9th Symposium on Languages, Applications and Technologies

SLATE 2020, July 13–14, 2020, School of Technology, Polytechnic Institute of Cávado and Ave, Portugal (Virtual Conference)

Edited by

Alberto Simões
Pedro Rangel Henriques
Ricardo Queirós

Editors

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

Pedro Rangel Henriques
Centro Algoritmi (CAlg-CTC), Department of Informatics, University of Minho, Braga, Portugal
pedrorangelhenriques@gmail.com

Ricardo Queirós
ESMAD, Polytechnic of Porto, Portugal
ricardoqueiros@esmad.ipp.pt

ACM Classification 2012
Computing methodologies → Natural language processing; Software and its engineering → General programming languages; Information systems → Web applications

ISBN 978-3-95977-165-8

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/978-3-95977-165-8.

Publication date
September, 2020

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.

Digital Object Identifier: 10.4230/OASIcs.SLATE.2020.0

ISBN 978-3-95977-165-8 **ISSN 1868-8969** **https://www.dagstuhl.de/oasics**

OASIcs – OpenAccess Series in Informatics

OASIcs aims at a suitable publication venue to publish peer-reviewed collections of papers emerging from a scientific event. OASIcs volumes are published according to the principle of Open Access, i.e., they are available online and free of charge.

ISSN 1868-8969

https://www.dagstuhl.de/oasics

Contents

Invited Talk

Regular Papers

9th Symposium on Languages, Applications and Technologies (SLATE 2020).
Editors: Alberto Simões, Pedro Rangel Henriques, and Ricardo Queirós
OpenAccess Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Short Papers

Preface

This book compiles the seventeen papers accepted for the ninth edition of the Symposium on Languages, Applications and Technologies, SLATE 2020, published in the OASIcs – the OpenAccess Series in Informatics.

The symposium discusses languages as the base of communication at three different levels. Firstly, it covers all aspects regarding the communication between humans using natural language, and how to process it. Secondly, it reserves a discussion space for the processing of languages used in the communication of humans with computers. In this realm, works often lean over language design, processing, assessment, and applications. Finally, with the advent of networks and IoT, the symposium also tackles works related to the communication between computers emphasizing the design of domain-specific languages, its serialization, communication, and orchestration.

In this quite strange year it was not easy to have SLATE running. We had a first attempt to organize in the south of Portugal but for different reasons that was not possible. After preparing the organization in the north of Portugal, in Barcelos, we had the COVID-19 epidemic, making it hard for conferences to survive, especially when even large conferences are getting canceled.

With all these issues, we are happy to have a nice selection of papers, and be able to have the conference taking place almost at the usual dates, with a similar number of accepted papers from previous years.

We believe that this selection of papers can contribute for the progress of the research on languages processing covering the different problems that arise when dealing with programming languages, annotations languages and natural languages.

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 (*Jang F.M. Graat* and *Samuel Ferreira*) 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.

Alberto Simões, Pedro Rangel Henriques, and Ricardo Queirós

9th Symposium on Languages, Applications and Technologies (SLATE 2020).
Editors: Alberto Simões, Pedro Rangel Henriques, and Ricardo Queirós
OpenAccess Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

List of Authors

José João Almeida (4, 6, 10)
Centro Algoritmi (CAlg-CTC),
Department of Informatics,
University of Minho, Braga, Portugal

Ana Alves (9)
CISUC, University of Coimbra, Portugal
ISEC, Polytechnic Institute of Coimbra,
Portugal

Mauro C. Argañaraz (2)
Departamento de Informática,
Facultad de Ciencias Física Matemáticas y
Naturales (FCFMyN),
Universidad Nacional de San Luis, Argentina

Jorge Baptista (11)
University of Algarve, Campus de Gambelas,
Faro, Portugal
INESC-ID, Lisboa, Portugal

Fernando Batista (3, 7, 12)
ISCTE – Instituto Universitário de Lisboa,
Portugal
INESC-ID, Lisboa, Portugal

Mario M. Berón (2)
Departamento de Informática,
Facultad de Ciencias Física Matemáticas y
Naturales (FCFMyN),
Universidad Nacional de San Luis, Argentina

Pedro Miguel Oliveira da Silva (6)
Departamento de Informática,
Universidade do Minho,
Braga, Portugal

Renato Preigschadt de Azevedo (8)
Centro Algoritmi (CAlg-CTC),
Department of Informatics,
University of Minho, Braga, Portugal

João Ferreira (16)
Centre for Informatics and Systems of the
University of Coimbra, Portugal

Soraia Filipe (12)
Iscte – Instituto Universitário de Lisboa,
Portugal

Hugo Gonçalo Oliveira (9, 16)
CISUC, Department of Informatics Engineering,
University of Coimbra, Portugal

Jang F. M. Graat (1)
Smart Information Design, Amsterdam,
The Netherlands

Pedro Rangel Henriques (2, 4, 8)
Centro Algoritmi (CAlg-CTC),
Department of Informatics,
University of Minho, Braga, Portugal

Álvaro Iriarte (10)
Centro de Estudos Humanísticos da
Universidade do Minho, Braga, Portugal

José Paulo Leal (5, 13, 14)
CRACS & INESC Tec LA, Porto, Portugal
Faculty of Sciences, University of Porto,
Portugal

Joaquim Macedo (10)
Algoritmi, Departamento de Informática,
Universidade do Minho, Braga, Portugal

Nuno Mamede (11)
Universidade de Lisboa, Instituto Superior
Técnico, Portugal
INESC-ID, Lisboa, Portugal

Helena Moniz (3)
CLUL/FLUL, Universidade de Lisboa, Portugal
INESC-ID, Lisboa, Portugal
UNBABEL, Lisboa, Portugal

Ezequiel José Veloso Ferreira Moreira (17)
University of Minho, Braga, Portugal

Pedro Moura (4)
Centro Algoritmi (CAlg-CTC),
Department of Informatics,
University of Minho, Braga, Portugal

Francisco José Moreira Oliveira (15)
University of Minho, Braga, Portugal

Miguel Oliveira (4)
Centro Algoritmi (CAlg-CTC),
Department of Informatics,
University of Minho, Braga, Portugal

José Carlos Paiva (14)
CRACS – INESC, LA, Porto, Portugal
DCC – FCUP, Porto, Portugal

Maria João Varanda Pereira (8)
Research Centre in Digitalization and Intelligent
Robotics (CeDRI),
Instituto Politécnico de Bragança, Portugal

Afonso Pinto (3)
ISCTE – Instituto Universitário de Lisboa,
Portugal

9th Symposium on Languages, Applications and Technologies (SLATE 2020).
Editors: Alberto Simões, Pedro Rangel Henriques, and Ricardo Queirós
OpenAccess Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Ricardo Queirós (14, 18)
CRACS – INESC, LA, Porto, Portugal
uniMAD – ESMAD, Polytechnic of Porto, Portugal

José Carlos Ramalho (15, 17)
Department of Informatics, University of Minho, Braga, Portugal

Ricardo Ribeiro (7, 12)
Iscte – Instituto Universitário de Lisboa, Portugal
INESC-ID, Lisboa, Portugal

João Rodrigues (7)
Iscte – Instituto Universitário de Lisboa, Portugal

Ricardo Rodrigues (16)
Centre for Informatics and Systems of the University of Coimbra, Portugal
Polytechnic Institute of Coimbra, College of Higher Education of Coimbra, Portugal

Bernardo Sacanene (10)
Centro de Estudos Humanísticos da Universidade do Minho, Braga, Portugal

André Santos (5)
CRACS & INESC Tec LA, Porto, Portugal
Faculty of Sciences, University of Porto, Portugal

João Bruno Silva (5)
Faculty of Sciences, University of Porto, Portugal

Pedro Mimoso Silva (4)
Centro Algoritmi (CAlg-CTC), Department of Informatics, University of Minho, Braga, Portugal

Alberto Simões (10, 18)
2Ai, School of Technology, IPCA, Barcelos, Portugal

Tiago Sousa (9)
ISEC, Polytechnic Institute of Coimbra, Portugal

Jakub Swacha (14)
University of Szczecin, Poland

Hristo Valkanov (13)
Faculty of Sciences, University of Porto, Portugal

Maria J. Varanda Pereira (2)
Research Centre in Digitalization and Intelligent Robotics (CeDRI), Instituto Politécnico de Bragança, Portugal

List of Committees

Organizing Committee

Alberto Simões
Instituto Politécnico do
Cávado e do Ave, PT

Ricardo Queirós
Escola Superior de
Media Artes e Design, PP, PT

Pedro Rangel Henriques
Universidade do Minho, PT

Maria João Varanda Pereira
Instituto Politécnico de Bragança, PT

Goreti Pereira
Universidade do Minho, PT

Scientific Committee

Salvador Abreu
University of Evora, PT

Ana Alves
University of Coimbra, PT

Jorge Baptista
Universidade do Algarve, PT

Fernando Batista
INESC-ID & ISCTE-IUL, PT

Mario Berón
National University of San Luis, AR

Nuno Ramos Carvalho
University of Minho, PT

Bárbara Cleto
Polytechnic of Porto, PT

Luísa Coheur
IST/INESC-ID Lisboa, PT

Brett Drury
LIAAD-INESC-TEC, PT

João Fernandes
University of Coimbra, PT

Jean-Christophe Filliatre
CNRS, FR

Mikel Forcada
DLSI – Universitat d'Alacant, ES

Pablo Gamallo
University of Santiago de Compostela, ES

Hugo Gonçalo Oliveira
University of Coimbra, PT

Xavier Gómez Guinovart
Universidade de Vigo, ES

Geylani Kardas
Ege University International Computer
Institute, TR

Jan Kollar
FEI TU Kosice, SK

António Leitão
Universidade de Lisboa, PT

João Lourenço
NOVA LINCS, PT

Ivan Luković
University of Novi Sad, RS

Simão Melo de Sousa
Universidade da Beira Interior, PT

Marjan Mernik
University of Maribor, SL

Luis Morgado Da Costa
Nanyang Technological University, SG

Antoni Oliver
Universitat Oberta de Catalunya, ES

Thiago Pardo
Universidade de São Paulo, BR

Ricardo Queirós
ESMAD- P.PORTO & CRACS - INESC
TEC, PT

Alexandre Rademaker
IBM Research Brazil , BR

Pedro Rangel Henriques
University of Minho, PT

Ricardo Rocha
University of Porto, PT

9th Symposium on Languages, Applications and Technologies (SLATE 2020).
Editors: Alberto Simões, Pedro Rangel Henriques, and Ricardo Queirós
OpenAccess Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Ricardo Rodrigues
Polytechnic Institute of Coimbra, PT

Irene Rodrigues
Universidade de Evora, PT

André Santos
Universidade do Ponto, PT

João Saraiva
University of Minho, PT

Jose Luis Sierra
Universidad Complutense de Madrid, ES

Josep Silva
Universitat Politècnica de València, ES

Alberto Simões
2Ai Lab – IPCA, PT

Bostjan Slivnik
University of Ljubljana, SL

Arkaitz Zubiaga
Queen Mary University of London, UK

How Humans Succeed While Failing to Communicate

Jang F. M. Graat
Smart Information Design, Amsterdam, The Netherlands

Abstract

Humans communication is full of errors, but this does not prevent us from achieving common goals. Most of the constant flow of misunderstanding is not even noticed. We think we understand each other – until the moment when something stops making sense. If the common goal is important enough to us, we will ask for explanation: "What do you mean with that?" After receiving more information, we backtrack and correct the reconstruction of meaning in the conversation. Humans are fault-tolerant communicators because nothing is ever fixed: meanings are always temporary constructions and can be changed at any time, when they appear to be outdated or misconstrued. We can "change out minds" about something and we do not need to have perfect knowledge to survive. Those humans who strive for certainty in their lives are often called control freaks, i.e. unnatural exceptions.

Computers live on the other side of the certainty spectrum: a 1 is a 1 and a 0 will always be a 0. Even if the programmer tells the computer – via string of 1s and 0s – to do something really stupid, the computer will simply do it. No room for error, as every decision is precisely defined. Of course, layers of automated translations between our modern programming languages and the processor's basic instructions may introduce errors, but those are never called "misunderstandings". Instead, they are called bugs and they can be found and corrected, after which everything will be in full control again.

The task of making a computer understand human communication therefore seems to be the hardest thing to do. It requires mechanisms of uncertainty, backtracking and feedback loops, each of which need to take into account that no meaning is ever fixed. In the end, the goal of understanding human communication remains subordinate to the goal of putting the communication partners in a position where they can achieve common goals. Whether the communication partners are both human, both computer or a mix of the two does not really change that. Communication of any type is always a means, never a goal.

2012 ACM Subject Classification Human-centered computing → Interaction paradigms

Keywords and phrases Natural Language processing, Communication

Digital Object Identifier 10.4230/OASIcs.SLATE.2020.1

Category Invited Talk

1 Introduction

Although I earn my living by programming in the business domain of technical communication, I am not an expert in computer technologies around language. Most of you will know much more about various methods that are being used and developed in this field. Still, when Ricardo Queirós asked me to deliver a keynote for the SLATE conference, I did not hesitate. As a philosopher, I feel I can offer some reflections on human language that may be useful for anyone who is working in this interesting and complex field. The philosopher's role is not to declare what is true but to question anything that is implied. At the end of the day, it is you who decides what is true and false in your own view of the world.

From 1979 to 1986, I studied Psychology and Philosophy at two universities in Netherlands. Both my studies revolved around language, in all its various aspects. After graduating and failing to find work in academia, I entered the high-tech computer industry where I

have written tons of technical articles and manuals. I presented technical topics on many conferences and delivered training in many companies and universities around the globe. When I got bored with writing manuals I taught myself programming to create smart software for the efficient production and handling of technical information.

I have learned a lot about language by studying different ways in which language was being used in the distant past, in distant cultures, in literature, in poetry, in music, and in the business domain of technical information. There is so much more to language than meets the ear (or eye, if you are reading). In this keynote, I will try to give an overview of my insights in how language works and I will try to indicate which concepts and mechanisms ought to be incorporated when trying to make computers work with human language. Possibly, a lot of these are already handled – even in that case, by confirming the importance of what you are already working on, my keynote should be of some value to you.

2 Our common misunderstanding

When studying Philosophy at the University of Amsterdam, I had the pleasure of attending talks by professor Hubert Dethier. He was a talented speaker and chose topics he was passionate about. One of his phrases has stuck with me ever since: "we communicate by means of misunderstanding". In the decades that have passed since then, I have not come across a better description of that strange phenomenon we call human communication. Rather than having common access to a shared idea (which the word "communication" hints at), we are constantly stabbing in the dark, but nevertheless feel like everything is bright as daylight.

Humans are remarkably fault-tolerant language processing machines: they might – and usually will – have completely different ideas about the world, but this does not mean they cannot work towards a common goal. Our feeling of mutual understanding is only harmed when unexpected things happen. Someone laughs where no joke was intended. Someone starts frowning where you were expecting everything to be crystal clear. Someone is hurt and walks away when you thought you were complimenting them. Those signals are indicators that we may assign different meanings to the same words. They invite us to elaborate and make our assumptions explicit. More often than not, elaborating on assumed shared knowledge clears the air and allows resuming the joint actions that we were involved in.

When we speak, we are constructing a vision of the world in our minds and our words refer to that internal construction. The same process happens when listening to someone else. But there is no guarantee that the reconstruction in the mind of the listener is in any way similar – let alone identical – to the one that is in the mind of the speaker. As long as the internal reconstruction still works, there is no reason to assume that you have misunderstood the speaker. And even when some details are obscure, they may not be deemed important enough to ask for an elaboration. As long as the common goal is met (a course of actions that can be taken based on the communication), we assume we have understood each other well enough. Of course, the perception of that common goal may differ between us as well.

To give an impression of the vastly different roles that language has played in European history, I have chosen to follow Michel Foucault's magnificent work "*Les Mots et les Choses*" – which was published in English as "*The Order of Things*". Even this simple translation of a book title shows the difficulty in expressing the same complex of meanings in two different languages. The English title, although it mentions the subject of the book, fails to convey the importance of language in that order of things. The original French title merely mentions

words and things without going into the type of relationship between them – if there is such a relationship at all. Two very different phrases describing the same subject. Such uncertainty in meanings, as we shall soon see, was not always possible.

3 Language as ordering mechanism

In the Middle Ages, there were two very different languages: one was for speaking in everyday life (with every region having its own local language) and one was for science, law and religion. Because of the strong influence of the Catholic Church, the formal language was Latin. This was deemed to be the divine language, created by God. In some minds the language even is the essence of God, as we can read in John 1:1,

In the beginning was the Word, and the Word was with God, and the Word was God.

The link between this divine Latin language and the world was unquestionable. In fact, the language defined and ordered (or even created) the world. After all, God created both, and they were "of the same". If you knew the Latin word for an animal, e.g. a poisonous snake, and pronounced it in the right way and at the right moment, you had power over that animal. The word for an object was truly a handle to that object – all of the object. Obviously, the divine language (Latin) could not evolve – as God is all-knowing and eternal, his divine language cannot be other than all-encompassing and eternal. Moreover, as everything was a creation of God, everything was an expression of God, i.e. everything was essentially "of the same".

In natural science, which was the only science outside of studying the Bible, language was the ordering principle of the world. This, again, was a given by the fact that God created the world according to, and even via uttering, the Word. Botanists knew which species of flowers existed even if they had not yet found all of them: If there was a three-leaf and four-leaf clover, there also had to be a five-leaf clover, etc. Also, since God's word is eternal, there was no History – in the sense that nothing ever really changed. Of course people were born, had children and died and sometimes there were events like the plague, a volcano erupting or an earthquake, but essentially nothing really changed.

4 Language as representation

When, over a long period spanning the Renaissance, educated people started discussing new ideas (like the sun being the center of the universe with the earth as a globe circling around it), History started happening. No longer was everything expressed in the dead Latin language. Still, even though languages like Italian, French and English were alive and allowed new words and expressions to be added, words were taken to be simply representations of things in the world. New words were invented but they did not create new meanings: they were simply shortcuts for a longer phrase that describes the object you are talking about: an "employee" is nothing more or less than "a person who works for me on a regular basis in exchange for a fixed amount of money".

In a way, this made language invisible, as the medium did not change anything to whatever it was representing. There was no way to think about language differently, as there was no concept of an individual mind yet. Everyone had access to the same common – and thereby universal – knowledge. It is like the glasses you might be using to look at the world: you never see the glasses themselves unless you take them off. Basically, nobody ever took the glasses off for several centuries – until logicians started formalising natural language just before the turn of the 20$^{\text{th}}$ century. The sentence that was discussed for decades is this:

The Phoenicians did not know that the Morning Star was the Evening Star

This seems clear enough to most people, but for logicians there was a huge problem. The laws of logic state that a word representing an object has a one-to-one relationship with the object it represents: it can be replaced with the object without changing the logical value of the statement. By consequence, if two words A and B represent the same object, you can replace the word A for the word B without changing the logical value of the statement. According to the laws of logic, you can replace the above sentence with this one:

The Phoenicians did not know that Venus was Venus

Obviously, this sentence is not logically the same as the previous one. Still, laws of logic allowed this as a valid substitution. But if a valid substitution changed the logical value of the statement, all of logic was in danger. Which is why logicians have been debating this simple sentence for decades, coming up with all kinds of fancy solutions – none of which really stuck. For some logicians, natural language was the real problem, as people did not really know what they were talking about. You can only speak of things that can be true or false. Everything else is completely senseless. To quote Austrian philosopher Wittgenstein:

Whereof one cannot speak, thereof one must be silent.

Obviously, not everyone was satisfied with this position. In fact, Wittgenstein, after moving to Cambridge, completely turned around in his later days, defining the meaning of words from their usage instead. This is why in Philosophy there is mention of Wittgenstein 1 and 2, both of whom have inspired an entire philosophical language school.

5 Language as a package delivery service

The logician's fierce debate led to closer inspection of the glasses everyone was using to represent the world. With this, the science of language was born and one of the main names in the early years of this new science was Swiss linguist and philosopher Ferdinand de Saussure. He formalised the way language represents the world: the sign is a combination of the signifier (a word, an image, a sound) and the signified (the meaning). The sign became a kind of package delivery service: you wrap a meaning in it by using the right signifier and send it to the receiver, who unwraps the sign and finds the meaning. It may well be that multiple signifiers point to the same signified, but that just means there are different signs for the same thing.

This did not solve all problems, though, as there were many real-world examples where meanings were lost or misconstrued. They could not all be explained by incompetence of the person using the language. The same sign seemed to acquire different meanings depending on the person who used it, the circumstances in which they were used, the context in which it appeared. It seems that language is not the fully transparent medium that allows our minds to share the same universal meanings. The signified might not be "out in the world" but exist only in our private minds. The closest we may get to objective universal truth is the feeling that we understand more or less the same thing at the end of our conversation.

This failure to get common access to objective truth is normally not the fault of the package delivery service: they truthfully transport the signs from sender to receiver. But instead of simply unwrapping the package we seem to be doing something entirely different with it. We seem to be reconstructing the signified instead – and this is where all of the uncertainty gets into the picture. How can we be sure that we are reconstructing the same world that the sender has before their mind's eye when they sent the word packages to us?

6 Learning to (re-)construct the world

Words allow us to construct an imagined reality. In the development of a young child, the first phase is simply identifying objects by single words. Once the child moves on to uttering short sentences, it is expressing relationships between objects. That is where the observer becomes a builder, creating representations of the world. And once the child starts playing with the relationships in its mind, just like it is moving objects around in the physical world, the child becomes a creator of new meanings.

Uncertainty is introduced in each of these stages: how can we know the child is pointing to a car when pronouncing the word for it? It may be pointing to a bird that sits on top of it. As long as we do not recognise the mismatch between the focus of our attention and the object the child is pointing to, we will not figure out that there is a misunderstanding. Imagine we do see the mismatch but never correct the child: it would simply learn to refer to a bird with the word "car" and get into serious trouble in primary school when stating it saw a car fly.

This simple example shows that feedback loops are essential in creating even the most primitive level of communication. Making mistakes is when we learn to make connections between what we see and what we hear. And making mistakes requires getting feedback: I cannot do anything wrong when there is nobody pointing it out to me.

7 There is not one single truth

When we move beyond very practical meanings like "car" and "bird", things quickly get much more complex, as the meanings we are trying to communicate are no longer physical objects that we can point at to identify them. This also happens to be the level where most of human communication takes place. We express relationships between physical objects, which may not be measurable and therefore remain uncertain.

This is also the domain where most of our common misunderstanding happens, and often these remain unnoticed. Two people may agree completely in words, only to find out much later that their impression of what was agreed turned out to be very different. The reverse can also happen: two people arguing about something eventually discover they are just using different words for – supposedly – the same concept.

When there is no single truth, no single meaning to a phrase, reconstructing a world view out of the words that are communicated to me becomes a hazardous task. It resembles building a house out of playing cards, which are carefully balanced on top of each other. When a new meaning does not fit into its designated space, the entire building may be in danger of collapsing.

But then, getting a complete and correct reconstruction in our minds is never really a goal (unless you are a philosopher and make it your goal to know exactly what someone else had in mind). Communication is a tool that allows us to pursue a common goal. It is possible, and even quite common, that people have very different goals when engaging in communication. Someone may have the goal to look smart – in which case he might purposely add some incompatible words into his construction, so that the listeners cannot understand him at all. There may be an intention to spread gossip about someone just to make them look bad. On the listener's side there are many possible goals as well: learn something new, find proof that the speaker's views are dumb or dangerous, get quotes from the speaker that can be used for propaganda – or in court cases against them, etc.

8 Speaking is acting

To add just one more thing to the possibly confusing language puzzle there are speech positions at work in every conversation (and in any type of media). This aspect is studied in the speech act theory that was initiated by the English philosopher J.L. Austin in his book "How to do things with words". Every utterance is not just a statement but also an act. Some acts really change something in the world, but the effects depend on the context.

Answering "yes" to a direct question may get you a drink, get you married or cause a punch in your face. And if a Japanese answers "yes" it may not mean anything at all – as answering "no" is found to be the most horrible offending thing you can do in the Japanese culture. When I say you are guilty, nothing much happens except possibly you feeling a little embarrassed or offended. When a judge says exactly the same words, you may get sent to prison. Of course, the judge must also be in the right context (in a court room) and should not start the session with this statement but wait until all arguments are heard and everyone has the impression that careful consideration is made.

In any conversation, an exchange of speech acts takes place, in which speakers and listeners take turns in uttering words. All participants in the conversation have their own ideas about themselves, about the others and about the subject that is being discussed. Speakers often tap into collective memory by referencing historic figures or events, whereby only a certain implicit aspect of those referenced object classes comes into play. As a speaker, you have no control over the associations in the minds of your listeners, although the best speakers can really captivate and almost hypnotise their audiences.

9 Natural Logic

My thesis in Psychology revolved around argumentation. Classical argumentation theory uses classical logic. When you state A is B and later on state something that logically leads to A is not B, you have lost the argument. As you can imagine, this was not exactly my style of reasoning, as I believe that normative theories have little to do with the real world.

I was much more interested in what really happens in everyday-life situations: how does an argument between people work? How can a simple exchange of words evolve into a fierce fight, or rather how can a fierce discussion be resolved by moving the line of argumentation into a different direction?

This is where a barely known theory of a barely known Swiss professor comes in. Jean-Blaise Grize was logics professor at the linguistics faculty of the University of Neuchâtel and his "Natural Logic" was only ever published in French. It is a descriptive theory that allows formalising real-life arguments between groups of people. The theory includes the alternating positions of speaker and listener as well as the context in which a conversation takes place. An important aspect is the changeability of everything that is being exchanged between the people who engage in a conversation: nothing is ever eternally fixed and there is no reference to universals at any time.

Probably the most important aspect of Natural Logic is the way an object is defined: instead of referring to a specific object (which would be the signified of de Saussure's sign and caused all kinds of problems for logicians), Grize is using an "object class". This is not to be taken in the sense of traditional classifications: it is basically a focus of attention in our mind, which allows us to group all kinds of characteristics together. When I state that I need to take care of my "body" this does not necessarily imply I am thinking about all of my body all the time. I may be planning to go to the gym and work on my muscles, going to the supermarket to buy food, getting a beauty treatment or even filling out a form to

become an organ donor after I die. The meaning of "body" hints at an object class that may have a wide variety of possible members: your muscles, your skin, your organs, or indeed all of the above taken together.

The context becomes much more important in this type of logic, as it helps the listener determine which of the possible meanings is probably in the focus of attention of the speaker. Context in turn is an object class as well, and speaker and listener may have different ideas about what it really is that they are talking about. Taking turns in a conversation, allowing the listener to become speaker and reflect what they think you were talking about is an important aspect of human conversation. It may even be the key to getting any level of understanding at all, as there cannot be any level of certainty without some kind of feedback.

10 Concluding remarks

So how does all of this translate to the digital world of computers? It seems to me that efforts to create a universe of fixed meanings is doomed to fail when it comes to handling real-life conversations. Humans do not refer to fixed meanings but to bundles of associations that are normally referenced with the same token over time. It looks like the object classes of Jean-Blaise Grize would be good candidates to map to computer software, together with the other concepts of his Natural Logic. Unfortunately, none of his writings were ever translated and even the ones in the original French language may be hard to find. But even without having direct access to texts on Natural Logic there are aspects that can be incorporated into software that deals with human communication. I will try to list those aspects that come to my mind and leave it to the audience to figure out what they would relate to in their work.

1. Everyone has their own inner representation of the world
   Just imagine what the world looks like for a young, old, blind, deaf, color blind, tall, short, etc. person. Imagine someone who has never seen an airplane and sees one passing high overhead.
2. Words may have multiple meanings, and may change over time
   Cool used to mean rather cold, but it has acquired a new meaning in social media and marketing. Being social used to mean doing good things for your neighbours – today it seems to mean bragging about yourself as much as you can.
3. Associations may come into play at any given time
   The mention of a word triggers thoughts of other words in my mind: ball leads to soccer, but also to round, which leads to square, etc. Associations work via sound resemblance, word resemblance, shape resemblance, memories, among others.
4. Context is important and consists of various factors
   The location (restaurant, theater, police station, home, university), the history (common or separate), the participants in the conversation, the medium in which it takes place (newspaper, television, letters, social media).
5. Understanding requires active solicitation of feedback
   I cannot be sure that you have understood my message unless I check what you have understood. Asking to explain to me what you think I told you is one of the main instruments in education.
6. We cannot know whether someone else has really understood us
   Most domestic disputes may be caused by the failure to give explicit feedback – you assume you know what your partner means when they say something, instead of checking this when the differences in understanding were not yet too complex to overcome.

7. Speech positions may determine what words can achieve
   When I state “this means war” to another academic, I show I am prepared to get into a serious discussion and we might both get out of it in better shape. When the president says it, a lot of physical damage might follow.
8. Speaking is hardly ever a goal, but a means to achieve something
   Nobody reads a manual to kill their time: they are looking for a solution to a real problem. When speaking, there is something you want to achieve with it (looking cool, getting someone to do something for you, etc.). It might be a goal in itself for little kids who are learning to speak, but after that it quickly becomes a means to an end.
9. Various goals may be met within the same conversation
   Not all participants need to have the same goal with the conversation, and often you may never find out the differences until much later. Someone was just saying “yes” to your ideas just to make you look stupid after the plan broke down. There are many other examples that can easily be imagined.
10. Human language is a fascinating subject to study
    I guess you all can agree on that without me giving you more examples than I already have today.

Detection of Vulnerabilities in Smart Contracts Specifications in Ethereum Platforms

Mauro C. Argañaraz
Departamento de Informática, Facultad de Ciencias Física Matemáticas y Naturales (FCFMyN), Universidad Nacional de San Luis, Argentina
marganaraz@gmail.com

Mario M. Berón
Departamento de Informática, Facultad de Ciencias Física Matemáticas y Naturales (FCFMyN), Universidad Nacional de San Luis, Argentina
mberon@unsl.edu.ar

Maria J. Varanda Pereira
Research Centre in Digitalization and Intelligent Robotics (CeDRI), Instituto Politécnico de Bragança, Portugal
mjoao@ipb.pt

Pedro Rangel Henriques
Centro Algoritmi (CAlg-CTC), Department of Informatics, University of Minho, Braga, Portugal
pedrorangelhenriques@gmail.com

Abstract

Ethereum is the principal ecosystem based on blockchain that provides a suitable environment for coding and executing smart contracts, which have been receiving great attention due to the commercial apps and among the scientific community. The process of writing secure and well performing contracts in the Ethereum platform is a major challenge for developers. It consists of the application of non-conventional programming paradigms due to the inherent characteristics of the execution of distributed computing programs. Furthermore, the errors in the deployed contracts could have serious consequences because of the immediate linkage between the contract code and the financial transactions. The direct handling of the assets means that the errors can be more relevant for security and have greater economic consequences than a mistake in the conventional apps. In this paper, we propose a tool for the detection of vulnerabilities in high-level languages based on automatized static analysis.

2012 ACM Subject Classification Security and privacy → Vulnerability scanners

Keywords and phrases blockchain, ethereum, smart contract, solidity, static analysis, verification

Digital Object Identifier 10.4230/OASIcs.SLATE.2020.2

Funding This work has been supported by FCT – Fundação para a Ciência e Tecnologia within the Project Scope: UIDB/05757/2020.

1 Introduction

Blockchain is a technology based on the combination between cryptography, networks and incentive mechanisms designed to support the verification, execution and registration of transactions among different peers. In other words, blockchain platforms can be defined as decentralized databases that offer attractive properties, such as immutability of the stored transactions and the creation of a sense of confidence between peers without the participation of a third party. Hence, this architecture is suitable as an open and distributed ledger that can save transactions in a verifiable and permanent manner.

Cryptocurrency was the first blockchain technology application and it is a new form of digital asset based on a network that is distributed across a large number of computers. This decentralized structure allows them to exist outside the control of governments and central authorities. The most famous and used cryptocurrencies are Bitcoin [16] and Ethereum [5]. They offer, in addition to the exchange of digital assets, the execution of smart contracts.

Smart contracts are basically two combined concepts. One of them is software. Insensible and austere code, that does what is written and executes it for the world to see. The other one is the sense of agreement between the parts. They are computer programs that facilitate, verify and ensure the negotiation and execution of legal contracts. They are executed through blockchain transactions, they interact with the cryptocurrencies and have interfaces to handle the information of the contract's participants. When a smart contract is executed on the blockchain, it transforms into an autonomous entity that automatically carries on specific actions under certain conditions.

Ethereum is the primary public distributed computing platform based on blockchain that provides an environment that enables the execution of smart contracts in a decentralized virtual machine, known as Ethereum Virtual Machine (EVM) [5, 25].

The virtual machine handles the compute and the state of the contracts and uses a language based on a stack structure with a predefined group of instructions (opcode) [25]. Essentially, a contract is simply a series of statements of opcodes that are sequentially executed by the virtual machine.

In this article, a tool is proposed that reinforces the security aspects of the specifications of Ethereum Platform's smart contracts. This has its basis on a solid foundation of design, established and tried code patterns that facilitate the functional code writing process with no errors, to provide a process that carries on a static program analysis and detect flaws automatically.

This document is structured in this way: Firstly, a summary about related work in Section 2 is provided. Furthermore, in Section 3 security aspects of the smart contracts are discussed, before presenting the vulnerabilities analysis and the detection rules in Section 4. In Section 5 the tool is proposed and in Section 6 a use case is being described. Finally, the conclusions and future works are being outlined.

2 Related Work

In the search for Ethereum's smart contracts safety, different approaches have been adopted, focusing on formal semantics, security patterns and verification tools. According to the analysis, a distinction can be made between verification and design.

The goal of the verification approaches is to check that the existent smart contracts, which are written in high level languages (such as Solidity) or in EVM bytecodes, meet a policy or a security specification. Some examples of works, techniques and tools in that direction are:

- **Static program analysis tools for the automatic search of bugs.** Oyente [14] is a static analysis tool for EVM bytecode that relies on symbolic execution. Oyente supports a variety of pre-defined security properties that can be checked automatically. Zhou et al. proposed SASC [27] that extends Oyente by additional patterns and provides a visualization of detected risks. Majan [17] extends the approach taken in Oyente to trace properties that consider multiple invocations of one smart contract.

- **Static program analysis tools for the automatic verification of generic properties.** ZEUS [13] is a static analysis tool that analyzes smart contracts written in Solidity using symbolic model checking. Other tools proposed in the realm of automated static analysis for generic properties are Securify [24], Mythril [7] and Manticore [18] for analysing bytecode and SmartCheck [23] for analyzing Solidity code.
- **Frameworks for semi-automated testing of specific contract properties.** Hirai [12] formalizes the EVM semantics in the proof assistant Isabelle/HOL and uses it for manually proving safety properties for concrete contracts. Hildebrandt et al. [11] define the EVM semantics in the K framework. Bhargavan et al. [4] introduce a framework to analyze Ethereum contracts by translation into F*.
- **Dynamic monitoring of predefined security properties.** Grossman et al. [10] propose the notion of effectively callback free executions and identify the absence of this property in smart contract executions as the source of common bugs. A solution compatible with Ethereum is offered by the tool DappGuard [8].

On the other hand, the design approaches are aimed at the creation of secure smart contracts by providing frameworks for the development: they take into account new languages that are more verifiable, they provide a clearer and simpler semantics that is understandable for the smart contracts' developers or that allows a direct codification of the security policies. The examples of design propositions can be classified into:

- **High level languages.** Coblenz [6] propose Obsidian, an object-oriented programming language that makes states explicit and uses a linear type system for quantities of money. Flint [21] is a type-safe, capabilities-secure, contract-oriented programming language for smart contracts that gets compiled to EVM bytecode. Flint allows for defining caller capabilities restricting the access to security sensitive functions. These capabilities shall be enforced by the EVM bytecode created during compilation.
- **Intermediate languages.** Scilla [22] comes with a semantics formalized in the proof assistant Coq and therefore allows for a mechanized verification of Scilla contracts.
- **Security patterns.** Wöhrer [26] describes programming patterns in Solidity that should be adapted by smart contract programmers for avoiding common bugs.
- **Tools.** Mavridou and Laszka [15] introduce a framework for designing smart contracts in terms of finite state machines.

3 Security and Smart Contracts

The smart contracts on Ethereum are generally written in high level language and then are compiled in EVM bytecodes. The most prominent and most widely adopted is Solidity [9], it is used even in other blockchain platforms. Solidity is a contract oriented high level programming language whose syntax is similar to Javascript.

A smart contract analysis carried out by Bartoletti and Pompianu [3] shows that Bitcoin and Ethereum primarily focus on financial contracts. The direct handling of the assets means that the flaws are more likely to be relevant to the security and have greater financial consequences that the errors on typical applications, as evidenced by the DAO attack on Ethereum.

According to Alharby and van Moorsel [1], the current investigation on smart contracts has its focus on identifying and addressing the smart contract's issues and they classify them in the following four categories: codification, security, privacy and problems of performance. The technology behind Ethereum's smart contracts is still in the early stages, thus, codification and security are the most discussed topics.

3.1 Security Challenges in Ethereum

Security is the main concern when talking about Ethereum's programming owing to the following factors:

- Unknown runtime environment: Ethereum is different to the centrally administered runtime environments, either mobile, desktop or in the cloud. Developers are not used to their code being executed in a global network of anonymous nodes, without a secure relationship and with a profit reason.
- New software stack: The Ethereum stack (the Solidity compiler, the EVM, the consensus layer) is still in the developing stages, and security vulnerabilities are still being discovered.
- Highly limited ability to correct contracts: A deployed contract cannot be corrected, hence, it has to be correct before the deployment. This opposes the traditional software development process that promotes iterative techniques.
- Financially motivated anonymous attackers: In comparison with several cibernetic crimes, exploiting smart contracts offers greater incomings (cryptocurrencies' price has rapidly risen), facility for the charging (the ether and the tokens can be instantly commercialized) and a minor risk of punishment due to the anonymity and the lack of legislation on the subject matter.
- Rapid pace of development: Blockchain companies make an effort to rapidly launch their products, usually at the expense of the security.
- Sub-optimal high level language: Some investigations claim that Solidity as itself leads the developers to unsecure development techniques [26, 1].

3.2 Design Challenges and Patterns Usage

Understanding how smart contracts are used and how they are implemented could help smart contracts platforms' designers to create new domain-specific languages, which, with their designs, avoid vulnerabilities such as the ones that are being outlined posteriorly. In addition, this knowledge could help improve the analysis techniques for smart contracts, by promoting the usage of contracts with specific programming patterns. To this day, little efforts have been made in the collection and categorization of patterns and the toolbox they use in an organized way [26]. In the following bullet points, a general description of the typical design patterns that are inherently frequent or practical when talking about the codification of smart contracts.

- Authorization: This pattern is used for restricting the code in accordance with the invoker's direction. The vast majority of analysed contracts verify if the invoker's direction is the same as the direction of the owner of the contract, before carrying out critical operations (for instance, sending ether, calling the method suicide or selfdestruct).
- Oracle: Is possible that some contracts have to acquire data outside the blockchain. The Ethereum platform does not allow the contracts to consult external sites: otherwise, the determinism of the calculations would break, due to the fact that different nodes could receive different results for the same consultation. The oracles are the interface between the contracts and the outside.
- Randomisation: Since the execution of the contract needs to be deterministic, all the nodes have to obtain the same numerical value when requesting a random number: this conflicts with the desired randomisation requirements.
- Time limitations: Many contracts require the implementation of time restrictions, for instance, for specifying when an action is allowed.

- Termination: Due to the fact that the blockchain is immutable, a contract cannot be eliminated when it is no longer being used. In view of this, developers have to foresee a way of disable it, in a way that it still exists, but without responding. Generally, only the contract's owner is authorized to disable a contract.

The presented patterns address typical issues and vulnerabilities related with the execution of smart contracts. Wöhrer and Zdun [26] worked particularly with the security patterns: Check-Effects-Interaction, Emergency Stop (Circuit Breaker), Speed Bump, Frequency, Mutex and Balance Limit.

The most important in the security level is the Check-Effects-Interaction pattern that describes how a function's code should be structured in order to avoid secondary effects and undesired execution behaviours. It defines a certain order of actions: first, verifying all the previous conditions, then, the changes on the contract's state should be done and, finally, interacting with other contracts. In accordance with this principle, the interactions with other contracts should be, whenever possible, the last step in every function. This is because the moment a contract interacts with another contract, including the ether transactions, it gives control to the other contract. This gives the called contract the possibility of executing potentially damaging actions.

For instance, an attack known as re-entrancy, where an invoked contract returns the call to the current contract, before the completion of the first invocation of the function that contains the call. This can lead to an undesired execution behaviour of the functions, modifying the state variables to unexpected values or repeating the operations (such as the sending of funds).

4 Analysis of Vulnerabilities and Rules for its Detection

In this section a summary of the security vulnerabilities in the Ethereum platform and its high level language Solidity is being provided. The second part has its focus on security from a contract developer point of view, and some code patterns which were implemented in the tool are being described.

4.1 Vulnerabilities

The causes of the vulnerabilities are organized in the taxonomy proposed by the author Atzei et al. [2], classifying them depending on the context into: programming high level languages, virtual machine (EVM) and the particularities of blockchain.

- High level programming languages: This category assesses the weaknesses or flaws of the programming languages in addition to the misuse of the language made by the developers. In this piece of work, Solidity language vulnerabilities are being presented.
- Virtual Machine: In this category the vulnerabilities related to the virtual machine where the smart contracts are executed are being grouped.
- Blockchain: It involves the vulnerabilities of blockchain's infrastructure.

The vulnerabilities related to the virtual machine and to blockchain's infrastructure are common to every programming language. Nevertheless, the vulnerabilities related to the programming languages are particular to each one of them, therefore, it surges the necessity of counting with an extensible mechanism to define static program analysis rules that depend on the language which is being analysed.

All the vulnerabilities listed in Table 1 can be exploited to carry out attacks which, for example, steal money from the contracts. It is the case of the vulnerability known as re-entrancy, that allowed the attackers to steal 50 million dollars from the DAO organization.

Table 1 Vulnerabilities classification.

High level languages (Solidity)	Call to the unknown Exceptions disorder Send without gas Types conversion Re-entrancy Keeping Secrets
EVM	Immutable errors Lost ether in the transactions Stack size limited
Blockchain	Unpredictable state Random generation Time restrictions

4.2 Code Patterns and Static Program Analysis Rules

This section focuses on security from a developer's point of view. The aforementioned high level programming languages classification, the topics related to the Solidity code are subdivided into:

- Security: issues that lead to attacks from an user or malicious contract.
- Functional: they cause the violation of a scheduled functionality.

This distinction between the functional problems and the security ones is presented due to the fact that the first pose a problem even without an adversary (even though a malicious external actor can aggravate the situation), while the last ones do not. In Table 2 a list of analysis rules and its severity is being shown.

Table 2 Static program analysis rules.

Security	Equality on the balance	Average
	Non-verified external call	High
	Use of send instead of transfer	Average
	Denial of a service because of an external contract	High
	Re-entrancy	High
	Malicious libraries	Low
	Use of tx.origin	Average
	Transfer of all the gas	High
Functional	Integer division	Low
	Blocked money	Average
	Non-verified maths	Low
	Dependence on the timestamp	Average
	Unsecure inference	Average

4.2.1 Security

Equality on the balance. Avoiding the verification of the strict equality of a balance, due to the fact that an adversary could send ether to any account through minery or through the selfdestruct call. The pattern detects comparison expressions with == and != that contain this.balance as right or left side. In Listing 1 example code is shown.

Listing 1 Avoid the verification of the balance equality.

```
if(this.balance == 1 ether) { ...} // Not Ok
if(this.balance >= 1 ether) { ...} // Ok
```

Non-verified external call. Wait until the calls to an external contract fail. When sending ether, verify the return's value and deal with the errors. The recommended way of carrying out the ether transactions is with the primitive transfer.

The pattern detects a call to an external function (call, delegate or send) that is not inside an if sentence. In Listing 2 some lines of vulnerable code are shown.

Listing 2 Fragment of vulnerable code.

```
addr.send(1 ether); // Not Ok
if(!addr.send(1 ether)) revert ; // Ok
addr.transfer(1 ether); // Recommended
```

Use of send instead of transfer. The recommended way of completing payments is addr.transfer(x),that automatically throws an exception if the transaction is unsuccessful, avoiding the aforementioned problem. The pattern detects the keyword send.

Denial of a service because of an external service. A conditional sentence (if, for, while) should not depend on an external call due to the fact that the invoked contract can fail (throw or revert) permanently, not allowing the invoker to complete the execution.

In Listing 3, the invoker expects that the external contract returns an integer, but the real implementation of the external contract can generate an exception in some cases or in all of them.

Listing 3 Fragment of vulnerable code.

```
function dos(address oracleAddr) public {
    badOracle = Oracle(oracleAddr);
    if(badOracle.answer() < 1) revert;
}
```

This rule contains multiple patterns:

- An if sentence with an external call in the condition and a throw or revert in the body.
- A for or if instruction with a call to an external function in the condition.

In Listing 4 a possible fraudulent implementation of the answer() method of the Oracle contract is presented.

Listing 4 Answer() method of the Oracle contract.

```
function answer() public returns int8 {
  throw;
}
```

Re-entrancy. In Section 3.2, it was stressed the importance of implementing the Check-Effects-Interaction pattern for mitigating the re-entrancy attacks. In Listing 5, a method exposed to the aforementioned vulnerability can be observed.

Listing 5 Method exposed to re-entrancy attack.

```
mapping (address => uint) balances;

function withdraw () public {
    uint balance = balances[msg.sender];
    if(msg.sender.call.value(balance)() {
        balances [msg.sender] = 0;
    }
}
```

The contract on msg.sender can obtain multiple refunds and recover all the ether of the contract put as an example through a recursive call to withdraw before its fee descends to 0. In Listing 6 it is showed the same functionality but taking into account the pattern: first the invariants are verified, then the internal state is updated and, finally, they communicate with external entities. The pattern detects a call to an external function that is followed by a call to an internal function.

Malicious libraries. Third parties' libraries can be malicious. Avoid the external dependencies or make sure that the third parties' code only implements the desired functionality. The pattern simply detects the keyword library.

Listing 6 Implementation of Check-Effects-Interaction.

```
function withdraw() public {
  uint balance = balances[msg.sender];
  balances[msg.sender] = 0;
  msg.sender.transfer(balance);
}
```

Use of tx.origin. The contracts can call the public functions of the rest. tx.origin is the first account in the call chain (not a contract); msg.sender is the immediate invoker. For example, in an A $\rightarrow$ B $\rightarrow$ C call chain, from C's point of view, tx.origin is A, and msg.sender is B. Use msg.sender instead of tx.origin for authentication. The pattern detects the environment variable tx.origin.

Transfer of all the gas. Solidity provides a myriad of ways of transferring ether. addr.call.value(x)() transfers x ether and redirects all the gas to addr, which could generate vulnerabilities such as re-entrancy. The recommended way of transfering ether is addr.call.value(x), that only provides an allowance of 2300 units of gas to the recipient. The pattern detects the functions whose name is call.value and whose argument list is empty.

4.2.2 Functional

Integer division. Solidity does not admit floating points types or decimals. For the integer division, the quotient is rounded down. Be aware of that, especially when calculating the ether or token quantities. The pattern detects the division (/) where the numerator and denominator are literal numbers.

Blocked money. The contracts programmed for receiving ether should implement a way of withdrawing it, in other words, call a transfer (recommended), send or call.value at least once. The pattern detects contracts that contain a payment function (payable), but that contain none of the aforementioned withdrawal functions.

Non-verified maths. Solidity is prone to suffer integer overflows. The overflow produces unexpected effects and could provoke a loss of funds if it is exploited by a malicious account. Use the SafeMath library [19] that verifies the overflows. The pattern detects arithmetic operations +.-.*, that are not inside of a conditional declaration.

Dependence on the timestamp. The miners can manipulate the environment variables and they are likely to do so if they can benefit from it. Use the block number and the average time between blocks for estimating the current time or use secure randomisation sources. The pattern detects the environment variable now.

Unsecure inference. Solidity admits type inference: the type of i in var = 42; is the smallest integer type that is enough for storing the value of the right side (uint8). Consider for loop in Listing 7:

Listing 7 Example of unsecure type inference.

```
for (var i = 0; i < array.length; i++) { ... }
```

The type of i is inferred to uint8. If array.length is bigger than 256, a overflow will occur. Explicitly define the type when declaring integer variables. The pattern detects allocations where the left side is a var and the right side is an integer (that matches with ^[0-9]+$).

5 Tool

In this piece of work a new project called OpenBalthazar is being proposed. It aims to raise a strategy for the detection of vulnerabilities in high level programming languages (Solidity) based on automatized static analysis.

5.1 Architecture

The static code analysis ensures complete coverage without executing the program and fast enough in a reasonably sized code. It usually includes three stages:

- Construction of an intermediate representation like the abstract syntax tree (AST), for a more thorough analysis in comparison with the analysed text.
- Enrichment of the AST with additional information using algorithms and language processing techniques.
- Detection of vulnerabilities, a repository of patterns that define the criteria when talking about intermediate language terms.

As it is shown in Figure 1, the tool is composed of four components:

Language Selector. This component is responsible for detecting the programming language in which the specification is written (source code) and determines the component that will be instantiate, through dynamic mechanisms that framework .NET (reflection) presents, for processing a request.

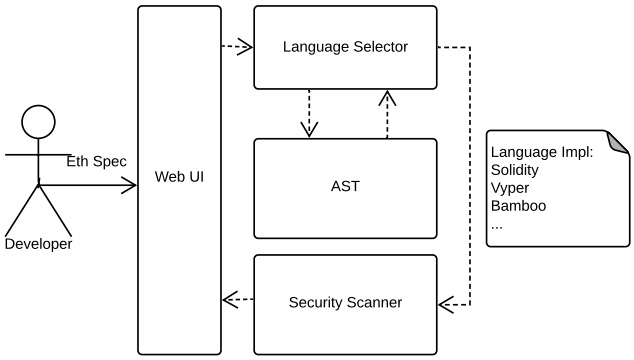


Figure 1 Principal components of the tool.

AST. This component builds an abstract syntax tree of a programming language, such as Solidity. Each tree's node has information of the language construction, for instance, sentences, condition structures, exception management, and applying different routes to the AST, the desired information can be extracted.

Security Scanner. This component takes as an input an AST of a specific language and executes the verification rules defined in a pattern repository for that language.

Web UI. This is the component that the developer interacts with. It provides tools for editing source code, such as file management, syntax coloring, line enumeration , etc. This frontend component was developed with the ReactJS javascript framework.

5.2 Implementation

OpenBalthazar is a static program analysis web tool for Ethereum platform smart contracts implemented with Microsoft.NET Core 3.1 in backend components. Security scanner component executes lexical and syntactic analysis in the Ethereum supported languages source code. Currently, Solidity, which is the *de facto* language of the industry, is implemented; still, the tool is extensible and the rest of the languages, such as Vyper or Bamboo, can be incorporated through the envisaged extensibility mechanisms.

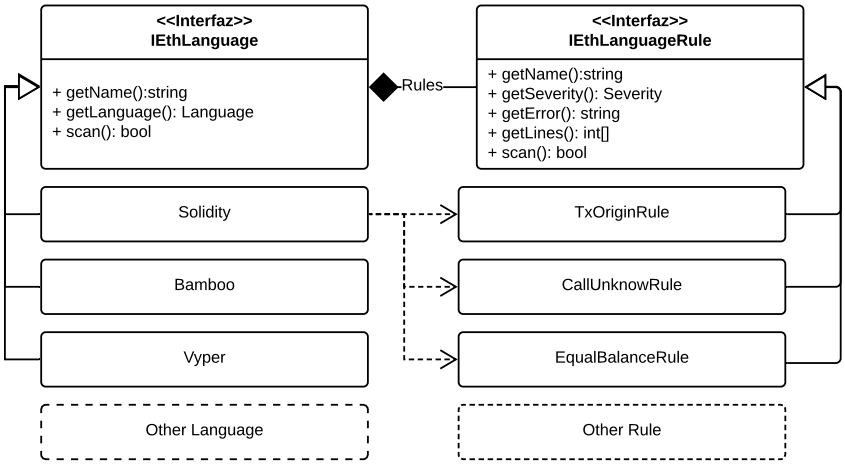


Figure 2 Extensibility mechanism.

To add a new language to the tool or add a new analysis rule to an existing language, the IEthLanguageRule and IEthLanguage interfaces ought to be implemented, respectively. Figure 2 shows how the Solidity language was implemented and how to proceed for incorporating the rest of the languages. In the same way, each language has a set of analysis rules that implement the IEthLanguageRule interface. In the `scan()` method the algorithm for the detection of a particular vulnerability is specified, indicating the error message and the specific severity.

In AST component, ANTLR 4 is used in addition to a Solidity grammar for generating the XML analysis tree (AST). In release 4.2, Antlr introduced the *visitor* and *listener* mechanisms that lets you implement DOM visiting or SAX-analogous event processing of tree nodes. This feature improves group translation operations by patterns in the tree rather than spreading operations across listener event methods. Another important idea is that, since we are talking about parse trees not abstract syntax trees, we can use concrete patterns instead of tree syntax. For example, we can say `x = 0;` instead of `AST (= x 0)` where the ";" would probably stripped before it went into the AST.

The vulnerability patterns are detected through the usage of parse tree patterns and XPath queries in the IR. Thus, OpenBalthazar provides a full coverage: the analised code is fully translated to the IR, and all of its elements can be reached through the XPath's selection mechanism. The line numbers are stored as XML attributes and help locating the findings in the source code. The tool can be expanded for admitting other smart contracts languages adding the ANTLR grammar and a pattern database.

If the '`use of tx.origin`' vulnerability is considered, the aim of the rule is detecting constructions that prove the existence of these identifiers in a contract. The analysis tree of this construction is shown in Figure 3.

The corresponding XPath pattern is shown in Listing 8. In this case, false positives are not expected, due to the fact that the aimed construction can be precisely described with XPath. The more complex rules cannot be precisely described with XPath , which may leads to false positives. In some scenarios false positives are related with the limitation of de static code analysis. Nevertheless, if the re-entrancy rule is considered, OpenBalthazar informs about the violations to the `Checks-Effects-Interactions` pattern, that not always leads to a re-entrancy (false positives).

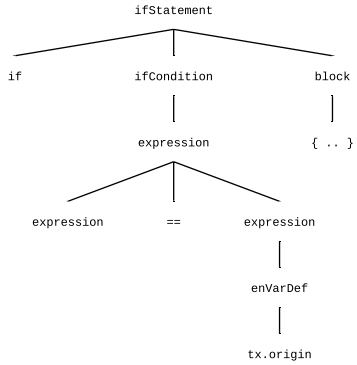


Figure 3 Analysis tree.

Listing 8 Fragment of the method Scan() corresponding to the class TxOriginRule.

```
IParseTree tree = solidityParser.sourceUnit();
ParseTreePattern pattern =
    solidityParser.CompileParseTreePattern("tx.origin",
    SolidityParser.RULE_expression);
IList<ParseTreeMatch> matches =
    pattern.FindAll(tree, "//expression");
foreach (ParseTreeMatch match in matches) {...}
```

In Listing 9 is shown some parse tree patterns.

6 Use Case: Etherscan Scanning

One of OpenBalthazar tools's strengths is the possibility of regaining the source code of the verified contracts of the EtherScan platform[1]. EtherScan is a platform for Ethereum that provides block exploration, search, analysis, source code's verification and APIs services. The source code's verification provides transparency for the developers who interact with smart contracts. When loading the source code, EtherScan will compare the compiled code with the one deployed in the blockchain.

OpenBalthazar provides the possibility to scan all the smart contracts whose source code has been published in EtherScan[2]. To accomplish this task, OpenBalthazar connects to the EtherScan API, retrieves the source code for published contracts, and then applies the set of rules predefined in Section 4.2. The result of the analysis is presented in a dashboard that allows examining the degree of vulnerability of the set of analyzed contracts.

The results of the analysis carried out by OpenBalthazar are shown in Figure 4. As it can be seen, 7.4% of the contracts are vulnerable. In this context and with the aim of simplifying the analysis, the results of the vulnerability known as re-entrancy will be shown, since, as

[1] `https://etherscan.io`

[2] At the time of writing this article, there are over one million smart contracts deployed on Ethereum. Out of these contracts, over 49.000 have been verified on EtherScan and 5462 contracts are available for viewing source code.

Listing 9 Fragments of parse tree patterns.

```
//EqualsBalanceRule
ParseTreePattern pattern =
    solidityParser.CompileParseTreePattern("this.balance (== | !=)
    <expression>", SolidityParser.RULE_expression);
IList<ParseTreeMatch> matches =
    pattern.FindAll(tree, "//expression");

//TimestampDependenceRule
ParseTreePattern pattern =
    solidityParser.CompileParseTreePattern("now",
    SolidityParser.RULE_expression);
IList<ParseTreeMatch> matches =
    pattern.FindAll(tree, "//expression");

// ReentrancyRule
ParseTreePattern pattern =
    solidityParser.CompileParseTreePattern("<expression>
        .<identifier>(<functionCallArguments>)",
        SolidityParser.RULE_expression);
IList<ParseTreeMatch> matches = pattern.FindAll(tree, "//expression");
foreach (ParseTreeMatch match in matches)
{
        if(match.Get("identifier").GetText().Equals("send") ||
                match.Get("identifier").GetText().Equals("value"))
        ...
}
```

mentioned in Section 3, it is considered the most important in terms of security. As of today, and with the improvements that have been incorporated into the language, it remains the responsibility of the programmers and auditors to mitigate their risk.

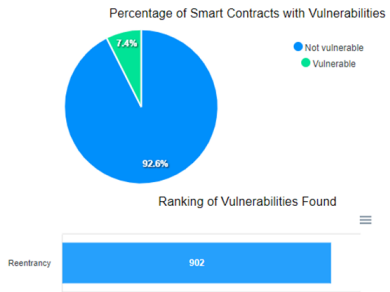


Figure 4 OpenBalthazar dashboard.

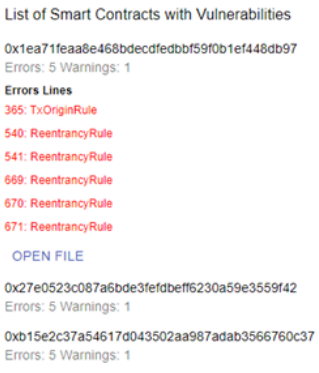


Figure 5 List of vulnerable contracts.

OpenBalthazar found 902 re-entrancy issues, it means contracts have calls to third-party contracts that could potentially be violated. In the right pane, shown in Figure 5, the addresses of vulnerable contracts are listed. Pressing on the address of the contract displays the list of vulnerabilities found and a button to open the source code of the contract in the editor of OpenBalthazar.

The facility for obtaining a deployed contract's source code in the principal network and the possibility of executing a complete analysis that detects the vulnerabilities transforms into a very powerful but hazardous characteristic, due to the fact that it enables the attacker to scan the network in the search of vulnerable contracts and prepare new contracts that exploit them. From the other side, a security specialist could use the vulnerabilities scan to activate the defense mechanisms that are usually implemented in the smart contracts for mitigating adverse situations.

7 Conclusions and Future Extensions

In the new programming paradigm and decentralized apps presented in the crypto ecosystem, where cryptography, distributed computing and the incentive mechanisms come together, the central issue that the high level programming languages face in the Ethereum platform is that the most complex language structures attempt against the security mechanisms and add more confusion to the contract developers, for example Solidity has 3 different way to send money to an account. Security methodologies such as SAMM [20] encourage the use of simple constructs to avoid introducing vulnerabilities. Hence, languages that are currently in the development stages has simpler structures and are focused on security concern. Until these new languages are available, it is necessary that tools are designed and constructed that helped in the chore of labour of programming, deploying and monitoring the smart contracts in terms of obtaining the best result possible.

The tool presented in this paper allows the contract developers to discover code vulnerabilities before deployment. In the case that the obtained result delivers high severity vulnerabilities, the professional will have to act consequently and try to stop the contract from providing services before it can be attacked. Another scenario where this tool aims to contribute is in Software Protection. In this context, we focus on `Man At The End` (MATE) attacks where the intruder may be a member of the development team or someone who was part of at some point.

EtherScan use case demonstrate how obtaining the source code of a group of deployed contracts in the principal network and how executing a complete analysis that detects yours vulnerabilities. However, only 0.5% of the source code is available in EtherScan to analyze, so it is essential to incorporate the EVM bytecode transformation, obtained from the principal network, into a high-level language (such as Solidity), in order to later carry out the aforementioned analyzes. Due to the novelty of this area we are still lacking good reverse-engineering tools to use, so we plan to build a Solidity decompiler for EVM bytecode.

Hereafter, some analysis included in the line of research are being succinctly described.

- Construction of an intermediate representation, enriched by additional information using algorithms and language processing techniques to facilitate the static program analysis of the contracts.
- Static program analysis of the source code and decision of use of security patterns.
- Implementation of a tool that allows the automatization of the process of analysis and detection of security flaws.

Researchers will continue with studies in this field aiming at improving skills and carrying out a follow-up on the new threats, vulnerabilities and cyberattacks regarding the deployment and execution of smart contracts. It is also planned to continue with the following future work:

- Generalization for other blockchain platforms that support smart contracts.
- Development of a Solidity decompiler for EVM bytecode.
- Dynamic analysis of the source code.
- Analysis of the security aspects that stem from the interoperability with other platforms.

References

1 M. Alharby and A. van Moorsel. Blockchain-based smart contracts: A systematic mapping study. *Fourth International Conference on Computer Science and Information Technology (CSIT-2017)*, 2017. arXiv:1710.06372v1.

2 N. Atzei, M. Bartoletti, and T. Cimoli. A survey of attacks on ethereum smart contracts sok. In *Proceedings of the 6th International Conference on Principles of Security and Trust - Volume 10204*, page 164–186, 2017. doi:10.1007/978-3-662-54455-6_8.

3 M. Bartoletti and L. Pompianu. An empirical analysis of smart contracts: Platforms, applications, and design patterns. In *Financial Cryptography and Data Security*, 2017.

4 K. Bhargavan, A. Delignat-Lavaud, C. Fournet, A. Gollamudi, G. Gonthier, N. Kobeissi, N. Kulatova, A. Rastogi, T. Sibut-Pinote, N. Swamy, and S. Zanella-Béguelin. Formal verification of smart contracts: Short paper. In *Proceedings of the 2016 ACM Workshop on Programming Languages and Analysis for Security*, 2016. doi:10.1145/2993600.2993611.

5 V. Buterin. Ethereum: A next-generation smart contract and decentralized application platform, 2014. URL: https://github.com/ethereum/wiki/wiki/White-Paper.

6 M. Coblenz. Obsidian: A safer blockchain programming language. In *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*, page 97–99, 2017.

7 ConsenSys. Mythril. https://github.com/ConsenSys/mythril, 2018.

8 T. Cook, A. Latham, and J.H. Lee. Dappguard: Active monitoring and defense for solidity smart contracts. https://courses.csail.mit.edu/6.857/2017/project/23.pdf, 2017.

9 Ethereum. Solidity. https://media.readthedocs.org/pdf/solidity/develop/solidity.pdf, 2018.

10 S. Grossman, I. Abraham, G. Golan-Gueta, Y. Michalevsky, N. Rinetzky, M. Sagiv, and Y. Zohar. Online detection of effectively callback free objects with applications to smart contracts. *Proc. ACM Program. Lang.*, 2(POPL), December 2017. doi:10.1145/3158136.

11 E. Hildenbrandt, E. Saxena, X. Zhu, N. Rodrigues, P. Daian, D. Guth, and G. Rosu. Kevm: A complete formal semantics of the ethereum virtual machine. In *2018 IEEE 31st Computer Security Foundations Symposium (CSF)*, pages 204–217, 2018.

12 Y. Hirai. Defining the ethereum virtual machine for interactive theorem provers. In *Financial Cryptography and Data Security*, page 520–535. Springer International Publishing, 2017.

13 S. Kalra, S. Goel, M. Dhawan, and S. Sharma. Zeus: Analyzing safety of smart contracts. In *25th Annual Network and Distributed System Security Symposium*, 2018. doi:10.14722/ndss.2018.23092.

14 L. Luu, D.H. Chu, H. Olickel, P. Saxena, and A. Hobor. Making smart contracts smarter. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, page 254–269, 2016. doi:10.1145/2976749.2978309.

15 A. Mavridou and A. Laszka. Designing secure ethereum smart contracts: A finite state machine based approach, 2017. arXiv:1711.09327.

16 S. Nakamoto. Bitcoin: Un sistema de efectivo electrónico usuario-a-usuario, 2008. URL: http://bitcoin.org/bitcoin.pdf.

17 I. Nikolic, A. Kolluri, I. Sergey, P. Saxena, and A. Hobor. Finding the greedy, prodigal, and suicidal contracts at scale. In *Proceedings of the 34th Annual Computer Security Applications Conference*, page 653–663, 2018. doi:10.1145/3274694.3274743.

18 Trail of Bits. Manticore. https://github.com/trailofbits/manticore, 2018.

19 OpenZeppeling. Safemath, 2019. Accessed: 2019-05-03. URL: https://github.com/OpenZeppelin/openzeppelin-solidity/blob/master/contracts/math/SafeMath.sol.

20 OWASP. Software Assurance Maturity Model: A guide to building security into software development. Version 1.5, 2020. URL: https://owasp.org/www-pdf-archive/SAMM_Core_V1-5_FINAL.pdf.

21 F. Schrans, S. Eisenbach, and S. Drossopoulou. Writing safe smart contracts in flint. In *Conference Companion of the 2nd International Conference on Art, Science, and Engineering of Programming*, page 218–219, 2018. doi:10.1145/3191697.3213790.

22 I. Sergey, V. Nagaraj, J. Johannsen, A. Kumar, A. Trunov, and K.C.G. Hao. Safer smart contract programming with scilla. *Proc. ACM Program. Lang.*, 2019. doi:10.1145/3360611.

23 SmartDec. Smartcheck: a static analysis tool that detects vulnerabilities and bugs in solidity programs, 2019. Accessed: 2019-05-03. URL: https://github.com/smartdec/smartcheck.

24 P. Tsankov, A. Dan, D. Drachsler-Cohen, A. Gervais, F. Bünzli, and M. Vechev. Securify: Practical security analysis of smart contracts. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, page 67–82, 2018. doi:10.1145/3243734.3243780.

25 G. Wood. Ethereum: A secure decentralised generalised transaction ledger, 2017. URL: https://ethereum.github.io/yellowpaper/paper.pdf.

26 I. Wöhrer and U. Zdun. Smart contracts: Security patterns in the ethereum ecosystem and solidity. In *2018 International Workshop on Blockchain Oriented Software Engineering*, 2018.

27 E. Zhou, S. Hua, B. Pi, J. Sun, Y. Nomura, K. Yamashita, and H. Kurihara. Security assurance for smart contract. In *2018 9th IFIP International Conference on New Technologies, Mobility and Security (NTMS)*, pages 1–5, 2018.

Detection of Emerging Words in Portuguese Tweets

Afonso Pinto
ISCTE – Instituto Universitário de Lisboa, Portugal
http://www.iscte-iul.pt
adcmm@iscte.pt

Helena Moniz
CLUL/FLUL, Universidade de Lisboa, Portugal
INESC-ID, Lisboa, Portugal
UNBABEL, Lisboa, Portugal
http://www.inesc-id.pt
Helena.Moniz@inesc-id.pt

Fernando Batista
ISCTE – Instituto Universitário de Lisboa, Portugal
INESC-ID, Lisboa, Portugal
http://www.inesc-id.pt
fernando.batista@iscte-iul.pt

Abstract

This paper tackles the problem of detecting emerging words on a language, based on social networks content. It proposes an approach for detecting new words on Twitter, and reports the achieved results for a collection of 8 million Portuguese tweets. This study uses geolocated tweets, collected between January 2018 and June 2019, and written in the Portuguese territory. The first six months of the data were used to define an initial vocabulary on known words, and the following 12 months were used for identifying new words, thus testing our approach. The set of resulting words were manually analyzed, revealing a number of distinct events, and suggesting that Twitter may be a valuable resource for researching neology, and the dynamics of a language.

2012 ACM Subject Classification Computing methodologies → Natural language processing

Keywords and phrases Emerging words, Twitter, Portuguese language

Digital Object Identifier 10.4230/OASIcs.SLATE.2020.3

Funding This work was supported by national funds through FCT, Fundação para a Ciência e a Tecnologia, under project UIDB/50021/2020.

1 Introduction

Social networks are basically a way/facilitator for people to communicate and exchange ideas among themselves, so it is natural that they play an important role in the evolution of writing, reading, and take part in the introduction of new words and expressions. In recent years, social networks have become more and more part of the daily life of Portuguese society. Twitter started as a chat room with a limited number of people, and it is now a noisy place [16], where people, regardless of the age, gender, or social class, produce all possible content about the aspects of their daily lives, thus being the ideal place for a wide range of language studies.

According to [10], if the structure of language is necessary and is considered the learning basis for any language, then the role of vocabulary can also not be neglected since it provides the necessary means. Traditionally, two types of lexical variation have been identified [13]. The semasiological variation refers to the variation in the meaning of words, such as the

variation in the meaning of the word "cedo", which may denote various concepts depending on the context of the sentence in which it is found, may refer in temporal terms (e.g. *antes do tempo/before the time*), as it may be used in its verbal form, for example *ceder o lugar/give the place*, meaning to offer the seat to someone. The onomasiological variation refers to the variation of how the concepts are identified, such as, for example, the variation in the words to identify the waiting place for the bus, which in European Portuguese would be a "*paragem de autocarro/bus stop*" and in Brazilian Portuguese would be a "*ponto de ônibus/bus stop*". According to [9] semasiological variations involve changes in the meaning of words, while onomasiological variation involves changes in how words are identified, which includes the formation of new words.

Regarding the nature of words and distinctions of meaning, [17] presents four different ideas. The first one is that, except for technical words, two different words contain different meanings, so a literal translation may partially distort the true meaning that is intended to be conveyed. The second idea is that in most languages the number of words with only one meaning is very small. The third idea is that a word gets its meaning according to the context in which it is inserted. Finally, the fourth idea considers that in the same language there is no single word that can replace another, such as the word ocean and sea, although they are almost synonyms, they are used for a similar but different effect, therefore one cannot replace one by the other in certain contexts. So, two questions arise:

- What is the role that social networks play in the development of the vocabulary of a language?
- How social networks may be regarded as a linguistic resource for vocabulary development?

In addition to studying the words that appear for the first time on social networks, it is also important to understand whether people are aware of their importance. According to [2, 4, 11], there are blogs where users (students) can obtain knowledge by consulting them. Similarly, [14] highlighted the idea that online blogs and social networks may be didactic, since the students, by leaving comments and chatting with others, may improve their competence in a given subject. The importance of social networks for the development and evolution of the vocabulary of a language is now undeniable.

The area of the geographic spread of linguistic change is also of great importance, and has been the target of several theories proposed [5], being the "*wave model*" most notable, which has replaced the "*tree model*". The *wave model* predicts that new emerging words will be propagated radially with a central location limited only by physical distance, while the "hierarchical" model predicts that its propagation will also be conditioned by population density, thus propagating between urban cities and only later can it reach rural areas. Through empirical evidence, it was possible to understand that both models are related to each other [18, 3, 15], to that sense, the "counter-hierarchical model" was proposed in order to explain the spread between urban and rural areas. With this, the researchers demonstrated that with regard to propagation act the factors physical distance, population density and cultural patterns, however the intensity with which these factors intervene is not clear and seems to partially explain the propagation of new emerging words. [1] discovered demographic similarities between cities, which makes this an important factor in explaining the spread.

This work studies the emergence of new words in the Portuguese territory, during one year time-span. The interest in this topic has been around the world for several years. In line with our goals, the study reported by [8] describes a quantitative method for identifying new emerging words over a long period of time and then describes the analysis of the lexical emergence on social networks in U.S. territory, based on a universe of millions of word

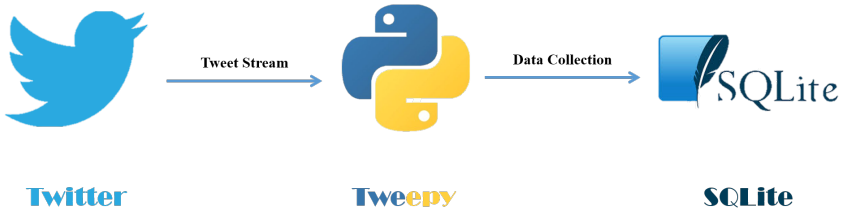


Figure 1 Data collection procedure.

creation, obtained over a period of one year through the internal Twitter API. The study identified 29 words and has examined them from various perspectives, in order to better understand the emergence process. To identify emerging words, two values were inserted into a data matrix, the relative frequency of each word at the beginning of the period and the degree of its frequency throughout that same period. After the list of potential emerging words was generated, several problems were identified: many of these words were names of people, products or company names, which did not meet the objective of the study, so they were manually excluded and 29 words were identified as emerging. The aim of this study was not to identify new words that only emerged during 2013 and 2014, but to identify words that emerged quickly on Twitter in 2014. Nevertheless, they used tools such as *Google Trends*[1] and *Urban Dictionary*[2] to identify the first appearance of the words in question. The analysis did not identify a large number of emerging words and there is a doubt as to whether only those identified will have emerged in Modern American English. A follow up of the above study was conducted by [7]. The same dataset of words was studied in order to *trace* the origin of words identified as *new words*. Through the mapping, five main regions where the appearance of new words is more productive were identified.

2 Corpus

Since tweets are an informal way of communicating, users feel more comfortable expressing themselves, which is a favorable factor for new words to emerge. This study uses a corpus of 7.721 million geolocated tweets, collected between January 2018 and June 2019 in the Portuguese territory, and corresponding to 18 months of data. Twitter allows, through the internal API, to obtain a large amount of data in a short period of time and provides a rich set of metadata, such as the username of the person who produced the content, the number of followers, the geolocation, date, among others. All data was obtained with geolocation information, date and time of publication. It is very common for the location to be obtained when publishing via a mobile phone (with the option of active geotracking), for example on the basis of geolocated data. Such information may be an important resource for finding patterns in the propagation path of a certain emerging word.

The data retrieval procedure is illustrated in Fig. 1. The data was collected directly from the Twitter platform API through python library *tweepy*, which allows access to the RESTful methods of the internal Twitter API. We have then stored the data into a lightweight SQLite database, which offers a reasonable performance, great portability, accessibility and may greatly reduce the complexity of the data analysis. Posts that have been *re-tweeted* have not been excluded from the database since they reflect an evidence of propagation of the emerging word.

[1] www.google.com/trends
[2] www.urbandictionary.com

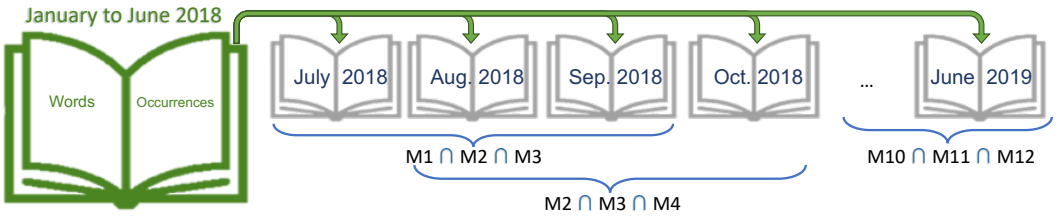


Figure 2 Approach for identifying new words.

3 Approach description

During a pre-processing stage, we have removed a number of tokens irrelevant for this study. Particularly, we have removed all the tokens starting with "*http*" or "@", which correspond to web addresses or user mentions, and all types of numbers. The word tokenization was performed using *TweetTokenizer* from the NLTK library, words with more than 3 repeating letters have also been normalized into a standard form keeping only 3 letters (e.g. the words "looooooove" is converted into "looove").

Our approach is illustrated in Fig. 2. We have started by creating an initial dictionary using the first six months of the data (between January and June 2018), corresponding to 2.9 million tweets. In a second stage we have identified all the possible new words for each of the following 12 months, considering only words appearing at least 50 times, by at least 15 different users. We have considered a minimum frequency of 50 in order to minimize the appearance of spelling errors. Finally, we have assumed that an emergent word would have to be used for at least a period of three consecutive months, so we have calculated the words appearing at least in three consecutive months. Table 1 shows the number of candidate words used at least in 3 consecutive months. It is interesting to notice that the number of words increases as the time period being considered is getting far away from the period used to create the initial dictionary, which suggests that the word usage changes over time.

In order to verify the existence of words in the Portuguese language, we have tested them using the DELAF[3] lexicon, which have all the words of the Portuguese language as well as their flexions, thus allowing to identify which of the emerging words are already in the dictionary of the Portuguese language.

FastText is a free open-source library that allows learning words and classifying text created by the AI (Artificial Intelligence) research laboratory. This library allows training sets of words supervised and unsupervised. The model allows the creation of algorithms to obtain vector word representations. The use of this library is intended to be a mechanism for explaining the origin motive of a given word. We have used two pre-trained word vectors for Portuguese[4], the first one trained on the Common Crawl[5], and the second trained on the Wikipedia[6]. The *fast*Text library has been used, since it is free and lightweight, when compared to other methods to achieve the same accuracy, (supervised) sentence vectors can be easily computed and *fast*Text works better in small datasets when compared with other libraries, such as *gensim*[7].

[3] http://www.nilc.icmc.usp.br/nilc/projects/unitex-pb/web/dicionarios.html
[4] https://fasttext.cc/docs/en/crawl-vectors.html
[5] https://commoncrawl.org
[6] https://www.wikipedia.org/
[7] https://pypi.org/project/gensim/

Table 1 Words shared by different periods of time.

set of months	shared words
2018-07, 2018-08, 2018-09	43
2018-08, 2018-09, 2018-10	70
2018-09, 2018-10, 2018-11	80
2018-10, 2018-11, 2018-12	83
2018-11, 2018-12, 2019-01	94
2018-12, 2019-01, 2019-02	104
2019-01, 2019-02, 2019-03	115
2019-02, 2019-03, 2019-04	129
2019-03, 2019-04, 2019-05	147
2019-04, 2019-05, 2019-06	146

4 Analysis of the results

We have combined all the words achieved in the previous stage, and reported in Fig. 1, into a single dictionary, totaling 496 words and containing all the candidate new words for one year time-span. We have also found that the 5 most prominent tokens are emojis/emoticons, possibly motivated by the appearance of a new application or by an update to an existent one. We have also found that some of the resulting tokens are, in fact, well-known words that were not present in the initial dictionary. That is due to several major factors: a) the initial dictionary, corresponding to the existing knowledge, was created from a very limited set of data; b) the typical vocabulary used in tweets is quite distinct from other sources, such as books or newspapers; c) the content produced in social networks and the corresponding word usage is highly influenced by ongoing events.

Finally, an important factor to take into consideration in order to identify a word as emerging is the number of different users using the word. In order to exclude words which had a reduced number of entries by different users it was required that they were mentioned by at least 15 different people. In fact, a few of the candidate words correspond to a typical situation were a person or a company starts using a given word in an exhausting way (e.g., *Saladumbras*, *Bonetto*), and they were discarded.

Table 2 presents the resulting top candidate tokens, after manually removing some of the entries mentioned above. Many of the identified tokens correspond to names of people, specially those related with soccer (e.g., *Keizer*, *Gudelj*, *Militao*, *Corchia*, *Phellype*, *Castaignos*, and *Manafá*). The remaining tokens were grouped as follows:

Emojis / Emoticons: emoticons, or textual portrayals of a writer's moods or facial expressions in the form of icons, usually used in conjunction with a sentence to express emotions.

Benfiquistão, minguem, Vagandas: these emerging words correspond to derivations of existing ones. The later are used in a very ironic sense, playing around with the name "Varandas", Sporting's President.

kbk: correspond to the formation of new slang words [6], used to abbreviate commonly-used expressions.

Table 2 Emerging words, and their corresponding frequency over 12 months.

token	freq.	comment
phellype	125723	football player
corchia	59514	football player
castaignos	52735	football player
bozo	41613	Brazilian Portuguese word, and English word for stupid man
120M	11786	price paid for a football player
benfiquistão	10381	football coach
trotinetes	9836	the word exists in Portuguese
trotinetas	9785	the same as trotinetes
taki	9710	name of a music (Taki Taki)
militao	9665	Militão is the name of a football player
minguem	9635	Minguém the same as “ninguém” (nobody)
manafá	8594	football player
vagandas	7146	irony for Varandas, Sporting’s President
shallow	5654	name of a music; used an in the English word
sicko	5237	name of a music (sicko mode)
keizer	4147	football coach
kbk	4147	slang for kill or be killed
lomotif	3135	name of an app
legacies	3082	name of a series
gudelj	2824	football player
guaidó	2689	Venezuelan politician
🥺	525	pleading face emoji
🥵	271	hot face emoji
🥳	163	face with party horn and party hat emoji
🥴	125	woozy face emoji
🥶	66	cold face emoji

Table 3 fastText outputs, using word vectors trained using *Wikipedia*.

Phellype	*corchia*	*castaignos*	*bozo*
Wesllem	Corchia	Castaignos	fiuk
Rithelly	Forchia	direntes	buneco
Fellype	Porchia	cosmonômicos	yudi
Sueliton	Torchia	cosmonômico	raxo
Laionel	Acrorchis	castanos	vsf
Douglão	Archia	compotados	veei
Aélson	Torchiara	presssibutraminabepantoldieta	adogo
Ediglê	Erythrorchis	insetrônicos	veii
Neílson	Brasiliorchis	sementeiros	affz
Roniel	Xerorchis	escravinhos	zuei

bozo: despite corresponding to an existing word, it's current true meaning was changed during the time period being studied. It is often used to refer to Bolsonaro, the President of Brazil since 1 January 2019, in a very depreciative way and making use of the phonetic realization of his name as in its Italian origins Bol[z]onaro.

Lomotif: correspond to names of new stores, brands or applications that were introduced meanwhile.

taki, sicko, shallow, legacies: the appearance of these words is related to new music or TV series. *Taki*, *sicko* and shallow both refer to the name of a music (Taki Taki, Sicko Mode and Shallow), while Legacies is the name of a new TV series..

The first four groups correspond to emergent words, now being in use by the Twitter community. While the remaining groups represent new events, usually Named Entities, which may not be considered interesting cases of emerging words.

Using DELAF lexicons, in order to denote which words, represented in Fig. 2, belong in the Portuguese dictionary, it was noticed that only the word "trotinetes" already exists, all the others are new or derivations of existing ones.

The 4 words with the highest number of occurrences were selected to be tested in the *fast*Text library. Tables 3and 4 contain the outputs for the words *Phellype*, *corchia*, *castaignos*, and *bozo*. "Fellype" or "fellype" correspond to a variation of the word *Phellype*, closer to the widely used Portuguese proper noun *Filipe*. Also for the word *corchia*, derivations include the Italian words "Porchia" or "forchia", which may suggest that this word may be originated in Italy. Similarly, the word *castaignos* obtained results in another language, this time in Spanish, which may also suggest that the origin of this word derives from this language. Finally, the fourth word with the most occurrences, "bozo", suggests that it may be related to the word "zombozo". This is the name given to a cartoon portrayed by a clown and cruel with a dark sense of humor from the world of Ben 10, an American animated series, which suggests that the context where the word "bozo" is found has a clown connotation.

It is important to mention that Twitter does not facilitate tracing the entire path of the emerging words, especially because the provided API does not allow to retrieve all the required tweets in a easy way. Despite that, we have analyzed the appearance and most frequent usage of some of the emerging words, paying attention to the regions where the

Table 4 fastText outputs, using word vectors trained using *Common Crawl*.

Phellype	*corchia*	*castaignos*	*bozo*
fellype	forchia	pelignos	zombozo
ype	torchia	castrais	esbozo
mayke	corchiano	malignos	dozo
jaílton	exárchia	intersignos	calabozo
lenílton	vitorchiano	signos	jozo
josimar	torchiara	castrados	kozo
rogerinho	sinerchia	bretaigne	bozomal
raţ	archia	moos	yozo
clebinho	anarchia	montaigne	bozon
neílton	senerchia	châtaigneraie	logozo

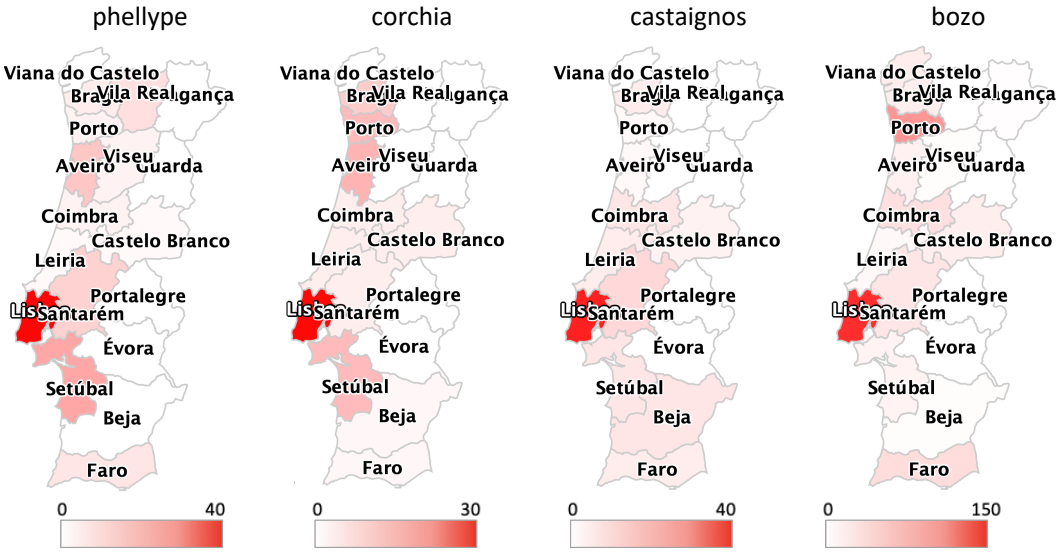


Figure 3 Frequency per district of words: *Phellype*, *corchia*, *castaignos*, and *bozo*.

word appeared first. Figure 3 shows the mapping of lexical innovation of the 4 words with the highest number of occurrences in the dataset that were identified as emerging words. Regions marked with a darker color correspond to the places where each word was more frequently used during the period in analysis. The figure reveals that they appear in urban areas where population density is higher and Internet is more frequently available [12], as expected.

We believe our study has made four general contributions:

- It is possible to observe regional patterns of word spread, even if it is not possible to affirm that the words occurred for the first time on social media.
- Words tend to follow a consistent path, as it is possible to see in Figure 3 that Lisbon will be the city with the center of propagation, registering a spread to the surrounding cities, with a natural decrease in the number of occurrences of the words.
- Population density has an important role in the spread of words and appears to be more fundamental than cultural or religious issues.
- Brazilian Portuguese will be one of the main sources of lexical innovation.

Twitter is only a source of virtual expression, which does not allow the generalization of results for the majority of the Portuguese population and presumably the words did not occur online for the first time. A corpus allows to partially detect patterns of lexical innovation in a language, especially in a modern world where virtual communications occur in the most varied platforms. However, since a large part of the words in twitter cover a common discourse we believe that the results achieved reflect a rather realistic scenario.

5 Conclusions and future work

We have presented an approach that uses a corpus containing a considerable number of tweets to detect the appearance of possible emerging words. We have applied our approach to tweets written in Portuguese, and produced in the Portuguese region over 18 months. Despite the limitations of our corpus, the proposed approach led to the discovery of a number of relevant linguistic phenomena in the achieved set of candidate words. The final set of possible emerging words was achieved by performing a manual analysis and selection of the retrieved words. This preliminary study has provided a methodological framework for future research in the field of neology. Our findings suggest that social networks, and Twitter in particular due to its nature, are a promising way of studying the dynamics of a language.

As a follow-up for the present study, we plan to use other sources of information, such as newspapers, blogs, and other social networks as means to trace a complete path of the emerging words, thus contributing to the better understanding of the evolution of the vocabulary in a given language. We plan to better characterize each word by providing an extended analysis over time, by mapping its usage, adoption by communities, and mapping its propagation path. We realize that there are no standard steps to address this issue, but most data sources are now providing geolocation and time information, which constitute a relevant advantage for tracing reliable geo-temporal propagation paths of a word in the near future.

References

1 David Bamman, Jacob Eisenstein, and Tyler Schnoebelen. Gender identity and lexical variation in social media. *Journal of Sociolinguistics*, 18(2):135–160, 2014. doi:10.1111/josl.12080.

2 Rebecca Blood. Weblogs: A history and perspective, September 2000. URL: http://www.rebeccablood.net/essays/weblog_history.html.

3 Charles Boberg. Geolinguistic diffusion and the U.S. - Canada border. *Language Variation and Change*, 12:1–24, March 2000. doi:10.1017/S0954394500121015.

4 Marilyn Dyrud, Rebecca Worley, and Marie Flatley. Blogging for enhanced teaching and learning. *Business Communication Quarterly*, 68, March 2005. doi:10.1177/108056990506800111.

5 Alexandre François. Trees, Waves and Linkages: Models of Language Diversification. In Claire Bowern and Bethwyn Evans, editors, *The Routledge Handbook of Historical Linguistics*, chapter Trees, Waves and Linkages: Models of Language Diversification, pages 161–189. Routledge, London, June 2014. doi:10.4324/9781315794013.ch6.

6 Jonathon Green and David Kendal. Writing and publishing green's dictionary of slang. *Dictionaries: Journal of the Dictionary Society of North America*, 38:82–95, January 2017. doi:10.1353/dic.2017.0003.

7 Jack Grieve. Dialect variation. In Douglas Biber and RandiEditors Reppen, editors, *The Cambridge Handbook of English Corpus Linguistics*, Cambridge Handbooks in Language and Linguistics, pages 362–380. Cambridge University Press, Cambridge (UK), 2015. doi:10.1017/CBO9781139764377.021.

8 Jack Grieve, Andrea Nini, and Diansheng Guo. Analyzing lexical emergence in modern american english online. *English Language and Linguistics*, 21(1):99–127, 2017. doi:10.1017/S1360674316000113.

9 Stefan Grondelaers, Dirk Geeraerts, and Dirk Speelman. Lexical variation and change. In Dirk Geeraerts and Hubert Cuyckens, editors, *The Oxford Handbook of Cognitive Linguistics*, pages 988–1011. Oxford University Press, 2012. doi:10.1093/oxfordhb/9780199738632.013.0037.

10 Jeremy Harmer. The Practice of English Language Teaching. *SERBIULA (Sistema Librum 2.0)*, January 2001.

11 Sara Kajder, Glen Bull, and Emily Van Noy. A space for "writing without writing.". *Learning and Leading with Technology*, 31:32–35, 2004. URL: https://files.eric.ed.gov/fulltext/EJ695756.pdf.

12 T. Lapa, Jorge Vieira, J. Azevedo, and G. Cardoso. As desigualdades digitais e a sociedade portuguesa: divisão, continuidades e mudanças. In *Desigualdades Sociais: Portugal e a Europa*, pages 257–257. Mundos Sociais, Lisboa, 2018. URL: http://www.mundossociais.com/livro/desigualdades-sociais/112.

13 M Lynne Murphy. Theories of lexical semantics by Dirk Geeraerts. *Journal of Linguistics*, 47:231–236, January 2011. doi:10.2307/41261748.

14 Dilip Mutum and Qing Wang. Consumer generated advertising in blogs. In M.S. Eastin, T. Daugherty, and N. Burns, editors, *Handbook of Research on Digital Media and Advertising: User Generated Content Consumption*, chapter 13, pages 248–261. IGI Global, 2010. doi:10.4018/978-1-60566-792-8.ch013.

15 John Nerbonne. Measuring the diffusion of linguistic change. *Philosophical Transactions of the Royal Society B: Biological Sciences*, 365(1559):3821–3828, 2010. doi:10.1098/rstb.2010.0048.

16 João Pedro Pereira. Era uma vez o Twitter em Portugal. *Público*, 77(3):95–106, 2016.

17 S.M. Shahid. Teaching of English an Introduction. *Majeed Book Depot Urdu Bazar Lahore*, 2002.

18 Peter Trudgill. Linguistic change and diffusion: Description and explanation in sociolinguistic dialect geography. *Language in Society*, 3(2):215–246, 1974. URL: http://www.jstor.org/stable/4166764.

BhTSL, Behavior Trees Specification and Processing

Miguel Oliveira
Centro Algoritmi (CAlg-CTC), Department of Informatics, University of Minho, Braga, Portugal
miguelmigpt@gmail.com

Pedro Mimoso Silva
Centro Algoritmi (CAlg-CTC), Department of Informatics, University of Minho, Braga, Portugal
pedro.miguel.mimoso@gmail.com

Pedro Moura
Centro Algoritmi (CAlg-CTC), Department of Informatics, University of Minho, Braga, Portugal
pedrorpmoura@gmail.com

José João Almeida
Centro Algoritmi (CAlg-CTC), Department of Informatics, University of Minho, Braga, Portugal
jj@di.uminho.pt

Pedro Rangel Henriques
Centro Algoritmi (CAlg-CTC), Department of Informatics, University of Minho, Braga, Portugal
pedrorangelhenriques@gmail.com

Abstract

In the context of game development, there is always the need for describing behaviors for various entities, whether NPCs or even the world itself. That need requires a formalism to describe properly such behaviors. As the gaming industry has been growing, many approaches were proposed. First, finite state machines were used and evolved to hierarchical state machines. As that formalism was not enough, a more powerful concept appeared. Instead of using states for describing behaviors, people started to use tasks. This concept was incorporated in behavior trees. This paper focuses in the specification and processing of Behavior Trees. A DSL designed for that purpose will be introduced. It will also be discussed a generator that produces LaTeX diagrams to document the trees, and a Python module to implement the behavior described. Additionally, a simulator will be presented. These achievements will be illustrated using a concrete game as a case study.

2012 ACM Subject Classification Theory of computation → Tree languages

Keywords and phrases Game development, Behavior trees (BT), NPC, DSL, Code generation

Digital Object Identifier 10.4230/OASIcs.SLATE.2020.4

Funding This work has been supported by FCT–Fundação para a Ciência e Tecnologia within the R&D Units Project Scope: UIDB/00319/2020.

1 Introduction

At some point in the video-game history, NPCs (Non-Playable Characters) were introduced. With them came the need to describe behaviors. And with these behaviors came the need of the existence of a formalism so that they can be properly specified.

As time passed by, various approaches were proposed and used, like finite and hierarchical state machines. These are state-based behaviors, that is, the behaviors are described through states. Altough this is a clear and simplistic way to represent and visualize small behaviors, it becomes unsustainable when dealing with bigger and more complex behaviors. Some time later, a new and more powerful concept was introduced: using tasks instead of states to describe behaviors. This concept is incorporated in what we call behavior trees.

Behavior Trees (`BT` for short) were first used in the videogame industry in the development of the game *Halo 2*, released in 2004 [5]. The idea is that people create a complex behavior by only programming actions (or tasks) and then design a tree structure whose leaf nodes are actions and the inner nodes determine the NPC's decision making. Not only these provide an easy and intuitive way of visualizing and designing behaviors, they also provide a good way to work with scalability through modularity, solving the biggest issue from state-based design. Since then, multiple gaming companies adopted this concept and, in recent years, behavior trees are also being used in different areas like Artificial Inteligence and Robotics.

In this context, we felt that it could be useful to have a DSL to specify BTs independently of application area and the programming language chosen for the implementation. The language must be compact and easy to use but it should be expressive enough to be applied to real situations. In that sense a new kind of node was included, as will be described.

This paper will introduce the DSL designed and the compiler implemented to translate it to a programming language, in this case Python. Additionally, the compiler also generates LaTeX diagrams to produce graphical documentation for each BT specified.

A small example will be described in our language as a case study to illustrate all the achievements attained.

The paper is organized as follow: Concepts and State of the Art frameworks are presented in Section 2. Architecture and language specification are proposed in Section 3. Compiler development is discussed in Section 4. An illustrative case study is presented in Section 5, before concluding the paper in Section 6. The paper also includes one appendix that contains the tokens table.

2 Concepts

This section will be built based on references [1, 4, 3].

Formally, a BT is a tree whose internal nodes are called control flow nodes and leafs are called execution nodes.

A behavior tree executes by peridiocally sending ticks to its children, in order to traverse the entire tree. Each node, upon a tick call, returns one of the following three states to its parent: `SUCCESS` if the node was executed with success; `FAILURE` if the execution failed; or `RUNNING` if it could not finish the execution by the end of the tick. In the last case, the next tick will traverse the tree until it reaches the running execution node, and will try again to run it.

2.1 Control Flow Nodes

Control flow nodes are structural nodes, that is, they do not have any impact in the state of the system. They only control the way the subsequent tree is traversed. In the classical formulation, there are 4 types of control flow nodes: **Sequence**, **Selector**, **Parallel** and **Decorator**. Even if not standard we use decorators as control flow nodes, according to [1]. A sequence node (figure 1a) visits its children in order, starting with the first, and advancing for the next one if the previous succeeded. Returns:

- `SUCCESS` – if all children succeed;
- `FAILURE` – if a child fails;
- `RUNNING` – if a child returns `RUNNING`.

Like the sequence, the selector node (figure 1c) also visits its children in order, but it only advances if the child that is being executed returns `FAILURE`. Returns:

- `SUCCESS` – if a child succeeds;
- `FAILURE` – if all children fails;
- `RUNNING` – if a child returns `RUNNING`.

A parallel node (figure 1e), as the name implies, visits its children in parallel. Additionally, it has a parameter M that acts as a success rate. For N children and $M \leq N$, it returns:

- `SUCCESS` – if M children succeed;
- `FAILURE` – if $N - M + 1$ children fail;
- `RUNNING` – otherwise.

A decorator (figure 1b) is a special node that has an only one child, and uses a policy (set of rules) to manipulate the return status of its child, or the way it ticks it. Some examples of decorator nodes are:

1. **Inverter** – inverts the `SUCCESS`/`FAILURE` return status of the child;
2. **Max-N-Times** – the child can only fail N times. After that it only returns `FAILURE` without ticking the child.

2.2 Execution Nodes

Execution nodes are the simplest, yet the most powerful. They are the ones that have access to the state of the system, and can update it. There are two types of execution nodes: **Action** and **Condition**.

Upon the execution of a tick, an action node (figure 1d) runs a chunk of code that can return either `SUCCESS`, `FAILURE` or `RUNNING`.

The condition node (figure 1f) verifies a proposition, returning `SUCCESS`/`FAILURE` if the proposition is/is not valid. This node never returns `RUNNING`.

2.3 Control Flow Nodes with memory

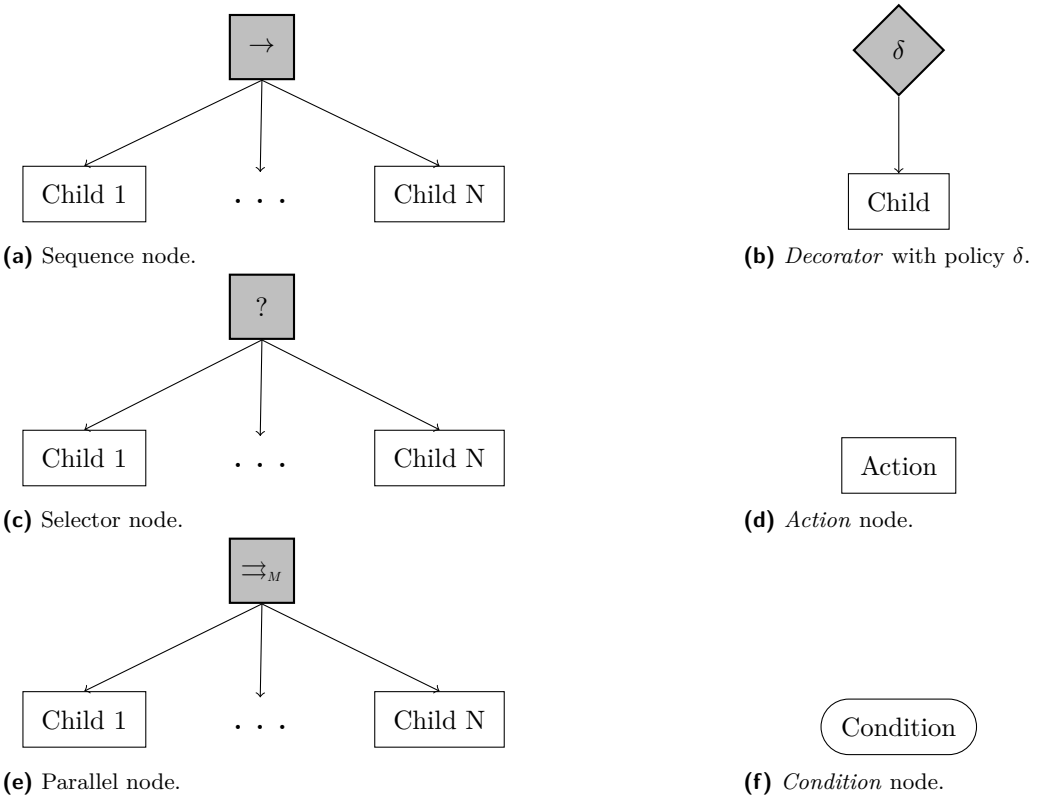
Sometimes, when a node returns `RUNNING`, we want it to remember which nodes he already executed, so that the next tick does not execute them again. We call this nodes with memory. And they are represented by adding a $\_^*$ to the symbols mentioned previously. This is only syntactic sugar because we can also represent these nodes with a non-memory BT, but that will not be discussed here.

Please note that, while we avoid the re-execution of nodes with this type of node, we also lose the reactivity that this re-execution provides.

2.4 State of The Art

In the gaming industry there is some interesting projects that use tools based on Behavior trees as the main focus to describe NPCs behaviors. Unreal Engine [2] and Unity[1] are two examples of major game engines that use them. In their case, instead of a language, they offer a graphical user interface (GUI) to specify the BTs, through a drag and drop tactic. Upon the creation of an execution node, the programmer needs to specify the action or condition that will be executed. The nodes mentioned before are all implemented in these engines, along with some extensions. All the nodes that were mentioned before are implemented in both of these engines, along with some extensions.

[1] `https://unity.com`



(a) Sequence node.

(b) *Decorator* with policy δ.

(c) Selector node.

(d) *Action* node.

(e) Parallel node.

(f) *Condition* node.

Figure 1 BT Nodes structure.

In addition to game engines, there are also frameworks like Java Behavior Trees[2] for Java and Owyl[3] for Python that implement BTs. In this case, they work as a normal library.

3 Architecture and Specification

In this section, it will be explained the general architecture of our system to process BTs, that is depicted in Figure 2. After introducing its modules, one subsection is devoted to the BhTSL domain specific language design.

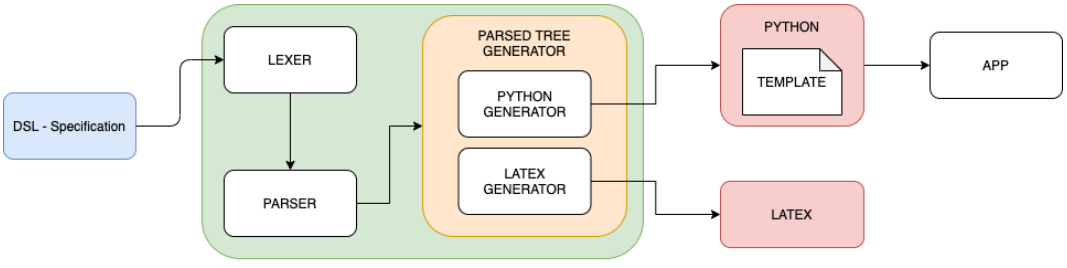


Figure 2 System Architecture.

[2] https://github.com/gaia-ucm/jbt
[3] https://github.com/eykd/owyl

The input for our system, `DSL - Specification`, is a text file describing the behavior which should follow the language syntax. The compiler, which is represented as the green rounded rectangle in the diagram of Figure 2, is composed of the following modules: a `Lexer`, a `Parser` and a `Code Generator`.
This generator has two sub-generators. The `Latex Generator`, that is responsible for the generation of the LaTeX code to draw the tree diagram representing the behavior specified. And the `Python Generator`, that produces the fragment of Python code that implements the desired behavior according to a template predefined by us in the context of this project; that code fragment can be later imported by any Python application that aims to.

3.1 BhTSL

Before we start describing the DSL, we will introduce a new node, called **Probability Selector** (Figure 3 depicts that concept), that provides us with a relevant extension to the standard formalism for a more powerful behavior specification. This extension improves the expressiveness of BhTSL language.

A probability selector node is like a normal selector node, but instead of visiting its children from left to right, it visits them randomly, taking into account that each child has a probability, defined by the user, of being chosen first.

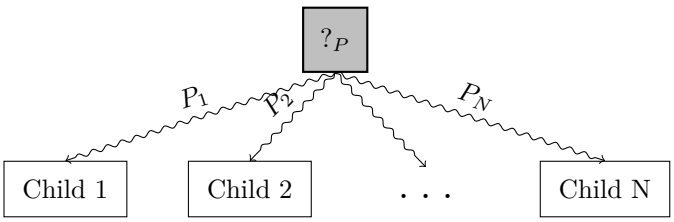


Figure 3 Probability Selector node.

Example

Now that all nodes have been introduced, let us see an example of a specific behavior.

Suppose that in some game, called `TGame`, it is intended to have a *guard that patrols a house*. The guard has the following behavior: while he is patrolling, if he sees the player, activates an alarm and then, depending on the level of courage he has, decides (based on probabilities) whether he runs away or fights the player. In case of running away, he constantly checks if he still sees the player, returning to patrolling in case he does not. If he still sees it, he keeps running. The same thing happens when he chooses to fight the player, only this time he checks if the player is already dead or not.

In Figure 4, we can see a possible tree specification for this behavior.

3.1.1 Syntax

In our language, each specification represents one and only one behavior. An input file, containing the behavior specification text, is divided into 3 components:

- *Behavior* – main behavior tree;
- *Definitions* (optional) – node definitions that can be referenced in other nodes or in the main BT;
- *Code* – Python block that contains the code fragments described the execution nodes, and other code that the programmer wishes to add.

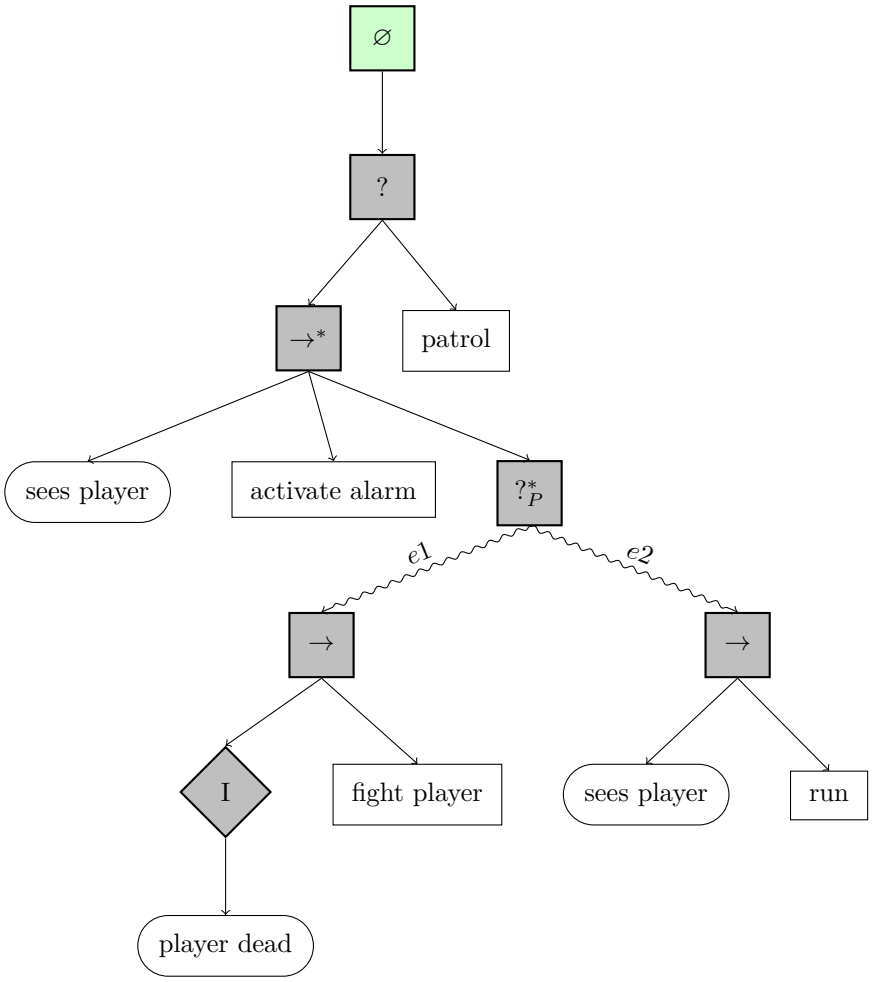


Figure 4 Example: `TGame` behavior tree diagram automatically generated by our tool.

To illustrate our idea about the DSL we plan to design (formally defined by a grammar in Section 4), we present below an example of a specification written in the intended language.

```
behavior : [
    sequence : [
        condition : $cond1,
        condition : $cond2
        memory selector : [
            parallel : $par1,
            prob_selector : $prob1
        ]
    ]
]

parallel par1 : 10 [
    action : $action1,
    action : $action2
]

prob_selector prob1 : [
```

```
    $e1 -> decoraror : INVERTER [
        action : $action1
    ],
    $e2 -> action : $action2
]

%%

def action1(entity):
    pass

def action2(entity):
    pass

def cond1(entity):
    pass

def cond2(entity):
    pass

def e1(entity):
    pass

def e2(entity):
    pass
```

4 Tool development

In the next subsection the implementation of the BhTSL processor will be detailed, as well as the language specification will be presented.

4.1 Lexical analysis

The first step in the development of a compiler is the lexical analysis, that converts a char sequence into a token sequence. The tokens table can be seen in Appendix A.

4.2 Syntatic analysis

Syntatic analysis, or parsing, it the process of analyzing a string of symbols conforming the rules of a grammar.
Below we list the context free grammar that formally specifies BhTSL syntax:

```
    root : behavior CODE
         | behavior definitions CODE
         | definition behavior CODE

    behavior : BEHAVIOR ':' '[' node ']'

    node : SEQUENCE ':' '[' nodes ']'
         | SEQUENCE ':' VAR
         | MEMORY SEQUENCE ':' '[' nodes ']'
         | MEMORY SEQUENCE ':' VAR
         | SELECTOR ':' '[' nodes ']'
```

```
        | SELECTOR ':' VAR
        | MEMORY SELECTOR ':' '[' nodes ']'
        | MEMORY SELECTOR ':' VAR
        | PROBSELECTOR ':' '[' prob_nodes ']'
        | PROBSELECTOR ':' VAR
        | MEMORY PROBSELECTOR ':' '[' prob_nodes ']'
        | MEMORY PROBSELECTOR ':' VAR
        | PARALLEL ':' INT '[' nodes ']'
        | PARALLEL ':' VAR
        | DECORATOR ':' INVERTER '[' node ']'
        | DECORATOR ':' VAR
        | CONDITION ':' VAR
        | ACTION ':' VAR

    nodes : nodes ',' node
          | node

    prob_nodes : prob_nodes ',' prob_node
               | prob_node

    prob_node : VAR RIGHTARROW node

    definitions : definitions definition
                | definition

    definition : SEQUENCE NODENAME ':' '[' nodes ']'
        | SELECTOR NODENAME ':' '[' nodes ']'
        | PROBSELECTOR NODENAME ':' '[' prob_nodes ']'
        | PARALLEL NODENAME ':' INT '[' nodes ']'
        | DECORATOR NODENAME ':' INVERTER '[' node ']'
```

4.3 Semantic analysis

As usual, from a static semantics perspective, the compiler will check the source text for non-declared variables and variable redeclaration.
A variable can only be accessed if it is declared, either in the *definitions* section (if it represents a control flow node), or in the *code* section (execution node). Additionally, a variable can only be declared once, to avoid ambiguity in the memory access by the processor.

The dynamic semantics is discussed in the next subsection.

4.4 Code generator

The compiler can generate two different outputs: a LaTeX file, that contains the LaTeX commands to draw a diagram for the BT specified; and a Python file, that contains the functions that implement the specified behavior.

The Python file is built using a **Template** file which contains markers that the **Code Generator** will fill with the processed data. This file also contains the class **Simulator** which is capable of executing the processed BT.

Due to the lack of time available, we have only been able to generate Python code, but we hope to be able to compile into different languages in the future.

4.5 BT Simulator

In order to test the Python code generated and to help the BT specifier to debug the behavior he is willing to describe, we also developed an interpreter that imports the Python behavior file and simulates its execution. This additional tool proved to be useful.

4.6 Implementation

The project was developed using `Python`. We chose this language due to our previous experience with it, but also due to its simplicity, flexibility and its cutting edge technology; it is also relevant to be a reflective language.

To automatically generate the compiler from the tokens and grammar specifications, we used the `PLY (Python Lex-Yacc)`[4] library, which is an implementation of the `lex` and `yacc` lexer and parser generator tools for Python. We chose to use PLY because we had prior experience with Lex-Yacc which enabled us to save time to implement the project. Moreover, we realized that the specification is lighter than an equivalent using attribute grammars and because the code produced by those generators is really efficient.

To implement the **Code Generator** module, we resorted to the well-known tree-traversal approach, that upon visiting every node of the parsed tree, produces the corresponding output. For that purpose, some standard libraries were used.

Additionally, we created a LaTeX library, `behaviortrees.sty`, to draw the trees specified. This library is used in the LaTeX generator.

5 Case Study

In order to test our tool, we designed a simple game that consists of an entity finding and grabbing a ball in a generated map. This entity has a range of vision that it is used to search the ball. When the entity finds it, it approaches the ball and, if it is within reach, grabs it.

The following code is an example of a specification for this behavior in our DSL, and Figure 5 depicts the behavior tree automatically generated by our tool.

```
behavior : [
    selector : [
        memory sequence : $seq1,
        action : $search_ball
    ]
]

sequence seq1 : [
    condition : $ball_found,
    selector : [
        sequence : [
            condition : $ball_within_reach,
            action : $grab_ball
        ],
        action : $approach_ball
    ]
]
```

[4] `https://www.dabeaz.com/ply/`

```
%%

def ball_found(player):
    return player.ball_found


def ball_within_reach(player):
    return player.ball_within_reach


def grab_ball(player):
    player.grab_ball()
    return SUCCESS


def approach_ball(player):
    player.approach_ball()
    return RUNNING


def search_ball(player):
    player.search_ball()
    return RUNNING
```

Now that we have the specification, we can generate the `Python` file to use in our code. Suppose that the generated file's name is `behavior.py`, we can import it by writing `import behavior`. With this, we can make use of the **Simulator** class to execute the behavior.

Below we show the code that exemplifies how to use this class to run the behavior.

```
game = Game()
game.setup()
S = behavior.Simulator(game.player1)

game.render(screen)

while True:

    game.process_events()

    S.tick()
    game.update(clock, 2)
    game.render(screen)
```

According to the specification, our entity can perform 3 different actions: search for the ball, approach the ball, and grab the ball.

Searching for the ball occurs in an initial moment, in which the entity does not know yet where the ball is. Figure 6 depicts an example of that moment, where the entity is the blue square and the ball is the red square.

Until it finds the ball, the entity roams freely through the map.

When it finds it, unless it is within arms reach, the entity will approach it. Figure 7 depicts the moment when the entity found the ball.

Lastly, when the ball is within reach, the entity grabs it, as it is shown in Figure 8.

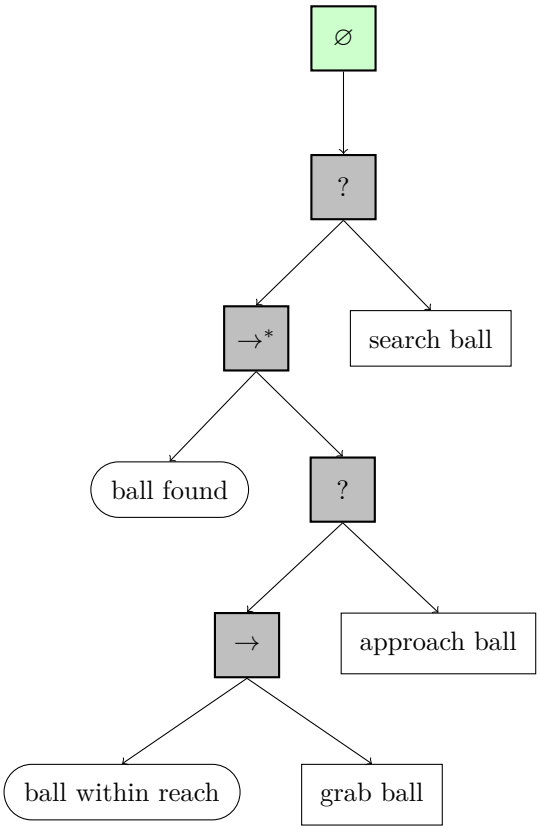


Figure 5 Behavior Tree generated from the case study.

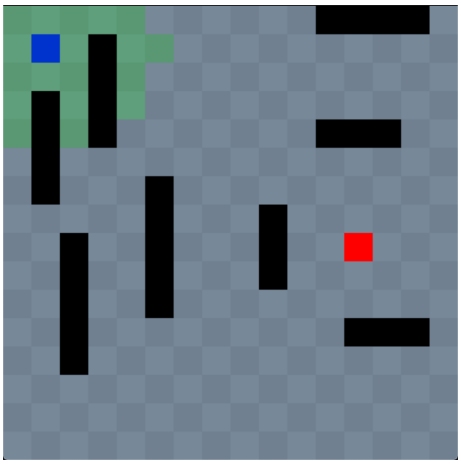


Figure 6 Searching for the ball.

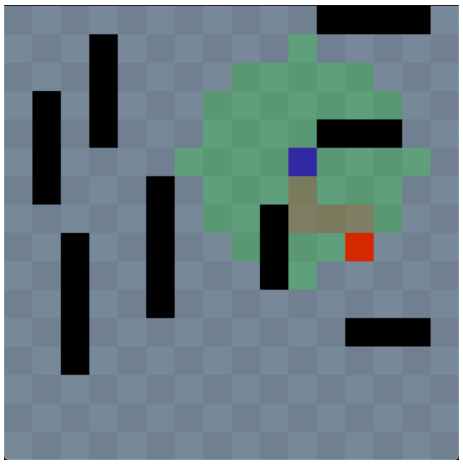


Figure 7 Approaching the ball.

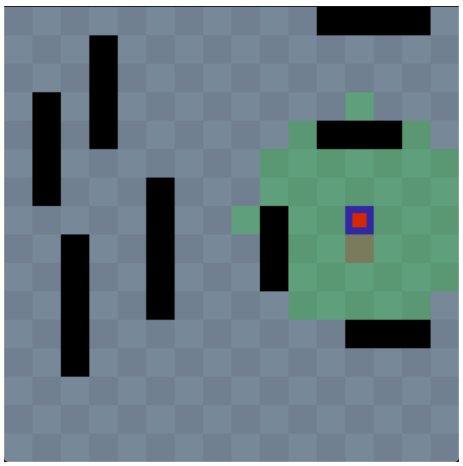


Figure 8 Grabbing the ball.

6 Conclusion

As games industry is growing significantly every day, the need for a formal way to describe behaviors is also increasing requiring more and more expressiveness keeping it easy to learn, to use and to understand. After some initial attempts not powerful enough, a new approach called Behavior Trees (BT) appeared. This paper describes a project in which we are working on, aimed at designing a DSL to write BT and developing the respective compiler to generator Python functions to be incorporated in final Python programs created to implement games or other kind of applications.

Along the paper the DSL designed, called BhTSL, was introduced by example and specified by a context free grammar. The architecture of the BhTSL processor was depicted and discussed, and the development of the compiler that produces the Python code library was described. In that context an example of a game specification was presented and the LaTeX fragment that is generated to draw the BT was shown.

Although not detailed or exemplified, the simulator developed to help on debugging the BT specified in BhTSL language was mentioned along the paper.

6.1 Future Work

As future work we intend to implement the generation of code for other programming languages, such as Java and C++ so that it can be widely used by the game development community. This should be a fairly standard procedure due to our usage of templates on the **Code Generation** stage of our program.

References

1 Michele Colledanchise and Petter Ogren. *Behavior Trees in Robotics and AI: An Introduction*. Chapman & Hall/CRC Press, July 2018. doi:10.1201/9780429489105.

2 Epic-Games. Behavior trees, 2020. Accessed: 2020-05-21.

3 Ian Millington and John Funge. *Artificial Intelligence for Games*. Morgan Kaufmann Publishers, January 2009. doi:10.1201/9781315375229.

4 Chris Simpson. Behavior trees for ai: How they work, 2014. Accessed: 2020-05-21.

5 Guillem Travila Cuadrado. Behavior tree library, 2018. Universitat Politècnica de Catalunya, Bachelor Thesis.

A Tokens Table

The following table displays the full set of tokens of BhTSL language defined in terms of regular expressions (REs) as utilized in our compiler.

Table 1 BhTSL Tokens Table for Lexical Analysis.

Tokens	
Name	**Value**
literals	`([]),:%`
RIGHTARROW	`->`
BEHAVIOR	`\bbehavior\b`
SEQUENCE	`\bsequence\b`
SELECTOR	`\bselector\b`
PROBSELECTOR	`\bprobselector\b`
PARALLEL	`\bparallel\b`
DECORATOR	`\bdecorator\b`
CONDITION	`\bcondition\b`
ACTION	`\baction\b`
INVERTER	`\bINVERTER\b`
MEMORY	`\bmemory\b`
INT	`\d+`
VAR	`$\w+`
NODENAME	`\b\w+\b`
CODE	`%%(.|\n)+`

DAOLOT: A Semantic Browser

João Bruno Silva
Faculty of Sciences, University of Porto, Portugal
up201406063@fc.up.pt

André Santos
CRACS & INESC Tec LA, Porto, Portugal
Faculty of Sciences, University of Porto, Portugal
afs@inesctec.pt

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

Abstract

The goal of the Semantic Web is to allow the software agents around us and AIs to extract information from the Internet as easily as humans do. This semantic web is a network of connected graphs, where relations between concepts and entities make up a layout that is very easy for machines to navigate.
At the moment, there are only a few tools that enable humans to navigate this new layer of the Internet, and those that exist are for the most part very specialized tools that require from the user a lot of pre-existing knowledge about the technologies behind this structure. In this article we report on the development of DAOLOT, a search engine that allows users with no previous knowledge of the semantic web to take full advantage of its information network. This paper presents its design, the algorithm behind it and the results of the validation testing conducted with users. The results of our validation testing show that DAOLOT is useful and intuitive to users, even those without any previous knowledge of the field, and provides curated information from multiple sources instantly about any topic.

2012 ACM Subject Classification Information systems → Web searching and information discovery; Information systems → Resource Description Framework (RDF)

Keywords and phrases Semantic, Web, Named Entities, Search, SPARQL, RDF, COMUNICA

Digital Object Identifier 10.4230/OASIcs.SLATE.2020.5

Funding *André Santos*: Ph. D. Grant SFRH/BD/129225/2017 from Fundação para a Ciência e Tecnologia (FCT), Portugal.
José Paulo Leal: Fundação para a Ciência e a Tecnologia (FCT), within project UIDB/50014/2020.

1 Introduction

The end of the last decade, in technological terms, was marked by an expansion of the Internet beyond human users. Looking at this trend, we can say that we are entering the era of an Internet shared with machines, where programs are able to browse the Internet independently of humans. However, the Internet was not designed to be used by autonomous machines. When entering a web page, the relationship between the data found there and other links may be obvious to us, but a machine cannot understand the relationship between several different pages: it only sees a list of links.

The semantic web[1] was designed to solve this problem: an extension to the normal Internet that allows web applications to navigate a network of interconnected data, just as one would navigate links on the normal Internet. In the semantic web, instead of a

network of links, the Internet is presented as a huge interconnected graph of information, where the nodes represent resources or properties, and the arrows represent the relationships between them.

This information is stored in RDF, a format in which the atomic information unit is called a triple[8]. A triple has the format of **Subject → Predicate → Object**, and these triples are what allow the semantic web to specify connections between entities and concepts, forming a knowledge graph.

The applications designed to explore these graphs are called semantic browsers. They are meant to help the user taking advantage of the semantic web to search for information.

The semantic browsers that currently exist and are mostly academic tools designed specifically for research, built for very specific purposes, and are mostly impossible to use without pre-existing technical knowledge.

The aim of the project presented in this article is to create a semantic search engine that is available and accessible to the general public, available to all desktop and mobile devices, which allows a human user to easily search and compare results in a simple and intuitive way. The browser developed for this goal is named DAOLOT. It consists of a web page which retrieves information from multiple endpoints, and processes this information as it arrives to be easily understood by the user. This article describes the current state of semantic browsers, followed by an overview of DAOLOT's architecture and the validation testing used to improve it.

This article consists of a brief introduction to the concept of the Semantic Web and the goals of DAOLOT, followed by a assessment of the existing semantic browsers, their similarities and limitations.

The following section details the architecture and algorithm behind DAOLOT, followed by the validation section, which details the tests conducted with users and the feedback which was used to improve DAOLOT.

Finally, the conclusion sums up some of the difficulties that we had during development and the implications of DAOLOT in the future of the Semantic Web.

2 State-of-the-Art

Semantic Browsers are browsers built to navigate semantic knowledge graphs, which are databases of information stored in the RDF format, and extract information from them. The RDF format is the basis of the semantic web, and it stores information in atomic structures called triples, which specify relations between entities in the format **Subject → Predicate → Object**. A group of triples referencing the same entities form a knowledge graph, as illustrated in the following image.

The existing semantic browsers are specialized tools, built mostly as either academic research or commercial use tools. The majority of semantic browsers are designed for navigating local RDF databases, not for retrieving information from existing remote endpoints.

Those that are designed to retrieve information from other endpoints require the user to type his queries in the SPARQL format[9, 3], a querying language made to retrieve information from semantic databases, requiring technical knowledge from the user. This section surveys some of the semantic browsers described in the literature, considering both their similarities to DAOLOT and their limitations:

/Facet

/Facet is a semantic browser designed to browse *RDF* repositories[7] located on web servers, entirely based on SWI Prolog, a development environment in the Prolog programming language designed to handle the *RDF* format.

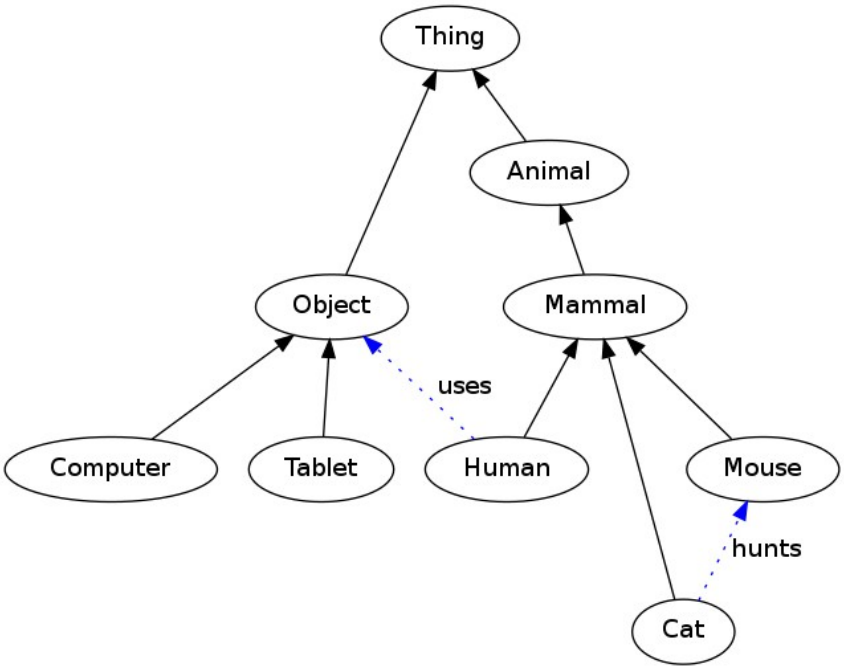


Figure 1 A visual representation of a knowledge graph[5].

Developed in 2006, /facet is one of the first semantic browsers to have been published, having since then become part of a larger browser called *ClioPatria*, which is also no longer functional.

/Facet allows to graphically navigate a graph as if it were a tree of files, and even allows searching for a *keyword*, with the addition of a graphic illustrator for temporal data.

The biggest limitation of this browser is the fact that it only allows the simple search for one argument, and, because it only works with local *datasets*, it greatly limits its usefulness for the general public.

Another diference of this system from DAOLOT is the presentation of the search results, which are presented at the end of the semantic tree of which they are part, requiring the user to scroll the graph up to the sheet for each individual result of the *query*.

Mspace

Mspace is a semantic browser developed in the University of Southampton, US, and consists of an interface and framework designed to explore and manipulate information in semantic datasets[6, 10]. Its interface system is based on columns, which present each property and possible values.

Although Mspace has a more complete search functionality than existing semantic browsers, it remains simply based on a rudimentary keyword search, and the interface is unnecessarily complex, while also having no visualization tools for aggregating data.

OpenLink Data Explorer

OpenLink Data Explorer is a browser that works as *add-on* for conventional web browsers like Chrome or Firefox.

Its main feature is to be able to extract metadata from any resource on the Internet at the user's command.

Although functional, this engine only gives us the *RDF* meta-data that describes the resource, always depending on manual search, and also does not offer any functionality that facilitates the navigation of this data.

Browser	/Facet	Mspace	Openlink data explorer
GUI	Yes	Yes	Yes
Search by keywords	Limited	Yes	No
Data browsing	Limited	Yes	No
Graphical representation of data	Yes	No	No
Access to the semantic web	No	Yes	Yes
Data analysis	No	No	No
Custom endpoints	Yes	No	No

The table above compares the features and limitations of each of the semantic Browsers mentioned:

There are a few more semantic browsers out there, but these are representative of the semantic browsers that exist at the moment and the current limitations that DAOLOT aims to correct.

3 System Overview

The DAOLOT search engine[1] is a responsive web page, available for desktop and mobile devices.

DAOLOT transforms the user-typed search term into a SPARQL query, and uses asynchronous requests to send it to several semantic endpoints. As the information arrives, it compiles and curates the results to minimize redundancy and maximize usefulness.

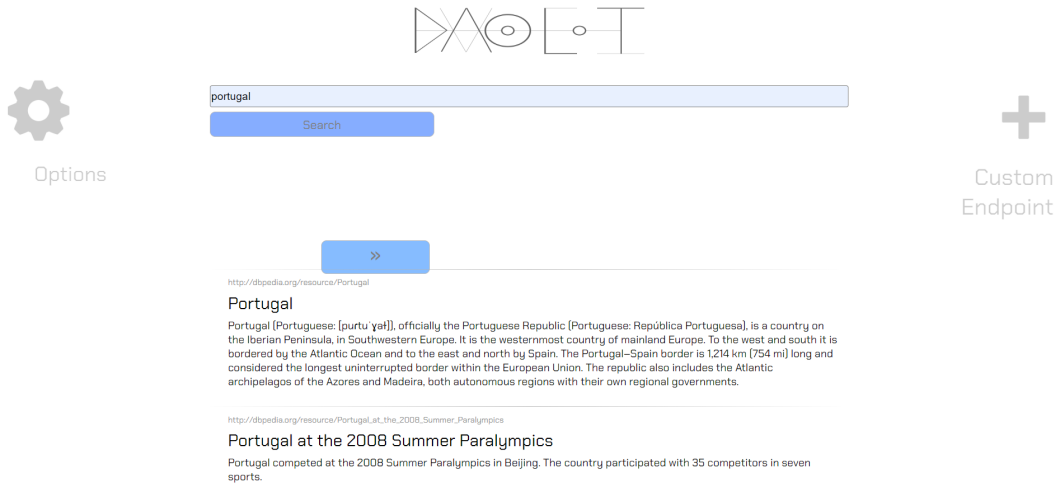


Figure 2 The DAOLOT Interface.

The inputs available to the user, presented in Figure 2, are:

The Search Bar

Allows the user to input his query;

The Options Menu

A pop-up menu on the side allows the user further control over the search parameters,

[1] DAOLOT: `https://daolot.dcc.fc.up.pt/`

such as the number of results to display, language, sources to be included in the search and even a manual mode for advanced users, which allows to directly write the SPARQL query to be sent;

Custom Endpoint Menu

This menu allows the user to add endpoints not included in the default selection of DAOLOT. Before adding the endpoint, DAOLOT runs a "compatibility check" on the endpoint, checking whether it responds to request and if it recognizes standard RDF ontologies which are essential for DAOLOT. Examples of such ontologies are the RDFS vocabulary[2], which specifies basic types of predicates such as `rdfs:comment` (a small summary about the subject), or the Web Ontology Language (OWL)[4], which helps to find aliases for a subject with the predicate `owl:SameAs`, both which are essential to the functioning of DAOLOT. If successful, the endpoint is added to the list.

While still providing in-depth options for more advanced users, all these options are handled automatically if the user does not modify them.

3.1 Result Gathering

DAOLOT's search works on a 2-phase system: In the first phase, the user inserts a search term about the topic or subject of interest. This query can be just a single word or a several keywords. When it is submitted, the algorithm prepares a query that searches the endpoints for subjects that contain any of the keywords either on its name or in any description associated with its name.

Then this query is sent to every endpoint selected in the configurations. As responses arrive, DAOLOT generates a list, each entry containing the name of the subject, its aliases on different endpoints, and a description compiled from all the descriptions linked to that particular subject. This first step presents to the subject a list of subjects that might be related to his search, and to help him choose which one to inquire further about.

3.2 Reduce redundancy

Because these descriptions usually have a high level of overlap, DAOLOT eliminates redundant sentences in these sections. Figure 3 provides an example on how redundant data from two sources is merged.

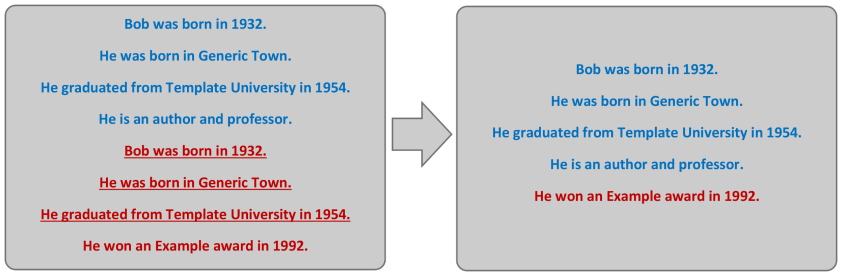


Figure 3 A visual representation of the redundancy reduction function.

3.3 Property Gathering

When the user selects a subject, this engages the search engine's second phase. In this phase, the engine will try to find all the available information and properties (e.g.: age, place of birth, related work) on this subject. As subject can be referred to by different labels across

the endpoints, the algorithm will build a query about all the triples that involve any of this subject's aliases, and send it to all the selected endpoints. In this phase DAOLOT relies on the use of the Comunica framework[11], which supports both SPARQL endpoints and Triple Pattern Fragments[13] endpoints (a simpler and lighter server interface for semantic data), allowing for a larger pool of available endpoints beyond the default ones, such as DBPedia, DBWik or YAGO.

After the query is sent to all the endpoints, DAOLOT adds to a list of information about this subject as the answers arrive. This profile of the subject will aggregate all the information, each property grouped with all the values associated with that property (if multiple different values appear). In addition, if any of the values of properties of this subject are images, DAOLOT will display these images as part of the results, as seen in Figure 4.

3.4 Information Checking

At this point, DAOLOT is able to check the information quality. For example, suppose that endpoint X returns the information that this subject's birth-date is `<S> <birthDate> 1932`. Suppose also that endpoint Y returns the same information. When the same value for the same property is returned by multiple sources, DAOLOT assumes that this piece of information is more trustworthy, and that property will be marked visually as containing trusted information, highlighting the trusted values.

On the other hand, DAOLOT is also able to detect contradictions. When a new property is introduced, the algorithm makes a "functionality check" to it, sending a request for all triples containing that property, and calculating the average amount of different values per subject on that property. If it determines that the property usually only has one value per subject, it will add it to a watch-list of properties that are supposed to only have one value.

Consider the previous example. Statistically, most people only have one birth-date. Hence, if a third endpoint Z says that the subject's birth-date is 1978, the property `<birthDate>` would appear in the list with a visual indicator that it is contradictory, and will have the source of each value displayed directly over it.

3.5 Navigating the results

If the value of a triple is also another subject the result will be presented as a link, and the user can now click this value in order to see all the information about that related subject. This process can be repeated indefinitely. As the user browses from one related subject to another, DAOLOT forms a search chain containing the sequence of subjects. This allows the user to use the arrow keys either on the screen or on a keyboard to navigate quickly trough all the gathered information, or to go back to a previous point in the search and follow a different path. Each time the user performs a new step in its search, the URL of the page is updated, allowing the user to save the location, to come back to this specific subject later or to share the results found.

Figure 5 bellow illustrates the different layers of the algorithm and it's stages.

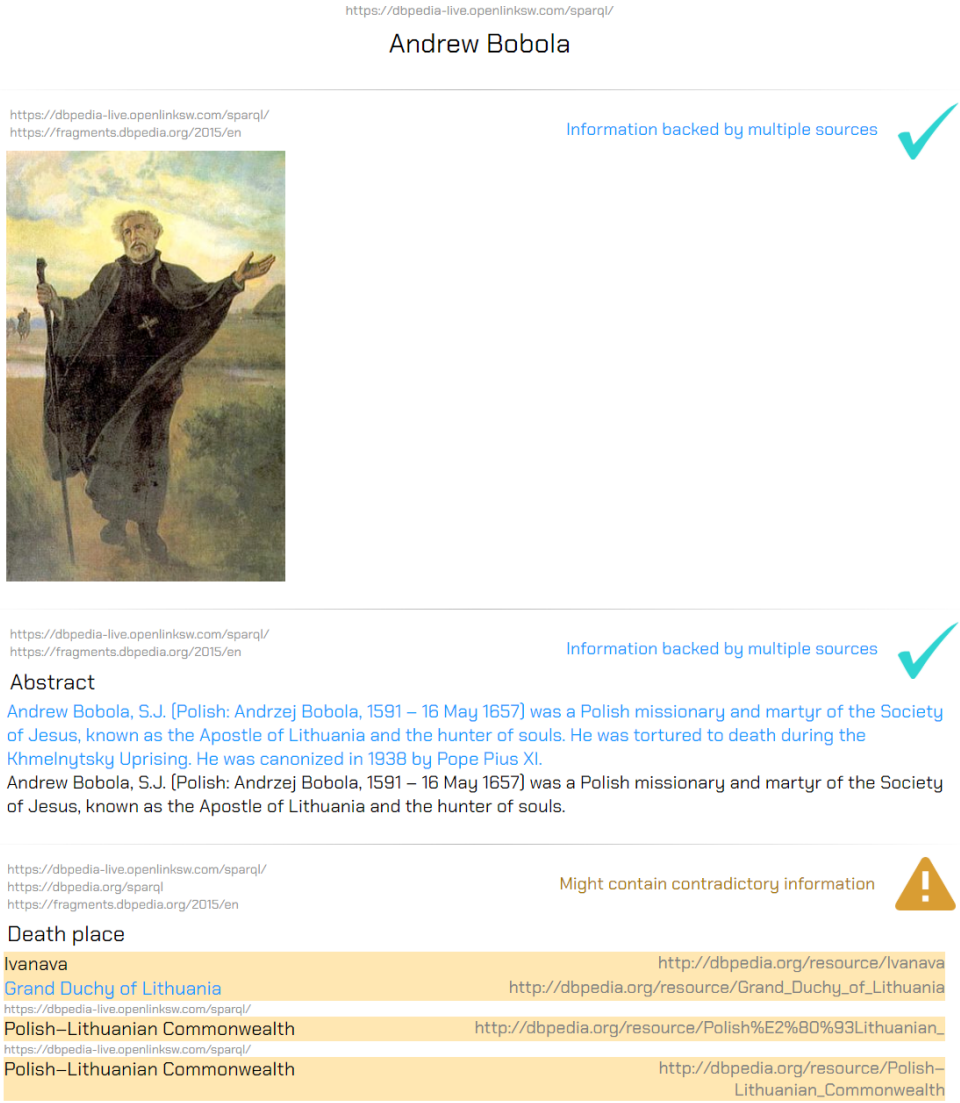


Figure 4 An example phase 2 search displaying images, contradicted and confirmed information.

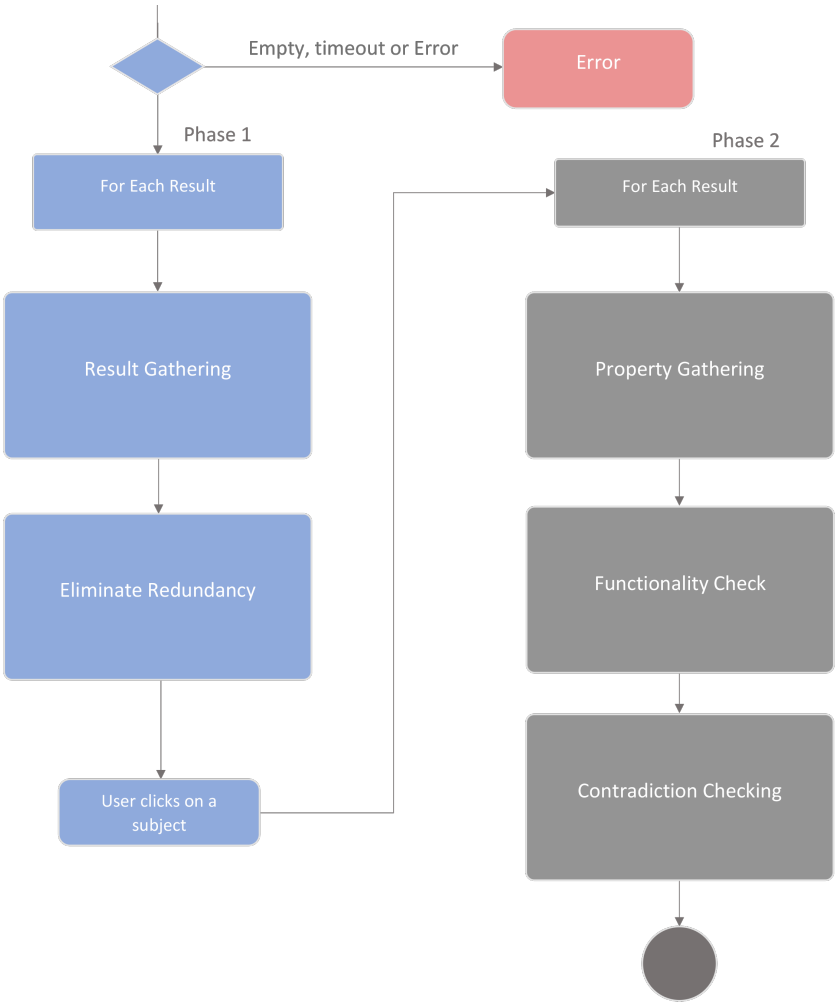


Figure 5 A diagram of the main data-processing function.

4 Validation

The validation of DAOLOT's usability followed the Nielsen Usability Model[12]. It was conducted with 7 participants, with ages between 20 and 25, without any experience or knowledge about web semantic.

Each was given a list of simple tasks to guide them through different features, each making them explore a different functionalities of DAOLOT:

Task 1
Find more about a certain person;
Task 2
Find his hometown and it's population;
Task 3
Find his main interests;
Task 4
Increase the search limit and remove an Endpoint from the settings;

Task 5
Find artists with the same name as the first search subject;
Task 6
Remove a specific endpoint and see if the results change;
Task 7
Add this custom endpoint to the settings;

After attempting those tasks, participants were asked to fill out a survey asking how much they agreed or disagreed with some statements about the usefulness and ease of use of DAOLOT. The screen of the participants was recorded for analysis, and the goal was to understand if they managed to use all the features of DAOLOT correctly without help.

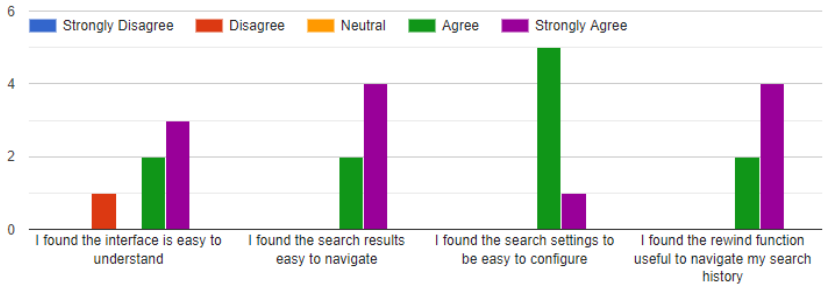


Figure 6 The responses given in the survey by the users.

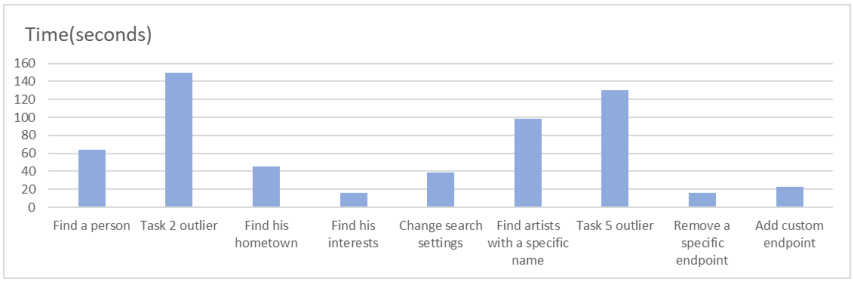


Figure 7 The average time of completion of each task in the survey.

In general, all participants managed to do most of the tasks with no guidance, except for the custom endpoint feature, which was due to their lack of knowledge on semantic web concepts.

As seen in Figure 7, most tasks were completed in under a minute, with the final two being completed on average under 30 seconds.

We can conclude from this data that after figuring out the core functionalities, the users were able to accomplish tasks much faster, as evidenced by the last tasks being completed much faster than the first two.

As Figure 6 illustrates, with one exception, all users found the interface easy to use and the results easy to navigate, despite their lack of technical knowledge.

After grasping the fundamentals about DAOLOT user interface with the first task, participants were able to complete most of the basic tasks in less than 25 seconds. The task which took longer, which consisted of finding artists with a particular name, was mainly due to a problem with DAOLOT's earlier version, which had trouble finding exact matches

for the participants's search, which led to unpredictable results. In response to this issue, the search algorithm was updated to have a secondary function designed to fetch only exact matches and put them on top of the list, before trying to search approximated results. The outlier that occurred in the task "find his hometown" with one participant was that the he tried to navigate between multiple subject as results were still loading, and thus ended up missing additional information. The solution was to implement a graphical queue on top of the subject's label, to indicate that more results are still incoming, which improved participant's reception.

The participants' feedback provided in the survey was also used to improve DAOLOT in several points. Some of them reported the UI as being "too colorless", and as such the UI was redesigned to have a more colorful aspect.

In the initial testing, some participants took longer to find the options, as the original UI just had the icons of the options visible, with no label as to what these icons did.

In response to this feedback, the UI was redesigned to have permanent labels on all the options, in addition to a loading icon to indicate to the user that more results are still arriving for the search.

Outside the programmed tasks, all the participants found DAOLOT to be easy to use and to produce relevant results.

5 Conclusions

The goal of DAOLOT is of being an intuitive, easy-to-use and reliable semantic browser for the average user. Based on the validation results, it can be concluded that DAOLOT has achieved this goal.

There were several major challenges in the implementation of this browser, such as dealing with contradictory information provided by several diferent sources, finding which entities (from distinct sources) were actually referring to the same subject, and how to maximize compatibility with endpoints. DAOLOT is unable to determine which information is correct, but it is able to point out contradictions and possible confirmations, by doing a statistical analysis of the variety of values of a property. It is also able to identify similar subjects by relying on a property of the OWL vocabulary, `owl:sameAs`. However, this represents one of the challenges of DAOLOT, which is how to deal with the lack of use of standard vocabularies. This browser requires that endpoints employ standard vocabularies such as RDFS and OWL in order to function correctly, and any endpoint who doesn't comply would require a custom implementation. This is why WikiData cannot be used as an endpoint on DAOLOT, despite being by far the largest and most complete general scope semantic source. These difficulties illustrate the challenge of building a completely autonomous agent to exploit resources from different sources – the goal of the semantic web – which would be significantly more complex than a semantic browser, such as DAOLOT.

The scientific contribution of this article is that DAOLOT provides a solution towards most of the problems that surround web semantics, by providing a way to easily convert the content of the semantic web into information that a human user can easily understand.

In the future, DAOLOT can be further improved by expanding its methods of data visualization, such as adding graphical representation of certain types of data and display statistics about certain properties when appropriate, such as a timeline of presidents in the history of a country.

DAOLOT is easily expandable, requiring little to no maintenance when dealing with new endpoints, and with the fast growth of the semantic web, it will only get more useful over time, as the amount of information available to it increases.

References

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

2 Dan Brickley, Ramanathan V Guha, and Andrew Layman. Resource description framework (RDF) schema specification. Technical report, World Wide Web Consortium (W3C), 1999.

3 Kendall Grant Clark, Lee Feigenbaum, and Elias Torres. SPARQL Protocol for RDF. W3c recommendation, W3C, January 2008. URL: `http://www.w3.org/TR/rdf-sparql-protocol/`.

4 W3C Owl Working Group et al. Owl 2 web ontology language document overview, 2009. URL: `http://www.w3.org/TR/owl2-overview/`.

5 Sébastien Harispe, Sylvie Ranwez, Stefan Janaqi, and Jacky Montmain. Semantic measures for the comparison of units of language, concepts or entities from text and knowledge base analysis, October 2013.

6 C. Harris, A. Owens, A. Russell, and D. A. Smith. mSpace: Exploring the semantic web. a technical report in support of the mspace software framework. Master's thesis, Faculty of Engineering, Science and Mathematics, University of Southhampton, 2004.

7 Michiel Hildebrand, Jacco Van Ossenbruggen, and Lynda Hardman. /facet: A browser for heterogeneous semantic web repositories. In *International Semantic Web Conference*, pages 272–285. Springer, 2006.

8 Ora Lassila, Ralph R Swick, et al. Resource description framework (RDF) model and syntax specification. Technical report, World Wide Web Consortium (W3C), 1998.

9 Eric Prud'hommeaux and Andy Seaborne. SPARQL Query Language for RDF. W3C Recommendation, January 2008. URL: `http://www.w3.org/TR/rdf-sparql-query/`.

10 MC Schraefel, Daniel A Smith, Alisdair Owens, Alistair Russell, Craig Harris, and Max Wilson. The evolving mspace platform: leveraging the semantic web on the trail of the memex. In *Proceedings of the sixteenth ACM conference on Hypertext and hypermedia*, pages 174–183, 2005.

11 Ruben Taelman, Joachim Van Herwegen, Miel Vander Sande, and Ruben Verborgh. Comunica: a modular sparql query engine for the web. In *International Semantic Web Conference*, pages 239–255. Springer, 2018.

12 Carl W Turner, James R Lewis, and Jakob Nielsen. Determining usability test sample size. *International encyclopedia of ergonomics and human factors*, 3(2):3084–3088, 2006.

13 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.

A Press Summary

DAOLOT is a search engine directly provides the user with the information it finds about a subject, providing sources and even qualifying pieces of information as trustworthy or not, and neatly presenting it in a convenient format, instead of just providing the user with a list of links to websites.

DAOLOT can provide the user with what would be usually a prolonged research in moments, instead of having to spend time collecting information about a subject across multiple websites.

Musikla: Language for Generating Musical Events

Pedro Miguel Oliveira da Silva
Departamento de Informática, Universidade do Minho, Braga, Portugal
pg38423@alunos.uminho.pt

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

Abstract

In this paper, we'll discuss a simple approach to integrating musical events, such as notes or chords, into a programming language. This means treating music sequences as a first class citizen. It will be possible to save those sequences into variables or play them right away, pass them into functions or apply operators on them (like transposing or repeating the sequence). Furthermore, instead of just allowing static sequences to be generated, we'll integrate a music keyboard system that easily allows the user to bind keys (or other kinds of events) to expressions. Finally, it is important to provide the user with multiple and extensible ways of outputing their music, such as synthesizing it into a file or directly into the speakers, or writing a MIDI or music sheet file. We'll structure this paper first with an analysis of the problem and its particular requirements. Then we will discuss the solution we developed to meet those requirements. Finally we'll analyze the result and discuss possible alternative routes we could've taken.

2012 ACM Subject Classification Computing methodologies → Language resources

Keywords and phrases Domain Specific Language, Music Notation, Interpreter, Programming Language

Digital Object Identifier 10.4230/OASIcs.SLATE.2020.6

1 Introduction

Musikla stands for Music and Keyboard Language. Our goal is to develop a DSL (*Domain Specific Language*) that allows treating musical events with the same importance as other basic types, like integers and booleans, are treated in most programming languages. More than generating these musical events offline, we want to be able to easily declare keyboards that map keys to expressions that either mutate the state or play musical events (or even both).

The project can be partitioned in three different, modular layers: inputs, the language, and outputs. While music events can be described as code literals inside our language, they can also originate from many other sources (such as files or physical devices such as pianos). After being processed by our language, they are then emitted as a stream of musical events to the **Player** component, which then multiplexes those events into however many outputs the user defined.

While the development of both the input and output layers, as well as their many respective components, presents by itself many interesting challenges that could be discussed, we will instead focus this paper on the aspects of the middle layer: the *interpreter*, while ackowledging the existence (and their effects) of the layers that wrap around it.

As such, the problem of developing the interpreter can be divided into two parts: the syntax used for describing the notes and the operators that compose them inside the language; and the semantics of the generated events, how they are stored in memory, and how their temporal properties (start time and duration) are handled without forcing the programmer/user to always manually type them.

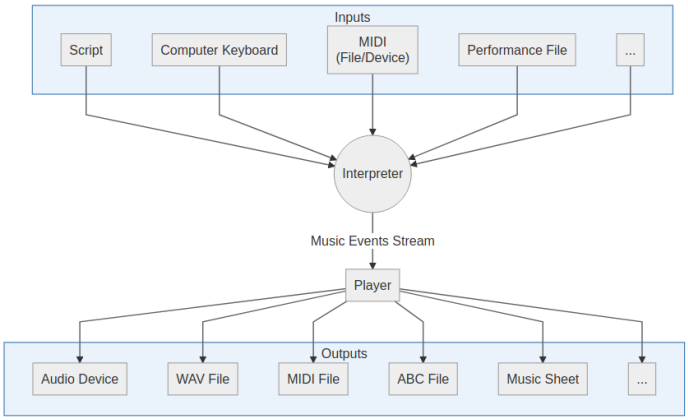


Figure 1 The three main layers of the project.

Designing the syntax for describing those musical expressions, especially given our strong desire to make those musical expressions first class citizens like other primitive data types in most programming languages (such as numbers, strings or arrays are), did unearth some challenges. To minimize the learning curve for new users, and avoid reinventing the wheel, we decided to adopt a subset of the very popular note declaration syntax from the ABC Notation project[3, 5] and integrate it with our language.

As for the execution model, we decided to go with a tree-walker interpreter[8]. Altough computationally slower than other alternatives (such as a bytecode virtual machine), the ease of implementation allowed us to prototype and develop features extremely fast. And with a more mature and stable language in the future, there is always the potential to rewrite the interpreter if performance or latency ever reveal themselves as potential problems.

The simplest way of generating such musical events in a programming language is to use already common, *low-level*, programming mechanisms, such as using a procedural approach where the user creates each event manually by calling a function and providing as parameters all the events' information, such as it's timestamp and duration. This is the approach used by some of the existing languages in this space, such as *SonicPi*[2], and it's usage can somewhat resemble the code in listing 1.

Listing 1 Example of a hypothetical imperative API for creating events.

```
play_note( 0, 100, 'A' );
play_note( 100, 50, 'B' );
play_note( 150, 200, 'C' );
```

Instead we decided to follow a more *functional* approach, with custom syntax and operators, as well as the hability to describe those events in a single expression. Musical events are treated as sequences, and as such can be stored in variables, passed around inside functions and trasformed. So, for musical events, we will be exploring a way to define them in code, as *musical literals*, such as what can be seen in listing 2.

Listing 2 Our proposed declarative syntax that calculates timings implicitly.

```
play( A B/2 C2 );
```

In the listing 2 we can see we have discarded the explicit timings in milliseconds that were being used in the listing 1, and instead adopted an approach more in line with music

notation, where notes' durations are written as *breve* or *double whole note* (2), *semibreve* or *whole note* (1, the default), *minim* or *half note* (1/2, with the number 1 being optional in fractions). The actual duration of the notes in milliseconds is then calculated behind the scenes and can be adjusted automatically by changing the tempo (which we will cover later). Note that from this point forward, all code listings present in this paper will contain code written using our own DSL.

2 Initial Problem and Desired Goals

Our overarching goal is to have a dynamic programming language that allows to input and manipulate musical arrangements of varying complexity. We want our language to be as flexible as possible, with features like variables, functions, conditionals and loops. More imporantly, we want to provide a standard, out-of-the-box way for the user to declare is musical keyboards that is tighly coupled with the language (with the keyboards actions being expressions or even statements in our language).

There are two important requirements we need to consider when evaluating possible solutions to this problem: the ability to produce music interactively, and to produce music lazily.

Having **interactivity** adds the requirement that musical events (like playing a note) have to be produced in an ordered fashion, so that we can play them in realtime.

And with the events being produced in an ordered fashion also opens up the possibiliy of baking **laziness** into the language, so that we can generate only the music that is needed. This is especially nice when we have a keyboard that can, for example, start playing a *tecnically-infinite* musical sequence that is being calculated on demand, and that the user can stop at any time he wants.

But like we've mentioned before, this requires events to be generated in order: if the first event to be played could be the last to be generated, we could not implement laziness. We would always have to generate the full musical expression before we knew what to play first. Therefore having a total order for our events is a key feature.

▷ Claim 1 (Total Order). To allow potentially infinite sequences to be played in realtime, music events must always be generated in the correct order: otherwise the sequence would have to be generated fully before we could be sure of what event was actually next in line to play.

Goals

We can then summarize our main goals for this language as follows:

- **Declarative.** Music sequences are described in a declarative (rather than imperative) fashion.
- **Dynamic.** Introduce programming or mathematical concepts, like functions and variables, to the music world.
- **Interactive.** Make it possible to create interactive keyboards out-of-the-box that integrate with all features provided by the language.
- **Lazyness.** Make lazyness for musical sequences the default, generating only events as they are need.
- **Rich events.** Music sequences can describe complex musical arrangements, containing simple notes, rest, chords, voices, and more.
- **Multiple Inputs.** Besides allowing musical arrangements to be declared inside our language, also allow for them to be imported and converted from multiple sources, like MIDI files and devices.

- **Multiple Outputs.** Store or write to multiple different outputs the music events generated inside our language.
- **Extensibility.** Make it easy to extend and customize the project, without needing to fork or recompile or hack it's internals.

3 Related Work

There are a number of both domain specific languages that provide facilities to generate music through programming, as well as libraries that try to align existing general-purpose programming languages to the same goal. When keeping in mind the goals defined in the section above, those existing approaches can be categorized in a handful of groups that share similar characteristics. None however, covers all the needs we had in mind.

In terms of music notation, there are markup languages such as **alda** and **abc notation** [3] (from which we take a lot of inspiration) that are static textual representations of musical notation. While they are easy to use, they do not possess any dynamic capabilities (no variables, no control structures such as conditionals or loops, no functions). It is possible to envision using a regular programming language to function as a sort of macro language, to allow some dynamism. But still, beyond being a slightly cumbersome approach that requires duct taping unrelated tecnologies together, we would most likely be working with string operations to generate the final output, and wouldn't be able to easily treat each note and chords as individual entities.

Other languages, such as **Faust** [9], **Chuck** [12] and **SuperCollider** [7] do provide programming mechanisms, but they work as audio processing languages, not specifically musical languages. And while the capabilities of the former are always good to have, they are too low level and require significant knowledge and time to implement the later on top of them.

SoniPi [1] is the one that most likely provides the functionality we are looking for, mixing programming mechanisms with the concepts of notes and chords. But still falls short on the usability side, employing a procedural style of programming (play this note now, advance time, play this note now, and so on) rather than a declarative style (play this complex musical expression).

On top of all that, neither of the solutions above handle very well extensibility in terms of implementing new *output* formats. And most importantly, none provides functionality to declare interactive keyboards to allow playing fragments of music defined in their respective languages, nor to change the state in realtime of the music being played.

4 Specification

We won't exhaustively cover all the details of our language in this paper, and will instead focus on the ones that deviate from the conventional semantics of a programming language. However, a more practical guide (but still a work in progress) documenting how to use our language is available online [1].

[1] `https://pedromsilvapt.github.io/miei-dissertation/`

Core Types

The basic premise is that our syntax has special constructs designed to facilitate the generation of the **Music** data type. Music is simply a sequence (or stream) of ordered musical events. However, it is also treated differently by many constructs in our language (which we will explore in more detail later in this paper). Just as a quick example, any statement in our language whose return value is of the type Music **and** is not captured (into a variable or passed as an argument to a funcion) is played immediately. Instead of having to call a `play` function when we want to play something (which in our language will be most cases), that function call is implicit. And when we don't want to play, we can use the `discard` function provided by the language, on the music we don't want.

A musical **Event** can be one of many things, such as a *note*, a *chord*, or even more implementation-specific events like MIDI messages[6]. While all events must have a start time, some events can be instantaneous (events with a duration of zero time units). Our goal is to have the language be extensible. And so even though we have core events (such as notes and chords) that most functions and operators are aware of, it is also possible for the user (or custom libraries) to create their own custom event types. With that in mind, all code that handles events should try to be as permissive as possible, interacting with events it knows and understands, and simply passing along any events it doesn't know, untouched.

The time unit used in the events timestamp and duration fields could be anything so long as it has a **total order**. We chose to represent it using milliseconds.

Syntax

While our goal is not to delve into every little detail of the language (because most are just similar to all popular programing languages), and we'd rather focus on how the tight integration of the Music data type into the language changes some aspects of it. Still, for the sake of making it easier to understand this paper, we will briefly discuss the syntactic features provided by the language.

Here is a non-exhaustive list of accepted expressions:

Table 1 List of valid expressions in Musikla.

EXPRESSIONS	SYNTAX
Variables	`$var`
Function Calls	`function_name( expr, named_arg = expr )`

Function Declaration	`$name = fun() {}`
	`fun name($arg, ref $ref_arg, optional = expr) {}`
	`fun name() => expr`
	`fun name() {}`

Modules	`import "file.mkl"`
Arithmetic Operators	`a + b; a * b; a - b; a / b`
Literals	`1, true, none, "string"`

A **function declaration** is an expression (allowing higher order functions). Function names are optional. Giving a name to a function is equivalent to assigning an anonymous function to a variable with that same name.

Table 2 List of valid statements in Musikla.

STATEMENTS	SYNTAX
Attribution	`$var = expr`
Code Blocks	`{ stmt; stmt; stmt }`
Conditionals	`if (expr) {stmt} else {stmt}`
Loops	`for ($var in expr) {stmt}` `while (expr) {stmt}`
Returns	`return expr`

Statements must always end with **semicolons**, even loops and conditionals, with the exception of the **last semicolon** in a list of statements, which is optional.

Table 3 List and Object operations in Musikla.

LISTS	SYNTAX
Declaration	`@[ "list" ]`
Indexed	`$var::[ $index ]`
OBJECTS	
Declaration	`@{ "object": 1 }`
Attributes	`$obj::property`
Methods	`$obj::method()`

To avoid ambiguities between chords, lists (square brackets), code blocks, objects (brackets), both list and object literals are prefixed with the `@` symbol.

Also, we use two colons as separators to access lists and objects' properties and methods, because the period is used in the musical notation for indicating dotted notes, and as such could be confusing and ambiguous sometimes.

Table 4 Keyboard declaration and body syntax.

KEYBOARDS	SYNTAX
Keyboard Declarations	`@keyboard {  }`
Keyboard with Modifiers	`@keyboard hold extend {  }`
KEYBOARD BODY	
Expression action	`key: expr`
Action Block	`key: { stmt; stmt }`
KEYS[2]	
Single Keys	`a, b, 1, 2, ".", up, f12, ctrl a, ctrl shift f`
Composite Keys	`ctrl a, ctrl shift left`
MIDI Keys	`[c] [D,] [a'']`

[2] Examples of the types of keys accepted. Rules for the key syntax are described below the table.

A keyboard can have modifiers (such as **hold** to start/stop the music when the key is pressed/released; **toggle** to start/stop the music when the key is pressed; **extend** to override the duration of the notes to last as long as the key is active;).

Keys can be any alphanumeric character, or any character wrapped in quotes. We can also create key combinations with the modifiers **ctrl**, **shift** and **alt**. Notes can be used as keys (to program MIDI keyboards connected to the computer) by wrapping them in square brackets. Each key can have an expression associated, or a code block with a list of arbitrary statements.

Table 5 Music literal expressions.

MUSIC	SYNTAX
Notes	`C, C c c' A2/3 _A ^A`
Chords	`[CFG] [Cm]/2`
Rests	`r r1/2`
Modifiers	`S4/4 T60 L/2 V70 O2 I2`
Parallel	`C E | F A`

A note's octave is expressed by the case of the note (uppercase and commas mean lower pitch, lowercase and single quotes mean higher pitch). They can be prefixed by an optional accidental (underscore for flats, caret for sharps). They can be suffixed by an optional note length (either an integer or a fraction). Chords can list each of their notes, or just the root note and a suffix (**CM** for *C major* and **Cm** for *C minor*, for example).

Operators

Operators are special operations defined at the syntactic level that allow *music* to be composed in different ways, such as concatenated, parallelized or repeated. Many of these operators can have equivalent functions available through the language that provide more costumization (such as a parallel function that stops when the smallest operand stops, instead of the longest).

Concatenation `Music1 Music2 ... MusicN`
type `List[Music] -> Music`

Parallel `Music1 | Music2 | ... | MusicN`
type `List[Music] -> Music`

Repetition `Music * Integer`
type `Music, Integer -> Music`

Arpegio `Chord * Music`
type `Chord, Music -> Music`

Retime `Music ** MusicOrLength`
type `Music, Union[Music, Float] -> Music`
Stretches (or shrinks) proportionally the length of the first music expression to be the same as the length of the second.

Transpose `Music + Integer` *and* `Music - Integer`
type `Music, Integer -> Music`

It is also useful to estabilish that while most operators work on sequences of musical events, they can also accept a singular event as their argument: one event can be trivially converted into a sequence of one element. Such ocorrence is so common and trivial that the conversion should therefore be implicit whenever necessary.

Grids

Another type available in our language are grids. Also known in most music applications as the process of quantization [11]. The reason it is so useful in our language is that when receiving input as musical events from a live keyboard, their timings are naturally more prone to having small discrepancies that can become more apparent when we then mix them with generated musical events (which have precise timings).

Having events always aligned with such a grid can also make computations and transformations of such events easier and simpler, which is always a plus for our language.

Create a grid `Grid(size)`
  type `Fraction -> Grid`
Aligning `grid::align(music)`
  type `Grid, Music -> Music`
Compose Grids `Grid::compose(grid1, grid2, ..., gridN)`
  type `List[Grid] -> Grid`

We can see that apart from the basic operations of creating a grid and aligning events to said grid, we also want the ability to compose multiple grids (of different precisions). We will approach this matter in more detail later.

Keyboards

A core part of the language is our hability to declare keyboards, which we can describe as mappings between `Keys` and *Musical Expressions*. Each expression can mutate the state (changing variables or calling functions), return some music (sequence of musical events) to be played, or both.

We've seen in the *syntax* section how to create these keyboards in our language. Behind the scenes, that syntax is merely a convenience that is translated into regular method calls (like registering a key). We can see some of the available methods for the keyboard object here.

Create a keyboard `keyboards\create()`
  type `() -> Keyboard`
Binding a Key `keyboard::register(key, expression)`
  type `Keyboard, Key, Expression -> Keyboard`
Mapping a keyboard `keyboard::map(transformer)`
  type `Keyboard, ( Music -> Music ) -> Keyboard`
Aligning with a Grid `keyboard::with_grid(grid)`
  type `Keyboard, Grid -> Keyboard`

5 Implementation

The reference implementation for this system is written in Python, although the approach here should be language agnostic.

One of the features that Python boasts (but are certainly not exclusive to it) that have eased our implementation of the language are generators[10]. They integrate very nicely into both our concept of emitting musical events as sequences (or iterators, as they are called in Python and other languages), as well as into our concept of laziness, where events are generated on demand when needed, and thus infinite musical sequences can be handled easily.

Context State

To keep track of the *cursor* (the current timestamp where the next event should start) each operator in our language is implemented as a function call that receives an implicit `Context` object. While here we'll mostly focus just on the methods related to time management provided by the context, it can be used to store other types of information, like the default length of a musical note, for instance, to avoid forcing the user to type it out all the time, or the tempo at which it is to be played.

It is important to keep in mind that there might be more than one context in execution at the same time. This can be most obvious with the use of the parallel operator, where each operand must run concurrently (and thus could not share the same context).

Let's describe what kinds of functionality our context should provide.

cursor(ctx) Return the current cursor position
seek(ctx, time) Advance the cursor to the given position
fork(ctx) Clone the parent context and return the new one. Allows multiple concurrent contexts to be used
join(parent, child) If the child's cursor is ahead, make the parent context catch up

5.1 Operators

Basic Events

The basic building block of our system are the **Note**, **Chord** and **Rest** events. We can use the current *context* to determine the event's timestamp, as well as it's default duration (in case the user does not explicitly state one). Any event(s) that is/are not captured in a variable or passed to a function are implicitly played. We can look at a very simple example of declaring a note event, two octaves up from middle `C`, and the length of a quarter note, in the Listing 3. The note syntax follows very closely the same approach as the *abc notation* project, so there is no much use in discussing it in detail.

Listing 3 Creating a Note Event

```
c'1/4
```

Voice Modifiers

In the following sub-sections we will cover how to declare many combinations of notes and chords. Each of those musical events needs a lot of information to be able to describe them thoroughly, and doing that for every event would be cumbersome. To solve that problem we choose to provide sensible defaults for those values, and allow the user to specify **modifiers** to customize those values when needed for a specific group of events.

There are multiple **modifiers**, and they can be used anywhere in a musical expression. Each modifier is identified by a single letter, followed by the value carried by the modifier. Those values can either be an integer, or a fraction. Modifiers are case insensitive. Some of the modifiers available are described below:

Time Signature. Identified by the letter **S** followed by a fraction. For example, the expression `S4/4` sets the time signature to common time, and `S3/4` set's it to *three-four time*.
Tempo. Identified by the letter **T** followed by an integer. For example `T60` sets the tempo (beats per minute) to 60.

Velocity. Identified by the letter **V** followed by an integer between 0 and 127, sets the velocity (roughly equivalent to volume) of the notes. For example, `V70` sets the velocity to 70.

Length. Identified by the letter **L**, followed by an integer or fraction, sets the base length of each note to it's value. If a note has a custom length, their values are multiplied. For instance, the expression `L2 C/4` would create a note with the actual length of $2 \times \frac{1}{4} = \frac{1}{2}$.

A modifier can be declared inside inside parenthesis, and so will only affect events emitted inside those parenthesis as well. For instance `(L2 C) C` is functionally equivalent to `C2 C`.

Concatenation

We've seen how single events' declarations are handled and how we can customize the settings that affect those declarations. Now it is important for us to see how we can combine those events together. And probably the most straightforward operator of all, concatenation, it simply consumes each event. Each event, as we've seen before, is responsible for seeking the context depending on the event's duration. In the Listing 4 we show how to concatenate some notes, as well as some modifiers, to recreate a portion of a song, whose generated music sheet can be viewed in figure 2. Note that the semicolon is only needed to separate statements. An expression is always a valid statement. In this case, it serves simply to better separate the configuration and the actual notes. They could however be written in a single line with no semicolons and have the same meaning.

Listing 4 Snippet of the song *Wet Hands* by C418

```
S4/4 T74 L/8 V90;
A, E A B ^c B A E D ^F ^c e ^c A3;
```

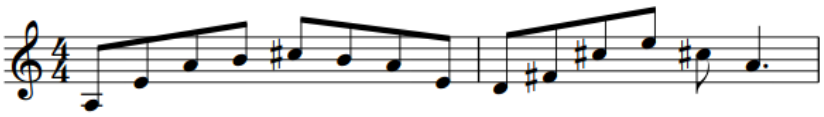


Figure 2 Generated music sheet for concatenation[3], audio version available here[4].

Repetition

The repetition operand is in a way very similar to the concatenation operator. It makes sense, since repeating any kind of music pattern N times could be thought as a particular case of as concatenation where there are N operands, all representing the same musical pattern.

Listing 5 Intro to Westworld's Theme by Ramin Djawadi.

```
I1 S6/8 T140 L/8 V90;
A*11 G F*12
```

[3] Rendered with $ABC_UI. Some hand made changes made for clarity.
[4] https://drive.google.com/open?id=1TP4lcul81s8iMCUFmD3HKnSpeftCzKT0
[5] https://drive.google.com/open?id=1IIm8PQkLsNFMK9MNSVubJSG6SP6KwhPL

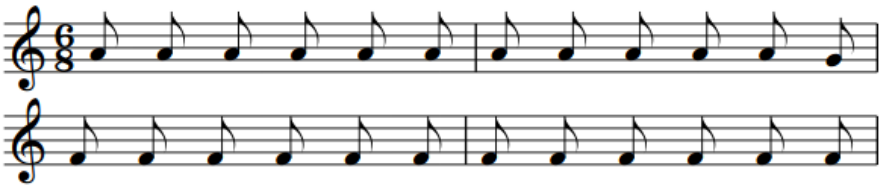


Figure 3 Generated music sheet for repetition, audio version available here[5].

Parallel

The parallel operator enables playing multiple sequences of musical events simultaneously. However our events are emitted as a single sequence of ordered events, thus requiring merging the multiple sequences into a single one, while maintaining the properties of laziness and order. The operator assumes that each of its operands already maintains those properties on their own, and so is only in charge of making sure the merged sequence does so as well. With this in mind, it relies on a custom *merge sorted* algorithm for iterables (not related to the most common merge sort algorithm by John von Neumann).

Listing 6 Snippet of the song *Soft to Be Strong* by Marina.

```
T120 V70 L1;
r/4 ^g/4 ^g/4 ^g/4   ^f/2 e/8 ^d3/8   ^c2 | [^Cm] [BM]  [AM] [BM]
```

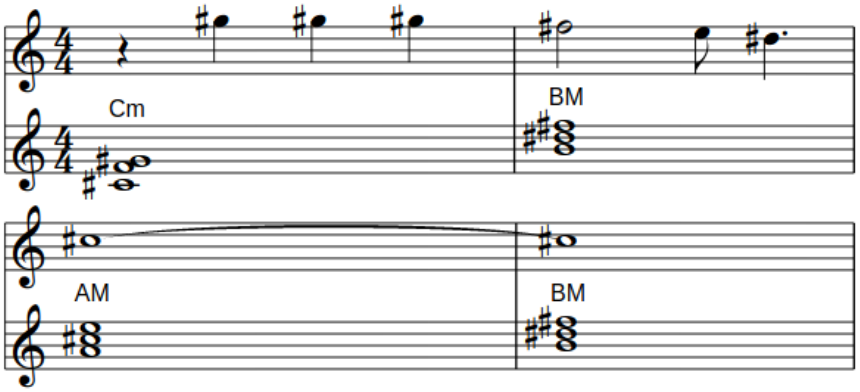


Figure 4 Generated music sheet for parallel, audio version available here[6].

The merge sorted function receives N operands and creates a buffer with the size N. For each operand it *forks* the context, so that they can execute concurrently and each will mutate their own context only. It then requests one single event for each operand.

After the buffer is prefilled (meaning it has at least one event for all non-empty operands), the algorithm finds the earliest event stored in the it. Let's assume it is stored in the K index of the buffer, with $K < N$. The method emits the value stored in `buffer[K]` and then fills requests the next event from the K operand (storing `null` if the operand has no more events to emit). It then repeats this step until all operands have been drained.

[6] `https://drive.google.com/open?id=1ENTm3hZonYHyQIOgRZ8TQ1Qz-AfRLt2I`

5.2 Integration in a Programming Environment

Apart from generating musical events from syntatic constructs, our goal is to have those events integrate into the rest of our own DSL in the same way integers, floats, strings and booleans do: as data that can be stored, passed around and manipulated. This, of course, must still retain all the properties we've laid out for our sequences of events: being lazy and always being ordered.

Variables and Functions

All expressions that are assigned to a variable run in a forked context, with it's cursor set to zero initially. Musical expressions inside variables are still lazy (meaning they only calculate each musical event when the variable is first used, not declared) but the events are cached to prevent the calculations from being performed every time the variable is used. This cache is then garbage collected when the variable is no longer in use.

Sometimes this lazyness can indeed be more trouble than it's worth, and that's why from the very beginning the language allows the user to explicitly consume a musical sequence (with the knowledge that it cannot be an infinite one, or the application will hang). Once consumed, all it's events will be cached in memory, stored in a list, and the user doesn't need to worry about laziness there anymore.

When events are stored in a variable, their timestamps are relative to the start timestamp of that variable (which is always zero). But when the variable is then used inside some expression, the events' timestamps must be updated to be relative to where the variable was inserted into. Since one variable can be inserted into more than one place, we cannot edit the timestamp of the event *in place*, otherwise the places where we had already used that variable would have their events' timestamps changed too. This highlights the need for musical events to be immutable, and for the need to copy them when we need to make a change to one of their properties.

This works well enough because those events are very lightweight objects, and the benefits of not having their values mysteriously changed midway during execution outweight the small cost of a possible unnecessary allocation of an event that would only be used in one place instead of many.

Function calls, on the other hand, pass the current context to the inside of the function, so that any events played there now their correct times.

When integrating functions into our language, we decided to keep the semantics simple. Returning musical events inside a function is similar to its return value being an iterator that gives out the emitted events on demand (similar to how many programming languages implement generator functions and emit values with the *yield* keyword). This means that a function cannot both return musical events, while also returning other values manually through a `return` statement.

There is no syntactic marker to distinguish regular functions from *"musical-emitting"* ones (meaning, there is no explicit *yield* keyword). Instead, the language runtime starts executing each function as a regular one, and automatically switches its execution mode into a generator-like implementation once a statement that returns musical events is executed (and it's value is not captures into a variable). Any return statements that are evaluated after this point must have no value (thus preserving the ability to early-stop a function). If they do try to return a custom value, a runtime exception is triggered.

Here in the Listing 7 we can see a small snippet of the beginning of *Fugue 2 in C minor in Book I of the J.S. Bach's Well-Tempered Clavier* written in our language, and how using functions and variables can help us visualize the structure behind music.

Listing 7 Using variables and functions to compose musical arrangements.

```
fun fugue ( $subj, $resp ) =>
    ( $subj $resp | r ** $subj ( $subj + 7 ) );

S8/4 T140 L/4 V120;

$subj = r c/2 B/2 c G _A c/2 B/2 c d
        G c/2 B/2 c d F/2 G/2 _A2 G/2 F/2;

$resp = _E/2 c/2 B/2 A/2   G/2 F/2 _E/2 D/2   C _e d c
        _B A _B c   ^F G A ^F;

play( fugue( $subj, $resp ) );
```

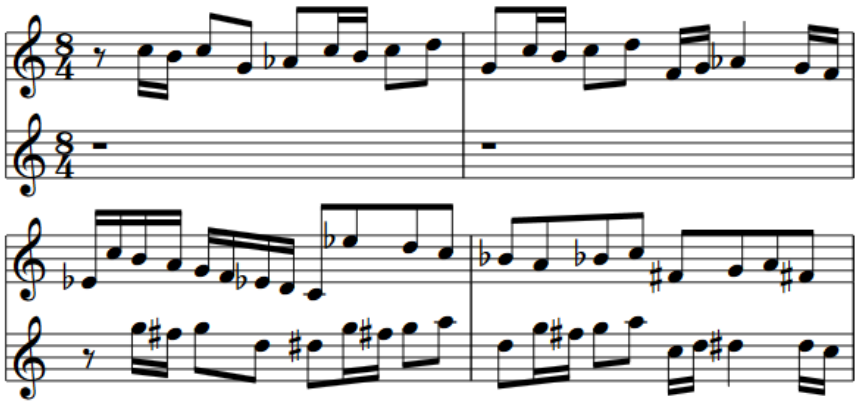


Figure 5 Generated music sheet for fugue example, audio version available here[7].

The music sheet generated by the code in the listing 7 can be seen in figure 5. The two staves of the first system correspond to the expressions `$subj` and `stretch( r, $subj )`, respectively (we can identify it easily in the second staff, where there are only rests). The two staves of the second system on the other hand, correspond to the expressions `$resp` and `( $subj + 7 )` (the later of the two being a transposition of `$subj` but with a higher pitch of 7 semitones).

Grids

To define a grid there is only one parameter required: the length of it's cells. When aligning musical events, anything that falls inside each cell will be pushed to the closest edge of the cell.

Grids are highly customizable too, however. They have multiple parameters, such as `forgiveness` and `range`, that determine when an event is affected by the grid (depending on how close it's start time is to the edge of the cell). Each parameter can even be customized separately for the left and right sides of the cell's edge.

Let's take a look at an example of a grid. In this example the grid has a cell size of **1**. We define the same values for both left and right sides just for the sake of this demonstration, but each side could have different values.

[7] `https://drive.google.com/file/d/1dIfvnhhKn73VppOW6ss6RLsv6PQ_HFTF/view`

Listing 8 Declaring a grid with some additional optional parameters.

```
$grid = Grid( 1,
    forgiveness_left = 125,  # 1/8
    range_left = 375,        # 3/8
    forgiveness_right = 125, # 1/8
    range_right = 375        # 3/8
);
```

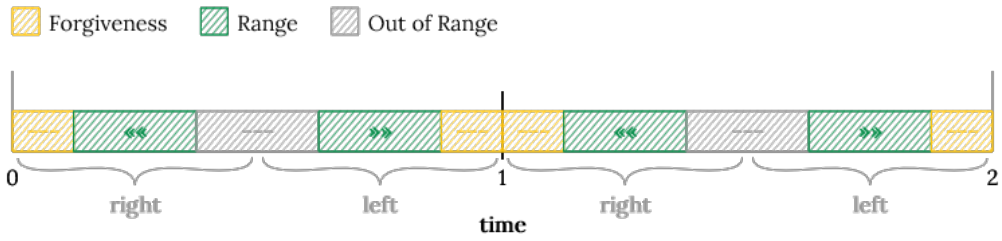


Figure 6 Representation of two cells from this grid.

We can see in this timeline two cells (each with a size of 1). Any events that fall in the yellow and grey areas are ignored (meaning their timestamps are not changed) while events in the green areas are pushed to whatever edge is closest. But even this behavior can be customized, forcing events to always go to the previous edge cell, or always to the next.

It is then trivial to see how we could compose multiple grids in sequence, each with different ranges (green areas) that capture different events and align them accordingly.

Keyboards

Finally we can combine all the systems we've described above, from musical expressions, grids, variables and functions, and devise a compact way of describing virtual keyboards.

To make the process of designing keyboards less verbose, we've added syntactic sugar to this process, that is translated in the background to regular function calls registering each key binding.

While a picture maybe worth a thousand words, a good example is worth maybe even more. So here we can take a brief look at the workflow for defining two keyboards (that are active at the same time). The first keyboard has all the musical keys (the chords and single notes we want), all aligned by a custom grid.

The second keyboard binds to the up and down arrow keys and allow us to change the virtual instrument through which we play the sounds of the notes in the keyboard (those instruments can be identified by an integer and usually follow the General MIDI standard[4]).

Listing 9 Creating a keyboard that can play multiple instruments.

```
$inst = 0;

fun spin_instrument ( ref $instrument, $change ) {
    $instrument = $instrument + $change;

    setinstrument( $instrument );
};
```

```
@keyboard {
    a: [^Cm];    s: [BM];    d: [AM];    f: [EM];    g: [^Fm];
    1: ^c;       2: ^d;      3: e;       4: ^f;      5: ^g;
    6: b;        7: ^c';     8: ^d';     9: e';
}::with_grid( Grid( 1 / 16 ) );

@keyboard {
    up: spin_instrument( $inst, 1 );
    down: spin_instrument( $inst, -1 );
};
```

Keyboards are objects (that we could save in a variable for example) and that can perform many operations, like unions and intersections, or maps and filters. They can be enabled and disabled at runtime, and their keys can be simulated to be pressed and released.

More than that, we don't need to restrict ourselves to computer keyboards. We can for instance, define bindings between MIDI events and musical expressions, so that when we connect a piano keyboard to our computer, we can use each piano key to play more than a single note.

Since like we've seen keyboard keys are not limited to computer keyboards, we can imagine the possibilities of events we could listen to: knobs, mouse buttons, the mouse scroll wheel. We could even create an event that could, for example, listen on a socket and trigger when a message is received, allowing in that way our musical applications to be controlled remotely.

The result is that our keyboards are extremely extensible and allow for a great deal of creativity. And thanks to our tight integration with the Python language, those extensions can be easily integrated and don't require hacking the source code or recompiling the application.

6 Results Discussion and Conclusion

The scope of this project could have been massive, mainly because implementing a way to fully describe hundreds of years of comulative musical notation would be a gigantic task. We chose instead to build a solid foundation, always with a strong focus on extensibility for the future.

This hability to extend the functionality of our project, thanks to our easy integration with regular Python code, means that new types of musical events, new inputs or outputs, new functions or libraries can be added with minimal effort by everyone using our application. There is no need to recompile the project or change the source, just include the *Python* code at runtime.

When it comes to keyboards, there are many more possibilities to explore too. While we've included examples of working with key presses, both from computer keyboards as well as pianos (through a MIDI connection), more rich events can also be used that were not demonstrated here. Since each event can also carry parameters with it, we can do more than boolean press/release types of events: we can model knobs or scroll wheels or other kinds of spatial events into our keyboards.

A work in progress implementation of the project can be found on GitHub[8], with an online work-in-progress documentation available as well[9]. It is possible that a more stable version can be published to the Python Package Index (PyPi) somewhere in the near future.

[8] https://github.com/pedromsilvapt/miei-dissertation
[9] https://pedromsilvapt.github.io/miei-dissertation/

The current version is already very usable in practice, which can be seen both in the examples provided in this paper (alongside with the generated audio and music sheets, both created by our application already) as well as many different experiments that have already been developed during this dissertation. We do think in our (obviously biased) humble opinion that the project has a lot of potential, particularly to serve as a swiss army knife, extensible and customizable solution for music creation and experimentation.

References

1. Sam Aaron. Sonic pi – performance in education, technology and art. *International Journal of Performance Arts and Digital Media*, 12(2):171–178, 2016. doi:10.1080/14794713.2016.1227593.
2. Samuel Aaron, Dominic A. Orchard, and Alan F. Blackwell. Temporal semantics for a live coding language. In *FARM '14*, 2014.
3. The abc music standard - the tune body, December 2011. URL: http://abcnotation.com/wiki/abc:standard:v2.1#the_tune_body.
4. General midi 1 sound set. URL: https://www.midi.org/specifications-old/item/gm-level-1-sound-set.
5. Guido Gonzato. Making music with abc 2, 2019.
6. Gareth Loy. Musicians make a standard: the midi phenomenon. *Computer Music Journal*, 9:8–26, 1985.
7. James McCartney. Rethinking the computer music language: Supercollider. *Computer Music Journal*, 26(4):61–68, 2002. doi:10.1162/014892602320991383.
8. Bob Nystrom. Crafting interpreters, 2020.
9. Yann Orlarey, Dominique Fober, and Stéphane Letz. FAUST : an Efficient Functional Approach to DSP Programming. In Editions DELATOUR FRANCE, editor, *New Computational Paradigms for Computer Music*, pages 65–96. HAL, 2009. URL: https://hal.archives-ouvertes.fr/hal-02159014.
10. Pep 255 – simple generators. https://www.python.org/dev/peps/pep-0255/, May 2001. URL: https://www.python.org/dev/peps/pep-0255/.
11. Quantization music. https://en.wikipedia.org/wiki/Quantization_(music). URL: https://en.wikipedia.org/wiki/Quantization_(music).
12. Ge Wang, Perry R. Cook, and Spencer Salazar. Chuck: A strongly timed computer music language. *Computer Music Journal*, 39(4):10–29, 2015. doi:10.1162/COMJ_a_00324.

Towards the Identification of Fake News in Portuguese

João Rodrigues
Iscte – Instituto Universitário de Lisboa, Portugal
http://www.iscte-iul.pt
jfcrs@iscte-iul.pt

Ricardo Ribeiro
Iscte – Instituto Universitário de Lisboa, Portugal
INESC-ID, Lisboa, Portugal
http://www.inesc-id.pt
ricardo.ribeiro@iscte-iul.pt

Fernando Batista
Iscte – Instituto Universitário de Lisboa, Portugal
INESC-ID, Lisboa, Portugal
http://www.inesc-id.pt
fernando.batista@iscte-iul.pt

Abstract

All over the world, many initiatives have been taken to fight fake news. Governments (e.g., France, Germany, United Kingdom and Spain), on their own way, started to take action regarding legal accountability for those who manufacture or propagate fake news. Different media outlets have also taken a multitude of initiatives to deal with this phenomenon, such as the increase of discipline, accuracy and transparency of publications made internally. Some structural changes have lately been made in said companies and entities in order to better evaluate news in general. As such, many teams were built entirely to fight fake news – the so-called "fact-checkers". These have been adopting different techniques in order to do so: from the typical use of journalists to find out the true behind a controversial statement, to data-scientists that apply forefront techniques such as text mining and machine learning to support the journalist's decisions. Many of these entities, which aim to maintain or improve their reputation, started to focus on high standards for quality and reliable information, which led to the creation of official and dedicated departments for fact-checking. In this revision paper, not only will we highlight relevant contributions and efforts across the fake news identification and classification status quo, but we will also contextualize the Portuguese language state of affairs in the current state-of-the-art.

2012 ACM Subject Classification Computing methodologies → Natural language processing

Keywords and phrases Fake News, Portuguese Language, Fact-checking

Digital Object Identifier 10.4230/OASIcs.SLATE.2020.7

Funding This work was supported by national funds through FCT, Fundação para a Ciência e a Tecnologia, under project UIDB/50021/2020.

1 Introduction

The way in which each of us, regular consumers of contents in the environment that surround us, bridges ignorance and knowledge has been changing over time. This bridge, the channel responsible for making available and disseminating "common interest" content, has been suffering changes in its form, content and perception of reliability, from the consumer's perspective. Contrary to the period prior to the beginning of the Internet, these interventions, that moved according to the political, economic, social and scientific context of each society, are now at the mercy of a new context that has been gaining strength in recent years –

technology. Since the beginning of our existence, until the early 2003, humanity has generated 5 Exabytes of data [1]. Today, that same volume is produced in only two days. In parallel with this fact, the access points to every kind of information also grew, both for information and misinformation. Today, as we live in a world where the surging of the aforementioned data is dramatical, where the struggle for audiences on traditional media increases and new forms of information are now found in uncontrollable proportions, and where the thoroughness in the management and proliferation of information is declining, it is urgent to provide ourselves a critical and attentive eye to fight the avalanche of disinformation to which we are exposed every day. In the last decade, more traditional information channels such as newspapers and television have been forced to give space, and consequently power, to a new giant phenomenon that has been conquering the market – the social media. This migration of content consumption is essentially due to the popularity that certain social platforms, such as Facebook and Twitter, have started to gain in society [6]. With the emergence of social media, both positive and negative aspects have shown up in terms of impact for its users. On one hand, social networks have brought to life a tool that, due to its regular use in conjunction with its massive popularity, allowed not only an easy way to search for others, but also a huge ease in the almost instantaneous proliferation of news. Hence, news with transverse interest to the entire population, such as the reporting of events in times of crisis, can be obtained almost in real time, either through official news channels or by any user who uses social platforms. Despite this positive side, social media have also seemingly harmed our society in a variety of ways and fields of interest. In the traditional media field, the way that these large entities reached their listeners had to be rethought and reformulated, since there was a major shift in the interest in consumption of news by their target audience towards social media. In fact, nowadays it is quite easy for a user without any contractual affiliation to an audiovisual entity to achieve more views, in a specific content shared only by himself, than some contents presented on FOX News, CNN, and New York Times [3]. Media outlets, in order to avoid completely losing the race for the attention of their target audience, were forced to emphasize and focus on the number of views / clicks of their publications at the prejudice of their content [6]. Ethics, integrity and accountability have been transformed into sensationalism, views number and, ultimately, greed. With this, the perfect environment for the appearance of fake news is settled. News with a willful lack of attention to source confirmation, fake news, misleading news, rumors and especially click bait news, which the only goal is to attract the user to a content that seems to be relevant and interesting, but, after a quick glance at said content, it ends up being far below the user's expectations [2].

However, despite the fact that fake news are recently growing in the most traditional media outlets, it was not here that they have taken such proportions for the first time. In 2016, after the elections in the United States of America that resulted in the victory of Donald Trump, it was immediately possible to realize, in a clear way, that great consequences come from the proliferation of fake news on a large scale. Many were those who addressed this topic and concluded that most of the most debated fake news preceding the election favored Donald Trump over Hillary Clinton. Furthermore, it is unanimous that Donald Trump would never win the elections without the influence of these fake news [3].

The exploration of tasks such as the detection and classification of "Fake News" is quite recent. Due to the enormous media exposure in which the subject has been involved, essentially after the North American elections in 2016 where, allegedly, Russian influence jeopardized the outcome of the results [8], a boom of contributions and initiatives began to take form. Together, the global community started to develop a common sense of urgency to address the problem and started, as a whole, searching for different solutions and approaches [6].

Detection and classification tasks deal with unstructured textual information using NLP (Natural Language Processing) techniques. Different techniques and methodologies provide quite different results, and a large number of these tools can be applied within the scope of this paper.

2 Related Work

In this section, we will begin by conducting an analysis of the different definitions of "Fake News". Subsequently, a survey and its respective analysis will be presented, regarding the most used and recommended datasets by the academic and business community in an attempt to find a resolution to this type of problems and concerns. For each one of them, a summary is made with a brief introduction to its structure and composition, a mention to the authors who contributed most to its exploration and the most used features in each of the datasets. Finally, the last point of this section deals with the different methods/classifiers used in this type of cases, associating, to each one, the datasets in which they were used and the respective authors.

2.1 Fake News

The idea behind the "Fake News" concept is not a novelty. In fact, if we go back a few years, to the time when the Internet did not even exist, and despite the terms not being the exact same, the idea of misinformation and disinformation was already circulating in society and in the traditional media of these times. There are many historical examples that support the aforementioned statement. As early as 1835, for instance, the first major hoax manufactured by the New York Sun newspaper appeared, where countless articles reporting the discovery of life on the moon were published [3]. Already in 2006, another major fabrication of false news appeared in Belgium, where a Belgian television station reported that the Flemish parliament had declared independence from Belgium [3]. At this time, although the market was not as fragmented as it is today, and the fact there were very few media outlets, the power that these entities had at their disposal was already unanimously recognized. Information is power, and the greater the power, the greater the appetite for promiscuous interests and consequent corruption [7].

As for the term Fake News itself, its definition began to gain popularity in the period during and after the North American elections of 2016, which culminated in the election of Donald Trump as president of the United States [3]. Also, alongside this term, two other are now commonly used to describe this phenomenon: misinformation and disinformation. However, these words are often incorrectly used for the same purpose, while they mean two different things. Both are used to refer to any spread piece of content that is false or misleading, however, there is a key difference - the intent behind each content. "Disinformation", in contrast to "Misinformation", is a term used for describing any false or misleading content that is intentionally spread while knowing about the content's lack of truth.

Many papers have emerged after Donald Trump's election, and plenty suggested different definitions for the term "Fake News". The most consensual definition used by most scholars emphasizes the importance of intention and verification: fake news are any and all new news that are proven intentionally false [3]. Therefore, any news that, through different sources, can be disproved, proving to be categorically false, or any news through which the author has the clear intention of misleading, are considered as fake news.

Other authors look at the concept of fake news from different perspectives. In [7], a definition is given regarding three different aspects: publications based on manufactured content; publications inserted in the context of large fraudulent and defamatory campaigns;

and humorous publications. The work done by [6] also mentions three different aspects - humorous content, the need for verifiability, and a new perspective presented as "malicious content". According to these authors, it is necessary to consider the humorist content as misleading, since although it is directed to a public that recognizes the author's humorist intention at the outset, there is also a large portion of the community that fails to correctly identify it. Along with this definition comes the concept of "malicious content", which represents the definition of intention, as mentioned above (disinformation).

The European Commission, a political independent entity with numerous goals, such as the creation of legislation, policies and action programs that cross the interests of the whole European Union, suggested, in 2018, the creation of a group of highly specialized experts in the subject of fake news. This group had a mission - to ensure that the democratic process of the 2019 European elections ran without any kind of interference from both misinformation and disinformation contents. At the time, this group defined the operational concept of Fake News as "all information which is proven to be false or misleading and which is created, presented and disseminated in order to obtain economic advantage or to deliberately mislead the public, and which is likely to cause public harm" [8]. This was the first concept to address the importance of public harm and economic advantages. With these, public harm arises and it is likely the most blatant and alarming consequence of the fake news phenomenon. The authors define it as all the threats to democratic political processes and public goods such as health, environment and security. On the other hand, we also have one of the causes of the immense growth of Fake News contents and consequent public harm - economic interests. Economic lobbies, entities who usually place their interests above the common human being, are one of the most important reasons for the large investment in fake news. This, combined with political interests as well, causes a great deal of pressure and influence on traditional media, coming from multinational companies, the state itself and magnanimous entities.

Finally, in the National context, the definition of fake news by the ERC, the Regulatory Authority for the Media in Portugal, appears. In 2019, the ERC conducted a study entitled "Disinformation – European and National Context", which deals with this subject in great detail [8]. Fake News are defined by this entity in a very similar way to the one given by the European Commission, but it adds a new perspective which should be taken into account in this context. ERC mentions that a news story, in its definition, can never be false, but that the contents of the narratives that are inserted in it, can be false or misleading. This means, according to the author, that labelling any news as a "Fake New" might be a little abusive, semantically speaking, and misleading.

2.2 Datasets

This section is dedicated to the detailed analysis of the most used datasets in the literature in the context of fake news. For each one an overview is made, specifying when, by whom and for what purpose it was created. An analysis of each one's content is also made, such as the number of records, the different labels and the variables that define it. Finally, a comparative analysis of the different authors is presented, defining who made use of each of the datasets, the approach taken by each one regarding the basic textual processing performed, the NLP tasks performed, and the textual representations implemented, which will serve as input features to the machine learning algorithms presented in the next section.

2.2.1 Fake News Challenge (FNC-1)

The creation of this dataset took place in a challenge called "Fake News Challenge", in 2017, and counted on the joint effort of 100 volunteers from the academic and business fields [14]. The goal of this challenge was to find new methods and approaches that would be useful to

present solutions to fight fake news. This dataset contains a set of news written in English. The dataset contains about 50,000 associations of statements with news. Each statement is associated with a particular news item and also with a label. The dataset is rather unbalanced in the sense that it has four different labels with a scatter sample distribution between them.

Table 1 Records by Label (FNC-1).

Label	Records	Percentage
Unrelated	36.545	73,1%
Disagree	8.909	17,8%
Agree	3.678	7,4%
Discuss	840	1,7%

This dataset has a different nature than the one expected in a typical Fake News problem. Often this problem is thought and addressed for the classification of titles and news bodies from a dichotomous perspective (usually called "truth labeling"), i.e., true or false. The purpose of this challenge is different: it is about trying to understand what is the "posture/relationship" of one statement before another, or a set of others (news/text body). Thus, the objective is to classify an affirmation using one of the previously mentioned labels: "Discuss" means that the body text neither confirms nor denies the statement; "Unrelated" means that there is no relationship between the body text and the statement; "Disagree" means that the body text does not agree with the statement; "Agree" means that the statement and the body text are related and agree with each other. According to the authors in [14], this approach is not a substitute for the truth labeling, but a means of supplementing it. The decision was made based on talks with journalists and fact-checkers where both parties mentioned that it is quite difficult to make a truth labeling classification. Both mentioned that they would rather have a semi-automatic solution to assist them in their work than a fully automatic solution whose performance would be far below expectations.

Many studies were based on this challenge and the respective dataset, and today this is one of the most used and explored datasets by all fake news researchers. Thus, each author has tried to approach it in his own way, using different tasks of text processing approaches and different types of representations of the text. These different representations form the features that will serve as input to the later machine learning and deep learning tasks.

Table 2 Pre-processing tasks and Textual Representations by Author (FNC-1).

<table>
<tr><th rowspan="2">Author</th><th colspan="2">Pre-processing</th><th rowspan="2">Textual Representation</th></tr>
<tr><th>Basic processing</th><th>NLP</th></tr>
<tr><td rowspan="2">[9]</td><td rowspan="2">Regular expressions</td><td>Lemmatisation</td><td rowspan="2">Glove Embeddings</td></tr>
<tr><td>Standford NER</td></tr>
<tr><td rowspan="7">[5]</td><td rowspan="2">Label mapping</td><td rowspan="7">Sentiment Analysis</td><td>Word-Embeddings</td></tr>
<tr><td>Google News CNN</td></tr>
<tr><td rowspan="2">Tokenization</td><td>Number of n-grams</td></tr>
<tr><td>TF-IDF</td></tr>
<tr><td rowspan="2">Stemming</td><td>word2vec</td></tr>
<tr><td>SVD</td></tr>
<tr><td>N-grams generation</td><td>Number of positive and negative words</td></tr>
</table>

In [9], the authors started by using regular expressions to eliminate all unwanted links. Following that, they made use of lemmatization - the process of grouping together the inflected forms of a word as a single item. The Stanford Entity Name Recognizer was also used to replace entities such as names, organizations, and locations. Finally, pre-trained GloVe Embeddings models were used to represent all words in global semantic vectors [12]. In [5], the authors describe the work that led them to win first place in the competition. The "Solat In the Swan" team chose to use the combination of two classification approaches, and to do so they needed to create two sets of features. In order to get to both sets it was necessary, first of all, to perform a pre-processing step. In this phase, the tokenization of both titles and news were made and stemming was applied to each of the tokens. Finally, the various unigrams, bigrams and trigrams were generated from the list of tokens. Once the pre-processing was carried out, the first textual representation was created. For this, pre-trained Google News vectors were used, both on the titles and on the news themselves. More traditional features were applied in the second set: n-grams count; TF-IDF; SVD-based; word2vec and sentiment features.

2.2.2 BuzzFace

This dataset was developed from a set of news items published by the media company "BuzzFeed" after the 2016 North American elections and was manually tagged by journalists [16]. This dataset was also enriched with information from Facebook, such as comments, number of shares and reactions to the news published by the company on its website.

The dataset has a total of 2,282 entries. Each entry line corresponds to a share on Facebook from an official media outlet's page. Each post can have one of four possible classifications: "no factual content", "mostly true", "mostly false" and "mixture of true and false".

Table 3 Records by Label (BuzzFace).

Label	Records	Percentage
True	1.665	73%
Non Factual Content	274	12%
Mixture of True and False	251	11%
Mostly False	91	4%

This dataset is also of great importance since it has features related to the "context" of the news, i.e., number of shares, reactions and comments to the news post. This is the only dataset contemplated in this study that contains these types of features, which also means that it is the only one that does not depend directly on the intrinsic content of the news.

Table 4 Pre-processing and Textual Representations by Author (BuzzFace).

Author	Pre-processment		Textual Representation
	Basic processing	NLP	
[15]	-	-	Bag of words
	-	POS Tagging	Number of pronouns, verbs, adverbs, hashtags, punctuation.
	By "Linguistic Inquiry and Word Count"	By "Linguistic Inquiry and Word Count"	Psycholinguistic features (detection of biased and persuasive language)
	By "Google's API"	By "Google's API"	Semantic features (toxicity)
	By "Text Blop's API"	By "Text Blop's API"	Features subjectivity (subjectivity and feeling)

In [15], the authors start by ensuring a greater balance of the dataset, making the distribution of the records (for each label) more uniform. Thus, only two labels were considered: true and false, all cases with the label "mostly true" became "true"; the cases with the label "no factual content" were removed; and the records with the two remaining labels were converted to the label "false". The authors then apply and detail a set of features appropriate to the problem in question, in which context-related features are also included. The features are thus divided into three main groups: (1) features extracted from the news content, (2) features extracted from the source of the news and (3) features extracted from the environment.

2.2.3 WSDM Cup

This dataset is one of the most recent in the literature and was developed with the purpose of being the object of study for those who participated in the challenge of the international conference WSDM (Web Search and Data Minning), WSDM Cup, which took place in 2019, organized by the ACM (Association for Computing Machinery).

The dataset in question was developed by ByteDance, a Chinese internet and technology company, which owns an online news platform. One of the greater challenges faced by ByteDance is the fight against fake news. To this end, the company has created a database to colect all kinds of fake news, so that all news can be properly verified regarding their veracity before being presented in the platform [10].

The dataset was originated from the database mentioned above, counting with a total of 360,767 records. Each record has in its constitution a title of a false news A, a title of a news B (news to be classified) and its label (Agreed, Disagreed or Unrelated). The objective here is to try to understand if the news item B addresses the same subject and agrees with the title A (agreed), which makes the news item B false; to try to understand if the title B does not agree with the news item A (disagreed), making the news item B true; or to identify that the news B has no relation whatsoever with news item A (unrelated). The dataset has news titles in two different languages, Chinese and English, its configuration follows a weight of 75% for training and 25% for testing, and presents the following label distribution:

Table 5 Records by Label (WSDM Cup).

Label	Records	Percentage
Unrelated	246,764	68.4%
Agreed	104,622	29.0%
Disagreed	9,379	2.6%

Table 6 shows some of the works using this dataset.

Table 6 Pre-processing and Textual Representations by Author (WSDM Cup).

Author	Pre-processing		Textual Representation
	Basic processing	NLP	
[10]	Dataset augmentation	-	Set of 25 pre-trained BERT's.
	Stopwords removal		
[13]	Dataset increase	N-grams	Text Based
	Text to lowercase		Statistics
	Add spaces between punctuations		Graph Based
	Tokenization		KNN (BERT's)

"Travel", the team that came second in the competition, developed an approach that achieved a "weighted accuracy score" of 0.88 [10]. Given the nature of the problem, the authors have chosen to attack the problem using NLI (Natural Language Inference) techniques. NLI is a subarea of NLP and its objective is to recognize textual implications - RTE (Recognizing Textual Entailment) - that is, in this case, from news B which is given as false, to be able to infer a hypothesis (which label characterizes news A) from a textual premise through the semantic similarities between them. Regarding the creation of the features that will feed the machine learning algorithms, the authors in question decided on three main steps. First, since the dataset was not properly balanced regarding the distribution of records per label, and in order to avoid an over-fit, an increase of quantity of the data was made through the transitivity of semantics. That is, if title A is related to title B and if title A is related to title C, then B and C are related. Subsequently, all stop words were removed, both in Chinese and English news. Finally, to address the problem of text representation, the authors chose to use BERT, a pre-trained linguistic model created by Google that has recently been gaining popularity.

In [13], the challenge's winning team ("IM"), suggests a pre-processing approach and textual representation similar to that used by the "Travel" team, but with some nuances. As far as pre-processing is concerned, the author suggests also separating text scores (by placing spaces) and the tokenisation of all titles. In the textual representation, an ensemble of features is suggested for input regarding future classification algorithms. The first set of features are "Text Based". Here all textual features are covered, such as the generation of n-grams of words and characters. After their generation, distance measurements are applied to pairs of titles, such as cosine, euclidean, city-block, jaccard, or simple addition and subtraction. The second set of features are "Statistics". This is where word counts are present, as well as stop words, tokens, characters, or a simple comparison of the textual size

of the title pairs. The third set of features are the "Graph Based". In this case, the texts of the titles are represented as graph networks. The objective of these networks is to make a representation of each title (a node of the graph) and of each pair of titles. Through metrics, such as minimum or maximum distance between nodes and news pairs, assumptions can be made about their relation. Finally, the features "KNN" represent BERT embeddings with reduced dimensionality.

2.2.4 Fake.Br Corpus

For the Portuguese language, Fake.Br Corpus is the only dataset we found in this context. According to [11], this is the first fake news dataset with Brazilian Portuguese news and also the first dataset in the Portuguese language. The dataset resulted from a joint effort of researchers and analysts who gathered and classified news manually. The dataset contains 7200 news items and is divided in a balanced way regarding the number of records that are associated with the different labels (Table 7).

Table 7 Records by Label (Fake.Br Corpus).

Label	Records	Percentage
True	3,600	50%
False	3,600	50%

The authors defined a time span of two years (January 2016 to January 2018) and only collected news that were inserted in it. Some news, however, reference other news in previous time spaces. Other relevant information, such as the author of the news, the date of publication, the number of views and comments, were kept. The news were manually tagged and, for each false one, the authors used a semi-automatic process to find true news that could prove the tag on the corresponding "false" ones. The false news were manually extracted from four different newspapers. After this process, through web scrapping, 40,000 real news were taken from other sites, based on the most frequent words, verbs and names that were in each of the fake news previously taken. Then, a lexicon similarity measure (cosine) was applied to determine which were the true news that were closest to the fake news. After this process was completed, having already a smaller number of news items, they made the manual selection of the news items based on their actual degree of similarity.

Once again, the work on this dataset explored different pre-processing procedures and different types of text representations.

In [11], the authors use various approaches to the representation of the news text. Apart from the most common forms of textual representation (bag of words, term count, etc.), the authors also explore the use of linguistic features that may also be interesting, such as pausality, uncertainty, expressiveness, non-immediacy, and number of semantic classes.

In [4], the author presents two alternative approaches to the work of [11]. First, the author mentions the use of the chi-square method as a means of selecting the most relevant terms that resulted from his pre-processing task. According to his work, this method is an added value for textual processing tasks in the Portuguese language. Finally, the author states that textual representations based on word embeddings have shown better results than classical representations based on term-to-document matrices.

Table 8 Pre-processing and Textual Representations by Author (Fake.Br Corpus).

Author	Pre-processing		Textual Representation
	Basic processing	NLP	
[11]	Stopwords Removal	Stemming	Bag of Words
	Punctuation Removal		
	-	POS Tagging	Number of each Part of Speech takes place
	-	Enriched Lexicon	Number of semantic classes
	Words Removal	-	Pausality (number of punctuation characters)
	-	POS Tagging	Expressiveness (sum of adjetives and adverbs over the sum of nouns and verbs)
	Extraction of Specific Words (can, might)	-	Incertainty (number of modal verbs)
	-	POS Tagging	Non-Immediacy (number of first and second pronouns)
[4]	Stopwords Removal Tokenisaton	Stemming	Word-Embeddings
		Lemmatisation	
		Chi-square	

2.3 Methods

This section is dedicated to the analysis of the most commonly used methods in fake news detection. There are several ways to approach this topic according to the literature, however, most approaches cast this as a classification problem. In a classification problem the aim is to be able to associate a label, for example true or false, with small (in the case of a title, for example) or large (in the case of a news' body text, for example) textual portions. In order to respond to this task, most of the research body dedicated to this subject implements machine learning and especially deep learning techniques [6].

Within the classification problem, authors employ different methods depending on the features they have at their disposal. Typically, most of the approaches focus on using features extracted from the news content itself. However, other sets of features, such as information related to the source or context of the news [15], present another type of detail that may enrich the analysis. Regarding fake news classification strategies, a combination of methods is typically used, the so-called "ensemble models". Since in most works that ensemble methods are used the evaluation metrics refer to the set and not to each method itself, it is relevant to make a comparative analysis not only between methods, but also between standalone methods and ensemble methods.

There are several methods of machine learning that have been applied in the task of fake news detection. Of the most recent and best performing methods, there are three that typically stand out from the more traditional methods (such as KNN, Naive Bayes, Decision Trees, etc). The first method is the SVM. The SVM (Support Vector Machine) is a discriminative classifier formally defined by a hyper plane of separation [6]. This method has been used in several fake news tasks. In two of the four datasets mentioned in the previous section, there are authors who propose the use of the SVM in isolation [11, 15].

In [11], in a study using the dataset "Fake.Br Corpus", the authors used the SVM with only content features. Among several combinations of textual representations, the best performance obtained was a F1-score of 0.89. In [15], in a study using the dataset

"BuzzFace", the authors made use of all kinds of features and in all of them applied an SVM. In this work, the performance was a F1-score of 0.76, a performance that was below other methods also applied, such as Random Forest (0.81 F1-score) and Gradient Boosting (0.81 F1-score). An interesting approach used SVM [18]. The authors used a variation of SVM called "Graph-Kernel-Based SVM" to identify rumors using propagation structures and content features. In this study, the authors reported an accuracy of 0.91.

Another method that has been gaining prominence over more traditional methods is the "Gradient Boosting" approach. Gradient Boosting is a meta algorithm based on decision trees, and it is used to reduce biased predictions. Catboost and LightGBM, for instance, are versions of Gradient Boost that have been gaining popularity in recent times due to their advantages of fast processing and high prediction performance [13]. These types of algorithms typically appear in stance detection problems, but can also appear in truth labeling problems [9, 13, 15]. In [9], in a specific stance detection problem, an accuracy of 0.83 was achieved for this method alone. Although attaining a good performance, gradient boosting ended up behind two other classification methods also tested: LSTM and BiLSTM neural networks models. In [15], in a specific truth labeling problem, gradient boosting was also used achieving the best performance (0.81 F1-score) against other algorithms: SVM, Naive Bayes, and Random Forest.

Finally, deep learning models have brought great advances in several areas of Artificial Intelligence, such as image identification, speech recognition, and textual processing [13]. In this topic, the most commonly used neural networks are CNN (Convolutional Neural Networks) and RNN (Recurrent Neural Networks). In [4], a study on the dataset "Fake.Br" using neural networks, more specifically a CNN, achieved a performance of 0.91 accuracy which translated in a very successful result for a truth labeling problem. In [9], the authors using LSTM (Long Short-Term Memory) and BiLSTM (Bidirectional Long Short-Term Memory) neural networks managed to obtain a 0.92 and 0.93 accuracy, respectively, in this work regarding stance detection task.

As previously mentioned, ensemble methods have been a very successful approach to fake news detection. Typically combining deep learning and traditional machine learning techniques, these approaches achieved better results, in most cases. In the two public challenges that are described in this paper, WSDM Cup and Fake News Challenge, the winning teams made use of an ensemble method. In the Fake News Challenge 2016 (FNC-1), the authors who came first [5], after several attempts to apply methods individually, concluded that the best performance was achieved by combining the methods they were exploring. Thus, the best performing approach was a combination of CNNs and Gradient-Boosting Decision Trees methods. This approach had an average weighted score of 82.02%.

In [10, 13], works that finished first and second in the "WSDM Cup 2019" competition, the response to the problem also included a set of methods. The second placed team [10] used a total of 6 methods (3 SVM's, 1 Naive Bayes, 1 KNN and 1 Logistic Regression), obtaining an average accuracy score of 88.15%. The first team [13] chose to combine 28 methods (18 Neural Network Models, 9 Tree Based Models and 1 Logistic Regression), resulting in an average accuracy score of 88.28%.

3 Challenges for European Portuguese

Fake news detection, or fact-checking, has become a prominent facet of political, economic, sports and social news coverage. This task is defined by employing a variety of methodological practices, such as treating a statement containing multiple facts as if it were a single fact and

categorizing it as accurate or inaccurate. These practices share the tacit presupposition that there cannot be genuine political debate about facts given that facts are unambiguous and not subject to interpretation. Therefore, when the black-and-white facts, as they appear to the fact checkers, conflict with the claims produced by politicians, these same fact-checkers are able to detect and expose lies [17].

In the past years, fact-checking has been a regular task that any journalist must practice in their work on a daily basis. It is not something they might do, but something they must do, since it is what their journalist deontological code implies. However, as mentioned, with the exponential growth of social media usage, and the urge of the traditional media to apply different methodologies to be able to keep up with the business and compete with those new forces, in many cases that deontological code has been put aside. Furthermore, since there does not seem to be, yet, legal consequences, at least not in Portugal, for the fabrication and propagation of fake news (mostly due to freedom of speech being a sensitive topic as we live in a democracy), they seem to have started appearing everywhere, from social media to most traditional media outlets.

With this high competition between media sectors alongside with this sense of impunity, sensationalism and made up news started to emerge. Today, not only are we living in the era of data – an era marked by the privilege of being able to access a massive amount of information (more than we ever could before) – which is already difficult to process, but we are also living in an era of disinformation. This is why it is imperative to have solutions to allow people to be better and well informed. Having this in mind, many journals around the world seem to have decided to adapt and build teams that are fully dedicated to fact checking duties. Although their focus, as any other private company, is also to make a profit, the strategy that they adopt is different - they focus on the quality and reputation of the group, and that is why, even knowing that this could cost more in the near-term, making sure that the information that they provide is reliable helps them build a sense of trust between the group and the community in the longer term.

In Portugal, as it was pointed out in the ERC study, there are two newspapers that have their own department of fact-checkers: **Polígrafo** and **Observador**. Both newspapers are certified by Poynter, the owner of the International Fact-Checking Network (IFCN), a unit fully dedicated to bringing together fact-checkers worldwide. This unit was launched in September 2015 in order to support a booming crop of fact-checking initiatives by promoting best practices and exchanges in this field. With this, both newspapers mentioned above should stand for Poynter principles.

Polígrafo is a recent digital newspaper launched by Sapo. It was announced for the first time in November 2018, during the Web Summit in Lisbon. Polígrafo is the first Portuguese newspaper dealing with fake news and its database has more than three thousand classified news. During the Covid-19 pandemic crisis, Polígrafo made a partnership with "Corona Verificado", a fact-check platform coordinated by the Brazilian "Agencia Lupa", which integrates information from 34 different fact-checkers entities from 18 Ibero-American countries. This platform alone has more than two thousand news classified.

Observador it is also a recent online newspaper, born in 2014. It defines itself as an independent and free online, daily newspaper that searches for the truth and submits to the facts. Observador stands for not being conditioned by partisan and economic interests or any group logic. They are accountable only to their readers. Over the years, Observador has been gaining a lot of popularity and respect by Portuguese population and even their peers. In 2015, the newspaper decided to create a section dedicated to fact-checking, and they became the first Portuguese newspaper having a department fully dedicated to fact-checking duties. Today, Observador has more than three hundred classified news.

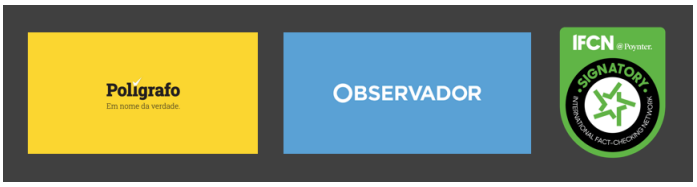


Figure 1 Polígrafo and Observador Recognition by Poynter.

These two sources can clearly constitute a fundamental resource for scientific research on automatic fake news detection in European Portuguese. Especially considering that specific aspects of language are cornerstone in this type of NLP task. The major challenges are constituting such resource and studying the suitability of the presented methods.

4 Conclusion

In this paper, we presented a detailed overview on automatic fake news detection. First, we introduced fake news regarding its context and definition according to the point of view of many different entities, from individual to national (ERC) and international ones (European Commission). Secondly, we presented the most used datasets in the context of fake news. For each dataset, we detailed its objectives, followed by how it was created, by how many records it has, and the distribution of entries by different target label. Also, for each dataset, an analysis was made on different works that used said dataset. The main papers that studied the datasets are here compared, not only by the type of pre-processing that they applied but also by their explored text representations. Thirdly, an overview of the different methods used in the last years was made, presenting their respective evolution and datasets in which they were applied. Finally, we discussed the importance of fact checking, where the need and the importance of those practices in our global society were shown. More specifically, in the case of the European Portuguese language, we presented the two major entities that are certified by Poynter as official fact-checkers, Polígrafo and Observador.

To conclude, most of the techniques necessary to successfully implement an automatic fake news detection system were addressed in this paper. From the most used preprocessing techniques, the most used classification methods (both for Portuguese and English datasets), to the resources available in Portuguese. This is a relevant asset for conducting future work on European Portuguese, where questions regarding the limitations of these resources and their impact on the Portuguese language itself can be, and should be, further analyzed and discussed.

References

1 Alberto Cairo. *The Functional Art: An Introduction to Information Graphics and Visualization*. New Riders, 2012.

2 Monther Aldwairi and Ali Alwahedi. Detecting fake news in social media networks. *Procedia Computer Science*, 141:215–222, 2018. doi:10.1016/j.procs.2018.10.171.

3 Hunt Allcott and Matthew Gentzkow. Social media and fake news in the 2016 election. *Journal of Economic Perspectives*, 31(2):211–236, 2017. doi:10.1257/jep.31.2.211.

4 Renan Rocha De Andrade. Utilização de técnicas de aprendizado de máquina supervisionado para detecção de Fake News. https://riuni.unisul.br/handle/12345/8649, 2019.

5 Sean Baird, Doug Sibley, and Yuxi Pan. Talos Targets Disinformation with Fake News Challenge Victory, 2017. URL: http://blog.talosintelligence.com/2017/06/talos-fake-news-challenge.

6 Alessandro Bondielli and Francesco Marcelloni. A survey on fake news and rumour detection techniques. *Information Sciences*, 497:38–55, 2019. doi:10.1016/j.ins.2019.05.035.

7 Yimin Chen, Niall J. Conroy, and Victoria L. Rubin. News in an online world: The need for an "automatic crap detector". *Proceedings of the Association for Information Science and Technology*, 52(1):1–4, 2015. doi:10.1002/pra2.2015.145052010081.

8 Entidade Reguladora para a Comunicação Social, editor. *A Desinformação - Contexto Europeu e Nacional*. Parlamento português, 2019.

9 Neema Kotonya and Francesca Toni. Gradual Argumentation Evaluation for Stance Aggregation in Automated Fake News Detection. In *Proceedings of the 6th Workshop on Argument Mining*, pages 156–166. ACL, 2019. doi:10.18653/v1/w19-4518.

10 Shuaipeng Liu, Shuo Liu, and Lei Ren. Trust or Suspect? An Empirical Ensemble Framework for Fake News Classification. *Proceedings of the 12th ACM International Conference on Web Search and Data Mining, Melbourne, Australia*, pages 1–4, 2019. URL: http://www.wsdm-conference.org/2019/wsdm-cup-2019.php.

11 Rafael A. Monteiro, Roney L.S. Santos, Thiago A.S. Pardo, Tiago A. de Almeida, Evandro E.S. Ruiz, and Oto A. Vale. Contributions to the Study of Fake News in Portuguese: New Corpus and Automatic Detection Results. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 11122 LNAI:324–334, 2018. doi:10.1007/978-3-319-99722-3_33.

12 Jeffrey Pennington, Richard Socher, and Christopher Manning. GloVe: Global Vectors for Word Representation. *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, page 4, 2014.

13 Lam Pham. Transferring, Transforming, Ensembling: The Novel Formula of Identifying Fake News. *Proceedings of the 12th ACM International Conference on Web Search and Data Mining, Melbourne, Australia*, 2019.

14 Dean Pomerleau and Delip Rao. The fake news challenge: Exploring how artificial intelligence technologies could be leveraged to combat fake news., 2017. URL: http://www.fakenewschallenge.org/.

15 Julio Reis, André Correia, Fabrício Murai, Adriano Veloso, and Fabrício Benevenuto. Supervised Learning for Fake News Detection. *IEEE Intelligent Systems*, 34(2):76–81, 2019. doi:10.1109/MIS.2019.2899143.

16 Giovanni C Santia and Jake Ryland Williams. BuzzFace : A News Veracity Dataset with Facebook User Commentary and Egos. In *Proceedings of the Twelfth International AAAI Conference on Web and Social Media (ICWSM 2018)*, pages 531–540. AAAI, 2018.

17 Joseph E. Uscinski and Ryden W. Butler. The Epistemology of Fact Checking. *Critical Review*, 25(2):162–180, 2013. doi:10.1080/08913811.2013.843872.

18 Ke Wu, Song Yang, and Kenny Q Zhu. False Rumors Detection on Sina Weibo by Propagation Structures. In *IEEE 31st International Conference on Data Engineering*, pages 651–662, 2015.

Development of Q&A Systems Using AcQA

Renato Preigschadt de Azevedo[1]
Centro Algoritmi (CAlg-CTC), Department of Informatics, University of Minho, Braga, Portugal
`https://acqa.di.uminho.pt`
renato@redes.ufsm.br

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

Pedro Rangel Henriques
Centro Algoritmi (CAlg-CTC), Department of Informatics, University of Minho, Braga, Portugal
pedrorangelhenriques@gmail.com

Abstract

In order to help the user to search for relevant information, Question and Answering (Q&A) Systems provide the possibility to formulate the question freely in a natural language, retrieving the most appropriate and concise answers. These systems interpret the user question to understand his information needs and return him the more adequate replies in a semantic sense; they do not perform a statistical word search like happens in the existing search engines. There are several approaches to developing and deploying Q&A Systems, making it hard to choose the best way to build the system. To turn easier this process, we are proposing a way to automatically create Q&A Systems (AcQA) based on DSLs, thus allowing the setup and the validation of the Q&A System independent of the implementation techniques. With our proposal (AcQA language), we want the developers to focus on the data and contents, instead of implementation details. We conducted an experiment to assess the feasibility of using AcQA. The study was carried out with people from the computer science field and shows that our language simplifies the development of a Q&A System.

2012 ACM Subject Classification Software and its engineering → Source code generation; Software and its engineering → Domain specific languages; Software and its engineering → Design languages

Keywords and phrases Question & Answering, DSL, Natural Language Processing

Digital Object Identifier 10.4230/OASIcs.SLATE.2020.8

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

With the increased usage of personal assistants, which allow the user to ask questions and get answers from various subjects, these types of systems are being used by a large number of people. Approximately forty or fifty years ago, studies began on Q&A (Question and Answering) Systems [39, 9, 33, 16], but due to computational limitations, these systems had limited scope. Some Q&A Systems were more or less successful; some of them were discontinued, demonstrating the difficulty of building and maintaining a system capable of understanding natural language as a human can. Even so, demand increased at a high pace, leading to researches to build better and more efficient systems.

[1] corresponding author

Questions are asked and answered several times per day by a human. Q&A Systems try to do the same level of interaction between computers and humans. This approach differs from standard search engines (Google, Bing, and other search engines) because it makes an effort to understand what the question expresses and try to give concise answers instead of using keywords from the question asked and provide documents as results.

A simple Q&A System is composed of several processes: question analysis, query processing (extraction of potential answers), and answer formulation [11]. To be able to create a functional Q&A System, all these processes have to be carefully chosen by the domain specialist and implemented by the programmer. The programmer has to be an expert in the chosen programming language, as well as the various libraries required to implement the system (Natural Language Processing, Neural Networks, Database Access, among others). These complex systems increase the possibility of errors occurring, making the implementation process more time consuming and costly.

Domain-specific Languages (DSLs) can simplify and accelerate the development of applications. The development of a DSL to construct a Q&A System allows the user to specify components used in these systems more abstractly. This approach makes the process of implementing the system more straightforward and less error-prone. To the best of our knowledge, this is the first work that uses DSL to create Q&A Systems. We did not identify any similar work to compare and discuss.

Taking into account the importance of Q&A Systems and the complex tasks that must be implemented to create such an intelligent tool, the goal for this paper is to discuss the design of AcQA (Automatic creation of Q&A Systems) DSL to make the development of these systems easier following a systematic and rigorous approach.

The structure of the paper is as follows: Q&A Systems are discussed in Section 2; in Section 3 some relevant aspects on the use of DSLs for automatic construction of language-based tools are presented; AcQA language is then presented in Section 4; Section 5 describes an experiment and discusses the results; and in Section 6 conclusions are presented and directions for future work are proposed.

2 Question & Answering Systems

The wideness of information available associated with the demand for direct answers from the users requires a different approach from standard search engines. The use of natural language to communicate with computer systems turns the information technology useful for all types of persons, allowing them to specify deeply the information needed. Q&A Systems are not new, but they still need to be improved in terms of answers accuracy, and in terms of knowledge domains. The main idea of these systems is to receive the user question and analyze not only the keywords but also the intention. To discover the intent within a question is not an easy task. Besides the capability of understanding the user question, the system must retrieve the best possible answers, ranking them.

There are several approaches in the literature explaining the construction of Q&A Systems [15, 25, 36, 38], explaining the typical sequence of development stages. At first, several technical approaches should be carefully studied to allow the processing of natural language. In the past, small Knowledges Bases were used, allowing the construction of simple Q&A Systems. These systems used simple schemas with a small number of entities and relations, ad-hoc approaches (manually constructed rules), among other strategies to create the knowledge base (KB). The KB was specifically tailored to the specific domain, needing much effort from the original designers of the Q&A System to add new content or add a new domain. These strategies do not support the construction of scalable systems and turn complex the development of open domain systems.

To construct a Q&A System, we need to develop three processes: question analysis, query processing (extraction of potential answers), and answer formulation. The techniques used for question analysis seeks to recover meaning from the input text, sometimes employing Natural Language Processing (NLP) to achieve the goal. Natural Language Processing is an area of computation that includes parsing, part-of-speech (POS) tagging, statistical models, ontologies, and machine learning [24]. Pattern matching and the use of tags can also be used to process the input text. Query processing approaches are responsible for handling the input text to create the queries necessary to extract relevant information from the KB. The answer formulation uses information gathered in question analysis and query processing to generate or retrieve possible answers.

The result of a Q&A System can be fragments of documents, a list of links to web pages, images, a simple and concise sentence, or a ranking of sentences. This work is focused on systems that retrieve one answer or a rank-ordered list of answer candidates.

Q&A Systems are classified as being closed or open domain. In open domain Q&A Systems, the questions are domain-independent, and the system aims to answer anything that the user asks. In this case, an extensive repository of information must be used, and the system must be able to answer questions of all kinds of subjects. Restricted domain Q&A Systems (closed-domain) work only with a specific domain, not being able to answer questions outside the proposed field. The information repository is based on a well-defined knowledge base, made out of data only related to the proposed field, being able to achieve better accuracy than open domain Q&A System.

Some examples of open domain Q&A Systems are Intelligent Q&A System based on Artificial Neural Network [2], Automatic Question-Answering Based on Wikipedia Data Extraction [22], A Content-Aware Hybrid Architecture for Answering Questions from Open-domain Texts [21], WolframAlpha [23] and IBM Watson [15]. Some examples of Closed domain Q&A Systems are Q&A System on Education Acts [28], Python Question Answer System (PythonQA) [35], and K-Extractor [4].

2.1 Available Q&A Systems

There are several Q&A Systems using NLP to process the user input data and provide answers accordingly. In the work proposed by Ramos et al. [35], the PythonQA system was developed to answers questions about the programming language Python. It was developed in the Python programming language, together with some libraries such as Natural Language ToolKit (NLTK) [8], Django, among others. The system processes the input from the user dividing a phrase into several components and trying to identify three main elements: action, keywords, and the question type. Then, these three elements are compared to the knowledge base, built with data from the Python Frequently Asked Questions, to retrieve and show answers to the users of the Q&A System.

MEANS [6] also used NLP to process the corpora and user questions. The medical subject is the domain that was used by the authors of this Q&A System. The knowledge base was created through sources of RDF annotated documents, based on an ontology. To provide better answers, the authors propose ten question types to classify user questions.

Other works use ontologies as KB, as in [34], [30], [7], [26]. The authors in [34] propose a Q&A System that uses ontology assistance, template assistance, and user modeling techniques to achieve 85% of accuracy in their experiments. The authors of [30] propose an algorithm to automatically update the ontology used by the system and use a semantic analyzer that operates on an ontology to extract answers.

To be able to answer questions about the United Kingdom Parliament education acts, a Q&A System was proposed in [28]. The knowledge base was made from the data publicly available from the United Kingdom parliament using NLP techniques. This proposed Q&A System uses only keywords from the user question, ignoring the question type and other pieces of information present in the user input text.

Some works use artificial intelligence (AI) to process questions and create the knowledge base for the Q&A System. In [10], a Q&A System, called AskHERMES, is proposed to solve complex clinical questions. The authors use five types of resources to construct the knowledge base (MEDLINE, PubMed, eMedicine, Wikipedia, and clinical guidelines). The user input question is classified by twelve general topics, made by a support vector machine (SVM). The authors process possible answers through a question summarization and answer presentation modules based on a clustering technique. Another work using AI is proposed by Weissenborn et al. [40], where the authors propose a Q&A System based on fast neural network techniques. According to results from their proposed system, the system uses a simple proposed heuristic to achieve the same performance compared to more complex systems.

There are Q&A Systems that work with different input, like images. In [14], an approach to generate image descriptions automatically is proposed. Firstly words describing the image are discovered and stored. Secondly, it generates sentences associating the objects in the picture. The final step is to rank the phrases according to the MERT [31] model to present the sentences to the user.

As it was said, the most well-known examples of open domain Q&A Systems are Wolfram[23] and IBM Watson [15]. WolframAlpha and IBM Watson are examples of this type of Q&A Systems. WolframAlpha is a well-established Q&A System to answer questions of any type and was initially a closed domain Q&A System to answer questions about mathematics. It allows the user to extend the version available online with the pre-existing knowledge base or to upload data through a paid subscription. IBM Watson is a Q&A System that was initially created to play in the Jeopardy TV quiz program. Watson is now an Artificial Intelligence framework provided by IBM. It allows working with a variety of areas, such as Q&A Systems and natural language processing. Watson is available through paid subscriptions.

3 Domain Specific Languages (DSL)

Domain-Specific Languages (DSLs) can simplify the development of applications [1]. According to Fowler [17], Domain-Specific Language is a computer programming language of limited expressiveness focused on a particular domain. Although the handicap of having to learn a new language[29], Domain-Specific Languages (DSLs) can accelerate and make simple the development of applications [1].

DSLs are relevant to developers for two main reasons: improving programmer productivity and allowing programmers and non-programmers to read and understand the source code. The improved programmer productivity is achieved because DSLs try to resolve a minor problem than general-purpose programming languages (GPL) [18], making it more straightforward to write and modify programs/specifications.

Since usually DSLs are smaller than GPLs, they allow domain specialists to see the source code and get a more general view of their business. DSLs offer the capacity to domain specialists to create a functional system, with no prior knowledge of GPLs.

What distinguishes DSLs from GPLs is the expressiveness of the language. Instead of providing all the features that a GPL must contain, such as supporting diverse data types, control, and abstraction structures, the DSL has to support only elements that are necessary to a real domain. Examples of commonly used DSLs, according to [18], are SQL, Ant, Rake, Make, CSS, YACC, Bison, ANTLR, RSpec, Cucumber, HTML, LaTeX.

Generative programming is related to the construction of specialized and highly optimized solutions through a combination of modules to decrease the conceptual gap between coding and domain concepts. This approach simplifies the management of several components, increasing efficiency (space and execution time) [13].

In that sense, generative programming recommends to apply separation of concerns, namely, to deal with one important element at a time, and combine these elements to generate a component; There is a clear separation from the problem and solution. Each component can be adapted for different scenarios using parameterization and creating different families of components, implying the management of dependencies and interactions to combine them.

Generative programming uses DSL at a modeling level [12] to allow users to operate directly with the domain concepts, instead of dealing with implementation details of GPLs. The system can automatically generate executable code from a textual or graphical DSL specification [13].

In this work, the concept of generative programming is applied through the use of a DSL to allow the automatic creation of Q&A Systems, avoiding the need to build the system line-by-line. A compilers generator is used to process the DSL formal specification (grammar) to build a Q&A System compiler. That new compiler will automatically produce the desired Q&A System taking into account the input description written using the DSL proposed. Section 4 presents the proposed approach.

4 AcQA – Automatic creation of Q&A Systems

The use of generative programming and domain-specific languages allows domain specialists to develop entire Q&A Systems without the need to code or have knowledge in GPLs. We used these concepts to enforce constraints and validate inputs, allowing domain specialists to build the Q&A System focusing on what techniques to use, rather than how to implement them.

In this Section, the domain-specific language AcQA (Automatic creation of Q&A Systems) is presented. AcQA allows an expert or regular user to specify this kind of system more straightforwardly than in a GPL. The focus is on the behavior of the whole system, rather than how to implement them. This approach allows the user to focus on the data that are made available for building the system knowledge base as well as change behaviors such as the techniques that need to be used to process the user inputs (questions) and its front-end (Web, WebService, others). Experienced Q&A Systems developers who already have tools to construct Q&A Systems can extend the AcQA language to use these tools. Domain experts who have no previous experience developing Q&A Systems can develop a functional Q&A System using only the AcQA DSL.

To achieve the objective of generating a Q&A System, we propose to apply the concepts discussed in the Sections 2 and 3. We developed the grammar for AcQA DSL to be recognized by the AcQA Engine. The AcQA Engine uses the ANTLR [32] tool to process the user specification written in AcQA DSL. This engine is responsible for recognizing specifications written in AcQA and preparing all the required configuration and code generation needed to run the Q&A System. Our work generates Python code to handle the tasks needed both by the compiler and the generated Q&A System.

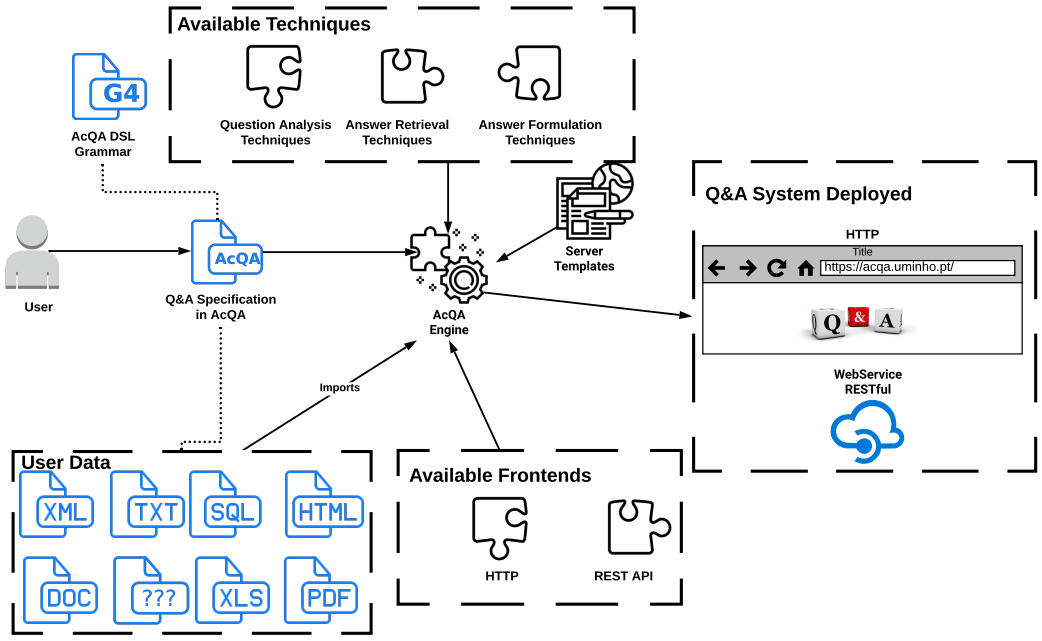


Figure 1 AcQA architecture.

Figure 1 depicts the steps needed to generate the Q&A System and describe how they are connected. First, the user writes a specification of the desired Q&A System in AcQA. It has to define at least three essential elements: the server where the system is going to be deployed, the input data to generate the knowledge base, and which user interface to use. These items and options are explained and discussed in Section 4.1. After the specification written by the user is validated, the AcQA Engine starts to generate the code for the Q&A System. First, the Engine generates the techniques needed to process the inputs from the user of the Q&A System. Secondly, the code for the front-end (user interface) is generated. The third step is to deploy the generated Q&A System into the server. When the server is ready, the data required to make the KB is supplied to the Q&A System to produce the initial KB. After these steps, the Q&A System is ready to process questions and provide answers to the final users. The KB is build by the natural language processing algorithms defined in the AcQA specification. More details on the techniques used to generate the KB are provided in the paper [3].

It is also possible for a user with proper knowledge of GPL and natural language processing techniques to implement new algorithms extending the existing techniques through an interface available in AcQA. The user can choose which parser is used for the system access and understand the data. The data is then processed by the AcQA Engine to generate the knowledge base used by the Q&A System. This module resorts to general-purpose parsers available to import XML, TXT, SQL, or customized parsers to deal with proprietary data formats.

The AcQA Engine processes all specified options to create a Q&A System accordingly to available front-ends (HTTP and Representational State Transfer (RESTful) web service). With the help of Server Templates, the AcQA Engine connects to a server and installs and configures all the requirements needed to deploy the Q&A System.

```
1        acqaFile: lines+ EOF;
2        lines:(comment | decl | TERMINATOR | EOF) ;
3        decl: inputfile | techniques | ui | server | nodeploy |
             cleankb;
4        inputfile: INPUT '(' PATH (',' input_options)* ')';
5        techniques: TECHNIQUES '(' TECHNIQUES_TYPE (','
             techniques_options)* ')';
6        ui: UI '(' UI_TYPE (',' ui_options)* ')';
7        server: SERVER '(' HOST (',' server_options)*')';
8        techniques_options: params;
9        server_options: (USER | PASSWORD | KEY  )'=' value;
10       ui_options: params;
11       input_options: INPUT_PARSER | params;
12       params:key'='value;
13       key: IDENTIFIER;
14       value:NUMERIC_LITERAL | STRING_LITERAL | INT | PATH;
15       comment:COMMENT | SINGLE_LINE_COMMENT | MULTILINE_COMMENT;
16       PATH: STRING_LITERAL;
17       STRING_LITERAL: '\'' ( ~'\'' | '\'\'' )* '\'';
18       SINGLE_LINE_COMMENT: '--' ~[\n]*;
19       MULTILINE_COMMENT: '/*' .*? ( '*/' | EOF );
```

Figure 2 AcQA grammar fragment.

To conclude, AcQA language is an external DSL, containing a custom syntax to make the specification and parameterization of a Q&A System more friendly for the user. There is also a default value for parameters, thus allowing the user to specify only a few values and build the Q&A System automatically. In Section 4.1, we present the main sections of AcQA grammar, and in Section 4.2, we present an example of a Q&A System written in AcQA syntax; the idea is to give more details on the AcQA language and illustrate its use.

4.1 DSL Design

The AcQA language already has several off-the-shelf elements to allow the construction of a Q&A System. This Section presents the elements available to be used by the Q&A System creator.

Figure 2 shows the main elements of AcQA grammar. The listing in that figure shows the declaration of the main definitions needed to set up the initial working Q&A System. Each line of a specification in AcQA can have a comment or a declaration (lines 1 and 2). AcQA DSL has six declaration blocks (line 3): Input File, Techniques, UI, Server, NoDeploy, and CleanKB. These blocks specify the behavior of the generated Q&A System.

The input file (line 4 in Figure 2) specifies the data that has to be imported to create the knowledge base of the Q&A System. Line 4 shows that AcQA specification needs the user to set the path where the input file is located. The input block also allows the user to set optional parameters (line 11) to change the parser's behavior, such as parser type, parser options. There are several parser types for parsing the user data needed to build the knowledge base. In this first release of AcQA, it is possible to parse the following file formats: eXtensible Markup Language (XML), Raw Text (in any encoding, as long as it works with Python), SQL, HTML, DOC, XLS or PDF. Other file formats can be processed through the extension of an interface provided by the AcQA language, and it is the developer's responsibility to program this extension. The optional parameters are in the form of *key* => *value* (as defined in lines 12-14) to change the behavior of the parser.

The Techniques block (line 6 in Figure 2) defines which techniques are used in the question analysis, answer retrieval, and answer formulation processes. If this block is not specified, the default behavior is to use techniques associated with the Triplets approach. These Triplets techniques are initially based on works described in [3] and [35], where a closed-domain Q&A System was developed. These techniques were used as the initial approaches to accelerate the development of AcQA DSL and use the know-how from our language processing group (gEPL). The not obligatory options are defined as *key => value*.

The UI block is responsible for specifying which type of UI the system deploys (line 6 in Figure 2). The initially available front-ends to provide access to the Q&A System are twofold: HTTP and RESTful WebService. The HTTP front-end is a graphical interface available through the HTTP protocol, having a responsive interface and can be accessed through computers, tablets, or cell phones. Using the RESTful front-end allows the creators of the Q&A System who already has some developed platform to provide access to the user who wants to ask questions, by integrating their platform with the generated Q&A System. Figure 5 shows the HTTP front-end visualization in a computer and on a smartphone. The block of AcQA that configures and deploys the system to a given location is the Server (line 7). The user needs to specify the server's hostname, the user name, and password, or the RSA key to access the server.

The last two declaration types are reserved words and change the way that the system is deployed, as shown in Figure 2 (end of line 3). If the user does not specify the keywords NoDeploy or CleanKB, the AcQA Engine deploys all the generated code and sends the input data to be processed by the Q&A System. All previous configurations are lost if there is already a running instance of the Q&A System in the server. The keyword NoDeploy does not reinstall the Q&A System and maintain all data already present on the server. When the developer wants to empty the KB, they can use the keyword CleanKB.

All the parameters written in AcQA are syntactically and semantically validated. For example, the parameters path and hostname are syntactically verified in the grammar of AcQA. AcQA Engine is responsible for enforcing semantic correctness.

4.2 Specifying a Q&A System for Board&Card Games in AcQA DSL

This Section illustrates AcQA DSL syntax by specifying a board & card games Q&A System. The language syntax of AcQA was defined to be simple, allowing the user to create a specification of a Q&A System without the need to have prior knowledge on GPL's. The language syntax also allows the user to parameterize the techniques used to build the knowledge base, process answers, and other parameters.

The Q&A System specified in AcQA is intended to answer questions concerning board & card games. This subsection gives an overview of the domain of the Q&A System. It also discusses the data used in the running example.

According to [19], board games are games with a fixed set of rules that limit the number of pieces on a board, the number of positions for these pieces, and the number of possible moves. There are several discussion groups on the Internet about these types of games; they allow users to ask questions about the rules, strategies, or even questions about the games. An example of one question about the game *Exploding Kittens* is: *How many persons can play the original game?*. In this running example, we use data available from StackExchange (SE)[2] in Archive.org[3].

[2] www.stackexchange.com
[3] `https://archive.org/details/stackexchange`

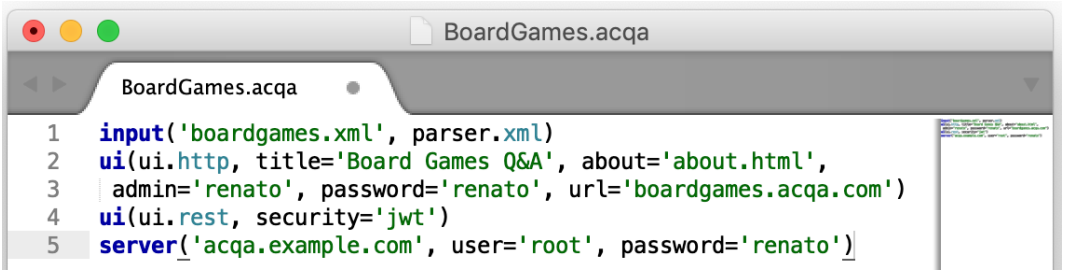


Figure 3 Board and Card Games Q&A System specification in AcQA.

StackExchange (SE) is an Online Social Question and Answering site which allows users to post questions and answers, in this case, handwritten into the system by other community members. Card & Board Games is one of the 166 Stack Exchange Community. We decided to analyze the Card & Board Games among another Community Question Answer site (CQAS) because of the public availability of the data, as well as being regularly updated.

The data used in this work were made available on 2019-03-04 and had approximately 40,7 MB of size and 31395 Posts (questions or answers). The data from SE was preprocessed to extract Questions that have answers from the data dump file.

Figure 3 presents a code fragment of a specification written in AcQA DSL to configure a Q&A System for Board Games. Line 1 in Figure 3 specifies the file name (with a valid path) and which parser is used to process and load the user data into the knowledge base of the Q&A System. It is possible to set optional variables to determine a specific behavior of the parser. This input file is processed by the AcQA Engine and is sent to the deployed Q&A System through the RESTful web service (specified in line 4). After receiving this file, the Q&A System start a task to process the records and generate the knowledge base. In this example, the techniques block was omitted, so the generated Q&A System uses the triplets technique by default.

The UI's are specified in lines 2–4. It is obligatory to specify at least one parameter: the UI type. In lines 2-3, an HTTP UI is defined with some parameters of the UI: the title of Q&A System, the HTML used in Section About, the admin credentials to access the admin page, and the URL used to configure the services in the Server, respectively. In line 4, a setup of a RESTful web service is shown. The parameter security defines which type of security is used by the RESTful web service. In this example, the JSON Web Token[5] is used to provide authentication and authorization in the web service.

The code at line 5 (Figure 3) configures in which Server the Q&A System is deployed, and the first parameter is the Server hostname. The last two parameters are the login information that is used to connect to the server through an SSH (Secure Shell) protocol [41]. The password parameter can be interchanged by the RSA (Rivest-Shamir-Adleman) cryptographic key for added security [37]. There is also an optional parameter to specify which type of the Server (Debian, Fedora, among others). Currently, the default value for the server is Debian-like[4].

The AcQA DSL allows the user to change the behavior of the whole Q&A System. For example, the user can change the language of the system to Portuguese instead of English using the parameter *language="Portuguese"* inside the UI block. The changes are applied in tokenizer, POS (Part-of-Speech) tagger, lemmatizer, and Wordnet language.

[4] `https://debian.org`

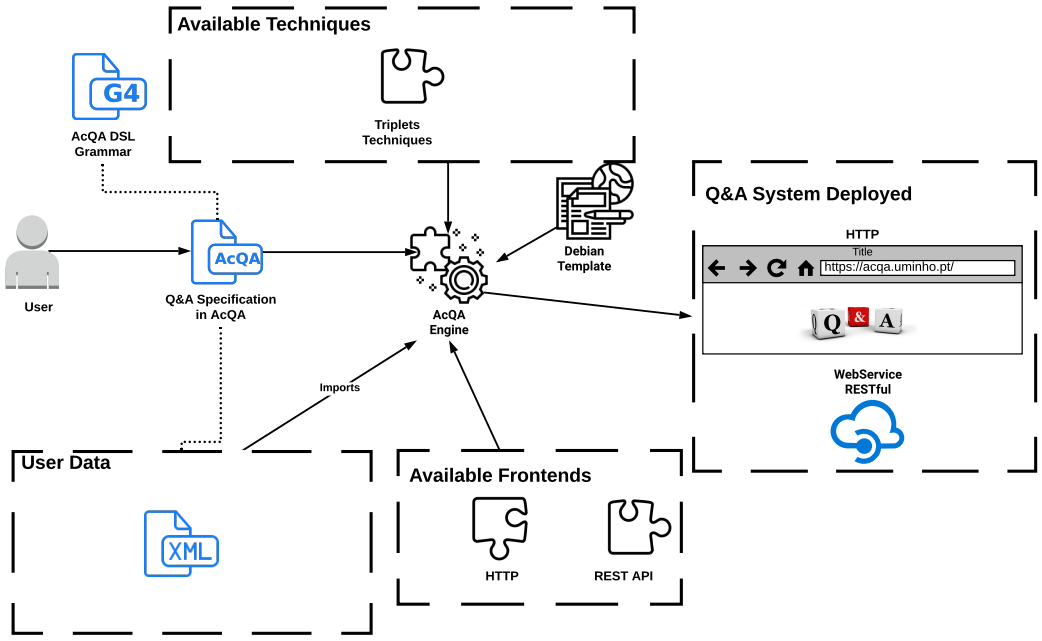


Figure 4 Steps to create a Q&A System.

The steps needed to generate the fully operational Q&A System are depicted in Figure 4. The AcQA grammar is processed by the compiler from AcQA to generate the AcQA Engine. The specification is written by the user and recognized by the AcQA compiler. The data from the user is imported through the Input Parser set in the AcQA specification. The AcQA Engine then generates a Q&A System specified by the user. The Q&A System is deployed into the Server when AcQA Engine executes the Deploy Engine. After that, the Deploy Engine process the User data to create the knowledge base that is used by the fully operational Q&A System.

When a user of the Q&A System accesses the provided URL in the AcQA specification (in our example: boardgames.acqa.com), an HTTP Web UI is displayed, as shown in Figure 5. If more than one answer is generated, the answers are presented according to the ranking generated by the trust value.

5 Assessment

In order to assess the use of AcQA language and test the performance and usability of the generator implemented, an experiment was designed and conducted. This experiment aims at recognizing if it is feasible to use a language like AcQA to create in short time and with a minimum effort Q&A Systems.

The participants in the experiment received a description of a possible scenario within which they were supposed to create, resorting to AcQA, a Q&A System to answer questions about a specific domain.

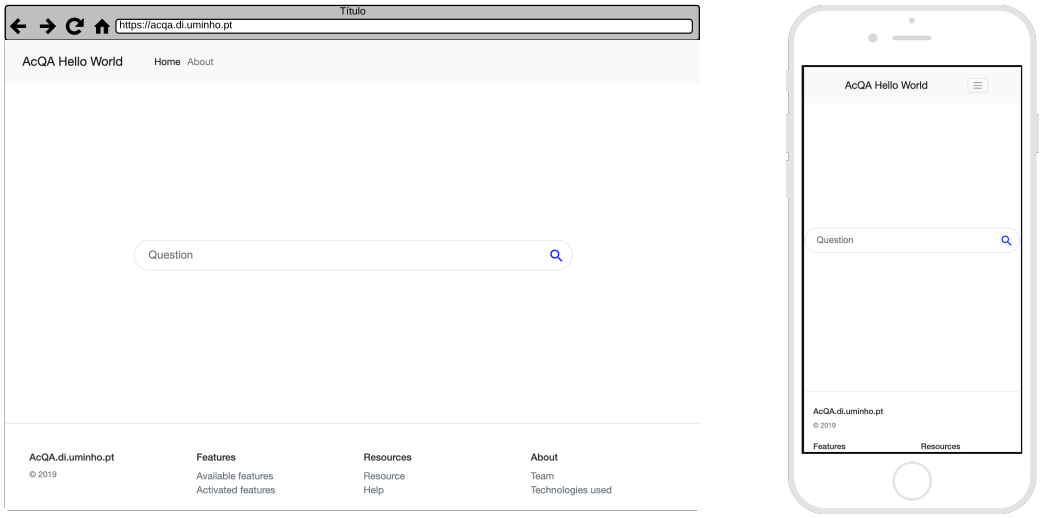


Figure 5 Screenshot of the Board Games Q&A System generated by AcQA.

5.1 Participants

This experiment was applied to people with distinct education, reaching undergraduate, M.S., or Ph.D. students, masters, and doctors. All the participants (actually 17) are from the computer science area and have prior programming experience, ranging from beginners to experts. There were seventeen participants that successfully answered the survey in this experiment.

5.2 Hypotheses definition

The experiment was planned to understand if the following research questions (RQ) are true:

- **RQ1**. Does the AcQA usage help to understand Q&A Systems design?
- **RQ2**. Does the AcQA usage affect the time required to deploy a Q&A System?
- **RQ3**. The tools provided with AcQA can successfully assist the user in the development of the Q&A System?
- **RQ4**. Can AcQA effectively help the user in the deployment of the Q&A System?

5.3 Experiment Design

AcQA language was introduced to the participants through a tutorial on the development and deployment of a complete Q&A System. This tutorial provides an example of a specification written in AcQA to produce a Q&A System to work in the *board & card games* domain. It explains the input file needed to create the knowledge base of the Q&A System. As input file, the participant can choose among seven XML files, extracted from Stack Exchange data dump (as well as the example presented in Section 4.2): cooking, DIY, fitness, hardware, lifehacks, mechanics, parenting. Credentials to get into a Windows Remote Desktop Connection were provided to each participant. This remote connection gives access to a fully functional Integrated Development Environment (IDE) based on Sublime Text Editor, configured to provide support and syntax highlighting for development under AcQA eco-system.

Credentials to a clean install of a Linux server (Ubuntu Server) are also provided to each participant; under that server, Q&A System can be deployed and tested.

After writing his AcQA specification and after building and testing the Q&A System generated by AcQA Engine, the participant was requested to answer a survey organized in four parts: section one contains a questionnaire that gathers information about the participants' prior experience and academic background; section two collects information about the participant's programming experience; section three asks the subject about his experience in the design and development of Q&A Systems; and finally, Section four enquires the participant about AcQA usage.

The experiment here described is accessible at `https://acqa.di.uminho.pt/experiment/`. It presents the tasks needed to evaluate AcQA language and is available to anyone that wants to participate and send us feedback about the experience, thus helping the development of the language.

5.4 Experiment results

As said above, a first instance (others are under preparation) of the described experiments was conducted recently involving seventeen experienced computer science participants, one undergraduate student, four Ph.D. students, three masters students, six masters, and three Ph.D. According to the answers, the majority of participants self-declared as beginners concerning the development of Q&A Systems.

The answers provided by the participants in this experiment were subjected to a reliability test using the Cronbach alpha scale. As the values for the Cronbach alpha scale computed were higher than the threshold of 0.66, the reliability of the measuring instrument can be confirmed.

Some participants stated that the support tools provided (syntax highlighting and build help on a code editor) made it easier to program and understand the AcQA code.

Table 1 shows the average, median, and standard deviation of the respondent answers, quantified in a 1–5 scale.

The research question 1 (Does the AcQA usage help to understand Q&A Systems design) got an average rate of 4, mostly because the respondents did not have prior experience (52.94% of respondents) or are beginners (35.29%) developing Q&A Systems. Only 5.88% of participants stated that were regular Q&A System developers, and 5.88% self-declared as experts.

Analysing the experiment outcomes and the answers collected from the seventeen questionnaires, it is fair to conclude that the four research questions were positively confirmed: (1) the exercise of writing a specification in AcQA DSL contributes for a clear understanding about the design of a Q&A System; (2) using AcQA specification language and engine the time required to deploy a Q&A System is reduced; (3) the editing tools provided with AcQA actually aid the user during the development of a Q&A System; (4) resorting to AcQA system, the deployment of a Q&A System becomes easier and faster.

These statements need further validation, but we are recruiting more people to participate in the next experiments. Anyway, at present, it is obvious that the results so far attained are promising.

[5] A five-grade scale, starting from strongly disagree (1) to strongly agree (5) was used in the questionnaires

Table 1 Statistics about the answers to the research questions (N=17).

	Average[5]	Median	St. Dev.
RQ1	4.00	4.00	0.50
RQ2	3.66	3.66	0.50
RQ3	3.91	4.00	0.41
RQ4	4.23	4.50	0.49

6 Conclusion

In this paper, we presented the domain-specific language AcQA, which allows for the specification of a Q&A System. The specification in AcQA does not require that the developer has a deep knowledge of general programming languages; instead, the developer can focus on the choice of most convenient techniques to include in his new Q&A System.

To show the viability of the proposed approach and expressiveness of AcQA language, a Q&A System to answer questions about *Board Games* was developed and described in the paper. The example used data extracted from StackExchange to create the knowledge repository; its implementation showed that it is possible to generate a Q&A System without the need for complicated and costly GPL development.

From the analysis of the experiment outcomes, we can conclude that the majority of the respondents agree with all the research questions. Moreover, we observed that every participant was able to successfully create and use a Q&A System during the experimental session. We created a website at `https://acqa.di.uminho.pt/` to describe the project and provide documentation and tools that support the AcQA language.

As previously said, in the near future we will conduct more experiments with AcQA eco-system, involving final users from different areas and with different backgrounds (non-programmers, domain specialists, etc.). These experiments will be tuned to test the usability and usefulness of the AcQA DSL.

As future work, we want to extend AcQA language to integrate more techniques [6, 40, 20, 27] for query interpretation and answer formulation, to improve the user interface, and to include server templates. Another direction for future research regards the improvement of the techniques' customization, allowing the developer to make more significant changes in the algorithms and parameters while he writes an AcQA specification.

References

1 Sorin Adam and Ulrik Pagh Schultz. Towards tool support for spreadsheet-based domain-specific languages. In *ACM SIGPLAN Notices*, volume 51, pages 95–98. ACM, 2015.

2 Ahlam Ansari, Moonish Maknojia, and Altamash Shaikh. Intelligent question answering system based on Artificial Neural Network. In *2016 IEEE International Conference on Engineering and Technology (ICETECH)*, pages 758–763. IEEE, March 2016. doi:10.1109/ICETECH.2016.7569350.

3 Renato Azevedo, Pedro Rangel Henriques, and Maria João Varanda Pereira. Extending PythonQA with Knowledge from StackOverflow. In Álvaro Rocha, Hojjat Adeli, Luís Paulo Reis, and Sandra Costanzo, editors, *Trends and Advances in Information Systems and Technologies, WorldCist2018*, volume 745 of *Advances in Intelligent Systems and Computing*, pages 568–575. Springer International Publishing, 1 edition, 2018. doi:10.1007/978-3-319-77703-0_56.

4 Mithun Balakrishna, Steven Werner, Marta Tatu, Tatiana Erekhinskaya, and Dan Moldovan. K-Extractor: Automatic Knowledge Extraction for Hybrid Question Answering. In *Proceedings – 2016 IEEE 10th International Conference on Semantic Computing, ICSC 2016*, 2016. doi:10.1109/ICSC.2016.30.

5 Victoria Beltran. Characterization of web single sign-on protocols. *IEEE Communications Magazine*, 54(7):24–30, 2016.

6 Asma Ben Abacha and Pierre Zweigenbaum. MEANS: A medical question-answering system combining NLP techniques and semantic Web technologies. *Information Processing and Management*, 51(5):570–594, 2015. doi:10.1016/j.ipm.2015.04.006.

7 Ghada Besbes, Hajer Baazaoui-Zghal, and Henda Ben Ghezela. An ontology-driven visual question-answering framework. *Proceedings of the International Conference on Information Visualisation*, 2015-Septe:127–132, 2015. doi:10.1109/iV.2015.32.

8 Steven Bird, Ewan Klein, and Edward Loper. *Natural Language Processing with Python*. O'Reilly Media, Inc., 1st edition, 2009.

9 Daniel G Bobrow. A question-answering system for high school algebra word problems. In *Proceedings of the October 27-29, 1964, fall joint computer conference, part I*, pages 591–614. ACM, 1964.

10 Yong Gang Cao, Feifan Liu, Pippa Simpson, Lamont Antieau, Andrew Bennett, James J. Cimino, John Ely, and Hong Yu. AskHERMES: An online question answering system for complex clinical questions. *Journal of Biomedical Informatics*, 44(2):277–288, 2011. doi:10.1016/j.jbi.2011.01.004.

11 Alexander Clark, Chris Fox, and Shalom Lappin. *The Handbook of Computational Linguistics and Natural Language Processing*. Wiley-Blackwell, 2010.

12 Pierre Cointe. Towards generative programming. In *Unconventional Programming Paradigms*, pages 315—-325. Springer, 2005.

13 Krzysztof Czarnecki. Overview of generative software development. In *Unconventional Programming Paradigms*, pages 326–341. Springer, 2005.

14 Hao Fang, Saurabh Gupta, Forrest Iandola, Rupesh K. Srivastava, Li Deng, Piotr Dollár, Jianfeng Gao, Xiaodong He, Margaret Mitchell, John C. Platt, C. Lawrence Zitnick, and Geoffrey Zweig. From captions to visual concepts and back. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 07-12-June:1473–1482, 2015. doi:10.1109/CVPR.2015.7298754.

15 D Ferrucci. Build watson: An overview of DeepQA for the Jeopardy! Challenge. In *2010 19th International Conference on Parallel Architectures and Compilation Techniques (PACT)*, page 1, 2010.

16 Lance Fortnow and Steve Homer. A short history of computational complexity. Technical report, Boston University Computer Science Department, 2003.

17 Martin Fowler. *Domain-specific languages*. Pearson Education, 2010.

18 Debasish Ghosh. *DSLs in action*. Manning Publications Co., 2010.

19 Fernand Gobet, Jean Retschitzki, and Alex de Voogt. *Moves in mind: The psychology of board games*. Psychology Press, 2004.

20 David C Gondek, Adam Lally, Aditya Kalyanpur, J William Murdock, Pablo Ariel Duboué, Lei Zhang, Yue Pan, Zhao Ming Qiu, and Chris Welty. A framework for merging and ranking of answers in deepqa. *IBM Journal of Research and Development*, 56(3.4):14–1, 2012.

21 Md Moinul Hoque and Paulo Quaresma. A Content-Aware Hybrid Architecture for Answering Questions from Open-domain Texts. In *19th International Conference on Computer and Information Technology*, 2016.

22 Xiangzhou Huang, Baogang Wei, and Yin Zhang. Automatic Question-Answering Based on Wikipedia Data Extraction. In *10th International Conference on Intelligent Systems and Knowledge Engineering, {ISKE} 2015, Taipei, Taiwan, November 24-27, 2015*, pages 314–317, 2015. doi:10.1109/ISKE.2015.78.

23 Inc., Wolfram Research. Wolfram Alpha, 2018.

24 Aditya Jain, Gandhar Kulkarni, and Vraj Shah. Natural language processing. *International Journal of Computer Sciences and Engineering*, 2018.

25 Michael Kaisser and Tilman Becker. Question Answering by Searching Large Corpora With Linguistic Methods. In *TREC*, 2004.

26 S. Kalaivani and K. Duraiswamy. Comparison of question answering systems based on ontology and semantic web in different environment. *Journal of Computer Science*, 8(8):1407–1413, 2012. doi:10.3844/jcssp.2012.1407.1413.

27 Jinhyuk Lee, Wonjin Yoon, Sungdong Kim, Donghyeon Kim, Sunkyu Kim, Chan Ho So, and Jaewoo Kang. Biobert: pre-trained biomedical language representation model for biomedical text mining. *arXiv preprint*, 2019. arXiv:1901.08746.

28 Sweta P. Lende and M. M. Raghuwanshi. Question answering system on education acts using NLP techniques. In *IEEE WCTFTR 2016 - Proceedings of 2016 World Conference on Futuristic Trends in Research and Innovation for Social Welfare*, 2016. doi:10.1109/STARTUP.2016.7583963.

29 Marjan Mernik, Jan Heering, and Anthony M Sloane. When and how to develop domain-specific languages. *ACM computing surveys (CSUR)*, 37(4):316–344, 2005.

30 V. A. Mochalova, Kuznetsov V. A., Mochalov V., and A. Ontological-semantic text analysis and the question answering system using data from ontology. *ICACT Transactions on Advanced Communications Technology (TACT) Vol.*, 4(4):651–658, 2015.

31 FJ Och. Minimum error rate training in statistical machine translation. In *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics - Volume 1*, volume 1, pages 160–167, 2003. doi:10.3115/1075096.1075117.

32 Terence Parr. *The Definitive ANTLR 4 Reference*. Pragmatic Bookshelf, 2nd edition, 2013.

33 Warren J. Plath. Request: A natural language question-answering system. *IBM Journal of Research and Development*, 20(4):326–335, 1976.

34 P Selvi Rajendran and Rufina Sharon. Dynamic question answering system based on ontology. In *2017 International Conference on Soft Computing and its Engineering Applications (icSoftComp)*, pages 1–6. IEEE, December 2017. doi:10.1109/ICSOFTCOMP.2017.8280094.

35 Marcos Ramos, Maria João Varanda Pereira, and Pedro Rangel Henriques. A {QA} System for learning Python. In *Communication Papers of the 2017 Federated Conference on Computer Science and Information Systems, FedCSIS 2017, Prague, Czech Republic, September 3-6, 2017.*, pages 157–164, 2017. doi:10.15439/2017F157.

36 Unmesh Sasikumar and L Sindhu. A Survey of Natural Language Question Answering System. *International Journal of Computer Applications*, 108(15), 2014.

37 William Stallings. *Cryptography and network security: principles and practice*. Pearson Upper Saddle River, 2017.

38 Maria Vargas-Vera and Miltiadis D Lytras. Aqua: A closed-domain question answering system. *Information Systems Management*, 27(3):217–225, 2010.

39 David L Waltz. An english language question answering system for a large relational database. *Communications of the ACM*, 21(7):526–539, 1978.

40 Dirk Weissenborn, Georg Wiese, and Laura Seiffe. FastQA: A simple and efficient neural architecture for question answering. *arXiv preprint*, 2017. arXiv:1703.04816.

41 Tatu Ylonen. Ssh–secure login connections over the internet. In *Proceedings of the 6th USENIX Security Symposium*, volume 37, 1996.

Exploring Different Methods for Solving Analogies with Portuguese Word Embeddings

Tiago Sousa
ISEC, Polytechnic Institute of Coimbra, Portugal
a21220135@isec.pt

Hugo Gonçalo Oliveira
CISUC, Department of Informatics Engineering, University of Coimbra, Portugal
hroliv@dei.uc.pt

Ana Alves
CISUC, University of Coimbra, Portugal
ISEC, Polytechnic Institute of Coimbra, Portugal
ana@dei.uc.pt

Abstract

A common way of assessing static word embeddings is to use them for solving analogies of the kind "*what is to king as man is to woman?*". For this purpose, the vector offset method ($king - man + woman = queen$), also known as 3CosAdd, has been effectively used for solving analogies and assessing different models of word embeddings in different languages. However, some researchers pointed out that this method is not the most effective for this purpose. Following this, we tested alternative analogy solving methods (3CosMul, 3CosAvg, LRCos) in Portuguese word embeddings and confirmed the previous statement. Specifically, those methods are used to answer the Portuguese version of the Google Analogy Test, dubbed LX-4WAnalogies, which covers syntactic and semantic analogies of different kinds. We discuss the accuracy of different methods applied to different models of embeddings and take some conclusions. Indeed, all methods outperform 3CosAdd, and the best performance is consistently achieved with LRCos, in GloVe.

2012 ACM Subject Classification Computing methodologies → Lexical semantics

Keywords and phrases analogies, word embeddings, semantic relations, syntactic relations, Portuguese

Digital Object Identifier 10.4230/OASIcs.SLATE.2020.9

1 Introduction

Computational representations of the words of a language and their meanings have followed two main approaches: symbolic methods like first-order logic and graphs, instantiated as lexical-semantic knowledge bases (LKBs), such as wordnets [7]; and distributional models, like word embeddings. The former organise words, sometimes also senses, often connected by relations, such as Hypernymy or Part-of, and may include additional lexicographic information (part-of-speech, gloss), while the latter follow the distributional hypothesis [10] and represent words as vectors of numeric features, according to the contexts they are found in large corpora. On distributional models, since 2013, the trend was to use efficient methods that learn word embeddings – dense numeric-vector representations of words, like word2vec [14] or GloVe [16]. Besides their utility for computing word similarity, such models have shown very interesting results for solving analogies of the kind "*what is to b as a* is to a?*" (e.g., what is to Portugal as Paris is to France?). So much that both previous tasks are extensively used for assessing word embeddings in different languages.

Popular analogy test sets cover syntactic and semantic relations of different types. Among them, the Google Analogy Test (GAT) [14], which contains syntactic and semantic analogies, was popularised by Mikolov et al. [14] and used in several experiments for assessing word embeddings. Even though there are other similar tests, to our knowledge, only GAT was translated to Portuguese, and rebaptised as LX-4WAnalogies [17].

The most popular method for solving analogy is the vector offset method, used by Mikolov et al. for assessing word2vec [14, 15], and for assessing Portuguese word embeddings [18, 11] in the LX-4WAnalogies. Also known as 3CosAdd, this method solves analogies with a simple operation on the word vectors of the given words ($king - man + woman = queen$).

In this paper, we also use the LX-4WAnalogies test but, more than comparing the performance of different word embeddings, we aim at testing alternative methods for solving analogies, namely 3CosMul [12], 3CosAvg and LRCos [5], in Portuguese word embeddings. An important difference of the last two methods is that they do not solve the analogy from a single pair of words. Instead, they consider a larger set with pairs of analogously-related words. Besides assessing the quality of word embeddings, the analogy solving task can be useful for many different tasks, from the automatic discovery of new word relations for populating knowledge bases [8], to text transformation [1].

The previous methods were tested in different models of word embeddings available for Portuguese [11], including GloVe [16], word2vec [14], fastText [2], and an alternative model, Numberbatch, part of the ConceptNet open data project [19]. Briefly, results achieved confirm what happens for English: all alternative methods tested outperform 3CosAdd and the best accuracy is consistently achieved with LRCos. Even for solving analogies from a single pair, 3CosMul achieves better accuracies than 3CosAdd. We may as well take some conclusions on the quality of the tested word embeddings for analogy solving, which may be broadly interpreted as how well they capture linguistic regularities. Again, as it happens for English, GloVe is often the model with best overall results. Numberbatch, not used in previous work, suffers from a lower word coverage. Yet, when uncovered pairs of words are ignored, it achieves the best accuracy in the semantic analogies.

The paper starts with a brief overview on the most common tasks for assessing word embeddings, with a focus analogy and available datasets for this purpose. After that, we describe the experimentation setup, covering the models used, the methods applied, the relations in the LX-4WAnalogies test, and the adopted data format, for compatibility with all methods and with the used framework. Before concluding, the results achieved are presented and discussed, with global figures as well as results for each type of analogy.

2 Background Knowledge

Models of static word embeddings are commonly assessed in two different tasks: word similarity and analogy solving. The goal of the former is to assign a suitable value for the semantic similarity between pairs of words (e.g., between 0 a 1). In static word embeddings, this value is often given by the cosine of the vectors that represent each word of the pair (e.g., $sim(a, b) = cos(\vec{a}, \vec{b})$). The higher the cosine, the higher the similarity (e.g., the similarity between *dog* and *cat* should be higher than the similarity between *dog* and *car*).

Analogy solving, on the other hand, aims to check how well linguistic regularities are kept in the embeddings. Its goal is to answer questions of the kind "*what is to b as* a^* *is to a*?" (e.g., what is to Portugal as Paris is to France?, for which the answer would be *Lisbon*). The most common method for this is the vector offset, also known as 3CosAdd ($b^* = b - a + a^*$), previously used for computing both syntactic and semantic analogies with word embeddings for different languages, including English [14, 15] and Portuguese [18, 11].

For English, popular datasets include the Microsoft Research Syntactic Analogies (MSR) [15] and the Google Analogy Test (GAT) [14], both used by Mikolov et al. when assessing word2vec. MSR contains 8,000 questions, covering eight different types of syntactic analogy. GAT covers nine types of syntactic analogy (e.g., adjective to adverb, opposite, comparative, verb tenses), roughly the same as MSR, plus five semantic (e.g., capital-country, currency, male-female) categories, with 20-70 unique example pairs per category, which may be combined in 8,869 semantic and 10,675 syntactic questions (see Section 3.3).

GAT was translated to Portuguese [17], rebaptised as LX-4WAnalogies, made publicly available[1], and originally used for assessing the LX-DSemVectors [18], based on word2vec. It was further used for assessing other word embeddings, such as the NILC embeddings [11], which cover different models (e.g., word2vec, GloVe, fastText) with vectors of different dimensions (50, 100, 300, 600, 1000).

Due to its simplicity, the previous experiments on analogy solving relied on the 3CosAdd method. However, other researchers proposed alternative methods that lead to significant improvements. Such methods include 3CosMul [12], 3CosAvg and LRCos [5]. The main difference of the last two is that they use more than a single pair $a : a^*$ for solving the analogy, and exploit a larger set of analogously-related pairs, which could be those in the same dataset (see Section 3.2). Both 3CosAvg and LRCos were presented along the creation of the Bigger Analogy Test Set (BATS) [8], which includes part of GAT, but is larger, covers more types of analogy, and is balanced among all covered types. Moreover, BATS adopted a different representation format, which suits the exploitation of the full dataset better and enables questions to have more than one possible answer (see Section 3.4).

3 Experimentation Setup

Our experimentation consisted of testing a set of methods for solving the analogies in the LX-4WAnalogies [17] dataset, using different pre-trained word embeddings, available for Portuguese. For this purpose, we used Vecto[2], a framework for testing word embeddings that includes the implementation of different methods and produces well-organised logs of the results. In order to test some of the methods in Vecto, we had to change the data format of the LX-4WAnalogies to a format closer to the BATS dataset. This section describes the tested embeddings, the tested methods, and the adopted data format for the dataset.

3.1 Word Embeddings

We tried to cover static word embeddings learned with different algorithms, for which pre-trained models are available in Portuguese. More precisely, we used the following models:

- GloVe, word2vec (CBOW and SKIP-gram) and fastText (CBOW and SKIP-gram), with 300-sized vector, available from the NILC repository of Portuguese word embeddings [11];
- Vectors of Portuguese words in the Numberbatch embeddings [19], version 17.02.

Even though Numberbatch is available in a similar vector format, also with size 300, it is significantly different from the others, because it was learned from several sources. Such sources include raw text (i.e., an ensemble of Google News word2vec, Common Crawl GloVe, Open Subtitles fastText) combined with the ConceptNet semantic network with retrofitting.

[1] https://github.com/nlx-group/LX-DSemVectors/tree/master/testsets
[2] https://github.com/vecto-ai

Selecting only the Portuguese words in Numberbatch is straightforward because all entries are identified by a URI that contains the language prefix (e.g., `/c/pt/banana` for the word *banana*). This made it possible to store Numberbatch in a 107MB text file, while all the other models are substantially larger, with sizes around 2.5GB. Moreover, we considered using the latest version of Numberbatch, 19.08. Yet, after doing the same process for extracting the Portuguese words, the file is about five times larger and some preliminary experiments showed that the new version contains many multiword expressions, which are not in the analogy tests. Therefore, we decided to test only the smaller old version.

3.2 Methods

Mikolov et al. [15] showed that word2vec vectors retain semantic and syntactic information and proposed the vector offset method for answering analogy questions such as "*what is to Portugal as Paris is to France?*". This method, also known as 3CosAdd, formulates the analogy as *a is to* a^* *as b is to* b^*, where b^* has to be inferred from a, a^* and b. More precisely, b^* will be the word with the most similar vector to the result of $a^* - a + b$ (see equation 1). Having in mind that, in a vector space, the similarity between two vectors is given by their cosine, the most similar vector will maximise its cosine with the resulting vector.

$$b^* = \underset{w \in V}{\operatorname{argmax}} \cos(w, a^* - a + b) \tag{1}$$

3CosMul (see equation 2) emerged as an alternative to the arithmetic operation of 3CosAdd, the sum. Using multiplication, Levy and Goldberg [12] refer that, this way, a better balance between the various aspects of similarity is achieved. This was confirmed when 3CosMul indeed achieved better performance in the MSR and GAT tests.

$$b^* = \underset{w \in V}{\operatorname{argmax}} \frac{\cos(b, w) \times \cos(w, a^*)}{\cos(w, a)} \tag{2}$$

3CosAvg and LRCos, both proposed by Drozd et al. [5], try to make the most out of the full test set, instead of a single pair of related words $(a : a^*)$. 3CosAvg computes the average offset between words in position a and words in position a^*, in a set of word pairs analogously related (see equation 3). The answer, b^*, must maximise the cosine with the vector resulting from summing the average offset to b.

LRCos (see equation 4) considers the probability that a word w is of the same class as other words in position a^* as well as the similarity between w and b, measured with the cosine. A classifier, in this case, logistic regression, is used for computing the likelihood of a word belonging to the class of words a^*. Since all methods were applied with the default parameters of the Vecto implementation, the classifier is trained with all entries of the dataset, except the target one $(b : b^*)$, as positive examples, and the same number of negative pairs, each generated from two arguments in different entries, i.e., a is from an entry and a^* is from another, meaning that they should not be related, at least not in as the positive examples.

$$b^* = \underset{w \in V}{\operatorname{argmax}} \cos(w, b + avg\_offset) \tag{3}$$

$$b^* = \underset{w \in V}{\operatorname{argmax}} P(w \in target\_class) \times cos(w, b) \tag{4}$$

Besides the previous methods, we follow Linzen's [13] suggestion and also test to what extent simply using the most similar words is enough for solving analogies and how much different it makes to computing the previous methods. Though not exactly an analogy-solving method, due to its simplicity, the SimilarToB (see equation 5) can be seen as a baseline for this purpose. This method simply retrieves words similar to b, based on the vector cosine, thus, achieving the best accuracy with it means that more complex analogy solving methods are not doing any good.

$$b^* = \underset{w \in V}{\operatorname{argmax}} \cos(b, w) \tag{5}$$

We should add that, as it happens in other implementations of 3CosAdd for assessing word embeddings, in Vecto, when one of the words a or a^* in a pair are not in the model of embeddings, this pair is discarded and it is not considered for computing the average accuracy. This means that the model coverage will not be considered in this evaluation. Though, impact should be minimal in all but Numberbatch, because all other models were learned from the same corpus.

3.3 Relations

The methods previously described were applied to the selected word embeddings for answering the analogy questions in LX-4WAnalogies. We should note that there are two versions of LX-4WAnalogies, one in Brazilian (`LX-4WAnalogiesBr`) and another in European Portuguese (`LX-4WAnalogies`), with minor differences. We used the latter. As in GAT, the questions in LX-4WAnalogies cover 14 types of relation, including five semantic and nine syntactic. Table 1 shows all relation types with two examples for each, in English (from GAT) and in Portuguese, also including the number of questions in the Portuguese version.

A quick look at the data shows some translation issues, not always easy to deal, due to the more complex morphology of Portuguese. Although we did take care of these issues, they can have a negative impact on the accuracy of analogy solving methods. Thus, we point some of them out, and will consider fixing them in future work. For instance, in Portuguese, some of the comparative and superlative analogies are translated equally (e.g., both worse and worst to *pior*). But, in Portuguese, there are two superlative degrees: the relative uses the same word as the comparative, through differently (e.g., *o pior*); the absolute uses a single word (e.g. *péssimo*). In LX-4WAnalogies, both types seem to be used interchangeably. Another issue occurs with the interpretation of the analogy class. For instance, in Portuguese, the verb plurals become a relation between the infinitive to the third person of the singular in the present tense. Finally, in Portuguese, names of nationalities (e.g., as *albanês*) do not start with a capitalised letter, which could be a problem in a case-sensitive scenario.

3.4 Data Format

The questions of GAT are represented in a single text file where each line contains four words: the three necessary for formulating the question, followed by the correct answer, i.e., a a^* b b^*. The type of analogy is identified by lines starting with `:`, indicating that all the following lines have questions of that type. This format, also adopted by LX-4WAnalogies, is illustrated in Figure 1 with sample lines of both datasets.

However, this format was not adopted in the experiments carried out in the scope of this work, because it did not suit some of the methods, namely 3CosAvg and LRCos. Instead, we adopted a BATS-like format, supported by Vecto. This means that there is a file for each

Table 1 Relations covered by GAT, original examples and Portuguese translations in LX-4WAnalogies.

Semantic	**GAT**	**LX-4WAnalogies**
capital-common-countries	Athens, Greece	Atenas, Grécia
(506 questions)	Baghdad, Iraq	Bagdade, Iraque
capital-world	Abuja, Nigeria	Abuja, Nigéria
(4,524)	Accra, Ghana	Acra, Gana
city-in-state	Chicago, Illinois	Chicago, Ilinóis
(2,467)	Houston, Texas	Houston, Texas
currency	Algeria, dinar	Argélia, dinar
(866)	Angola, kwanza	Angola, kwanza
family	boy, girl	rapaz, rapariga
(462)	brother, sister	irmão, irmã
Syntactic	**GAT**	**LX-4WAnalogies**
gram1-adjective-to-adverb	amazing, amazingly	fantástico, fantasticamente
(930)	apparent, apparently	aparente, aparentemente
gram2-opposite	acceptable, unacceptable	aceitável, inaceitável
(756)	aware, unaware	consciente, inconsciente
gram3-comparative	bad, worse	mau, pior
(30)	big, bigger	grande, maior
gram4-superlative	bad, worst	mau, pior
(600)	big, biggest	grande, maior
gram5-present-participle	code, coding	programar, programando
(1,056)	dance, dancing	dançar, dançando
gram6-nationality-adjective	Albania, Albanian	Albânia, Albanês
(1,599)	Argentina, Argentinean	Argentina, Argentino
gram7-past-tense	dancing, danced	dançando, dançou
(1,560)	decreasing, decreased	diminuindo, diminuiu
gram8-plural	banana, bananas	banana, bananas
(1,332)	bird, birds	pássaro, pássaros
gram9-plural-verbs	decrease, decreases	diminuir, diminuem
(870)	describe, describes	descrever, descrevem

kind of analogy, where each row has a single pair of two related words: one to be used in the formulation of a question (b), and another to be used as the target answer (b^*). Although, in BATS, the latter could include more than a single word (i.e., more than one possible answer), this does not happen in LX-4WAnalogies. With this format, GAT-like questions can be formulated by combining two rows of the same file. This is also how Vecto applies the 3CosAdd and 3CosMul methods. Figure 2 illustrates how the LX-4WAnalogies lines in Figure 1 become in the adopted format, where a box representing a single file with file name on top.

This conversion resulted in some differences in the new LX-4WAnalogies. First, this dataset contains a small amount of duplicate rows, some of them originating from the English to Portuguese translation. For instance, GAT has entries such as:

- `father mother grandfather grandmother`
- `father mother grandpa grandma`
- `father mother dad mom`

With corresponding lines in LX-4WAnalogies:

- `pai mãe avô avó`
- `pai mãe avô avó`
- `pai mãe pai mãe`

```
:  capital-common-countries
Athens Greece Baghdad Iraq
Athens Greece Bangkok Thailand
Athens Greece Beijing China
...
Baghdad Iraq Bangkok Thailand
Baghdad Iraq Beijing China
Baghdad Iraq Berlin Germany
...
:  gram4-superlative
bad worst big biggest
bad worst bright brightest
bad worst cold coldest
...
:  gram8-plural
banana bananas bird birds
banana bananas bottle bottles
banana bananas building buildings
...
```

```
:  capital-common-countries
Atenas Grécia Bagdade Iraque
Atenas Grécia Banguecoque Tailândia
Atenas Grécia Pequim China
...
Bagdade Iraque Banguecoque Tailândia
Bagdade Iraque Pequim China
Bagdade Iraque Berlim Alemanha
...
:  gram4-superlative
mau pior grande maior
mau pior brilhante brilhantíssimo
mau pior escuro escuríssimo
...
:  gram8-plural
banana bananas pássaro pássaros
banana bananas garrafa garrafas
banana bananas edifício edifícios
...
```

Figure 1 Sample lines of format of GAT and corresponding lines in LX-4WAnalogies.

```
capital-common-countries.txt
Atenas Grécia
Bagdade Iraque
Banguecoque Tailândia
Pequim China
Berlim Alemanha
...
```

```
gram3-comparative.txt
mau pior
grande maior
brilhante brilhantíssimo
escuro escuríssimo
...
```

```
gram8-plural.txt
banana bananas
pássaro pássaros
garrafa garrafas
edifício edifícios
...
```

Figure 2 Sample lines of LX-4WAnalogies in a Vecto-compatible data format.

In the adopted format, this would also result in duplicate pairs, and thus duplicate lines, which were removed. After this, the number of questions that can be formulated for 3CosAdd and 3CosMul decreases for three types (see Table 2).

Table 2 Analogy types with less formulated questions in the conversion of LX-4WAnalogies.

Type	family	gram3-comparative	gram7-past-tense
#Questions (original)	462	30	1,560
#Questions (new)	380	20	1,482

Moreover, we noticed that, in some analogy types, LX-4WAnalogies does not contain all possible combinations of two related pairs. Since, with the adopted format, all combinations are tested, the number of analogies of five types increased in the conversion of the test (see Figure 3). The increase is especially high for the capital-world relations.

Table 3 Analogy types with more formulated questions in the conversion of LX-4WAnalogies.

Type	cap-world	city-in-state	currency	gr2-opposite	gr6-nat-adj
#Questions (original)	4,524	2,467	866	756	1,599
#Questions (new)	13,340	4,032	870	812	1,640

Our conversion of LX-4WAnalogies to the adopted data format was baptised as TAP, acronym for "Teste de Analogias em Português" (Test of Portuguese Analogies), and is available online[3], for anyone willing to use it.

[3] https://github.com/NLP-CISUC/PT-LexicalSemantics/tree/master/Analogies

4 Results

With Vecto, the LX-4WAnalogies test was solved with all combinations of selected methods (Section 3.2) and word embeddings (Section 3.1). Table 4 shows the macro and micro average accuracy of each combination, also splitted by the semantic and syntactic analogies. Macro averages consider that the accuracy for each relation is worth the same, no matter the number of questions of their type, and thus gives a better perspective on how balanced each combination is for different relations. On the other hand, for micro-averages, every single question is worth the same, meaning that each relation is weighted according to the number of questions its type.

Table 4 Average accuracies achieved with each method and each model of word embeddings.

Model	Method	Macro-Accuracy			Micro-Accuracy		
		Sem	Synt	Avg	Sem	Synt	Avg
GloVe	SimilarToB	6.61%	10.05%	8.82%	4.35%	7.06%	5.75%
	3CosAdd	26.32%	29.79%	28.55%	17.97%	30.67%	21.95%
	3CosMul	29.04%	33.27%	31.76%	21.75%	33.75%	25.51%
	3CosAvg	34.51%	43.01%	39.98%	27.27%	42.01%	34.87%
	LRCos	**51.87%**	**48.34%**	**49.60%**	**56.13%**	**48.33%**	**52.11%**
word2vec CBOW	SimilarToB	2.00%	1.19%	1.48%	0.79%	1.12%	0.96%
	3CosAdd	8.26%	18.07%	14.57%	2.37%	14.60%	6.20%
	3CosMul	9.72%	21.25%	17.14%	2.73%	17.40%	7.33%
	3CosAvg	17.39%	27.01%	23.58%	10.67%	24.54%	17.82%
	LRCos	13.47%	30.46%	24.39%	8.30%	28.62%	18.77%
word2vec SKIP	SimilarToB	2.00%	2.18%	2.12%	0.79%	2.23%	1.53%
	3CosAdd	15.06%	21.33%	19.09%	7.26%	20.52%	11.42%
	3CosMul	16.37%	25.09%	21.98%	9.69%	24.32%	14.28%
	3CosAvg	23.16%	31.16%	28.30%	16.21%	29.74%	23.18%
	LRCos	30.54%	37.88%	35.26%	27.67%	37.92%	32.95%
fastText CBOW	SimilarToB	2.00%	0.40%	0.97%	0.79%	0.37%	0.57%
	3CosAdd	12.79%	28.05%	22.60%	4.85%	31.68%	13.23%
	3CosMul	15.11%	28.28%	23.57%	6.29%	32.42%	14.45%
	3CosAvg	15.51%	39.00%	30.61%	9.88%	39.41%	25.10%
	LRCos	34.30%	36.95%	36.00%	31.23%	37.17%	34.29%
fastText SKIP	SimilarToB	4.21%	4.31%	4.27%	2.37%	4.83%	3.64%
	3CosAdd	25.31%	35.75%	32.02%	15.35%	37.90%	22.39%
	3CosMul	28.62%	38.01%	34.65%	20.73%	39.62%	26.63%
	3CosAvg	29.52%	42.43%	37.82%	20.95%	43.49%	32.57%
	LRCos	50.00%	44.99%	46.78%	51.38%	46.47%	48.85%
Numberbatch	SimilarToB	14.17%	2.26%	6.51%	15.02%	2.97%	8.81%
	3CosAdd	21.45%	8.97%	13.43%	18.21%	10.22%	15.72%
	3CosMul	23.94%	16.15%	18.93%	21.29%	12.12%	18.43%
	3CosAvg	29.99%	13.09%	19.13%	27.27%	11.90%	19.35%
	LRCos	43.81%	23.14%	30.52%	42.69%	22.30%	32.18%

By comparing the accuracy of different methods, for any model of embeddings, it becomes clear that the best accuracy is always achieved by the methods that exploit more than a single pair of related words. This is especially true for LRCos and suggests that this is the best option for solving this kind of problem, at least when a dataset of analogously-related pairs is available. However, we recall that the figures for 3CosAdd and 3CosMul imply many

more questions, i.e., when using each entry pair as $b : b^*$ when each of the remaining entries is used as $a : a^*$. On the other hand, for 3CosAvg and LRCos, a single question is made for each entry $b : b^*$, with all the remaining entries used at the same type.

Still, when a single pair is available, 3CosMul showed to be a better choice than the popular 3CosAdd, which it consistently outperforms. One the other hand, the worst accuracy is always for the SimilarToB, which was expected. Improving the accuracy of SimilarToB shows that all analogy solving methods are indeed doing more than simply looking at the most similar words, also confirming that linguistic regularities in the embedding space go further than just similarity.

Overall, both the best macro and micro accuracies are achieved by LRCos in the GloVe embeddings (50% and 52%, respectively). Despite the different language of GloVe, this is also the combination that achieved the best results in BATS [5]. Although LRCos seems to be the best option overall, some exceptions arise against this absolutism, namely the syntactic relations with 3CosAvg in fastText CBOW.

LX-4WAnalogies had previously been used for assessing Portuguese word embeddings [11], using all models used here, except Numberbatch, and always with 3CosAdd. However, our results do not match the previous. This happens, first, due to the adoption of the BATS data format, which made that, for 3CosAdd and 3CosMul, the number of formulated questions was not the same for all types of analogy (see Section 3.4). Second, for every pair $a : a^*$ for which the model did not include either a or a^* (i.e., they were unknown to the model), the answer was automatically considered incorrect. Both differences made the test more difficult, but we would also say that it increased fairness in the comparison of the models. Nevertheless, although our results with 3CosAdd are lower than in previous work, the main conclusions are the same for this method. The best results for semantic analogies are achieved with GloVe and for the syntactic analogies with fastText-SKIP. Since fastText also considers character n-grams, it makes sense that it handles morphology well. However, a curious outcome of our results is that this is no longer true when LRCos is used in fastText-SKIP. It is not only outperformed by GloVe, but the macro-accuracy is also higher for the semantic relations than for the syntactic. In fact, this is a consequence that, when comparing 3CosAdd and LRCos, the increase of performance is always higher for the semantic than for the syntactic analogies, suggesting that LRCos is more suitable for semantic relations.

Even though we considered that questions with unknown words were automatically incorrect, we also looked at the results when those questions were simply ignored. As expected, all performances increase slightly but, for Numberbatch, the increase is substantial. Table 5 shows the results computed this way for all methods in three models. Figures suggest that Numberbatch is indeed a very accurate model, especially concerning semantic relations. However, it is much smaller and its performance in the previous experiment was heavily affected by its lower coverage.

For a finer-grained analysis, Tables 6 and 7 show the specific results, respectively for each semantic and syntactic relation. Again, we consider that, when a word in the question is not covered, the question is automatically incorrect. The first impression is that accuracy varies significantly, depending on the relation, meaning that different relations pose different challenges, with different levels of difficulty. For instance, with few exceptions, all combinations struggle in the city-in-state and currency analogies. The language of the embeddings may contribute to both of them, especially for the former, as names of states and cities in USA may not appear too often in Portuguese text. Still, in city-in-state, LRCos achieves the best accuracy in all but one model. On the currency, the only accuracies above 0 are those with LRCos in fastText-SKIP (3%) and Numberbatch, especially with LRCos (27%),

Table 5 Average accuracies when questions with unknown words are ignored.

Model	Method	Macro-Accuracy			Micro-Accuracy		
		Sem	Synt	Avg	Sem	Synt	Avg
GloVe	SimilarToB	6.75%	10.11%	8.91%	4.44%	7.20%	5.86%
	3CosAdd	26.32%	29.79%	28.55%	17.97%	30.67%	21.95%
	3CosMul	29.04%	33.27%	31.76%	21.75%	33.75%	25.51%
	3CosAvg	35.25%	43.41%	40.50%	27.82%	42.80%	35.55%
	LRCos	52.93%	**48.87%**	**50.32%**	57.26%	**49.24%**	53.13%
fastText SKIP	SimilarToB	4.26%	4.36%	4.33%	2.42%	4.92%	3.71%
	3CosAdd	25.31%	35.75%	32.02%	15.35%	37.90%	22.39%
	3CosMul	28.62%	38.01%	34.65%	20.73%	39.62%	26.63%
	3CosAvg	30.14%	42.91%	38.35%	21.37%	44.32%	33.20%
	LRCos	51.06%	45.55%	47.51%	52.42%	47.35%	49.80%
Numberbatch	SimilarToB	21.12%	2.65%	9.25%	25.17%	6.11%	16.31%
	3CosAdd	21.45%	8.97%	13.43%	18.21%	10.22%	15.72%
	3CosMul	23.94%	16.15%	18.93%	21.29%	12.12%	18.43%
	3CosAvg	41.93%	16.13%	25.34%	45.70%	24.62%	35.94%
	LRCos	**63.82%**	33.13%	44.09%	**71.52%**	45.80%	**59.57%**

which might have benefited of the amount of world knowledge included in ConceptNet. The best accuracies are for the capitals and family analogies. This is especially true for the capital-common-countries, where the highest accuracies are achieved with LRCos (e.g., 78% with GloVe or fastText).

On average, accuracies are lower for the syntactic analogies. As expected, the SimilarToB baseline is still the less accurate method. For almost every model and method, the highest accuracies are for the present-participle (e.g., 66% in fastText with 3CosAvg) and for the comparative (e.g., 80% in GloVe with LRCos or 3CosAvg). However, for the latter relation, results are not very representative, as they are only based on 20 questions, for 3CosAdd and 3CosMul, and on 5 questions, for the remaining methods. On the other hand, the good accuracy for the present-participle makes sense, because the questions are quite regular, and going from one form to the other is just a matter of adding the suffix *-ndo*. Therefore, a single example, as in 3CosAdd or 3CosMul, might be enough for solving the analogy. This is probably why the difference between methods is not so pronounced in this relation. Also, higher regularity works well for 3CosAvg, which is the best method in some models.

In opposition to the comparative, the superlative relation stands out has having almost all combinations with accuracies equal 0 or close. This should be mostly due to issue discussed in Section 3.3, related to the different superlative degrees in Portuguese. In the examples in Figure 1, it is clear that both types of superlative are used interchangeably (e.g., *brilhantíssimo* is the absolute superlative of *brilhante*, but *(o) pior* is the relative superlative of *mau*). This is even more problematic because the relative superlative uses the same forms as the comparative, even though they are used differently (e.g., *pior do que* for comparative and *o pior* for superlative). The relation with the second lowest accuracy was the opposite, which was quite surprising, because most opposites are obtained by adding prefixes like *i(n/m)-*, *des-* or *anti-*. It also becomes clear that relying exclusively on the Portuguese part of Numberbatch is not a suitable approach for several relations, mainly because ConceptNet will cover mostly lemmatised words without information on inflection.

Table 6 Accuracy for semantic relations.

Model	Method	Accuracy				
		cap-common	cap-world	city-in-state	currency	family
GloVe	SimilarToB	13.04%	3.45%	1.56%	0.00%	15.00%
	3CosAdd	52.77%	20.25%	6.72%	0.00%	51.84%
	3CosMul	55.73%	25.28%	7.12%	0.23%	56.84%
	3CosAvg	65.22%	26.72%	15.63%	0.00%	**65.00%**
	LRCos	**78.26%**	**66.38%**	**54.69%**	0.00%	60.00%
word2vec CBOW	SimilarToB	0.00%	0.00%	0.00%	0.00%	10.00%
	3CosAdd	10.47%	1.99%	0.67%	0.00%	28.16%
	3CosMul	11.46%	2.32%	0.62%	0.00%	34.21%
	3CosAvg	26.09%	7.76%	3.13%	0.00%	50.00%
	LRCos	30.43%	6.90%	0.00%	0.00%	30.00%
word2vec SKIP	SimilarToB	0.00%	0.00%	0.00%	0.00%	10.00%
	3CosAdd	30.43%	7.37%	3.00%	0.00%	34.47%
	3CosMul	32.61%	10.65%	3.32%	0.00%	35.26%
	3CosAvg	43.48%	12.93%	9.38%	0.00%	50.00%
	LRCos	56.52%	29.31%	21.88%	0.00%	45.00%
fastText CBOW	SimilarToB	0.00%	0.00%	0.00%	0.00%	10.00%
	3CosAdd	19.37%	4.89%	0.74%	0.00%	38.95%
	3CosMul	25.69%	6.57%	0.89%	0.00%	42.37%
	3CosAvg	17.39%	8.62%	1.56%	0.00%	50.00%
	LRCos	56.52%	36.21%	18.75%	0.00%	60.00%
fastText SKIP	SimilarToB	4.35%	1.72%	0.00%	0.00%	15.00%
	3CosAdd	48.22%	17.64%	2.95%	0.11%	57.63%
	3CosMul	55.14%	24.84%	3.60%	0.57%	58.95%
	3CosAvg	56.52%	19.83%	6.25%	0.00%	**65.00%**
	LRCos	**78.26%**	61.21%	42.19%	3.33%	**65.00%**
Numberbatch	SimilarToB	13.04%	21.55%	6.25%	0.00%	30.00%
	3CosAdd	34.19%	22.77%	1.86%	2.64%	45.79%
	3CosMul	41.11%	26.78%	2.03%	4.25%	45.53%
	3CosAvg	43.48%	35.34%	7.81%	3.33%	60.00%
	LRCos	69.57%	56.90%	10.94%	**26.67%**	55.00%

5 Conclusions

We have tested different methods that exploit word embeddings for solving analogies of the kind *What is to b as a^* is to a?*, in Portuguese. Although this problem had been tackled before [11], the previous goal was mainly to compare embeddings of different sizes and learned with different algorithms, always using the same analogy solving method, 3CosAdd. Here, we tested alternative methods for this purpose, always outperforming 3CosAdd, especially those methods that exploit a set and not just a single pair of related words ($a : a^*$), namely 3CosAvg and LRCos [5].

Despite working on a different language, initial conclusions are not much different from those for English. Different types of analogy pose different challenges, with varying accuracies. Still, overall, GloVe embeddings showed to be good at keeping linguistic regularities, with best results achieved by LRCos. In fact, when more than one pair is available, LRCos proved to be the best method for any model. As far as we know, this was the first time when the alternative analogy solving methods were tested for Portuguese, in the LX-4WAnalogies.

Table 7 Accuracy for syntactic relations.

Model	Method	Accuracy								
		adj-adv	opposite	comp	superl	pres-part	nation-adj	past	plural	plural-v
GloVe	SimilarToB	3.23%	3.57%	40.00%	4.00%	12.12%	0.00%	2.56%	21.62%	3.33%
	3CosAdd	4.95%	3.20%	60.00%	0.83%	49.81%	53.48%	26.52%	41.82%	27.47%
	3CosMul	6.02%	3.57%	70.00%	1.00%	52.84%	57.20%	32.39%	44.14%	32.30%
	3CosAvg	16.13%	17.86%	**80.00%**	0.00%	60.61%	68.29%	33.33%	67.57%	43.33%
	LRCos	25.81%	14.29%	**80.00%**	0.00%	60.61%	68.29%	56.41%	**72.97%**	**56.67%**
word2vec CBOW	SimilarToB	0.00%	10.71%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
	3CosAdd	2.47%	6.77%	55.00%	0.67%	37.59%	12.68%	19.64%	9.68%	18.16%
	3CosMul	3.66%	6.53%	65.00%	1.00%	41.29%	15.37%	26.45%	12.24%	19.77%
	3CosAvg	9.68%	14.29%	60.00%	0.00%	54.55%	26.83%	28.21%	16.22%	33.33%
	LRCos	16.13%	17.86%	60.00%	0.00%	48.48%	29.27%	33.33%	32.43%	36.67%
word2vec SKIP	SimilarToB	0.00%	10.71%	0.00%	0.00%	3.03%	0.00%	2.56%	0.00%	3.33%
	3CosAdd	3.66%	7.51%	45.00%	0.67%	43.75%	29.88%	21.26%	14.86%	25.40%
	3CosMul	6.02%	7.02%	55.00%	0.83%	46.40%	38.35%	27.94%	17.04%	27.24%
	3CosAvg	12.90%	14.29%	60.00%	0.00%	57.58%	46.34%	25.64%	27.03%	36.67%
	LRCos	29.03%	17.86%	60.00%	0.00%	54.55%	56.10%	46.15%	40.54%	36.67%
fastText CBOW	SimilarToB	0.00%	3.57%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
	3CosAdd	14.52%	9.26%	30.00%	0.33%	56.72%	36.52%	47.23%	29.73%	28.16%
	3CosMul	11.72%	9.39%	30.00%	0.33%	56.91%	42.93%	47.23%	29.43%	26.55%
	3CosAvg	22.58%	10.71%	60.00%	0.00%	**66.67%**	48.78%	58.97%	43.24%	40.00%
	LRCos	19.35%	17.86%	60.00%	0.00%	51.52%	51.22%	51.28%	51.35%	30.00%
fastText SKIP	SimilarToB	3.23%	3.57%	0.00%	0.00%	9.09%	0.00%	0.00%	16.22%	6.67%
	3CosAdd	9.46%	10.19%	60.00%	0.67%	64.49%	52.99%	43.25%	47.22%	33.45%
	3CosMul	11.72%	10.45%	70.00%	0.83%	63.92%	57.32%	46.96%	47.52%	33.33%
	3CosAvg	19.35%	14.29%	60.00%	0.00%	**66.67%**	60.98%	53.85%	56.76%	50.00%
	LRCos	29.03%	17.86%	60.00%	0.00%	63.64%	63.41%	**61.54%**	59.46%	50.00%
Numberbatch	SimilarToB	3.23%	0.00%	0.00%	0.00%	0.00%	17.07%	0.00%	0.00%	0.00%
	3CosAdd	8.92%	2.38%	20.00%	0.50%	0.19%	45.24%	0.00%	1.20%	2.30%
	3CosMul	14.09%	5.56%	70.00%	2.50%	0.38%	49.63%	0.00%	1.13%	2.07%
	3CosAvg	9.68%	0.00%	40.00%	0.00%	0.00%	56.10%	0.00%	5.41%	6.67%
	LRCos	**41.94%**	**32.14%**	40.00%	**8.00%**	6.06%	**70.73%**	0.00%	2.70%	6.67%

Another difference regarding previous work is that we included a different kind of word embeddings, Numberbatch. When ignoring questions with unknown pairs, performances achieved with this model are high, especially on semantic relations, where it achieved the best accuracy with LRCos. However, it is also a smaller model, with performance highly affected otherwise. In the future, we should test the larger newest version of Numberbatch (19.08). On the other hand, when solving analogies from a single pair, 3CosMul is generally better than 3CosAvg. In this specific case, fastText-SKIP is the best model for syntactic relations.

A limitation of the GAT and LX-4WAnalogies tests is that they are not balanced among the covered relations. This makes it harder to compare the performance for each relation and to rely on micro-average for analysing the performance in the full dataset. This is why we mainly looked at the macro-average of the accuracy. This is an issue that the BATS dataset tries to answer. It does not only cover a broader set of relation types but has exactly 50 instances for each relation type.

One of the additional categories of relation covered by BATS is precisely lexicographic relations, which are extremely useful for testing how suitable a model of embeddings is for augmenting lexical-semantic knowledge bases. Besides assessing how well word embeddings capture linguistic regularities, and thus how suitable they are for exploitation in many different tasks, analogy solving can be useful for supporting or discovering new lexical-semantic relations automatically, for instance, for populating knowledge bases. The latter may consider general language knowledge bases, including wordnets [7], and also domain ontologies, especially if embeddings are learned from a corpus of the same domain.

Since LX-4WAnalogies does not cover lexicographic relations (i.e., those one would find in a dictionary or wordnet), we have recently explored available lexical knowledge bases on the creation of a new dataset for assessing Portuguese word embeddings in the discovery of such relations [9]. This was an alternative to avoid time-consuming manual translation of BATS and language issues that may arise with the process, such as those we have identified in LX-4WAnalogies. Our first impression is that lexicographic relations are significantly more challenging than most of the relations covered by GAT. Nevertheless, manually accepting good answers out of the top candidates should still be less time-consuming than populating or augmenting a knowledge base completely from scratch. In this case, evaluation measures that consider the ranked candidates (e.g., Mean Average Precision) are relevant. Furthermore, we should test if better results are achieved when we combine several of the methods tested here (e.g., in an ensemble), and possibly explore alternative methods proposed more recently [3].

Finally, better results on this task might be achieved with more recent language models, also known as contextual embeddings, like BERT [4], for which pre-trained models are available for Portuguese. Even though words in analogy tests are not lack context, recent work has showed that the first principal component of such contextualized representations in a given layer (apparently, the lower, the better) can outperform static word embeddings in analogy solving [6].

References

1. Benjamin Bay, Paul Bodily, and Dan Ventura. Text transformation via constraints and word embedding. In *Proc. 8th International Conference on Computational Creativity*, ICCC 2017, pages 49–56, 2017.
2. Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5:135–146, 2017.
3. Zied Bouraoui, Shoaib Jameel, and Steven Schockaert. Relation induction in word embeddings revisited. In *Proceedings of the 27th International Conference on Computational Linguistics*, COLING 2018, pages 1627–1637, Santa Fe, New Mexico, USA, August 2018. ACL.
4. Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proc. of Human Language Technologies, Vol 1*, NAACL-HLT 2019, pages 4171–4186. ACL, 2019.
5. Aleksandr Drozd, Anna Gladkova, and Satoshi Matsuoka. Word embeddings, analogies, and machine learning: Beyond king - man + woman = queen. In *Proceedings the 26th International Conference on Computational Linguistics: Technical papers (COLING 2016)*, COLING 2016, pages 3519–3530, 2016.
6. Kawin Ethayarajh. How contextual are contextualized word representations? comparing the geometry of bert, elmo, and gpt-2 embeddings. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 55–65, 2019.
7. Christiane Fellbaum, editor. *WordNet: An Electronic Lexical Database (Language, Speech, and Communication)*. The MIT Press, 1998.
8. Anna Gladkova, Aleksandr Drozd, and Satoshi Matsuoka. Analogy-based detection of morphological and semantic relations with word embeddings: what works and what doesn't. In *Proceedings of the NAACL 2016 Student Research Workshop*, pages 8–15. ACL, 2016.
9. Hugo Gonçalo Oliveira, Tiago Sousa, and Ana Alves. Tales: Test set of portuguese lexical-semantic relations for assessing word embeddings. In *Proceedings of the ECAI 2020 Workshop on Hybrid Intelligence for Natural Language Processing Tasks (HI4NLP)*, page In press, 2020.
10. Zelig Harris. Distributional structure. *Word*, 10(2-3):1456–1162, 1954.

11 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 11th Brazilian Symposium in Information and Human Language Technology (STIL 2017)*, 2017.

12 Omer Levy and Yoav Goldberg. Linguistic regularities in sparse and explicit word representations. In *Proceedings of 18th Conference on Computational Natural Language Learning*, CoNLL 2014, pages 171–180. ACL, 2014.

13 Tal Linzen. Issues in evaluating semantic spaces using word analogies. In *Proceedings of the 1st Workshop on Evaluating Vector-Space Representations for NLP*, pages 13–18, Berlin, Germany, August 2016. ACL.

14 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.

15 Tomas Mikolov, Wen-tau Yih, and Geoffrey Zweig. Linguistic regularities in continuous space word representations. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 746–751, Atlanta, Georgia, June 2013. ACL.

16 Jeffrey Pennington, Richard Socher, and Christopher D. Manning. GloVe: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*, EMNLP 2014, pages 1532–1543. ACL, 2014.

17 Andreia Querido, Rita Carvalho, João Rodrigues, Marcos Garcia, João Silva, Catarina Correia, Nuno Rendeiro, Rita Pereira, Marisa Campos, and António Branco. LX-LR4DistSemEval: a collection of language resources for the evaluation of distributional semantic models of Portuguese. *Revista da Associação Portuguesa de Linguística*, 3:265–283, 2017.

18 João Rodrigues, António Branco, Steven Neale, and João Ricardo Silva. LX-DSemVectors: Distributional semantics models for Portuguese. In *Proceedings of 12th International Conference on the Computational Processing of the Portuguese Language PROPOR*, volume 9727 of *LNCS*, pages 259–270, Tomar, Portugal, 2016. Springer.

19 Robert Speer, Joshua Chin, and Catherine Havasi. Conceptnet 5.5: An open multilingual graph of general knowledge. In *Proceedings of Thirty-First Conference on Artificial Intelligence (AAAI)*, pages 4444–4451, 2017.

Towards a Morphological Analyzer for the Umbundu Language

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

Bernardo Sacanene
Centro de Estudos Humanísticos da Universidade do Minho, Braga, Portugal
besacanene@gmail.com

Álvaro Iriarte
Centro de Estudos Humanísticos da Universidade do Minho, Braga, Portugal
alvaro@ilch.uminho.pt

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

Joaquim Macedo
Algoritmi, Departamento de Informática, Universidade do Minho, Braga, Portugal
macedo@di.uminho.pt

Abstract

In this document we present the first developments on an Umbundu dictionary for a jSpell, a morphological analyzer. Initially some comments are performed regarding the Umbundu language morphology, followed by the discussion on jSpell dictionaries structure and its environment. Last, we describe the Umbundu dictionary bootstrap process and perform some final experiments on its coverage.

2012 ACM Subject Classification Computing methodologies → Language resources

Keywords and phrases Umbundu, Angolan Languages, Morphological Analysis, Spell Checking

Digital Object Identifier 10.4230/OASIcs.SLATE.2020.10

Funding This research was partially funded 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. Bernardo Sacanene acknowledges from the Angolan govenment his PhD grant, through INAGBE (Instituto Nacional de Gestão de Bolsas de Estudos).

1 Introduction

A large part of African populations just as many other populations from other parts of the world, live in a situation of diglossia, in the most restricted sense of the term, presented by Ferguson [4], in which communities have two languages at their disposal: one language A (or prestigious language) and a language B (familiar or domestic language).

In these situations, it is normal to produce, sooner or later, a process of linguistic substitution (by direct imposition or as a result of a valuation that the new speaker makes, in terms of usefulness, of the language), starting from an original monolingual situation and going through a provisional bilingual state (in languages A and B) to finally reach a monolingual state (surviving solely the language A) [2].

But linguistic diversity is always a treasure. Each language has a type of relationship with reality and diversity, a guarantee of the system's durability and stability, since it does not destroy resources (in this case, linguistic, cultural, etc.). In the case of Africa, "*the*

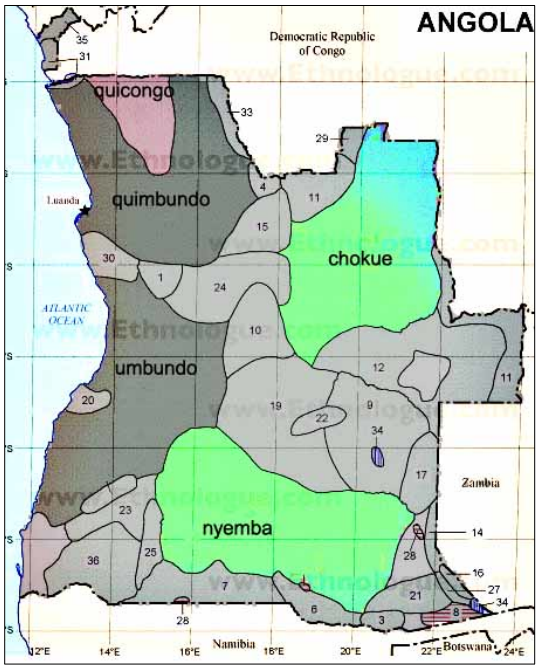


Figure 1 Map of languages in Angola[1].

languages of some peoples which have attained sovereignty are consequently immersed in a process of language substitution as a result of a policy which favours the language of former colonial or imperial powers" [12].

Only linguistic policies can change this process, together with the speaker's appreciation of the language, in terms of necessity or usefulness. That is the reason we must create conditions for people to enjoy their linguistic rights. These conditions include ensuring the use of languages B by different communities and creating conditions for interrelation and dialogue of these less favoured languages with both prestigious languages and formal languages.

Defending the linguistic rights of speakers involves not only linguistic policies that favor the use of community languages in as many contexts as possible, including formal contexts, but also by creating conditions for the interrelation of these languages with others, encouraging the practice of translation between both languages and the connection of these languages with formal languages, with the possibility of being used as an object of study in the Processing of Natural Languages, in corpus linguistics, automatic translation, data mining for the extraction of terms and lexicon for the elaboration of terminological and lexicographic products, etc.

The creation of Natural Language Processing tools for African languages is an urgent and imperative task to defend the wealth that current linguistic diversity signifies.

In this document we explore the development of a first Natural Language Processing (NLP) tool for Umbundu, one of Angola's languages (see Figure 1).

The construction of Umbundu corpora and other Bantu languages corpora (as well as other African languages) will allow access to a set of information (linguistic, cultural, etc.) that can be used for the elaboration of traditional dissemination products (newspapers,

[1] Map obtained from `http://seguindoadiante.blogspot.com/2008/08/torre-de-babel.html` [accessed on 2020-07-03].

school textbooks, dictionaries, grammars, medical records, restaurant menus, tourist guides, etc.). It could also be used for the development of Apps and other resources, which will be available for agents involved in the management and modernization of the cultural, linguistic, tourist, etc. of the African states and the different communities that make them up.

The next section explores the Umbundu language, namely some of the most basic rules of its morphology. Section 3 presents the environment of jSpell, the morphological analyzer used in our developments. Some of the rules included in the Umbundu morphological analyzer are depicted on Section 4. In the last section, we discuss the obtained results.

2 Contextualization of the Umbundu Language

Umbundu is the language of the ovimbundo ethnolinguistic group, belonging to zone R, group 10, R11 [6] and comes from the bantu group. It is spoken in central and Southern of Angola, specifically in Bié, Huambo, Benguela and extends to neighbouring province such as Namibe, northwest Cuando Cubango and northern Huíla.

According to the data from the last census [7] regarding the languages usually spoken at home, in the first place we have Portuguese (71%), and Umbundu as the second language (23%), followed by Kikongo and Kimbundu.

2.1 Orthographic Binormativism

The Umbundu language presents, for a specific linguistic reality, different written forms. As an example, “tch”, “ch”, “c” in “tchina”, “china”, “cina” (thing), “nɡ̃”, “ñ” in “Nɡ̃ala”, “Ñala” (God) or “dj” and “j” in “ondjo” and “onjo” (house). This is a common problem for languages before any attempt of standardization, and related to the fact that the first studies about the Umbundu languages where performed by religion persons that used an alphabet from another language with Latin origins, modifying it in order to simulate the sounds that are nonexistent in European languages [3].

The study of Umbundu by persons tied to religious institutions, together with UNESCO recommendations, lead to different ways to represent graphically this language. Therefore, there is a “catholic version and a protestant one,” [8, p. 479] as well as another, adopted by the Institute of National Languages [3], as an attempt for standardization. This orthographic duplicity or even triplicity is a problem, not just for the literacy of the population, but also for the creation of natural language processing tools for these languages.

2.2 A brief introduction to Umbundu

To build a morphological analyzer of the Umbundu language we are studying three grammatical categories: noun, adjective and verb. First and foremost we analyze how the formation of the number and the gender are done, and also the inflexion of verbs.

2.2.1 Nouns

As other bantu languages, Umbundu presents the following characteristics:

1. Nouns are organized in classes defined by prefixes. The classes are combined to distinguish the opposition of the number (singular/plural). Table 1 shows how number affix is done. According to Fernandes and Ntondo [5], the class changeover is done in two ways: replacement of prefixes (exchange of class prefix with that of the class in which it is inserted) and addition of prefixes (the noun inserted is close to the prefix of that class).

Table 1 Noun prefixes. Based on [5].

Umbundu language			Examples	
Class	Prefix	Class Information	Umbundu	English
1	omu-, u-, o-	human-related	omunu, ukombe	person, guest
2	oma-, ova-, a-		omanu, akombe	people, guests
3	u-	plants, animals, body parts and other	uti / uta	tree / gun
4	ovi-		oviti / ovita	trees / guns
5	e-	human realities, plants, animals	eka / epela, epito, etimba	hand / baldness, door, body
6	a-, ova-		ovaka / apela, apito, atimba	hands / baldness, doors, bodies
7	oci-	several concepts	ocimunu*, ocitangi	thief, problem
8	ovi -		ovimunu, ovitangi	thieves, problems
9	o-, ∅-	common for animals	ombweti, omoko	stick, knife
10	olo-		olombweti, olomoko	sticks, knives
11	olu-	delimitation	olumapo, olukalo, alupandu	model, opportunity, thanks
12	oke-	several concepts and diminutives	okalenda, okamõla, okambeyi	small tumor, little child, sicky
13	atu-		okatumola, otumbeyi	small tumors, little children, sickies
14	u-		ukolo	rope
15	oku-	verb-nominal	okulota, eloto	to dream, dream
Locative classes: Number variation does not occur for nouns [5, 10].				
16	pa-	surface	okulya ikasi po mesa	the food is on the table
			wacipaka p'osi	put down
			watumala k'omangu	sit in the chair
17	ko-	direction, route	eye wanda k'epya	he went to the farm
			weya k'imbo	he came from the village
			kosindikile k'onembele	walk him in to the church
			eye okasi ko samwa	he it out
18	vu-	inside	eye okasi v'onjo	he is at home
			vapiluka v'ocitali	they danced in the backyeard
			wakupukila v'ocikungu	fell in the hole

*Umbundu allows three different orthographic forms for *ocimunu*. While this is the most common, the pronunciation and the attempt to make phonetic part of the word makes *ochimunu* and *otchimunu* acceptable orthographic forms.

Table 2 Umbundu regular paired noun classes. Based on [5].

Umbundu		English	
Singular	Plural	Singular	Plural
class 1: u- ukombe	class 2: a- akombe	guest	guests
class 3: u- upange	class 4: ovi- ovopange	work	works
class 3: u- ulume / ukãyi	class 6: a- alume / akãyi	man / woman	men / women
class 5: e- ekepa / etimba / etapalo / ekandu	class 4: a- akepa / atimba / atapalo / akandu	bone / body / road / sin	bones / bodies / roads / sins
class 5: e- ekomohiso, ewe	class 4: ovi ovikomohiso, ovawe	wonder, rock	wonders, rocks
class 7: oci- ocitungu, ocitangi	class 8: ovi- ovitungu, ovitangi	bundle, problem	bundles, problems
class 9: o-, ∅- ohumbo, omunda	class 10: olo- olohumbo, olomunda	needle, mountain	needless, mountains
class 11: olu- olunyi	class 10: olo- olonyi	fly	flies
class 12: oka okavisungo	class 13: otu okatuvisungo	little song	little songs
class 14: u- utima	class 6 ova- ovitima / ovilwa	heart / whistle	hearts / whistles
class 15: oku- okulama	class 4: ovi- ovilamo		greetings
class 5: e- eteke	class 10: olo- oloneke	day	days

Table 3 Umbundu irregular paired noun classes. Based on [5].

Class	Prefix merge	English
Class 11: olu olumbo	class 6 (a-) + class 11 (olu) alumbo	fence
Class 15: oku okwenye	class 6 (a) + class 15 (oku) akwenye	dry season

The pairing of the class system allows us to know how the plural is done in Umbundu. Although some slant is recognized, there is, as Katamba [9, p. 129] states, "distributional characteristics of noun classes showing a high degree of coherence."

2. There is no specification of the gender, but there are words that gives us hints:
   - Common nouns of human kinship: *ulume* (man) and *ukãyi* (woman); *ukwenje* (boy) and *ufeko e ukano* (girl); *ise* (father) and *ina* mother; *mulume* (brother) and *mukãyi* (sister).
   - Kinship nouns more closely related to membership: *tate*, *so* and *ise* (father); *nyoho* and *ina* (mother).
   - Gender-related nouns for animals:
     onwi (bull), *ongombe* (ox), *onjindi* (cow), *onale* (calf), *ombelipa* (heifer), *ekondombolo* (rooster), *osanji* (hen), *ochitupi* (goat), *oselenge* (castrated goat), *ondume* (animal male) and *omange* (animal females)
3. The gender of the nouns is indicated by the postposition of the words for male or female (*ulume* or *ukãyi*), and the plural being made in the class leaving intact the radical of the word.

- For humans *ulume* (man), *omola ulume* (son), *ukãyi*: (woman), and *omola ukãyi*: (daughter).
- For irrational animals, the indication of gender occurs in two ways:
  a. Applying *ulume* (man) and *ukãyi* (woman) as *ombwa yulume* (dog), *ombwa yukãyi* (female dog), *ongulu yulume* (pig) and *ongulu yukãyi* (sow).
  b. Applying *ondume* or *ochilume* for male and *omange* for female: *onjamba yondume* (male elephant) and *ombwa yomange* (female dog).
- For birds and little animals, when in the diminutive, the concordant is the one corresponding to the class, but the determinant can be diminutive for females and augmentative for males: *okanjila k'okamange* (little female bird) or *okanjila k'ochilume* (little male bird).
- There are no articles: *Omoko* (knife), *Ositu* (meat) or *Oviti* (trees).

2.2.2 Adjectives

In Umbundu, the adjectives are divided into two groups: simple and verbal (derived from verbs), as shown in Table 4.

Table 4 Adjectives classes.

Simple adjectives		Verbal adjectives	
Umbundu	**English**	**Umbundu**	**English**
nene	big	pepa	tasty, delicious
tito	small, little	lula	bitter
lepa	tall	Lehã, neha	smell
Vi	evil, bad	yela	white
wa	good	tekãva	black
ewa	very, much	vela	sick
mbumbulu	short	pya	boiled
sumuluhwa	happy	vola	rotten
osuke	poor	Nyolehã	spoiled
umahele	young	kola	strong
umosi	uno	lile	weak
ukwangusu	forceful, strong	neta	fat
kavali	mutual	kachikapa	despicable
ukwafeka	Native, home	leluka	easy
		Letiwe, moleha	visible

Adjectives are neutral or common for gender, just the names [13]: *ukãyi walepa* (tall woman) and *ulume walepa* (tall man); *ukãyi una uvi* (that woman is evil) and *ulume una uvi* (that man is evil), *ongulo inene* (the pig is big) and *omange yongulo inene* (the sow is big), *ekondombolo inene* (big cock) and *onsanji inene* (big hen).

2.2.3 Verbs

According to Le Guennec and Valente [10], the verbs in Umbundu can be simple or derived. There are only three modes: indicative, conjunctive and imperative. Regarding the indicative mode, taking as reference *-linga*, the radical of the verb *to do* in the three main tenses (present, past and future) is marked by the particles *e-* (for the present: *ndi-linga* – I do]); *a-* (for the past tense: *nda-linga* – I did) and *ka-* (for the future: *ndi-ka-linga* – I will do).

There is no change in the radical of the verb for either the time or the number:

oku-linga (to do)
ame ndi-linga (I do)
etu tu-linga (we do)
ame nda-linga (I did)
etu twa-linga (we did)
ame ndi-ka-linga (I will do)
etu tu-ka-linga (we will do)

Regarding concordance patterns [5, 13]:

1. Noun + Adjective (singular/plural)
   uti unene / *oviti vinene* (big tree / big trees)
   ukãyi wafina / *akãyi vafina* (pretty woman / pretty women)
2. Noun + Determinant Demonstrative
   Onjo ina / *olonjo vina* (that house / those houses)
3. Verb (verbal form) + Noun
   twayeva ondaka (we heard the message)
4. Pronoun + verbal form
   Ame ndilya (I eat)
   Ovo valya (they eat)
5. Noun + numeral
   omoko imosi (one knife)
   olomoko vitatu (three knives)

3 jSpell Environment

jSpell was developed as a fork of the well known Unix spell checker, ispell[2] with the main objective of allowing it to work as a morphological analyzer [1]. It was originally used to construct a morphological analyzer for the Portuguese language.

While jSpell might be used both as a command line application (with a similar interface as ispell) or as a C programming library, a Perl module was developed to help in the process of using it from within NLP tools developed in the Perl programming language [11].

jSpell dictionaries are comprised of two different files:

- A dictionary with a list of lemmas (word roots), the morphological properties for the lemma, and a list of flexion paradigm identifiers.
- An affix file, containing the flexion paradigm rules. Each flexion paradigm includes rewriting rules that specify how a word form can be generated from the lemma (what portions of the word are removed, added or rewritten) and how the morphological analyses changes with the flexion (for example, changing gender, number, or verb tense).

From this set, jSpell is not only able to check spelling, but also perform morphological analysis (without disambiguation) and to generate word forms accordingly with the morphological constrains needed.

[2] `https://www.gnu.org/software/ispell/` [accessed on 2020-07-03]

Another interesting property of jSpell is the ability to work in a guess mode, where unknown words are analyzed for possible derivation from any known lemma. This allows the tool to do morphological analysis on unknown words, while trying to apply any flexion paradigm to any lemma in the dictionary.

As the Portuguese dictionary for jSpell was the source of the first spell checking dictionary for the Portuguese language, that was shipped within all major Linux distributions together with ispell, the necessity to adapt the dictionary to other engines, like aspell[3], HunSpell[4] and MySpell[5] soon arrived. This lead to the development of an environment able to convert from dictionaries from jSpell format into any other dictionary format [14], automating the creation of packages for all major open source tools, like LibreOffice, OpenOffice, Firefox, Thunderbird and all Linux spell checking engines.

4 Dictionary Bootstrap

To help in the bootstrap process, the first task was to compile corpora from the Internet. Given we do not intend to release (yet) an Umbundu corpus, at the moment the approach did not take into account copyright issues.

Being a language with oral tradition, and with a relatively small number of written resources, some of the issues found with other languages years ago are still present in Umbundu: there is a lot of different ways to write words. While there are some dictionaries, the majority of the population is illiterate.

Therefore, our task cannot be completely guided from corpora, forcing us to query language users and to consult written references, as Le Guennec and Valente [10] dictionary of Portuguese/Umbundu.

We focused on nouns and regular verbs. The next sections present some of the rules currently in use, exemplifying the results.

4.1 Nouns

For the plural paradigm, the rules are written as:

```
flag p:
  > -E    , A     ; "N=p"     # door: epito / apito
  > -S    , OLOS  ; "N=p"     # monkey: sima / olosima
  > -OMU  , OMA   ; "N=p"     # person: omunu / omanu
  > -OMO  , OMA   ; "N=p"     # child: omola / omala
  > -OLU  , OLO   ; "N=p"     # grain: olumema / olomema
  > -OCI  , OVI   ; "N=p"     # song: ocisungo / ovisungo
  > -OTCHI, OVI   ; "N=p"     # white: otchindele / ovindele
  > -OTCHE, OVYE  ; "N=p"     # election: otchela / ovyela
  > -OKU  , OVI   ; "N=p"     # food: okulya / ovilya
  > -OKA  , OTU   ; "N=p"     # bread roll: okambolo / otumbolo
  > -M    , VAM   ; "N=p"     # brother: mange / vamange
  > -U    , A     ; "N=p"     # man: ulume / alume
```

This paradigm is comprised of different rules for different prefixes. Before the > sign, a pattern to match lemmas prefixes is specified. In this example, none of the rules have a pattern, meaning that the rules will be applied if their rewrite rule can be applied. After

[3] http://aspell.net/ [accessed on 2020-07-03]
[4] https://hunspell.github.io/ [accessed on 2020-07-03]
[5] https://www.openoffice.org/lingucomponent/dictionary.html [accessed on 2020-07-03]

the > sign, there is the rewrite part: first the characters to remove from the beginning of the word, and then the characters to be prefixed. After the semicolon, a set of properties to rewrite the morphological analysis metadata is presented. Everything after the sharp sign are comments, illustrating the rules usage.

Note that the rules, themselves, do not specify if the rewrite will take place in the beginning of the words or at the end. That information is provided by a separator in the rules file, specifying that rules following that separator will be applied as prefixes.

Unfortunately not all words follow the generic plural construction. For some of them there was the need to create another paradigm, regarding irregular plurals:

```
flag q:
  E [WKYP] > -E, OVA  ; "N=p"  # hand: eka / ovaka
  E [^WTKY]> -E, OVI  ; "N=p"  # wrinkle: enha / ovanha
  U [^TN]  > -U, OVO  ; "N=p"  # work: upange / ovopange
  U [TN]   > -U, OVI  ; "N=p"  # heart: utima / ovitima
           > -O, OLO  ; "N=p"  # window: ombana / olombana
```

4.2 Verbs

A first set of rules for the basic regular verb flexion was already added, including the present, perfect-past and future tenses, and the semantic negative form for the present tense. A subset of the rules follows.

```
flag v:
 # Examples for okulya (to eat)

 # TENSE: present
 > -OKU, NDI  ; "P=1,N=s,T=ip" # ndilya
 > -OKU,  TU  ; "P=1,N=p,T=ip" # tulya
 > -OKU,   U  ; "P=2,N=p,T=ip" # ulya
 > -OKU,  VA  ; "P=3,N=p,T=ip" # valya

 # TENSE: perfect-past
 > -OKU, NDA  ; "P=1,N=s,T=ipp" # ndalya
 > -OKU,   O  ; "P=2,N=s,T=ipp" # olya
 > -OKU,  WA  ; "P=3,N=s,T=ipp" # walya
 > -OKU, TWA  ; "P=1,N=p,T=ipp" # twalya

 # TENSE: future
 > -OKU, NDIKA ; "P=1,N=s,T=if" # ndikalya
 > -OKU,   UKA ; "P=2,N=s,T=if" # ukalya
 > -OKU,   OKA ; "P=3,N=s,T=if" # okalya
 > -OKU,  VAKA ; "P=3,N=p,T=if" # vakalya

 # Negated sense: to starve

 # TENSE: present
 > -OKU,  SI ; "P=1,N=s,T=ip,M=neg" # silya
 > -OKU,  KU ; "P=2,N=s,T=ip,M=neg" # kulya
 > -OKU,KAVU ; "P=2,N=p,T=ip,M=neg" # kavulya
 > -OKU,KAVA ; "P=3,N=p,T=ip,M=neg" # kavalya
```

Another different kind of rule is the modal abstraction, creating a noun from a verb:

```
flag u:
  > U ; "CAT=nc,MO=abs"  # to fear: sumba / fear: usumba
```

4.3 Dictionary

The dictionary was constructed from different sources in the Internet. As stated previously, at this moment we are not pretending to create a public corpus, and therefore we did not manage the rights for the obtained text. Nevertheless, these texts allowed us to create lists of terms, and understand and test the morphological rules.

The dictionary itself, is a list of lemmas, followed by a morphological information (in the examples presented, `#v` stands for a verb, while `#nc` stands a common name. For reference, and to allow further uses of the dictionary, we also included a Portuguese translation. At this point we are not concerned with polysemy. Our main goal is to register the information, to help us understand and work with the language. The Portuguese language was chosen given it is the main language used in Angola's written documents. At the end of each line, there is a flag, that represents the inflexion paradigm that is being applied to that word.

```
okulya/#v,pt=comer/v
okwiva/#v,pt=roubar/v
okupapala/#v,pt=brincar,#d/v
okutanga/#v,pt=estudar,#d/v

ekandu/#nc,pt=pecado/p
epito/#nc,pt=porta/p
etali/#nc,pt=pedra/p
sekulu/#nc,pt=ancião/p
ukãyi/#nc,pt=mulher/p
ukombe/#nc,pt=visita/p
ulume/#nc,pt=homem/p
olunyi/#nc,pt=formiga/p
okavisungo/#nc,pt=canção/p
okulama/#nc,pt=saudação/p

upange/#nc,pt=trabalho/q
ekomohiso/#nc,pt=maravilha/q
ewe/#nc,pt=pedra/q
ondjo/#nc,pt=casa/q
ovilwa/#nc,pt=assobio/q
eteke/#nc,pt=dia/q
```

5 Results Discussion

As previously referred, this is the kick-off for a project on the creation of tools and resources for the Angolan indigenous languages. Although we intend to collaborate with institutions and researchers from Angola, this first proof of concept was developed to understand these language characteristics.

The current morphological analyzer coverage is quite limited. The dictionary is comprised of two different sources:

- about 650 lemmas, encoded with their derivation paradigm, generating more than 6 500 different forms;
- a second dictionary with more than 1 700 forms, that are being manually validated, and classified with a derivation paradigm.

These two sources together cover more than 8 000 different word forms.

While in an early stage of development, we foresee to have a free and publicly available dictionary for the Umbundu language available in the near future.

References

1. José João Almeida and Ulisses Pinto. Jspell – um módulo para análise léxica genérica de linguagem natural. In *Actas do X Encontro da Associação Portuguesa de Linguística (APL'1994)*, pages 1–15, 1995.
2. Lluís V. Aracil. *Papers de sociolingüística*. Edicions La Magrana, Barcelona, 1982.
3. Boubacar Diarra. Choice and description of national languages with regard to their utility in literacy and education in Angola. In *A paper delivered at the UNESCO Expert Pool Meeting on Language issues in Literacy and Basic Education*, 1992.
4. Charles Ferguson. Diglossia. *Word*, 15:325–340, 1959.
5. João Fernandes and Zavoni Ntondo. *Angola: povos e línguas*. Editorial Nzila, Luanda, 2002.
6. Malcolm Guthrie. *The classification of the Bantu languages*. Oxford University Press, 1948.
7. INE. Resultados definitivos do recenseamento geral da população e da habitação de angola. Technical report, Instituto Nacional de Estatística, Gabinete Central do Censo, Subcomissão de Difusão de Resultados, Luanda, 2016.
8. Botelho Isalino Jimbi. A reflection on the umbundu corpus planning for the Angola education system: towards the harmonization of the catholic and the protestant orthographies. In *Actas Do XIII Congress Internacional de Linguistica Xeral*, page 475–482, 2018. Retrieved from `http://cilx2018.uvigo.gal/actas/pdf/661789.pdf`.
9. Francis Katamba. Bantu nominal morphology. In D. Nurse and G. Philippson, editors, *The Bantu Language*. Routledge Tayloer & Francis Group, London, 2014.
10. Gregoire Le Guenec and José Francisco Valente. *Dicionário Português-Umbundu*. Escolar Editora, Lobito, 2010.
11. Alberto Manuel Simões and José João Almeida. `jspell.pm` – um módulo de análise morfológica para uso em processamento de linguagem natural. In *Actas da Associação Portuguesa de Linguística (APL'2001)*, pages 485–495, 2002.
12. Universal Declaration of Linguistic Rights, 1996. (Barcelona Declaration) World Conference on Linguistic Rights, Barcelona, Espanha, Junho de 1996. Retrieved January 31, 2020, from `https://unesdoc.unesco.org/ark:/48223/pf0000104267`.
13. João Francisco Valente. *Gramática Umbundu. A língua do centro de Angola*. Junta de Investigação do Ultramar, Lisboa, 1964.
14. Rui Vilela. Geração de dicionários para correcção ortográfica do português. Master's thesis, Escola de Engenharia, Universidade do Minho, 2009.

Syntactic Transformations in Rule-Based Parsing of Support Verb Constructions: Examples from European Portuguese

Jorge Baptista[1]
University of Algarve, Campus de Gambelas, Faro, Portugal
INESC-ID, Lisboa, Portugal
https://www.researchgate.net/profile/Jorge_Baptista
jbaptis@ualg.pt

Nuno Mamede
Universidade de Lisboa, Instituto Superior Técnico, Portugal
INESC-ID, Lisboa, Portugal
Nuno.Mamede@inesc-id.pt

Abstract

This paper reports on-going work on building a rule-based grammar for (European) Portuguese, incorporating support verb constructions (SVC). The paper focuses on parsing sentences resulting from syntactic transformations of SVC, and presents a methodology to automatically generate testing examples directly from the SVC Lexicon-Grammar matrix where their linguistic properties are represented. These examples allow both to improve the linguistic description of these constructions and to test intrinsically the system parser, spotting unforeseen issues due to previous natural language processing steps.

2012 ACM Subject Classification Computing methodologies → Natural language processing; Computing methodologies → Natural language generation; Computing methodologies → Language resources

Keywords and phrases Support verb constructions, Rule-based parsing, syntactic transformations, language resources, European Portuguese

Digital Object Identifier 10.4230/OASIcs.SLATE.2020.11

Funding This work was supported by national funds through FCT, Fundação para a Ciência e a Tecnologia, under project UIDB/50021/2020.

Acknowledgements The authors would like to thank Sónia Reis (U.Algarve and INESC-ID Lisboa for her help in the linguistic data compilation and revision, as well as her attentive reading of initial versions of this manuscript. Naturally, any errors herein are our own responsibility alone.

1 Transformations on Support Verb Constructions: Why is this still a thing?

This paper addresses some issues involved in parsing Support Verb Constructions (hence-forward *SVC*), considering not only the basic, elementary sentence forms, but also the sentences that result from the basic form having undergone some type of transformation (both some very general transformations and other not-so-general operations, but specific of these construnctions).

[1] Corresponding author

SVC are a large set of the elementary (or base) sentences of many languages, and consist of a *predicate noun* (*Npred*) and a *support verb* (*Vsup*), along with its subject and eventual essential complements. The concept of support verb can be traced back to Zellig S.Harris [31, p.216], though the term has been coined much later by M. Gross [25]. In a sentence such as (1):

(1) *O Pedro deu um soco ao João* "Pedro gave a punch to João"

we say that *soco* "punch" is a predicate noun and *deu* "gave" is a support verb. This sentence is a clear example of a SVC: the predicate noun *soco* "punch" is the nucleus of the elementary sentence, the element that conveys the semantic predicate, while the support verb can be considered a specialised type of auxiliary, practically devoid of meaning, and whose function is, basically, to convey the person-number and tense values, which the noun cannot express morphologically. It is the predicate noun (and not the verb!) that selects the elements that fill its argument slots; and it is the noun that selects support verb itself (and not vice-versa). It is also this particular verb-noun combination that imposes the sentence structure, including the prepositions introducing the prepositional complements (if any), as well as the syntactic properties of the construction.

Though the study of SVC is a well-established field of enquiry, dating at least from the early 1960s [31], when the linguistic status of these constructions came into the focus of theoretical debate ([17]), it has gained a renewed impetus with the recent growing interest in processing multiword expressions (MWE) [18, 46, 47] and the development of linguistic resources (especially annotated *corpora*) [37], particularly those envisioned for machine-learning approaches to MWE extraction [52].

Extensive literature has been produced on SVC, from the linguistic viewpoint, and for many languages (see [33] for an overview and references therein), and much work has been invested in the description of (European) Portuguese SVC, namely on the construction with Vsup *estar Prep* [38], *ser de*, *dar* [4, 6, 51], *fazer* [16] and others [2, 20]. More recently, extensive surveys of SVC from the Brazilian variety of Portuguese have been produced: [21] (*fazer*), [45] (*ter*), [41] (*dar*) and others [14, 43].

As multiword expressions [15, 18, 44], SVC constitute a challenge for Natural Language Processing (NLP), both in the perspective of their automatic recognition in texts [32, 47] and their integration in NLP systems [46, 40, 39]. Some corpora are also available for testing the processing of MWE, including some types of SVC [37, 42] (see [18] for an overview).

In spite of the volume of the work already produced, not much attention has been given to the challenges posed by transformations to the parsing of SVC. Not only do SVC give rise to specific transformations, such as:

Conversion [24]:

(2) *O Pedro deu um soco ao João = O João levou um soco do Pedro*
"Pedro gave a punch to (punched) João = João took a punch from Pedro"

complex NP formation [25]:

(3) *o soco que o Pedro deu ao João <...> = o soco do Pedro ao João <...>*
"the punch that Pedro gave to João = the punch of Pedro to João"

Nasp aspectual noun insertion [38]:

(4) *A empresa está em (processo de) reestruturação*
"The company is in (process of) restructuring"; and

Vop Vsup reduction and CSV restructuring under the so-called (causative) operator verbs (Vopc)[25]:

(5) *O Pedro tem medo do escuro = Aquele incidente causou-lhe medo do escuro*
"Pedro has fear of the dark (Pedro is afraid of the dark) = That incident caused him fear of the dark"

Still, SVC can also undergo very general transformations, such as [Passive], [Relative], [Symmetry] [5], and [NP restructuring] [3, 29, 34]. Even if most of these operations are already relatively well-known, their combined application to SVC render the task of parsing these complex constructions a non-trivial task. For lack of space, the reader will refer to the references above for a more detailed description of the SVC specific properties and the associated transformations. This paper main contribution resides, thus, in a method to systematically explore this complex interaction of SVC lexicon-grammar and associated transformations within the scope of building a rule-based grammar for parsing Portuguese texts.

The paper reports on an on-going project to build an integrated lexicon-grammar of Portuguese SVC, within the Lexicon-Grammar (LG) theoretical and methodological framework [25, 28, 33]. Extant linguistic descriptions date from the late 1980's til more recent work on the Brazilian variety (mid-2010s). In this paper, the focus is the European Portuguese SVC. In the development of this research, we have come to realize that some authors did not always use precisely the same definitions for many of their distributional and transformational descriptions, so we put to ourselves the task of compiling and revising all this immense bulk of data, and systematically provide a coherent and explicit description of the linguistic properties encoded in the LG. In the process of doing so, it became obvious that only the more recent work provided illustrative (either artificial or corpus-retrieved) examples for the linguistic description. The change in perspective was slow but steady, very probably having begun with [27] (French adverbial idioms). Older work (until the late 1990s) had few to no examples next to the LG resources, which were typically encoded in binary matrices. It was up to the linguist to creatively devise the adequate wording for the abstract, structural (and often theoretically motivated) description encoded in the matrices, though taking several precautions not to produce biased examples [26]. Naturally, the technological evolution brought by the personal computer and the renewed impetus of corpus-based, data-driven Linguistics also had some influence in this shift.

Example-building is not trivial, and several strategies can be combined to achieve different purposes. More recently, when describing Portuguese verbal constructions (full or distributional verbs) [7, 8, 11], and verbal idioms [9, 12, 19, 23] in view of their integration into STRING [35], a NLP pipeline system, with a rule-based parser (XIP)[1], we also felt the need to produce in a systematic way a comprehensive set examples. In these cases, first steps were taken to deal with lexically constraint transformations, that is, a limited set of transformations, specific to the verbal constructions and the verbal idioms. These transformations include pronominalisations, passive constructions (with both auxiliary verbs *ser* and *estar* "be"), symmetry[5, 10], and some types of NP restructuring [3] (see below).

The goal of automatically generating examples directly from the linguistic description in the LG served two main purposes:

- to validate the grammar rules devised for the parser, and thus serving as a testing benchmark; previous processing steps (POS-tagging and disambiguation, chunking, and dependency extraction) may fail and the error is not a fault of the piece of grammar produced for that particular phenomena under study, but it results, instead, from the pervasive ambiguity and complexity of natural language and the considerable difficulty in solving it in full;

- to facilitate the task of spotting linguistic inconsistencies or inadequacies in the LG description, thus enabling the linguist to revise, correct or complete the linguistic data in the LG resource and, eventually, aid in the development of a more precise grammar.

Both these situations will be exemplified.

Naturally, using a mechanical instead of a manual process to produce examples for the LF of SVC was soon necessary due to the complexity of the task, the many linguistic factors involved and the complex interaction between successive transformations applied to the base form. This is not to say that using a real-life, corpus-based, evaluation scenario, such as the one used in [37], could not be used for evaluating both the linguistic resources and the rule-based grammar, as that type of evaluation can be made to improve both, adding to structural description the dimension of usage. This, however, is out of the scope of this work.

The paper is organized as follows: Next, in Section 2, a brief description of the example generation process is provided, and preliminary results are presented (Section 3). The paper concludes (Section 4) with some remarks on current issues and perspectives for future work.

2 Example generation

To automatically generate examples of SVC directly from the linguistic information encoded in the SVC lexicon-grammar matrix, a Perl software was developed in-house. During the LG construction, another software, also developed in-house, validates the format and the consistency of the data and outputs error messages, allowing the correction and maintenance of the data matrix. This is done by a set of several dozens of rules. For example, if the number of arguments of a Npred is only one, then all the properties for the N_1 and N_2 argument slots must be marked "-", otherwise an error message is produced.

In the LG matrix, each line corresponds to a lexicon-grammar entry (a predicate noun); multiple word senses appear in distinct lines. Each Npred is defined according to the arity of its argument domain, and this can be either "1" (only subject, N_0), "2" (subject N_0 and first complement N_1), or "3" (subject N_0, first N_1 and second complements complement N_2). Example-generating rules are structured according to the number of arguments.

Distributional constraints (on argument slots) are used to generate the examples. These include human/non-human opposition, for instance, but can sometimes be further refined using semantic features. The semantic features were adapted from E. Bick semantic prototypes [13][2]. Besides those features, particularly relevant lexical items are explicitly stated, distinguishing lemmas and inflected/invariant forms The set of distributional constraints is then translated into a *basic string*. These also help define in a more precise way those properties. For example, for subject (N_0) distributional constraints, the following basic strings are used :

Nhum ± human noun; typically, a proper noun: *o Pedro*;

NñHum ± non-human noun; typically a concrete noun: *esta coisa* "this thing"; for consistency, other non-human features [Npc], [Nloc] and [Npred_de_N] (see below) imply that [NñHum] be marked as "-".

Nnr ± non-constraint noun; weakly constraint slot, with a <cause> semantic role; only used for subject: *isto* "this" ;

Npc ± body-part noun, represented by the semantic prototype "sem-an" in the appropriate matrix column, and by a list of nouns, adequate for a given Npred; the basic string is produced by using the first lexical item of that list; otherwise, it uses *a mão* "the hand" as a *portmanteau* word (irrespective of its adequacy);

[2] Semantic roles, based on [48, 49, 50] are indicated for each argument slot but they are not used for example generation.

Nloc ± locative noun: *este lugar* "this place";
Npred_de_N ± complex NP with a Npred head and its arguments (currently not implemented);
Vinfw ± infinitive subclause: *o Pedro fazer isso* "Pedro to_do this";
QueFconj ± finite sub-clause in the subjunctive "mood": *que o Pedro faça isto* "that Pedro does this";
QueFind ± finite sub-clause in the indicative "mood": *que o Pedro faz isto* "that Pedro does this";
O_facto_de_queF ± factive sub-clause: **o facto de o Pedro fazer isto** "the fact that Pedro does this";
Npl-obr ± obligatory plural (currently not implemented);

First (N_1) and second (N_2) complement distributional constraints are encoded in a similar way. For consistency, different proper names were used for N_1 (*João*) and N_2 (*Rui*) complements. Also, different determiners (e.g. *essa coisa* "that thing", and *aquela coisa* "the other thing") and, in the case of completives, different indefinite pronouns (*isso* and *aquilo* "that") were used to distinguish these syntactic slots. Prepositions introducing the complements ($Prep_1$ and $Prep_2$, respectively) are taken directly from the matrix, where they are explicitly provided.

Three different sentence structures are associated to Vsup Npred constructions and represented in the LG, both for the standard and the converse constructions:
CDIR ± for direct-transitive support verbs, where the Npred is the direct complement of the Vsup, e.g. *dar um soco* "give a punch";
PREDSUBJ ± for copula-like Vsup like *estar Prep* "be Prep", with a Prep introducing the Npred, e.g. *O Pedro está em crise* "Pedro is in crisis"; and
MOD ± for verbs with the Npred in a prepositional complement; e.g. *O Pedro sofre de asma* "Pedro suffers from asthma".
For each type of these three types of SVC construction, the Vsup selected by each Npred are listed; Vsup-Prep pairings in the `PREDSUBJ` and `MOD` construction are also indicated. The preposition introducing the <agent-like> complement in the converse construction is also explicitly indicated (mostly, Prep *de* and *da/por parte de*).

These structures have to do with the dependencies produced by the system's parser using the Portuguese grammar. As explained in [40], we identify the SVC by a specific dependency `support`, linking the Npred to the Vsup; a feature `_vsup-standard/converse` indicates wether this is a standard or a converse construction, which will be relevant for semantic role labelling at a later stage; e.g.,

(6) *O Pedro estabeleceu uma aliança com o João* "Pedro established an alliance with João"
`SUPPORT_VSUP-STANDARD(aliança,estabeleceu)`

A similar structural description is also used here to automatically generate the SVC examples. Hence, to generate the example sentence for a `CDIR`-type SVC, the structural elements are aligned, using the basic strings for the arguments, an inflected form of the Vsup, an eventual determiner[3] for the Npred and the prepositions it selects to introduce its eventual complements. In case multiple values appear in the same cell (e.g Prep or Vsup), or for different combinations of distributional constraints on the argument-slots (e.g. human/non-human subject), the algorithm explores all variants and combinations, producing a separate example for each.

[3] For lack of space, determiner-modifier variation has not been described here.

For generating the examples derived by transformations, a similar procedure is carried out. The [dative] pronominalization of the complement arguments, encoded next to the constituent description, is translated by a dative pronoun *-lhe* "to_him", attached to the Vsup, e.g., *O Pedro deu um soco ao João=O Pedro deu-lhe um soco* "Pedro gave him a punch".

The [NP restructuring] involving body-part nouns (Npc; only encoded for N_1), produces a complex subject NP, from two independent constituents, e.g. *O Pedro tem acne no rosto = O rosto do Pedro tem acne* "Pedro has acne on his face = Pedro's face has acne". Complex noun phrase [Complex NP] generation uses the Npred lexical item, followed by the preposition *de* "of" and the subject basic string; for 2- and 3-argument predicates, the corresponding prepositions ($Prep_1$ and $Prep_2$, respectively) are used along with the basic strings of those slots; the basic order of the arguments is maintained.

The [Symmetry] transformation consists in the coordination (*e* "and") of two arguments in a given syntactic slot, using the basic strings of those arguments; in the case of 3-argument predicates, either a subject-object or an object-object coordinated NP is produced, depending on the type of symmetry involved. Hence, for the subject-object symmetric noun *acordo* "agreement" the basic strings produce *esta pessoa e aquela pessoa* [*estão de acordo*] "This person and that person [are in agreement]"; while for the object-object symmetric noun *mistura* "mixture", the basic strings produce [O Pedro fez uma mistura] dessa coisa e aquela coisa "[Pedro did a mixture] of this thing and that thing".

The [ObligNeg] (obligatory negation) property can be seen in SVC that contain an negation element [22], e.g. *O Pedro não esteve pelos ajustes* lit:"Pedro was not by the adjustments" "not to accept or disagree with something that is proposed, presented or required", otherwise the sentence is meaningless or has another unrelated meaning. Generating this examples involves introducing a negation adverb não "not" before the Vsup.

The aspectual nouns [Nasp] insertion [38], come next. These are a type of auxiliary elements that can be inserted in the base sentence leaving the Npred as its complement. They convey an aspectual value, hence the term, and they usually render the sentence more natural. Their function in the SVC is homologous to that of auxiliary verbs (*aka.* verbal periphrasis) in full verb constructions. With Vsup *estar Prep*, the most frequente Nasp are *estado* "state", *fase* "phase", *processo* "process" and, less frequently, *vias* "verge" (7):

(7) *Esta espécie está em extinção = Esta espécie está em* ***vias*** *de extinção* "This species is in extinction (endangered) = This species is on the verge of extinction"

Certain Npred with Vsup *ter* ou *estar com*, denoting "illness/desease" select other *Nasp*, such as *ataque* "attack" and *crise* "crisis" (8):

(8) *O Pedro tem/está com asma = O Pedro está com um* ***ataque****/uma* ***crise*** *de asma*
"Pedro has/is with asthma = Pedro is with an asthma attack/crisis"

Finally, (causative) operator-verbs (Vopc) [25] insertions are described. These verbs reshape the basic SVC structure, absorving the Vsup, and altering the syntactic dependencies associated to the Npred arguments. Two structurally different constructions are considered: (i) [VOP-CDIR], when the Npred is a direct complement of the Vop:

(9) *O Pedro tem sede = Isto deu/fez sede ao Pedro*
"Pedro has thirst (is thirsty) = This gave/made thirst to Pedro (made Pedro thirsty)"

(ii) [VOP-MOD] when the Npred is a prepositional complement of the Vop:

(10) *O Pedro está com sede = Isto deixou o Pedro com sede*
"Pedro is with thirst (is thirsty) = This left Pedro with thirst (left Pedro thirsty)"

In the [Passive] constructions, not only is the sentence with auxiliary verb *ser* "be" generated, but also all the reductions that it can undergo both in the standard and in the converse constructions:

(11) *O Pedro deu um soco ao João* "Pedro gave a punch to João" [STD]

(12) [Passive] = *Um soco foi dado pelo Pedro ao João* = [Relative] *O soco que foi dado pelo Pedro ao João* = [RedRel] *O soco dado pelo Pedro ao João* = [RedVsup] *O soco do Pedro ao João*
"A punch was given by Pedro to João = The punch that was given by Pedro to João = The punch given by Pedro to João = The punch by Pedro to João"

(13) *O João apanhou um soco do Pedro* "João got a punch from Pedro"

(14) [Passive] = *Um soco foi apanhado pelo João do Pedro* = [Relative] *O soco que foi apanhado pelo João do Pedro* = [RedRel] ?*O soco apanhado pelo João do Pedro*
"A punch was caught by João from Pedro = The punch that was caught by João from Pedro = The punch caught by João from Pedro"

A specific column was added to the LG matrix representing gender-number values of the Npred, in order to ensure the correct agreement with the sentences' elements (determiner, modifier and Vsup agreement). A list of tensed forms for each Vsup was used to produce more natural sentences.

3 Results

At the time of submission, the SVC Lexicon-Grammar of European Portuguese contains approximately 7,150 entries. So far, 2,741 (38.3%) have been carefully revised. From these, 1,487 have only one argument, 1,178 have 2 arguments, and 76 have 3 arguments. For an easier inspection of the generated examples, each sentence-type is outputted to a different file. Table 1 shows the breakdown of the generated examples per sentence type and per number of the Npred arguments. A dash "–" indicates that the sentence type cannot be construed, while the note (a) corresponds to work still in progress.

First, we remark that these are just preliminary results. Still, even if only a little more than 1/3 of the LG entries have been processed, it is already evident that the number of automatically generated examples (48,421) is quite impressive. Furthermore, some transformations are still being worked out. It is likely that other, though less productive, transformations are to be added.

In order to assess the generated examples, same caution is required, keeping in mind that our goal is *not* the generation of entirely natural utterances, but their analysis. In other words, the purpose of those example sentences is to test the STRING system's [35] and its parsing module XIP [1], when dealing with SVC, and, particularly at this stage, to extract the SUPPORT dependency out of those examples. Stylistic considerations, though important in a generation system, are secondary here.

Furthermore, the size of the list of examples being so large, it is difficult to provide a direct quantitative assessment of the generated examples, so we will limit ourselves to highlight the main issues detected. For example, in these artificial sentences, constituents

Table 1 SVC automatically generated examples.

Sentence type	Arg=1	Arg=2	Arg=3
STD	10,458	9,059	795
STD-Pass	1,125	4,628	570
STD-NP	2,002	1,889	280
STD-Nasp	815	818	0
STD-ObligNeg	4	0	0
STD-VOP-CDIR	4,687	2,571	(a)
STD-VOP-MOD	1,388	1,693	(a)
STD-NP-Restr	–	1,035	(a)
STD-Dat	–	84	58
STD-Sym	–	2,567	113
CNV	–	(a)	(a)
CNV-Pas	–	1,476	306
Sub-total	20,479	25,820	2,122
Total		**48,421**	

are produced in the basic word order. This often produces formally (syntactically) correct but stylistically dubious (or even unacceptable) sentences; e.g. a subject infinitive sub-clause is more natural if moved to the end of the sentence:

(15) ?*O Pedro fazer isto está na moda* "Pedro to-do this is in fashion (=is fashionable)"
= *Está na moda o Pedro fazer isto* "[it] is in fashion Pedro to-do this"

Also, notice that in the sub-clause a zero-indefinite or an indefinite subject is preferable that the basic string *o Pedro*:

(16) *Está na moda fazer isto* "[it] is in fashion to-do this";
Está na moda as pessoas fazerem isto "[it] is in fashion people to-do this".

Distributional constraints are only approximated by the basic strings chosen for the generation process. This produces sometimes quite bizarre expressions. For example, *estar de esperanças* "be of/with hopes/expectations" or *estar no seu estado interessante* "be in her interesting state", which means "to be pregnant", can hardly accept a masculine subject like *o Pedro*. In other cases, a human-collective noun would better suit the Npred:

(17) *O Pedro fez uma inspeção a* (?*o João, ao pelotão, à empresa*)
"Pedro made an inspection to (João/the platoon/the company)"

Co-reference constraints holding between the Npred arguments and the sub-clause arguments (especially its subject) were simply ignored at this stage, in order to simplify the generation process, which produces borderline (if not altogether unacceptable) sentences (co-reference is marked by co-reference indexes in the examples below):

(18) **O Pedro*$_i$ *teve a intenção de o João*$_j$ *fazer isso*
"Pedro had the intention of John to-do this"

(19) cp. *O Pedro*$_i$ *teve a intenção de 0*$_i$ *fazer isso*
"Pedro had the intention of to-do this"

Concerning the Npred determiners, their representation in the Lexicon-Grammar is limited to those the noun selects in the base form. The rationale for this decision is that most of times, the constraints on the Npred determiners are very similar across multiple Vsup constructions of the same Npred. This, in fact, is not always so, and same generated examples are quite awkward. For example, the Npred *juramento* "oath", besides the elementary (basic) Vsup *fazer* "to do/make", also accepts, the variant *prestar* "pay". An exact match query in the `.pt` top domain of the web using Google shows that the first Vsup rarely accepts the zero determiner, while the second is significantly more frequent with this determiner.

(20) *O Pedro ?*fez/prestou juramento ao João/a esta coisa*
"Pedro made/payed oath to João/that thing"

The reverse situation occurs with the indefinite article *um* "a".

(21) *O Pedro fez/?*prestou* um *juramento ao João/a esta coisa*
"Pedro made/payed an oath to João/that thing"

In order to mimic the situations where the Npred imposes the presence of a modifier, we decided to use the basic string *um certo* "a certain". The vagueness of the determiner sometimes produce unnatural examples. The selection of an adequate adjetive can significantly improve the acceptability of the sentence:

(22) *O Pedro está com uma ?certa/forte cãimbra no pé*
"Pedro has got a certain/strong cramp in the foot (=a foot cramp)"

Another aspect that hinders the acceptability of generated examples is the fact that some Npred, though allowing number variation, are much more frequent in the plural with a given Vsup that with another one. This constraint is often associated with the determiners (and some of these combined restrictions may show high regularity). Since in the LG matrix Npred are indicated by their lemma and examples are generated directly form the LG entry, some examples, though grammatically correct, may sound awkward. For example, the Npred *borbulha* "pimple" with Vsup *ter* is much more acceptable in the plural with determiner zero, while both number values are natural with the indefinite article:

(23) *O Pedro tem borbulhas/*borbulha na cara*
"Pedro's got pimples/*pimple on the [=his] face"

(24) *O Pedro tem umas borbulhas/uma borbulha na cara*
"Pedro's got some pimples/a pimple on the [=his] face"

Obligatory Npred plural/singular forms are represented by their surface forms, irrespective of this constraint on number value being strictly morphologic, e.g., *férias* "holidays", *pêsames* "condolences"; or strictly syntactic, e.g., *braços* "arms": *O Pedro está a braços com um problema grave* "Pedro's in having to deal with a serious problem."

Another way to assess the generated examples is to parse them in STRING and check in the output whether the support dependency was correctly extracted. Table 2 shows the breakdown of error rate (false-negatives) per sentence type. The new note (b) indicates that this assessment does not applies to complex NP, as there is no Vsup in such structures.

The system takes 1h19m50s to process the generated examples' files. The overall error-rate is 0.0506 but this value varies widely depending of the sentence type. A detailed error analysis was carried out and the main issues found had to do with inadequacies in different processing stages is STRING, which prevent the SVC detection and the SUPPORT dependency extraction. Here are the most relevante situations found:

Table 2 Parsing automatically generated SVC examples: error rate (false-negatives).

Sentence type	Arg=1	Arg=2	Arg=3
STD	20/10,458=0.0019	183/9,059=0.0202	14/795=0.0176
STD-Pass	12/1,125=0.0106	1,397/4,628=0.3018	2/570=0.0035
STD-NP	(b)	(b)	(b)
STD-Nasp	2/815=0.0025	12/818=0.0146	(a)
STD-ObligNeg	0/4=0.0000	0/0=0.0000	(a)
STD-VOP-CDIR	15/4,687=0.0032	212/2,571=0.0824	(a)
STD-VOP-MOD	167/1,388=0.1203	323/1,693=0.1907	(a)
STD-NP-Restr	–	14/1,035=0.0135	(a)
STD-Dat	–	16/84=0.1904	14/58=0.2413
STD-Sym	–	22/2,567=0.0085	21/113=0.1858
CNV	–	(a)	(a)
CNV-Pas	–	10/1,476=0.0067	0/306=0.0000
Sub-total	216/20,479=0.0105	2,189/25,820=0.0847	51/2,122=0.0240
Total		**2,454/48,421=0.0506**	

(i) lacunae in the system's lexicon; e.g. *bolandas*: *andar em bolandas* "in a bustle"; the new entry was then added to the lexicon;

(ii) misspelling of the Npred lemma in the LG, particularly in the case of compound words and the use of hyphen, as an exact match with that lemma in the system's lexicon; e.g. *dor-de-cotovelo*/*dor de cotovelo* lit:"pain in the elbow" "envy/jealousy"; the entry was corrected in the lexicon-grammar;

(iii) incorrect tokenization of a string as a multiword expression (MWE), especially compound prepositions and adverbs; as tokenization of MWE has priority over simple word sequences, capturing a compound precludes the Npred identification and all subsequent processing steps; e.g. *na direção de* (Prep) "towards" vs. *O Pedro está na direção da empresa* "Pedro is at the head of the company/on the company's board"; conditions were added to the system, in order to prevent the tokenization of the string as a MWE;

(iv) incorrect statistical POS-disambiguation; several situations arose:

- **(a)** an incorrect assignment of a verb tag to the Npred, e.g. *O Pedro teve tosse*$_{N/V}$ "Pedro had a cough"; in some cases, a contextual POS-disambiguation rule could be construed, either selecting the correct POS-tag or discarding the incorrect tag; in other cases, the incorrect tag is extremely rare in the language, so we could discard it as an "exotic" homograph;
- **(b)** an incorrect assignment of a preposition or definite article tag to *a* "to/the-fs", which produces either an incorrect PP chunk, e.g. *a infância* "the childhood": *O Pedro deixou a infância <para trás>* "Pedro left the [=his] childhood behind"; or an incorrect NP chunk, e.g. *a dieta* lit: "to diet" "on a diet": *O Pedro está a dieta* "Pedro is on a diet"; these cases could not be resolved for the moment.

(v) subtle interaction of lexical features with the chunking module of the parser: with determiner *um certo* "a certain", chunking rules fail to adequately identify the NP or PP headed by the Npred, when this can have a reading as a type of measure unit, a container, a group-of-things, or human collective noun (this semantic prototypes are encoded in the nouns lexical entries. For example, in the sentence *O Pedro tinha um certo comando dessa coisa* "Pedro had a certain command of that thing", the noun

comando designates, among other things the abstract action noun "command". However, the same noun could also designate the human collective noun, as in *o comando de mercenários* "the command of mercenaries'; in this situations, a contextual rule removes the spurious features, preventing the chunking rule to be triggered; and, finally,

(vi) certain nouns are used by the system's local grammars to create complex NOUN nodes, which are relevant for Named Entity Recognition (NER) [30, 36]; for example, the noun *associação* "association" is often used to build Named Entities designating an organisation. For that purpose, the system produces a NOUN node, which leads to an inadequate chunking of the sentence. This issue has not been addressed yet: `680>TOP{NP{O Pedro} VF{fez} NP{a NOUN{associação de o João com o Rui}} .}`

At the deadline for the submission of this paper, most errors from the basic standard construction have been corrected (for CSV with 1 and 3 arguments). Work on 2 arguments, CSV examples continues but it has already dramatically decreased. For the remainder files, attention must be paid to [Passive] and Vop examples, responsible for most false-negative cases. The generation process for the remaining sentence types continues. Several transformations are yet to be formalised.

4 Conclusion and future work

This paper presented a method to generate examples of SVC directly from the linguistic properties encoded in these constructions Lexicon-Grammar, built for European Portuguese. Our focus here was on the transformations allowed by SVC, both the operations that are specific of this type expressions, and other operations with a broader scope, such as Passive. Though only 1/3 of the extant SVC have been processed so far, the system already generates over 48 thousand examples. The current distribution of the generated sentences per sentence type is likely to undergo significant changes, as much of the data already processed was derived from a subset of Portuguese SVC.

Special care was taken to make sentences as simple and as natural as possible. This includes producing adequate nouns for each syntactic slot, as well as choosing the best tense and word order. For commodity, examples from each sentence type are group in distinct output files. Preliminary observations confirm not only that most examples are perfectly natural, but having them systematically spelled out helps correct the linguistic data encoded in the Lexicon-Grammar matrix.

Several transformations still await an adequate representation in the LG matrix for example generation and SUPPORT dependency extraction. These include the alternation between 3-argument and 2-argument symmetric Npred, where the longer structure has a <cause> or <agent-cause> semantic role, e.g., *O Pedro fez uma mistura dessa coisa com aquela coisa* "Pedro made the mix of this thing and that thing", while the 2 argument drops the subject of the longer sentence, e.g. *Esta coisa fez uma mistura com aquela coisa* "This thing made the mix with that thing". A similar process occurs with nouns designating medical procedures, with a 3-argument, agentive subject CSV, e.g., *O Pedro fez um raio-X ao peito do João* "Pedro did an X-ray to João's chest"; and an equivalent, apparently 2-argument, patient subject, *O João fez um raio-X ao peito* "João did an X-ray to the [=his] chest".

In the future, once this step is finished, we would like to devise methods for populating the data base with data from corpora, using the information available as heuristics for corpus exploration. Also, other sentence types have not been considered yet, for example, sentences involving clefting, negation, etc. The interaction between current transformations and new sentence types to be produced will certainly make this work useful for testing, in a systematic way, the robustness of NLP systems when detecting SVC in texts.

References

1 S. Ait-Mokhtar, J. Chanod, and C. Roux. Robustness beyond shallowness: incremental dependency parsing. *Natural Language Engineering*, 8(2/3):121–144, 2002.

2 M. F. Athayde. Construções com verbo-suporte (funktionsverbgefuge) do português e do alemão. *Cadernos do CIEG Centro Interuniversitário de Estudos Germanísticos*, 1, 2001.

3 Jorge Baptista. Conversão, nomes parte-do-corpo e restruturação dativa. In Ivo Castro, editor, *Actas do XII Encontro da Associação Portuguesa de Linguística*, volume 1, pages 51–59, Lisboa, 30 de setembro a 2 de outubro de 1996, Braga-Guimarães, Portugal 1997. Associação Portuguesa de Linguística, APL/Colibri.

4 Jorge Baptista. *Sermão*, *tareia* e *facada*: uma classificação das expressões conversas *dar-levar*. *Seminários de Linguística 1*, pages 5–37, 1997.

5 Jorge Baptista. Construções simétricas: argumentos e complementos. In Olga Figueiredo, Graça Rio-Torto, and F. Silva, editors, *Estudos de homenagem a Mário Vilela*, pages 353–367. Faculdade de Letras da Universidade do Porto, 2005.

6 Jorge Baptista. *Sintaxe dos Nomes Predicativos com verbo-suporte SER DE*. Fundação para a Ciência e a Tecnologia/Fundação Calouste Gulbenkian, Lisboa, 2005.

7 Jorge Baptista. Viper: A Lexicon-Grammar of European Portuguese verbs. In Jam Radimsky, editor, *Actes du 31e Colloque International sur le Lexique et la Grammaire*, pages 10–17, République Tchèque, 2012. Université de Bohême du Sud.

8 Jorge Baptista. ViPEr: uma base de dados de construções léxico-sintáticas de verbos do Português Europeu. In Fátima Silva, Isabel Falé, and Isabel Pereira, editors, *Actas do XXVIII Encontro da APL - Textos Selecionados*, pages 111–129, Lisboa, 2013. APL/Colibri.

9 Jorge Baptista, Graça Fernandes, Rui Talhadas, Francisco Dias, and Nuno Mamede. Implementing European Portuguese verbal idioms in a natural language processing system. In Gloria Corpas Pastor, editor, *Computerised and Corpus-based Approaches to Phraseology: Monolingual and Multilingual Perspectives / Fraseología computacional y basada en corpus: perspectivas monolingües y multilingües*, pages 102–115. Proceedings of Conference of the European Society of Phraseology (EUROPHRAS 2015), June 28-July 2, 2015, Málaga, Spain, Geneva, Switzerland 2016.

10 Jorge Baptista and Nuno Mamede. Reciprocal Echo Complements in Portuguese: Linguistic Description in view of Rule-based Parsing. In Jorge Baptista and Mario Monteleone, editors, *Proceedings of the 32nd International Conference on Lexis and Grammar (CLG'2013)*, pages 33–40, Faro, Portugal, September 10–14, 2013 2013. CLG'2103, Universidade do Algarve – FCHS.

11 Jorge Baptista and Nuno Mamede. *Dicionário Gramatical de Verbos do Português Europeu*. Universidade do Algarve, December 2020.

12 Jorge Baptista, Nuno Mamede, and Ilia Markov. Integrating verbal idioms into an nlp system. In Jorge Baptista, Nuno Mamede, Sara Candeias, Ivandré Paraboni, Thiago Pardo, and Maria das Graças Volpe Nunes, editors, *Computational Processing of the Portuguese Language*, volume 8775 of *Lecture Notes in Computer Science / Lecture Notes in Artificial Intelligence*, pages 251–256, Berlin, 2014. 11^{th} International Conference PROPOR'2014, October 8-10, 2014, São Carlos – SP, Brazil, Springer.

13 Eckhard Bick. Noun sense tagging: Semantic prototype annotation of a Portuguese treebank. In *Proceedings of TLT*, pages 127–138, 2006.

14 Nathalia Calcia. Descrição e classificação das construções conversas no português do Brasil. Master's thesis, Universidade Federal de São Carlos, São Carlos-SP, Brasil, 2016.

15 Nicoletta Calzolari, Charles J. Fillmore, Ralph Grishman, Nancy Ide, Alessandro Lenci, Catherine Macleod, and Antonio Zampolli. Towards best practices for Multiword Expressions in Computational Lexicons. In *Proceedings of LREC'02*, pages 1934–1940, Las Palmas, Spain, May 2002.

16 Lucília Chacoto. *O Verbo Fazer em Construções Nominais Predicativas*. PhD thesis, Universidade do Algarve, Faro, 2005.

17 Noam Chomsky. *Remarks on nominalization*. Linguistics Club, Indiana University, 1968.

18 Mathieu Constant, G. Eryigit, Joana Monti, L. van der Plas, Carlos Ramisch, Michael Rosner, and A. Todirascu. Multiword expression processing: A survey. *Computational Linguistics*, pages 837–892, 2017.

19 Gloria Corpas Pastor, Ruslan Mitkov, Maria Kunilovskaya, and María Araceli Losey León, editors. *Processing European Portuguese Verbal Idioms: From the Lexicon-Grammar to a Rule-based Parser*, Malaga (Spain), September, 25–27 2019. Tradulex.

20 Maria Francisca Mendes Queiroz-Pinto de Athayde. *A estrutura semântica das construções com verbo-suporte preposicionadas do português e do alemão*. PhD thesis, Faculdade de Letras da Universidade de Coimbra, Coimbra, 2000.

21 Cláudia Dias de Barros. *Descrição e classificaçãoo de predicados nominais com o verbo-suporte FAZER: especificidades do Português do Brasil*. PhD thesis, Universidade Federal de São Carlos, São Carlos-SP, Brasil, 2014.

22 Graça; Fernandes and Jorge Baptista. Frozen sentences with obligatory negation: Linguistic challenges for natural language processing. In Carmen Mellado-Blanco, editor, *Colocaciones y fraseología en los diccionarios*, pages 85–96. Peter Lang, Frankfurt, 2008.

23 Ana Galvão, Jorge Baptista, and Nuno Mamede. New developments on processing European Portuguese verbal idioms. In Carlos Augusto Prolo and Leandro Henrique Mendonça de Oliveira, editors, *12th Symposium in Information and Human Language Technology*, pages 229–238, Salvador, BA (Brazil), October, 15–18 2019.

24 Gaston Gross. *Les construction converses du français*. Droz, Genève, 1989.

25 Maurice Gross. Les bases empiriques de la notion de prédicat sémantique. *Langages*, 15(63):7–52, 1981.

26 Maurice Gross. Methods and tactics in the construction of a lexicon-grammar. In *Linguistics in the Morning Calm, Selected Papers from SICOL*, pages 177–197, Seoul, 1988. Hanshin Pub. Co.

27 Maurice Gross. *Grammaire transformationnelle du français: 3 - Syntaxe de l'adverbe*. ASSTRIL, Paris, 1996.

28 Maurice Gross. Lexicon-grammar. In Keith Brown and Jim Miller, editors, *Concise Encyclopedia of Syntactic Theories*, pages 244–259. Pergamon, Cambridge, 1996.

29 Alain Guillet and Christian Leclère. Restructuration du groupe nominal. *Langages*, 15e année(63):99–125, 1981.

30 Caroline; Hagège, Jorge; Baptista, and Nuno João Mamede. Reconhecimento de entidades mencionadas com o xip: Uma colaboração entre o inesc-l2f e a xerox. In Cristina; Mota and Diana Santos, editors, *Desafios na avaliação conjunta do reconhecimento de entidades mencionadas: Actas do Encontro do Segundo HAREM (Aveiro, 11 de Setembro de 2008)*. Linguateca, 2009.

31 Zellig Harris. The elementary transformations. In Henry Hiz, editor, *Papers on Syntax*, pages 211–235. D. Reidel Publishing Company, 1964.

32 Adam Kilgarriff, Pavel Rychly, Vojtech Kovar, and Vit Baisa. Finding multiwords of more than two words. In *EURALEX*, Oslo, 2012.

33 Béatrice; Lamiroy. Le lexique-grammaire. In *Travaux de Linguistique*, volume 37. Duculot, 1998.

34 Christian Leclère. Sur une restructuration dative. *Language Research*, 31:179–198, 1995.

35 Nuno Mamede, Jorge Baptista, Cláudio Diniz, and Vera Cabarrão. STRING - A Hybrid Statistical and Rule-Based Natural Language Processing Chain for Portuguese. In Alberto Abad, editor, *International Conference on Computational Processing of Portuguese (PROPOR 2012) - Demo Session*, Coimbra, Portugal, April, 17–20 2012.

36 Diogo Oliveira. Extraction and classification of named entities. Master's thesis, Instituto Superior Técnico - Universidade Técnica de Lisboa Universidade Técnica de Lisboa, L^2F/INESC-ID, Lisboa, 2010.

37 Carlos Ramisch, Silvio Cordeiro, Agata Savary, Veronika Vincze, Verginica Mititelu, Archna Bhatia, Maja Buljan, Marie Candito, Polona Gantar, Voula Giouli, et al. Edition 1.1 of the parseme shared task on automatic identification of verbal multiword expressions. In *Joint Workshop on Linguistic Annotation, Multiword Expressions and Constructions*, 2018.

38 Elisabete Ranchhod. *Sintaxe dos predicados nominais com "estar"*. Instituto Nacional de Investigação Científica (INIC), 1990.

39 Amanda Rassi, Nuno Mamede, Jorge Baptista, and Oto Vale. I. Integrating support verb constructions into a parser. In *Proceedings of the Symposium in Information and Human Language Technology (STIL'2015)*, pages 57–62, 2015.

40 Amanda Rassi, Cristina Santos-Turati, Jorge Baptista, Nuno Mamede, and Oto Vale. The fuzzy boundaries of operator verb and support verb constructions with dar "give" and ter "have" in Brazilian Portuguese. In *Proceedings of the Workshop on Lexical and Grammatical Resources for Language Processing (LG-LP 2014), COLING 2014*, Dublin, Ireland, August 2014. Workshop on Lexical and Grammatical Resources for Language Processing (LG-LP 2014), COLING 2014, Dublin, August 24, 2014, COLING 2014.

41 Amanda P. Rassi. *Descrição, classificação e processamento automático das construções com o verbo* dar *em português brasileiro*. PhD thesis, Universidade Federal de São Carlos, São Carlos-SP, Brasil, 2015.

42 Amanda. P. Rassi, Jorge Baptista, and Oto Araújo Vale. Um corpus anotado de construções com verbo-suporte em português. *Gragoatá*, 39(1):207–230, June, 2015 2015.

43 Amanda. P. Rassi, N. P. Calcia, Oto A. Vale, and Jorge Baptista. Análise comparativa das construções conversas em português do brasil e português europeu. In *Abstracts from I Congresso Internacional de Estudos do Léxico e suas Interfaces (CINELI)*, page 45, Araraquara, SP (Brazil), March 2014. Congresso Internacional de Estudos do Léxico e suas Interfaces (CINELI), FCL-UNESP.

44 I. A. Sag, T. Baldwin, F. Bond, A. Copestake, and D. Flickinger. Multiword Expressions: A Pain in the Neck for NLP. In *Proceedings of Computational Linguistics and Intelligent Text Processing*, volume 2276 of *LNAI/LNCS*, pages 1–15, Berlin, 2002. 3rd International Conference CICLing-2002, Springer.

45 Cristina Santos. *Construções com verbo-suporte* ter *no Português do Brasilrasil*. PhD thesis, Universidade Federal de São Carlos, São Carlos-SP, Brasil, 2015.

46 Agata Savary, Carlos Ramisch, Silvio Cordeiro, Federico Sangati, Veronika Vincze, Behrang QasemiZadeh, Marie Candito, Fabienne Cap, Voula Giouli, and Ivelina Stoyanova. The PARSEME shared task on automatic identification of verbal multiword expressions. In *13th Workshop on Multiword Expressions (MWE)*, pages 31–47, 2017. doi:10.18653/v1/W17-1704.

47 Agata Savary, Manfred Sailer, Yannick Parmentier, Michael Rosner, Victoria Rosén, Adam Przepiórkowski, Cvetana Krstev, Veronika Vincze, Beata Wójtowicz, Gyri Smørdal Losnegaard, et al. PARSEME–PARSing and multiword expressions within a European multilingual network. In *Multiword expressions at length and in depth: Extended papers from the MWE workshop*, 2015.

48 Rui Talhadas. Semantic Role Labelling in European Portuguese. Master's thesis, Universidade do Algarve, Faculdade de Ciências Humanas e Sociais, Faro, Portugal, 2014.

49 Rui Talhadas, Jorge Baptista, and Nuno Mamede. Semantic roles annotation guidelines. Technical report, L2F/INESC ID Lisboa, 2013.

50 Rui Talhadas, Nuno Mamede, and Jorge Baptista. Semantic Roles for Portuguese Verbs. In Jorge Baptista and Mario Monteleone, editors, *Proceedings of the 32nd International Conference on Lexis and Grammar (CLG'2013)*, pages 127–132, Faro, Portugal, September 10–14, 2013 2013. CLG'2103, Universidade do Algarve – FCHS.

51 Aldina Vaza. Estruturas com nomes predicativos e verbo-suporte dar. Master's thesis, Faculdade de Letras da Universidade de Lisboa, Lisboa, Portugal, 1988.

52 Nicolas Zampieri, Carlos Ramisch, and Géraldine Damnati. The impact of word representations on sequential neural MWE identification. In *Joint Workshop on Multiword Expressions and WordNet (MWE-WN)*, pages 169–175, 2019. doi:10.18653/v1/W19-5121.

Different Lexicon-Based Approaches to Emotion Identification in Portuguese Tweets

Soraia Filipe
Iscte – Instituto Universitário de Lisboa, Portugal
http://www.iscte-iul.pt
Soraia_Filipe@iscte-iul.pt

Fernando Batista
Iscte – Instituto Universitário de Lisboa, Portugal
INESC-ID, Lisboa, Portugal
http://www.inesc-id.pt
fernando.batista@iscte-iul.pt

Ricardo Ribeiro
Iscte – Instituto Universitário de Lisboa, Portugal
INESC-ID, Lisboa, Portugal
http://www.inesc-id.pt
ricardo.ribeiro@iscte-iul.pt

Abstract

This paper presents the existing literature on the identification of emotions and describes various lexica-based approaches and translation strategies to identify emotions in Portuguese tweets. A dataset of tweets was manually annotated to evaluate our classifier and also to assess the difficulty of the task. A lexicon-based approach was used in order to classify the presence or absence of eight different emotions in a tweet. Different strategies have been applied to refine and improve an existing and widely used lexicon, by means of automatic machine translation and aligned word embeddings. We tested six different classification approaches, exploring different ways of directly applying resources available for English by means of different translation strategies. The achieved results suggest that a better performance can be obtained both by improving a lexicon and by directly translating tweets into English and then applying an existing English lexicon.

2012 ACM Subject Classification Computing methodologies → Natural language processing

Keywords and phrases Emotion detection, tweets, Portuguese Language, Emotion lexicon

Digital Object Identifier 10.4230/OASIcs.SLATE.2020.12

Category Short Paper

Funding This work was supported by national funds through FCT, Fundação para a Ciência e a Tecnologia, under project UIDB/50021/2020.

1 Introduction

The popularization of the Internet and its evolution over the years, along with the growing need for individuals to remain connected and communicate faster, has given rise to social networks. The adhesion to these social networks is enormous and continues to grow, making them extremely important since they are chosen by users as the main way to generate and seek knowledge, explore interests, among others. These platforms are the stage for expressing opinions, habits, tastes, and customs, representing an enormous sample and diversity of the population, as well as a great source of data, which demonstrates the immense potential for this study. The detection and analysis of emotions on this content expose the overall opinion of individuals concerning any event, allowing us to better understand how to reach a target audience, draw psychological profiles, detect preferences, trends, among many others.

Emotions are defined, in psychology, as states that reflect the evaluation of judgments of social agents, including the self and the environment, taking into consideration the objectives and beliefs of the person, which encourage and coordinate the adaptation of behavior. Emotions are generally categorized as basic or fundamental, and non-basic or complex. The latter is more difficult to classify and include *pride*, *shame*, *guilt*, *love*, among others [6]. There is no universally accepted model of emotions, but a large portion of the articles concerning the detection of emotions, and multi-class classification approaches, is based on Ekman's model that counts six basic emotions: *joy*, *sadness*, *anger*, *fear*, *disgust*, and *surprise* [4]. Another widely used model is the one described by [13], which combines adjacent pairs of basic emotions: *joy*, *trust*, *fear*, *surprise*, *sadness*, *disgust*, *anger,* and *anticipation*. Plutchik defended that emotions are extremely important for all living beings, evolutionary and that influence much of the social functioning, being the mixed emotions and the different states observed in all people and animals derivations of the basic emotions. They can be described in various languages that include subjective feelings, cognition, impulses to action, and behavior [13].

This paper attempts to identify and analyze emotions in tweets written in Portuguese, adopting a model that is derived from [13], and that contains eight basic emotions and two sentiments. Due to the scarce resources available for this language, we tested different translation approaches, either to improve an existing lexicon or to predict emotions in Portuguese tweets using an English lexicon on translated tweets.

2 Related Work

Methods for detecting sentiments emotions in a text usually require huge quantities of previously annotated data for algorithm training and tuning. For sentiment analysis, there are many annotated datasets and developed systems already with quite good results [1], but for emotion detection such resources are rare. A popular lexicon in this area is the NRC Word-Emotion Association Lexicon[1], also known as EmoLex, which associates about 14000 words in English with eight basic emotions (*joy*, *sadness*, *anger*, *fear*, *disgust*, *surprise*, *anticipation*, and *trust*) and two sentiments (*positive* and *negative*) [9, 10].

Emotion classification has recently gained significant importance as a research topic. *Emoji2emotion* is a method proposed by [14] that establishes the correspondence between the most common emojis on Twitter and online text in general, obtained using the project [12] and *Emojitracker*[2], and possible related classes, to detect feelings or emotions[3]. [18] produced a dataset of annotated data[4], relating hashtags to emotions. This relation is made using the work of [15], where the authors organize the emotions in two layers: six basic emotions and twenty-five secondary emotions, subcategories of the basic ones. Each emotion has a list of words associated with it. [18] expand these lists, managing to associate 131 words used on hashtags with the 7 basic emotions considered - the 6 mentioned above plus *gratitude*. BrainT, described in [5], uses a perceptron in a multi-class approach to classifying the emotions implicit in tweets, which proves the benefits of relating emotions to word sets and the use of bigrams, trigrams, skip-one-tetragrams, and part-of-speech (POS) tags. [17] proposed a model for predicting the intensity of emotions in tweets.

[1] http://saifmohammad.com/WebPages/NRC-Emotion-Lexicon.htm
[2] http://www.emojitracker.com/
[3] https://github.com/Aisulu/emoji2emotion
[4] http://knoesis.org/projects/emotion

Concerning the Portuguese language, [16] presents instructions for the annotation of emotions using the Plutchik's Wheel Theory, and built a collection of two corpora. Each tweet was manually annotated by two people with up to four emotions (*joy* or *sadness*, *anger* or *fear*, *trust* or *disgust*, *anticipation* or *surprise*) or *neutral*. *EmoSpell* is an extension of the *Jspell* morphological analyzer that uses a dictionary composed by the *Jspell* Portuguese dictionary and the lexical resources *EMOTAIX.PT* and *SentiLex-PT*, aiming at increasing the recognition power of *EMOTAIX.PT* [8]. [11] carry out a study to understand which approaches for subjectivity classification are better: machine learning algorithms or lexicon-based approaches. Their results suggest that machine learning algorithms may have better results than lexicon-based approaches. To predict the personality of Portuguese and English Twitter users, [7] built a profile analysis platform focusing on the Big 5 personality traits model (*extraversion*, *agreeableness*, *conscientiousness*, *neuroticism*, and *openness*). Finally, [3] reports a study exploring emojis for emotion recognition in Portuguese texts.

3 Data

The data used for this paper corresponds to about 16M geolocated tweets written in Portuguese, produced around the world. The majority of tweets are from Brazil (14995078), followed by Portugal (383874), East Timor, and PALOP countries (15369). The data consists of unstructured text, containing: abbreviations; hashtags; emojis; slang; emoticons; spelling mistakes, lexicon, syntax, and grammar. It presents a set of phenomena that make the analysis more difficult, such as some tweets incorrectly identified as Portuguese; spelling, syntax, and semantic errors; unknown abbreviations; slang; they can use emojis without taking into account their supposed meaning; among others.

Data pre-processing is a relevant stage because it has a direct implication in the extraction of the text characteristics and includes: tokenization, cleaning, and stemming or lemmatization. Tokenization ensures the decomposition of tweets into words, punctuation, emoticons, and emojis, allowing a token by token analysis. Cleaning reduces the noise in words to facilitate the comparison of them, replacing the upper case letters by lower case and all equal letters three or more times, by only one of the letters. Finally, we tested stemming and lemmatization. For Portuguese, since this is a language with many inflected words, stemmers reduce the words too much, so lemmatization turned out to be the best option. We tested *Spacy* and *Hunspell*. Taking into consideration the experiments carried out with both, and the results described by Lars Nieradzik in his blog[5], Spacy was chosen. The *PorterStemmer* and *LancasterStemmer* were used for the English language. The former creates stems by suffix stripping, while the latter uses an iterative algorithm with external rules, having a more severe approach[6]. Experiences have shown that stemming works well for English, and it is much faster than lemmatization. However, it is essential for the classification task that the lemma of each word is found, which makes the lemmatization the most appropriate process. Once again, Spacy was chosen, since, in addition to the internal functionalities it incorporates, it presents good results in structured texts.

In order to validate our classification approaches, and overcoming the lack of annotated data, two people have manually annotated 1000 random tweets, extracted from our data. Table 1 shows the agreement between them, by emotion. The low agreement observed in most of the emotions is sometimes due to the lack of examples, but also is due to the

[5] https://lars76.github.io/2018/05/08/portuguese-lemmatizers.html
[6] https://www.datacamp.com/community/tutorials/stemming-lemmatization-python

Table 1 Reliability coefficients of the manual annotations of the two persons.

	Anger	Ant.	Disgust	Fear	Joy	Sad.	Surp.	Trust
Average pairwise Percent Agreement	90.2%	94.3%	82.9%	98.9%	78.4%	82.5%	89%	88.3%
Cohen's Kappa	0.431	0.191	0.273	0.347	0.478	0.507	0.271	0.270

subjectivity of the emotions, associated mainly with the complexity and difficulty of the task. The understanding of tweets, assuming that a person can decipher them, varies from person to person and is influenced by numerous simple factors (e.g. age and personal experience). It was perceptible an increase in the difficulty of this task resulting from the type of service that Twitter offers: short messages; informal writing; in real-time for any person and part of the world. The recurrent use of jargon and slang was notable; spelling and semantics errors; abbreviations; expressions of other languages; confusing writing; misuse of caps lock and punctuation; and immense tweets written in Brazilian and some even in Spanish and English. *Irony* and *mockery* are also a point to be taken into account, since they are quite present in tweets and are sometimes not completely clear, raising the doubt of whether or not the subject is being ironic or joking and if not, what real emotions he or she is covering up. For this reason, most tweets with these indications were not classified.

4 Towards a Refined Lexicon

Our experiments use the EmoLex Lexicon, developed for the English language by [9] through manual annotation using crowdsourcing. This lexicon considers two sentiments (negative and positive) and eight basic emotions (*anger*, *disgust*, *fear*, *joy*, *sadness*, *surprise*, *trust,* and *anticipation*), assigning to each word the appropriate emotions and sentiments. It contains 14182 words, of which 2312 are associated with a positive sentiment, and 3324 with a negative. Concerning emotions, 1247 words are associated with *anger*, 839 with *anticipation*, 1058 with *disgust*, 1476 with *fear*, 689 with *joy*, 1191 with *sadness*, 534 with *surprise*, and 1231 with *trust*. A word may be associated with zero or more emotions, being that 9719 words are associated with no emotions. EmoLex was translated in November 2017 by its authors, into more than 40 languages using Google Translator[7].

We have noticed that some English words could be translated into more Portuguese words, a large number of words were incorrectly translated, and that more than 250 English words were not translated [10]. Therefore, we have tested different approaches for producing an improved and more reliable Portuguese version of the lexicon. In order to do so, we have translated the original English version again using Google Translate[8], because automatic machine translation systems have significantly improved since 2017, and also using DeepL[9], another well-known and recognized translation platform. We decided to include the translation of both platforms in the same lexicon, as sometimes they are different but equally certain, and in this way, we also covered more Portuguese words, with a total of 18920 words. This lexicon does not contain only 1429 words out of the 14182 that constitute the lexicon translated by the authors. To quantify the reliability of the translation, we attribute to the emotions present in the 9424 words with equal translations weight 2, and the emotions present in the 4748 words with divergent translations weight 1. Some of the words with different translations

[7] https://saifmohammad.com/WebPages/NRC-Emotion-Lexicon.htm
[8] https://translate.google.pt/
[9] https://www.deepl.com/translator

differ only in gender (e.g. *agravada* and *agravado*), spelling agreement (e.g. *abjecto* and *abjeto*), number (e.g. *conselho* and *conselhos*), the root of the word (e.g. *afixar* and *afixo*) and language variety (e.g. *acadêmico* and *académico*). The more than 250 words that had not originally been translated were replaced by the word itself or by a possible translation. It should be noted that about 500 words remained the same as the original NRC lexicon because it contains words in languages other than English (e.g. *amour* and *aloha*); words that are written the same way in English and Portuguese (e.g. *total* and *zebra*); words that do not have a direct translation, such as proper nouns (e.g. *Toby* and *Billy*) or foreign words (e.g. *byte* and *versus*).

Finally, we have used the MUSE [2] English-Portuguese dictionary to translate the NRC lexicon, originating a new Portuguese emotion lexicon (Lex-II) that contains 16713 distinct words. MUSE (Multilingual Unsupervised and Supervised Embeddings) is a library based on fastText, developed by researchers from Facebook to understand, represent, and classify the thousands of data entered in the platform. It offers 110 ground-truth bilingual dictionaries, including English for 44 different languages and vice versa, and word embeddings for 30 different languages, representing each word in a single vector space. We observed that different English words were translated into the same Portuguese word (e.g. *abandon* and *abandonment*). For such cases, the resulting score for each emotion was the average of the scores for such emotion, rounded to zero or one.

5 Emotion Detection in Portuguese Tweets

The data has been tested with three references so that we can better understand what the results represent. Due to the low agreement among the annotators, the reference that had the best results was the one that considers that classification is correct if at least one of the annotators agrees with such classification. We have tested six different approaches to emotion detection. Table 2 shows the best results achieved for each of the approaches, i.e., evaluating with the reference described above. Accuracy is generally high due to the unbalanced nature of the data, but precision, recall, and F-measure provide more useful information about the complexity of the task. In overall, the refined LEX-I performed better than the original EmoLex translation. Contrarily to our expectations, the refined LEX-II does not present better results than LEX-I. That may be due to the restriction made to the original MUSE dictionary and the non-inclusion of significant amounts of words from the EmoLex. Approaches 4 and 5 achieve good performance. The majority voting approach achieves the second-best overall performance, achieving the best precision of all the experiments. Our results show that *joy* and *sadness* are easier to predict than the other emotions, not only because they are the most expressed on Twitter, but also because they are the ones with lower disagreement, as seen in Table 1. *Fear* and *anticipation* are the most difficult ones, not only because they are the less common in tweets, but also because these emotions can have different perceptions from person to person. Sometimes one uses the word *fear* wrongly to express similar emotions of lesser intensity (e.g. insecurity) or we say that we anticipate something, but in reality, we only deduce it taking into account something we have seen or know.

Finally, Table 3 presents the agreement between the first 5 approaches, revealing that, in general, the classifiers show low/moderate agreement values, an expected result due to the small sample in terms of variety among emotions. The agreement is lower for *trust, fear,* and *surprise*. The different experiences show quite similar values, as is evidenced by the agreement between them, this is because they are both based on the same lexicon.

Table 2 Evaluation performance of metrics of the experiences based on reference.

	Anger	Ant.	Disgust	Fear	Joy	Sad.	Surp.	Trust	macro average
Approach 1: Original Portuguese translation of EmoLex									
Accuracy	0.887	0.794	0.894	0.816	0.772	0.805	0.854	0.802	0.828
Precision	0.333	0.103	0.394	0.027	0.522	0.491	0.186	0.218	0.284
Recall	0.662	0.821	0.633	0.714	0.460	0.560	0.604	0.869	0.665
F-measure	0.443	0.183	0.485	0.052	0.489	0.523	0.284	0.349	0.351
Approach 2: Refined LEX-I									
Accuracy	0.872	0.723	0.878	0.782	0.766	0.794	0.844	0.687	0.793
Precision	0.327	0.093	0.377	0.031	0.521	0.478	0.208	0.206	0.280
Recall	0.712	0.875	0.640	0.875	0.547	0.605	0.655	0.952	0.733
F-measure	0.448	0.168	0.474	0.060	0.534	0.534	0.316	0.338	0.359
Approach 3: Refined LEX-II									
Accuracy	0.838	0.736	0.883	0.754	0.754	0.787	0.852	0.750	0.794
Precision	0.271	0.094	0.389	0.024	0.509	0.489	0.164	0.203	0.268
Recall	0.671	0.844	0.690	0.750	0.594	0.662	0.543	0.912	0.708
F-measure	0.386	0.170	0.498	0.047	0.548	0.563	0.253	0.332	0.350
Approach 4: English EmoLex, and translate tweets using Google Translate									
Accuracy	0.869	0.748	0.902	0.811	0.777	0.781	0.855	0.763	0.813
Precision	0.359	0.092	0.446	0.026	0.546	0.436	0.167	0.212	0.286
Recall	0.735	0.862	0.690	0.714	0.622	0.516	0.556	0.912	0.701
F-measure	0.482	0.166	0.542	0.050	0.582	0.472	0.256	0.343	0.362
Approach 5: English EmoLex, and translate tweets using DeepL									
Accuracy	0.878	0.757	0.909	0.823	0.777	0.788	0.865	0.757	0.819
Precision	0.322	0.091	0.444	0.033	0.541	0.437	0.194	0.199	0.283
Recall	0.671	0.857	0.667	0.750	0.588	0.484	0.596	0.908	0.690
F-measure	0.435	0.165	0.533	0.063	0.564	0.459	0.293	0.327	0.355
Approach 6: Fusion - Combine all the previous classifiers, and use majority voting for classification									
Accuracy	0.880	0.766	0.902	0.809	0.780	0.798	0.877	0.770	0.823
Precision	0.331	0.095	0.417	0.026	0.543	0.480	0.214	0.208	0.289
Recall	0.653	0.828	0.641	0.714	0.574	0.560	0.583	0.922	0.684
F-measure	0.439	0.170	0.505	0.050	0.558	0.517	0.313	0.339	0.361

As previously mentioned, an automatic emotion classifier of tweets was developed that constitutes our initial approach for identifying the emotions present in the text. Our classifier detected at least one emotion in about 57% of the tweets, which corresponds to about 9 million tweets, but a large number of tweets were classified with more than one emotion. The first chart from Figure 1 shows the number of tweets containing each one of the emotions being considered, revealing that the prevailing emotion is *sadness* (3.8 million tweets), followed by *anticipation* (3.3 million tweets), and that *surprise* is the least prevalent emotion (2.2 million tweets). The second chart shows the number of tweets by the number of emotions present on them. In general, people do not experience several emotions at the same time, so as expected there are more tweets with fewer emotions. When people express their written opinions/ideas, they tend to think more about the subject and, depending on the complexity of it, to express more than one emotion. However, on Twitter, many people post what they feel at the moment without thinking too much and that is why there is also a large number of tweets with an emotion. About 30% of classified tweets have four emotions and about 19% one emotion. We have found that the most frequent combinations by number of emotions

Table 3 Reliability coefficients for the classification of 1000 random tweets.

	Anger	Ant.	Disgust	Fear	Joy	Sad.	Surp.	Trust
Average pairwise Percent Agreement	89.8%	86.4%	91.1%	86.8%	87.3%	85.6%	89.3%	82.1%
Cohen's Kappa	0.621	0.655	0.617	0.599	0.671	0.601	0.591	0.580

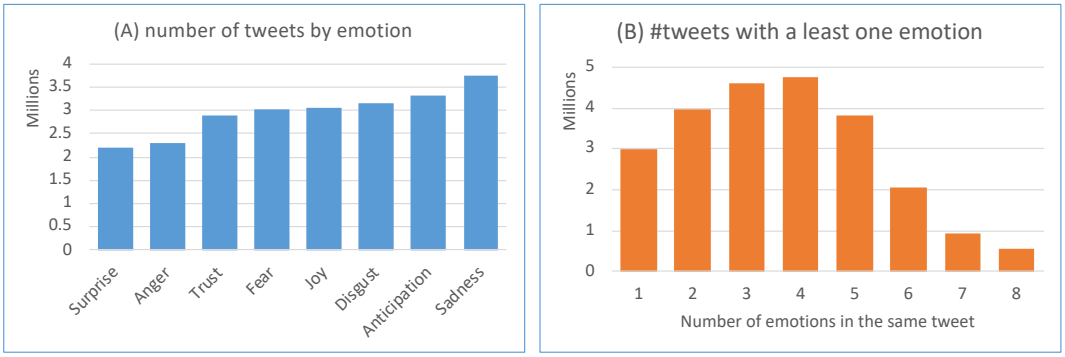


Figure 1 Number of tweets a) per emotion; b) marked with one or more emotions.

are *joy* and *trust*; *anticipation, joy,* and *trust*; *anger, disgust, fear,* and *sadness; anticipation, fear, joy, surprise,* and *trust*; *anticipation, fear, joy, sadness, surprise,* and *trust*; *anger, anticipation, fear, joy, sadness, surprise,* and *trust.*

Our classifier succeeded in assigning at least one emotion to 118046 tweets from Brazil, 6435 tweets from Portugal, 148 tweets from PALOP countries, and another 3459 tweets to other countries. Taking into account the annotated tweets with at least one emotion, Figure 2 shows the occurrence of each emotion per country. Portugal and Brazil show the same tendency for all emotions. *Anger* and *disgust* are the least predominant emotions in all cases. *Sadness* is the most dominant emotion for Portugal and Brazil, but *sadness* is surpassed by *joy* in PALOP countries and by *anticipation* in other countries.

6 Conclusions and Future Work

We have described our efforts to improve an existing emotion lexicon for Portuguese and presented several approaches for emotion classification using Portuguese tweets, based on an emotion lexicon. In order to overcome the lack of emotion resources for Portuguese, we have compared two methods: translating a lexicon into Portuguese vs. translating the tweets into English. Experiences have shown that the two methods achieve similar values, and one may be preferred over the other depending on the problem. We have manually annotated 1000

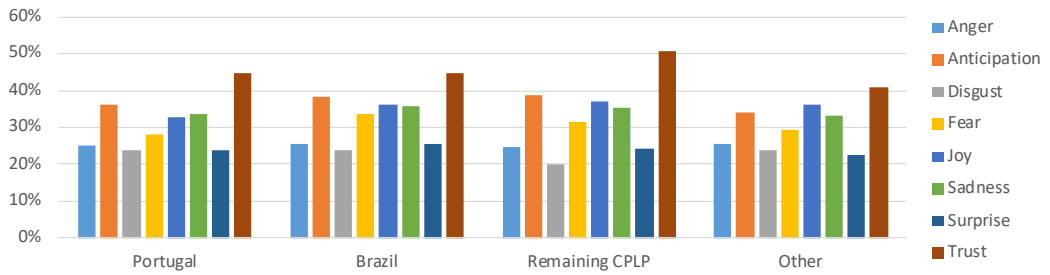


Figure 2 Percentage of tweets associated with a given emotion per country, using approach 1.

tweets that were used to validate our experiments. Our findings suggest that some emotions are easier to detect than others. *Joy*, *sad*, *anger,* and *disgust* were the most successfully predicted emotions, but the task is difficult even for humans. In the future, we plan to train a classifier in a supervised way, to improve the identification of emotions in tweets and explore emojis and emoticons as additional cues, since they are both widely used on social networks as a way for people to express what they are feeling.

References

1 Sattam Almatarneh and Pablo Gamallo. A lexicon based method to search for extreme opinions. *PLOS ONE*, 13(5):1–19, 2018. doi:10.1371/journal.pone.0197816.

2 Alexis Conneau, Guillaume Lample, Marc'Aurelio Ranzato, Ludovic Denoyer, and Hervé Jégou. Word translation without parallel data. *arXiv preprint*, 2017. arXiv:1710.04087.

3 Luis Duarte, Luís Macedo, and Hugo Gonçalo Oliveira. Exploring emojis for emotion recognition in portuguese text. In Paulo Moura Oliveira, Paulo Novais, and Luís Paulo Reis, editors, *Progress in Artificial Intelligence*, pages 719–730, Cham, 2019. Springer International.

4 Paul Ekman. An argument for basic emotions. *Cognition and Emotion*, 6(3-4):169–200, 1992.

5 Vachagan Gratian and Marina Haid. BrainT at IEST 2018: Fine-tuning multiclass perceptron for implicit emotion classification. In *9th Workshop on Comp. Approaches to Subjectivity, Sentiment and Social Media Analysis*, pages 243–247, Brussels, Belgium, 2018. ACL.

6 Eva Hudlicka. Guidelines for Designing Computational Models of Emotions. *International Journal of Synthetic Emotions*, 2(1):26–79, January 2011. doi:10.4018/jse.2011010103.

7 A. Jusupova, F. Batista, and R. Ribeiro. Characterizing the Personality of Twitter Users based on their Timeline Information. In *Proc. of 16ª CAPSI*, pages 292–299, 2016.

8 Maria Inês Maia and José Paulo Leal. An Emotional Word Analyzer for Portuguese. In *SLATE 2017*, volume 56 of *OpenAccess Series in Informatics (OASIcs)*, pages 17:1–17:14. Schloss Dagstuhl, 2017. doi:10.4230/OASIcs.SLATE.2017.17.

9 S. Mohammad and P. Turney. Emotions evoked by common words and phrases: Using mechanical turk to create an emotion lexicon. In *Proc. of the Workshop on Comp. Approaches to Analysis and Generation of Emotion in Text*, pages 26–34, LA, California, 2010.

10 Saif M Mohammad and Peter D Turney. Crowdsourcing a Word-Emotion Association Lexicon. *Computational Intelligence: an International Journal*, 29(3):436–465, 2013.

11 S. Moraes, A. Santos, M. Redecker, R. Machado, and F. Meneguzzi. Comparing approaches to subjectivity classification: A study on portuguese tweets. In *Computational Processing of the Portuguese Language*, volume 9727, pages 86–94. Springer International, 2016.

12 Petra Kralj Novak, Jasmina Smailović, Borut Sluban, and Igor Mozetič. Sentiment of emojis. *PLOS ONE*, 10(12), 2015. doi:10.1371/journal.pone.0144296.

13 Robert Plutchik. A psychoevolutionary theory of emotions. *Social Science Information*, 21(4-5):529–553, 1982.

14 Aisulu Rakhmetullina, Dietrich Trautmann, and Georg Groh. Distant supervision for emotion classification task using emoji2emotion. In *Proc. of Emoji 2018*, volume 2130, 2018.

15 P. Shaver, J. Schwartz, D. Kirson, and C. O'Connor. Emotion knowledge: Further exploration of a prototype approach. *Journal of Personality and Social Psychology*, 52(6):1061–1086, 1987.

16 F. Silva, N. Roman, and A. Carvalho. Building an emotionally annotated corpus of investor tweets. Technical Report IC-18-05, Instituto de Computação, Univ. Estad. Campinas, 2018.

17 Min Wang and Xiaobing Zhou. Yuan at SemEval-2018 Task 1: Tweets Emotion Intensity Prediction using Ensemble Recurrent Neural Network. *Proc. of the 12th Int. Workshop on Semantic Evaluation*, pages 205–209, 2018. doi:10.18653/v1/S18-1031.

18 Wenbo Wang, Lu Chen, Krishnaprasad Thirunarayan, and Amit P. Sheth. Harnessing twitter "big data" for automatic emotion identification. *ASE/IEEE Int. Conf. on Social Computing, SocialCom/PASSAT 2012*, pages 587–592, 2012. doi:10.1109/SocialCom-PASSAT.2012.119.

Integrating Multi-Source Data into HandSpy

Hristo Valkanov
Faculty of Sciences, University of Porto, Portugal
up200803561@fc.up.pt

José Paulo Leal
CRACS & INESC Tec LA, Porto, Portugal
Faculty of Sciences, University of Porto, Portugal
jpleal@fc.up.pt

Abstract

To study how emotions affect people in expressive writing, scientists require tools to aid them in their research. The researchers at M-BW use an Experiment Management System, called HandSpy to store and analyze the hand-written productions of participants. The input is stored as digital ink and then displayed on a web-based interface.

To assist the project, HandSpy integrates with new sources of information to help researchers visualize the link between psychophysiological data and written input. The newly acquired data is synchronized with the existing burst-pause interval model and represented on the user interface of the platform together with the already existing information.

2012 ACM Subject Classification Applied computing → Psychology; General and reference → Experimentation; Human-centered computing → Visualization

Keywords and phrases HandSpy, emotion, handwriting, psychology, psychophysiological, data, ems

Digital Object Identifier 10.4230/OASIcs.SLATE.2020.13

Category Short Paper

Funding *José Paulo Leal*: Fundação para a Ciência e a Tecnologia (FCT), within project UIDB/50014/2020.

Acknowledgements We want to thank the researchers and investigators at M-BW and José Paiva for the input and assistance on the project.

1 Introduction

The link between emotions and psychophysiological events is one of the means used in the psychological analysis of behavior. Based on this theory, the scientists in Mind-Body Interactions in Writing (M-BW) Project are studying the cognitive processes of writing[4]. Experimental Management Systems (EMS) is software oriented towards aiding researchers in their studies. One such EMS, used by the project is HandSpy, which allows scientists to visualize data together with a representation of hand-writing. The input is displayed based on intervals of interest, called *bursts*, and the pauses between them.

The project studies the real-time psychophysiological and linguistic markers of expressive writing. Expressive writing is a particular form of writing in which the study participant engages in narrating a personal, deeply charged, emotional event. Despite the numerous healing effects, the mechanism through which expressive writing operates is still poorly understood. To store and analyze the digital ink representation of the strokes, M-BW uses HandSpy, which is collected using smartpens and microdotted paper.[4] The information is stored in XML format, called InkML which is the recommended standard for storing digital ink traces.[6]

To further aid the research, improvements were implemented, so that the platform supports the functionalities required by the project. The new features are based on the collection and representation of psychophysiological indicators from various external sources and centralizing them in a cohesive way inside HandSpy. The newly obtained data will be stored and adapted to the Pause-Burst model and visually represented on the current interface implementation of written text. The implementation of those improvements is the focus of this paper.

2 State of the Art

Without the use of proper tools, trivial tasks can become time-consuming and can hinder the workflow of scientific experiments. While many commercial solutions that can partially manage experiment data exist, they are oriented, mostly, towards business use. They are useful for their mainstream usage but are not entirely adequate for the scientific experimental environment. Those solutions also require a significant monetary and human resource investment, not to mention the proprietary restrictions, which limit their distribution within the research community.[3]

2.1 Experimental Management Systems

One of the problems faced by the experimental sciences is the lack of proper software. Many of the studies require software tailored, specifically, to satisfy the requirements of the area of study, and the researchers involved in it. The need for such tools led to the creation of software, called Experimental Management Systems (EMS), that targeted scientific research environment, specifically. There have been many attempts to identify what are the proper requirements for an EMS. They have been narrowed down to the following requirements[3]:

- **Abstraction from the users** – The system should be as abstract as possible and, ideally, require no programming knowledge from the users.
- **System Integration** – The integration should be handled only at the highest level by the user. All the implementation details should be, previously, provided by the EMS programmers and be readily available to the researchers.
- **Data Integration** – The data should be integrated, in a manner that will allow for it to be handled, without requiring any knowledge of the source.
- **Remote Collaboration Facilities** – The platform should be available for access remotely. Web technologies are a natural choice for this goal.
- **Advanced Data Type Management** – Should handle on its own all the types of data required by the experiment.
- **Intelligent and adaptive UI** – The interface should be intuitive and without as little unneeded information as possible.

2.2 Existing Solutions

While the requirements for building an EMS seem simple, they are hard to fulfill. Currently, there is no commercial solution that stores the digital ink oriented towards studying handwriting, which can substitute an EMS. Those solutions that can perform some tasks or execute a part of the process exist, but none of them serves as a complete EMS. Other downsides to them include: not being directed towards managing scientific experiments, their cost, among others, which limit their adoption. There are other EMS that exists for

handwriting analysis, but most of them are not suited for the research that HandSpy assists, namely M-BW. Therefore they were not picked as a target for the improvements in question. The specific shortcomings of each of them are explained below.

2.2.1 Eye and Pen

Eye and Pen is a system that is mainly, but not exclusively, used in the context of handwriting studies. It is oriented towards the analysis of both text and drawings. The platform is designed to record, synchronously, the gaze, and the process of handwriting. For the writing part, the researchers use a digitizing tablet, to obtain the location and state of the pen. An eye-tracking system records the writer's gaze while performing tasks. The main purpose of the system is to study the synchronization between eye and pen movements writing periods. The devices used are focused on allowing users to participate in the investigation, without hindering the activity or increasing their cognitive load.[1] Its shortcomings, compared to HandSpy, due to the usage of a tablet, which is more invasive and less scalable, than the HandSpy approach. Eye and Pen also lacks support for the psychophysiological data, required by M-BW.

2.2.2 Dictus

Dictus is a software that combines stimulus presentation, such as images, or sounds, and movement analysis. The data is gathered by a module that acquires the measures from a hand-writing movement using only a Wacom tablet. This approach has the downside of collecting information form a single source. The main advantage of this method is that the information gathering process is flexible. Because of this, the system is usable in any environment without the need for more sophisticated machinery. This advantage has many upsides. For example, it can be used in both school rooms and laboratories. Another advantage is that it does not involve any obstructing equipment which allows the experiments to be carried with the target population while maintaining the feeling of writing, as they usually do in their everyday life.[2] It shares the same disadvantages as Eye and Pen while being even more restricted due to the limitations of the tablet brand.

2.2.3 HandSpy

The HandSpy platform is a web-based EMS that excels in handwriting research studies. It supports large amounts of data sets. It is designed as a web platform to support remote use and to eliminate the need for installation on multiple interfaces and accommodate collaborative and remote work. It currently supports collections of data, related to the handwriting process such as the write/pause intervals and the distance covered. HandSpy supports input from smartpens to provide data for the studies, which is less intrusive than the input devices used by the other EMS, which is crucial since obstructions during writing affect the collected data.

The input is collected and stored in the server, and available on request. HandSpy also supports annotations such as the start and end of the burst, with a blue letter "p" and a red letter "q", respectively, as shown in Figure 1, and storing additional information, such as the number of words in the interval. [5]

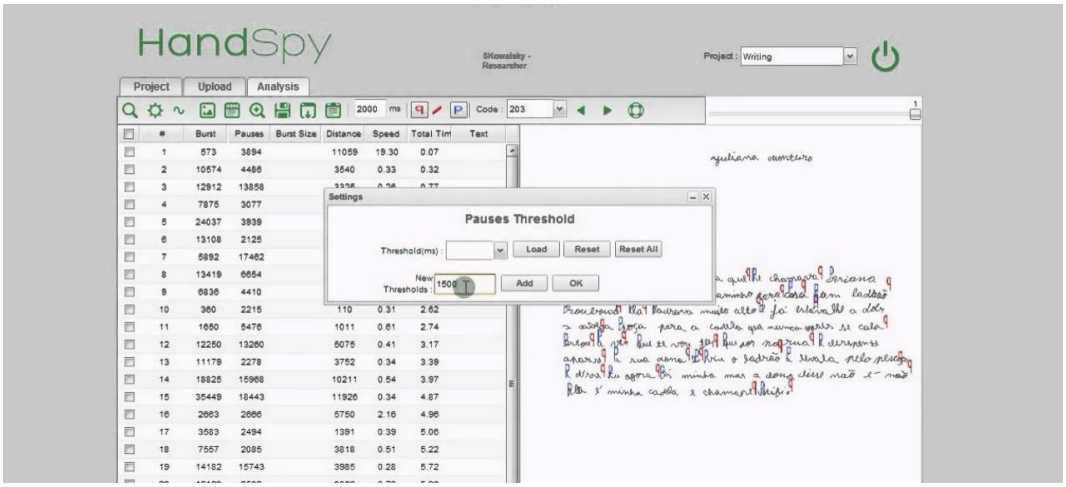


Figure 1 HandSpy showing the annotations for the start and end of the burst.

3 Improvements to HandSpy

HandSpy consists of two components: a server and a client. The server obtains data, in the form of digital ink, from the smartpen and pre-processes it. The information is made available, upon request, to the local client, which allows the users to further work with it. The client is used to filter the data set sent form the server and display it on the UI.

Until now, HandSpy had only a few forms of representation of research related data. They used to be limited to 3 markers: a trace that runs through the selected area; the beginning of a burst; and the end of a burst. This interface is insufficient for the ongoing project, and improvements were required so that HandSpy could assist the M-BW project.

The changes to the current process include the integration with new data sources and the changes to the UI, which will allow us to show more annotations, without cluttering the front-end. HandSpy is aggregating the data from the new sources, based on the previously existing pause-burst intervals.

To display the data, the UI is adapted to create overlays over the points of interest. They are calculated based on the traces of the digital ink, already stored in HandSpy. The overlays are shaped as bounding boxes and mark the separate lines of text.

3.1 Server

The server is divided into three layers, Core, Commands, and Protocol. Due to the architecture of the software, the only change required was in the last of those components. The protocol module handles the request processing and the response generation for the different controllers. Most of the business logic is stored here, inside the Protocol class. This class obtains the connection information from the core, fetches data from the database, parses the requests, generates responses, calculates intervals, and so on.

HandSpy stores the data related to the written productions. They are stored as trace entries, using InkML format, inside the database. Each trace contains a set of points and represents the movement of the pen, starting from the moment the pen touches the paper and ends when the pen leaves the paper. It consists of a set of points that describe the trajectory of the movement. Each “point” holds three values, the InkML bi-dimensional coordinates for X and Y, combined with a timestamp of when the reading was measured.

This list of coordinates is used by the platform to calculate the distance covered by the smartpen, while writing. Meanwhile, the timestamps define the intervals of interest on which the research calculations are based.

3.1.1 Extraction of Intervals

While the existing annotations, such as the beginning and the end of the intervals, served their purpose at the time, the platform lacks specific characteristics, namely the integration with additional sources of data. This improvement is significant because it is one of the requirements for a fully functioning experiment data management system[3]. To support this feature, a dedicated entity, which represents the intervals, was created. This separation was, later used, as a baseline for the implementation of all future metrics.

The code which performed the calculation of the existing intervals was extracted from the protocol class. The newly created modules, which hold the new structure, consists of separate "classes" that represent the different elements of the studies, namely pauses and bursts. An interval contains the logic required to extract the currently existing metrics and all the points for this calculation. Burst objects are identical to the previously calculated pause-bursts, and pauses represent the interval between them.

3.1.2 Lines

The previous version of HandSpy (2.3) displays only the beginning and the end of an interval. The visual representation had to change due to the new functionalities implemented for the next version of the platform. The choice for the visual representation of lines is by using rectangle shaped overlays that encapsulate the points that the interval contains. The bounding box calculation of the lines is performed by grouping the points first into a histogram and then again by using a density threshold and a tolerance range.

The method of calculation is simple since the visual representation does not require very high precision. For each burst interval, the InkML coordinate points are separated by the vertical coordinate value to obtain the number of points that share the same horizontal line. The density threshold is, then, determined by applying a sensitivity percentage to the group with the highest point count. The number obtained, determines if a horizontal line will or will not be part of the marker system. The groups are, then, compared to the threshold, and if they are above it, they are combined with the neighboring lines, until two consecutive lines are below the limit. When this happens, the last observed line is marked finalized, and the bounding box calculation occurs. The corners of the rectangle, which defines it, assume the maximum and minimum values of the X and Y values.

The entire line calculation process occurs on the server, and solely a summary of the result is passed to the client in the end. This separation exists to reduce the server's load by caching the line calculation for later use, thus avoiding unnecessary recalculation.

3.1.3 Metrics

After obtaining the lines' bounding boxes, HandSpy has all the requirements to create an overlay on top of the representation of the strokes, which will represent an interval of interest. What follows is to calculate and aggregate the newly requested data into indicators, called metrics. To obtain a metric entity, we calculate various indicators based on the universe of entries, grouped in the same time windows as the burst interval. Those entities contain data for both burst and global levels for the selected performance indicator. The calculation of every metric is implemented in a dedicated customized class, tailored to suit the necessities of the researchers.

As proof of concept, the first metric to be implemented was the user writing speed. To calculate the value, HandSpy uses the InkML coordinates of neighboring points and the timestamp of when the entry was stored. To obtain the speed metric, the calculator class iterates over the bursts and, then, calculates the aggregations on burst level using the Euclidean distance between two consecutive readings, divided by the difference in the timestamps. The result of the calculation is, then, stored inside a "double" data type value, which represents the speed in mm/ms.

3.2 Client

To allow the interaction between the user and the platform, a new section was added to the front-end component of HandSpy. The new section allows scientists to create a list of metrics that are to be requested from the server. It also allows them to select criteria, that will be later used to determine if it is to be highlighted on the representation of the strokes. The highlight is used to mark important intervals, by paining a transparent overlay over the target area as shown in Figure 2

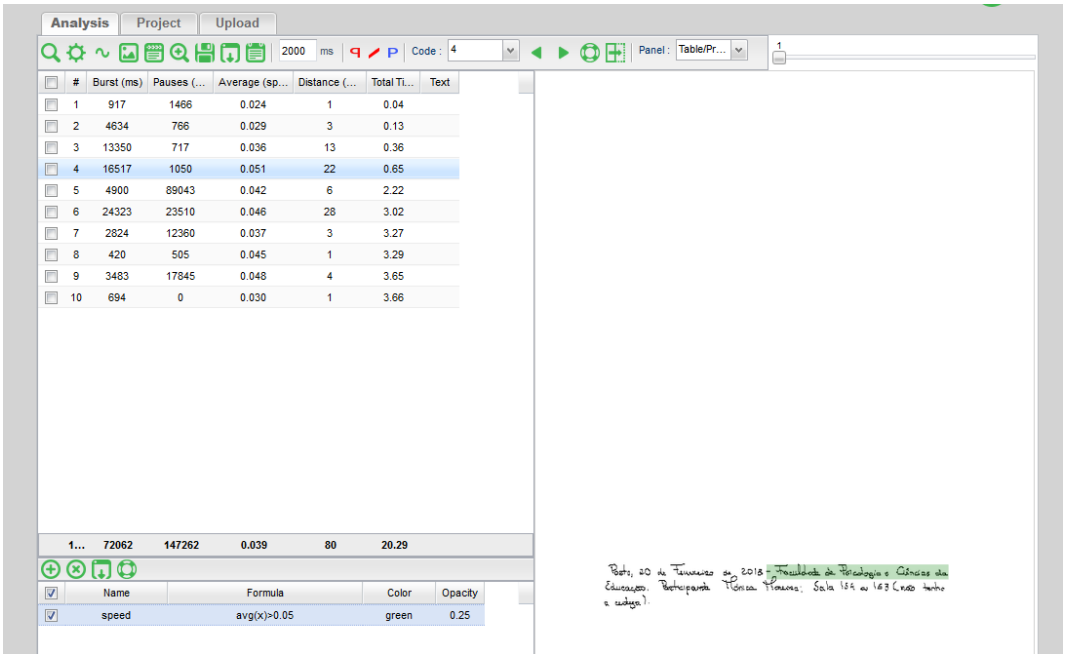


Figure 2 Example of marking the interval with speed above 0.05mm/ms in green.

The criteria for the markers consist of 3 parts: a metric selection; a pseudo programming language formula, which includes the metrics calculated by the server; and overlay options. Specifying the target metrics is as simple as writing its name in the first column. Users have to write the formula, to select the criteria for the points of interest. It has to be satisfied, and the platform will create an overlay over the corresponding stokes.

The rest of the formula represents the aesthetic part of the marking over the lines of the matched intervals. The color column determines the filling of the overlay. It can be filled-in, with either a hexadecimal code or a color name in English. For example, the color can be either "green" or "red" or "ff0000". Since we need to be able to see the text under the overlay, we also need to select an opacity value that will be applied on the line, unless we want to hide the written input under it. The opacity should be a floating-point number between 0 and 1, with recommended values under 0.25, to keep the text, underneath, visible.

4 Future Work

With the new feature working, HandSpy is fully prepared to receive experimental data from multiple sources. This development can assist the project in the future, but it's insufficient in its current state. The existing implementation is not particularly user friendly, thus breaking the last requirement for a good EMS - Intelligent and adaptive UI [3]. The future work on the platform is going to focus on UI and usability improvements that will make HandSpy usable without training.

One of the changes will be on the metric criteria definition. This segment of the new feature was considered complex by the users. An improvement of the usability is needed so that the scientists will not be required to learn an expression syntax, to be able to use the feature. Besides this, the current information on how the metric aggregations lack proper explanations.

Another improvement discussed with the members of the M-BW project is the inclusion of relative metrics. The current criteria for them can be compared only by using static values. To overturn this, besides the currently available method, we will implement relative metrics, such as quartiles, and percentages of the global values. This change will give them more flexibility in the marking and analysis of points of interest, and will also save the users the time to calculate manually the values to which they wish to compare.

Another change requested that will be useful to the project is the HandSpy front-end separates pauses and bursts so that the users can observe psychophysiological indicators even when no points exist. This change will require the separation of the metrics on both server and client-side.

The current work introduces only some metrics to HandSpy. It is planned for more of them to be implemented in the future. The first ones will be the heart rate and the conductivity of the skin. Those metrics will be collected and stored on the HandSpy side and will be used to aid further the research done by M-BW.

The visual representation of digital ink is also going to receive an overhaul. Some of the discussed changes include the color and opacity, based on the values of the metrics and overlay representation for colorblind users.

The new minor version of HandSpy is developed together with the scientists from the M-BW project. They will validate the usability of the features and assure that the newly implemented features satisfy the needs of the project.

References

1 Denis Alamargot, David Chesnet, Christophe Dansac, and Christine Ros. Eye and pen: A new device for studying reading during writing. *Behavior research methods*, 38:287–99, June 2006. doi:10.3758/BF03192780.

2 Sonia Kandel Eric Guinet. Ductus: a software package for the study of handwriting production. *Behavior Research Methods*, 2010. doi:10.3758/BRM.42.1.326.

3 R Jakobovits, S Soderland, Ricky Taira, and James Brinkley. Requirements of a web-based experiment management system. *Proceedings / AMIA 2000 Annual Symposium. AMIA Symposium*, February 2000.

4 © 2020 M-BW. Mind-Body Interactions in Writing (M-BW) psychophysiological and linguistic synchronous correlates of expressive writing. http://m-bw.up.pt/index.php/m-bw. Accessed: 2020-05-18.

5 Carlos Monteiro and José Paulo Leal. HandSpy – a system to manage experiments on cognitive processes in writing. In Alberto Simões, Ricardo Queirós, and Daniela da Cruz, editors, *1st Symposium on Languages, Applications and Technologies*, volume 21 of *OpenAccess Series in Informatics (OASIcs)*, pages 123–132, Dagstuhl, Germany, 2012. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/OASIcs.SLATE.2012.123.

6 © 2011 W3C. InkML ink markup language. URL: https://www.w3.org/TR/2011/REC-InkML-20110920/. Accessed: 2020-05-18.

Yet Another Programming Exercises Interoperability Language

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

Ricardo Queirós
CRACS – INESC, LA, Porto, Portugal
uniMAD – ESMAD, Polytechnic of Porto, Portugal
http://www.ricardoqueiros.com
ricardoqueiros@esmad.ipp.pt

José Paulo Leal
CRACS – INESC, LA, Porto, Portugal
DCC – FCUP, Porto, Portugal
https://www.dcc.fc.up.pt/~zp
zp@dcc.fc.up.pt

Jakub Swacha
University of Szczecin, Poland
jakub.swacha@usz.edu.pl

Abstract

This paper introduces Yet Another Programming Exercises Interoperability Language (YAPExIL), a JSON format that aims to: (1) support several kinds of programming exercises behind traditional blank sheet activities; (2) capitalize on expressiveness and interoperability to constitute a strong candidate to standard open programming exercises format. To this end, it builds upon an existing open format named PExIL, by mitigating its weaknesses and extending its support for a handful of exercise types. YAPExIL is published as an open format, independent from any commercial vendor, and supported with dedicated open-source software.

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, programming learning

Digital Object Identifier 10.4230/OASIcs.SLATE.2020.14

Category Short Paper

Funding This paper is based on the work done within the Framework for Gamified Programming Education project supported by the European Union's Erasmus Plus programme (agreement no. 2018-1-PL01-KA203-050803).

1 Introduction

Learning programming relies on practicing. Practicing in this domain boils down to solve exercises. Regardless of the context (curricular or competitive learning), several tools such as contest management systems, evaluation engines, online judges, repositories of learning objects, and authoring tools use a different panoply of formats in order to formalize exercises. Though this approach remedies individual needs, the lack of a common format hinders interoperability and weakens the development and sharing of exercises among different

educational institutions. Moreover, the existence of a common data format will increase innovation in programming education with a high practical impact, as it will help to save a lot of instructors' time that they would otherwise have to spend on defining new exercises or recasting existing ones themselves.

At the same time, all of these programming exercise formats focus on describing traditional programming exercises, such as blank sheet exercises, where the student is challenged to solve, from scratch, a presented problem statement. In fact, up to this date, there are no open formats that explore the fostering of new competencies such as understanding code developed by others and debugging. To enhance these skills, new types of exercises (e.g., solution improvement, bug fix, gap filling, block sorting, and spot the bug) can be defined and applied at different phases of a student's learning path. This diversity can promote involvement and dispel the tedium of routine associated with solving exercises of the same type. To the best of our knowledge, there are several formats for defining programming exercises but none of them supports all the different types of programming exercises mentioned.

This paper introduces a new format developed for describing programming exercises - the Yet Another Programming Exercises Interoperability Language (YAPExIL). This format is partially based in the XML dialect PExIL [4], but (1) it is a JSON format instead of XML, (2) transfers the complex logic of automatic test generation to a script provided by the author, and (3) supports different types of programming exercises.

The remainder of this paper is organized as follows. Section 2 surveys the existing formats, highlighting both their differences and similar features. In Section 3, YAPExIL is introduced as a new programming exercises format and four facets are presented. Then, Section 4 validates the format expressiveness and coverage. In the former, the Verhoeff model [6] is used to validate the expressiveness of YAPExIL. In the later, the YAPExIL coverage for new types of exercises is shown. Finally, Section 5 summarizes the main contributions of this research and discusses plans for future work.

2 Programming Exercises Format

The increasing popularity of programming encourages its practice in several contexts. In formal learning, teachers use learning environments and automatic evaluators to stimulate the practice of programming. In competitive learning, students participate in programming contests worldwide resulting in the creation of several contest management systems and online judges. The interoperability between these types of systems is becoming a topic of interest in the scientific community. To address these interoperability issues, several programming exercise formats were developed in the last decades.

In 2012, a survey [5] synthesized those formats according to the Verhoeff model [6]. This model organizes the programming exercise data into five facets: (1) Textual information - programming task human-readable texts; (2) Data files - source files and test data; (3) Configuration and recommendation parameters - resource limits; (4) Tools - generic and task-specific tools; (5) Metadata - data to foster exercise discovery among systems. For each facet of the model, a specific set of features was analyzed and verified the support of each format. In that time, the study confirmed the disparity of programming exercise formats and the lack or weak support for most of the Verhoeff model facets. Moreover, the study concludes that this heterogeneity hinders the interoperability among the typical systems found on the automatic evaluation of exercises. To remedy these issues, two attempts to harmonize the various specifications were developed: a new format [4] and a service for exercises formats conversion [5].

Since then, new formats were proposed to formalize exercises. In this section, we present a new survey that aims to compare existent programming exercise formats based on their expressiveness. Based on a comprehensive survey done of systems that store and manipulate programming exercises, we found about 10 different formats. Since some of them lack a published description we concentrated on 7 formats, namely: (1) Free Problem Set (FPS); (2) Kattis problem package format; (3) DOM Judge format; (4) Programming Exercise Markup Language (PEML); (5) the language for Specification of Interactive Programming Exercises (SIPE); (6) Mooshak Exchange Format (MEF); and (7) Programming Exercises Interoperability Language (PExIL).

The study follows the same logic as its predecessor, but with the following changes:

- Expressiveness model - the Verhoeff model is extended with a new facet that analyzes the support of new types of exercises. For this study eight types of programming exercises were tackled, namely: blank sheet, extension, improvement, bug fix, fill in gaps, spot bug, sort blocks, and multiple choice.
- Data formats - given that several years have passed since the last study, CATS and Peach Exchange formats are removed and new formats are added (Kattis, DOM Judge, PEML, and SIPE).

Each format is evaluated for its level of coverage of all features of each facet. The evaluation values range from 1 – low support to 5 – full support. Then, all the values are added and a final percentage is presented corresponding to the coverage global level of each format and facet, based on the extended Verhoeff model.

Table 1 gathers the current coverage level of the selected programming exercises formats.

Table 1 Coverage level comparison on programming exercises data formats.

Facets/Formats	FPS	KTS	DOMJ	PEML	SIPE	MEF	PExIL	**TOTAL**
1. Textual	2	3	3	3	3	4	5	66%
2. Data files	3	2	3	3	3	3	5	63%
3. Config	2	2	1	1	3	3	5	49%
4. Tools	1	3	3	2	2	3	5	54%
5. Metadata	2	2	3	3	3	3	5	60%
6. Ex. types	1	1	1	1	1	2	1	23%
TOTAL	37%	43%	47%	43%	50%	60%	87%	

Based on these values, we can see that, regarding format coverage, PExIL assumes a prominent role with 87% of facets' coverage rate based on the Verhoeff extended model. This is mainly because, despite being a format created eight years ago, it is still one of the most recent formats (excluding the PEML format). All the other formats cover more or less half of the facets.

Regarding facets coverage, one can conclude that, on the one hand, textual (66%), data files (63%), and metadata (60%) facets are the most covered. On the other hand, the support for different exercise types, beyond the blank sheet type (typical format), is scarce.

3 YAPExIL

Yet **A**nother **P**rogramming **Ex**ercises **I**nteroperability **L**anguage (YAPExIL) is a language for describing programming exercise packages, partially based in the XML dialect PExIL (**P**rogramming **Ex**ercises **I**nteroperability **L**anguage) [4]. In comparison to PExIL, YAPExIL (1) is formalized through a JSON Schema instead of a XML Schema, (2) removes complex

logic for automatic test generation while still supporting it through scripts, (3) supports different types of programming exercises and (4) adds support for a number of assets (e.g., instructions for authors, feedback generators, and platform information).

YAPExIL aims to consolidate all the data required in the programming exercise life-cycle, including support for seven types of programming exercises:

- `BLANK_SHEET` provides a blank sheet for the student to write her solution source code from scratch;
- `EXTENSION` presents a partially finished solution source code (the provided parts are not subject to change by the student) for the student to complete;
- `IMPROVEMENT` provides correct initial source code that does not yet achieve all the goals specified in the exercise specification (e.g., optimize a solution by removing loops), so the student has to modify it to solve the exercise;
- `BUG_FIX` gives a solution with some bugs (and, possibly, failed tests) to foster the student to find the right code;
- `FILL_IN_GAPS` provides code with missing parts and asks students to fill them with the right code;
- `SPOT_BUG` provides code with bugs and asks students to merely indicate the location of the bugs;
- `SORT_BLOCKS` breaks a solution into several blocks of code, mixes them, and asks students to sort them.

To this end, the YAPExIL JSON Schema can be divided into four separate facets: **metadata**, which contains simple properties providing information about the exercise; **presentation**, which relates to what is presented to the student; **assessment**, which encompass what is used in the evaluation phase; and **tools**, which includes any additional tools that the author may use in the exercise.

Figure 1 presents the data model of YAPExIL format, with the area of each facet highlighted in a distinct color. The next subsections describe each each of these facets.

3.1 Metadata Facet

The Metadata facet, highlighted in blue in Figure 1, encodes basic information about the exercise that can uniquely identify it and to which subject(s) it refers to. Elements in this facet are mostly used to facilitate searching and consultation in large collections of exercises and the interoperability among systems. For instance, an exercise can be uniquely identified by its `id`, which is a Universally Unique Identifier (UUID) of the exercise.

Furthermore, the metadata includes many other identifying and non-identifying attributes such as the `title` of the programming exercise, the `module` in which the exercise is in (i.e., a description of its main topic), the name of the `author` of the exercise, a set of `keywords` relating to the exercise, its `type` – which can be `BLANK_SHEET`, `EXTENSION`, `IMPROVEMENT`, `BUG_FIX`, `FILL_IN_GAPS`, `SORT_BLOCKS`, or `SPOT_BUG` –, the `event` at which the exercise was created (if any), the `platform` requirements (if any), the level of `difficulty` (one of `BEGINNER`, `EASY`, `AVERAGE`, `HARD`, or `MASTER`), the current `status` (i.e., whether it is still a `DRAFT`, a `PUBLISHED` or `UNPUBLISHED` exercise, or it has been moved to `TRASH`), and the timestamps of creation and last modification (`created_at` and `updated_at`, respectively).

3.2 Presentation Facet

The Presentation facet, highlighted in green in Figure 1, includes all elements that relate to the exercise visualization, both by the students and the instructors. More precisely, these will be the elements placed on the screen while the student solves the problem, and when the teacher firstly opens the exercise.

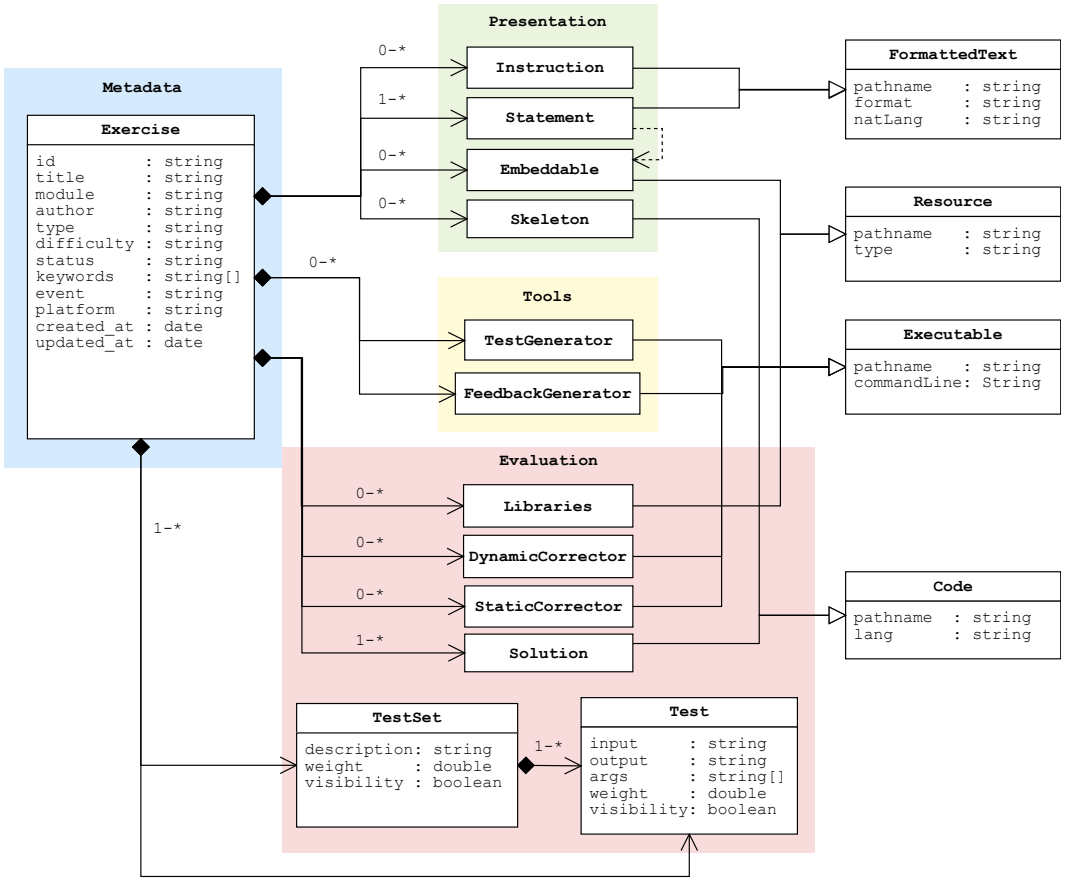


Figure 1 Data model defined by the YAPExIL format.

The supported elements include `instruction` – a formatted text file with instructions to teachers about how to deliver or some remarks on the exercise –, `statement` – a formatted text file with a complete description of the problem to solve –, `embeddable` – an image, video, or another resource file that can be referenced in the statement –, and `skeleton` – a code file containing part of a solution that is provided to the students, from which they can start developing theirs.

All of these elements are allowed multiple instances, being required only a single statement in this facet to have a complete exercise. Hence, formatted text files may be translated to other natural languages or formats whereas code files can be written in several programming languages.

3.3 Assessment Facet

The automated assessment is the end goal of a programming exercise definition language. In order to evaluate a programming exercise, the learner must submit the source code to an evaluation engine. The evaluation engine will then use the necessary and available elements to judge it.

All the elements used in the evaluation belong to the Assessment facet, highlighted in red in Figure 1, and include `template` – code file containing part of a solution that wraps students' code without their awareness –, `library` – code library that can be used by solutions, either

in compilation or execution phase –, `static_corrector` – external program (and associated command line) that is invoked before dynamic correction to classify/process the program's source code –, `dynamic_corrector` – external program (and associated command line) that is invoked after the main correction to classify each run –, `solution` – a code file with the solution of the exercise provided by the author(s), `test` – a single public/private test with input/output text files, a weight in the overall evaluation, and a number of arguments –, and `test_set` – a public/private set of tests.

Each element in this facet also supports multiple instances, being required only a single solution and either a test or a testset with one test. Hence, multiple correctors, libraries, and test/testsets, and solutions in different programming languages may be provided.

3.4 Tools Facet

The Tools facet, highlighted in yellow in Figure 1, encompasses any additional scripts that may be used during the programming exercise life-cycle. These include external programs (and their associated execution command line) that generate (1) the feedback to give to the student about her attempt to achieve a solution (i.e., `feedback_generator`) and the test cases to validate a solution (i.e., `test_generator`).

4 Validation

YAPExIL was designed to fulfil two objectives: (1) support the definition of different types of programming exercises, in addition to the traditional ones; (2) cover all aspects of the model proposed by Verhoeff [6], while being simple and easily convertible. Hence, the validation of YAPExIL must prove the two objectives of this new data model for programming exercises.

The validation of the first objective consists of iterating through each proposed type of exercise and describing how YAPExIL can fulfil its requirements. Table 2 presents the results of this validation.

Table 2 Fulfilment of proposed exercise types.

Type	Fulfilment
`BLANK_SHEET`	Traditional programming exercise that only requires a statement, a test generator (or tests), and a solution. All these elements are part of YAPExIL.
`EXTENSION`	In addition to a `BLANK_SHEET` exercise, this one requires a skeleton, which is part of YAPExIL.
`IMPROVEMENT`	In addition to an `EXTENSION` exercise, this one needs to test other program metrics besides its acceptance. This is achieved through static and/or dynamic correctors.
`BUG_FIX`	This type requires the same elements as an `EXTENSION` exercise (skeleton can be the code with bugs).
`FILL_IN_GAPS`	This type requires the same elements as an `EXTENSION` exercise (skeleton can be the code with gaps marked with gap placeholder).
`SORT_BLOCKS`	This type requires the same elements as an `EXTENSION`, but multiple skeletons. This is supported by YAPExIL.
`SPOT_BUG`	This exercise requires a statement, a skeleton with bugged code, and one test where the output is the bug(s) location. All these elements are part of YAPExIL.

The evaluation of the expressiveness of programming assignments' formats has already been conducted in several works [2, 3, 6]. Table 3 validates the expressiveness of YAPExIL according to the model proposed by Verhoeff [6]. In particular, it iterates through each facet and correspondent features of the model and explains its fulfilment with YAPExIL.

Table 3 Fulfilment of Verhoeff expressiveness model for programming exercise packages.

Facet	Feature	Fulfilment
Textual	Multilingual	Multiple statements and instructions are supported and each of them can have a natural language associated (see Subsection 3.2).
	HTML format	Statements and instructions support HTML files (see Subsection 3.2).
	Other open standard text formats	Statements and instructions support Markdown files (see Subsection 3.2).
	Presentation formats	Statements and instructions support PDF files (see Subsection 3.2).
	Image	Embeddables can be images (see Subsection 3.2).
	Attach files	Embeddables can be attachments (see Subsection 3.2).
	Description	The description is provided in a statement. (see Subsection 3.2).
Data files	Solution	Solution is an element of YAPExIL (see Subsection 3.3).
	Skeleton	Skeleton is an element of YAPExIL (see Subsection 3.3).
	Multi-language	All kinds of elements with source code support multiple files and an associated programming language (see Subsection 3.3).
	Tests	Test is an element of YAPExIL (see Subsection 3.3).
	Test groups	Test elements can be part of a set (see Subsection 3.3).
	Sample tests	Tests can be public (see Subsection 3.3).
	Grading	Tests can have a weight (see Subsection 3.3).
	Feedback	Rich feedback can be provided by a generator.
Configuration & recommendation	Memory limit	May be provided as an argument of a test.
	Size limit	Checked through a static corrector command line.
	Time limit	May be provided as an argument of a test.
	Code lines	Checked through a static corrector command line.
Tools	Test gen.	Test generator is an element of YAPExIL (see Subsection 3.4).
	Feedback gen.	Feedback generator is an element of YAPExIL (see Subsection 3.4).
	Corrector	Static and dynamic correctors are elements of YAPExIL (see Subsection 3.3).
	Library	Library is an element of YAPExIL (see Subsection 3.3).
Metadata	Title	Title is an element of YAPExIL (see Subsection 3.1).
	Topic	Module is an element of YAPExIL (see Subsection 3.1).
	Difficulty	Difficulty is an element of YAPExIL (see Subsection 3.1).
	Author	Author is an element of YAPExIL (see Subsection 3.1).
	Event	Event is an element of YAPExIL (see Subsection 3.1).
	Keywords	Keywords are elements of YAPExIL (see Subsection 3.1).
	Platform	Platform is an element of YAPExIL (see Subsection 3.1).
	Management	Status, creation date, and update date are elements of YAPExIL (see Subsection 3.1).

5 Conclusion

There are plenty of programming exercise formats nowadays. The need for automated assessment of programming exercises led to a massive proliferation of formats following the emergence of automated assessment tools [6, 2, 3, 4]. Notwithstanding, despite some attempts to converge into a common format [5], the interoperability and reusability were left behind, resulting in the absence of a standard open format for representing programming exercises and working tools to convert among them.

In addition to the current inability to share and reuse exercises, the focus of existing formats, largely due to programming contests, has been on traditional programming exercises where the solver must develop a complete solution, starting from a blank file. However, different kinds of programming exercises can foster other programming skills and/or are more adequate at the different stages of learning programming (e.g., spot the bug, fill-in the gaps).

This paper introduces YAPExIL, a JSON format that aims to fulfill both of the identified gaps by building upon an existing (and failed) candidate to standard open format, PExIL. To this end, YAPExIL addresses the pinpointed flaws of PExIL: (1) the exaggerated complexity of the test generation mechanism; (2) using the slower and heavier language XML comparing to JSON format; and (3) the lack of support for non-traditional programming exercises.

YAPExIL is independent of the evaluation engine, heading all efforts to expressiveness and easy conversion from/to other programming exercise formats. Moreover, it is already being used as the main format of a collaborative web programming exercises editor, developed as open-source software [1]. This editor has an internal conversion mechanism for the most known formats, currently supporting two.

Our most immediate future work is the development of support for new formats within this tool, to build a large open collection of programming exercises (featuring gamification in a separate layer).

References

1 FGPE AuthorKit. `http://fgpe.dcc.fc.up.pt`, 2020.

2 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, November 2008. `doi:10.1145/1473195.1473240`.

3 A. Klenin. Common problem description format: Requirements, 2011. accessed on April 2019. URL: `https://ciiwiki.ecs.baylor.edu/images/1/1a/CPDF_Requirements.pdf`.

4 Ricardo Queirós and José Paulo Leal. PExIL: Programming exercises interoperability language. In *Conferência Nacional XATA: XML, aplicações e tecnologias associadas, 9.ª*, pages 37–48. ESEIG, 2011.

5 Ricardo Queirós and José Paulo Leal. Babelo - an extensible converter of programming exercises formats. *IEEE Trans. Learn. Technol.*, 6(1):38–45, 2013. `doi:10.1109/TLT.2012.21`.

6 Tom Verhoeff. Programming task packages: Peach exchange format. *International Journal Olympiads In Informatics*, 2:192–207, 2008.

Open Web Ontobud: An Open Source RDF4J Frontend

Francisco José Moreira Oliveira
University of Minho, Braga, Portugal
a78416@alunos.uminho.pt

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

Abstract

Nowadays, we deal with increasing volumes of data. A few years ago, data was isolated, which did not allow communication or sharing between datasets. We live in a world where everything is connected, and our data mimics this. Data model focus changed from a square structure like the relational model to a model centered on the relations. Knowledge graphs are the new paradigm to represent and manage this new kind of information structure.

Along with this new paradigm, a new kind of database emerged to support the new needs, graph databases! Although there is an increasing interest in this field, only a few native solutions are available. Most of these are commercial, and the ones that are open source have poor interfaces, and for that, they are a little distant from end-users.

In this article, we introduce Ontobud, and discuss its design and development. A Web application that intends to improve the interface for one of the most interesting frameworks in this area: RDF4J. RDF4J is a Java framework to deal with RDF triples storage and management.

Open Web Ontobud is an open source RDF4J web frontend, created to reduce the gap between end users and the RDF4J backend. We have created a web interface that enables users with a basic knowledge of OWL and SPARQL to explore ontologies and extract information from them.

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 RDF4J, Frontend, Open Source, Ontology, REST API, RDF, SPARQL, Graph Databases

Digital Object Identifier 10.4230/OASIcs.SLATE.2020.15

Category Short Paper

1 Introduction

An ontology [32] comprises a formal naming, representation and definition of the categories, properties and relations between the concepts, data and entities/individuals that substantiate one or many domains. Ontologies are very easy to visualize as a graph, where nodes represent concepts/entities, and edges represent relations between the concepts. Using these formalities, knowledge can be defined in a simple way. Because of this, computers can use rules and logic, like the *syllogisms* to extract additional knowledge using inference[13].

RDF4J [24] is an open source Java framework developed to manipulate and/or access RDF data. This framework enables the storage of RDF based ontologies (RDF [17], RDFS [5], and OWL [1]) and their exploitation and management with SPARQL [27] (W3C language and protocol for querying web ontologies). RDF4J works both locally and remotely thanks to its REST API and implements a SPARQL endpoint for all the ontologies stored in it. RDF4J fully supports SPARQL1.1 [28], all mainstream RDF file formats, and RDFS inference.

SPARQL is a query language designed to explore RDF ontologies, and SPARQL 1.1 is an improvement over the original by adding update queries, capable of manipulating an ontology. Inference [14] uses a reasoner and a set of rules to generate new relations between resource in the ontology. These relations are often known as implicit triples. RDFS inference mechanism looks only at RDFS vocabulary within the ontology when generating new relations.

These basic features make RDF4J a versatile and capable tool, which many databases such as GraphDB, Amazon Neptune and Virtuoso have used as their foundation and then improved with additional features [24].

Despite being often used as a stepping stone framework, RDF4J remains a useful RDF Store, still receiving updates and new versions, and its Server and Workbench can be used by anyone thanks to its Open Source license [25, 8]. However the updates focus heavily on bug fixes and sometimes a few new features for the RDF4J Server, while the RDF4J Workbench remains untouched [26].

Open Web Ontobud is a web frontend that brings all the RDF4J features to the user in a simple and complete interface, creating an appealing and easy to use workbench. With its simplicity any user with a basic RDF/OWL and SPARQL knowledge is able to analyze, extract and manipulate information from ontologies.

This section introduced ontologies, RDF4J, its qualities and shortcomings, and the motivation to develop Ontobud. Section 2 presents currently existing frameworks, discusses and compares their strengths and flaws. Section 3 shows the current state of Ontobud and its components, explains its design choices and each component purpose. In Section 4 the paper is concluded and future work is presented.

2 Existing Ontology Frameworks and Databases

Before developing a new web frontend, we analyzed the already existing options. Firstly, some RDF Stores do not have a workbench or frontend, and others provide only a SPARQL query editor. Secondly, we would prefer an open source solution.

The next subsections will talk about some of the existing options that provide an interface and list what features they lack and why they were not chosen.

2.1 GraphDB

GraphDB is a highly efficient, robust, and scalable graph database [10], owned by Ontotext. GraphDB is built on top of the RDF4J framework, using it for storage and querying, and has ample support for query languages, such as SPARQL and SeRQL, and RDF syntax's, like Turtle, RDF/XML, N3, and many others [10]. The usage of all the support provided by RDF4J makes GraphDB easy to use and compatible with industry standards.

It adds the capacity to execute OWL reasoning [12] and is one of the few RDF databases that can perform semantic inferencing at scale, handling heavy query loads and real time inferencing [11].

To support different user demands, GraphDB comes with three commercial editions: GraphDB Free, GraphDB Standard Edition and GraphDB Enterprise Edition [11]. Unfortunately, the commercial nature of GraphDB goes against our desire for an open source solution and prevents us from selecting it as a valid option. Out of all the options provided, only the Free Edition would be viable, but it is still not entirely open source. GraphDB has a good interface, targeting the maintenance and exploitation of stored ontologies, but its main focus is the speed optimizations, allowing fast operations even for massive volumes of data. On the other side, there is a lack of user-centered options. For example, SPARQL queries can only be stored globally, which is not the best way since they are often ontology dependent.

2.2 Blazegraph

Blazegraph is an open source [3], ultra-high-performance graph database used in commercial and government applications [2, 4].

Blazegraph supports different operating modes (triples, provenance, and quads), and RDF/SPARQL API, allowing its use as a RDF Store, and covers application needs from small to large applications, up to 50GB statements stored in NanoSparqlServer [4].

Despite its incredible performance and adaptability, Blazegraph workbench provides only basic tools, such as executing a SPARQL Query and Update, exploring a resource by URI, and listing namespaces. Blazegraph focus is its database features, speed, and size, resulting in a poor workbench, making its management harder. This is a noticeable problem as Wikidata [33, 34], one of Blazegraph biggest users, does not use the default Blazegraph workbench and instead uses the CodeMirror [6] framework to build its workbench, which offers more features such as premade query templates, history, among others.

2.3 Neo4j

Neo4j is an open source, NoSQL native graph database, and comes with a Community and Enterprise edition [20]. In terms of speed, Neo4j delivers constant time traversals for both depth and breadth thanks to its efficient representation of nodes and relationships, all while maintaining a flexible property graph schema that can be adapted and changed at any time [20].

To explore stored ontologies, Neo4J uses Cypher [20]. Cypher is a declarative query language similar to SQL but optimized for graphs, currently used by other databases such as SAP HANA Graph and Redis graph via the openCypher project. However, when compared with SPARQL, it is much more low level. Additionally, up to version 3 (version 4 was recently released), Neo4J only allowed the storage of one single ontology. Fortunately, version 4 fixed this limitation.

Unfortunately, Neo4J natively does not have any inference engine and does not support SPARQL. There are some attempts to work around this problem using plugins such as GraphScale [19], which shows promising results but still has limitations and is currently working on improvements [16].

3 Development of Open Web Ontobud

After analyzing the currently available options and their features, we concluded that no one fulfilled our requirements. This conclusion gave the motivation and led us to propose and develop the Open Web Ontobud, an Open Source RDF4J Frontend.

Ontobud is not just a simple frontend, being divided into four main components: Frontend, Backend, MongoDB, and RDF4J Server, as can be seen in Figure 1. Each of these components, which will be discussed in detail in the following subsections, can be deployed using Docker, or if Docker is not available, a Dockerless deployment is also possible.

The frontend, as the main component, is intended to provide the interface the user will use to interact with and to manipulate his ontologies. The backend exists, most importantly, to manage the authentication process for the users and their access control, but also adds some new queries and facilitates the communication between frontend and databases.

RDF4J was the chosen RDF Store for this project. It can be any RDF4J Server, but a version 3.x or higher is advised, as a few Ontobud functionalities are not supported by earlier RDF4J versions. MongoDB stores all information that RDF4J cannot handle, such as user account information, preferences, context, and saved queries.

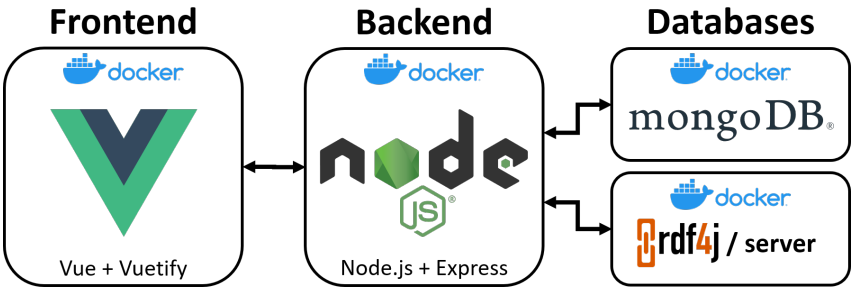


Figure 1 Ontobud Architecture.

3.1 Frontend

The frontend goal is to have an intuitive user interface capable of providing some statistical information for any given ontology, and easy access to frequently queried data. This way, new users to RDF/OWL and SPARQL can use this readily available information while learning the workings and the SPARQL queries behind it. Furthermore, experienced users can benefit from easy access to information by running fewer SPARQL queries.

Vue [30] and Vuetify [31] are the frameworks used to develop Ontobud frontend. Vue is a JavaScript Framework for the development of reactive web frontends. Vuetify is a JavaScript and CSS framework developed for Vue that adds new components with many possible configurations.

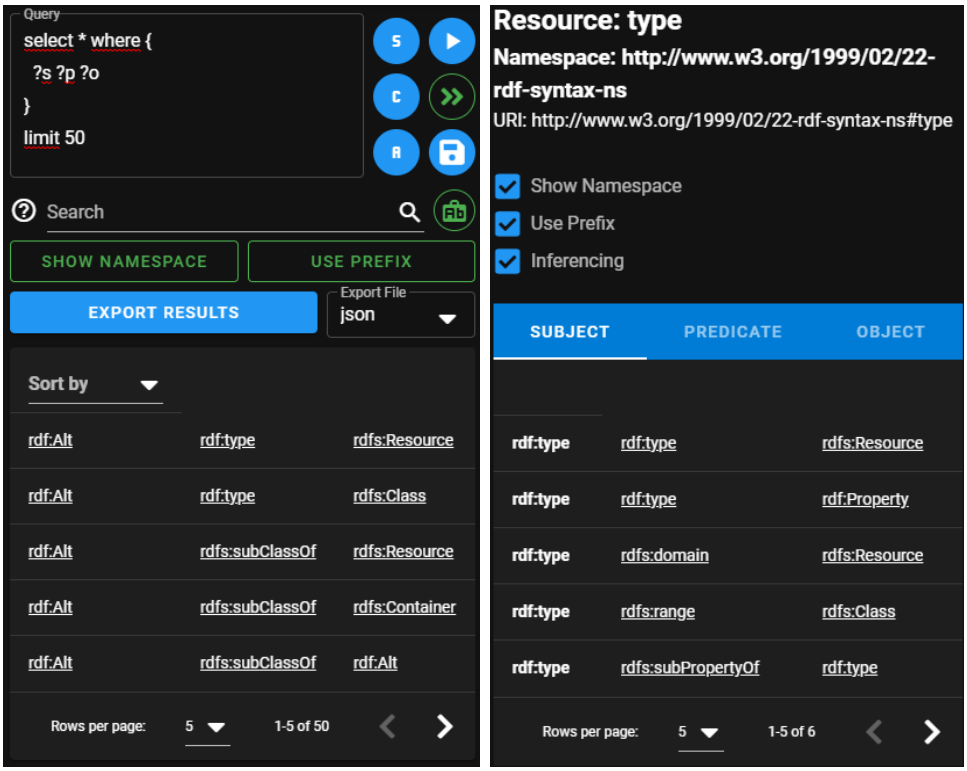
Currently, the frontend allows many operations, most notably:

- Account creation, allowing access to saved queries;
- Repository list and management (add and delete);
- Repository information such as explicit and implicit statement number, used namespaces and a list of existing classes;
- Import and export repository (accepts multiple RDF syntaxes);
- SPARQL 1.1 Query and Update execution, including:
  - SPARQL Query syntax verification (using PEG.js [23] and following the SPARQL Grammar [29]);
  - Query results search, export, and navigation;
- Saved queries:
  - repository specific (not global) or usable in all repositories (global);
  - management (user can add, reuse, edit and delete).

Figure 2a shows the results of a query placed in a table and Figure 2b shows the navigation table, where we can see all triples related to a specific URI. All URI are links that can be clicked and appear underlined to distinguish from literals, which cannot be clicked. This allows the user to navigate the ontology from any query result or resource.

By using a web browser as an interface, which is commonly pre-installed on most computers nowadays, Ontobud Frontend comes with no pre-requisites allowing any person to use the frontend. This enables anyone to use this platform anywhere, including cellphones and tablets, as the responsive nature of Vue allows the page to fit the screen with small changes to the original code.

The frontend is only a component in this project, and due to its open source nature, anyone can modify the original design, improve it, or fix bugs. It would be possible to create alternate interface designs, such as a design oriented to users unfamiliar with SPARQL, or a



(a) Query results. **(b)** Navigation.

Figure 2 Query results and navigation.

one-page design if the users want. The room for customization and improvement is there. With multiple interface designs, each user could run their favorite frontend design, made by themselves or someone else, in their computer while connected to a remote backend.

Running the frontend on your computer can be done using Docker [7] or NPM [21], both simple options with few requirements.

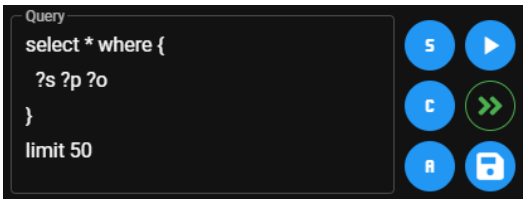
3.1.1 SPARQL syntax parser using PEG.js

Using PEG.js [23] online, we defined a parser to detect SPARQL Query syntax errors. For this, we followed the SPARQL Grammar [29] as a reference. SPARQL Update is currently not included in this parser, and the frontend will warn the user about it, as shown in Figure 3c.

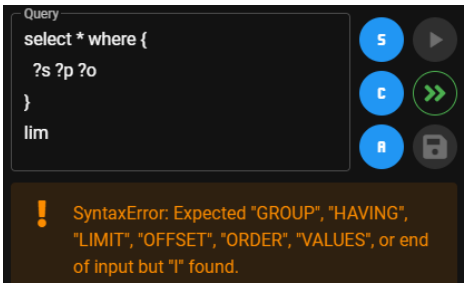
We then downloaded the parser from PEG.js and integrated it into the frontend. This allowed us to notify the user about errors in real-time, as the warning is updated whenever the query changes, and not when executed. This can be seen in Figure 3b. Figure 3c shows a warning, notifying the user that we cannot verify SPARQL Update queries, namely inserts and deletes, and Figure 3a shows a correct SPARQL Query, and therefore no warning is displayed.

3.2 Backend

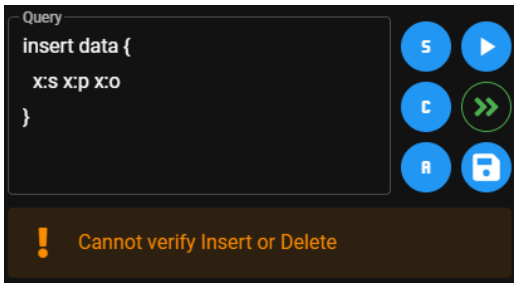
Access control and user authentication are the main focus of the backend. The frameworks used are Node.js [21], and Express [9]. User accounts require an email (account ID), password, and account name. The created account will be processed, having its password encrypted using the NPM [21] package *bcrypt* [22], and then be saved in MongoDB.



(a) SPARQL Query without errors.



(b) SPARQL Query with error.



(c) SPARQL Update notification.

Figure 3 SPARQL Query editor.

User authentication requires the account email and password. Upon authentication success, the backend will return a JSON Web Token [15] (JWT), which will be used for access control, created using asymmetric encryption. The returned JWT will allow the backend to verify if an authenticated user has permission to execute its request.

At this time, the authentication is implemented, and the user access control is currently in development. The access control will mainly allow admins access to special features, such as non-admin account management, and no restrictions in their actions. It also enables request verification from non-admins, for example, preventing userA from deleting an userB saved query.

Furthermore, the backend has its own REST API, divided into authentication, MongoDB, and RDF4J routes. The first two manage the user accounts and access to MongoDB information, respectively. The RDF4J grants access to the repositories and its routes were inspired by the original REST API, but are not all identical. We did this in an attempt to create a more friendly and simple API for users unfamiliar with the original one or interacting with API in general. Currently, not all routes provided by RDF4J are mapped into the backend, only the most common and necessary for the frontend, but direct access to the RDF4J Server is possible if needed.

The frontend uses most of the existing API to supply its functionalities but some of the provided backend routes remain unused, in particular:

- Account information – Returns a JSON object containing user information, such as, account name, email and other information fields to be added;
- Delete user account – Allow an user to delete its account or an admin to delete any account;
- Repository configuration information – Return information about a repository, such as, repository name and ID, which storage type is using and if it is using inference;
- Repository contexts – Returns a list context identifiers from a repository;

3.3 Databases

For data persistence two databases were chosen. The ontologies are stored in RDF4J [24] and the remaining information is stored in MongoDB [18]. Most notably, Ontobud uses the REST API to access the SPARQL endpoints, allowing all its components to run locally or in different machines.

MongoDB is a NoSQL document database, storing data in JSON-like documents [18]. MongoDB is used to store all information RDF4J cannot handle. This includes information such as user account, preferences, information, context and saved queries. User accounts exist to enable access control while saved queries assist the user in its work. We decided to save queries on the server-side because this allows the user access to them on any computer. Creating an account is fast, meaning new users can quickly start saving queries.

4 Conclusion

Along the paper we discussed the development of a frontend for RDF4J. The intent behind this project, which led us to propose and implement Ontobud, was to create an open source and easy to use platform. We compared the possible ontology frameworks and selected the one fitting our requirements.

The application introduced in this paper aims to improve the user experience, for both new and experienced users in the ontology domain, by providing an intuitive interface without cutting on functionalities and an easy access to analytic information about the ontology. We will test it in the context of an academic class with students recently introduced to ontologies.

4.1 Future work

As future work, we aim to finish the user access control, improving the safety in a multi-user environment, and implement a graph visualization and navigation component for a more natural exploration and understanding of any given ontology. The current navigation system presents information in tables by returning triples where a given resource URI is defined as a subject, predicate or object, but this can be confusing and overwhelming to new users.

Additionally, we plan to improve the inference mechanism of Ontobud. RDF4J can make RDFS inferences on uploaded ontologies but lacks OWL inference. This issue is already being worked on another project of our team, where we are using SPARQL Construct queries to simulate inference. A Construct query has two parts, a select clause, and a construct clause. By selecting the axiom conditions and constructing the axiom consequences, we effectively obtain inferred triples from the ontology. Afterward, we inject this inferred triples in the ontology with an Insert query. This process could be added to Ontobud, equipping it with a simple and powerful OWL inference mechanism built with the technology already in place.

References

1 Sean Bechhofer, Frank Van Harmelen, Jim Hendler, Ian Horrocks, Deborah L McGuinness, Peter F Patel-Schneider, Lynn Andrea Stein, et al. OWL web ontology language reference. *W3C recommendation*, 10(02), 2004.

2 The Eclipse BlazeGraph framework. `https://rdf4j.org/about/`. Accessed: 2020-04-19.

3 BlazeGraph License. `https://github.com/blazegraph/database/blob/master/LICENSE.txt`. Accessed: 2020-04-19.

4 BlazeGraph Wiki. `https://github.com/blazegraph/database/wiki`. Accessed: 2020-04-19.

5 Dan Brickley, Ramanathan V Guha, and Brian McBride. Rdf schema 1.1. W3C recommendation. *World Wide Web Consortium, February*, 2014. Accessed: 2020-02-04. URL: https://www.w3.org/TR/rdf-schema/.

6 CodeMirror. https://codemirror.net/. Accessed: 2020-04-19.

7 Docker. https://www.docker.com/. Accessed: 2020-04-19.

8 Eclipse Public License – v 1.0. https://www.eclipse.org/org/documents/epl-v10.php. Accessed: 2020-04-06.

9 Express. https://expressjs.com/. Accessed: 2020-04-19.

10 GraphDB Free. http://graphdb.ontotext.com/documentation/free/. Accessed: 2020-04-07.

11 GraphDB Free – About. http://graphdb.ontotext.com/documentation/free/about-graphdb.html. Accessed: 2020-04-07.

12 GraphDB Free – Free Version. http://graphdb.ontotext.com/documentation/free/free/graphdb-free.html. Accessed: 2020-04-07.

13 Aidan Hogan. Linked data and the semantic web standards. In *Linked Data and the Semantic Web Standards*. Chapman and Hall/CRC Press, 2013.

14 Inference. https://www.w3.org/standards/semanticweb/inference. Accessed: 2020-04-23.

15 JWT. https://jwt.io/. Accessed: 2020-04-30.

16 Thorsten Liebig, Vincent Vialard, Michael Opitz, and Sandra Metzl. GraphScale: Adding Expressive Reasoning to Semantic Data Stores. In *International Semantic Web Conference (Posters & Demos)*, 2015. Accessed: 2020-01-09. URL: http://ceur-ws.org/Vol-1486/paper_117.pdf.

17 Frank Manola, Eric Miller, Brian McBride, et al. RDF primer. *W3C recommendation*, 10(1-107):6, 2014. Accessed: 2020-01-01. URL: http://www.w3.org/TR/2004/REC-rdf-primer-20040210/.

18 MongoDB. https://www.mongodb.com/. Accessed: 2020-04-20.

19 Neo4j: A Reasonable RDF Graph Database & Reasoning Engine [Community Post]. https://neo4j.com/blog/neo4j-rdf-graph-database-reasoning-engine/. Accessed: 2020-04-07.

20 Neo4j Overview. https://neo4j.com/developer/graph-database/#neo4j-overview. Accessed: 2020-04-08.

21 Node.js. https://nodejs.org/en/. Accessed: 2020-04-19.

22 NPM bcrypt package. https://www.npmjs.com/package/bcrypt. Accessed: 2020-04-30.

23 PegJS. https://pegjs.org/. Accessed: 2020-04-30.

24 The Eclipse RDF4J framework. https://rdf4j.org/about/. Accessed: 2020-04-06.

25 RDF4J License. https://rdf4j.org/download/#license. Accessed: 2020-04-06.

26 RDF4J Release notes. https://rdf4j.org/release-notes/. Accessed: 2020-04-07.

27 SPARQL. https://www.w3.org/TR/rdf-sparql-query/. Accessed: 2020-04-23.

28 SPARQL 1.1. https://www.w3.org/TR/sparql11-overview/. Accessed: 2020-04-23.

29 SPARQL Grammar. https://www.w3.org/TR/sparql11-query/#rQueryUnit. Accessed: 2020-04-30.

30 Vue. https://vuejs.org/. Accessed: 2020-04-19.

31 Vuetify. https://vuetifyjs.com/en/. Accessed: 2020-04-19.

32 W3 Standards – Ontology. https://www.w3.org/standards/semanticweb/ontology. Accessed: 2020-07-01.

33 Wikidata. https://www.wikidata.org/wiki/Wikidata:Main_Page. Accessed: 2020-04-19.

34 Wikidata Query. https://query.wikidata.org/. Accessed: 2020-04-19.

Assessing Factoid Question-Answer Generation for Portuguese

João Ferreira
Centre for Informatics and Systems of the University of Coimbra, Portugal
jdcoelho@student.dei.uc.pt

Ricardo Rodrigues
Centre for Informatics and Systems of the University of Coimbra, Portugal
Polytechnic Institute of Coimbra, College of Higher Education of Coimbra, Portugal
rmanuel@dei.uc.pt

Hugo Gonçalo Oliveira
Centre for Informatics and Systems of the University of Coimbra,
Department of Informatics Engineering, Portugal
hroliv@dei.uc.pt

Abstract

We present work on the automatic generation of question-answer pairs in Portuguese, useful, for instance, for populating the knowledge-base of question-answering systems. This includes: *(i)* a new corpus of close to 600 factoid sentences, manually created from an existing corpus of questions and answers, used as our benchmark; *(ii)* two approaches for the automatic generation of question-answer pairs, which can be seen as baselines; *(iii)* results of those approaches in the corpus.

2012 ACM Subject Classification Computing methodologies → Natural language processing

Keywords and phrases Question-Answer Generation, Corpus, NLP, Portuguese

Digital Object Identifier 10.4230/OASIcs.SLATE.2020.16

Category Short Paper

Funding This work was supported by FCT's INCoDe 2030 initiative, in the scope of the demonstration project AIA, "Apoio Inteligente a Empreendedores (*chatbots*)".

1 Introduction

Natural language (NL) interfaces for computer systems are common these days. Many companies rely on *chatbots* for quick customer interactions, such as providing answers to common questions on a given domain, or providing links to documents where answers are likely to be found. Bearing this in mind, we present a new benchmark for question-answer generation (QAG) in Portuguese, followed by two baseline approaches. The corpus includes nearly 600 factoid questions, based on an existing question-answering (QA) corpus. The results of a QAG system are useful, for instance, for creating question-answer pairs, structured as frequently asked questions (FAQ) lists, which can be useful for QA agents or *chatbots*.

In the remaining document, we present a brief description of QA, question-generation (QG) and QAG, introduce the corpus, describe the used approaches and each of its modules, draw some conclusions, and reflect about future work.

2 Background Concepts

Question answering is the process of automatically answering questions formulated in natural language, also using NL. It is a subfield of information retrieval (IR), but answers should contain precise information about the question, rather than a collection of documents. As in other subfields of IR, this often requires the semantic classification of entities, relations, or of semantically relevant phrases, sentences, or larger excerpts [13].

QA systems can use shallow or deep methods for analysing textual elements [1], and can be classified according to question-answer complexity, volume and quality of the source texts, type of corpora used, or how difficult it is to generate answers [12]. QA systems are also characterized by the type of user query – form, type and intent – and by the type of answers provided – named entities, phrases, factoids, links to documents, and summaries. QA can be further divided into open or closed domains, operating on structured data, such as databases or ontologies, or on unstructured data, such as text [12]. Although the latter is, arguably, more difficult, it is also the most common form of human produced documents.

Question generation [17, 2] deals with the analysis of text, identifying sentences and topics that are then used to formulate questions. QG is used in situations such as pedagogical environments, intelligent tutoring systems, conversation with virtual agents and IR [8].

Predictive questioning approaches to QA rely on the generation of questions from a raw corpus. They may assist humans in the creation of new questions, or in the selection of questions proposed by the system [7], possibly associated with the source excerpts. Foundations for such approaches lay on the creation of question-answer pairs, in the guise of a frequently asked questions system, suggesting multiple matches for a user question or presenting questions related to those, providing guidance in the process of selecting further questions on a given topic. QAG deals specifically with the generation of question-answer pairs [10], being used by QA systems or for assessing students, among others.

3 Data and Corpus Creation

Evaluating questions and answers is a complex process, mainly due to the fact that, for a single sentence, a large number of valid questions and respective answers can be generated. In order to do that, three elements are required: a target question, a target answer, and a factoid text that contains both of the previous. The latter is the information the system needs for generation, while the former two are the data required for evaluation. It is important to keep in mind that all these elements can usually be written in more than one way.

Multieight-04 [11] is a corpus of 700 factoid questions and respective answers, all available in seven languages, produced for the CLEF-2004 Multilingual QA Evaluation Exercise. It has two of the three elements required for assessing QAG: questions and answers. To add the third element, for each question-answer pair in Portuguese, we manually created a factoid sentence from which the pair could have been extracted. Examples are shown in Fig. 1.

No effort was made to balance how questions are created, but it is hard to say whether they would be balanced in real text. Moreover, requests not ending with a question mark ("?"), such as "*Mencione um bonecreiro.*", were disregarded. This resulted in a corpus of 581 entries, each including a factoid-like sentence, a question and an answer. This corpus can be used as a benchmark for assessing the automatic generation of questions and respective answers, in Portuguese. It is available at `https://github.com/jdportugal/multieight_pt_facts`.

Of course, multiple factoid sentences could have been produced for the same question. Likewise, a multitude of valid questions and answers may be produced for the same factoid sentence. By having a single possible variation of each element, the corpus will not be able

```
Factoid:  Umberto Bossi é o líder da Liga Norte.
Question:  Quem é Umberto Bossi?
Answer:  Líder da Liga Norte.

Factoid:  O prémio Nobel foi atribuído a Thomas Mann em 1929.
Question:  Em que ano foi atribuído o prémio Nobel a Thomas Mann?
Answer:  1929

Factoid:  Há 100.000 genes humanos.
Question:  Quantos genes humanos há?
Answer:  100.000
```

Figure 1 Examples of Multieight-04 question and answer, with added created factoid sentences.

to determine whether a generated question or answer that is different from the expected is indeed valid. Thus, assessment should be made on a minimum result basis; by that, we mean that the result obtained will be the worst score, with "real" results actually being better.

4 Question Generation Approaches

We compare two predictive approaches for QG in the created corpus: *(A)* exploits chunks and named entities, together with handcrafted rules (see Subsection 4.1); *(B)* relies on Semantic Role Labelling (SRL), using named entities in the process (see Subsection 4.2).

4.1 Approach A – Chunks, Entities, and Rules

Granularity-wise, syntactic chunks are a good option (if not the best) for breaking apart a sentence in blocks, easily rearranged for creating a question from an affirmative statement. This comes from the fact that, when transforming an affirmative sentence into an interrogative one, tokens in the same a chunk can usually be used as whole.

4.1.1 Chunks and Named Entities

This approach relies on the NLPPort suite of NLP tools [16], namely, for splitting sentences in chunks and for named entity recognition (NER). Both included models were trained in the Portuguese treebank Bosque 8.0 [6]. Resulting chunks are classified as nominal (NP), verbal (VP), prepositional (PP), adjectival (ADJP) or adverbial (ADVP) phrases. As for entities, they are classified as abstract, artprod (article or product), event, numeric, organization, person, place, thing, or time. Fig. 2 shows an example of a sentence and its chunks, then checked for the presence of named entities and replacing them by their type, later used for creating a question.

```
Factoid:  John L. Baird foi o inventor da televisão.

Chunks:  [NP John L. Baird][VP foi][NP o inventor][PP de][NP a televisão]
Entities (and types):  <START:person> John L. Baird <END>
→  [NP ?PERSON?][VP foi][NP o inventor][PP de][NP a televisão]
Rule (regex):  ^\[NP \?PERSON\?\].*

Question:  QUEM foi o inventor de a televisão?
Answer:  John L. Baird.
```

Figure 2 Example of the main constituents of Approach A.

As Fig. 2 also shows, generating the question is a matter of replacing the chunk with the entity, by, in that particular case, "QUEM" (WHO) – the interrogative pronoun for the type of the entity PERSON. The text of the answer is taken from the chunk that contains the entity. Of course, not all rules are this simple and not all the sentence structures are like this.

4.1.2 Rules

Rules are based on regular expressions to be applied on the chunked sentences, with entities replaced by their type, as in Fig. 2. When matching the rule, the pivotal chunk would also be the answer. Currently, 33 rules were written for addressing all the factoid sentences, including two generic rules that apply when no entity is found. In that case, the verb in the VP chunk is the main attribute for choosing which of the two rules to use: if the verb is "ser" (to be), and the VP chunk is surrounded by two NP chunks, the generated question is "O que é [left NP chunk] ?" (*What is ... ?*); if it is another verb, but also surrounded by NP chunks, the question is "O que é que [left NP chunk + verb] ?" (*What has ... ?*). In both cases, the answer would be the other NP chunk.

4.2 Approach B – Semantic Role Labeling and Named Entities

The second approach for QAG is based on SRL, the process of classifying words according to their semantic role in a sentence. Given its utility for this task, some approaches for QG already use SRL [4]. For this purpose, we relied on NLPNet [5], based on a Convolutional Neural Network (CNN) trained for performing SRL in Portuguese, and NLPyPort [3] for NER. The main idea was to exploit the *Argument* tags (Arg_0, Arg_1, ..., Arg_n) and the *Verb* form in the sentence for generating the question, while the corresponding answer would be the argument or entity not used in the question. This aimed at creating simple, yet concise answers. Although, at first, this sounds like a feasible approach for QAG, sometimes, a single argument is found, or no arguments are found. Therefore, the proposed approach is divided in three strategies, all used in parallel for generating all possible questions and respective answers. A set of question-answer pairs is thus produced for each sentence, some better than others, i.e., some pairs are complete with a type in the beginning and a coherent wording, but others not so much.

4.2.1 Basic SRL Generation

The first strategy finds all arguments for the subject, all modifier arguments, and the verb, to then generate the question and the answer. QG follows the rule: `Verb + Argument + Modifier Arguments + "?"`, as depicted in Fig. 3, using the rule `Alternative Argument + "."` for generating the answer.

```
Factoid:  O prémio Nobel foi atribuído a Thomas Mann em 1929.

Arg_1:  O prémio Nobel                Arg_2:  a Thomas Mann
V: atribuído                          AM-Temp (Modifier Argument):  em 1929

Question:  Atribuído o prémio Nobel em 1929?
Answer:  A Thomas Mann.
```

Figure 3 Example of question generation using *basic* SRL (Approach B).

For the previous example, the alternative argument would be "A Thomas Mann", since it was not used in the question, and was thus considered to be the answer. Following this template, a question is generated for each of the arguments found, and the corresponding answer uses the arguments not included in the question. The result is an often incomplete question, because the first word, which usually characterises the subject of the answer, is missing and, without additional rules, is indeterminable.

4.2.2 Temporal and Spacial Generation using SRL

The second strategy is similar to the previous, but with a narrowed domain, by considering the sentences where a place or time argument is found. These arguments allow for determination of the type of subject in the generated answer – always a time/place argument – thus enabling the generation of complete questions. A rule for generating this type of question is `"Quando/Onde" + Verb + Argument 0 + Argument 1 + Modifiers Arguments + "?"`, as depicted in Fig. 4, using the rule `AM-TMP/AM-LOC + "."` for the answer. The generated question is close to the expected, but the verb complement "`foi`" is missing.

```
Factoid:  O prémio Nobel foi atribuído a Thomas Mann em 1929.

Arg_1:  O prémio Nobel                  Arg_2:  a Thomas Mann
V: atribuído                            AM-TMP (Modifier Argument):  em 1929

Question:  Quando atribuído o prémio Nobel a Thomas Mann?
Answer:  Em 1929.
```

Figure 4 Example of temporal and spacial question generation using SRL (Approach B).

4.2.3 Entity and Rule Based Generation

When no arguments are identified by SRL, a fallback strategy is used. If an entity is found, its type is determined and the question is generated using a set of rules. If two entities are found and a rule using them exists, an "alternative type" is added to represent both entities' types, and the answer would be the entity not used in the question. One rule of this type for questions is `Type <verb> <Entity1>?` / `Alternative Type <verb> <Entity2>?`, with corresponding answers `Entity2` / `Entity`. An example is shown in Fig. 5.

```
Factoid:  Aleksandr Vassilievich Korjakov nasceu em Moscovo.

Entity:  Aleksandr Vassilievich Korjakov [PESSOA]       Entity:  Moscovo [LOCAL]
V: nasceu

Question:  Onde nasceu Aleksandr Vassilievich Korjakov?
Answer:  Moscovo.

Question:  Quem nasceu em Moscovo?
Answer:  Aleksandr Vassilievich Korjakov.
```

Figure 5 Question-answer pairs generated by the entity based fallback strategy (Approach B).

Since the number of rules is still limited, this part of the approach can be further improved, even if it is only useful when entities are present. Additional rules are shown in Fig. 6.

```
PESSOA LOCAL                            PESSOA PESSOA

Onde <verb> <Entity1>?                  Quem <verb> <Entity1>?
Quem <verb> <Entity2>?                  Quem <verb> <Entity1>?
```

Figure 6 Sample rules used in Approach B, on entity based back-up strategy.

5 Evaluation and Results

Systems based on both QAG approaches were assessed against the created corpus (Section 3). This assessment was based on the similarity between questions and answers generated for each factoid, and those in the corpus, given by two main metrics: BLEU [15], commonly used in the context of machine translation, but also for assessing generated questions [18]; and ROUGE, often used in the context of automatic summarization, for comparing the obtained summary

with human-written summaries. Actually, we used ROUGE-L, a sub-metric of ROUGE that considers the longest matching subset [9]. BLEU relies on the comparison of n-grams in a candidate solution with n-grams of the expected solution, in a position-independent way. The more matches, the better the candidate solution is. Using only unigrams is equivalent to comparing the set of tokens used, which can lead to inflated results. Therefore, towards a complete evaluation, different values for n were considered, namely 1 to 4-grams, as well as a final average that considers the same weight for n-grams of each size.

We knew beforehand the limitations of this evaluation, the main of which is the existence of a single question-answer pair for a factoid, while many others were possible. However, manually creating a corpus with all the possible variations of each of those elements would be impractical. Rather, a single question-answer pair in the corpus was used, and all scoring was based on that pair alone. This outcome penalizes the system, because many valid questions and answers are rendered invalid. However, those that match the expected results are properly scored, and thus enable to assess the minimum performance of the system – not necessarily the best, because questions or answers that would otherwise be correct can be marked as incorrect. To better understand both the question and answer generation performance, results were computed for each of these elements, not for the pair as a whole.

In any case, if the system could not generate a question or an answer, it would be scored 0. Also, since both systems could generate more than a question-answer pair for a single factoid, two evaluation strategies were adopted: *(a)* considering the average of all the generated candidate solutions for each factoid, thus ensuring that over-generation of elements further apart from the expected is penalised; *(b)* in order to give a fair evaluation to the system, considering only the best generated question and ignoring the remaining.

5.1 Evaluation Figures

BLEU scores for QAG with each approach are in Table 1, and ROUGE scores are in Table 2. For any metric, Approach A, based on chunks, performs better than Approach B. This is not unexpected, because the first approach is more polished, due to the use of handcrafted rules, which are also in a larger number, allowing for the generation of more refined solutions.

Table 1 BLEU for the generation of questions and answers of both systems.

Evaluation	***BLEU n-grams***	**Approach A**	**Approach B**
average results for questions	1-gram	0.612	0.270
	2-grams	0.505	0.219
	3-grams	0.450	0.200
	4-grams	0.409	0.189
	weighted 1-4-grams	0.480	0.213
maximum results for questions	1-gram	0.614	0.358
	2-grams	0.507	0.297
	3-grams	0.451	0.273
	4-grams	0.411	0.258
	weighted 1-4-grams	0.482	0.289
average results for answers	1-gram	0.461	0.212
	2-grams	0.342	0.149
	3-grams	0.321	0.135
	4-grams	0.316	0.124
	weighted 1-4-grams	0.323	0.137
maximum results for answers	1-gram	0.463	0.279
	2-grams	0.344	0.201
	3-grams	0.323	0.181
	4-grams	0.318	0.165
	weighted 1-4-grams	0.324	0.185

Table 2 ROUGE for the generation of questions and answers of both systems.

Evaluation	*ROUGE-L*	Approach A	Approach B
average results for questions	Precision	0.438	0.164
	Recall	0.398	0.177
	F1	0.410	0.167
maximum results for questions	Precision	0.440	0.222
	Recall	0.400	0.239
	F1	0.412	0.225
average results for answers	Precision	0.345	0.210
	Recall	0.347	0.176
	F1	0.330	0.182
maximum results for answers	Precision	0.346	0.291
	Recall	0.348	0.240
	F1	0.332	0.291

5.2 Error Analysis

The main problems of Approach A are intrinsic: rules are handcrafted, even if they were thought to be as generic as possible. On average, a rule accounts for an excess of 15 generated questions (and corresponding answers) in the presented corpus. In the same corpus, there are 74 sentences with no questions generated for. Such sentences differ most from the other, in terms of structure, length and hence complexity – this can be a harbinger of the results to be expected in other types of text. Approach B fails more often. A common source of problems is in the SRL system used, NLPNet: some semantic roles are incorrectly classified and result in incorrect solutions; other roles are simply not classified, forcing the system to use the fallback strategies, which are more prone to errors and often generate results of lower quality. An example of a SRL-related problem is depicted in Fig. 7, where bad question and answer were the result of mislabeling the `Arg_0` and `AM-LOC` arguments.

```
Factoid:  A melhor maneira de combater as alergias é administrar, em quantidades
mínimas, as substâncias que causariam ao paciente reacções alérgicas.

Arg_0:  que                          Arg_1:  ao paciente reacções alérgicas.
AM-LOC: em quantidades mínimas.      V: causariam

Question:  Onde causariam que ao paciente reacções alérgicas?
Answer:  Em quantidades mínimas.
```

Figure 7 Sample questions and answers generated by the entity based fallback strategy.

Even the fallback strategies can fail to generate a question, thus resulting in a penalisation when computing the score. For instance, no questions (nor answers) are generated for the factoid "*O aumento da população mundial por ano é de 94 milhões.*", because no semantic roles were assigned and no useful entity types were found. The lack of rules was one of the main problems of the fallback entity based strategy.

To overcome the noted flaws, we could start by improving SRL, so that less solutions depended on the fallback strategies. NLPNet dates back from 2013 and relies on a CNN. Yet, since then, more powerful architectures of neural network were developed and used on sequence-labelling tasks. Those include Recurrent Neural Networks with LSTM layers or Transformers [14]. Handcrafting more rules for the entity based solution generator of Approach B would also allow for the generation of better solutions.

Overall, we believe that Approach B has more room for improvement. Though, it may be a good idea to develop an hybrid system that combines the best parts of the two approaches.

6 Conclusion

We have presented a corpus with 581 factoid sentences, in Portuguese, each with a corresponding question-answer pair. The corpus was used for testing two approaches for QAG, applicable, for instance, to the automatic creation of FAQ-style lists from raw documents, and for automatic population of websites or knowledge bases for NL interfaces, including *chatbots*. Both approaches are rule-based, with rules operating on the output of: *(i)* a syntactic chunker and named entity recogniser – with the best results; and *(ii)* a semantic role labeller. Even though the corpus is not that complex, as most sentences use a straight subject-verb-object structure, we do feel that, as we have done for comparing our approaches, it can be used as a benchmark for other researchers working on QAG for Portuguese.

In the future, it is our intention to apply the presented approaches, or improved versions, to more complex text. In fact, we have already performed preliminary experiments on law text, but results were unsatisfying. Yet, we see the work presented here as a good starting point, where we can build on for pursuing our goal. This includes not only improving the proposed approaches, but also testing more recent machine learning approaches, which would avoid the manual creation of rules, a time-consuming process.

References

1 Johan Bos and Katja Markert. Combining Shallow and Deep NLP Methods for Recognizing Textual Entailment. In *Proceedings of the Pascal Challenges Workshop on Recognising Textual Entailment*, Southhampton, UK, April 2005.

2 Daniel Diéguez, Ricardo Rodrigues, and Paulo Gomes. Using CBR for Portuguese Question Generation. In *Proceedings of the 15th Portuguese Conference on Artificial Intelligence (EPIA 2011)*, pages 328–341, Lisbon, Portugal, October 2011. APPIA.

3 João Ferreira, Hugo Gonçalo Oliveira, and Ricardo Rodrigues. Improving NLTK for Processing Portuguese. In Ricardo Rodrigues, Jan Janoušek, Luís Ferreira, Luísa Coheur, Fernando Batista, and Hugo Gonçalo Oliveira, editors, *Proceedings of 8th Symposium on Languages, Applications and Technologies (SLATE'19)*, OpenAccess Series in Informatics, pages 18:1–18:9. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, June 2019.

4 Michael Flor and Brian Riordan. A Semantic Role-Based Approach to Open-Domain Automatic Question Generation. In *Proceedings of the 13th Workshop on Innovative use of NLP for Building Educational Applications*, pages 254–263, 2018.

5 Erick Fonseca and João Luís Rosa. A Two-Step Convolutional Neural Network Approach for Semantic Role Labeling. In *The 2013 International Joint Conference on Neural Networks (IJCNN)*, pages 1–7. IEEE, 2013.

6 Cláudia Freitas, Paulo Rocha, and Eckhard Bick. Floresta Sintá(c)tica: Bigger, Thicker and Easier. In *Proceedings of the 8th International Conference on Computational Processing of the Portuguese Language (PROPOR '08)*, pages 216–219. Springer-Verlag, 2008.

7 Sanda Harabagiu, Andrew Hickl, John Lehmann, and Dan Moldovan. Experiments with Interactive Question-Answering. In *Proceedings of the 3rd Annual Meeting of the ACL (ACL '05)*, pages 205–214, Morristown, New Jersey, USA, 2005. ACL.

8 Ghader Kurdi, Jared Leo, Bijan Parsia, Uli Sattler, and Salam Al-Emari. A Systematic Review of Automatic Question Generation for Educational Purposes. *International Journal of Artificial Intelligence in Education*, 30:121–204, 2020.

9 Chin-Yew Lin. ROUGE: A Package for Automatic Evaluation of Summaries. In *Proceedings of Workshop on Text Summarization Branches Out, Post2Conference Workshop of ACL*, 2004.

10 Bang Liu, Haojie Wei, Di Niu, Haolan Chen, and Yancheng He. Asking Questions the Human Way: Scalable Question-Answer Generation from Text Corpus. In *Proceedings of The Web Conference 2020 (WWW '20)*, pages 2032–2043. IW3C2, 2020.

11 Bernardo Magnini, Alessandro Vallin, Christelle Ayache, Gregor Erbach, Anselmo Peñas, Maarten de Rijke, Paulo Rocha, Kiril Ivanov Simov, and Richard F. E. Sutcliffe. Overview of the CLEF 2004 Multilingual Question Answering Track. In *Multilingual Information Access for Text, Speech and Images, 5th Workshop of the Cross-Language Evaluation Forum (CLEF), Revised Selected Papers*, volume 3491 of *LNCS*, pages 371–391. Springer, 2004.

12 Mark T. Maybury, editor. *New Directions in Question Answering*. AAAI Press and The MIT Press, Menlo Park, California, and Cambridge, Massachusetts, USA, 2004.

13 Marie-Francine Moens. *Information Extraction: Algorithms and Prospects in a Retrieval Context*. Springer-Verlag, Berlin Heidelberg, 2006.

14 Hiroki Ouchi, Hiroyuki Shindo, and Yuji Matsumoto. A Span Selection Model for Semantic Role Labeling. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1630–1642. ACL, 2018.

15 Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. BLEU: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting on ACL*, pages 311–318. ACL, 2002.

16 Ricardo Rodrigues, Hugo Gonçalo Oliveira, and Paulo Gomes. NLPPort: A Pipeline for Portuguese NLP. In *Proceedings of the 7th Symposium on Languages, Applications and Technologies (SLATE'18)*, OpenAccess Series in Informatics, pages 18:1–18:9, Germany, June 2018. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing.

17 Vasile Rus and Arthur C. Graesser. The Question Generation Shared Task and Evaluation Challenge. Workshop Report, The University of Memphis, 2009.

18 Xingdi Yuan, Tong Wang, Caglar Gulcehre, Alessandro Sordoni, Philip Bachman, Sandeep Subramanian, Saizheng Zhang, and Adam Trischler. Machine Comprehension by Text-to-Text Neural Question Generation. In *Proceedings of the 2nd Workshop on Representation Learning for NLP*, pages 15–25, Vancouver, Canada, 2017. ACL.

SPARQLing Neo4J

Ezequiel José Veloso Ferreira Moreira
University of Minho, Braga, Portugal
pg38413@alunos.uminho.pt

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

Abstract

The growth experienced by the internet in the past few years as lead to an increased amount of available data and knowledge obtained from said data. However most of this knowledge is lost due to the lack of associated semantics making the task of interpreting data very hard to computers.

To counter this, ontologies provide a extremely solid way to represent data and automatically derive knowledge from it.

In this article we'll present the work being developed with the aim to store and explore ontologies in Neo4J. In order to achieve this a web frontend was developed, integrating a SPARQL to CYPHER translator to allow users to query stored ontologies using SPARQL. This translator and its code generation is the main subject of this paper.

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 SPARQL, CYPHER, Graph Databases, RDF, OWL, Neo4J, GraphDB

Digital Object Identifier 10.4230/OASIcs.SLATE.2020.17

Category Short Paper

1 Introduction

With the rise of the internet of things and the growing interconnectivity of our world the necessity to store data in an organised and related faction as well as associating semantic metadata to gather useful knowledge from it becomes paramount.

As such ontologies, formal specification of knowledge understandable by both machines and humans, become more relevant and worthy of study. In particular web ontologies based in RDF, RDFS and OWL.

Neo4J is a well known graph database with an attached query language to explore them: CYPHER. While a graph is not an ontology, an ontology can be represented in a graph: RDF graphs are an example of this. The challenge in this work is to use CYPHER to query not just a graph but a knowledge graph. We will describe the work made in order to enable Neo4J to use SPARQL, a query language for ontologies, explaining the steps necessary to translate queries from SPARQL to CYPHER.

In the following sections we give an introduction to graph databases and graph/ontology query languages. We present the work being carried out and we finish with some conclusions and ideas for future work.

2 Graph Databases

Graph databases are purpose-built to store and navigate relationships. Relationships are first-class citizens in graph databases, and most of the value of graph databases is derived from these relationships.

Graph databases use nodes to store data entities, and edges to store relationships between entities. An edge always has a start node, end node, type, and direction, and an edge can describe parent-child relationships, actions, ownership, and the like. There is no limit to the number and kind of relationships a node can have.

A graph in a graph database can be traversed along specific edge types or across the entire graph. In graph databases, traversing the joins or relationships is very fast because the relationships between nodes are not calculated at query times but are persisted in the database. Graph databases have advantages for use cases such as social networking, recommendation engines, and fraud detection, when you need to create relationships between data and quickly query these relationships.

In this work we are using a graph database, Neo4J, to store OWL ontologies.

An OWL ontology can be serialized as a list of triples. Each triple having the form $(subject, predicate, object)$ in which *subject* is an element of the ontology, *predicate* represents either a property of the element or a relationship with another element(dependant on whether the object is a value or an element of the ontology), and *object* stands for a scalar value or another element's identifier.

We can map subjects and objects to nodes and predicates to edges, thus allowing us to use a graph database to store an ontology.

2.1 Neo4J

Neo4j[6] is an open-source NoSQL native graph database that utilises a generic graph structure to store it's data.

Some of its core features are constant time on depth and breadth traversal and a flexible schema that can be changed at any time. Furthermore it's easy to learn and use thanks to its simple and intuitive UI and language used.

Neo4J stores its data using a generic graph structure: each element is a node that can have properties associated and be related to other elements. These relations can have properties associated with them besides their type and, as such, be considered a special kind of node in the database

To explore stored data, Neo4J uses CYPHER, a declarative query language similar to SQL but optimised for graphs. Unlike SPARQL, CYPHER deals with a generic graph structure(as opposed to an ontology), and as such is a lower level query language if we try to query an ontology.

Furthermore, Neo4J does not support inference (since a graph does not have axioms), and while there are attempts at creating one(like GraphScale[3]) there is currently no fully realised project to resolve this lack.

Up to the version 3 (the latest release was version 4) Neo4J only permitted the creation of one database per installation,a handicap that was solved in the latest version.

2.1.1 Neosemantics

A very important note to make before we move on is that due to not supporting SPARQL and using a generic graph schema, Neo4j does not contain, by default, any way of importing RDF ontologies into it's databases.

To solve this we are using an extension called Neosemantics[7], under the Neo4J Labs[4] project that aims to extend Neo4J capabilities. This extension provides a way to manipulate ontologies using Neo4J through a simple mapping between the ontology and the graph schema.

2.2 GraphDB

GraphDb [2] is an open-source NoSQL graph database that stores its data using RDF graphs.

Some of its core features are native ontology manipulation tools, multiple ontology management, direct query creation and results using a basic Web text editor and a powerful inference engine capable of fully taking advantage of the knowledge gathering capabilities of ontology stored data.

Unlike Neo4J, GraphDB is built specifically for ontologies , and as such provides far more tools for its manipulation (that could not exist in Neo4J due to the more general data schema it employs) and is far more efficient at processing them.

Furthermore it fully implements SPARQL, a query language designed by W3C and becoming the standard in quering ontologies, giving it a far more robust language to deal with the type of data it stores than the more general CYPHER language.

3 Ontology Query Languages

Both Neo4J and GraphDB allow users to interact directly with their databases using queries, written in a given query language(CYPHER and SPARQL, respectively).

Despite having very distinct syntaxes, the actions that they perform over the database are so similar that a partial mapping has already been created as part of this work.

3.1 CYPHER

CYPHER [1] is a declarative query language inspired by SQL and designed for querying, updating and administering graph databases. It is used almost exclusively in Neo4J.

Many of its keywords are inspired by SQL, with pattern matching functionnalities similar to SPARQL, as well as other minor inspirations(such as Python list semantics and query structure that mimics English prose).

The structure of the language is borrowed from SQL, with queries being built up from various clauses that are chained together and feed intermediate results between themselves.

Two simple CYPHER query examples are provided below.

Query C1 – returns every node in the database that contains a property *code* with value 200:

```
match(n) where n.code = 200 return n
```

Query C2 – returns all nodes in the database that contain a relation $hasFather$ with another node in the database:

```
match(n)-[r:hasFather]->(m) return n
```

3.2 SPARQL

SPARQL [14] is a W3C standard language for the manipulation and querying of RDF graphs.

It fully utilises set theory in it's evaluation strategy[12] and provides an extremely solid standard language, as noted by its generalised use anywhere where RDF is involved.

There are four type of SPARQL queries:

SELECT – Select queries are the most used type of SPARQL queries, they are used to retrieve information from RDF graphs;

ASK – Ask queries are boolean, their only purpose is to check if some triple is in the graph;

CONSTRUCT – Construct queries are a powerful tool and can serve multiple purposes. They are composed by two parts, a SELECT clause and a CONSTRUCT clause. SELECT clause enables us to identify specific situations in the graph and the CONSTRUCT clause enables us to program the generation of new triples;

INSERT and DELETE – INSERT and DELETE queries enable us to implement transactions on the ontology. They were added to SPARQL in the last revision.

In the following examples, the same two queries previously coded in CYPHER are now presented in SPARQL syntax.

Query S1 – returns every node in the database that contains a property *code* with value 200:

```
select ?s where { ?s :code "200" }
```

Query S2 – returns all nodes (URIs) in the database that contain a relation *hasFather* with another node in the database:

```
select ?son where { ?son :hasFather ?o }
```

4 From SPARQL to CYPHER, featuring PEG.js

In order to achieve our goals we must create a way to automatically translate from SPARQL to CYPHER. To do this, we need a SPARQL grammar that we will decorate with semantic actions to translate between the 2 languages.

To specify those semantic actions we must look at how ontology data is stored in Neo4J and see how it can be translated into CYPHER from a SPARQL query.

We can observe that Neosemantics stores data from an ontology using a very simple method[8]: for each triple it creates a node for the subject if it does not exist already(using its uri as an identifier), and then checks if the object is an URI or not. If it is, it creates a node with that URI (if it does not exist already) and then creates a relation between the subject node and the object node, where the relation type equals the URI of the predicate. If it isn't, it creates a node with that URI (if it does not exist already) and then adds a property to that node whose name is the predicate and whose value is the object. This process is repeated for every triple in the ontology.

With this knowledge we can already observe that a query to obtain data from a given element of the ontology will require a match with a node that contains the specific uri in its properties. In the same vein, we can see that due to the way that properties and relations work in Neo4J we will need to find a way to differentiate a property from a relation and gather the data accordingly.

To do all this, we first need a way to create a parsing grammar that is compatible with Javascript (in order to be easily integrated in our application). We have decided to use PEG.js[9], a parser generator for Javascript that provides fast parsing and good error handling, mainly due to its easiness of integration with other tools or apps.

With this tool, an initial version of a grammar that translates SPARQL into CYPHER has been developed. This translation grammar is based upon the W3C documentation that describes the syntax rules for the SPARQL language[13](modified to work with PEG.js's way of parsing) with added semantic actions that allow it to transform valid SPARQL queries into valid CYPHER queries.

4.1 Query translation example in depth

To give an idea of how translation works, we present the following example.

Let's take the following query in SPARQL and translate it using our grammar:

Query S3 – returns all pairs $(elem, father)$ where $father$ is the URI of every element with a $: hasFather$ relation with the subject with id $http://www.my_ontology.com\#elem$. We can see the declaration of a namespace as default:

```
prefix : <http://www.my_ontology.com#>
select * where{ :elem :hasFather ?father }
```

Before going into the translation note that: the translation showed here had small changes made to them that do not alter the query in any other way then to make it prettier for this paper, namely the fact that all variables that are generated by the grammar that are not directly related to the results we want to get use a pseudo RNG algorithm([11]) in their names to guarantee that will never collide with user set variables.

This does, however, make them quite hard to understand in a paper. As such, they have been altered to gVX, where X is an integer that denotes a unique internal variable.

Query C3 – query **S3** translated to CYPHER by the grammar:

```
match(gV1) where gV1.uri ="http://www.my_ontology.com#elem"

unwind [key in keys(gV1) where key = "http://www.my_ontology.com#hasFather" | [
    gV1[key],null,key]] + [(gV1)-[gV2]->(customVar_father) where type(gV2) = "
    http://www.my_ontology.com#hasFather" | [customVar_father.uri,
    customVar_father,type(gV2)]] as gV3
with *,gV3[0] as father,gV3[1] as customVar_father

with father where (father is not null)
RETURN *
```

The first line of the translation contains a match with a node that contains a specific uri in the ontology(in this example,$< http://www.my\_ontology.com\#elem >$). This is necessary since in order to obtain the triples associated to a given element of the ontology we must first match with it in the database. As such, this first line corresponds to us matching with $: elem$ in the SPARQL query.

After matching with our subject, we have to obtain the desired object that is associated to our subject and the predicate $: hasFather(http://www.my\_ontology.com\#hasFather)$. To do that, we must search for the predicates and objects associated to our subject node and find the ones that have a predicate $: hasFather$, and to do that we must use arrays and unwinds inside the query.

This unwind expression and the corresponding with expression is complex, so we'll break it down in small parts:

(1) **[key in keys(gV1) where key= "http://www.my_ontology.com#hasFather" | [gV1[key],null,key]]** – with this expression we gather all the information about triples from the ontology that are properties of our subject that follow the conditions we want (in this case, the predicate must have a value: $: hasFather$).

The gathering of the properties is done by
key in keys(gV1) where key=
"http://www.my_ontology.com#hasFather"
and the return of the data is done by
[gV1[key],null,key]
The reason we return a null value has to do with the fact we may need to join the results of this search in the properties with the search in the relations, and that search returns 3 results, only one is relevant to the other search.

(2) **[(gV1)-[gV2]->(customVar_father) where type(gV2)="http://www.my_ontology.com#hasFather" | [customVar_father.uri,customVar_father,type(gV2)]]** – with this expression we gather all the information about triples from the ontology that are relations of our subject that follow the conditions we want.
The gathering of the relations is done by
(gV1)-[gV2]->(customVar_father) where
type(gV2)="http://www.my_ontology.com#hasFather"
and the return of the data is done by
[customVar_father.uri,customVar_father,type(gV2)]
Unlike in the properties, we have to search the relations and the nodes that are associated to those relations. Because of these nodes, our return has another value to return: the node reference itself. The reason we must return this is because we might use this variable again down the line in the query, and if we do that then we can access the node directly without searching the database again, saving us a lot of time in processing the query.

(3) **(1) + (2)** – this expression allows us to gather results from all the triples that our subject has and join them in a single array of results.
The reason we must use this in this case as to do with the fact that since the grammar does not know if : $hasFather$ denotes a relationship or a property name we cannot limit our search to either set of elements, and must search both of them.
This is not always the case. If the object is a literal value, this expression will be **(1)**, and if the object is an uri this expression will be **(2)**.

(4) **unwind (3) as gV3** – with (3) processed we now have an array of data that has the data we want to gather. However, it has an array form, and as such is not easy to manipulate or reference if we later need the data that we have gathered.
To make this data available, we must process element by element of the array, and we use the unwind command to do that.

(5) **with *,gV3[0] as father,gV3[1] as customVar_father** – finally, we return the data we have gathered using appropriate names to easily reference them later.
Since we don't care about the predicate value(since it's a fixed value and thus not returned by the query) we only need to concern ourselves with 2 values: the value that corresponds to the ?$father$ variable and the node that corresponds to the element of that variable(if it exists).
As such, we return the first value as $father$ and the node value as $customVar\_father$.

Finally, since we don't have to process any other triples we reach the query data return, last 3 lines of the query. In this data return we catch only the data we have captured (in this case, the $father$ value), then we check if it is not null (since we don't have the optional keyword working yet in the translation) and finally we return what we have been asked to return(in this case its a **RETURN *** since the SPARQL query is a **select ***).

4.2 Translated types of queries

At this moment, the grammar can translate a SPARQL portion into CYPHER:

prefixes – in SPARQL a PREFIX statement defines an alias for the ontology namespace, it has the form: $<prefix : namespace>$. This prevents the need to write down the ontology full URI whenever we want to reference a subject or a property from that ontology. When the name that precedes the ":" is empty then we are in a special case, we are declaring the default namespace that will be associated with the empty prefix;

same subject or different subject triples – SPARQL uses triples in order to make matches with the graph data. It can make multiple of these matches in a single query, through the use of ";" to fix the subject between triples and the use of "." to denote completely different triples that should be matched. This allows us to chain them together to obtain more refined data, as well as extract more specific knowledge from the database;

filter expressions – SPARQL FILTER statement restricts the query results, only triples that satisfy the predicate inside FILTER clause are returned as result.

Unlike the previous, only part of the filter expression has been dealt with by the grammar due to its complexity. As of now, only basic mathematical and logical expressions are working, as well as the $EXISTS$ keyword;

limit and order by – SPARQL LIMIT expression limits the amount of returned results. SPARQL ORDER BY expression orders the results of the query by one of the columns present in the result;

ask query – a SPARQL ASK query is used to determine if a given triple is present in the ontology;

describe query – a SPARQL DESCRIBE query is used to gather every triple associated with the elements that it matches with.

A note about the translation grammar is that due to the complexity of the operation it does not directly do the describe operation. Instead it captures the elements that the query should capture and then the result from the query must be used with the neosemantics export operation using CYPHER query to obtain the desired result.

5 Grammar validation

In order to validate our grammar we have choosen 2 base ontologies for our tests: one that contains information about the periodic table and another that contains information about the pokedex (an encyclopedia for the Pokemon game franchise).

Both ontologies were imported to GraphDB and Neo4J after being processed through the use of a reasoner in protégé([10]). This was done so that any triples that arise from the rules laid out by the ontology are explicitly stated inside the ontology we are importing.

Next, we created numerous queries in SPARQL testing different types of queries.

After that we had to query the databases. The method used to do this was: directly make the query to the database in GraphDB, use the grammar to translate it into CYPHER and then query Neo4J with the translated query.

While numerous queries were made, we show the results of one of them made to the periodic table dataset, specifically a query that is used to obtain all triples associated to a given subject of the ontology *"give me anything you know about Carbon"*:

Query S4 – returns all predicates and object that have the subject <http://www.daml.org/2003/01/periodictable/PeriodicTable#C>:

```
PREFIX : <http://www.daml.org/2003/01/periodictable/PeriodicTable#>
select * where { :C ?p ?o . }
order by ?p
```

Query C4 – S4 translation to CYPHER

```
match(gV1) where gV1.uri ="http://www.daml.org/2003/01/periodictable/
    PeriodicTable#C"
unwind [key in keys(gV1) | [gV1[key],null,key]] + [(gV1)-[gV2]->(customVar_o) |
     [customVar_o.uri,customVar_o,type(gV2)]] as gV3
with *,gV3[0] as o,gV3[1] as customVar_o,gV3[2] as p
with p,o
where (p is not null) and (o is not null)
RETURN *
ORDER BY p
```

After making these queries to the respective platforms, we obtained the result tables shown in Figure 1.

Result tables show that results are similar between both result sets. However, there are a couple differences, namely:

- Cypher's results have an additional row. This row contains the predicate "uri" and the object ":C". The reason this row exists is due to how neosemantics handles uri's. In order to guarantee that every uri is unique inside the ontology being imported, neosemantics requires the addition of a uniqueness constraint([5]). In order for this to work, neosemantics adds an uri field to every element of the ontology it imports, serving as an identifier to what that element is supposed to represent.
- Values in SPARQL that have a type attached to them have that type disappear and are imported as literal values. The reason for this is because of how neosemantics deals with custom data types versus the default data types. If the datatype is default and during the setup of the database we choose to allow custom data types, then they will be appended to the end of the value, working the same way as you'd expect them to in GraphDB. The reason it does not work like that is that neosemantics import only does this for custom data types, i.e., data types that are not already implemented in Neo4J. For those, it simply imports the value into the dataset without the tag at the end. Unfortunately, despite Neo4J fully implements floating point numbers, the import process seems to simplify them into integers, and thus the differences between the results.

6 Conclusion

Translating SPARQL into CYPHER is a complex process due to differences both in the languages syntax and how the data is stored in the associated platforms, making a full correlation between the 2 extremely difficult.

Throughout this article we have exposed part of that process, from explaining the engines that operate with those languages to the specifics of each one, as well as showing the translation capabilities of the parser implemented so far, as well as showing part of the validation process for the developed grammar.

p	o
:atomicNumber	“6”^^xsd:integer
:atomicWeight	“12.0107”^^xsd:float
:block	:p-block
:casRegistryID	7440-44-0
:classification	:Non-metallic
:color	graphite is black, diamond is colourless
:group	:group_14
:name	carbon
:period	:period_2
:standardState	:solid
:symbol	C
rdf:type	:Element
rdf:type	owl:NamedIndividual

(a) Result table for query in GraphDB.

p	o
:atomicNumber	6
:atomicWeight	12
:block	:p-block
:casRegistryID	7440-44-0
:classification	:Non-metallic
:color	graphite is black, diamond is colourless
:group	:group_14
:name	carbon
:period	:period_2
:standardState	:solid
:symbol	C
rdf:type	:Element
rdf:type	owl:NamedIndividual
uri	:C

(b) Result table for query in Neo4J.

Figure 1 Result tables.

References

1 Cypher introduction. `https://neo4j.com/docs/cypher-manual/current/introduction/`. Accessed: 2020-04-30.

2 Graphdb free. `http://graphdb.ontotext.com/documentation/free/`. Accessed: 2020-04-07.

3 Neo4j: A reasonable rdf graph database & reasoning engine [community post]. `https://neo4j.com/blog/neo4j-rdf-graph-database-reasoning-engine/`. Accessed: 2020-04-07.

4 Neo4j labs:incubating the next generation of graph developer tooling. `https://neo4j.com/labs/`. Accessed: 2020-04-30.

5 Neoseantics create uniqueness constraint. `https://neo4j.com/docs/labs/nsmntx/current/config/#create-resource-uniqueness-constraint`. Accessed: 2020-06-30.

6 Neo4j overview. `https://neo4j.com/developer/graph-database/#neo4j-overview`. Accessed: 2020-04-08.

7 Nsmntx – neo4j rdf & semantics toolkit. https://neo4j.com/labs/nsmtx-rdf/. Accessed: 2020-04-30.
8 Importing rdf data into neo4j. https://jbarrasa.com/2016/06/07/importing-rdf-data-into-neo4j/. Accessed: 2020-07-04.
9 Peg.js website. https://pegjs.org/. Accessed: 2020-04-30.
10 protégé:a free, open-source ontology editor and framework for building intelligent systems. https://protege.stanford.edu/. Accessed: 2020-06-30.
11 Linear congruential generator. https://en.wikipedia.org/wiki/Linear_congruential_generator. Accessed: 2020-06-30.
12 Definition of sparql. https://www.w3.org/TR/sparql11-query/#sparqlDefinition. Accessed: 2020-04-30.
13 Sparql: Grammar. https://www.w3.org/TR/sparql11-query/#sparqlGrammar. Accessed: 2020-04-30.
14 Sparql 1.1 query language. https://www.w3.org/TR/sparql11-query/. Accessed: 2020-04-30.

bOWL: A Pluggable OWL Browser

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

Ricardo Queirós
CRACS – INESC, LA, Porto, Portugal
uniMAD – ESMAD, Polytechnic of Porto, Portugal
http://www.ricardoqueiros.com
ricardoqueiros@esmad.ipp.pt

Abstract

The Web Ontology Language (OWL) is a World Wide Web Consortium standard, based on the Resource Description Format standard. It is used to define ontologies. While large ontologies are useful for different applications, some tools require partial ontologies, based mostly on a hierarchical relationship of classes. In this article we present bOWL, a basic OWL browser, with the main goal of being pluggable into others, more significant, web applications. The tool was tested through its integration on LeXmart, a dictionary editing tool.

2012 ACM Subject Classification Information systems → Web Ontology Language (OWL)

Keywords and phrases OWL, Web Plugin, OWL Browser, Ontology

Digital Object Identifier 10.4230/OASIcs.SLATE.2020.18

Category Short Paper

Funding This work was partly founded 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/50014/2020.

1 Introduction

With the Semantic Web, there has been a widespread of ontologies, linked data, and other resources. This is excellent, as these resources are encoded in machine readable formats, allowing their processing automatically. Nevertheless, while large data-sets are released every week, there are some specific tools that can benefit from smaller and simpler ontologies. In fact, any website that we currently use, and that is known as being Web 2.0, can use an ontology: blog software allows the use of trees of concepts, photo galleries allow tagging images with classes. All these structures could be codified as ontologies. But unfortunately, the way these sites treat classification is quite limited.

The main reason for using simple trees for classification (something that is quite near the taxonomy) is the relatively simple way they are built. While we lack proof, a reason might be the quite complicated way ontologies are presented. Indeed, ontologies are complicated structures, but at their core, they use a backbone which is (almost) a tree. In fact, the most used ontology building software (Protégé[1]) uses an interface that emphasis this specific structure.

In this paper, we explore how a simple ontology can be built using any external editor and used in a Web application, where the user can navigate it to classify individuals, and can only perform basic operations on top of the ontology directly.

[1] https://protege.stanford.edu/

Some sites already allow the navigation on an ontology, but our main contribution is to build this ontology browser (and minimal visualizer) as a plug-in, to be easily embedded in larger applications, and with a small code footprint.

2 The OWL Standard and OWL Editors

The Web Ontology Language [6] is a language designed by the World Wide Web Consortium (W3C) on top of the Resource Description Framework (RDF) specification. Its specification allows different dialects. One of the basic exportation formats by tools like Protégé [2] and its Web variant, Webprotégé [5] is a flat format, that describes the ontology classes, its annotation properties, and relations, one at a time.

2.1 The OWL Standard

In this section, we describe a few of the structures that the OWL standard allows, that are crucial for the tool being developed. Although we are aware that the OWL standard is a larger whole world, we currently are targeting a small subset, that might be expanded in the future.

- Prefix definition:

```
<Prefix name="rdf" IRI="http://www.w3.org/1999/02/22-rdf-syntax-ns#"/>
```

  These are single elements, that describe some IRI prefixes which can be abbreviated in the document. This allows that some elements have a `IRI` attribute, with a full IRI, and some others have an `abbreviatedIRI` attribute, that uses the prefix notation.
- Classes definition:

```
<Declaration>
    <Class IRI="http://webprotege.stanford.edu/agricultura"/>
</Declaration>
```

  These declarations are used to define which classes are recognized by the ontology. Each declaration includes just the class IRI.
- Data properties definition:

```
<Declaration>
    <DataProperty IRI="http://webprotege.stanford.edu/abbreviation"/>
</Declaration>
```

  Similar to the classes definition, these structures declare all data properties that can be used in the ontology. Data properties are treated as entities very similar to classes as we will see, as they can also have relations and properties.
- There are two different kind of sub classes definitions. The first one, for standard ontology classes hierarchy, OWL describe them as:

```
<SubClassOf>
    <Class IRI="http://webprotege.stanford.edu/citologia"/>
    <Class IRI="http://webprotege.stanford.edu/biologia"/>
</SubClassOf>
```

  Thus, in this example, the class “citologia” [citology] is a subclass of “biologia” [biology]. The relationship between classes and data properties are also described as a `SubClassOf` element, as in the next example:

```
<SubClassOf>
   <Class IRI="http://webprotege.stanford.edu/citologia"/>
   <DataHasValue>
      <DataProperty IRI="http://webprotege.stanford.edu/abbreviation"/>
      <Literal>citol.</Literal>
   </DataHasValue>
</SubClassOf>
```

These structures describe a relation from a class with a literal (a string) by a data property previously defined.

- OWL treats data properties as classes. Therefore, they can have their own hierarchy. Also, as they define relationships between elements, it is possible to describe their domain and range:

```
<SubDataPropertyOf>
    <DataProperty IRI="http://webprotege.stanford.edu/abbreviation"/>
    <DataProperty abbreviatedIRI="owl:topDataProperty"/>
</SubDataPropertyOf>
<DataPropertyDomain>
    <DataProperty IRI="http://webprotege.stanford.edu/abbreviation"/>
    <Class abbreviatedIRI="owl:Thing"/>
</DataPropertyDomain>
<DataPropertyRange>
    <DataProperty IRI="http://webprotege.stanford.edu/abbreviation"/>
    <Datatype abbreviatedIRI="xsd:string"/>
</DataPropertyRange>
```

The first block describes the abbreviation relation as a child to a top level data property. The second block defines its domain (any class child of `owl:Thing`). Finally, the latter, describes the range or co-domain of the relation: *strings*.

- Finally, there are annotation assertions, that annotate classes and relations with external objects. These are similar to relations whose range are strings:

```
<AnnotationAssertion>
    <AnnotationProperty abbreviatedIRI="rdfs:label"/>
    <IRI>http://webprotege.stanford.edu/abbreviation</IRI>
    <Literal xml:lang="pt">abbreviation</Literal>
</AnnotationAssertion>
<AnnotationAssertion>
    <AnnotationProperty abbreviatedIRI="rdfs:label"/>
    <IRI>http://webprotege.stanford.edu/agricultura</IRI>
    <Literal xml:lang="pt">agricultura</Literal>
</AnnotationAssertion>
```

These two examples show that annotations assertions can be defined both for properties (first example) or to classes (second example). For each, a relation (property) is defined, and a target language for the literal annotating the element.

These structures comprise a small subset of the OWL standard. Nevertheless, they are the kind of structures that we expect to be useful to embed on a pluggable Web widget.

2.2 OWL Libraries and Editors for the Web

There are two main types of tools to handle OWL in the Web: libraries to be used by programmers, and online editors, to be used by end-users.

As a library, we can find some JavaScript modules to handle RDF/OWL, and perform queries on them. Their main goal is not to allow the creation of the ontology, but to reason over it.

There are some of the libraries available that are written to be run in the browser (and not as a server library, running on node.js). Most are prepared only as triple stores, to load RDF documents and allow queries using SPARQL [1]:

- `rdfstore-js`[2] is built as a triple store, supporting the SPARQL query language. While it might be useful as a library, it is over complicated for our intents. Also, given bOWL is a plugin for other applications, it is important to use to keep it with a small footprint.
- `rdflib.js`[3] has similar goals as `rdfstore-js`, allowing the loading of RDF in different formats, and performing queries using SPARQL. Its versatility makes it a huge library to be embedded in a web application.
- `SPARQL.js`[4] as expected from its name, is another triple store library. In fact, it is more devoted to the parsing of the SPARQL language than on the processing of RDF.
- `rdfquery`[5] is not maintained for more than 10 years, and its main goal is to be used as a query library for RDF documents.

Regarding editors, there is not much choice. The main used editor for OWL is Protégé [2], that includes a web version, WebProtégé [5]. But this is a full blown web application, and is not prepared to be plugged into other applications, as a simple module to manage the main ontology hierarchical structure.

Protégé is implemented in Java, and WebProtégé is a simple Web wrapper to the Protégé library. They both use the OWL Api[6] Java library to generate and parse OWL files. While there is an attempt to port this library to JavaScript[7], it does not run in a browser, neither in node.js.

3 Implementation Details

The main goal when deciding for the implementation of a new tool to browse ontologies was making it embeddable, as independent as possible from other tools, and event oriented. It should also support basic edition capabilities.

3.1 bOWL internal OWL representation

Figure 1 shows the class diagram for bOWL. As it can be observed, we are focusing on very specific constructions. While in the future there might be the option to add new features, at the moment the main goal is to have it working for basic ontologies that define, mostly, a kind-of taxonomy.

Currently bOWL stores information about prefixes – in order to be able to expand or compact abbreviated URIs –, information about classes, and information about data properties.

For each class, its IRI is stored along with its parents (using the subClass relation) and its data and annotation properties. Each annotation property has the respective literal and the used language for that literal. Regarding data properties, only the IRI for the property relation and the value are stored.

The information about data properties define their IRI along with their domain and range, and annotation properties.

---

[2] `https://github.com/antoniogarrote/rdfstore-js`
[3] `https://github.com/linkeddata/rdflib.js`
[4] `https://github.com/RubenVerborgh/SPARQL.js`
[5] `https://code.google.com/archive/p/rdfquery/`
[6] `https://github.com/owlcs/owlapi`
[7] `https://github.com/cmungall/owljs`

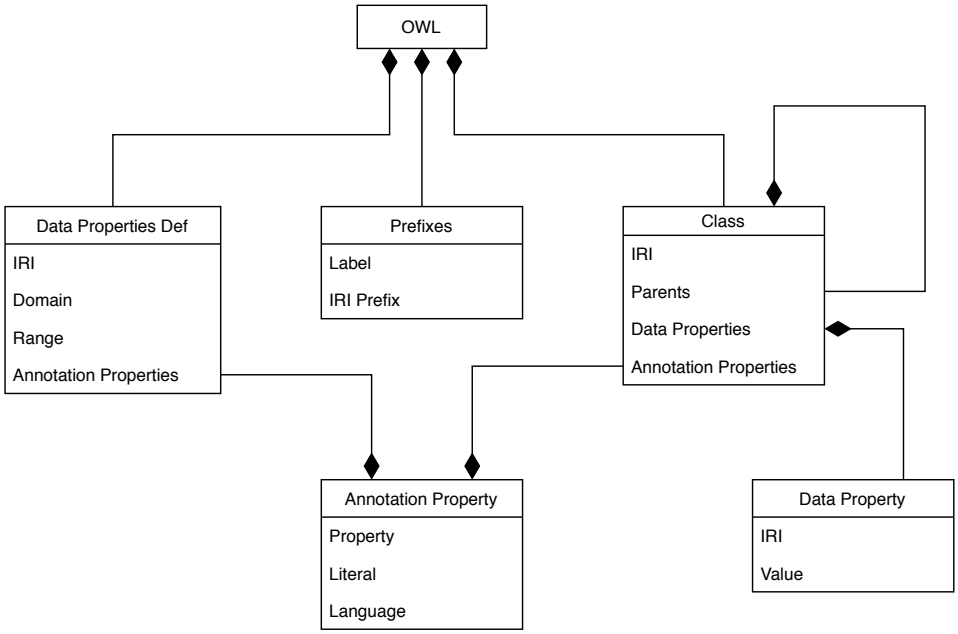


Figure 1 bOWL class diagram.

3.2 bOWL interface

bOWL is prepared to work by HTTP requests (using a GET request to fetch the ontology and a POST request to save it) or to work directly with the internal browser storage. In the future we will also implement a synchronization tool that guarantees that the ontology in the browser storage is up to date with a remote master ontology.

Given bOWL uses, internally, a JSON representation, the load and save mechanisms allow to fetch OWL directly, or to use bOWL internal JSON representation.

The communication from the world with bOWL is defined in three events/methods (see Table 1):

- The constructor, responsible for loading the ontology and showing it inside a specific element; The ontology can be load from the internal storage (in that case the constructor parameter is a string, representing the ontology name) or from a remote server (in that case, the constructor receives an URI). To make the tool more usable, the constructor can receive a second parameter with a default prefix for all created classes. This prefix will not be added in the OWL as a standard prefix: all classes will have their IRI fully qualified.
- bOWL allows simple editing of new classes (further edition capabilities are planned). Therefore, when closing the bOWL widget, the programmer might want to call a method to save the data. For saving it, the programmer might supply an URI, where the full ontology will be sent as a POST request, a string with the name of the ontology to be saved in the browser storage, or a code reference that will be run with the ontology representation as a string. A second parameter can be used to supply the saving format.
- One of the main goals is to make bOWL usable as a classification widget, allowing the user to open a popup window with the ontology and choose a class she wants to apply to some document. Thus, by default, and “apply” button is presented in the widget. When clicking this button, an event will be triggered, calling a user defined callback with that selected element data.

Table 1 bOWL programming interface.

Method	Description
`Bowl load (URI [, prefix])`	Receives an URI/Key as parameter and an optional prefix. Returns the bOWL object. Mime-type (json or xml) is automatically detected.
`Object get (event)`	Associates an event that will be triggered when the user clicks the "Apply" button. It returns an object with the selected class data.
`save (URI [, format])`	Given an URI/Key as parameter (treated like in the load method), stores the full ontology. By default, saving in local storage will use JSON while saving remotely will use OWL. This can be overridden with a second parameter. This method can also define a specific code callback to be called that should be responsible for the data save process.

3.3 Supporting technologies

bOWL is implemented using ECMAScript 8. For the XML parsing we use `js2xml`[8], that converts the XML file into a JSON data structure. Then, JSONPath Plus[9] is used to traverse the data structure and extract the required information.

For the front-end, `jstree`[10] was chosen for its versatility and easy of use. While an ontology is not necessarily a tree, it can be shown as such, repeating an element as a child of multiple parents. This is the current approach used by other tools like Protégè.

4 Proof of Concept

Our case-study is LeXmart [4, 3], a Web editor for general language dictionaries. It is developed as a Web application on top of eXist-DB[11], using Web Standards like XQuery for the application development, HTML5 and CSS for the frontend, and JavaScript for dynamic content. LeXmart features an integrated TEI editor, based on Xonomy[12].

One of the features under work, allowing the lexicographer to link dictionary senses with ontology concepts, required the ability to both show a basic ontology structure (specially its bare-bone taxonomy) and to navigate and edit it. This way, bOWL was projected as a simple JavaScript library to be integrated into LeXmart, but to be as independent as possible from it, allowing its integration with other tools.

Figure 2 shows an example of the widget in use. There is the main tree, where classes are presented in hierarchy accordingly with their OWL structure, and their `rdfs:label` annotations. When selecting an element it can be used as a parent to create a new subclass, as shown in the modal window. There, the user needs to add the IRI and the label. The parent class is automatically selected. Finally, the "Apply" button is used to send the control back to the hosting application, together with the information of the selected class.

[8] `https://github.com/x2js/x2js`
[9] `https://github.com/s3u/JSONPath`
[10] `https://www.jstree.com/`
[11] `https://exist-db.org/`
[12] `https://github.com/michmech/xonomy`

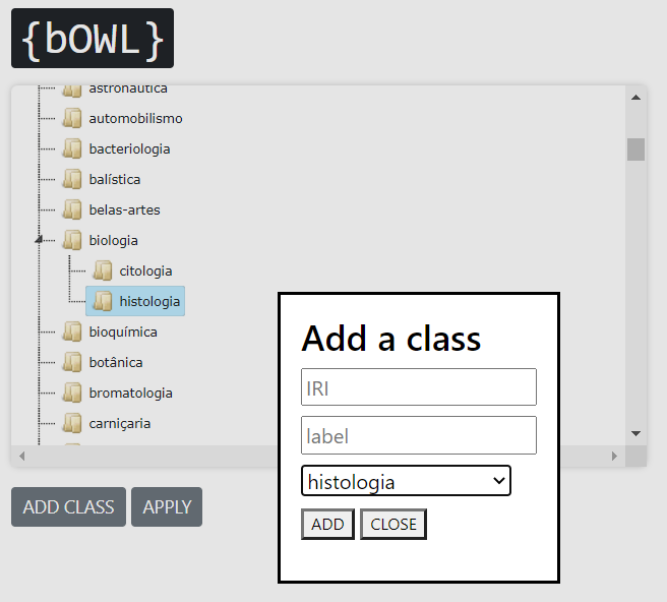


Figure 2 bOWL widget showing a (still flat) ontology from a dictionary.

5 Conclusions

At the moment bOWL is just a prototype. Nevertheless, we think it is useful for very different Web applications, and therefore, it should not be built tighten up with the underlying code. The library should be as versatile and independent as possible, in order to be pluggable in different scenarios.

As future work, we plan to improve the functionalities, but also OWL support. While some properties from the OWL data will be ignored (as they will not have a direct impact with the bOWL interface), we intend that bOWL should be able to store everything in order that, loading an ontology and saving it back is, always, the identify function.

References

1 Steven Harris and Andy Seaborne. SPARQL 1.1 query language. W3C recommendation, W3C, 2013. URL: `http://www.w3.org/TR/2013/REC-sparql11-query-20130321/`.

2 Natalya Fridman Noy, Monica Crubézy, Ray W Fergerson, Holger Knublauch, Samson W Tu, Jennifer Vendetti, and Mark A Musen. Protégé-2000: an open-source ontology-development and knowledge-acquisition environment. In *AMIA Annual Symposium Proceedings*, volume 2003, pages 953–953, 2003.

3 Alberto Simões, José João Almeida, and Ana Salgado. Building a Dictionary using XML Technology. In Marjan Mernik, José Paulo Leal, and Hugo Gonçalo Oliveira, editors, *5th Symposium on Languages, Applications and Technologies (SLATE)*, volume 51 of *OASIcs*, pages 14:1–14:8, Germany, 2016. Schloss Dagstuhl. `doi:10.4230/OASIcs.SLATE.2016.14`.

4 Alberto Simões, Ana Salgado, Rute Costa, and José João Almeida. LeXmart: A smart tool for lexicographers. In I. Kosem, T. Zingano Kuhn, M. Correia, J. P. Ferreira, M. Jansen, I. Pereira, J. Kallas, M. Jakubíček, S. Krek, and C. Tiberius, editors, *Electronic lexicography in the 21st century. Proceedings of the eLex 2019 conference*, pages 453–466, 2019.

5 Tania Tudorache, Jennifer Vendetti, and Natalya Fridman Noy. Web-protege: A lightweight owl ontology editor for the web. In *OWLED*, volume 432, page 2009, 2008.

6 W3C OWL Working Group. OWL 2 web ontology language document overview (2nd edition). W3C recommendation, World Wide Web Consortium, December 2012. URL: `http://www.w3.org/TR/2012/REC-owl2-overview-20121211/`.