resource language, and during the course of this work, we could not find a sentiment analysis tool that could be directly applied to detect the sentiment of a sentence in Portuguese. For that reason, our initial strategy, represented in the top-middle box of Figure 1, consisted of translating the Portuguese sentences into English, and then using one of the existing English tools. However, sentiment analysis is known to be domain dependent, and soon we have realised that the commonly used tools could not be easily applied to the economic news domain. So, in order to overcome this problem, we have manually created a set of rules adapted to the economic domain, and we adopted the rule-based approach represented in the middle box of the figure. Finally, we have used our ruled-based approach to label our large dataset of economic news, and, in order to improve our results even further, we have trained several machine learning models, both in a supervised and semi-supervised way, as represented in the bottom rectangle of Figure 1. All the described approaches are evaluated using the same manually labeled dataset, described in Section 3.

5 Experiments and Evaluation

In this section, we present the details about the three different approaches used to perform sentiment analysis. As previously mentioned, first we have tried to use existing tools to perform sentiment analysis, but we soon realised that the resources available for the Portuguese language are scarce. Thus, as a first approach, we translated our data into English and then used VADER and TextBlob to perform the analysis. We concluded that this approach is limited when applied to the economic context. So, we tried a second approach where we observed the most common patterns appearing in economy news stories and created a set of rules to classify each sentence, which proved to perform well in our data. In order to improve our results even further, we experimented a third approach where we trained different machine learning models. In the end of the section, we present a summary with the results obtained with the mentioned approaches.

5.1 Baseline/Translation-based Approach

When facing the lack of tools for a given language, one possible immediate solution is to translate the existing data to another language and then use the available tools for that language. In fact, during the course of this work, we did not find any available tools to perform sentiment analysis in Portuguese. As so, we have adopted TextBlob and NLTK VADER [15], two well-known tools for sentiment analysis, with the latter reported to perform well when applied to the finance domain [25]. So, after translating our reference data from Portuguese to English using Googletrans, a python library that implemented Google Translate API, VADER achieved an accuracy of 46.5% and TextBlob achieved an accuracy of 32.0%. These results show that this approach is not suitable to the economic context, which was not an unexpected result since we know that sentiment analysis is a domain-dependent task.

We have then applied these tools to our unlabeled data in order to analyse the results in more detail. From the analysis we have observed that, for example, headlines with negative words like unemployment, crisis, deficit, etc., were classified incorrectly most of the times. In fact, the polarity associated with these words is negative, although we have seen that many news involving these words, such as "*unemployment is decreasing*" and "*crisis is slowing down*", should be positive, and that VADER and TextBlob were not taking that into consideration.

Table 3 Expressions related to "unemployment".

Word 1	Word 2	Word 3	Sentiment
unemployment	reaches	minimum	1
unemployment	reaches	maximum	−1
unemployment	decreased		1
unemployment	increased		−1
...			
unemployment			−1

5.2 Rule-based Approach

Our rule-based approach is similar to the approach proposed by Aguilar et al. [1] for classifying the polarity of economic news, where the sentiment attributed to each headline is also based on rules. For example, when combined with the word "economy", the word "increase" becomes positive, and the word "decrease" becomes negative.

After identifying the errors and limitations in the classification performed by the previous approach, we have started looking at the more prominent words and combination of words in our unlabeled data. We have manually analysed through frequency analysis, the set of words co-ocurring with words like "unemployment", to understand the most frequent patterns. As a result of the analysis of these associations of words, we constructed a list of expressions/rules related to the economic context, and labeled the sentiment associated to them. We have observed meaningful combinations of two and three words, which derived in rules of one, two and three words, accordingly. For example, for the word "unemployment", we can think of expressions involving words such as the ones presented in Table 3. We did the same for other words related to the economic context such as "consumption", "debt", "economy", "recovery", and we end up with approximately 600 rules with the corresponding associated sentiment (-1 if the expression is negative and 1 if it is positive).

Algorithm 1 details our rule-based classification process. First we try to match all the rules involving 3-words expressions. If more than one rule can be applied, we sum the sentiment associated with all the matching rules, and check if the resulting sum is positive, negative, or neutral. If none of the 3-words rules matches our sentence, we try to search all the rules involving 2-words, and again sum the sentiment of each one that we find. Finally, if none of the 2-words rules matches our sentence, we try to match 1-word rules. At the end of this process, if the sentence did not match any rule, then we assume it is neutral. When applied to our corpus of 90,000 news, 9.5% of the headlines where classified as negative and 7.5% as positive. The descriptions, 5.8% were classified as negative and 5.8% as positive. When applied to our reference data, our rule-based approach achieves an accuracy of 86.3%, a significant improvement over the baseline approaches.

5.3 Machine Learning Approach

In order to improve the results even further, we have performed additional experiments using our unlabeled economic news dataset for training our machine learning models.

The dataset described in Section 3.1 was used to create four different collections of texts that were used in our Machine Learning experiments: 1) sentences extracted from the headlines (about 90,000 sentences); 2) sentences extracted from the descriptions (about 85,000 sentences); 3) a combination of the previous two collections (about 175,000 sentences); and 4) a combination of the previous collection with its variant where lemmatization was

Algorithm 1 Classification of each sentence based in the rules created.

```
input: sentence, rules

sentiment = 0
for rule in [rules with 3 words]:
    if rule.applies_to(sentence):
        sentiment += rule.sentiment
if sentiment = 0 then
    for rule in [rules with 2 words]:
        if rule.applies_to(sentence):
            sentiment += rule.sentiment
if sentiment = 0 then
    for rule in [rules with 1 word]:
        if rule.applies_to(sentence):
             sentiment += rule.sentiment

if sentiment < 0 then
    return -1
else if sentiment > 0 then
    return 1
else
    return 0
```

applied to the words in the texts, in order to capture a broader set of economic terms (about 350,000 sentences). Lemmatization is a preprocessing step often used in text mining and natural language processing, that consists of converting each word in its basic/root form, analyzing its morphology in order to remove the inflected affixes, leaving only the lemma [4]. For this task we used Spacy and, for example, this process will convert the words "increasing", "increased", and "increases" into the word "increase".

Our rule-based classifier was used to classify all the sentences in each one of the collections. Then, we have converted all the labeled sentences into their corresponding document representation, using unigrams, bigrams, and trigrams.

We have applied the following classical supervised machine learning methods, used extensively for classification and regression tasks: Naïve Bayes (NB), Stochastic Gradient Descent (SGD), Logistic Regression (LR), and Support Vector Machines (SVM). Each one of the methods was applied to each one of the four previously described text collections, using their default parameters. Concerning the feature weights, we have used simple counts for Naïve Bayes, and Term Frequency – Inverse Document Frequency (TF-IDF) weights for all the other methods. The corresponding evaluation results for our reference data are presented in Table 4.

The results attained show that, in general, the text contained in the title is better for training than the text of the descriptions, but the best results are achieved when combining both fields. With NB and SVM we could see that the use of lemmatization contributed to a better result, we could not see the same when using SGD and LR.

Using only the title texts for training leads to a better accuracy with LR (74.0%), and using only the sentences from descriptions perform better with SGC (73.0%). The collection containing the titles and the descriptions led to better accuracy scores for SGD (76.3%), but the best accuracy (77.0%) was achieved by training the SVM with features produced with lemmatization.

Table 4 Model evaluation in our reference Data.

Model	Accuracy	Precision	Recall	F1
VADER (baseline)	0.465	0.477	0.465	0.470
TextBlob (baseline)	0.320	0.352	0.320	0.301
Rule-based approach	**0.863**	**0.863**	**0.863**	**0.863**
Naïve Bayes				
Titles	0.615	0.638	0.615	0.607
Descriptions	0.593	0.645	0.593	0.591
Titles + Descriptions	0.633	0.648	0.634	0.627
Titles + Descriptions + Lemmatization	0.663	0.675	0.663	0.661
Stochastic Gradient Descent				
Titles	0.740	0.739	0.740	0.739
Descriptions	0.730	0.743	0.730	0.733
Titles + Descriptions	0.763	0.763	0.763	0.762
Titles + Descriptions + Lemmatization	0.740	0.738	0.740	0.739
Logistic Regression				
Titles	0.738	0.737	0.738	0.737
Descriptions	0.693	0.718	0.693	0.698
Titles + Descriptions	0.758	0.764	0.758	0.759
Titles + Descriptions + Lemmatization	0.758	0.764	0.758	0.759
Support Vector Machine				
Titles	0.738	0.736	0.738	0.736
Descriptions	0.678	0.697	0.678	0.683
Titles + Descriptions	0.755	0.761	0.755	0.757
Titles + Descriptions + Lemmatization	0.770	0.773	0.770	0.771

We also have performed additional self-training classification experiments, considering all the texts labeled as positive or negative as the initial labels, and performing label propagation to all the remainder data. Nonetheless, the results achieved did not surpass our previous reported results.

5.4 Summary

Table 4 summarizes the results achieved with each one of the approaches when evaluated using our reference data. The baseline approaches, using the NLTK VADER and TextBlob, performed poorly in the economic context, where accuracies of 46.5% and 32.0% were obtained, respectively. The best result was obtained with the rule-based approach with an accuracy of 86.3%. The machine learning approaches were not able to surpass our rule-based system: the best result of the machine learning approaches was achieved using SVM, with an accuracy of 77.0%.

6 Conclusions and Future Work

The lack of tools for sentiment analysis for Portuguese and the difficulty to obtain a labeled dataset to train a sentiment analysis system for economic context led us to explore a set of approaches in order to solved this practical problem. First, we have tried a baseline approach where we translated our texts into English and used well-known sentiment analysis tools, such as NLTK VADER and TextBlob. Given the poor results achieved in the economic context, we tried a rule-based approach for which we have created manual rules, based on the economic domain, and used those rules to classify the polarity of each economic text. Finally, we have created a set of machine learning models, based on the large amount of economic texts that we had available, aiming at improving our results even further.

In order to compare and evaluate the performance of the proposed approaches, we have also created a reference dataset, containing 400 economic sentences, manually classified. The performed experiments have shown that the baseline approach achieves poor results, when applied to the economic domain. The rule-based approach achieved an impressive performance of 86.3% accuracy, a significant increase of performance over the baseline approaches. The machine learning models that we have explored were not able to generalise and surpass the rule-based approach.

Our rule-based approach still lacks proper treatment of the negation, and adversative conjunctions. In the near future, in addition to the rules created, we plan to improve the classifier by treating differently words after a word such as "not" or "don't", and consider ways of dealing with the classification of sentences with adversative conjunctions. Concerning the negation, we should classify each sentence with the opposite sentiment of the rule it matches, for example, "*unemployment did not increase*" should have a positive polarity, whereas "*unemployment increased*" has a negative one. Adversative conjunctions introduce additional challenges. They express opposition or contrast between two statements and it is difficult even for a human, to tell the sentiment that it expresses. For example, in "*unemployment decreased but GDP increased*", the statement before the conjunction "*but*" has a positive sentiment, but the statement after it has a negative sentiment. Our rule-based approach would assign a neutral sentiment to this example, since it matches both negative and positive rules.

References

1 Pablo Aguilar, Corinna Ghirelli, Matías Pacce, and Alberto Urtasun. Can News Help Measure Economic Sentiment? An Application in COVID-19 Times. *SSRN Electronic Journal*, 2020. `doi:10.2139/ssrn.3673825`.

2 Jeelani Ahmed and Muqeem Ahmed. A framework for sentiment analysis of online news articles. *International Journal on Emerging Technologies*, 11(3):267–274, 2020.

3 Alexandra Balahur, Ralf Steinberger, Mijail Kabadjov, Vanni Zavarella, Erik Van Der Goot, Matina Halkia, Bruno Pouliquen, and Jenya Belyaeva. Sentiment analysis in the news. In *Proceedings of the 7th International Conference on Language Resources and Evaluation, LREC 2010*, pages 2216–2220. ELRA, 2010. `arXiv:1309.6202`.

4 Ivan Boban, Alen Doko, and Sven Gotovac. Sentence retrieval using Stemming and Lemmatization with different length of the queries. *Advances in Science, Technology and Engineering Systems*, 5(3):349–354, 2020. `doi:10.25046/aj050345`.

5 Clément Bortoli, Stéphanie Combes, and Thomas Renault. Nowcasting GDP Growth by Reading the Newspapers. *Economie et Statistique / Economics and Statistics*, 505-506:17–33, 2018. URL: `https://EconPapers.repec.org/RePEc:nse:ecosta:ecostat_2018_505-506_2`.

6 Charles W. Calomiris and Harry Mamaysky. How news and its context drive risk and returns around the world. *Journal of Financial Economics*, 133(2):299–336, 2019. `doi:10.1016/j.jfineco.2018.11.009`.

7 Cagatay Catal and Mehmet Nangir. A sentiment classification model based on multiple classifiers. *Applied Soft Computing Journal*, 50:135–141, 2017. `doi:10.1016/j.asoc.2016.11.022`.

8 Alyt Damstra and Mark Boukes. The Economy, the News, and the Public: A Longitudinal Study of the Impact of Economic News on Economic Evaluations and Expectations. *Communication Research*, page 009365021775097, 2018. doi:10.1177/0093650217750971.

9 Ronen Feldman. Techniques and applications for sentiment analysis. *Commun. ACM*, 56(4):82–89, 2013. doi:10.1145/2436256.2436274.

10 Samuel P. Fraiberger. News Sentiment and Cross-Country Fluctuations. *SSRN Electronic Journal*, pages 1–18, 2016. doi:10.2139/ssrn.2730429.

11 Ippei Fujiwara, Yasuo Hirose, and Mototsugu Shintani. Can News Be a Major Source of Aggregate Fluctuations? A Bayesian DSGE Approach. *Journal of Money, Credit and Banking*, 43(1):1–29, 2011. doi:10.1111/j.1538-4616.2010.00363.x.

12 Elena Georgiadou, Spyros Angelopoulos, and Helen Drake. Big data analytics and international negotiations: Sentiment analysis of Brexit negotiating outcomes. *International Journal of Information Management*, 51(November):102048, 2020. doi:10.1016/j.ijinfomgt.2019.102048.

13 Joe Bob Hester and Rhonda Gibson. The economy and second-level agenda setting: A time-series analysis of econom... *Journalism & Mass Communication Quarterly*, 80(1), 2003.

14 Chengyu Huang, Sean Simpson, Daria Ulybina, and Agustin Roitman. News-based Sentiment Indicators. *IMF Working Papers*, 19(273), 2019. doi:10.5089/9781513518374.001.

15 Clayton J. Hutto and Eric Gilbert. VADER: A parsimonious rule-based model for sentiment analysis of social media text. In Eytan Adar, Paul Resnick, Munmun De Choudhury, Bernie Hogan, and Alice H. Oh, editors, *Proceedings of the Eighth International Conference on Weblogs and Social Media, ICWSM 2014, Ann Arbor, Michigan, USA, June 1-4, 2014*. The AAAI Press, 2014. URL: http://www.aaai.org/ocs/index.php/ICWSM/ICWSM14/paper/view/8109.

16 Samina Khalid, Tehmina Khalil, and Shamila Nasreen. A survey of feature selection and feature extraction techniques in machine learning. *Proceedings of 2014 Science and Information Conference, SAI 2014*, pages 372–378, 2014. doi:10.1109/SAI.2014.6918213.

17 Anuj Mahajan, Lipika Dey, and Sk Mirajul Haque. Mining financial news for major events and their impacts on the market. In *Proceedings - 2008 IEEE/WIC/ACM International Conference on Web Intelligence, WI 2008*, pages 423–426, 2008. doi:10.1109/WIIAT.2008.309.

18 Walaa Medhat, Ahmed Hassan, and Hoda Korashy. Sentiment analysis algorithms and applications: A survey. *Ain Shams Engineering Journal*, 5(4):1093–1113, 2014. doi:10.1016/j.asej.2014.04.011.

19 Ayman Mohamed. An Evaluation of Sentiment Analysis and Classification Algorithms for Arabic Textual Data. *International Journal of Computer Applications*, 158(3):29–36, 2017. doi:10.5120/ijca2017912770.

20 Hamed Naderi Semiromi, Stefan Lessmann, and Wiebke Peters. News will tell: Forecasting foreign exchange rates based on news story events in the economy calendar. *North American Journal of Economics and Finance*, 52(December 2018):101181, 2020. doi:10.1016/j.najef.2020.101181.

21 Rickard Nyman, Sujit Kapadia, and David Tuckett. News and narratives in financial systems: exploiting big data for systemic risk assessment. *Journal of Economic Dynamics and Control*, 2021. doi:10.1016/j.jedc.2021.104119.

22 Nataliia Ostapenko. *Macroeconomic Expectations: News Sentiment Analysis.* Working Paper Series. Eesti Pank, 2020. doi:10.23656/25045520/052020/0178.

23 Denilson Alves Pereira. A survey of sentiment analysis in the Portuguese language. *Artificial Intelligence Review*, 54(2):1087–1115, 2020. doi:10.1007/s10462-020-09870-1.

24 Peng Qi, Yuhao Zhang, Yuhui Zhang, Jason Bolton, and Christopher D. Manning. Stanza: A Python natural language processing toolkit for many human languages. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, 2020. URL: https://nlp.stanford.edu/pubs/qi2020stanza.pdf.

25 Sahar Sohangir, Nicholas Petty, and DIngding Wang. Financial Sentiment Lexicon Analysis. In *Proceedings - 12th IEEE International Conference on Semantic Computing, ICSC 2018*, volume 2018-January, pages 286–289. Institute of Electrical and Electronics Engineers Inc., April 2018. doi:10.1109/ICSC.2018.00052.

26 Michael Stanger. A Monthly Indicator of Economic Growth for Low Income Countries. *IMF Working Papers*, 20(13), 2020. doi:10.5089/9781513525853.001.

27 Paul C. Tetlock. Giving content to investor sentiment: The role of media in the stock market. *Journal of Finance*, 62(3):1139–1168, 2007. doi:10.1111/j.1540-6261.2007.01232.x.

28 Leif Anders Thorsrud. Nowcasting Using News Topics. Big Data versus Big Bank. In *Norges Bank Research*, Working papers from Norges Bank. Norges Bank, 2017. doi:10.2139/ssrn.2901450.

29 Yenny Villuendas-Rey, Carmen F. Rey-Benguría, Ángel Ferreira-Santiago, Oscar Camacho-Nieto, and Cornelio Yáñez-Márquez. The Naïve Associative Classifier (NAC): A novel, simple, transparent, and accurate classification model evaluated on financial data. *Neurocomputing*, 265:105–115, 2017. doi:10.1016/j.neucom.2017.03.085.

30 Ritu Yadav, A. Vinay Kumar, and Ashwani Kumar. News-based supervised sentiment analysis for prediction of futures buying behaviour. *IIMB Management Review*, 31(2):157–166, 2019. doi:10.1016/j.iimb.2019.03.006.

31 Shanshan Yi and Xiaofang Liu. Machine learning based customer sentiment analysis for recommending shoppers, shops based on customers' review. *Complex & Intelligent Systems*, 6(3):621–634, 2020. doi:10.1007/s40747-020-00155-2.

Bootstrapping a Data-Set and Model for Question-Answering in Portuguese

Nuno Ramos Carvalho
Rua A 350 2E, 4810-217 Guimararães, Portugal

Alberto Simões
2Ai, School of Technology, IPCA, Barcelos, Portugal

José João Almeida
Centro Algoritmi, Departamento de Informática, University of Minho, Braga, Portugal

Abstract

Question answering systems are mainly concerned with fulfilling an information query written in natural language, given a collection of documents with relevant information. They are key elements in many popular application systems as personal assistants, chat-bots, or even FAQ-based online support systems.

This paper describes an exploratory work carried out to come up with a state-of-the-art model for question-answering tasks, for the Portuguese language, based on deep neural networks. We also describe the automatic construction of a data-set for training and testing the model.

The final model is not trained in any specific topic or context, and is able to handle generic documents, achieving 50% accuracy in the testing data-set. While the results are not exceptional, this work can support further development in the area, as both the data-set and model are publicly available.

2012 ACM Subject Classification Computing methodologies → Discourse, dialogue and pragmatics

Keywords and phrases Portuguese language, question answering, deep learning

Digital Object Identifier 10.4230/OASIcs.SLATE.2021.18

Category Short Paper

Supplementary Material *Software (Source Code)*: `https://github.com/nunorc/qaptnet`

Funding This project was partly funded through the European Regional Development Fund (FEDER), by Portuguese national funds (PIDDAC), through the FCT – Fundação para a Ciência e Tecnologia and FCT/MCTES under the scope of the projects "UIDB/05549/2020" and "UIDB/00319/2020"

1 Introduction

Modern applications use different channels of communication to exchange information with their user base, from businesses to governmental organizations. The increase of information available in digital format has resulted in a high demand for effective, continuous and uninterrupted, information provision services. In this context, tools capable of answering simple questions in real time, without human intervention, are currently in high demand. Some examples of these tools are commonly known as *chatbots* or digital assistants [4].

In the last years there has been a large amount of research in the development of these applications, mostly motivated by the increase of available data, recent advances in Machine Learning techniques, namely on Deep Neural Networks or Word Embeddings, and the increase of computational power readily available. This also spawned some frameworks providing *out-of-the-box* applications that enable a quick, and cheap, implementation of such systems with an acceptable quality.

© Nuno Ramos Carvalho, Alberto Simões, and José João Almeida;
licensed under Creative Commons License CC-BY 4.0

A common key element in these frameworks, and corresponding workflows, is a question answering model. These models are able to find an answer to an information query, described by the user in natural language, from a collection of texts. Deep learning neural networks [5] currently achieve some of the best results for this specific task [3, 2, 9].

This paper introduces an exploratory work carried out to build a model for performing question answering tasks for the Portuguese language. Although the current literature is rich on models and techniques, some languages (in particular, the Portuguese language) lack state-of-the-art implementations of such models. Therefore, the guiding research question for this work is defined as:

> Can transfer learning be explored to train a model, for the Portuguese language, capable of providing satisfying results for finding an answer to a information query writen in natural language from a collection of texts?

Given the lack of computational power and enough data to train a model from scratch, and also in order to reuse information obtained from previously training in other languages, we start the work with the parameters from a pre-trained model. This means that we take advantage of the parameters exploration during previous training, and fine tune the model for the Portuguese language. Transfer learning encompasses the idea of not starting the model training stage from scratch, bootstrapping the model using parameters from previously training steps [7]. This usually allows the training process to achieve better results with less data or fewer training steps.

The goal of the model is: given a snippet of text and an information query written in natural language, find the answer to the query in the text. This snippet of text is usually referred to as a context, and it is assumed that the given context contains the answer to the query. This enables the implementation of *chatbots* (or similar applications) that can answer questions systematically without being explicitly programmed to do so, and independently of the subject or topic at hand.

This paper is organized as follows: Section 2 introduces the data-set created for training; Section 3 discusses the architecture of the used model; Section 4 depicts the model training and its validation process, illustrating the results with some examples; Section 5 follows with an analysis of the obtained results. The document concludes with some final remarks and possible directions for future work.

2 The data-set

The Stanford Question Answering data-set (SQuAD) [8] contains a collection of around 400 generic articles from Wikipedia and, for each article, a set of questions and corresponding answers written in natural language. The collection is organized as a sequence of paragraphs (contexts) and, for each one, a set of questions and their answers is available. The answers can be gathered from the context paragraph.

Given the original data-set is only available in English, and there is no similar data-set available for the Portuguese language[1], we bootstrapped a Portuguese data-set using Machine Translation.

The original SQuAD data-set was automatically translated using the Google Translate API. Although the translation is not at human level, given that the paragraphs, questions, and answers are relatively small (just a few words in the case of the answers, a single sentence

[1] Note that there are other translations from SQuAD that are based on our work. Nevertheless, this article is relevant as our resource is being used for other works, and because implicitly we are evaluating Machine Translation systems.

for the questions, and a few sentences in the case of the contexts), most of the translations are acceptable. The main problem, as we will discuss later, is the lack of coherence between different translations, depending on the sequence context.

Given that the goal of the model is to find patterns between the structure of the text and the formulated questions, grammatical accuracy, exceptions, inconsistencies and similar shortcomings usually associated with automatic translations should not have a big influence in the results.

The current version of the translated data-set is available online[2]. The data-set is divided in two parts: *train* and *dev*, that have the same format, and can be used for training and testing respectively.

3 The Model

Bidirectional Encoder Representations from Transformers (BERT) is a state-of-the-art model for many NLP tasks as, for instance, document classification, named entity recognition, or question-answering [3]. The model is implemented using a deep neural network with a set of different layers. It can be seen as the complement of two major networks: (i) a pre-trained transformer that acts as a language model, and (ii) a network with different outputs depending on the task at hand.

In the context of question-answering tasks, the input to the model is composed by the context (the snippet of text that contains the answer) and the information query (the question). The output of the model is the span of text, from the context, that contains the answer. This is represented by the starting and ending tokens of the span of the answer in the context:

$$\text{qaptnet} :: (\text{Context}, \text{Question}) \rightarrow (\text{Start}, \text{End})$$

4 Training and Validation

Google provides different flavors of pre-trained parameters for the BERT model, both as monolingual or multilingual, and with different depth size (with different amounts of hidden layers). To train QAPTNET we used one of the multilingual models, case sensitive and 12 hidden layers, as a bootstrap model. The model was then trained using the training data-set, for 2 epochs, using a batch size of 8. The optimizer used was the Adam algorithm with a fixed weight decay, and a learning rate of 3×10^{-5} [6]. Implementation, training and testing of the model was performed done using the PyTorch-Transformers Python package[3].

The model was then tested with the validation data-set, scoring around 50% accuracy, i.e. it was able to correctly find the answer for half the questions. The final version of the model is available publicly[4], including a Python package companion that helps using it. The main reason for this result is that the model is not able to completely capture a pattern between the question and the corresponding answer. This may be due to some factors including: not enough data for training, or the model not being complex enough to capture the intended pattern.

The following examples (in Portuguese) illustrate the use of the final model. For example, given the following `context`:

[2] Available at `https://github.com/nunorc/squad-v1.1-pt`.
[3] Available at `https://pytorch.org/hub/huggingface_pytorch-transformers/`.
[4] Available at `https://github.com/nunorc/qaptnet`.

> *Arquitetonicamente, a escola tem um caráter católico. No topo da cúpula de ouro do edifício principal é uma estátua de ouro da Virgem Maria. Imediatamente em frente ao edifício principal e de frente para ele, é uma estátua de cobre de Cristo com os braços erguidos com a lenda Venite Ad Me Omnes. Ao lado do edifício principal é a Basílica do Sagrado Coração. Imediatamente atrás da basílica é a Gruta, um lugar mariano de oração e reflexão. É uma réplica da gruta em Lourdes, na França, onde a Virgem Maria supostamente apareceu a Santa Bernadette Soubirous em 1858. No final da unidade principal (e em uma linha direta que liga através de 3 estátuas e da Cúpula de Ouro), é um estátua de pedra simples e moderna de Maria.*

One could pose the following `question`:

> *A quem a Virgem Maria supostamente apareceu em 1858 em Lourdes, na França?*

The model result is:

```
>>> qaptnet.query(context = context, question = question)
'Santa Bernadette Soubirous'
```

This example uses the model Python package companion to simplify the use of the model, a complete example on how to use this package is available in [1].

5 Data-Set and Results

One of the main challenges of this work is the data-set scale. In fact, it was automatically translated and not manually corrected. This means that some translations are not at a trained human translator level, and also that some translations were performed differently whether the string appeared with context (in the context paragraph) or without context (in the short answer). A simple example if the answer to the question shown in the previous section. While "Santa Bernadette Soubirous" is the correct answer, it will be accounted as wrong, as when translating the answer the system used "Saint Bernadette Soubirous".

As a simple exercise to evaluate the quality of the data-set, for each answer we checked whether it is present in the context. Performing this analysis in the training data-set, we found up that 27 710 answers were not present in the context, from a total amount of 87 599 answers (about 30% of the training set). Doing the same analysis in the validation data-set, 11 081 answers of the total amount of 34 726 was not present in the context paragraph (about 31%).

This analysis did not account for other problems, like the lack of quality of the translation or even the upper-case versus lower-case comparison of the answers with the context (as we had a case-sensitive model).

Nevertheless, after training, a 50% accuracy was obtained. If we account the 30% of problematic entries, that the system would never guess, from the total amount of 34 736 entries, only 23 645 entries could be correct. Doing the proportion, the system would get up to 70% accuracy accounting only the correct parts of the validation set.

6 Conclusion

This paper introduces QAPTNET a model for performing question answering tasks, i.e. given a context and a question find the span of text in the context that answers the question, for the Portuguese language. The final version of the model achieves around 50% accuracy on the development part of the data-set, which is interesting enough for a first exploratory attempt, given the complexity of the task, and the initial lack of data to train the model. The other major outcome of this work is the data-set, the first of its' kind, in Portuguese, publicly available, that other researches can use to explore different models, and be improved by the community.

Returning to the research question guiding this work, the final model is able to find a satisfying number of answers for question, context pairs written in Portuguese. The transfer learning approach was the main enabler for achieving these results with such a small number of steps and shortcomings in the data-set. Of course this is still a work in progress, some trends for future work include:

- Correcting the data-set, validating the translations and the existence of the answer in the contexts;
- Fine-tuning model the hyper-parameters: some parameters used during training can be tuned to achieve better results, e.g. learning rate and batch size, also increasing the model complexity might help capturing patterns in the data;
- The analysis of a similar approach using the 2.0 version of SQuAD data-set.

References

1 N.R. Carvalho, 2019 (last accessed: 28- 08-2019). URL: `https://github.com/nunorc/qaptnet`.

2 Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc V Le, and Ruslan Salakhutdinov. Transformer-xl: Attentive language models beyond a fixed-length context. *arXiv preprint*, 2019. `arXiv:1901.02860`.

3 Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint*, 2018. `arXiv:1810.04805`.

4 SurveyMonkey Audience Drift and Myclever Salesforce. The 2018 state of chatbots report. how chatbots are reshaping online experiences, 2019.

5 Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

6 Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint*, 2014. `arXiv:1412.6980`.

7 Emilio Soria Olivas. *Handbook of Research on Machine Learning Applications and Trends: Algorithms, Methods, and Techniques: Algorithms, Methods, and Techniques*. IGI Global, 2009.

8 Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100,000+ questions for machine comprehension of text. *arXiv preprint*, 2016. `arXiv:1606.05250`.

9 Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le. Xlnet: Generalized autoregressive pretraining for language understanding. In *Advances in neural information processing systems*, pages 5754–5764, 2019.

Development of Self-Diagnosis Tests System Using a DSL for Creating New Test Suites for Integration in a Cyber-Physical System

Ricardo B. Pereira ✉
Department of Informatics, University of Minho, Braga, Portugal

José C. Ramalho ✉
Centro Algoritmi (CAlg-CTC), Department of Informatics, University of Minho, Braga, Portugal

Miguel A. Brito ✉
Centro Algoritmi, Department of Information Systems, University of Minho, Guimarães, Portugal

Abstract

Testing Cyber-physical systems (CPS) requires highly qualified engineers to design the tests since its computational part is programmed in low-level languages. The origin of this work arises from the need to find a solution that optimizes this problem and allows abstracting the current methods so that the tests can be created and executed more efficiently. We intend to do this by creating a self-diagnosis tests system that allows us to automate some of the current processes in the creation and execution of test suites. The work presented here addresses the problem by creating a new self-diagnosis tests system that will guarantee the reliability and integrity of the CPS. In detail, this paper begins by exposing a study on the current state of the art of test automation, Keyword-driven Testing (KDT) methodology and Domain-specific Languages (DSL). A new modular and extensible architecture is proposed for self-diagnosis tests systems based on two main concepts: the creation of a DSL combined with the use of the KDT methodology, as well as a methodology to extend it and integrate it into a CPS. A new self-diagnosis tests system has been proposed that applies the proposed architecture proving that it is possible to carry out the self-diagnosis in real-time of the CPS and allowing the integration of any type of test. To validate the implementation of the system, 28 test cases were carried out to cover all its functionalities. The results show that all test cases passed and, therefore, the system meets all the proposed objectives.

2012 ACM Subject Classification Software and its engineering → Parsers; Computer systems organization → Embedded and cyber-physical systems; Information systems → RESTful web services

Keywords and phrases Web Application, DSL, Self-diagnosis, Test automation, Cyber-physical systems

Digital Object Identifier 10.4230/OASIcs.SLATE.2021.19

1 Introduction

Today, the production of many industrial companies is supported by cyber-physical systems (CPS) and, therefore, they must be able to obtain the maximum performance of these systems. For this, it is necessary that these systems remain reliable and can guarantee their functionality [9]. However, to ensure that these systems work correctly, a diagnosis of them is necessary regularly. Testing CPS requires highly qualified engineers to design the tests since its computational part is programmed in low-level languages. The origin of this work arises from the need to find a solution that optimizes this problem and allows abstracting the current methods so that the tests can be created and executed more efficiently. We intend to do this by creating a self-diagnosis tests system that allows us to automate some of the current processes in the creation and execution of test suites.

© Ricardo B. Pereira, José C. Ramalho, and Miguel A. Brito;
licensed under Creative Commons License CC-BY 4.0

The work presented here addresses the problem by creating a new self-diagnosis tests system that will guarantee the reliability and integrity of the CPS. In detail, this paper begins by exposing a study on the current state of the art of test automation, Keyword-driven Testing (KDT) methodology and Domain-specific Languages (DSL). A new modular and extensible architecture is proposed for self-diagnosis tests systems based on two main concepts: the creation of a DSL combined with the use of the KDT methodology, as well as a methodology to extend it and integrate it into a CPS. A new system of self-diagnosis tests system has been proposed that applies the proposed architecture and aims to prove that it is possible to perform the self-diagnosis in real-time of the CPS and allow the integration of any type of test through the combination of a DSL with the KDT methodology. Some test cases were also carried out to validate the implemented solution.

Section 2 analyzes the state of the art in test automation, KDT methodology and DSL. Section 3 describes the structure and architecture of the system. Section 4 describes the implementation of the system. Finally, Section 5 concludes and identifies future work.

2 State of the art

In this section, a review of the state of the art in test automation will be presented in Section 2.1. In Section 2.2, KDT methodolody is presented as well as the advantages and disadvantages of using it. In Section 2.3, a brief introduction is made to the concept of DSL and, more specifically, how to apply this concept with the Another Tool for Language Recognition (ANTLR).

2.1 Test Automation

The importance of testing automation is directly related to the quality of the final product. The execution of all functional tests before delivery guarantees the lowest incidence of errors in the post-delivery of the final product. As such, software developers/creators are required that their projects maintain a certain quality standard during all phases of development until the launch of a new product. Therefore, testing at the end of each stage no longer works in a professional environment. This is because the occurrence/discovery of unforeseen obstacles can significantly delay the development of the software. In recent years, it has been found that the software development market has increased its competitiveness, due to the modernization of the technologies involved and due to the maturity of the capacity to develop software. Thus, the range of information technology solutions, to meet the needs of consumer organizations, has increased considerably, which ends up making it difficult for users to choose when purchasing a product. In this competitive scenario, consumer organizations, when opting for software, are increasingly relying on quality criteria. One of the pillars for ensuring this quality of the software product is the testing process [1].

In the current software market, the concern for creating quality and error-free products has led companies to look for models and processes that guarantee quality to satisfy the needs of their customers. Unsuccessful projects, with expired deadlines and defective products, lead to customer dissatisfaction, high maintenance costs and compromise the company's image. The main objective of a software test is to define the implementation of this software that meets all the specifications and expectations defined and expected by the customer, that is, the objective is to "verify" if what was specified in the requirements phase is what really was developed. When verifying that the implemented software meets all specifications and expectations defined and expected by the customer, it also looks for errors in the software. The software test must be seen as a part of its quality process.

Test automation is not limited to just performing the tests but above all being aware of when and where the tests need to be carried out, thus leaving the test team more time to plan more effective tests with better quality accuracy instead of worrying about scheduling them. Thus, automation results in the mechanization of the entire process of monitoring and managing the needs for testing and evaluation associated with software development [3].

2.2 Keyword-Driven Testing

KDT is a type of functional automation testing methodology that is also known as table-oriented testing or action-based testing. In KDT, we use a table format, usually a spreadsheet, to define keywords or action words that represent the content of the tests in a simple way. But it also allows the use of a keyword to represent part of the test case and in this way make the creation of the test case simpler, since we can reuse the keywords and the whole process they represent in different test cases. It allows novice or non-technical users to write tests more abstractly and it has a high degree of reusability. Industrial control software has been having an enormous increase in complexity as technology has developed and requires a systematic testing approach to enable efficient and effective testing in the event of changes. KDT has been proving that it is a valuable test method to support these test requirements [16]. Recent results from other researchers have shown that the design of the KDT test is complex with several levels of abstraction and that this design favours reuse, which has the potential to reduce necessary changes during evolution [6]. Besides, keywords change at a relatively low rate, indicating that after creating a keyword, only localized and refined changes are made. However, the same results also showed that KDT techniques require tools to support keyword selection, refactoring, and test repair [4].

2.2.1 Advantages

- Fast execution of test cases;
- Software testing in less time;
- All manual testing problems are solved by automated testing;
- Repeating test cases are handled in an easy way.

2.2.2 Disadvantages

- Sometimes some knowledge of programming and skill is needed to use these tools;
- Maintenance is a complicated task and can be expensive;

2.3 Domain-Specific Language

DSL is a language meant to be used in the context of a particular domain. A domain could be a business context or an application context. A DSL does not attempt to please all. Instead, it is created for a limited sphere of applicability and use, but it's powerful enough to represent and address the problems and solutions in that sphere [5]. A DSL can be used to generate source code from a keyword. However, code generation from a DSL is not considered mandatory, as its primary purpose is knowledge. However, when it is used, code generation is a serious advantage in engineering. DSL will never be a solution to all software engineering problems [10], but their application is currently unduly limited by the lack of knowledge available to DSL developers, so further exploration of this area is needed [7]. Other researchers used DSL in CPS and left their testimony of how the specification language hides the details of the implementation. The specifications are automatically enriched with the implementation through reusable mapping rules. These rules are implemented by the developers and specify the execution order of the modules and how the input/output variables are implemented [2]. This allows the reuse of software components (e.g. modules or classes) and improves software productivity and quality [8].

2.4 ANTLR

ANTLR is a parser generator, a tool that helps you to create parsers [12]. A parser takes a piece of text and transforms it into an organized structure, a parse tree, also known as an Abstract Syntax Tree (AST) [15]. AST is like a story describing the content of the code, or its logical representation, created by putting together the various pieces [13]. Figure 1 shows the parsing process.

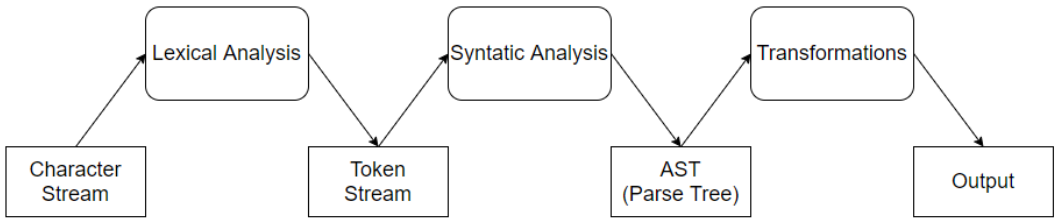


Figure 1 Block diagram of a standard Language Processor.

The Parsing process shown in Figure 1 goes through three major phases explained below:

- Lexical Analysis:
  - It is performed by a component usually called Lexer, Lexical Analyser or Tokenizer;
  - The Lexer reads and divides the input (character or byte stream) into tokens applying lexical rules;
  - Lexical rules are defined using regular expressions and aim to identify terminal symbols and specify tokens;
  - In the end, the Lexer generates a token stream as output.
  - Figure 2 shows the illustration of this process.
- Syntactic Analysis:
  - It is performed by a component usually called Parser, Syntatic Analyser or Grammar;
  - The parser gives the token stream a structure by checking token order against structural rules;
  - These Structural rules define the order and structure of token combination;
  - In the end, the Parser generates a parse tree as output.
  - Figure 3 shows the illustration of this process.
- Transformations:
  - It is performed by a component usually called Transformer or Walker and it follows the pattern Visitor or Listener;
  - The Transformer traverses the parse tree in order to produce some output;
  - The traversal defines an action for each node of the parse tree;
  - The action can output text (string) or any other complex object.
  - Figure 4 shows the illustration of this process.

ANTLR is a parser generator that uses ALL(*). It parses the input dynamically at runtime and uses a top-down parser left to right by constructing a Leftmost derivation of the input and looking any number of ahead tokens when selecting among alternative rules [14]. The Visitor pattern let us decide how to traverse the tree and wich nodes we will visit. It also allows us to define how many times we visit a node [11].

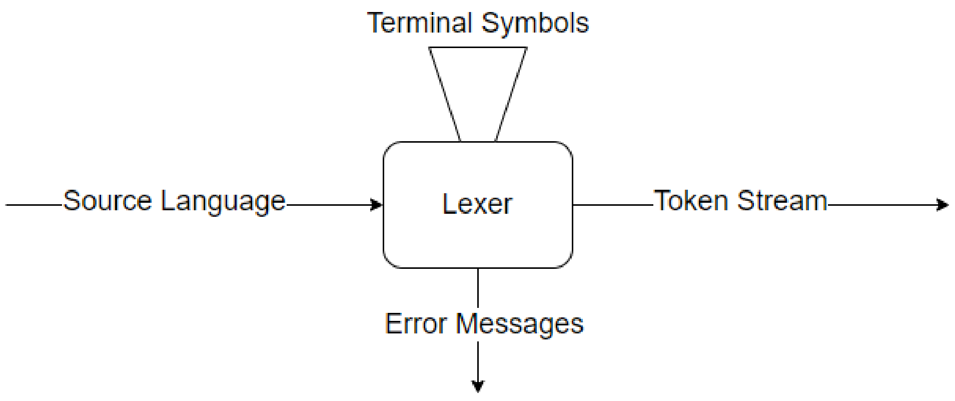


Figure 2 Block diagram of a Lexical Analyzer.

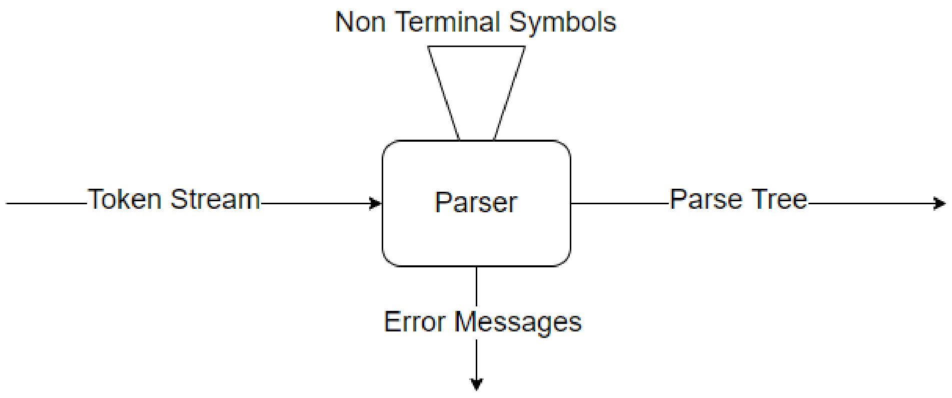


Figure 3 Block diagram of a Syntactic Analyzer.

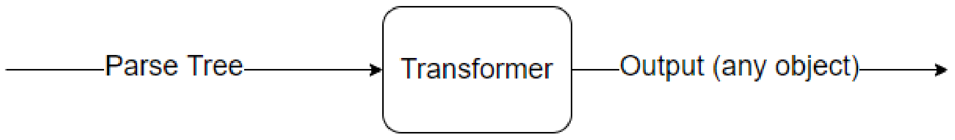


Figure 4 Block diagram of a Transformer.

3 Architecture

As the architecture incorporates several diversified components, its modelling was divided into two phases. In the first phase, the part of the architecture that refers to the system to be developed and that includes the management and configuration of the tests are explained. In the second phase, the general architecture of the CPS is presented.

3.1 Self-diagnosis Tests System Architecture

To obtain a complete understanding of this architecture, it is necessary to understand the 3 tiers that are present, Frontend, Backend and Database. We can see the architecture in Figure 5, shown below:

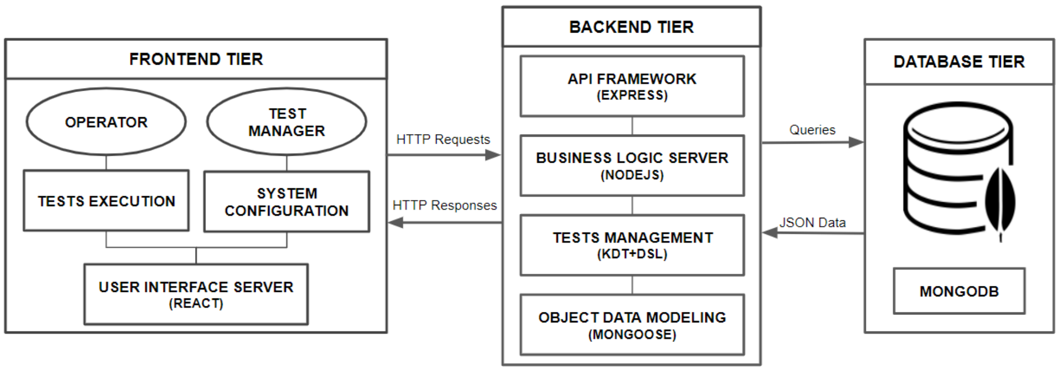


Figure 5 Proposed architecture for self-diagnosis tests system.

3.1.1 Frontend

In this tier, we have two first elements, `OPERATOR` and `TEST MANAGER`, which represent the two types of users that the system has. Therefore, according to the permissions of each one, this tier makes available to each user the respective interface that will give access to the realization of the functions of each one in the system. The two elements below in the tier, `TESTS EXECUTION` and `SYSTEM CONFIGURATION`, represent the different interfaces that each user will have access to. In this case, the `OPERATOR` type user will have access to the system `TESTS EXECUTION` mode and the `TEST MANAGER` type user will have access to the `SYSTEM CONFIGURATION` mode. The last element of this tier, `USER INTERFACE SERVER`, represents the logic of the Client. It is in charge of implementing any logic that exists in this tier, such as, for example, providing an adequate interface for the type of user that must comply with it or even the manipulation of data in the formation of web pages. It is also this server that establishes the connection to the Backend tier, making HTTP requests to request data or actions, receiving and validating data that arrives through HTTP responses.

3.1.2 Backend

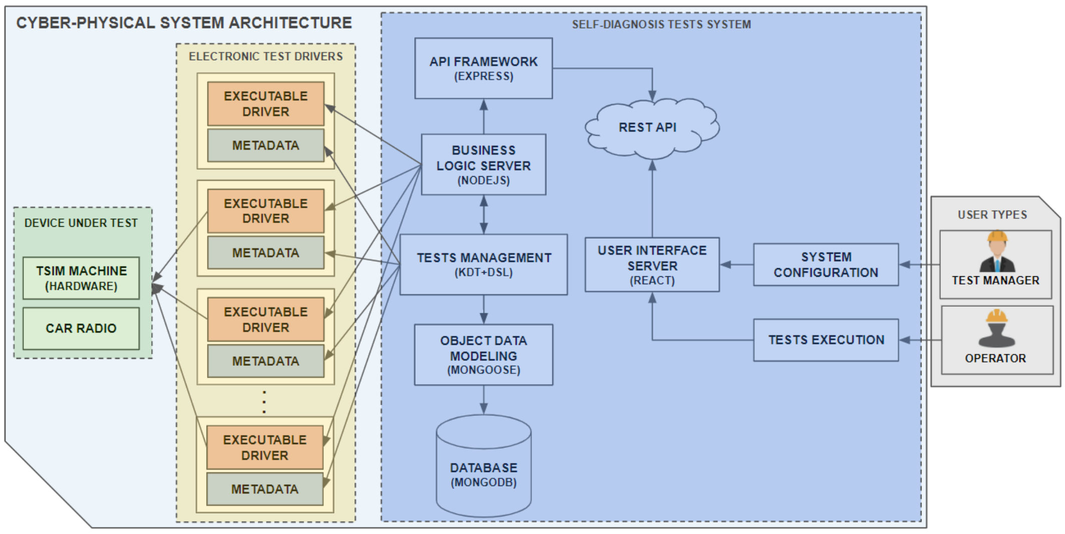
The Backend tier, unlike what was done in the Frontend tier, will be analyzed from the bottom up, as it will be understood more intuitively. In this tier, we start by looking at two elements in parallel. The `OBJECT DATA MODELING` element represents the module responsible for establishing the connection between this tier and the Database tier, that is, it is this module that performs the queries and receives data from the database. Element TESTS MANAGEMENT is responsible for the acquisition and management of the primitive tests of the system and the configuration of new test suites for the system, using the KDT methodology and a DSL. Above, we see the BUSINESS LOGIC SERVER element that represents the Server that implements all the logic of this tier. This component is responsible for executing the tests and for the internal organization of all other components of this tier. Manages all data arriving at the system, guaranteeing its integrity, and also provides the routes or services through which this tier responds to Clients requests. The last element of this tier, API FRAMEWORK, is responsible for building and making the REST API available to the Client. This element implements the routes that are created in the BUSINESS LOGIC SERVER element and, in this way, the Client can make HTTP requests to the Server.

3.1.3 Database

Finally, it remains only to present and explain the Database tier, which is also the simplest tier of this architecture. It consists of the system database, which is a document database that stores documents in JSON. All data sent to the Backend tier, via OBJECT DATA MODELING, is in JSON, which is an advantage because all data processing and manipulation in the system is always done in this format.

3.2 General Architecture for Cyber-Physical System

In this section, the final CPS architecture is presented and explained, where we integrate all its components with the self-diagnosis tests system. This architecture enables the CPS to diagnose itself and, thus, be able to identify the failures in case of any internal error. The architecture, being the final abstraction of the system, can be seen in Figure 6.



■ **Figure 6** Proposed architecture for a self-diagnosis test system integrated with the CPS.

In this architecture, we can easily identify 4 component groups in which three of them will form an integral part of the CPS: Devices Under Test, Electronic Test Drivers and the Self-Diagnosis Tests System. The last group will be an important intervenient, but it is not an integral part of the CPS, the User Types. Each of these groups will be explained in detail, as each has its particularities.

The Devices Under Test group contains, as the name implies, the devices that can be subjected to tests which are the car radios and the machine itself. The elements `CAR RADIO` and `TSIM MACHINE` represent the two types of devices, the car radio and the machine, respectively. The Electronic Test Drivers group is responsible for the primitive tests of the system, which in this case will be mostly electronic tests, but which can be any type of test as long as they respect the same integration format. Each element of this group must respect the following format:

- `EXECUTABLE DRIVER` – Provides an executable driver file to run that will contain several primitive tests that can be run and test the Devices Under Test;
- `METADATA` – Provides a metadata file that contains all the information about the tests that the driver can perform.

The Self-Diagnosis Tests System group is where the system developed in this work is represented, which will allow users to manage and execute the system tests. This system will be fed with primitive tests from the group of Electronic Test Drivers. The `TESTS MANAGEMENT` element is responsible for loading all the metadata of the primitive tests, available in the `METADATA` files of the Electronic Test Drivers group, and managing them so that they are saved in the system database and are available for execution.

The link element with the system database is the `OBJECT DATA MODELING` that will make the connection and handle queries and transactions to the database, which is the `DATABASE` element.

This test management is done through the KDT methodology, and the configuration of new test suites made through the developed DSL. The tests will be performed by the `BUSINESS LOGIC SERVER` element, which will receive the execution orders from the end-user and proceed with the executions. The way to do this is to execute the drivers that are available as executable files. This Server will know which tests are available to execute on each driver since the `TESTS MANAGEMENT` element has already collected the metadata of all drivers and at that moment made available for execution, all the tests contained therein.

This entire organization is orchestrated by the Server, which is responsible for the logic of the system and is represented by the element `BUSINESS LOGIC SERVER`. This Server not only controls all the data and logic of the system but also defines the routes and types of requests that can be made by the Client-side. It defines the services that will be available and this is called an API. The `API FRAMEWORK` element is responsible for creating and providing a REST API for any client to access, but obviously with the appropriate permissions, also defined by the `BUSINESS LOGIC SERVER`.

In this system architecture, `USER INTERFACE SERVER` represents the Client-side, that is, it is the server responsible for creating the web interface for end-users. It makes HTTP requests specifying the services, through routes, that it wants to access, to obtain the data it needs for its pages. Two types of interfaces are available, the execution interface, represented by the `TESTS EXECUTION` element, and the test and configuration management interface, represented by the `SYSTEM CONFIGURATION` element. Each of these interfaces will have its correspondent as a user, which brings us to the last group specified in the architecture, the User Types.

This group is represented by the `USER TYPES` element and represents the different types of users of the final system. The first and most basic type of user is the `OPERATOR`, that is, the industrial operator who is working and commanding the CPS and performs only the tests or test packages of the system. The second type of user, already more sophisticated, is the `TEST MANAGER`, who is someone with the responsibility of managing the entire system, using the appropriate interface for that.

4 Implementation

This section describes the implementation of the system and its validation. Thus, Section 4.1 explains each collection of data maintained in our database. Section 4.2 describes the Backend tier where the system logic is, including the management of the system data and the configuration and execution of the tests. Section 4.3 describes the Frontend tier that contains the user interface and the different features available for each type of user. Finally, Section 4.4 presents the results obtained from the validation performed to ensure the correct functioning of the system.

4.1 Database

For the database, MongoDB was used, which is a document database, that is, it stores the data in the form of JSON documents. According to the data that the system needs, 5 collections of data have been identified to be stored in the database: Configurations, Tests, Packages, Reports and Schedules.

The configuration collection contains attributes about some configurations that may differ from machine to machine and are necessary to ensure the correct functioning of the system. The tests collection stores all metadata for the system's primitive tests. This metadata is provided by those who create and make the primitive tests available, so they are only imported into the system database and updated whenever there are changes. The packages collection stores all metadata for the new test suites that are created in the system from the primitive tests. The reports collection stores all reports of execution of primitive tests or test packages in the system. The schedules collection stores all primitive test executions or test suite executions scheduled for a specific time by the user.

After specifying the data to be saved in each collection of the system's database, the next section will explain how the system interacts with the database, through queries, to obtain the data for its operation.

4.2 Backend

The Backend is the system tier responsible for managing the database and making the data available to Frontend. Therefore, framed in the MVC architecture, it is the Controller of the system and establishes the connection between the database and the user interfaces, thus guaranteeing the integrity of the data, not allowing other components to access or change them.

The technology used to develop this server was Node.js combined with Framework Express. This server is organized so that there is a division of the code according to its function, that is, instead of all the code being in one file, it was divided into different files and directories according to its purpose on the server. This will allow the reuse and modularity of the developed code, which will also facilitate its maintenance and understanding in the future.

Thus, the server structure is as follows:

- Models: Here are the models that correspond to the collections saved in the database. Each model contains the attributes corresponding to its collection and performs validations related to data types to ensure that wrong data types are not inserted into the database;
- Controllers: Here are the files responsible for performing all system operations, such as database queries, executing primitive tests and test suites, and creating new test suites using the DSL defined;

- Grammar: Corresponds to the DSL developed for the system, where is the grammar, composed by a Lexer and a Parser, and the Visitor that generates the code for the new test suites;
- Routes: Here is the file that routes the requests, from the client, that is, from the user interfaces to the controllers, according to the URL request. As soon as the requested operations are completed, sends the requested data to the client.

Each of these elements mentioned above, has a fundamental role in the Server's logic, so each of them will be explained in the next subsections individually.

4.2.1 DSL

The DSL developed aims to enable the creation of new test suites, from the primitive tests available in the system, with rules and logic applied. This will allow the test suites to be optimized to execute in the shortest possible time and may shorten certain executions whenever the suite specifies it. The language was created from the identification of terminal symbols, that is, the symbols that would be identified by Lexer. After this step, the Parser was created, where the rules of logic and sentence construction of the grammar are specified.

The terminal symbols of the DSL are shown in Table 1, where the respective descriptions are also shown.

Table 1 DSL Symbols Description.

Symbol	Description
keyword	Catches the keywords in the script
->	Catches the "next" symbol, which means that after that symbol the next block to be executed arrives
(	Catches the opening parenthesis
)	Catches the closing parenthesis
?	Catches the conditional expressions from the script
:	Catches the next block of code to be executed when a condition is false
&	Catches the logical operator that means intersection
|	Catches the logical operator that means union
;	Catches the end of the script

The Lexer structure is shown below in Listing 1:

Listing 1 Grammar Lexer.

```
lexer grammar TestLexer;

NEXT        :    '->'     ;
AND         :    '&'      ;
OR          :    '|'      ;

IF          :    '?'      ;
ELSE        :    ':'      ;

RPAREN      :    ')'      ;
LPAREN      :    '('      ;

END         :    ';'      ;
```

```
KEYWORD      :    ([A-Za-z]+([/ _-][A-Za-z]+)*)
             ;

WS
        : [ \r\n\t] -> skip
        ;
```

The structure of the Lexer is quite simple, starting with its identification and then just specifying all terminal symbols that must be recognized. The way these symbols are specified is through regular expressions, that is, for each symbol the regular expression that represents it is defined, however, always taking care that this definition does not include unexpected elements and, therefore, is not ambiguous.

The symbols we see in this grammar are very intuitive and this is also one of its advantages, as it will be easy for the end-user to understand, which is one of the objectives. The only symbol that gives rise to any further explanation is the `KEYWORD` symbol. This symbol must recognize all the names of the primitive tests introduced in the script and, therefore, its regular expression includes isolated words or also the composition of several words, thus giving the user some freedom to be more expressive in the choice of keywords since this it is also the purpose of the KDT methodology applied in the system.

After defining the terminal symbols and the Lexer specification, it is time to specify the sentence construction rules with these symbols and this is done in the Parser, which is shown below in Listing 2:

Listing 2 Grammar Parser.

```
parser grammar TestParser;

options {
    tokenVocab=TestLexer;
}

test
    : statement END
    ;

statement
    : condition                                 #Conditional
    | seq                                       #Sequence
    ;

condition
    : expr IF statement ELSE statement          #IfElse
    | expr IF statement                         #If
    ;

seq
    : KEYWORD (NEXT statement)*
    ;

expr
    : LPAREN KEYWORD (AND KEYWORD)* RPAREN      #And
    | LPAREN KEYWORD (OR KEYWORD)* RPAREN       #Or
    ;
```

The Parser also starts with its identification, following the reference for the Lexer that it provides the symbols to be able to know which are the terminal symbols. After these two steps, the sentences of the grammar are specified and here there is no more than a specification of the sequences that the elements of the language can follow. We can see, for example, in the element `statement` two possibilities. One possible `statement` is the `condition` that represents a conditional expression and the other possibility is a `seq` that represents a tests sequence. The most important part of the Parser to retain is the elements that come at the end of the lines for each possibility determined at the beginning of words by a `#`. This allows the Visitor to know the possible paths in the parsing tree that this Parser will generate.

So that this grammar can now be used by the system and generate the parsing tree that will be interpreted by the Visitor, it is still necessary to find a way to use it in the system. Since ANTLR offers the transformation of these grammars for several known programming languages, we will proceed to transform the grammar into JavaScript and include the code directly in the system. For this, it is necessary to execute the following command:

```
$   antlr4 -Dlanguage=JavaScript Lexer.g4 Parser.g4 -no-listener -visitor
```

In this command, we specify the Lexer and Parser to be transformed and we also specify that we do not want the generation of a Listener because, by default, it generates the Listener. Finally, we specify the generation of a Visitor because, by default, it does not generate the Visitor. After executing this command, several files will be generated, among which, the Visitor that will be the most important in the next steps, as this is where the code to be generated for the new test suites will be specified.

We can see below, in Listing 3, an example of a Visitor function:

Listing 3 Grammar Visitor.

```
TestParserVisitor.prototype.visitAnd = function (ctx) {
    this.auxOp = 0;
    for (let i = 0; i < ctx.KEYWORD().length; i++) {
        this.auxList.push(ctx.KEYWORD(i));
    }
    return "";
};
```

The Visitor's strategy developed is to go through the code script through the elements specified in the Parser and each element generate the corresponding code. The generated code, within the Visitor, is nothing more than a string that is incremented and filled up to the end of the parsing tree. All keywords are also being saved in a list so that the list and the string containing the generated script are returned at the end. The list of keywords is necessary because after generating this code it will be necessary to match the keywords with the primitive tests but this is a process already done in the packages controller.

4.3 Frontend

The frontend is the system tier responsible for creating and managing graphical interfaces for end-users. In this case, there are two types of users in the system, and it is important to understand well the limits on what each one should be allowed to do or not do. The first type of user, more basic, will only have access to the execution of primitive tests and test suites. The second type of user, already responsible for managing the system and also the test suites for it, has access to all other features. The technology used to develop this tier was React, as it will allow us to create dynamic interfaces, with components managing their state and the possibility to compose the components themselves. This allows the code to be modularized and, in the future, it will be easier to understand the code.

4.3.1 Components

As mentioned, the development of components in React becomes an asset, but to master the use of technology it is necessary to understand the fundamentals and the way the components interact with each other. The three concepts that we highlight are the following:

- State: The state of a component is mutable and can be changed by the component itself, due to the actions performed by the user. Information stored in a component's state can be accessed as attributes of the component, such as "this.state.name";
- Props: Props are state information from a parent component to a child component, so the child cannot directly change the props but can access them in the same way as the parent, such as "this.props.name". They are generally used to determine some properties of the child component when it is created;
- Events: Events are how the child component should inform the parent component of changes that have occurred. This is how a child component can change the state of the parent component, through events that will inform the parent component so that it updates its state.

Thus, to understand how these concepts apply in practice and make the most of the use of React components, we can see below, in Figure 7, an illustration of how these concepts are related:

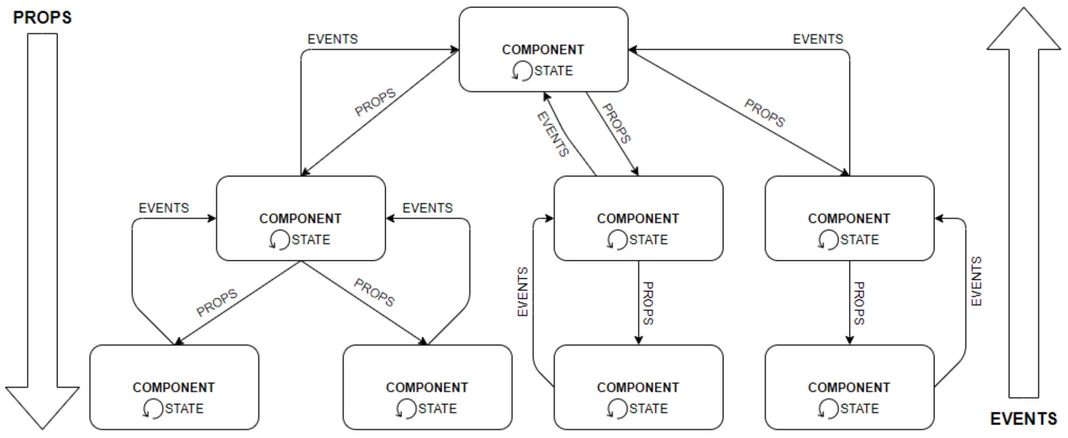


Figure 7 Interactions between reaction components.

4.3.2 Obtaining API data

Another important aspect for this part of the system to work as planned is to obtain the data that is managed by the Backend tier. For the graphical interfaces built to be as optimized as possible and quick in obtaining data, so that the user does not have to wait long to load the pages, the data must be obtained in the best way. And here the decision made was that the parent components of each page make the data requests to the API at the time of its creation. With this, what happens on the system pages is that whenever the user changes the page or enters a new page, the data is requested and loaded. This will allow the actions taken by the user on the components belonging to these pages to be carried out much more quickly, giving the user the perception that nothing has happened when real events and state changes have already occurred witch allows the page to become dynamic with desired speed.

The way to obtain the data is through HTTP requests, explained previously, therefore, to make the code clearer, a dedicated file was created for request methods. This file contains the base URL of the Data API and all methods add only the route and sub-route as needed. We can see below, in Listing 4, an example of a method of obtaining data by making an HTTP request to the data API:

Listing 4 Example of request to obtain API data.

```
export const getTests = async () => {
    try {
        const response = await axios.get(`${url}/tests`);
        return response.data;
    } catch (error) {
        const statusCode = error.response ?
            error.response.status :
            500;
        throw new Error(statusCode.toString());
    }
};
```

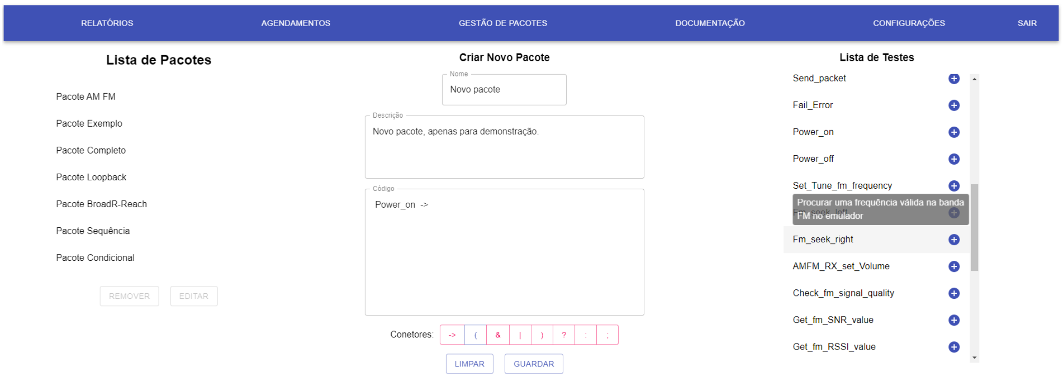
In this example, we can see how HTTP requests are made to the API. These requests are made through the imported module "Axios" since the technology does not provide this functionality natively. Another important feature that we see in this example is the use of the keyword "await", which in this particular case makes the method wait for the results of the API. This is also one of the strong characteristics of the technologies used, as they perform I/O operations asynchronously by default.

4.3.3 User Interfaces

Only one page will be demonstrated in this paper for the same reason that previously only the implementation of DSL was demonstrated. This is the page for managing and configuring new test suites for the system, which can be seen in Figure 8. The user has on this page at his disposal the list of existing packages in the system, where he can remove or edit them. There is also a form for creating a new test suite, where the user only needs to specify the name, description and code of the new test suite. The code is written with the DSL presented earlier. In this case, the elements that can be used to write the code are the connectors below the form that are made available to the user according to the status of their script, to help the user and try to avoid errors. The other elements to include in the script are the primitive tests, and these are made available in a list next to the form where the user can even see their description to understand what the test does. To include a test in the script, the user just needs to click on it and it is automatically added to the script. This way, the user does not need to write anything manually, having to select the elements he wants to add to the script.

4.4 Validation

Having already implemented the system with all the requirements that were established, several test cases were created to be carried out in the system to validate the solution and confirm the fulfilment of all the proposed objectives. The first tests were carried out on the



Lista de Pacotes

Pacote AM FM
Pacote Exemplo
Pacote Completo
Pacote Loopback
Pacote BroadR-Reach
Pacote Sequência
Pacote Condicional

Criar Novo Pacote

Nome: Novo pacote

Descrição: Novo pacote, apenas para demonstração.

Código:

```
Power_on ->
```

Lista de Testes

Send_packet
Fail_Error
Power_on
Power_off
Set_Tune_fm_frequency
Procurar uma frequência válida na banda FM no emulador
Fm_seek_right
AMFM_RX_set_Volume
Check_fm_signal_quality
Get_fm_SNR_value
Get_fm_RSSI_value

Figure 8 Package creation and management page.

most fundamental functionalities of the system, the execution of the tests and the automation of the update in the face of changes introduced in its supply. Several test scenarios were simulated and the system behaved as expected, passing all performed tests.

In total, 28 test cases were carried out covering all the functionality of the system and in some of them with more than one test case. No more test cases were carried out because the time it would take to do so is immense, but the test cases performed were considered to be the most comprehensive cases and therefore will give the greatest coverage of requirements. After analyzing all the results obtained in the tests and verifying that they all passed, we can say that all requirements have been successfully implemented and the system is ready to be integrated with the other components.

5 Conclusions and Future Work

The main contributions of this paper are the design of the architecture to integrate a self-diagnosis tests system into a CPS and its implementation. This architecture provides a modular and extensible solution so that the system can be integrated with the CPS and perform any type of test. The system was implemented based on the proposed architecture, but only the part of the implementation corresponding to the DSL was demonstrated due to the paper size limit. To validate the implementation of the system and its compliance with the established requirements, 28 test cases were carried out to cover all requirements. All test cases have passed and, therefore, the system meets all the objectives.

The proposed modular and extensible architecture represents an innovation for research in self-diagnosis systems and CPS, as it allows the combination of these two types of systems, through the use of KDT methodology with a DSL to manage and configure the tests of the system. This architecture also allows the execution of the tests to be done remotely or by any other system with permission to make HTTP requests to the API REST provided. Although the focus of the architecture is the application in a CPS, it is also applicable to any type of system, since it is generic to accept any type of test. With this work, we proved that it is possible to integrate self-diagnosis tests systems into a CPS with a practical and also generic solution that can be integrated with other types of testing systems.

As future work, it would be interesting to improve the interface for creating new test suites in the system. Although the solution currently implemented is practical and allows good use, it could be even more practical and simple for the user if a drag and drop window were developed in the design of new test suites instead of writing a code script.

References

1 Márcio Filipe Alves Carvalho. Automatização de testes de software, 2010. URL: `https://files.isec.pt/DOCUMENTOS/SERVICOS/BIBLIO/teses/Tese_Mest_Marcio-Carvalho.pdf`.

2 S. Ciraci, J. C. Fuller, J. Daily, A. Makhmalbaf, and D. Callahan. A runtime verification framework for control system simulation. In *2014 IEEE 38th Annual Computer Software and Applications Conference*, pages 75–84, 2014. `doi:10.1109/COMPSAC.2014.14`.

3 Guru99. What is automation testing?, 2021. URL: `https://www.guru99.com/automation-testing.html`.

4 R. Hametner, D. Winkler, and A. Zoitl. Agile testing concepts based on keyword-driven testing for industrial automation systems. In *IECON 2012 - 38th Annual Conference on IEEE Industrial Electronics Society*, pages 3727–3732, 2012. `doi:10.1109/IECON.2012.6389298`.

5 Felienne Hermans, Martin Pinzger, and Arie Van Deursen. Domain-specific languages in practice: A user study on the success factors. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2009. `doi:10.1007/978-3-642-04425-0_33`.

6 Jingfan Tang, Xiaohua Cao, and A. Ma. Towards adaptive framework of keyword driven automation testing. In *2008 IEEE International Conference on Automation and Logistics*, pages 1631–1636, 2008. `doi:10.1109/ICAL.2008.4636415`.

7 Tomaž Kosar, Sudev Bohra, and Marjan Mernik. Domain-Specific Languages: A Systematic Mapping Study. *Information and Software Technology*, 2016. `doi:10.1016/j.infsof.2015.11.001`.

8 Charles W. Krueger. Software Reuse. *ACM Computing Surveys (CSUR)*, 1992. `doi:10.1145/130844.130856`.

9 Edward A. Lee. Cyber physical systems: Design challenges. In *Proceedings – 11th IEEE Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing, ISORC 2008*, 2008. `doi:10.1109/ISORC.2008.25`.

10 Marjan Mernik, Jan Heering, and Anthony M. Sloane. When and how to develop domain-specific languages. *ACM Computing Surveys*, 2005. `doi:10.1145/1118890.1118892`.

11 Jens Palsberg and C. Barry Jay. The essence of the Visitor pattern. In *Proceedings – International Computer Software and Applications Conference*, 1998. `doi:10.1109/CMPSAC.1998.716629`.

12 T. J. Parr and R. W. Quong. ANTLR: A predicated-LL(k) parser generator. *Software: Practice and Experience*, 1995. `doi:10.1002/spe.4380250705`.

13 Terence Parr and Kathleen Fisher. LL(*): The foundation of the ANTLR parser generator. In *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, 2011. `doi:10.1145/1993498.1993548`.

14 Terence Parr, Sam Harwell, and Kathleen Fisher. Adaptive LL(*) parsing. *ACM SIGPLAN Notices*, 2014. `doi:10.1145/2714064.2660202`.

15 Gabriele Tomassetti. The antlr mega tutorial, 2021. URL: `https://tomassetti.me/antlr-mega-tutorial/`.

16 X. Zhou, X. Gou, T. Huang, and S. Yang. Review on testing of cyber physical systems: Methods and testbeds. *IEEE Access*, 6:52179–52194, 2018. `doi:10.1109/ACCESS.2018.2869834`.