# 6th Symposium on Languages, Applications and Technologies

**SLATE 2017, June 26–27, 2017, Vila do Conde, Portugal**

Edited by

Ricardo Queirós

Mário Pinto

Alberto Simões

José Paulo Leal

Maria João Varanda

*Editors*

Ricardo Queirós
School of Media Arts and Design
Polytechnic of Porto
Porto, Portugal
ricardoqueiros@esmad.ipp.pt

Mário Pinto
School of Media Arts and Design
Polytechnic of Porto
Porto, Portugal
mariopinto@esamd.ipp.pt

Alberto Simões
Technology School
Polytechnic of Cávado and Ave
Barcelos, Portugal
asimoes@ipca.pt

José Paulo Leal
Science's Faculty
University of Porto
Porto, Portugal
zp@dcc.fc.up.pt

Maria João Varanda
Technology and Management School
Polytechnic of Bragança
Bragança, Portugal
mjoao@ipb.pt

*ACM Classification 1998*
D.3 Programming Languages, H.3.3 Information Search and Retrieval, I.2.7 Natural Language Processing, I.7 Document and Text Processing

**ISBN 978-3-95977-056-9**

*Published online and open access by*
Schloss Dagstuhl – Leibniz-Zentrum für Informatik GmbH, Dagstuhl Publishing, Saarbrücken/Wadern, Germany. Online available at http://www.dagstuhl.de/dagpub/978-3-95977-056-9.

*Publication date*
October, 2017

*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 http://dnb.d-nb.de.



Digital Object Identifier: 10.4230/OASIcs.SLATE.2017.0

**ISBN 978-3-95977-056-9** **ISSN 1868-8969** **http://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 2190-6807**

**http://www.dagstuhl.de/oasics**

In my father's-in-law memory.
*Ricardo Queirós*

In my father's memory.
*Alberto Simões*

# Contents



6th Symposium on Languages, Applications and Technologies (SLATE 2017).
Editors: R. Queirós, M. Pinto, A. Simões, J. P. Leal, and M. J. Varanda
Open Access Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

## Human-Human Languages

# Preface

This book contains the proceedings of the sixth edition of SLATE, the 6th Symposium on Languages, Applications and Technologies, held at the Media Arts and Design School, from the Polytecnic of Porto, located at Vila do Conde, Portugal, during June 26–27, 2017.

In a global and increasingly technological society, communication is crucial for the sustained development of organizations. In this realm, languages are the cornerstone, allowing humans and machines to interact effectively to achieve common goals. In this sense we have the need to use languages in a way that allows communication between humans, between us and the computers, and finally, between machines. At SLATE we discuss these three types of languages by organizing the conference into 3 main tracks:

- HHL (Human-Human Languages): this track dedicates to the discussion of research projects and ideas involving natural language processing and their industrial application.
- HCL (Human-Computer Languages): this track aims to discuss topics on the latest academic or industrial work on language design, processing, assessment and applications.
- CCL (Computer-Computer Languages): the main goal of this track is to discuss topics related with mark-up and interchange formats and techniques, giving special importance to serialization languages such as XML and JSON

In this 6th edition we received 32 submissions (20 full papers and 12 short papers). After a peer-review process, in which each paper was reviewed at least by two anonymous reviewers, 23 papers were accepted (15 as full papers and 8 as short papers). In the overall process, we had 71% of paper acceptance rate. This book contains the revised versions of all the papers presented at SLATE 2017.

Finally, we would sincerely like to thank many people without whom this Conference would never have been possible. Firstly, to the Media Arts and Design School that received so well this event. Secondly, to our sponsors for believing in the importance of sharing scientifical knowledge on the languages topic. Also, a word of appreciation for the local organization of the event who truly engaged in a spirit of cooperation and cared for all the organization details. In this context, exceptionally, this year we count on the help of a designer student from ESMAD which managed the image of SLATE. Regarding the publish of the proceedings, we kindly regards the OASIcs for giving us the opportunity to publish this proceedings and MDPI journals for accepting submissions of additionally revised and extended journal-oriented versions of the best papers presented at the symposium. For the two invited keynoters, Mikel Forcada, and Daniela da Cruz, a big thank you for coming talking about interesting subjects, making the event more relevant. We want to thank the Program Committee members for their valuable contribution on reviewing the papers and the authors of the submitted papers for their contribution and interest in the symposium. At last, a sincere thank you to all those who participated in this symposium and who contributed to a new rebirth of this fantastic event.

*Ricardo Queirós*
*Mario Pinto*

**Event Chairs**

6th Symposium on Languages, Applications and Technologies (SLATE 2017).
Editors: R. Queirós, M. Pinto, A. Simões, J. P. Leal, and M. J. Varanda
Open Access Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

# Program Committee

### Main Chairs

Ricardo Queirós
Esc. Sup. de Media Artes e Design, IPP

Mário Pinto
Esc. Sup. de Media Artes e Design, IPP

### Track Chairs

Alberto Simões (HHL track)
Instituto Politécnico do Cávado e Ave

José Paulo Leal (CCL track)
Universidade do Porto

Maria João Varanda Pereira (HCL track)
Instituto Politécnico de Bragança

### Publication Chair

Alberto Simões
Instituto Politécnico do Cávado e Ave

### Local Organization Committee

Ricardo Queirós
Esc. Sup. de Media Artes e Design, IPP

Mário Pinto
Esc. Sup. de Media Artes e Design, IPP

Alberto Simões
Instituto Politécnico do Cávado e Ave

Carlos Filipe Portela
Universidade do Minho

### Program Committee

Salvador Abreu
Universidade de Évora

José João Almeida
Universidade do Minho

Jorge Baptista
Universidade do Algarve

Fernando Batista
Instituto Universitário de Lisboa

Mario Beron
Universidad Nacional de San Luis

Michele Bugliesi
University of Venice

João Paiva Cardoso
Universidade do Porto

Nuno Carvalho
Universidade do Minho

Gabriel David
Universidade do Porto

Brett Drury
NUI Galway

Luis Ferreira
Instituto Politécnico do Cávado e do Ave

Jean-Cristophe Filliâtre
Laboratoire de Recherche en Informatique

Mikel Forcada
Universitat d'Alacant

Niklas Fors
University of Lund

Pablo Gamallo
Universidade de Santiago de Compostela

Alda Lopes Gançarski
Institut National des Télécommunications

Marcos Garcia
Universidade da Coruña

Xavier Gómez Guinovart
Universidade de Vigo

Hugo Gonçalo Oliveira
Universidade de Coimbra

Pedro Rangel Henriques
Universidade do Minho

Jan Janousek
Czech Technical University

Geylani Kardas
Ege University

6th Symposium on Languages, Applications and Technologies (SLATE 2017).
Editors: R. Queirós, M. Pinto, A. Simões, J. P. Leal, and M. J. Varanda

Open Access Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Jan Kollar
Technical University of Kosice

Michal Kratky
VŠB-Technical University of Ostrava

Eugenijus Kurilovas
Centre of Information Technologies in Education

José Paulo Leal
Universidade do Porto

António Menezes Leitão
Universidade Técnica de Lisboa

Giovani Librelotto
Universidade Federal de Santa Maria

João Correia Lopes
Universidade do Porto

Ivan Lukovic
University of Novi Sad

Paulo Matos
Instituto Politécnico de Bragança

Marjan Mernik
Univerza v Mariboru

Alexander Paar
TWT GmbH Science & Innovation

Lluís Padró
Universitat Politècnica de Catalunya

Mário Pinto
Esc. Sup. de Media Artes e Design, IPP

Carlos Filipe Portela
Universidade do Minho & Instituto Politécnico do Porto

Jaroslav Porubän
Technická univerzita v Košiciach

Ricardo Queirós
Instituto Politécnico do Porto

Alexandre Rademaker
IBM Research

José Carlos Ramalho
Universidade do Minho

Cristina Ribeiro
Universidade do Porto

Ricardo Rocha
Universidade do Porto

Dietmar Seipel
University of Würzburg

José Luis Sierra
Universidad Complutense de Madrid

Josep Silva
Universidad Politècnica de Valencia

Alberto Simões
Universidade do Minho

Bostjan Slivnik
Univerza v Ljubljani

Peter Sloep
Open Universiteit

Kamel Smaili
Loria

Jakub Swacha
University of Szczecin

Kari Systa
Tampere University of Technology

António Teixeira
Universidade de Aveiro

Marco Temperini
Università di Roma

Jörg Tiedemann
Uppsala University

Juan-Manuel Torres-Moreno
Université d'Avignon

Maria João Varanda Pereira
Instituto Politécnico de Bragança

### Sub-Reviewers

Stephen Bradshaw
NUI Galway

Kim Nguyen
Laboratoire de Recherche Informatique

Falco Nogatz
University of Würzburg

# List of Authors


Ana de Almeida
ISCTE-IUL & CISUC
ISCTE Instituto Universitário de Lisboa,
Lisboa, Portugal
ana.almeida@iscte.pt

Miguel Areias
CRACS & INESC TEC
Fac. Ciências, Universidade do Porto
Porto, Portugal
miguel.areias@dcc.fc.up.pt

Patrícia Amorim Barros
Centro Algoritmi
Dept. Informática, Universidade do Minho
Braga, Portugal
bpatrcia@gmail.com

Jorge Baptista
INESC-ID Lisboa, $L^2F$
University of Algarve
Faro, Portugal
jbaptis@ualg.pt

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

Jorge Daniel Caldas
Centro Algoritmi
Dept. Informática, Universidade do Minho
Braga, Portugal
a67691@alunos.uminho.pt

Moharram Challenger
International Computer Institute
Ege University
Bornova, Izmir-Turkey
moharram.challenger@ege.edu.tr

Helder Correia
CRACS & INESC-TEC
Faculdade de Ciências
Universidade do Porto, Portugal
up201108850@fc.up.pt

Orhan Dagdeviren
International Computer Institute
Ege University
Bornova, Izmir-Turkey
orhan.dagdeviren@ege.edu.tr

José Devezas
INESC TEC & Faculdade de Engenharia
Universidade do Porto, Portugal
jld@fe.up.pt

Caglar Durmaz
International Computer Institute
Ege University
Bornova, Izmir-Turkey
caglar.durmaz@gmail.com

Alda Lopes Gançarski
Institut Telecom, Telecom SudParis
CNRS Sammovar, Evry, France
alda.gancarski@telecom-sudparis.eu

Hugo Gonçalo Oliveira
CISUC, Dept. of Informatics Engineering
University of Coimbra
Coimbra, Portugal
hroliv@dei.uc.pt

Ricardo Gonçalves
CRACS & INESC TEC
Fac. Ciências, Universidade do Porto
Porto, Portugal
rgoncalves@dcc.fc.up.pt

Hossein Hassani
University of Kurdistan Hewlêr
Erbi, Kurdistan Region, Iraq
hosseinh@ukh.edu.krd

Hawete Hattab
Umm Al-qura University
Al-Jumum University College
Makkah, KSA
hshattab@uqu.edu.sa

Pedro Rangel Henriques
Centro Algoritmi
Dept. Informática, Universidade do Minho
Braga, Portugal
prh@di.uminho.pt



6th Symposium on Languages, Applications and Technologies (SLATE 2017).
Editors: R. Queirós, M. Pinto, A. Simões, J. P. Leal, and M. J. Varanda
Open Access Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Milan Jančár
Department of Computers and Informatics
Technical University of Košice
Košice, Slovakia
milan.jancar@tuke.sk

Jan Janoušek Faculty of Information Technology
Czech Technical University in Prague
Praha 6, Czech Republic
jan.janousek@fit.cvut.cz

Geylani Kardas
International Computer Institute
Ege University
Bornova, Izmir-Turkey
geylani.kardas@ege.edu.tr

André Lara
Centro Algoritmi
Dept. Informática, Universidade do Minho
Braga, Portugal
a64362@alunos.uminho.pt

José Paulo Leal
CRACS & INESC-TEC
Faculdade de Ciências
Universidade do Porto, Portugal
zp@dcc.fc.up.pt

Maria Inês Maia
CRACS & INESC-TEC
Faculdade de Ciências
Universidade do Porto, Portugal
up201101593@fc.up.pt

Nuno Mamede
University of Lisboa
IST & INESC-ID Lisboa
Lisboa, Portugal
nuno.mamede@l2f.inesc-id.pt

Rabeb Mbarek
Sfax University, Multimedia Inf. Systems and Advanced Computing Laboratory
Sfax, Tunisia
rabeb.hattab@gmail.com

Pedro Miguel
Dept. of Electronics, Telecommunications and Informatics / IEETA
University of Aveiro, Portugal

Sérgio Nunes
INESC TEC & Faculdade de Engenharia
Universidade do Porto, Portugal
ssn@fe.up.pt

José Carlos Paiva
CRACS & INESC-Porto LA
Faculdade de Ciências
Universidade do Porto, Portugal
up201200272@fc.up.pt

Rui Pascoal
Instituto Universitário de Lisboa (ISCTE-IUL), Lisboa, Portugal

José Casimiro Pereira
Instituto Politécnico de Tomar
Tomar, Portugal
casimiro@ipt.pt

Joaquim Sousa Pinto
Dept. of Electronics, Telecommunications and Informatics / IEETA
University of Aveiro, Portugal

Jaroslav Porubän
Department of Computers and Informatics
Technical University of Košice
Košice, Slovakia
jaroslav.poruban@tuke.sk

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

Ricardo Ribeiro
ISCTE-IUL & INESC-ID
Lisboa, Portugal
ricardo.ribeiro@inesc-id.pt

Tom Ridge
University of Leicester
Leicester, United Kingdom
tr61@le.ac.uk

Ricardo Rocha
CRACS & INESC TEC
Fac. Ciências, Universidade do Porto
Porto, Portugal
ricroc@dcc.fc.up.pt

Mário Rodrigues
ESTGA/IEETA
University of Aveiro, Portugal
mjfr@ua.pt

Eliška Šestáková
Faculty of Information Technology
Czech Technical University in Prague
Praha 6, Czech Republic
eliska.sestakova@fit.cvut.cz

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

Matúš Sulír
Department of Computers and Informatics
Technical University of Košice
Košice, Slovakia
matus.sulir@tuke.sk

Jakub Swacha
Institute of Information Technology in
Management, University of Szczecin
Szczecin, Poland
jakubs@uoo.univ.szczecin.pl

António Teixeira
Dept. of Electronics, Telecommunications
and Informatics / IEETA
University of Aveiro, Portugal
ajst@ua.pt

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

John Whitington
University of Leicester
Leicester, United Kingdom
jw642@le.ac.uk

# Applying Attribute Grammars to Teach Linguistic Rules*

**Patrícia Amorim Barros[1], Maria João Varanda Pereira[2], and Pedro Rangel Henriques[3]**

1 **Departamento de Informática/Centro Algoritmi, Universidade do Minho, Braga, Portugal**
bpatrcia@gmail.com
2 **Departamento de Informática e Comunicações, IPB/Centro Algoritmi, Bragança, Portugal**
mjoao@ipb.pt
3 **Departamento de Informática/Centro Algoritmi, Universidade do Minho, Braga, Portugal**
pedrorangelhenriques@gmail.com

## Abstract

An attribute grammar is a very well known formalism to describe computer languages but it can also be successfully used to describe linguistic phenomena. Since natural languages can also be expressed in grammars it is natural to describe rules using the same formalism. Linguistic teachers of the University Complutense of Madrid started using attribute grammars but they lack a tool that helps them to specify linguistic rules in a friendly and natural way. Therefore we propose a domain specific language (NLSdsl) carefully designed for non-programmers that will be implemented on an AnTLR based system.

**1998 ACM Subject Classification** D.3.4 Processors

**Keywords and phrases** Attribute Grammars, DSL, Linguistics

**Digital Object Identifier** 10.4230/OASIcs.SLATE.2017.1

## 1 Introduction

Attribute Grammars were first developed with the intent of describing the semantics of context-free languages by Donald Knuth [5, 6]. The difference between an attribute grammar and a context-free grammar is basically that the attribute grammar provides context sensitivity using attributes and assigning them values,that are calculated using evaluation rules [3].

Even though when they first appeared their main purpose was to specify programming languages and to be used in compiler development [11], currently, attribute grammars have several types of applications due to their adaptability. They can be used to define languages, generate compilers, design and specify algorithms, etc. Linguistics is also an application of attribute grammars.

Attribute grammars may be used to specify the way sentences can be formed in a natural language. There are several rules in every idiom that define the way sentences can be correctly formed (e.g. number agreement between adjectives of one termination and nouns, etc.). Many rules that exist in natural languages can be specified with an attribute grammar. Resorting

* This work has been supported by COMPETE: POCI-01-0145-FEDER-007043 and FCT – Fundação para a Ciência e Tecnologia within the Project Scope: UID/CEC/00319/2013.


to some of the Attribute Grammar features it is possible to verify semantic and syntactic correction of any sentence [4]. Given a sentence, it is possible to determine if it is written correctly, and if not, where the errors occurred.

The idea of using attribute grammars to specify syntax and semantics of natural languages has been somewhat practiced. However the technique requires knowledge on programming to code the evaluation rules of the attribute grammar [3], as will be discussed in Section 2. Previous work falls short in making the tools available to the people who are most interested in using them: linguists, who in most of the cases don't have the knowledge necessary to program.

So, the work described in this paper is focused on a new pedagogical tool for linguistic students. The construction of a friendly interface based on a simple domain specific language allows the students to use attribute grammars without programming background. In order to understand linguistic rules, the students usually construct tree based structures and they use those structures to understand the sentence parts and the relations between those parts. In that sense, if they have a computational version of these specifications the restrictions they specify can be automatically verified. Moreover with an appropriate framework it would be very useful to visualize the tree and the attribute evaluation. Although this, as was said, the specification of attribute grammars in processing language tools is not an easy task for people without programming experience. In this paper, we propose an AnTLR based framework (see description in Section 7) that includes a new friendly interface, a new domain specific language to specify linguistic rules (introduced in Section 5 and illustrated in Section 6), an automatic way to verify the correction of the sentences, a set of useful visualizations that will help the students to understand the computational version of their linguistic rules. For that, linguistic exercises (described in Section 3, and implemented in AnTLR in Section 4) were analysed and an appropriate DSL was created to cope with them. The new DSL is called NLSdsl and an AnTLR based framework translate NLSdsl programs into AnTLR specifications. After that other facilities will be implemented: friendly interface and generated visualizations. For this work, several case studies drawn by teachers from University Complutense de Madrid and University of Minho were used. At the end, groups of linguistics students of these institutions will test the proposed approach and tool.

## 2 Related work

This section focus on describing and explaining tools and approaches that already use attribute grammars in teaching contexts.

Context free grammars (CFG) have been proposed and used by linguists since Chomsky's proposal to fundamentally describe the syntax of natural languages and there are tools to construct and visualize syntactic trees [2]. But CFGs are not used for syntactic or syntactic-semantic analysis because of the complexity of specifying the constraints of natural languages (like concordances) and semantic calculations. For this reason, the computational linguists need grammars that incorporate attributes to the terminal and non-terminal symbols and use some mechanism for checking constraints. Use a more complete formalism based on attribute grammars has proved to be much more effective from the didactic point of view than starting from more traditional formalisms of Linguistics [10, 13]. So, attribute grammars come from the language processing field and are not known or used by linguists despite their effectiveness in representing and conducting analysis guided by syntactic-driven analysis. Therefore, there are no effective tools that allow linguists to write and test grammars at a sufficiently high level of abstraction.

PAG is the tool currently being used by the linguistic students at Universidade Complutense de Madrid. PAG has the same objective as this work: make attribute grammars easier for linguists to specify. To achieve that goal they created a *Prolog* based framework in which linguists can specify their grammars in a specific language, directly execute their specifications and see the results in a decorated parse tree [14].

Linguistic students have deep knowledge of natural language so they produce good formal specifications but have serious difficulties in translating their knowledge to computer models, since they lack Computer Science skills. Hence the need to create a tool that facilitates this process. To use PAG, in addiction to writing the specification for the language, the user has to type some information in a user interface: the sentence he wants to analyze (it can be uploaded or written) and the values of the inherited attributes. Then this information is processed and the program makes all the decorated parse trees possible (there may be more than one) and notifies the user of all the errors (if any) that occurred. Students find the decorated parse trees very useful to understand ambiguity of sentences, since it allows them to see a table that shows attribute values for each node. Also, each entry of this table links to the corresponding semantic equation used to calculate the attribute. Notwithstanding that this tool solves some of the problems this group of students and teachers faced, we feel that we can improve this solution by making the specification of the rules even simpler and a much user friendlier appealing user interface including animated visualizations.

Outside the field of linguistics, three tools (that will be presented in the next paragraphs) were developed all of them aiming at simplify and help the attribute grammars teaching-learning process.

*EvDebugger* is a software tool created on 2014 based on attribute grammars for language specification with the purpose of helping Computer Science students with compiler construction [12]. Students usually need to design and develop language processors as their final project and they usually face difficulties defining the correct semantic equations. This happens because they have a hard time understanding the dependencies among attributes, in particular their evaluation order. *EvDebugger* helps them by providing a visual debugger that displays a syntax tree view and a table with the values of the attributes being computed. In addition to the debugger, this tool also offers a Grammar Manager and a Grammar Editor to facilitate the proccess of creating an attribute grammar to the students.

Hafix et al. [3] proposed a useful system to modelling natural-language phenomena which allows language processors to be created as executable attribute grammars. This is a system build in a purely-functional language (Haskell) where developers can specify semantic and syntactic descriptions of natural languages using attribute grammars. This is achieved by using a top-down analysis method that allows the production of a compact representation for ambiguous parses and the computation of meanings of sentences, using semantics. The language in which the specifications can be written is really simple and similar to AnTLR.

Another tool, called *VisualLISA*, allows attribute grammars to be written in a visual way (dragging and dropping shapes). Since it makes attribute grammars simpler to specify can also be used by linguistic students but in a visual manner, requiring less mental effort. *VisualLISA*[1] (*A Visual Programming Environment for Attribute Grammars*) was created in 2010 by Pedro Oliveira [9]. Users create drawings to specify their attribute grammars and *VisualLISA* allows them to verify at any time the structural and semantic correctness of the specification. The language of *VisualLISA* consists on a set of icons that can be conjugated. Nevertheless, *VisualLISA* doesn't fit properly enough the needs of our target users. Linguists

[1] `http://www4.di.uminho.pt/~gepl/VisualLISA/`

are not familiar with terms like terminal, non-terminal, computation rule, etc., therefore they may not be able to understand the meaning of all the icons used to design the attribute grammar.

To close the section, we will reference another tool, *CONSTRUCTOR* [1], that uses attribute grammars to help construct geometrical figures using natural language because it demonstrates how attribute grammars can be used in different fields. This tool allows users to input instructions for the construction of geometrical figures (in natural language) and using attribute grammars transforms those directives into actual geometric figures.

All of the tools demonstrate how attribute grammars can be used to help specify every kind of things: since geometrical figures to linguistic phenomena. Furthermore they demonstrate ways to make the specification of attribute grammars simpler.

## 3 Case Studies, description

In order to get familiar with the type of problems that linguists usually deal and need to solve and specify, some examples of exercises were obtained from the computational linguistic courses at UCM and also at UM. In concrete, the case studies 1,2 and 3 are based in the final project of the students Petra Horáková & Juan Pedro Cabanilles Gomar, Laura Canedo Caravaca & Lara de Santos Tabares and Alfredo Polves Luelmo of the Degree in Linguistics and Applied Languages of UCM. Those examples were carefully analysed and the solution was written in AnTLR. In this section we can find the explanation of these exercises.

The first case study to be implemented consisted in the syntactic analysis of prepositional phrases and their components: the preposition and the noun phrase. The grammar should be able to verify number and gender concordance between the components of the noun phrase (determinant, noun and adjective). In order to make this verification it was necessary to specify two attributes for the words to be used in the input sentences: number and gender. For example, the Portuguese sentence “O amiga dos irmã” would throw an error because it has multiple concordance errors, both in gender and number: The determinant “O” is masculine and singular, so it requires the subject to have the same attributes. That doesn’t happen because the noun “amiga” is feminine. The same happens with the determinant “dos” which is masculine and plural and doesn’t agree with the noun “irmã,” that is feminine and singular. For better understanding, the AnTLR code for this example will be presented and explained in the next section.

The second case study was related to the German language: in German each noun, pronoun and article has four cases: nominative, genitive, dative and accusative. These cases affect the way a sentence must be structured, and the types of verbs (movement or position) that can be used in it. This exercise required us to verify if a sentence was correctly structured taking into account if the case of the sentence agreed with the strucutre and with the type of the verb. For example, the German sentence “Er legt der Teller auf den Tisch” is wrong while the senctence “Er legt den Teller auf den Tisch” is right even though they both translate to the same result: “He puts the plate on the table.” The reason why the first sentence is incorrect is because the article “den” must be used only with accusative sentences and this sentence is nominative.

The third case study consisted in a syllabic divider: starting from a word already divided in syllables the exercise was to determine if the division was well made, and if not, determine why. For example, the input phrase “v-en-to” is not correctly divided by syllables and the correct division would be “ven-to.” In order to verify this it is necessary to analyze the type (vogal and consoant) and subtype (strong, weak. . . ) of every letter of the word, taking also into account its position in the word.

The fourth exercise was also related to noun and gender concordance, in this case for Portuguese sentences. In Portuguese, the noun determines the number and gender of the determinant and adjectives that are related to it. This kind of concordance in Portuguese can be hard to verify, specially when the components that need to concur are far apart in the sentences or when we need to deal with anaphora. For example, the sentence "Um especialista em fibras óticas ótimo" and the sentence "Um especialista em fibras óticas ótima"" are both equally correct. The challenge was to figure out with which one of the nouns the adjective must agree and verify the correctness of the sentence.

The fifth case study consisted in more semantic aspects of the Portuguese language: some verbs require nouns and complements of a certain type (animated or in-animated). This exercise required that we verified if the sentences respected those kind of constraints. For example, the sentence "O Carlos assusta a sinceridade" is incorrect because the verb "assustar" requires an animated complement, and it is not the case of the noun "sinceridade," while the sentence "A sinceridade assusta o Carlos" is correct, because "Carlos" is an animated noun. For better understanding, the AnTLR code for this example will be presented and explained in the next section.

The sixth and last case study was related to a Portuguese phenomenon called anaphora. An example of an anaphora can be found in the following sentence: "Eu gosto de **me** lavar com água quente." In this sentence, the word "de" is an anaphora. Anaphora must be connected to a local antecedent that is the subject of the sentence, in this case, "Eu." That means that for example the sentence "Eu gosto de **se** lavar com água quente" is incorrect, because the anaphora "se" is used with third person subjects, and "Eu" is in the first person.

The exercise requires us to verify if the anaphora in a sentence is correctly formed which means that agrees in noun and gender with the subject.

## 4 Case Studies, AnTLR implementation

For lack of space, in this section we only describe the resolution of the first and fifth case studies (introduced in the previous section) using AnTLR notation and technology.

### Case Study 1

The first example consists in analysing the syntax of *prepositional phrases* and their components: the `preposition` and the `noun` phrase. The grammar should be able to verify *number* and *gender* concordance between the components of the noun phrase (determinant, noun and adjective).

In order to make this verification it was necessary to specify two attributes (exemplified in the listing below) for the *words* (the lexicon members) that can be used in the input sentences: `number` and `gender`.

```
adjective returns[String gender, String number]
   : 'pequeno'  { $gender = "m"; $number = "s"; }
   | 'pequena'  { $gender = "f"; $number = "s"; }
   | 'pequenos' { $gender = "m"; $number = "p"; }
   | 'pequenas' { $gender = "f"; $number = "p"; }
   | 'maior'    { $gender = "i"; $number = "s"; }
   ;
```

Adding semantic actions (specially conditional statements) to the syntactic rules of the AnTLR context free grammar, it is possible to declare that the `gender` and `number` of the *determinant* and the *adjective* must concur with the `number` and `gender` of the *noun*.

```
sn : detPos noun prepositionalPhrase?
{
  if($detPos.number != $noun.number) { System.out.println("ERROR No
     number agreement between determinant and noun! ("+$detPos.text
     +","+$noun.text+")"); } }
| detPos noun adjective prepositionalPhrase?
{
  if($detPos.number != $noun.number) { System.out.println("ERROR No
     number agreement between determinant and noun! ("+$detPos.text
     +","+$noun.text+")"); }
  if($noun.number != $adjective.number) { System.out.println("ERROR No
      number agreement between noun and adjective! ("+$noun.text+","+
     $adjective.text+")"); }
  if(($noun.gender != $adjective.gender) && ($noun.gender != "i") && (
     $adjective.gender != "i")) { System.out.println("ERROR No gender
     agreement between noun and adjective! ("+$noun.text+","+
     $adjective.text+")"); } }
| detArt noun prepositionalPhrase?
{
  if($detArt.number != $noun.number) { System.out.println("ERROR No
     number agreement between determinant and noun! ("+$detArt.text
     +","+$noun.text+")"); }
  if(($detArt.gender != $noun.gender)  && ($detArt.gender != "i") && (
     $noun.gender != "i")) { System.out.println("ERROR No gender
     agreement between determinant and noun! ("+$detArt.text+","+$noun
     .text+")"); } }
| detArt noun adjective prepositionalPhrase?
{
  if($detArt.number != $noun.number) { System.out.println("ERROR No
     number agreement between determinant and noun! ("+$detArt.text
     +","+$noun.text+") "); }
  if($detArt.gender != $noun.gender) { System.out.println("ERROR No
     gender agreement between determinant and noun! ("+$detArt.text
     +","+$noun.text+") "); }
  if(($noun.number != $adjective.number)  && ($noun.gender != "i") &&
     ($adjective.gender != "i")) { System.out.println("ERROR No number
      agreement between noun and adjective! ("+$noun.text+","+
     $adjective.text+") "); }
  if(($noun.gender != $adjective.gender)  && ($noun.gender != "i") &&
     ($adjective.gender != "i")) { System.out.println("ERROR No gender
      agreement between noun and adjective! ("+$noun.text+","+
     $adjective.text+") "); } }
| noun prepositionalPhrase?
| noun adjective prepositionalPhrase?
{
  if($noun.number != $adjective.number) { System.out.println("ERROR No
      number agreement between noun and adjectiveetivo! ("+$noun.text
     +","+$adjective.text+")"); }
  if(($noun.gender != $adjective.gender)  && ($noun.gender != "i") &&
     ($adjective.gender != "i")) { System.out.println("ERROR No gender
      agreement between noun and adjective! ("+$noun.text+","+
     $adjective.text+")"); }
}
;
```

### Case Study 5

The second example consists in analysing the *type of subjects and complements that each verb accepts* in a sentence. The grammar should be able to verify if the subject and complement of a sentence agree with the verb, in a semantic way.

In order to make this verification it was necessary to specify only one attribute: the `type`. The next listing shows the AnTLR productions that assign an initial value the `type` attribute of the *verbs* and *nouns* (lexicon elements) that can be used in the input sentences.

```
verb returns[String type]
        : 'teme'     { $type = "subjAnimated";   }
        | 'assusta'  { $type = "complAnimated";  }
        | 'sucedeu'  { $type = "subjInanimated"; }
        | 'durou'    { $type = "complTemporal";  }
        ;

noun returns [String type]
        : 'Maria'       { $type = "animated";   }
        | 'sinceridade' { $type = "inanimated"; }
        | 'Carlos'      { $type = "animated";   }
        | 'homem'       { $type = "animated";   }
        | 'acidente'    { $type = "inanimated"; }
        | 'animal'      { $type = "animated";   }
        | 'reuniao'     { $type = "inanimated"; }
        | 'horas'       { $type = "temporal";   }
        ;
```

Now the approach is similar to the one followed to solve the previous case study. Again, adding semantic actions (specially conditional statements) to the syntactic rules of the AnTLR context free grammar, it is possible to declare that the `type` of the *subject* or *complement (direct or indirect)* must agree with the `type` of the *verb*: for example, if the verb type is `subjAnimated` it means that the subject of every sentence in which that verb appears must be of type '`animated`'.

```
predicate returns [String typeVerb, String typeComplDir, String
    typeComplInd,String verbTxt, String complementDirTxt, String
    complementIndTxt]
  : verb complementDir {
    $typeVerb          = $verb.type;
    $typeComplDir      = $complementDir.type;
    $verbTxt           = $verb.text;
    $complementDirTxt = $complementDir.text; }
  | verb complementInd {
    $typeVerb          = $verb.type;
    $typeComplInd      = $complementInd.type;
    $verbTxt           = $verb.text;
    $complementIndTxt = $complementInd.text; }
  | verb complementDir complementInd {
    $typeVerb          = $verb.type;
    $typeComplDir      = $complementDir.type;
    $typeComplInd      = $complementInd.type;
    $verbTxt           = $verb.text;
    $complementDirTxt = $complementDir.text;
    $complementIndTxt = $complementInd.text; }
  ;
```

```
sentence: subject predicate '.' {
  switch($predicate.typeVerb) {
    case("subjAnimated") :
      if($subject.type == "inanimated")
        System.out.println("ERROR! The verb " + $predicate.verbTxt + "
            requires an animated subject !");
      break;
    case ("complAnimated"):
      if($predicate.typeComplDir == "inanimated")
        System.out.println("ERROR! The verb " + $predicate.verbTxt + "
            requires an animated complement !");
      break;
    case ("subjInanimated"):
      if($subject.type == "animated")
        System.out.println("ERROR! The verb " + $predicate.verbTxt + "
            requires an inanimated subject!");
      break;
    case ("complTemporal"):
      if(($predicate.typeComplDir != "complTemporal") && ($predicate.
          typeComplDir != null))
        System.out.println("ERROR! The verb " + $predicate.verbTxt + "
            requires a temporal complement!");
      if(($predicate.typeComplInd != "complTemporal") && ($predicate.
          typeComplInd != null))
        System.out.println("ERROR! The verb " + $predicate.verbTxt + "
            requires a temporal complement!");
      break;
  }
};
```

## 5 NLSdsl

A Domain Specific Language (DSL) is a language designed to describe a specific domain [7, 8]. In this section a new Domain Specific Language for linguistic rules specification will be introduced. Its main purpose is to turn easier the rules specification avoiding the complexity of the AnTLR code for non-programmer users.

The first step needed to specify our Domain Specific Language was to define what would be the main characteristics of the language:

- It needed to be intuitive for non-programmers;
- It needed to be as close as possible to natural language;
- It couldn't have many programming elements (such as keys, semicolons, etc.);
- It should follow the functional notation instead of the object oriented (it is more intuitive for non-programmers to understand *function(argument)* than *function.argument*).

Taking these items into account it was possible to make the first sketch of the new DSL, NLSdsl, defined and explained below – our proposal to leverage the use of attribute grammars by Linguists.

```
grammar NLSdsl;
   specification : (grammarRule)+;
```

The first rule of the DSL says that the specification should be formed by at least one grammar rule. Then, a grammar rule defines the components of a sentence (one or more symbols), a possible set of calculations (`evalRule`) and a possible set of conditions (ifthen statements).

```
grammarRule: ntSymb':'(symbol('?')?)+ (evalRule)* (condition)*;
```

The symbols can be terminals or non-terminals in the sense that can be used to define other rules or not. A terminal symbol must be written using uppercase letters.

```
symbol : ntSymb | tSymb;
ntSymb : PAL;
tSymb  : PALCAPIT;
```

`evalRule` is where the value of the attributes is evaluated, when necessary.

```
evalRule : '->' attName '=' expr ('&' attName '=' expr)*;
```

The evaluation rule must start with the character '->' and must be formed by an attribute name followed by the '=' sign and an expression that determines the value the attribute will take. One or more attributes can be defined. The expression for now may only be a word, a number or an attribute, because in all the examples we tested there was no need for more complex expressions.

```
expr  : attrib | PAL | NUM;
```

An attribute is the way to obtain the value of a symbol's attribute. It is formed by the attribute name followed by the symbol between brackets.

```
attrib : attName '(' symbol ')';
```

Returning to the first rule of the grammar, the conditions refer to rules that must be verified and if they turn out to be false an error message must be printed.

```
condition : '=>' logicExp errorMsg;
```

A condition must start with the character '=>' and then have a logic expression and an error message. The error message should only be printed if the logic expression evaluates false.

A logic expression is composed of one or more relations connected by Boolean operators.

```
logicExp : rel (opBOOL rel)*;
opBOOL   : 'AND' | 'OR';
rel      : attrib opREL expr;
```

A relation is formed by an attribute followed by a relational operator and an expression. The error message must be written between two exclamation marks and can contain various elements (maybe strings, attribute names, or symbol names) aggregated by the concatenation operator '+'.

```
errorMsg : '!' elem ('+' elem)* '!';
elem     : STR | attrib | symbol;
```

This is useful to print attribute values or symbols embedded in the error message to make clearer what is wrong in the input sentence.

## 6 Revisiting the Case Studies of Section 4 with NLSdsl

For a better understanding of the practical application of the DSL we proposed in the previous section, we revisit in this section the two case studies discussed in Section 4, and show excerpts of what their specification in NLSdsl would look like. We also show the textual output that our tool (see Section 7) would produce when executed with an input sentence, and the decorated parse tree it would generate.

### Case Study 1

Recalling that the first case study presented is concerned with *noun and gender concordance* in a prepositional sentence, part of the NLSdsl specification for this linguistic phenomenon is shown in the listing below.

```
prepositionalPhrases : (prepositionalPhrase '.')*

prepositionalPhrase : Preposition nominalPhrase
prepositionalPhrase : Contraction nominalPhrase

nominalPhrase : noun prepositionalPhrase?;
nominalPhrase : noun adjective prepositionalPhrase
  => number(noun) == number(adjective)
      ! ERROR, no number agreement between noun a adjective !

  => gender(noun) == gender(adjective)
      ! ERROR, no gender agreement between noun e adjective !

nominalPhrase : positionDeterminant noun prepositionalPhrase?
  => number(positionDeterminant) == number(noun)
      ! ERROR, no number agreement between noun e determinant !

nominalPhrase : positionDeterminant noun adjective
    prepositionalPhrase?
  => number(positionDeterminant) == number(noun)
          ! ERROR, no number agreement between noun e determinant !

  => number(noun) == number(adjective)
          ! ERROR, no number agreement between noun a adjective !

  => gender(noun) == gender(adjective)
          ! ERROR, no gender agreement between noun e adjective !

adjective: PEQUENO  -> gender = masculine & number = singular
adjective: PEQUENA  -> gender = feminine  & number = singular
adjective: PEQUENOS -> gender = masculine & number = plural
adjective: PEQUENAS -> gender = feminine  & number = plural
adjective: MAIOR    -> gender = undefined & number = singular
```

The output of our tool when invoked with the input `"Para teu irmãs pequeno."` (sentence one) would be the following:

```
ERROR No number agreement between determinant and noun!(teu,irmas)
ERROR No number agreement between noun and adjective!(irmas,pequeno)
ERROR No gender agreement between noun and adjective!(irmas,pequeno)
```

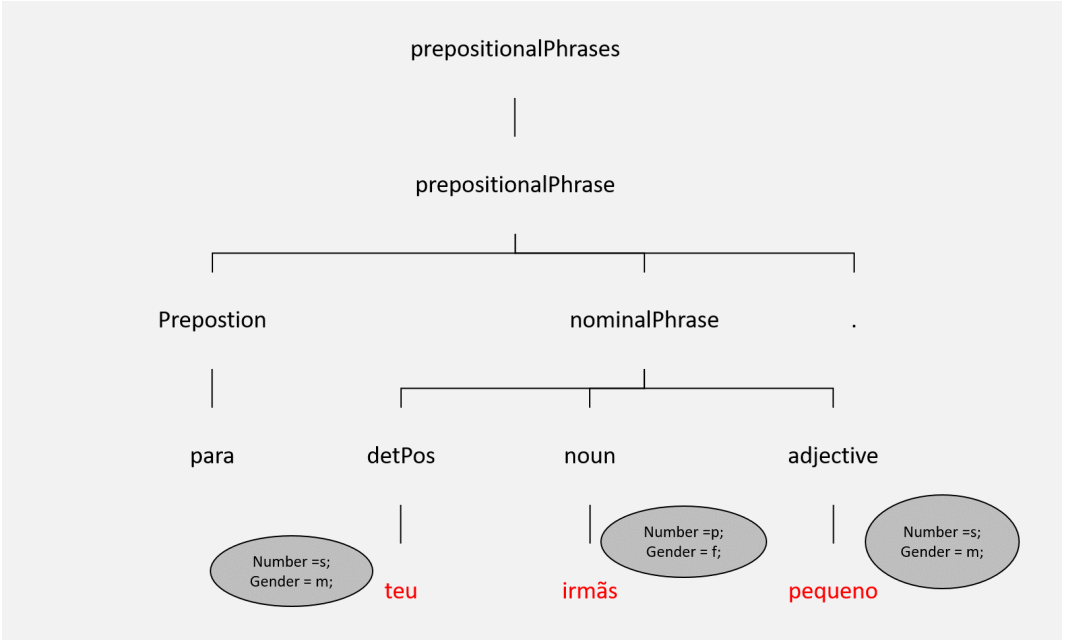



**Figure 1** Attributed Parse Tree generated by our tool for case study one, sentence one.

And the parsing tree that complementary would be generated by our tool, decorated with attribute values in each node and colored (in red) to enhance the nodes where errors where detected, is shown in Figure 1.

## Case Study 5

As previously said, the fifth case study is related to *the type of verbs and the type of complements and subjects they require*. So, part of the NLSdsl specification for this linguistic phenomenon is shown in the listing below.

```
sentence : subject predicate '.'
 => typeVerb(predicate) == "subjAnimated" && type(subject) == "
    animated"
        ! ERROR, The verb requires an animated subject !

 => typeVerb(predicate) == "complAnimated" && typeComplDir(predicate)
    == "animated"
        ! ERROR, The verb requires an animated complement !

 => typeVerb(predicate) == "subjInanimated" && type(subject) == "
    inanimated"
        ! ERROR, The verb requires an inanimated subject !

 => typeVerb(predicate) == "complTemporal" && typeComplDir(predicate)
    == "temporal"
        ! ERROR, The verb requires a temporal complement !

 => typeVerb(predicate) == "complTemporal" && typeComplInd(predicate)
    == "temporal"
        ! ERROR, The verb requires a temporal complement !
```

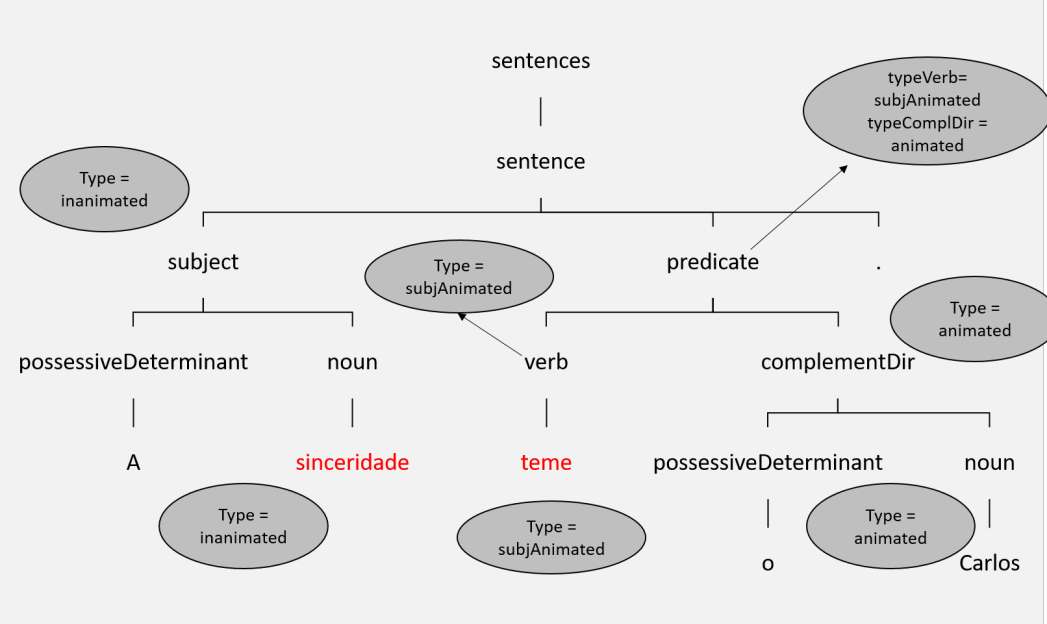



**Figure 2** Attributed Parse tree generated by our tool for case study five, sentence two.

The output of our tool when invoked with the input `A sinceridade teme o Carlos.` (sentence two) would be the following:

```
ERROR! The verb 'teme' requires an animated subject !
```

The parsing tree generated, with attribute values decorating its nodes and red color to enhance the nodes where errors where detected, is presented in Figure 2.

As we can see in Figure 2, there is an error in the sentence because the word `sinceridade` does not comply with the requirement of the verb `teme` of being of type `animated`.

## 7 Our tool: description and architecture

This section presents our tool, based on the description of the system architecture. Figure 3 represents the architecture of our system.

The central component is a translator that takes a Domain Specific Language specification and transforms it into the equivalent AnTLR specification.

This translator is generated by AnTLR using the NLSdsl grammar (described in Section 5). This process is straight-forward and does not deserve a more detailed explanation here. Actually, the core of this translation process is the set of rules that map NLSdsl constructors into the AnTLR attributed productions, but this contribution (under work) raise up directly from the handwritten examples (like the ones presented in Section 4).

After that process (transformation of an NLSdsl specification into the AnTLR grammar) is completed, a compiler is automatically constructed by AnTLR to cope with sentences written in L Language. Then the user can input a sentence to test and this new compiler produces the sentence verification assessment (an Ok or an error message) and a tree based visualization to show, more clearly, the structure of the input sentence locating and enhancing the error occurrences in the appropriate tree nodes. Some examples were presented in Section 6.

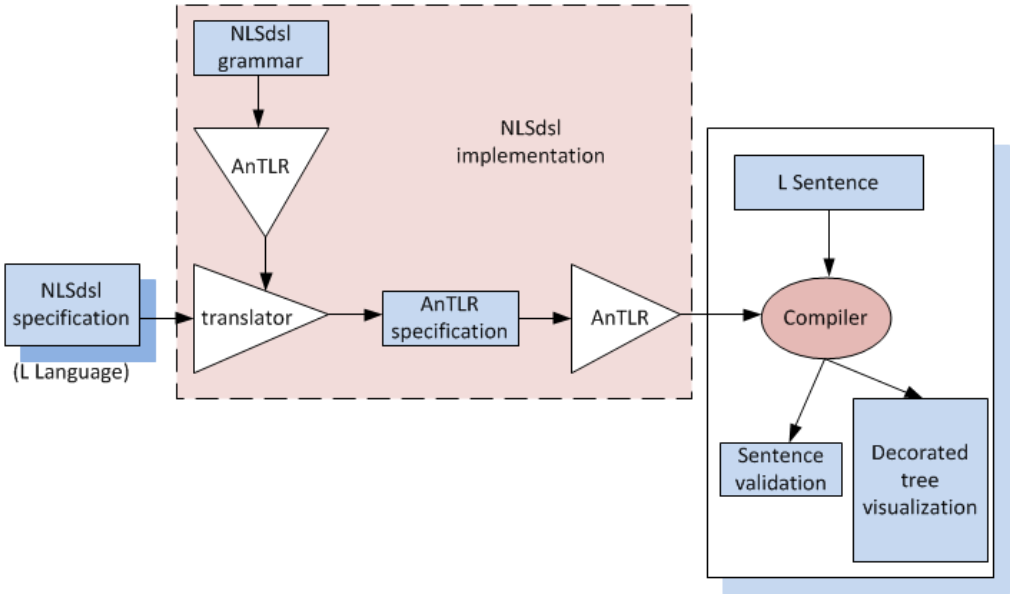



**Figure 3** System Architecture.

The use of the NLSdsl to describe a linguistic rule and the subsequent validation process of concrete sentences allows the linguistics' students to better understand those rules and the errors that occur when they are not obeyed. In our opinion the decorated tree visualization (also under development) will be crucial to aid in the understanding of those rules and their verification process.

## 8 Conclusion

Even though the objective of the master's thesis project, underlying the work here discussed, is to develop the entire system described in this document, in this paper we focused on the three contributes below:

- the specification in AnTLR of several natural language case studies in order to understand how these linguistic phenomena can be formally described, as explained in Section 4;
- the creation of NLSdsllanguage specifically designed to be intuitive for non-programmers but still powerful enough to express the linguistic phenomena required, thoroughly explained in Section 5;
- the proposal of a system architecture, described in Section 7, that aggregates all of the components to develop.

As future work the development of the NLSdsl to AnTLR translator will be finished and tested for sentence validation.Then, after implementing the visualization module, a set of experiments in classroom will be conducted in order to assess the usability and efectiveness of new tool. These experiments will take place at the beginning of the next school year at UCM.

**Acknowledgements.** We are deeply indebt to Ana Fernandez-Pampillon and Jose Luis Sierra, from Universidade Complutense de Madrid, for introducing us to this topic, challenging us to cooperate in the linguistic project, and also for all for the discussion sustained and all the material/examples provided.

## References


1 Zoltán Alexin, József Dombi, Károly Fábricz, and Tibor Gyimóthy. CONSTRUCTOR: A natural language interface based on attribute grammars. *Acta Cybernetica*, 9(3):247–255, 1990.

2 Steven Bird, Ewan Klein, and Edward Loper. *Natural Language Processing with Python*. O'Reilly Media, Inc., 1st edition, 2009.

3 Rahmatullah Hafiz and Richard A. Frost. Modular natural language processing using declarative attribute grammars. In Ildar Batyrshin and Grigori Sidorov, editors, *Advances in Artificial Intelligence: MICAI 2011*, pages 291–304. Springer Berlin Heidelberg, 2011. doi:10.1007/978-3-642-25324-9_25.

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

5 Donald E. Knuth. Semantics of context-free languages. *Mathematical Systems Theory*, 2(2):127–145, 1968.

6 Donald E. Knuth. The genesis of attribute grammars. In *Proceedings of the international conference on Attribute grammars and their applications*, pages 1–12. Springer-Verlag, 1990.

7 Tomaž Kosar, Sudev Bohra, and Marjan Mernik. Domain-specific languages: A systematic mapping study. *Information and Software Technology*, 71:77–91, 2016.

8 Marjan Mernik, Jan Heering, and Anthony M. Sloane. When and how to develop domain-specific languages. *ACM Comput. Surv.*, 37(4):316–344, December 2005.

9 Nuno Oliveira, Maria João Varanda Pereira, Pedro Rangel Henriques, Daniela da Cruz, and Bastian Cramer. VisualLISA: A visual environment to develop attribute grammars. *Computer Science an Information Systems (Special issue on Advances in Languages, Related Technologies and Applications)*, 7(2):266–289, May 2010.

10 Fernando. Pereira and David H. D. Warren. Definite clause grammars for language analysis – a survey of the formalism and a comparison with augmented transition networks. *Artificial Intelligence*, 13:231–278, 1980.

11 Daniel Ravan. *A graphical structure-editor that generates code for attribute grammar systems*. PhD thesis, University of Windsor, 1995.

12 Daniel Rodríguez-Cerezo, Pedro Rangel Henriques, and José-Luis Sierra. Attribute grammars made easier: EvDebugger a visual debugger for attribute grammars. In *International Symposium on Computers in Education (SIIE)*, pages 23–28. IEEE, 2014.

13 Stuart Merrill Shieber. *An introduction to unification-based approaches to grammar*. CSLI Publications, Stanford, California, 1986.

14 José-Luis Sierra, Ana María Fernández-Pampillon, and Alfredo Fernández-Valmayor. An environment for supporting active learning in courses on language processing. *SIGCSE Bull.*, 40(3):128–132, June 2008.

# Towards an Automated Test Bench Environment for Prolog Systems*

**Ricardo Gonçalves[1], Miguel Areias[2], and Ricardo Rocha[3]**

1 **CRACS & INESC TEC and Faculty of Sciences, University of Porto, Porto, Portugal**
rgoncalves@dcc.fc.up.pt

2 **CRACS & INESC TEC and Faculty of Sciences, University of Porto, Porto, Portugal**
miguel-areias@dcc.fc.up.pt

3 **CRACS & INESC TEC and Faculty of Sciences, University of Porto, Porto, Portugal**
ricroc@dcc.fc.up.pt

## Abstract

Software testing and benchmarking is a key component of the software development process. Nowadays, a good practice in big software projects is the *Continuous Integration (CI)* software development technique. The key idea of CI is to let developers integrate their work as they produce it, instead of doing the integration at the end of each software module. In this paper, we extend a previous work on a benchmark suite for the Yap Prolog system and we propose a fully automated test bench environment for Prolog systems, named *Yet Another Prolog Test Bench Environment (YAPTBE)*, aimed to assist developers in the development and CI of Prolog systems. YAPTBE is based on a cloud computing architecture and relies on the Jenkins framework and in a set of new Jenkins plugins to manage the underneath infrastructure. We present the key design and implementation aspects of YAPTBE and show its most important features, such as its graphical user interface and the automated process that builds and runs Prolog systems and benchmarks.

**1998 ACM Subject Classification** D.2.5 Testing and Debugging, D.1.6 Logic Programming

**Keywords and phrases** Software Engineering, Program Correctness, Benchmarking, Prolog

**Digital Object Identifier** 10.4230/OASIcs.SLATE.2017.2

## 1 Introduction

In the early years of software development, it was a well-known rule of thumb that in a typical software project approximately 50 percent of the elapsed time and more than 50 percent of the total cost were spent in benchmarking the components of the software under development. Nowadays, despite the new development systems and languages with built-in tools, benchmarking still plays an important role in any software development project. Software benchmarking is a process, or a series of processes, designed to make sure that computer code does what it was designed to do and, conversely, that it does not do anything

* This work was funded by the ERDF (European Regional Development Fund) through Project 9471 – *Reforçar a Investigação, o Desenvolvimento Tecnológico e a Inovação (Projeto 9471-RIDTI)* – and through the COMPETE 2020 Programme within project POCI-01-0145-FEDER-006961, and by National Funds through the FCT (Portuguese Foundation for Science and Technology) as part of project UID/EEA/50014/2013. Miguel Areias was funded by the FCT grant SFRH/BPD/108018/2015.





unintended [15]. Software benchmarking techniques can be broadly classified into *white-box benchmarking* and *black-box benchmarking*. The former refers to the structural benchmarking technique that designs test cases based on the information derived from source code. The latter, also called data-driven or input/output driven benchmarking, views the program as a black box, and its goal is to be completely unconcerned about the internal behavior and structure of the program and, instead, it concentrates on finding circumstances in which the program does not behave according to the specifications [15]. Nowadays, a good practice in big software projects is the *Continuous Integration (CI)* software development technique [9]. The key idea of CI is to let developers integrate their work as they produce it, instead of doing the integration at the end of each software module. Each integration is then verified by an automated benchmark environment which ensures the correctness of the integration or detect the integration errors. One of the greatest advantages of the CI is an earlier detection of errors, leading to smaller and less complex error corrections.

Prolog is a language with a long history whose community has seen a large number of implementations which evolved independently. This situation is totally different from more recent languages, such as, Java, Python or Perl, that either have a single implementation (Python, Perl) or are controlled centrally (Java implementations can only be called Java if they satisfy certain standards). The international standard for Prolog ISO/IEC 13211 [11] was created to standardize Prolog implementations. However, due to the different sources of development, the standard is not completely implemented in most Prolog systems. The Prolog community knows that different Prolog systems have different dialects with different syntax and different semantics for common features. A good example is Wielemaker's recent work on dictionaries and new string extensions to Prolog [22], which are not part of the ISO/IEC 13211. A different direction is the one followed by Wielemaker and Santos Costa [20, 21], where they studied the status of the standardization of Prolog systems and gave a first step towards a new era of Prolog, where all systems are fully compliant with each other. While this new era is not reached yet, every publicly available significant piece of Prolog code must be carefully examined for portability issues before it can be used in any Prolog system. This creates a significant obstacle, if one wants to compare Prolog systems in performance and/or correctness measurements.

Benchmark suite frameworks for Prolog have been around for some time [6, 10] and several still exist that are specially aimed to evaluate Prolog systems. Two good examples are China [5] and OpenRuleBench [12]. China is a data-flow analyzer for constraint logic programming languages written in C++ which performs bottom-up analysis deriving information on both call-patterns and success-patterns by means of program transformations and optimized fix-point computation techniques. OpenRuleBench is an open community resource designed to analyze the performance and scalability of different rule engines in a set of semantic web information benchmarks.

In previous work, we have also developed a first benchmark suite framework based in the CI and black-box approaches to support the development of the Yap Prolog system [17]. This framework was very important for our work [1, 2], mainly to ensure Yap's correctness in the context of several improvements and new features added to its tabling engine. The framework handles the comparison of outputs obtained through the run of benchmarks for general Prolog queries and for the answers stored in the table space if using tabled evaluation. It also supports the different Prolog dialects of the XSB Prolog [16] and B-Prolog [23] systems. However, the framework still lacks important user productive features such as automation and a powerful graphical user interface.

In this paper, we extend such a previous work and we propose a fully automated test bench environment for Prolog systems, named *Yet Another Prolog Test Bench Environment*

*(YAPTBE)*, aimed to assist developers in the development and integration of Prolog systems. YAPTBE is based in a cloud computing architecture and relies in Jenkins [18] and in a set of new Jenkins plugins to manage the underneath infrastructure. Arguably, Jenkins is one of the most successful open source automation tools to manage a CI infrastructure. Jenkins, originally called Hudson, is written in Java, provides hundreds of plugins to support building, deploying and automating any project, and is used by software teams of all sizes, for projects in a wide variety of languages and technologies.

YAPTBE includes the following features: (i) a graphical user interface which coordinates all the interactions with the test bench environment; (ii) the definition of a cloud computing environment including different computing nodes running different operating systems; (iii) an automated process to synchronize, compile and run Prolog systems against sets of benchmarks; (iv) an automated process to handle the comparison of output results and store them for future reference; (v) a smooth integration with state-of-the-art version control systems such as GIT; (vi) a publicly available online version that allows anonymous users to interact with the environment to follow the state of the several Prolog systems. To be best of our knowledge, YAPTBE is the first environment specially aimed for Prolog systems that supports all such features. For simplicity of presentation, we will focus our description in the Yap Prolog system, but YAPTBE can be used with any other system.

The remainder of the paper is organized as follows. First, we briefly introduce some background about Prolog and tabled evaluation. Next, we discuss the key ideas of YAPTBE and how it can be used to support the development and evaluation of Prolog systems. Then, we present the key design and implementation details and we show a small test-drive over YAPTBE. Finally, we outline some conclusions and indicate further working directions.

## 2 Background

Arguably, one of the most popular logic programming languages is the Prolog language. Prolog has its origins in a software tool proposed by Colmerauer in 1972 at *Université de Aix-Marseille* which was named *PROgramation en LOGic* [8]. In 1977, David H. D. Warren made Prolog a viable language by developing the first compiler for Prolog. This helped to attract a wider following to Prolog and made the syntax used in this implementation the *de facto* Prolog standard. In 1983, Warren proposed a new abstract machine for executing compiled Prolog code that has come to be known as the Warren Abstract Machine, or simply WAM [19]. The WAM became the most popular way of implementing Prolog and almost all current Prolog systems are based on WAM's technology.

A logic program consists of a collection of Horn clauses. Using Prolog's notation, each clause may be a rule of the form:

$$a(\vec{X_0}) :- b_1(\vec{X_1}),\ b_2(\vec{X_2}),\ ...,\ b_n(\vec{X_n}).$$

where $a(\vec{X})$ is the head of the rule, $b_i(\vec{X_i})$ are the body subgoals and $\vec{X_i}$ are the subgoals' arguments, or it may be a fact (without body subgoals) and simply written as:

$$a(\vec{X_0}).$$

The symbol :− represents the logic implication and the comma (,) between subgoals represents logic conjunction, i.e., rules define the expression:

$$b_1(\vec{X_1}) \wedge b_2(\vec{X_2}) \wedge ... \wedge b_n(\vec{X_n}) \Rightarrow ga(\vec{X_0})$$

while facts assert $a(\vec{X_0})$ as true.

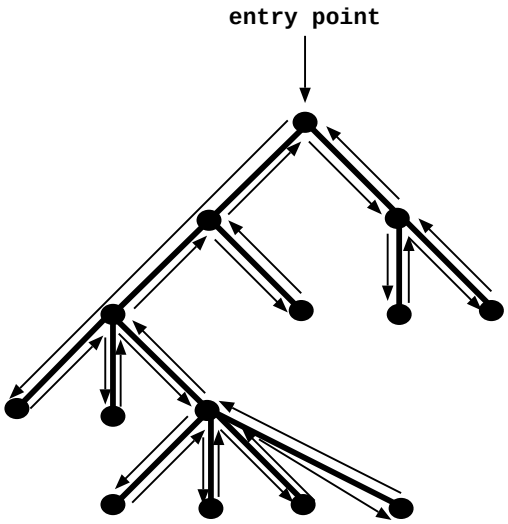



**Figure 1** Depth-first left-to-right search with backtracking in Prolog.

Information from a logic program is retrieved through query execution. Execution of a query $Q$ with respect to a program $P$ proceeds by reducing the initial conjunction of subgoals in $Q$ to subsequent conjunctions of subgoals according to a refutation procedure called SLD resolution [13]. Figure 1 shows a pure and sequential SLD evaluation in Prolog, which consists in traversing a search space in a *depth-first left-to-right* form. Non-leaf nodes of the search space represent stages of computation (*choice points*) where alternative branches (clauses) can be explored to satisfy the program's query, while leaf nodes represent solution or failed paths. When the computation reaches a failed path, Prolog starts the *backtracking* mechanism, which consists in restoring the computation up to the previous non-leaf node and schedule an alternative unexplored branch.

SLD resolution allows for efficient implementations but suffers from some fundamental limitations in dealing with recursion and redundant sub-computations. Tabling [7] is a refinement of Prolog's SLD resolution that overcomes some of those limitations. Tabling is a kind of dynamic programming implementation technique that stems from one simple idea: save intermediate answers for current computations in an appropriate data area, called the *table space*, so that they can be reused when a *similar computation* appears during the resolution process. With tabling, similar calls to tabled subgoals are not re-evaluated against the program clauses, instead they are resolved by consuming the answers already stored in the corresponding table entries. During this process, as further new answers are found, they are stored in their tables and later returned to all similar calls. Figure 2 shows the evaluation of a tabled program.

The top left corner of the figure shows the program code and the top right corner shows the final state of the table space. The program defines a small directed graph, represented by two *edge/2* facts, with a relation of reachability, defined by a *path/2* tabled predicate. The bottom of the figure shows the evaluation sequence for the query goal *path(a,Z)*. Note that traditional Prolog would immediately enter an infinite loop because the first clause of *path/2* leads to a similar call (*path(a,Y)* at step 1).

First calls to tabled subgoals correspond to *generator nodes* (depicted by white oval boxes) and, for first calls, a new entry representing the subgoal is added to the table space (step 0). Next, *path(a,Z)* is resolved against the first *path/2* clause calling, in the continuation,

```
:- table path/2.

path(X,Z) :- path(X,Y), edge(Y,Z).
path(X,Z) :- edge(X,Z).

edge(a,b).
edge(b,a).
```

Table

| subgoals | answers |
|---|---|
| 0. path(a,Z) | 3. Z=b<br>5. Z=a |

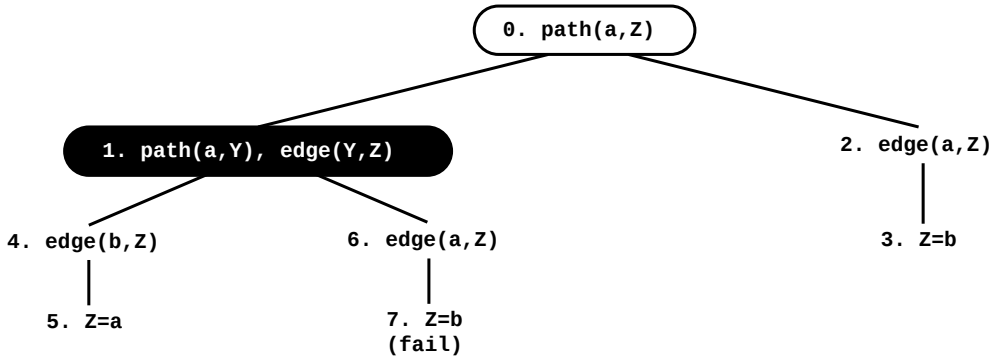



**Figure 2** An example of a tabled evaluation.

*path(a,Y)*. Since *path(a,Y)* is a similar call to *path(a,Z)*, the engine does not evaluate the subgoal against the program clauses, instead it consumes answers from the table space. Such nodes are called *consumer nodes* (depicted by black oval boxes). However, at this point, the table does not have answers for this call, so the computation is suspended (step 1). The only possible move after suspending is to backtrack and try the second clause for *path/2* (step 2). This originates the answer $\{Z{=}b\}$, which is then stored in the table space (step 3). At this point, the computation at node 1 can be resumed with the newly found answer (step 4), giving rise to one more answer, $\{Z{=}a\}$ (step 5). This second answer is then also inserted in the table space and propagated to the consumer node (step 6), which originates the answer $\{Z{=}b\}$ (step 7). This answer had already been found at step 3. Tabling does not store duplicate answers in the table space and, instead, repeated answers *fail*. This is how tabling avoids unnecessary computations, and even looping in some cases. Since there are no more answers to consume nor more clauses left to try, the evaluation ends and the table entry for *path(a,Z)* can be marked as *completed*.

For our test bench environment, tabling plays an important role because, with tabling, we might want to handle not only the comparison of outputs obtained through the run of general Prolog queries, but also the comparison of the structure/configuration of the tables stored during such executions. Moreover, if we table all predicates involved in a computation, we can use tabling as a way to keep track of all intermediate subcomputations that are done for a particular top query goal. Tabling can thus be used as a built-in powerful tool to check and ensure the correctness of the Prolog engine internals. To take advantage of tabling, we thus need to design our test bench environment to take into account the kind of output given by tabling.

## 3 Yet Another Prolog Test Bench Environment

In this section, we introduce the key concepts about YAPTBE's design.

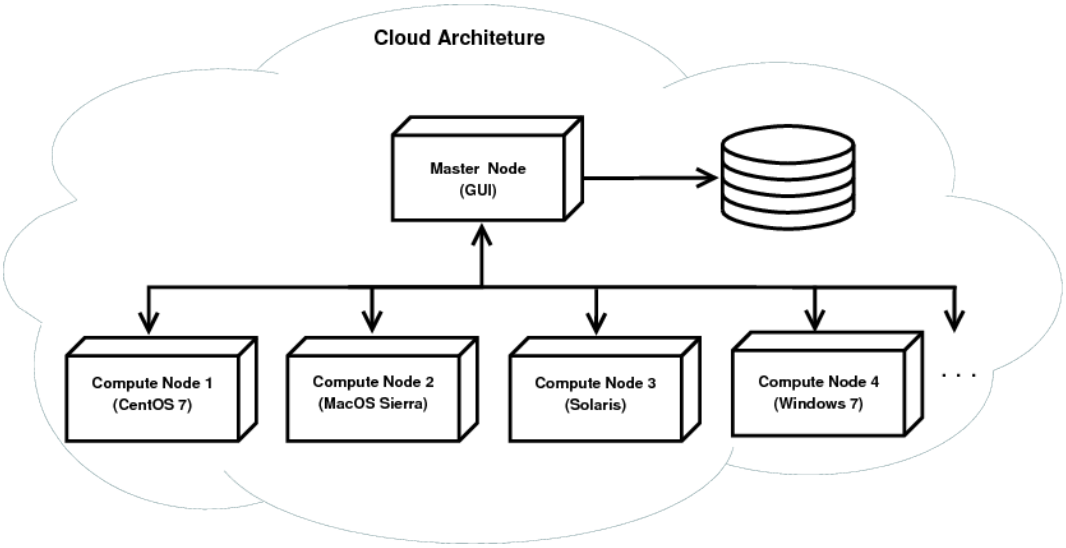



**Figure 3** The cloud-based architecture of YAPTBE.

## 3.1 Cloud-Based Architecture

In the early years of software development, a piece of software was designed having in mind a specific operating system and hardware architecture. As time passed by, operating systems and hardware architectures became more sophisticated and branched out into a multiplicity of platforms and versions, which are often variations of the same software or hardware component. In order to go along with this new reality, nowadays, whenever a new piece of software is designed, developers must ensure that it will work correctly in different operating systems and hardware architectures. Fortunately, cloud computing has emerged as an excellent alternative for software development. Cloud computing is very powerful because it provides ubiquitous access to multiple operating systems, heterogeneous and non-heterogeneous hardware architectures, which can be seen and manipulated as being similar resources. In what follows, we explain how we tried to bring the advantages of cloud computing into YAPTBE's design.

Figure 3 shows a general perspective of YAPTBE's cloud-based architecture. At the entry point, a *master node* with a *Graphical User Interface (GUI)* allows users to interact with YAPTBE's cloud-based infrastructure. The master node is then connected, through an intranet connection, to a *storage device* (shown at right in Fig. 3), which stores and backups all relevant information, and to several *computing nodes* (or *slave nodes*) which can be connected through an intranet or internet connection, depending if they are or not close enough to the master node. Each computing node has its own version of an operating system. In Fig. 3, we can see four computing nodes running the *CentOS 7*, *MacOS Sierra*, *Solaris* and *Windows 7* operating systems. Each computing node is then organized in a working space specially aimed to store the resources available in the node and to store the files generated by the users during the usage of the node. Figure 4 shows an example of a tree hierarchy for the working space of a computing node.

At the top of the hierarchy, we have the root folder named *'Computing Node'*. The root folder is divided in two sub-folders, the *Developers* and the *Resources* folders.

The *Resources* folder is then used to store the sources of the Prolog systems and the sources of the test bench suites available in the computing node. Figure 4 shows two Prolog

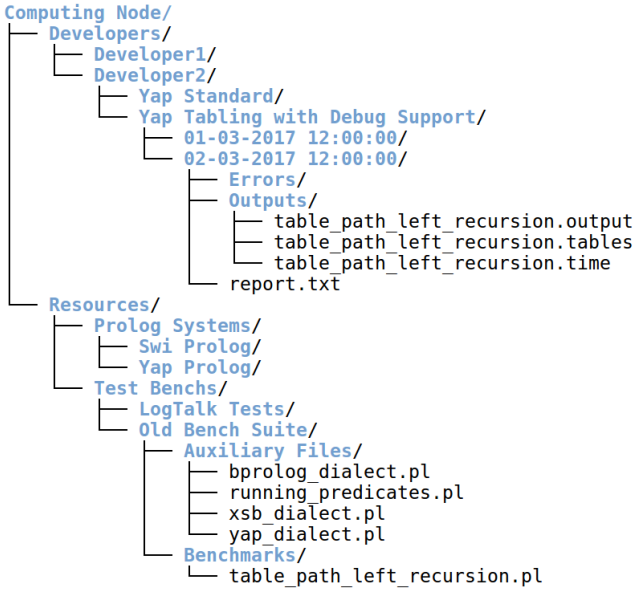



**Figure 4** Working space of a computing node.

systems (represented by the *'Swi Prolog'* and the *'Yap Prolog'* folders) and two test bench suits (represented by the *'LogTalk Tests'* and the *'Old Bench Suite'* folders) available in the computing node. The folder structure under each particular resource is then independent from YAPTBE. Figure 4 shows the specific structure for the *'Old Bench Suite'* resource. The *'Old Bench Suite'* resource corresponds to the benchmark suite we have developed in previous work to ensure Yap's correctness in the context of several improvements and new features added to its tabling engine [1, 2]. It contains two sub-folders, one named *'Auxiliary Files'* and another *Benchmarks*. The *Benchmarks* folder stores the Prolog files for the benchmarks (such as *table_path_left_recursion.pl* representing the example in Fig. 2). The *'Auxiliary Files'* folder holds the files related with the dialects specificities of each Prolog system and with the running of the benchmarks (used to launch/terminate a run; obtain the running time; print outputs; print internal statistics about the run; or print the results stored in the tables, if using tabling).

The *Developers* folder stores the information for the *builds* and the *jobs* of each developer (Fig. 4 shows the folder structure for *Developer2*). The first level sub-folders represent the developer's builds and the second level sub-folders represent the developer's jobs for a particular build. Prolog sources can be configured and/or compiled in different fashions. Each build folder corresponds to such a configuration and holds all the files required to launch the Prolog system. In Fig. 4, we can see that *Developer2* has two builds, one named *'Yap Standard'*, holding the binaries required to run Yap compiled with the default compilation flags, and another named *'Yap Tabling with Debug Support'*, holding the binaries required to run Yap compiled with tabling and debugging support. Finally, each job folder stores the outputs obtained in a particular run of a build. In Fig. 4, we can see that the *'Yap Tabling with Debug Support'* build has two jobs (by default, jobs are named with the time when they were created). The folder structure under each particular job is then independent from YAPTBE. For the job named *'02-03-2017 12:00:00'*, we can see a *report.txt* file with a summary of the run and two folders used to stored auxiliary error and output information about the run, in this case, the query output, the structure of the table space and the execution time.

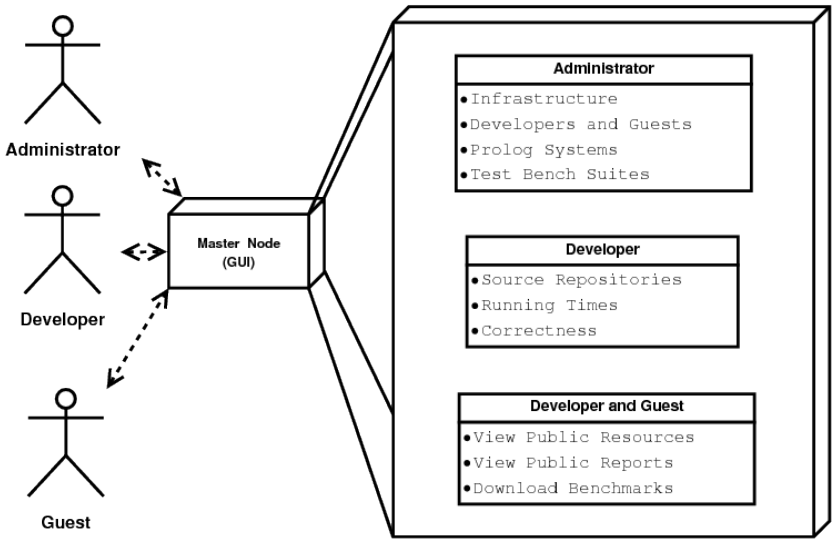



**Figure 5** Users and services provided.

## 3.2 Services

YAPTBE is designed to provide different services to different types of users. We consider three different types of users: (i) the system administrators; (ii) the developers; and (iii) the guest users. Figure 5 shows the key services provided to each user.

The system administrators will manage the infrastructure and configure the several aspects of the test bench environment. They can manage the infrastructure by adding/removing computing nodes, manage the accounts and access permissions for developers and guests, and manage the available resources by setting up the source repositories for the Prolog systems and for the test bench suites.

The developers will use the environment for performance measurements and for ensuring the correctness of the integration of the code being developed. They can manage all features related with the source repositories, such as merging, branching, pulling, configure and compile, run benchmarks and compare the running times obtained in different dates with different Prolog systems, test the correctness of the Prolog systems and check specific features, such as tabling or multithreading.

The guest users can use the environment to check and follow the state of the several Prolog systems. They can view the resources, navigate in the existent reports from previous runs, and download the available benchmarks.

Since YAPTBE's main target users are the developers, they will have a special access to the infrastructure. They will be allowed to include their machines into the cloud in such a way that they can develop and deploy their work in an computing node where they can control the environment of the run. This special feature is important because, often, developers want to quickly access what went wrong with their integration. As expected, the machine of the developer will be protected against abusive workloads by other users. We allow developers to define if their computing nodes are private or public and, in the latter case, we also allow them to define the resources that they want to share with other developers. The public resources that can be defined vary from the maximum disk space to be used, to the maximum number of cores to be used and to the maximum number of jobs to be accepted.

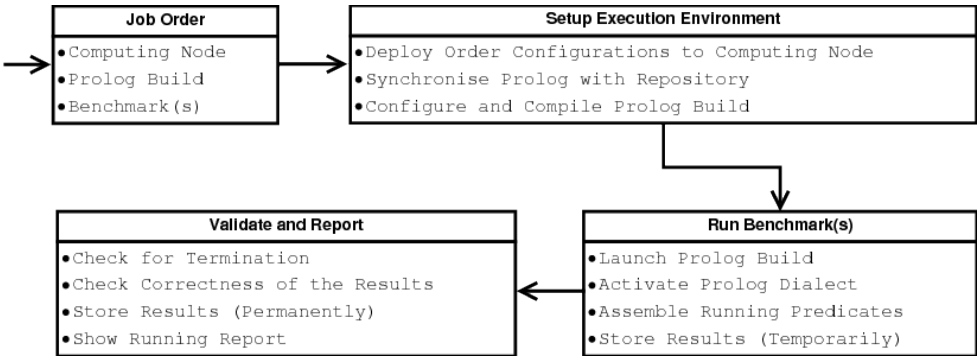



**Figure 6** Pipeline of a job request to test the correctness of a Prolog system build.

## 4 Implementation Details

In this section, we introduce some extra details about YAPTBE's implementation, which relies on the Jenkins framework [18] to manage the cloud-based architecture. Jenkins has some important advantages: (i) the user interface is simple, intuitive, visually appealing, and with a very low learning curve; (ii) it is extremely flexible and easy to adapt to multiple purposes; and (iii) has several open source plugins available, which cover a wide range of features, such as, version control systems, build tools, code quality metrics, build notifiers, integration with external systems, and user interface customization.

In a nutshell, we use Jenkins to manage the GUI, the computing nodes and the scheduling of jobs. The master node has a main Jenkins agent that runs the GUI and connects the master node with the computing nodes. Jobs are deployed by the master node to the computing nodes and, to run a job, each computing node has a Jenkins slave agent that manages the run. At the end of a run, the slave agent sends back a minor report with the results obtained to the main agent. The full details of the run are stored locally in the computing node. If a storage device is available, it can be used to backup the results. Next, we give more details about the scheduling of a job.

### 4.1 Job Scheduling

Job scheduling is one of the most important features of YAPTBE. We consider a job to be any automated service that can be provided by the environment. Jobs can vary from downloading and installing a Prolog system in a computing node to executing a run order from a developer. Figure 6 shows the pipeline for running a job request made by a developer. For the sake of simplicity, we will assume that a developer has all the permissions necessary to run the job and wants to run the latest committed version in the repository of the Prolog system.

On the initial stage of the pipeline, the developer creates an order for a job through the GUI of the master node. The order defines the computing node, the Prolog system build and the (set of) benchmark(s) to be run. The scheduling of the order is managed by Jenkins, which will insert the order in the computing node pool. When the computing node is ready to execute the order, the pipeline moves to the next stage to setup the execution environment. In this stage, Jenkins activates a set of internal scripts that will deploy the configurations of the order to the computing node. These scripts will synchronize the Prolog system with its repository, configure and compile the corresponding build in the computing node.


**(a)** Adding Yap Prolog as a resource.

**(b)** Installing Yap Prolog in a computing node.

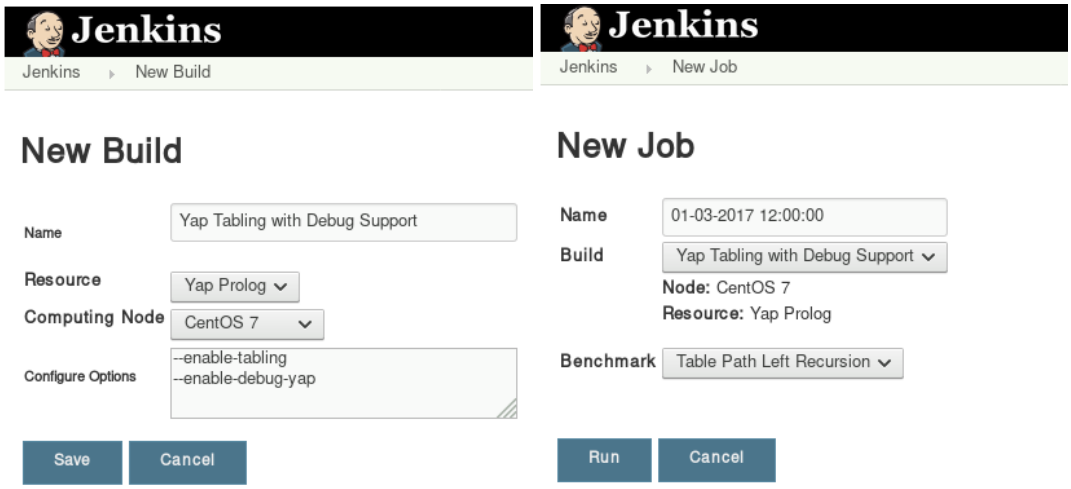
**Figure 7** Resource management GUI for administrators.

On the next stage of the pipeline, the *Run Benchmark(s)* stage, the Prolog build is launched and the (set of) benchmark(s) is ran. This can include selecting the Prolog dialect, which will activate a set of compatibility predicates that will be used to run the benchmark, and selecting specific running predicates to obtain specific outputs, such as the structure of the table space, if using tabling. Afterwards, the results are stored temporarily within a folder structure similar to the one described in Fig. 4, which can be used to store auxiliary error and/or output information, such as output answers, tabled answers, execution time, the structure of the table space, or internal Prolog statistics.

On the last stage, to validate the results, YAPTBE searches for execution failures, such as segmentation fault errors, and if no failures exist, it checks the correctness of the results. We assume that results are correct if at least two Prolog system give the same solutions. For our old bench suite, we are using the Yap Prolog and the SWI Prolog for standard benchmarks and the Yap Prolog and the XSB Prolog for tabled benchmarks (in this case, we store the output results and the answers stored in the tables). Thus, at this stage, we compare the results obtained in the run with the results pre-stored and assumed to be correct. If the results match, then the run is considered to be correct, otherwise the run is considered to be a error. Finally, the results are stored in an permanent and unique location, and a report with information is sent to the Jenkins master agent. The report has the status of the run, the execution times and the folder locations for the full output and error details.

## 4.2 Test-Driving YAPTBE

In this section, we show a small test-drive of YAPTBE. Jenkins is already packed with a huge amount of tools and has also several highly valued plugins that can be easily installed on demand. Even so, to allow administrators, developers and guests to use YAPTBE in an easier fashion, we have developed a new custom made plugin that was integrated in Jenkins. The following figures illustrate a scenario where a developer wants to use the Yap Prolog system and a computing node running the CentOS 7 operating system.




**(a)** Creating a new build for Yap Prolog system. **(b)** Running a job with a previously defined build.

**Figure 8** Job management GUI for developers.

Figure 7 shows the resource management GUI for administrators for adding Yap Prolog as a resource (Fig. 7a) and to install it in the computing node running the CentOS 7 operating system (Fig. 7b). In both cases, the GUI is quite simple. To add a new resource (Fig. 7a), the administrator has to define the name of the resource, the link to the repository with the source code, and a template with the commands to build the binary for the resource. The template can include optional arguments to be defined by the developers. For example, in Fig. 7a, the build template starts with a configure command which includes optional arguments (*'Configure Options'*) to be later defined by the developers when building a specific build of this resource. To install a resource in a specific computing node (Fig. 7b), the administrator defines the desired computing node and resource and then presses the *Install* button. If the resource installs correctly, it becomes immediately available in the computing node.

Figure 8 then shows the job management GUI for developers for creating a new build for the Yap Prolog system in the CentOS 7 computing node (Fig. 8a) and to deploy a job using such build (Fig. 8b). Again, in both cases, the GUI is quite simple. To create a new build (Fig. 8a), the developer has to define the name of the build, the resource and computing node to be used and, if the administrator has defined optional arguments in the build template commands, then such optional commands can be included here. This is the case of the *'Configure Options'* entry as previously defined in Fig. 7a. In this particular example, the developer is building Yap with tabling and debug support. After the build be saved, it becomes available for the developer to use it in future orders for a job. To deploy a job (Fig. 8b), the developer sets a name for the job and defines the build to be used (up on the definition, the computing node and resource will automatically appear in a non-editable fashion, thus that the developer can see if it is using the correct build settings). At the end, the developer defines the benchmark or set of benchmarks to be run and presses the *Run* button to launch the corresponding job. The job will enter in the job scheduler and follow the pipeline described in the previous subsection.

Although we have already implemented all the features shown, there are still many other important features that are undergoing, such as: (i) implementation of a storage node to

backup all important data; (ii) design and implement a GUI for guest users; (iii) implement a set of strict security policies for all users; (iv) increase significantly the number of tests and benchmarks available. We expect to conclude these features soon and to have the first version of YAPTBE available online in the near future.

## 5 Conclusions and Further Work

Software testing and benchmarking is a key component of the software development process. In this paper, we extended a previous work on a benchmark suite for the Yap Prolog system and we proposed a fully automated test bench environment for Prolog systems, named *Yet Another Prolog Test Bench Environment (YAPTBE)*, aimed to assist developers in the development and integration of Prolog systems. YAPTBE is based in a cloud computing architecture and relies in Jenkins and in a set of new Jenkins plugins to manage the underneath infrastructure. We presented the key design and implementation aspects of YAPTBE and showed several of its most important features, such as its graphical user interface and the automated process that builds and runs Prolog systems and benchmarks.

Besides assisting in the development of Prolog systems, we hope that YAPTBE may, in the future, contribute to reduce the gap between different Prolog dialects and to create a salutary competition between Prolog systems in different benchmarks.

In the recent past, multiple features have been added to Prolog's world. One such feature is the ISO Prolog multithreading standardization proposal [14], which currently is implemented in several Prolog systems including Ciao, SWI Prolog, XSB Prolog and Yap Prolog, providing a highly portable solution given the number of operating systems supported by these systems. Arguably, one of the features that promises to have a significant impact is the combination of multithreading with tabling [3, 4], since Prolog users will be able to exploit the combination of higher procedural control with higher declarative semantics. Future work plans include the extension of YAPTBE to support the execution and output analysis of standard and tabled multithreaded Prolog runs.

## References


1 Miguel Areias and Ricardo Rocha. On combining linear-based strategies for tabled evaluation of logic programs. *Journal of Theory and Practice of Logic Programming, (Special Issue, International Conference on Logic Programming)*, 11(4–5):681–696, July 2011.

2 Miguel Areias and Ricardo Rocha. On extending a linear tabling framework to support batched scheduling. In Alberto Simões, Ricardo Queirós, and Daniela da Cruz, editors, *Symposium on Languages, Applications and Technologies (SLATE 2012)*, pages 9–24, June 2012.

3 Miguel Areias and Ricardo Rocha. Towards multi-threaded local tabling using a common table space. *Journal of Theory and Practice of Logic Programming, (Special Issue, International Conference on Logic Programming)*, 12(4–5):427–443, September 2012.

4 Miguel Areias and Ricardo Rocha. On scaling dynamic programming problems with a multithreaded tabling system. *Journal of Systems and Software*, 125:417–426, 2017.

5 Roberto Bagnara. China – A Data-Flow Analyzer for CLP Languages. Available: `http://www.cs.unipr.it/China/` (accessed April 2017).

6 Klaus Bothe. A Prolog space benchmark suite: A new tool to compare Prolog implementations. *SIGPLAN Notices*, 25(12):54–60, 1990.

7 Weidong Chen and David S. Warren. Tabled evaluation with delaying for general logic programs. *Journal of the ACM*, 43(1):20–74, January 1996.

**8** Alain Colmerauer, Henry Kanoui, Robert Pasero, and Philippe Roussel. Un système de communication homme–machine en francais. Technical report, Groupe Intelligence Artificielle, Université Aix-Marseille II, 1973.

**9** Paul Duvall, Stephen M. Matyas, and Andrew Glover. *Continuous Integration: Improving Software Quality and Reducing Risk*. Addison-Wesley Professional, 2007.

**10** Ralph Haygood. A Prolog benchmark suite for aquarius. Technical report, University of California at Berkeley, 1989.

**11** ISO/IEC 13211-1:1995: Information technology – Programming languages – Prolog – Part 1: General core, 1995.

**12** Senlin Liang, Paul Fodor, Hui Wan, and Michael Kifer. OpenRuleBench: An analysis of the performance of rule engines. In *Internacional World Wide Web Conference*, pages 601–610. ACM, April 2009.

**13** John Wylie Lloyd. *Foundations of Logic Programming*. Springer, 1987.

**14** Paulo Moura. ISO/IEC DTR 13211–5:2007 Prolog Multi-threading Predicates, 2008. URL: `http://logtalk.org/plstd/threads.pdf`.

**15** Glenford J. Myers, Corey Sandler, and Tom Badgett. *The Art of Software Testing*. Wiley Publishing, 3rd edition, 2011.

**16** Konstantinos Sagonas and Terrance Swift. An abstract machine for tabled execution of fixed-order stratified logic programs. *ACM Transactions on Programming Languages and Systems*, 20(3):586–634, May 1998.

**17** Vítor Santos Costa, Ricardo Rocha, and Luís Damas. The YAP Prolog system. *Journal of Theory and Practice of Logic Programming*, 12(1 & 2):5–34, 2012.

**18** John Ferguson Smart. *Jenkins: The Definitive Guide*. O'Reilly Media, Inc., 2011.

**19** David H. D. Warren. An abstract Prolog instruction set. Technical Note 309, SRI International, 1983.

**20** Jan Wielemaker and Vítor Santos Costa. Portability of Prolog programs: theory and case-studies. *CoRR*, abs/1009.3796, 2010. `http://arxiv.org/abs/1009.3796`.

**21** Jan Wielemaker and Vítor Santos Costa. On the portability of Prolog applications. In Ricardo Rocha and John Launchbury, editors, *13th International Symposium on Practical Aspects of Declarative Languages (PADL2011)*, pages 69–83. Springer Berlin Heidelberg, 2011.

**22** Jan Wielemakers. Swi-prolog version 7 extensions. In *International Joint Workshop on Implementation of Constraint and Logic Programming Systems and Logic-based Methods in Programming Environments*, pages 109–123, 2014.

**23** Neng-Fa Zhou, Yi-Dong Shen, Li-Yan Yuan, and Jia-Huai You. Implementation of a Linear Tabling Mechanism. In *Practical Aspects of Declarative Languages*, number 1753 in LNCS, pages 109–123. Springer, 2000.

# Generating Method Documentation Using Concrete Values from Executions

**Matúš Sulír[1] and Jaroslav Porubän[2]**

1 **Department of Computers and Informatics, Faculty of Electrical Engineering and Informatics, Technical University of Košice, Košice, Slovakia**
`jaroslav.poruban@tuke.sk`
2 **Department of Computers and Informatics, Faculty of Electrical Engineering and Informatics, Technical University of Košice, Košice, Slovakia**
`jaroslav.poruban@tuke.sk`

## Abstract

There exist multiple automated approaches of source code documentation generation. They often describe methods in abstract terms, using the words contained in the static source code or code excerpts from repositories. In this paper, we introduce DynamiDoc – a simple yet effective automated documentation approach based on dynamic analysis. It traces the program being executed and records string representations of concrete argument values, a return value, and a target object state before and after each method execution. Then for every concerned method, it generates documentation sentences containing examples, such as "When called on [3, 1.2] with element = 3, the object changed to [1.2]". A qualitative evaluation is performed, listing advantages and shortcomings of the approach.

**1998 ACM Subject Classification** D.2.7 [Distribution, Maintenance, and Enhancement] Documentation, D.2.5 [Testing and Debugging] Tracing, D.3.3 [Language Constructs and Features] Procedures, Functions, and Subroutines

**Keywords and phrases** documentation generation, source code summarization, methods, dynamic analysis, examples

**Digital Object Identifier** 10.4230/OASIcs.SLATE.2017.3

## 1 Introduction

When developers try to comprehend what a particular method or procedure in source code does or what are its inputs and outputs, they often turn to documentation. For instance, in Java, the methods can be documented by comments specially formatted according to the Javadoc specification [6]. Similar standards and possibilities exist in other languages.

However, the API (application programming interface) documentation is often incomplete, ambiguous, obsolete, lacking good examples, inconsistent or incorrect [23]. In the worst case, it is not present at all.

Consider the simplified excerpt from Java 8 API presented in Listing 1[1] – a method `getAuhority()` in the class `URL` and its documentation.

For a person who is not a domain expert in the field of Internet protocols, this documentation in certainly not as useful as it should be. To get at least partial understanding of what the method does, he must study the corresponding RFC (Request for Comments) documents, various web tutorials and resources, or browse the rest of the lengthy API documentation.

[1] `https://docs.oracle.com/javase/8/docs/api/java/net/URL.html#getAuthority--`


**Listing 1** Excerpt from JAVA 8 API documentation.

```
public class URL {
  ...
  /**
    * Gets the authority part of this URL.
    * @return the authority part of this URL
    */
  public String getAuthority() {...}
}
```

Now, consider the same method, but with a documentation containing concrete examples of arguments and return values, as presented in Listing 2.

**Listing 2** Excerpt from JAVA 8 API documentation enriched with concrete examples.

```
public class URL {
  ...
  /**
    * Gets the authority part of this URL.
    * @return the authority part of this URL
    * @examples When called on http://example.com/path?query,
    *     the method returned "example.com".<br>
    *     When called on http://user:password@example.com:80/path,
    *     the method returned "user:password@example.com:80".
    */
  public String getAuthority() {...}
}
```

This kind of documentation should give a developer instant "feeling" of what the method does and help him to understand the source code.

Creating and maintaining documentation manually is time-consuming and error-prone. There exist multiple approaches for automated documentation generation [13]. Many of these approaches work by combining static analysis with Natural Language Processing (NLP) [17, 10] or repository mining [5]. NLP-based and static analysis approaches have an inherent disadvantage: they only present information already available in the static source code in another way. Mining-based methods require large repositories of code using concerned APIs. Furthermore, none of the mentioned approaches provide concrete, literate string representations of arguments, return values and states from runtime: at best, they provide code examples which use the API. While there exist dynamic analysis approaches collecting run-time values of variables [8, 7], they are not oriented toward textual documentation generation.

In this paper, we describe a method documentation approach based on dynamic analysis. The program of interest is executed either manually or using automated tests. During these executions, a tracer saves string representations (obtained by calling a `toString()`-like method) of the methods' arguments, return value, and object states before and after executing the method. For each method, a few sample executions are chosen, and documentation sentences similar to the exhibit shown above are generated. The generated Javadoc documentation is then written into the source files.

**Listing 3** The tracing algorithm executed around each method.

```
1  function around(method)
2    // save string representations of arguments and object state
3    arguments ← []
4    for arg in method.args
5      arguments.add(to_string(arg))
6    end for
7    before ← to_string(method.this)
8
9    // run the original method, save return value or thrown exception
10   result ← method(method.args)
11   if result is return value
12     returned ← to_string(result)
13   else if result is thrown exception
14     exception ← to_string(result)
15   end if
16
17   // save object representation again and write record to trace file
18   after ← to_string(method.this)
19   write_record(method, arguments, before, returned, exception, after)
20
21   // proceed as usual (not affecting the program's semantics)
22   return/throw result
23 end function
```

To show the feasibility of our approach, named DynamiDoc, a prototype implementation was constructed.[2] We applied DynamiDoc on multiple open-source projects and performed qualitative evaluation: we will describe its current benefits and drawbacks.

## 2 Documentation approach

Now we will describe the documentation approach in more detail. Our method consists of three consecutive phases: tracing, selection of examples, and documentation generation.

### 2.1 Tracing

First, all methods in the project we want to document are instrumented to enable tracing. Each method's definition is essentially replaced by the code presented in Listing 3. In our implementation, we used bytecode-based AspectJ instrumentation; however, the approach is not limited to it.

The target project is then executed as usual. This can range from manual clicking in a graphical user interface of an application to running fully automatized unit tests of a library. Thanks to the mentioned instrumentation, selected information about each method execution is recorded into a trace file. A detailed explanation of the tracing process for one method execution follows.

All parameter values are converted to their string representations (lines 3–6 in Listing 3).

[2] It is available at `https://github.com/sulir/dynamidoc`.

By a string representation, we mean a result of calling the `toString()` method[3] on an argument. For simple numeric and string types, it is straightforward (e.g., the number 7.1 is represented as `7.1`). For more complicated objects, it is possible to override the `toString()` method of a given class to meaningfully represent the object's state. For instance, `Map` objects are represented as `{key1=value1, key2=value2}`.

Each non-static method is called on a specific object (called `this` in Java), which we will call a "target object". We save the target object's string representation before actual method execution (line 7). This should represent the original object state. In our example from Section 1, a string representation of a URL object constructed using `new URL("http://example.com/path?query");` looks like `http://example.com/path?query`.

We execute the given method and convert the result to a string (lines 10–15). For non-void methods, this is the return value. In case a method throws an exception, we record it and convert to string – even exceptions has their `toString()` method. In the example we are describing, the method returned `example.com`.

After the method completion, we again save the string representation of the target object (`this`, line 18) if the method is non-static. This time, it should represent the state affected by the method execution. Since the `getAuthority()` method does not mutate the state of a URL object, it is the same as before calling the method.

Finally, we write the method location (the file and line number) along with the collected string representations to the trace file. We return the stored return value or throw the captured exception, so the program execution continues as it would do without our tracing code.

To sum up, a trace is a collection of stored executions, where each method execution is a tuple consisting of:

- *method* – the method identifier (file, line number),
- *arguments* – an array of string representations of all argument values,
- *before* – a string representation of the target object state before method execution,
- *return* – a string representation of the return value, if the method is non-void and did not throw and exception,
- *exception* – a thrown exception converted to a string (if it was thrown),
- *after* – a string representation of the target object state after method execution.

## 2.2 Selection of examples

After tracing finishes, the documentation generator reads the written trace file which contains a list of all method executions. Since one method may have thousands of executions, we need to select a few most suitable ones – executions which will be used as examples for documentation generation. Each execution is assigned a metric representing its suitability to be presented as an example to a programmer. They are then sorted in descending order according to this metric and the first few of them are selected. In the current implementation, we limit the number of examples for each method to 5.

In the current version of DynamiDoc, we use a very simple metric: execution frequency. It is a number of times which the method was executed in the same state with the same arguments and return value (or thrown exception) and resulted in the same final state –

[3] There are some exceptions – for example, on arrays, we call `Arrays.deepToString(arr)`. Similar methods exist in other languages, such as C# or Ruby.

considering the stored string representations. This means we consider the most frequent executions the most representative and use them for documentation generation. For instance, if the `getAuthority()` method was called three times on `http://example.com/path?query` producing `"example.com"`, and two times on `http://user:password@example.com:80/path` producing `"user:password@example.com:80"`, these two examples are selected, in the given order.

## 2.3 Documentation generation

After obtaining a list of a few example executions for each method, a documentation sentence is generated for each such execution. The generation process is template-based.

First, an appropriate sentence template is selected, based on the properties of the method (static vs. non-static, parameter count and return type) and the execution (whether an exception was thrown, or a string representation of the target object changed by calling the method). The selection is performed using a decision table displayed in Table 1. For instance, the `getAuthority()` method is non-static (instance), it has 0 parameters, does not have a void type, its execution did not throw an exception and the URL's string representation is the same before and after calling it (the state did not change from our point of view). Therefore, the sentence template "When called on {before}, the method returned {return}." is selected.

Next, the placeholders (enclosed in braces) in the sentence template are replaced by actual values. The meaning of individual values was described at the end of Section 2.1. An example of a generated sentence is: "When called on http://example.com/path?query, the method returned "example.com"." Note that we use past tense since in general, we are not sure the method always behaves the same way – the sentences represent some concrete recorded executions.

Finally, we write the sentences into Javadoc documentation comments of affected methods. Javadoc documentation is structured – it usually contains tags such as `@param` for a description of a parameter and `@see` for a link to a related class or method. We append our new, custom `@examples` tag with the generated documentation sentences to existing documentation. If the method is not yet documented at all, we create a new Javadoc comment for it. When existing examples are present, they are replaced by the new ones. The original source code files are overwritten to include the modified documentation. Using a custom "doclet" [6], the `@examples` tag can be later rendered as the text "Examples:" in the HTML version of the documentation.

# 3 Evaluation

Since the approach and its implementation is preliminary, we did not perform quantitative analysis of computational performance or program comprehension efficiency improvement. Instead, a qualitative evaluation was performed. We applied our documentation approach on three real-world open source projects and observed its strengths and weaknesses by inspecting the generated documentation.

The mentioned open source projects are:

- Apache Commons Lang,[4]
- Google Guava,[5]
- and Apache FOP.[6]

[4] `https://commons.apache.org/lang/`
[5] `https://github.com/google/guava`
[6] `https://xmlgraphics.apache.org/fop/`

**Table 1** A decision table for the documentation sentence templates.

| Method kind | Parameters | Return type | Exception | State changed | Sentence template |
|---|---|---|---|---|---|
| static | 0 | void | no | no | - |
| | | non-void | | | The method returned {return}. |
| | | any | yes | | The method threw {exception}. |
| | $\geq 1$ | void | no | | The method was called with {arguments}. |
| | | non-void | | | When {arguments}, the method returned {return}. |
| | | any | yes | | When {arguments}, the method threw {exception}. |
| instance | 0 | void | no | no | The method was called on {before}. |
| | | | | yes | When called on {before}, the object changed to {after}. |
| | | non-void | | no | When called on {before}, the method returned {return}. |
| | | | | yes | When called on {before}, the object changed to {after} and the method returned {return}. |
| | | any | yes | no | When called on {before}, the method threw {exception}. |
| | | | | yes | When called on {before}, the object changed to {after} and the method threw {exception}. |
| | $\geq 1$ | void | no | no | The method was called on {before} with {arguments}. |
| | | | | yes | When called on {before} with {arguments}, the object changed to {after}. |
| | | non-void | | no | When called on {before} with {arguments}, the method returned {return}. |
| | | | | yes | When called on {before} with {arguments}, the object changed to {after} and the method returned {return}. |
| | | any | yes | no | When called on {before} with {arguments}, the method threw {exception}. |
| | | | | yes | When called on {before} with {arguments}, the object changed to {after} and the method threw {exception}. |

The first two projects are utility libraries aiming to provide core functionality missing in the standard Java API. To obtain data for dynamic analysis, we executed selected unit tests of the libraries. The last project is a Java application reading XML files containing formatting objects (FO) and writing files suitable for printing, such as PDFs. In this case, we executed the application using a sample FO file as its input.

A description of selected kinds of situations we encountered and observations we made follows.

## 3.1 Utility methods

For simple static methods accepting and returning primitive or string values, our approach generally produces satisfactory results. As one of many examples, we can mention the method `static String unicodeEscaped(char ch)` in the "utility class" `CharUtils` of

Apache Commons Lang. The generated sentences are in the form:

```
When ch = 'A', the method returned "\u0041".
```

For many methods, the Commons Lang API documentation already contains source code or pseudo-code examples. Here is an excerpt from the documentation of the aforementioned method:

```
CharUtils.unicodeEscaped('A') = "\u0041"
```

Even in cases when a library already contains manually written code examples, DynamiDoc is useful for utility methods on simple types:

- to save time spent writing examples,
- to ensure the documentation is correct and up-to-date.

The latter point is fulfilled when the tool is run automatically, e.g., as a part of a build process. Sufficient unit test coverage is a precondition for both points.

## 3.2 Data structures

Consider the data structure `HashBasedTable` from Google Guava and its method `size()` implemented in the superclass `StandardTable`. The DynamiDoc-generated documentation includes this sentence:

```
When called on {foo={1=a, 3=c}, bar={1=b}}, the method returned 3.
```

Compare it with a hypothetical manually constructed source-code based example:

```
Table<String, Integer, Character> table = HashBasedTable.create();
table.put("foo", 1, 'a');
table.put("foo", 3, 'c');
table.put("bar", 1, 'b');
System.out.println(table.size()); // prints 3
```

Instead of showing the whole process how we got to the given state, our approach displays only a string representation of the object state (`{foo={1=a, 3=c}, bar={1=b}}`). Such a form is very compact and still contains sufficient information necessary to comprehend the gist of a particular method.

## 3.3 Changing target object state

When the class of interest has the method `toString()` meaningfully overwritten, DynamiDoc works properly. For instance, see one of the generated documentation sentences of the method `void FontFamilyProperty.addProperty(Property prop)` in Apache FOP:

```
When called on [sans-serif] with prop = Symbol, the object changed to
[sans-serif, Symbol].
```

Now, let us describe an opposite extreme. In the case of Java, the default implementation of the `toString()` method is not very useful: it displays just the class name and the object's hash code. When the class of interest does not have the `toString()` method overridden, DynamiDoc does not produce documentation of sufficient quality. Take, for example, the generated documentation for the method `void LayoutManagerMapping.initialize()` in the same project:

```
The method was called on
org.apache.fop.layoutmgr.LayoutManagerMapping@260a3a5e.
```

While the method probably changed the state of the object, we cannot see the state before and after calling it. The string representation of the object stayed the same – and not very meaningful.

Although this behavior is a result of an inherent property of our approach, there exists a way how this situation can be improved: to override `toString()` methods for all classes when it can be at least partially useful. Fortunately, many contemporary IDEs support automated generation of `toString()` source code. Such generated implementations are not always perfect, but certainly better than nothing.

We plan to perform an empirical study assessing what portion of existing classes in open source projects meaningfully override the `toString()` method. This will help us to quantitatively assess the usefulness of DynamiDoc.

## 3.4 Changing argument state

The current version of DynamiDoc does not track changes of the passed parameter values. For example, the method `static void ArrayUtils.reverse(int[] array)` in Apache Commons Lang modifies the given array in-place, which is not visible in the generated documentation:

```
The method was called with array = [1, 2, 3].
```

Of course, it is possible to compare string representations of all mutable objects passed as arguments before and after execution. We can add such a feature to DynamiDoc in the future.

## 3.5 Operations affecting external world

Our approach does not recognize the effects of input and output operations. When such an operation is not essential for the method, i.e., it is just a cross-cutting concern like logging, it does not affect the usefulness of DynamiDoc too much. This is, for instance, the case of the method `static int FixedLength.convert(double dvalue, String unit, float res)` in Apache FOP. It converts the given length to millipoints, but also contains code which logs an error when it occurs (e.g., to a console). A sample generated sentence follows:

```
When dvalue = 20.0, unit = "pt" and res = 1.0, the method returned
20000.
```

On the other hand, DynamiDoc is not able to generate any documentation sentence for the method `static void CommandLineOptions.printVersion()`, which prints the version of Apache FOP to standard output.

## 3.6 Methods doing too much

Our approach describes methods in terms of their overall effect. It does not analyze individual actions performed during method execution. Therefore, it is difficult to generate meaningful documentation for methods such as application initializers, event broadcasters or processors. An example is the method `void FObj.processNode(String elementName, Locator locator, Attributes attlist, PropertyList pList)`. An abridged excerpt from the generated documentation follows.

```
The method was called on ...RegionAfter@206be60b[@id=null] with
elementName = "region-after", locator = ...LocatorProxy@292158f8,
attlist = ...AttributesProxy@4674d90
and pList = ...StaticPropertyList@6354dd57.
```

## 3.7 Example selection

The documentation of some methods is not the best possible one. For instance, the examples generated for the method `BoundType Range.lowerBoundType()` in Google Guava are:

```
When called on (5..+∞), the method returned OPEN.
When called on [4..4], the method returned CLOSED.
When called on [4..4), the method returned CLOSED.
When called on [5..7], the method returned CLOSED.
When called on [5..8), the method returned CLOSED.
```

This selection is not optimal. First, there is only one example of the OPEN bound type – but this is only a cosmetic issue. The second, worse flaw is the absence of a case when the method throws an exception (`IllegalStateException` when the lower bound is $-\infty$).

The method `Range encloseAll(Iterable values)` in the same class has much better documentation, which shows the variety of inputs and outputs (although ordering could be slightly better):

```
When values = [0], the method returned [0..0].
When values = [5, -3], the method returned [-3..5].
When values = [0, null], the method threw
    java.lang.NullPointerException.
When values = [1, 2, 2, 2, 5, -3, 0, -1], the method returned [-3..5].
When values = [], the method threw java.util.NoSuchElementException.
```

Improvement of the example selection metric will be necessary in the future. A possible option is to include the most diverse examples: some short values, some long, plus a few exceptions.

Furthermore, like in any dynamic analysis approach, care must be taken not to include sensitive information like passwords in the generated documentation.

# 4 Related work

In this section, we will present related work with a focus on documentation generation and source code summarization. The related approaches are divided according to the primary analysis type they use – static analysis (sometimes enhanced by repository mining) or dynamic analysis.

## 4.1 Static analysis and repository mining

Sridhara et al. [16, 17] generate natural-language descriptions of methods. The generated sentences are obtained by analyzing the words in the source code of methods; therefore, it does not contain examples of concrete variable values which can be often obtained only at runtime. In [18], they add support for parameter descriptions, again using static analysis only.

McBurney and McMillan [10] summarize also method context – how the method interacts with other ones. While the approach considers source code outside the method being described, they still use only static analysis.

Buse and Weimer [1] construct natural language descriptions of situations when exceptions may be thrown. These descriptions are generated for methods, using static analysis.

Long et al. [9] describe an approach which finds API functions most related to the given C function. If adapted to Java, it could complement DynamiDoc's documentation by adding `@see` tags to Javadoc.

Moreno el al. [12] automatically document classes instead of methods. Their natural language descriptions utilize class stereotypes like "Entity", determined using source code analysis.

The tool eXoaDocs [5] mines large source code repositories to find usages of particular API elements. Source code examples illustrating calls to API methods are then added to generated Javadoc documentation. In Section 3.2, we described the difference between source code based examples and examples based on string representations of concrete variable values. Furthermore, the main point of source code examples is to show how to use the given API. Therefore, the descriptions of results or outputs are often not present in the examples.

Buse and Weimer [2] synthesize API usage examples. Compared to Kim et al. [5], they do not extract existing examples from code repositories, but use a corpus of existing programs to construct new examples.

The APIMiner platform [11] uses a private source code repository to mine examples. The generated Javadoc documentation contains "Examples" buttons showing short code samples and related elements.

## 4.2 Dynamic analysis

Hoffman and Strooper [4] present an approach of executable documentation. Instead of writing unit tests in separate files, special markers in method comments are used to mark tests. Each test includes the code to be executed and an expected value of the expression, which serves as specification and documentation. Compared to our DynamiDoc, they did not utilize string representations of objects – the expected values are Java source code expressions. Furthermore, in the case of DynamiDoc, the run-time values can be collected from normal program executions, not only from unit tests.

Concern annotations [19] are Java annotations above program elements such as methods, representing the intent behind the given piece of code. AutoAnnot [20] writes annotations representing features (e.g., `@NoteAdding`) above methods unique to a given feature obtained using simple dynamic analysis. While AutoAnnot describes methods using only simple identifiers, DynamiDoc generates full natural language sentences containing concrete variable values.

ADABU [3] uses dynamic analysis to mine object behavior models. It constructs state machines describing object states and transitions between them. Compared to our approach which used class-specific string representations, in ADABU, an object state is described using a predicate like "isEmpty()". Furthermore, they do not provide examples of concrete parameter and return values.

Lo and Maoz [8] introduce a combination of scenario-based and value-based specification mining. Using dynamic analysis, they generate live sequence charts in UML (Unified Modeling Language). These charts are enriched with preconditions and postconditions containing string representations of concrete variable values. However, their approach does not focus on

documentation of a single method, its inputs and outputs; rather they describe scenarios of interaction of multiple cooperating methods and classes.

Tralfamadore [7] is a system for analysis of large execution traces. It can display the most frequently occurring values of a specific function parameter. However, it does not provide a mapping between parameters and a return value. Furthermore, since it is C-based, it does not present string representations of structured objects.

TestDescriber by Panichella et al. [14] executes automatically generated unit tests and generates natural language sentences describing these tests. First, it differs from DynamiDoc since their approach describes only the unit tests themselves, not the tested program. Second, although it uses dynamic analysis, the only dynamically captured information by TestDescriber is code coverage – they do not provide any concrete variable values except that present literally in the source code.

FailureDoc [25] observes failing unit test execution. It adds comments above individual lines inside tests, explaining how the line should be changed for the test to pass.

SpyREST [15] generates documentation of REST (representational state transfer) services. The documentation is obtained by running code examples, intercepting the communication, and generating concrete request-response examples. While SpyREST is limited to web applications utilizing the REST architecture, DynamiDoc produces documentation of any Java program.

@tComment [22] is a tool using dynamic analysis to find code-comment inconsistencies. Unlike DynamiDoc, it does not produce new documentation – it only checks existing, manually written documentation for broken rules.

## 5 Conclusion and future work

In this paper, we presented DynamiDoc – a novel approach of automated method documentation and its preliminary implementation. It traces program execution to obtain concrete examples of arguments, return values, and object states before and after calling a method. Thanks to `toString()` methods, the values are converted to strings during the program runtime and only these converted values are stored in a trace. For each method, a few execution examples are selected, and natural-language sentences are generated and integrated into Javadoc comments.

The approach should facilitate program comprehension. The documentation generated by DynamiDoc is not intended as a complete replacement for manually written documentation, but it can be a good complement. Compared to natural language processing and simple static analysis, automated documentation approaches utilizing dynamic analysis require more effort, e.g., periodically building the software (which often fails [21]) and executing automated tests or running the software manually (which takes time). An investigation whether this additional effort is worth the benefits provided by the generated documentation should be performed.

The currently presented version has some shortcomings which we would like to mitigate before assessing its effect on code understanding.

First, the string representations of objects are not always ideal. We plan to find out the current prevalence of `toString()` methods in Java (and probably C# and other) classes, determine the "state of the art" in object string representation generation and try to improve it.

Second, we do not have empirical findings what are the attributes of a "useful example". Therefore, we used only a very simple metric of execution frequency to sort the example

executions and select the best ones. We would like to investigate, both qualitatively and quantitatively, what examples are considered the best by developers, and modify the selection metric accordingly. Existing knowledge in the area of test prioritization [24] and source code example selection [5] can be adapted and extended.

Finally, we could record also the changes of argument states and interactions with external world (input/output operations) to improve the generated summaries. In cases when summaries of overall effects of methods are insufficient, describing also individual actions inside them using runtime values of variables could help.

## References


1. Raymond Buse and Westley Weimer. Automatic documentation inference for exceptions. In *International Symposium on Software Testing and Analysis*, pages 273–282. ACM, 2008.
2. Raymond Buse and Westley Weimer. Synthesizing API usage examples. In *34th International Conference on Software Engineering (ICSE'2012)*, pages 782–792, 2012.
3. Valentin Dallmeier, Christian Lindig, Andrzej Wasylkowski, and Andreas Zeller. Mining object behavior with ADABU. In *International Workshop on Dynamic Systems Analysis*, pages 17–24. ACM, 2006.
4. Daniel Hoffman and Paul Strooper. Prose + Test Cases = Specifications. In *34th International Conference on Technology of Object-Oriented Languages and Systems*, pages 239–250, 2000.
5. Jinhan Kim, Sanghoon Lee, Seung-won Hwang, and Sunghun Kim. Adding examples into Java documents. In *IEEE/ACM International Conference on Automated Software Engineering*, pages 540–544, 2009.
6. Douglas Kramer. API documentation from source code comments: A case study of Javadoc. In *17th Annual International Conference on Computer Documentation*, pages 147–153, 1999.
7. Geoffrey Lefebvre, Brendan Cully, Christopher Head, Mark Spear, Norm Hutchinson, Mike Feeley, and Andrew Warfield. Execution mining. In *8th ACM SIGPLAN/SIGOPS Conference on Virtual Execution Environments*, pages 145–158, 2012.
8. David Lo and Shahar Maoz. Scenario-based and value-based specification mining: Better together. In *IEEE/ACM International Conference on Automated Software Engineering*, pages 387–396, 2010.
9. Fan Long, Xi Wang, and Yang Cai. API hyperlinking via structural overlap. In *7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering*, pages 203–212, 2009.
10. Paul W. McBurney and Collin McMillan. Automatic documentation generation via source code summarization of method context. In *22nd International Conference on Program Comprehension*, pages 279–290, 2014.
11. João E. Montandon, Hudson Borges, Daniel Felix, and Marco T. Valente. Documenting APIs with examples: Lessons learned with the APIMiner platform. In *20th Working Conference on Reverse Engineering (WCRE'2013)*, pages 401–408, October 2013.
12. Laura Moreno, Jairo Aponte, Giriprasad Sridhara, Andrian Marcus, Lori Pollock, and K. Vijay-Shanker. Automatic generation of natural language summaries for Java classes. In *21st International Conference on Program Comprehension*, pages 23–32, May 2013.
13. Najam Nazar, Yan Hu, and He Jiang. Summarizing software artifacts: A literature review. *Journal of Computer Science and Technology*, 31(5):883–909, 2016.
14. Sebastiano Panichella, Annibale Panichella, Moritz Beller, Andy Zaidman, and Harald C. Gall. The impact of test case summaries on bug fixing performance: An empirical investigation. In *38th International Conference on Software Engineering*, pages 547–558, 2016.

**15** Sheikh Mohammed Sohan, Craig Anslow, and Frank Maurer. SpyREST: Automated RESTful API documentation using an HTTP proxy server. In *30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 271–276, 2015.

**16** Giriprasad Sridhara, Emily Hill, Divya Muppaneni, Lori Pollock, and K. Vijay-Shanker. Towards automatically generating summary comments for Java methods. In *IEEE/ACM International Conference on Automated Software Engineering*, pages 43–52, 2010.

**17** Giriprasad Sridhara, Lori Pollock, and K. Vijay-Shanker. Automatically detecting and describing high level actions within methods. In *33rd International Conference on Software Engineering*, pages 101–110, 2011.

**18** Giriprasad Sridhara, Lori Pollock, and K. Vijay-Shanker. Generating parameter comments and integrating with method summaries. In *19th IEEE International Conference on Program Comprehension*, pages 71–80, 2011.

**19** Matúš Sulír, Milan Nosáľ, and Jaroslav Porubän. Recording concerns in source code using annotations. *Computer Languages, Systems & Structures*, 46:44–65, November 2016.

**20** Matúš Sulír and Jaroslav Porubän. Semi-automatic concern annotation using differential code coverage. In *IEEE 13th International Scientific Conference on Informatics*, pages 258–262, November 2015.

**21** Matúš Sulír and Jaroslav Porubän. A quantitative study of Java software buildability. In *7th International Workshop on Evaluation and Usability of Programming Languages and Tools*, pages 17–25, 2016.

**22** Shin Hwei Tan, Darko Marinov, Lin Tan, and Gary T. Leavens. @tComment: Testing Javadoc comments to detect comment-code inconsistencies. In *Fifth IEEE International Conference on Software Testing, Verification and Validation*, pages 260–269, 2012.

**23** Gias Uddin and Martin P. Robillard. How API documentation fails. *IEEE Software*, 32(4):68–75, July 2015.

**24** Shin Yoo and Mark Harman. Regression testing minimization, selection and prioritization: a survey. *Software Testing, Verification and Reliability*, 22(2):67–120, 2012.

**25** Sai Zhang, Cheng Zhang, and Michael D. Ernst. Automated documentation inference to explain failed tests. In *26th IEEE/ACM International Conference on Automated Software Engineering*, pages 63–72, 2011.

# Towards Employing Informal Sketches and Diagrams in Software Development*

**Milan Jančár[1] and Jaroslav Porubän[2]**

1 **Department of Computers and Informatics, Technical University of Košice, Košice, Slovakia**
`milan.jancar@tuke.sk`
2 **Department of Computers and Informatics, Technical University of Košice, Košice, Slovakia**
`jaroslav.poruban@tuke.sk`

## Abstract

Programmers write notes and draw informal sketches and diagrams. We hypothesize about understandability and helpfulness of these sketches and their deeper inclusion into software development process. We are leveraging the fact that we have a collection of such sketches affiliated to a commercial software system. We have the opportunity to study sketches that were created naturally, not intentionally for research purposes. The oldest sketch was created a year and a half ago and the most recent one a half a year ago. Our initial experiment shows that these sketches are pretty understandable even after some time – even for another person.

**1998 ACM Subject Classification** D.2.2 Design Tools and Techniques, D.2.7 [Distribution, Maintenance, and Enhancement] Documentation

**Keywords and phrases** sketches, diagrams, design, maintenance, comprehension

**Digital Object Identifier** 10.4230/OASIcs.SLATE.2017.4

## 1 Introduction

Many developers spontaneously write or create notes, lists, tables, ER/class diagrams, drawings etc. [4] For the sake of brevity, let us refer to all kinds of these informal artifacts simply as a *sketch*. There are various reasons why these sketches may come into being, mainly "to understand, to design and to communicate" [3]. Many important design decisions are made on whiteboards [3]. These sketches may resemble UML but do not strictly adhere to it [4], they are spontaneous, ad-hoc and informal. We believe this spontaneity and informality is great to capture *immediate* thoughts. These sketches can be later changed, refined, even formalized (if necessary) – usually such sketches also have a longer lifespan and are more likely to be archived [1]. An interesting fact is what medium developers use for sketching: almost two thirds are on some analog medium (mainly paper and whiteboards), whereas modern means such as *interactive* whiteboards, tablets and smartphones are almost never used [1].

It is not sufficient to just *archive* those sketches – some *organization* must be brought in. Not to forget about the time dimension – as mentioned, the sketches *evolve* and we need means for handling this. Baltes et al. [2] created a tool for managing these sketches, especially for linking them to relevant parts of source code. There is still room for improvement and

* This work was supported by the project KEGA No. 047TUKE-4/2016: "Integrating software processes into the teaching of programming".



for studying usability and influence of sketches on program comprehension when embedded to corresponding code fragment.

We believe the potential of these sketches is yet to be fully utilized and we have a direct experience with these sketches, which in fact inspired this kind of research. It must be emphasized that we are only in the early stage of the research and the paper presents preliminary results obtained by observing and analyzing the sketches of one programmer. The aim was to better understand this field of study. Future work will focus on more rigorous experiments involving more programmers. The ultimate objective of our future work is to validate a deeper inclusion of sketches into software development process and to propose new approaches and recommendations for developers and for creators of those sketches to better utilize their potential.

We collected sketches capturing internals of a commercially developed system we participate on. The added value of our initial experiments lies in the fact that those sketches are *quite old* – their origin is in range from *a year and a half ago* to *a half a year ago* (October 2015 – October 2016).

The contribution of the paper lies in the *analysis and evaluation of understandability and (perceived) helpfulness of sketches* after a relatively long time period since their origin. In summary, the following aspects are regarded:

- the *nature* of the sketches – type of elements, notations, etc.,
- perceived and objective *understandability*,
- perceived *helpfulness*,
- perception of the sketches by a *non-author*.

## 2 Background

A little context is given – an explanation of the origin of the sketches and a brief description of a software project they are related to.

### 2.1 Software System

The software system documented by the mentioned sketches has a following nature. The system is a company internal system for managing their customers (people applying for a job). It is a web application consisting of a server side created in Grails (a Groovy-based framework for the Java Virtual Machine) and a client side created in AngularJS (a JavaScript framework). Source files have ca. 150 KLOC[1] and following programming languages are used[2]: Groovy (48%), JavaScript (28%), HTML (14%), Java (7%) and some others.

### 2.2 Origin of Sketches

The sketches are private notes taken by one of the authors of the paper. It is important to note that they were *not* intended for such a research. Their studying is purely incidental. Knowing about the consecutive research beforehand might have invalidated obtained results because the sketches might have been (subconsciously) adjusted or a greater effort might have been put into creating them (cf. Hawthorne effect).

[1] Kilo Lines of Code (no blank or comment lines), based on the output of `cloc` tool applied to files stored in a Git repository (`git ls-files`).
[2] As reported by *GitLab* in Graphs–Languages section.

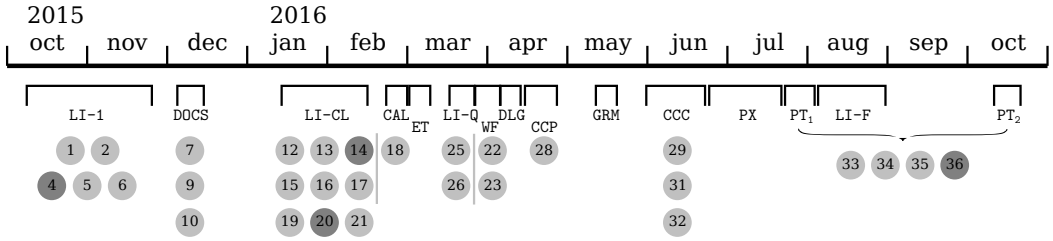



**Figure 1** Time frame of sketches, the dark ones were selected for closer studying.

## 3 Data Collection

We are taking a collection of 37 sketches and are trying to get as much useful information as possible from it.

First, we assigned identifiers to *all* of the found sketches in the exact order as they have been found filed, starting with the paper on the bottom given the ID 1.

Then, from *37 sketches*, 7 were discarded (namely # 3, 8, 11, 24, 27, 30, and 37) because they were archived either by mistake or because they were remains of to-do lists (crossed out to large extent) with very little information value. So we have *30 sketches* for further analysis.

The next important thing, since we are focusing on the fact that the value of sketches lies in their *age*, was to define a time frame in which the sketches originated. That turned out nontrivial since the sketches were not timestamped. So we chose the following approach:

1. Utilizing a Git repository of the affiliated software project, we listed all commits by the author of the sketches and skimmed over them to get a list of tasks being done on the project plus their time range. By *a task* we mean some self-contained functionality, a feature addition/improvement, we might say *a user story* (although we do not want to imply any development methodology) which is sufficiently significant (lasting at least five days).
2. We analyzed sketches to assign a task from the list obtained in the step 1 to each sketch.
3. Based on the assignment *sketch→task* (step 2) and *task→time* (step 1) we can approximately define a time of origin of respective sketches. See Fig. 1 for the resultant time frame. (Tasks are assigned IDs such as `LI-1`, `DOCS`, but their meaning is internal and not important for this study.)

## 4 Nature of the Sketches

Technically, all but two sketches were created by hand on a piece of (scrap) paper, those two were drawn on a computer (by another person). The most of them (ca. 80%) were on a single page A4 paper.

We analyzed types of elements and notations used in sketches. When appropriate, well known standard notation elements were used such as those for entity/relation diagrams, class diagrams, state transition diagrams etc. However, they were not used precisely, rather freely with additional useful ad-hoc symbols.

A summary of observed recurring patterns found in sketches is in Table 1. Number of sketches having a particular type is stated. The sum does not yield 30 (100%) because one sketch may contain elements of more than just one type.

**Table 1** Types of sketches.

| Type | Abs. no. | Rel. no. |
|---|---|---|
| entity/relation (ER) diag. | 9 | 30% |
| domain entities and relations between them | | |
| algorithm | 7 | 23% |
| important steps, strategies, if-then cases | | |
| user interface sketch | 5 | 17% |
| a graphical user interface (GUI) design | | |
| state transition diag. | 4 | 13% |
| a diagram of states and transitions between them | | |
| table/matrix | 4 | 13% |
| arbitrary information captured in a table/matrix manner | | |
| Q/A list | 4 | 13% |
| a list of questions and answers which are obtained from a more experienced fellow developer | | |
| modules overview | 2 | 7% |
| a high-level view on modules structure and their dependencies | | |
| generic notes | 18 | 60% |
| notes, prevalently in form of a list, not matching any above-mentioned characteristic | | |
| *all* | *30* | *100%* |

## 5 Initial Experiment

We ask these questions: *Are sketches, after a time has passed, understandable for their author and/or for other people?* And if the answers is yes: *Are those sketches still helpful, for instance, in maintenance tasks?*

From the author perspective, sketches are still very well understandable, almost all notation symbols and abbreviations are familiar.

Nonetheless, to evaluate a hypothetical understandability and (perceived) helpfulness of the personal sketches by someone else, we performed a quick exploratory experiment testing a rather vague hypothesis that *personal sketches related to a software project, even after some time, may be understandable and helpful for other developers participating on that project.*

We chose a questionnaire approach. A respondent is our colleague who also works on the mentioned software project. Thus he has some general domain knowledge. However, he has not worked on the tasks the sketches were about. He could have some marginal knowledge about them, though.

There are probably two main factors impairing understanding for him: *a time factor* (sketches may be outdated) and *"created by someone else"* factor. On the other hand, there is one thing that probably has a positive effect on understanding – the mentioned fact that the respondent has an experience with the related system.

### 5.1 Method

The questionnaire was comprised from the following 5 questions.

- **Q1:** Which project artifacts (on a file level) are related to a given sketch? To what extent – on the scale: marginally (less than 1/3 of a file content), partially (more than 1/3 of a file content but less than 2/3), largely (more than 2/3 of a file content)? [Time to solve: 10 minutes max. per a sketch]

- **Q2:** The given sketch involves various texts, pictograms, abbreviations, symbols, notations, diagram elements etc. Subjectively, to what extent do you think you understand the sketch? The scale: 0–20%, 20–40%, 40–60%, 60–80%, 80–100%.
- **Q3:** What hinders you the most in understanding the given sketch? (Open question)
- **Q4:** With the statement "The given sketch will help me in the future solve related maintenance tasks (adding or modifying a feature, bug fixing)", I: a) strongly agree, b) agree, c) do not know to take a stand (neutral), d) disagree, e) strongly disagree.
- **Q5:** Briefly summarize what you have grasped from the given sketch. If you caught some interesting details, include them. (Open question)

The 1st and 5th questions examine indirectly the respondent's *objective* level of *understanding* (when compared to the reality). The 2nd question targets the *perceived* level of *understanding* and the 4th question the *perceived* level of *helpfulness*.

For the sake of simplicity, only four sketches were selected for this initial experiment. They were selected in such a way that the following qualities were covered in the widest possible spectrum: age, neatness, notation types. Their ordering reflects our assumptions about their difficulty to be understood by a stranger (starting with the presumably easiest one). They have the following characteristics (for their age, see Fig. 1):

1. **ID 20:** a neat diagram, adhering (although not strictly) to well-known UML class diagram notation, it contains 7 classes (domain entities); we consider it easy to understand, it is basically a "control" sketch,
2. **ID 14:** basically a state transition diagram; contains some unexplained symbols and notations which makes it nontrivial to understand; a context (states and transitions of *what*) is not explicitly given,
3. **ID 36:** an intricate diagram drawn with no standard notation in mind; captures relationships among controllers and services with some additional notes; contains many unexplained abbreviations what makes it even harder to understand,
4. **ID 4:** has a form of a table; contains notes describing a strategy (algorithm); contains many unexplained abbreviations and also notations such as circles, arrows and wavy lines.

To create an image of what the sketches used in the questionnaire look like, photos have been taken, see Figures 2, 3 and 4.

## 5.2 Results

The answers to the questions Q2–Q4 from the filled out questionnaire are in Table 2. For the Q1, *we* decided which artifacts should be considered related, hence "correct", as we have a deep understanding of the system and also sketches. Based on this, we computed recall and precision of artifacts included in the obtained answer for this question.

A little elaboration is needed to clarify how we computed recall and precision. Let us have two sets: a set of "correct" (relevant) elements and a set of "selected" elements. It is well known that we compute recall $r$ and precision $p$ as follows:

$$r = \frac{|correct \cap selected|}{|correct|} \quad p = \frac{|correct \cap selected|}{|selected|}\,.$$

In our case, source artifacts (files) are those elements. However, our respondent was asked to not only select related artifacts but also to state an extent (marginal, partial, large) to which they were related. To regard extents of artifacts, we assigned weights (1–marginal, 2–partial or 3–large) to the artifacts in both sets (of selected and correct artifacts), effectively creating two (so-called) weighted sets or *multisets*. Above-mentioned formulas still hold but

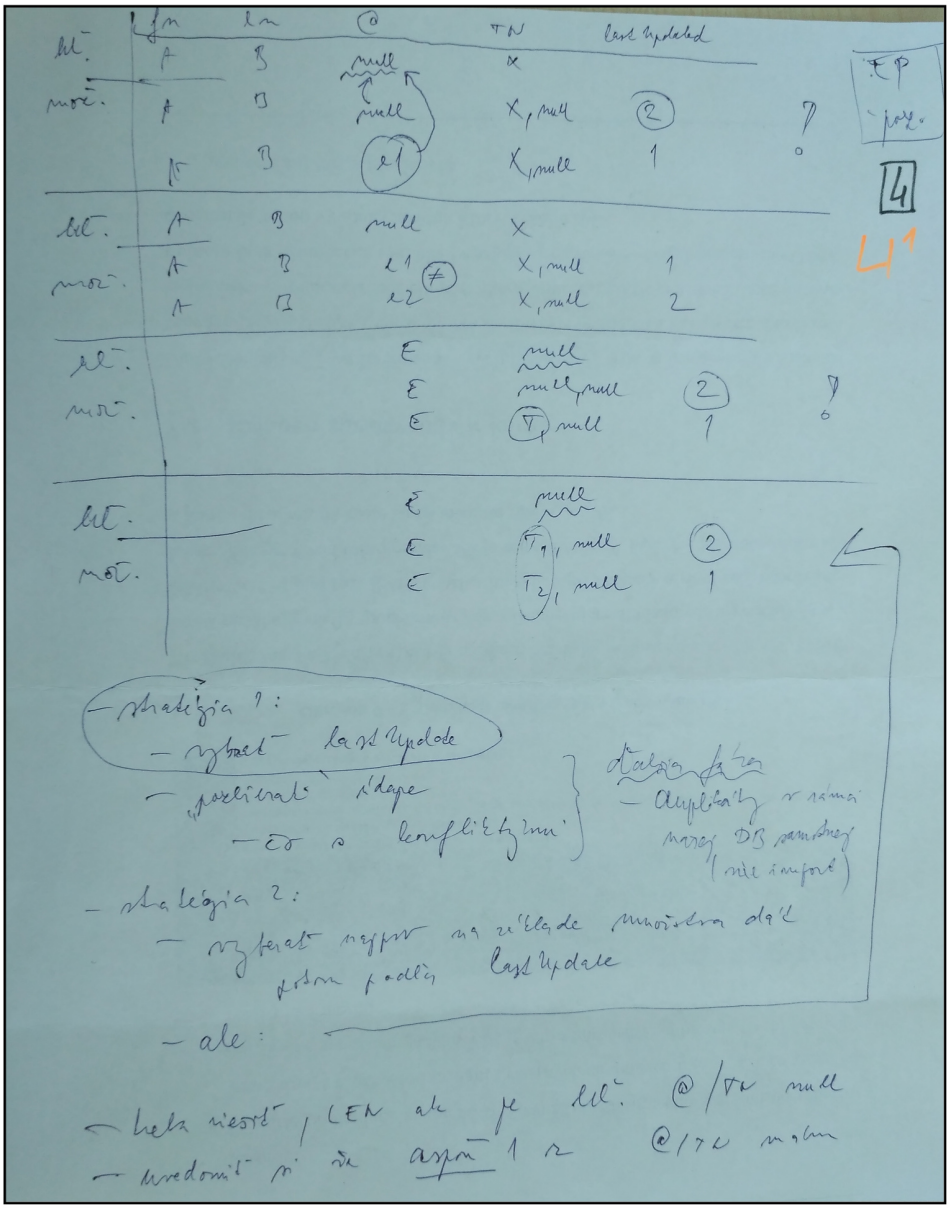


**Figure 2** Sketch #4.

*correct* and *selected* are now multisets, and thus special rules, e.g. for intersection, apply. For instance, let *correct* multiset be $\{a, a, a, b\}$ (meaning an artifact $a$ to a large extent and an artifact $b$ to a marginal extent) and *selected* be $\{a, a, c\}$. Then, their intersection is $\{a, a\}$ – meaning "the artifact $a$ to a partial extent".

Answers to the question Q5 and a high rate of confidence expressed in the Q2 aroused our interest. We extended the given textual questionnaire by interview questions asking about concrete facts captured in a sketch in order to find out if the respondent really understood them (or just *thought* he understood them).

Conclusions drawn from the Q5 combined with the interview are in Table 3.

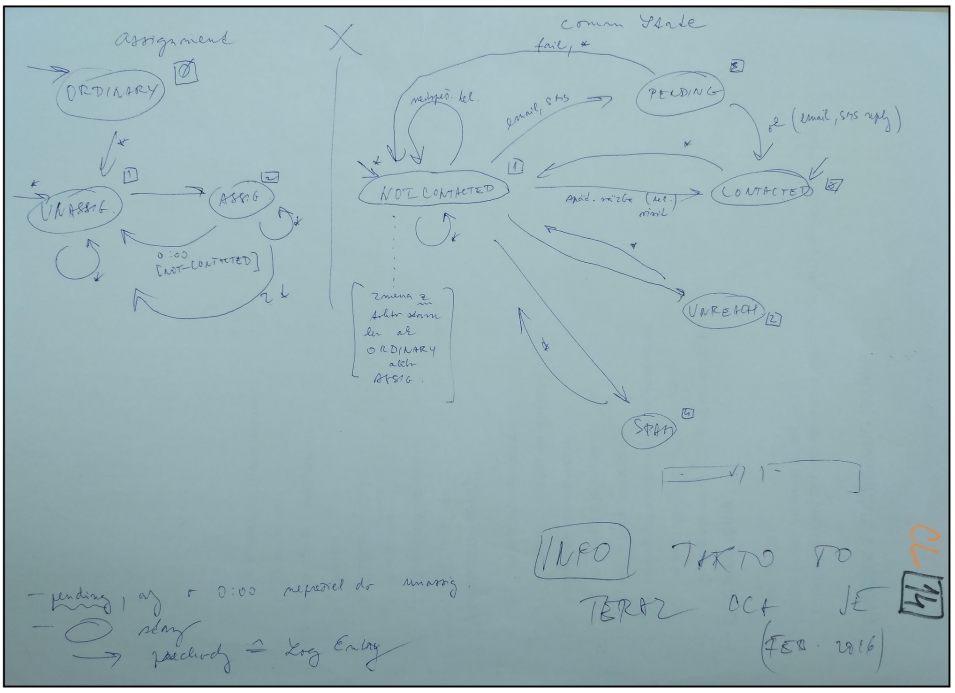



**Figure 3** Sketch #14.

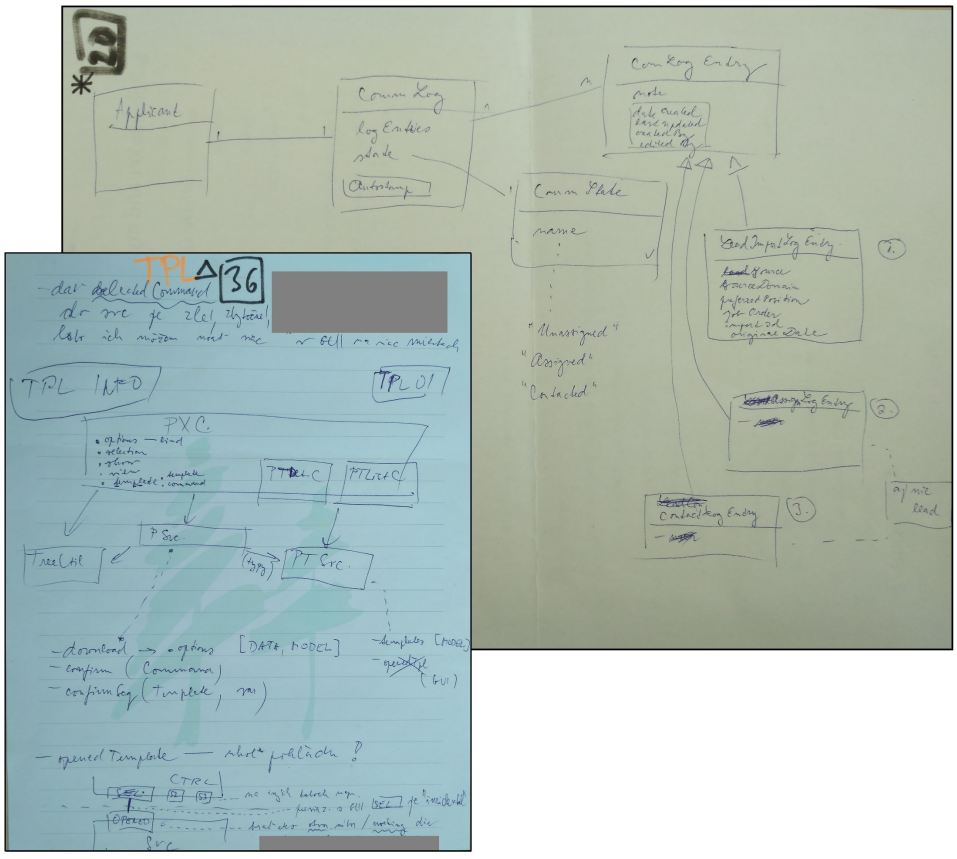



**Figure 4** Sketches #20 and #36 (top-down, blank space overlapped).

**Table 2** Obtained results from questionnaire.

| Sketch | Q1 (artif.) | Q2 (perceiv. und.) | Q3 (hindrance) | Q4 (perceiv. help.) |
|---|---|---|---|---|
| #20 | $r = 84.6\%$<br>$p = 64.7\%$ | 80–100% | illegible handwriting, notation (numbers) | agree |
| #14 | $r = 37.5\%$<br>$p = 25.0\%$ | 60–80% | illegible handwriting, notation (a math sign) | neutral |
| #36 | $r = 0.0\%$<br>$p = 0.0\%$ | 0–20% | illegible handwriting, abbrevs., small text, notation (rectangle, arrows) | disagree |
| #4 | $r = 100.0\%$<br>$p = 66.7\%$ | 80–100% | illegible handwriting | agree |

**Table 3** Question Q5 and the interview.

| Sketch | Report on the answer to Q5 and additional interview questions |
|---|---|
| #20 | understood well, no more interview questions asked |
| #14 | understood pretty well, although 1 fact was misunderstood and 2 symbols (*, ×) not understood |
| #36 | completely not understood/misunderstood; misunderstood abbreviations and illegible handwriting caused that the respondent saw totally unrelated elements |
| #4 | understood well, all additional questions (except for one) answered correctly or satisfactorily |

## 5.3 Discussion

Evaluating the question Q1 turned out to be very difficult. There are significantly different views on what it means for an artifact to be related to a sketch. That caused relatively low values of recall and precision even for the sketch #20 where we thought the artifacts are obvious. Differences were also caused by various opinions on "covered extent" of artifact.

The perceived understanding (Q2) nicely correlate with the perceived helpfulness (Q4) so the respondent thinks that if he can understand a sketch, it will also help him.

A high level of confidence (Q2) was suspicious but based on Q5 and the additional interview questions, it seems that the respondent really understands what he claims. It is most surprising at the sketch #4, which we considered to be hardly understandable but the respondent understood it pretty well. The respondent admitted working on a task similar to the one described by this sketch. But still, he had to infer the meaning of obscure abbreviations and unexplained notation (arrows, circles, lines) and that was nontrivial.

What hinders understanding the most (Q3) is obvious and expected – mainly *illegible handwriting* and *unexplained notation*; also abbreviations, but not that much, probably because the respondent (a fellow developer) shares some domain knowledge and is able to infer them.

Sketch #36 came as a surprise: not because it was not understood but because the respondent was able to see *something what was not there*. It was caused mainly by ambiguous abbreviations and notation.

## 6 Future Work

Our future work may focus on measuring the *helpfulness* of embedding (or linking) sketches directly in (to) source code for *program comprehension*. "Program comprehension is an internal cognitive process that inherently eludes measurement" [5]. Measuring program comprehension (understanding) is hard and only *indirect*. Siegmund [5] enumerates the following approaches for measuring:

1. *software measures* measuring the code itself: based on the assumptions that the more complex code, the harder it is to comprehend – e.g. lines of code, cyclomatic complexity;
2. *subjective rating* of developer's understanding;
3. *performance of developers*: based on the assumed correlation between ease of understanding and speed of fulfilling a given task;
4. *think-aloud protocols*: allow observing a process of comprehending by verbalizing subject's thoughts.

Considering these standard approaches, software measures (1) are not applicable for obvious reasons: our approach does not affect existing code. In (2) and (4), measurements would be hardly comparable and obviously biased. The approach (3) – measuring performance of developers – seems like the most appropriate choice.

We plan to conduct a controlled experiment, where participants (programmers) will be given a task and source code of the relevant part of the project. A task may be oriented on programming (altering a functionality) or deriving some knowledge from the code. Both approaches require *comprehending* and can be measured – by measuring a time to fulfill the task. Participants of the experimental group will also be given a relevant *sketch*, since we are hypothesizing about their *helpfulness*.

## 7 Conclusion

Our small experiment showed that sketches, even after a long time, have a potential to be understandable and helpful – even for other people.

We will conclude this paper with lessons learned from our initial experiment about utilizing sketches in software development:

- One should care about his or her handwriting and to explain abbreviations and especially notations used.
- Being consistent in abbreviations and notations across many sketches increases the chances of their understandability.
- From various reasons, it might be useful to timestamp all sketches.
- It is also a good practice to label sketches with a related task – while a sketch is current, its context is obvious, but later the context is lost. Also these labels may streamline sorting out sketches or finding sketches related to a specific subject (task).
- We observed the following three reasons why a sketch was created: (1) to help think/reason, (2) as a medium to ask questions and capture answers (for instance from a fellow more experienced programmer), (3) to capture important design decision for future reference.

As the biggest hindrance to fully employ sketches in the development process, we consider the fact that those sketches are put away, archived, with no live connection to software source artifacts. They simply do not automatically pop up when we need them and this also is the area of our future interest.

## References


1 Sebastian Baltes and Stephan Diehl. Sketches and diagrams in practice. In *22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 530–541, 2014.

2 Sebastian Baltes, Peter Schmitz, and Stephan Diehl. Linking sketches and diagrams to source code artifacts. In *22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 743–746, 2014.

3 Mauro Cherubini, Gina Venolia, Rob DeLine, and Andrew J. Ko. Let's go to the whiteboard: how and why software developers use drawings. In *SIGCHI conference on Human factors in computing systems*, pages 557–566, 2007.

4 Nicolas Mangano, Thomas D. LaToza, Marian Petre, and André van der Hoek. How software designers interact with sketches at the whiteboard. *IEEE Transactions on Software Engineering*, 41(2):135–156, February 2015.

5 Janet Siegmund. Measuring program comprehension with fMRI. *Softwaretechnik-Trends*, 34(2), 2014.

# Modelling Contiki-Based IoT Systems*

**Caglar Durmaz[1], Moharram Challenger[2], Orhan Dagdeviren[3], and Geylani Kardas[4]**

1 **International Computer Institute, Ege University, İzmir, Turkey; and Integra Project Management Engineering Consultancy Ltd., İzmir, Turkey**
caglar.durmaz@gmail.com, caglar@integra-pm.com.tr
2 **International Computer Institute, Ege University, İzmir, Turkey**
moharram.challenger@ege.edu.tr
3 **International Computer Institute, Ege University, İzmir, Turkey**
orhan.dagdeviren@ege.edu.tr
4 **International Computer Institute, Ege University, İzmir, Turkey**
geylani.kardas@ege.edu.tr

## Abstract

In this paper, we investigate how model-driven engineering (MDE) of Internet of Things (IoT) systems and Wireless-Sensor Networks (WSN) can be supported and introduce a domain-specific metamodel for modeling such systems based on the well-known Contiki operating system. The unique lightweight thread structure of Contiki makes it more preferable in the implementation of new IoT systems instead of many other existing platforms. Although some MDE approaches exist for IoT systems and WSNs, currently there is no study which addresses the modelling according to the specifications of Contiki platform. The work presented in this paper aims at filling this gap and covers the development of both a modeling language syntax and a graphical modeling environment for the MDE of IoTs according to event-driven mechanism and protothread architecture of Contiki. Use of the proposed modeling language is demonstrated with including the development of an IoT system for forest fire detection.

**1998 ACM Subject Classification** D.1.7 Visual Programming, D.2.6 [Programming Environments] Graphical Environments, D.4.7 [Organization and Design] Real-Time Systems and Embedded Systems, Embedded Software

**Keywords and phrases** Domain-specific Modelling, Metamodel, Model-driven Engineering, Internet of Things, Wireless Sensor Networks, Embedded Software, Contiki Operating System

**Digital Object Identifier** 10.4230/OASIcs.SLATE.2017.5

## 1 Introduction

Internet as a global information and communication infrastructure, is evolving to form of a platform for letting machines and smart objects communicate, dialogue, compute and coordinate [17]. The term "Internet-of-Things" (IoT) is broadly used for these interconnected objects which build the smart environments, e.g. smart homes [12]. IoT systems do not only present diverse context of smart implementations, but also present diverse computing and communication capabilities. This heterogeneity in devices brings management challenges in architectural and protocol issues [17] which requires network, embedded and distributed

* Authors would like to thank the Scientific and Technological Research Council of Turkey (TUBITAK) Electric, Electronic and Informatics Research Group (EEEAG) for covering SLATE conference attendance and paper presentation expenses under the project grant 115E449.



programming knowledge [2, 18]. Although advances in the development of low-cost and low-power micro-controllers play an important role during the construction of IoT systems, the scarcity of application developers who possess the required knowledge and experience for such systems, limits the power of using the micro-controller technologies in IoT. Proper software platforms, which facilitate the design and implementation of IoT applications, may bring more developers into this domain. For the sake of increasing need for application development on IoT, several operating systems for Wireless Sensor Network (WSN) and IoT motes are realized. Contiki [9] and TinyOS [14] are two of these best known sensor node operating systems (OS) [20]. TinyOS provides interfaces and components for common abstractions to implement algorithms designed for eneergy-efficient devices [1, 5]. Contiki gained popularity recently because of built in TCP/IP stack and lightweight preemptive scheduling over event-driven kernel [9] which is a very motivating feature for IoT.

Even if operating systems play the role of an abstraction layer for low-level hardware heterogeneity, distributed programming and network related concerns dominate the workload in software development. The separation of hardware-related and application related concerns will improve the software engineering processes of IoT systems [15] which may pave the way to deal with the system's structural complexity coming from the heterogeneity.

One possible approach to cope with this complexity is to increase the abstraction level using system models [16], for WSN and IoT in a Model Driven Engineering (MDE) approach. MDE moves software development from code to models and may increase productivity and reduce development costs [23]. Fruitfulness of this approach is demonstrated in several other domain studies, e.g. [4, 26].

MDE for IoT systems and WSNs is being researched in several studies especially for the purpose of separation of concerns [3, 11, 21, 25]. Although, some of these studies deal with the structural complexity of IoT, most of them do not provide a complete and/or a systematic approach for their MDE solutions. Moreover, most of these studies only consider the MDE of IoT only on TinyOS as reported in [11]. In order to contribute these studies and fill the gap for providing an MDE approach for developing IoT systems based on Contiki OS, in this paper, we introduce a domain-specific metamodel, which can be used for modeling Contiki applications. Based on the metamodel, a visual concrete syntax is derived and a graphical modeling environment is developed for modeling IoT systems. Use of the proposed modeling language is demonstrated with including the development of an IoT system for forest fire detection.

Rest of the paper is organized as follows. In Section 2, Contiki OS is briefly discussed. Section 3 introduces the metamodel proposed for Contiki. Section 4 covers the derived concrete syntax, its notations and the related graphical modeling environment. A use case study is given in Section 5 for exemplifying the use of the proposed modeling environment. Section 6 includes the related work. Finally, Section 7 concludes the paper and states the future work.

## 2 Contiki Operating System

Contiki[1] is a lightweight open source OS written in C for IoT. Contiki connects tiny low-cost, low-power micro-controllers to the Internet. As indicated in [24], Contiki and its micro IP (uIP) stack are used worldwide by hundreds of projects and companies. The uIP implements only the minimal set of features needed for a full TCP/IP stack.

[1] Contiki: `http://www.contiki-os.org/index.html`.

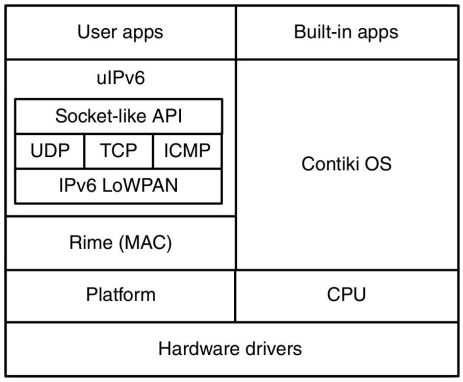



**Figure 1** The architecture and main components of Contiki [24].

Operating systems, like TinyOS [14] and SOS [13] are based on event-driven model which is often used on memory constrained devices. Contiki is also event-driven and provides a lightweight thread model called protothreads [10] over event-driven kernel which is not available in the other peer operating systems. User and built-in applications can be implemented over protothreads which are basic OS threads. Protothreads simplify event-driven programming by reducing the need for explicit state machines by providing abstraction of conditional blocking wait operation [10]. PT_WAIT_UNTIL() statement in Contiki [10] blocks conditional execution of a process. On the other hand, PT_YIELD() statement blocks execution of process unconditionally. Protothread waits until the next time the protothread is invoked and continues executing the code following the PT_YIELD() statement.

The architecture of Contiki is shown in Figure 1. Hardware Drivers, Platform and CPU abstractions form the abstraction for the real low-level hardware. Platform and CPU layers are implemented independently due to portability concern. The Rime system provides medium access control and a set of lightweight communication primitives for network protocols. The uIPv6 stack makes use of Rime, and provides TCP, UDP and ICMP and also a socket-like API, protosockets [2]. The protosocket implementation makes use of Contiki protothreads.

The Contiki also provides timer support and dynamic linking capabilities as system services. Both updating the system which contains hundreds or even thousands of nodes with new functionality and correcting software bugs are often needed in deployed IoT systems [8]. Developing full system image replacement is a solution for this situations but it is not feasible to physically collect and reprogram all sensor devices [9]. By the help of dynamic linking, new code modules can be added at runtime to the application running on a node.

In this study, we address the modelling of the following sections of Contiki architecture given in Figure 1: ContikiOS, User apps, and uIPv6 (including Socket-like API, UDP, TCP). In this way, by using the provided metamodel of ContikiOS and uIPv6, we can model a user application for an IoT system.

## 3 The Metamodel for Contiki-based IoT Systems

In this section, a metamodel is presented for MDE of Contiki-based systems to provide the concepts and their relations pertaining to IoT domain. This metamodel can be used as an abstract syntax and pave the way for developing a modelling language for Contiki.

[2] ContikiOS 2.6 Documentation: `http://contiki.sourceforge.net/docs/2.6/index.html`.

As shown in Figure 2, the proposed metamodel provides the meta-entities and their relations required for the structural architecture of Contiki programs. Entities of the metamodel are given in italics throughout the paper.

Since Contiki is primarily an OS, main entities of Contiki are process and thread inside each *Node*. A process in Contiki consists of a single protothread. *Process_Thread* is used to define this single protothread of the process [10]. A process thread can also call several stand alone protothreads represented as *PT_Thread* in the metamodel by the use of *PT_Thread_Call*. Boolean value of *Autostart* attribute of *Process_Thread* designates whether the thread will be started at the beginning of the mote execution, or not. Also, *Name* attribute is the declaration of the thread while *Description* attribute represents a short description for debugging concerns. Being an event-driven OS, Contiki kernel sends events to *Process_Thread* with event and data arguments. The attributes, *EventArgument* and *DataArgument* in the metamodel, define variable names of event and data arguments.

*PT_Thread_Call* holds the destination *PT_Thread* and a concrete PT_Thread_Struct argument of *PT_Thread*. PT and Psock are concrete structures that can be sent as arguments when calling *PT_Threads*. Psock ,which can only be used in *PT_Threads*, offers TCP and UDP socket implementations in Contiki.

Messaging among processes and threads is another core function of operating systems. Due to being again event-driven, messages received from other processes are handled by a specialized event, *Process_Event*, and the messages are sent by *Process_Post*. *Process_Post* has exactly one Data attribute that stores message payload and exactly one *Process_Event* that is going to be triggered. Sync attribute of *Process_Post* defines the execution of caller process thread which is going to be synchronous or asynchronous.

Besides this messaging between processes in the same mote, messaging among processes residing in different motes is another requirement of an IoT OS. Contiki is differentiated itself from other WSN operating systems by implementing network IP stack and built-in TCP/IP and UDP support. Related feature is modelled in the proposed metamodel as follows: One mote can start a connection and send first message via *Client_Connection* with an outgoing *Data* with *RemoteIP* and *RemotePortNumber* attributes. This connection can also be used to retrieve incoming messages from the relevant remote host. Contiki fires *TCPIP_Event* when a UDP or TCP packet is arrived. There is no special event type for UDP IP events. By the help of connection and *TCPIP_event*, messages from other host can be detected and processed. On the other side, *Server_Connection* entity can receive messages from hosts by defining *ListeningPort*. Same *TCPIP_Event* and connection entity types are also used to process messages on the server side. Connections can be used for several messages with the remote host, several incoming and outgoing data may be related to a connection in the model. The Last event type is *Time_Event* and it occurs when *Etimer* gets to zero. *Etimer* has *Name* attribute which holds a variable name in *Process_Thread*, and *Period* attribute which defines durations between the current moment and the time of next firing of *Time_Event*.

## 4 The Concrete Syntax

The metamodel introduced in the previous section represents an abstract syntax for a modeling language for Contiki-based IoT systems. While the abstract syntax includes the concepts and their relations between those concepts, a concrete syntax provides a mapping between these concepts (meta-elements) and their textual and/or graphical representation. In this study, we also introduce a graphical concrete syntax for modelling Contiki applications. Table 1 lists the graphical notations for some of the important entities discussed in Section 3.

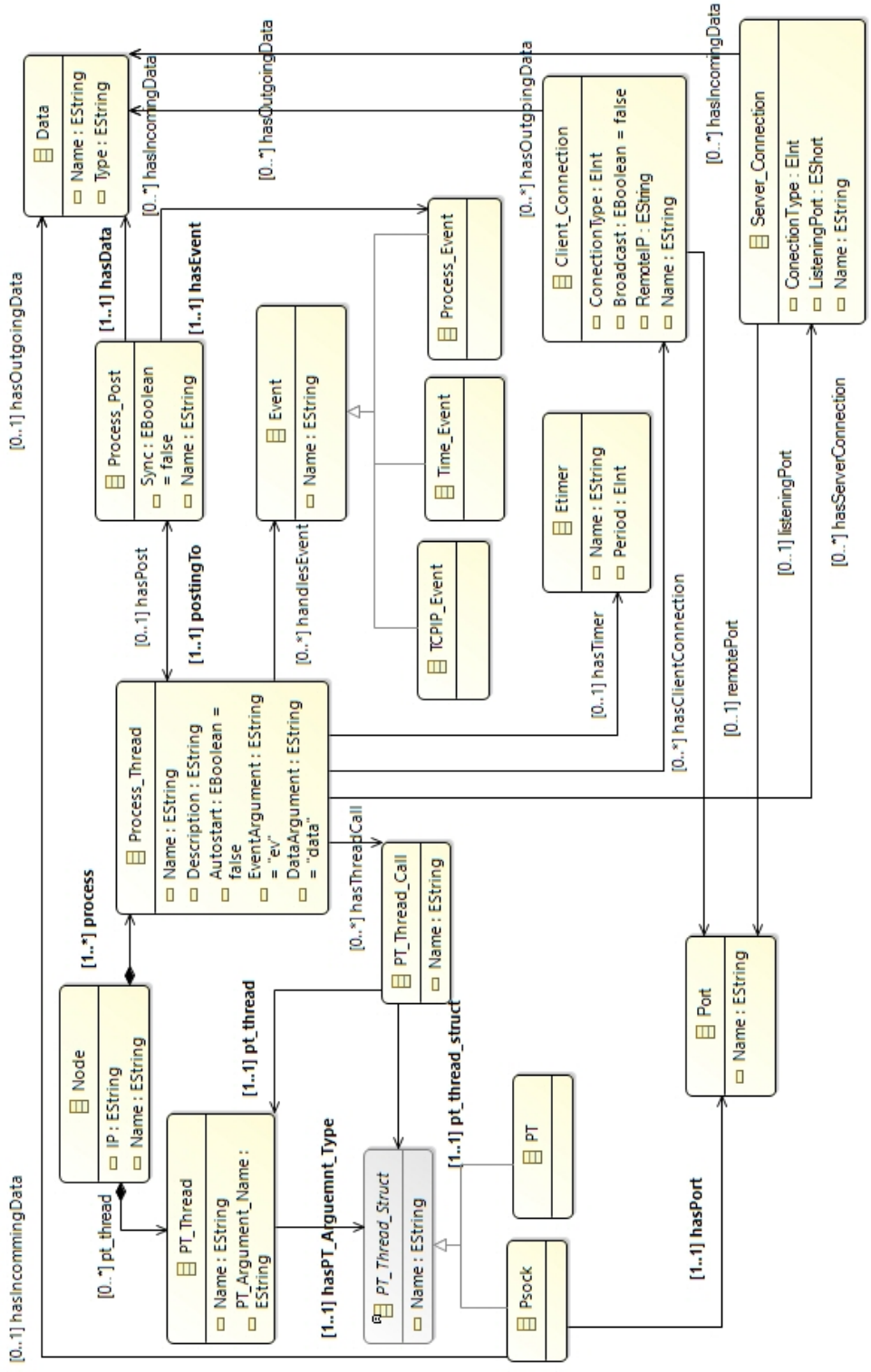



**Figure 2** A domain-specific metamodel for Contiki OS.

**Table 1** Some of the concepts and their notations for the Contiki modeling environment.

| Concept | Notation | Concept | Notation |
|---|---|---|---|
| Node |  | Data |  |
| Process_Thread |  | Process_Event |  |
| PT_Thread |  | TCPIP_Event |  |
| PT_Thread_Call |  | Port |  |
| Process_Post |  | Server_Connection |  |
| Etimer |  | Client_Connection |  |
| Event |  | Psock |  |
| Time_Event |  | PT |  |

Some of the graphics used for the concept representations are adopted and modified from Flaticon[3].

As it is shown in Table 1, the notations are selected in a way that the concepts with the same range of semantics have common symbols. For example, the elements with interior structure of processes and threads, such as *Process_Thread*, *PT_Thread*, *Process_Event*, and *PT*, have the same symbol of a gear as part of their notation. Similarly, the event related concepts, such as *Event*, *Time_Event*, *Process_Event*, and *TCPIP_Event* have the symbol of a screen as part of their notations. The notations are selected in a way that their presentation in both black-white or color will let the user to differentiate them from each other.

The metamodel discussed in the previous section is encoded in Ecore format inside Eclipse Modelling Framework (EMF[4]). Using this Ecore file, the notations depicted in Table 1, are mapped in Eclipse Epsilon Framework[5] to develop a graphical editor, as shown in Figure 3. To this end, the Ecore model, as our abstract syntax, is converted to Epsilon format which is used by Epsilon Eugenia tool. Then the required configurations are applied to inject the concrete syntax related information as some annotations in the Epsilon file.

Since we already provide some constraints in the abstract syntax (the metamodel), such as multiplicities, the graphical editor can check some of the connection rules during modelling, e.g. checking the source and target and also the number of relations for an element. Furthermore, we have benefited from the features of Eclipse Graphical Modeling Framework (GMF[6]) for automatically checking some consistency constraints when the model is modified, e.g. removing all the input/output links for an element when the element is removed from the model.

In addition to the GMF-based constraint checks, we have provided some domain rules to be checked to restrict the user to provide a more accurate model. This will lead to have artifacts with less errors in the generation phase. To this end, some static semantic rules

[3] http://www.flaticon.com
[4] http://www.eclipse.org/modeling/emf/
[5] http://www.eclipse.org/epsilon/
[6] https://www.eclipse.org/modeling/gmp/

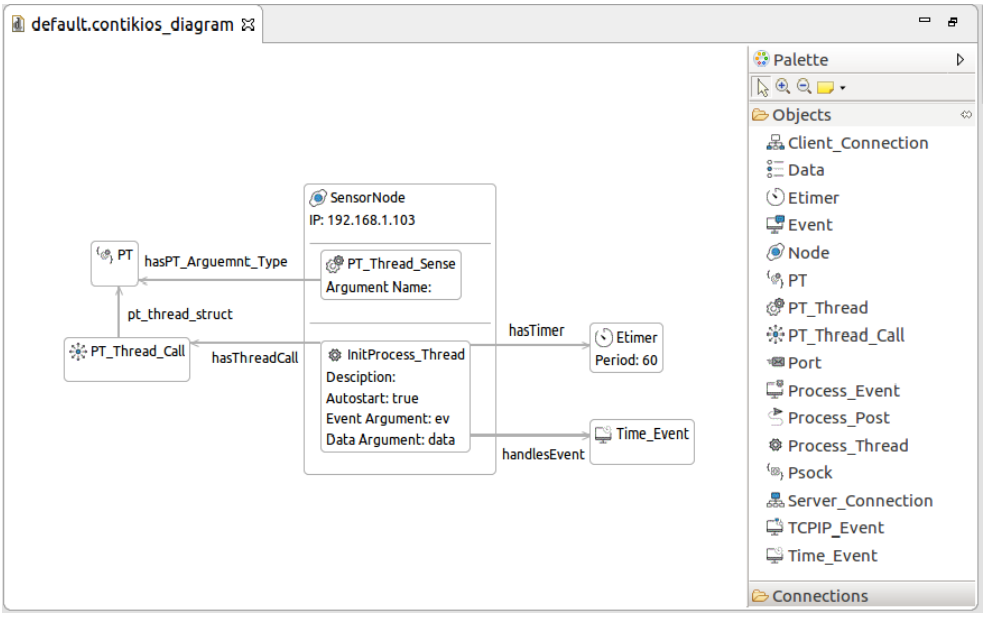



**Figure 3** A screenshot from the developed graphical editor for modeling Contiki-based IoT systems.

are provided for the system. These rules are implemented in Eclipse Validation Language (EVL[7]) to be integrated with the provided graphical modeling environment. Some of these rules are given below:

- A *Port* element in an instance model must have exactly one incoming link from a *Server_Connection*, but it should also have at least one incoming link from some *Client_Connections*. In this way, no Port element will be used improperly (without client side or server side links).
- The message payloads which are held in *Data* element are forced to be same in the relevant client and server connections.
- All client and server Connections are forced to be linked to *Process_Threads*, as each *Process_Threads* may have 0-* *Client_Connection* and/or *Server_Connection*.
- All events must be handled by a *Process_Thread*

The graphical editor developed in this study provides a convenient modeling environment in which the developers can create instance IoT models conforming to our metamodel by using the graphical concrete syntax. As can be seen in Figure 3, entities are listed in the palette residing in the right side of the graphical modeling editor. A developer can drag and drop an entity from this palette to create an instance of this entity. While the instances are added into the model, the related associations between the entities are automatically established and controlled by the defined constraints.

## 5 Case Study: Modeling an IoT System for Forest Fire Detection

There are many applications of WSNs [27] and IoT [19]. One of these applications is a Forest Fire Detection System [28]. In this application domain, IoT components are used to recognize

[7] `http://www.eclipse.org/epsilon/doc/evl/`

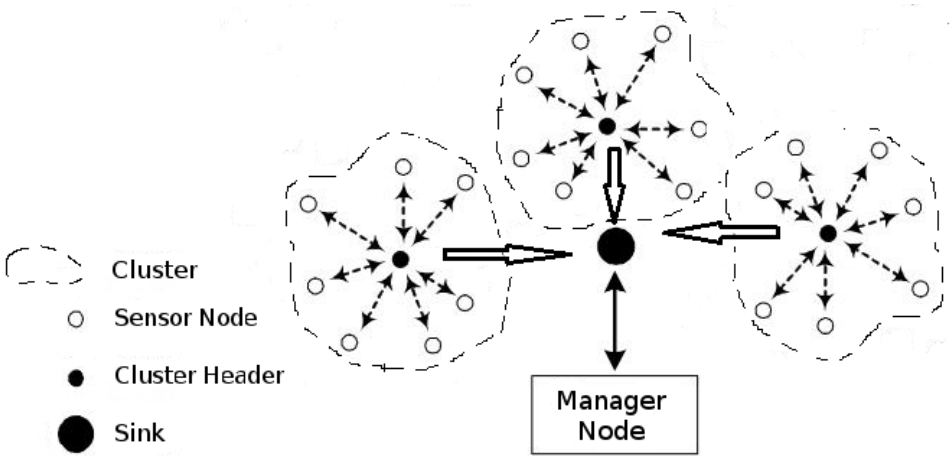



**Figure 4** A WSN for real-time forest fire detection.

the symptoms of a fire in an area such as a jungle and take in-time reactions. By the help of periodic measurements of some critical parameters in a wide area, the system predicts the risk of forest fire and warns in order to minimize the loss of forests, wild animals, and people in the forest. In this section, to give some flavour of using both the proposed metamodel and our modeling environment, we discuss the development of such an IoT system.

Densely deployed large number of sensor nodes collect measured data and send them to their respective cluster nodes that collaboratively process the data. Battery limitation forces WSN researchers to minimize power consumption. Since, communication among nodes consume more energy over computing, the option of sending aggregate data from sensor nodes to cluster header and from cluster header to sink node is chosen as an architectural design. The total size of messages transmitted in the system is reduced by clustering nodes and aggregating measured data.

The system specifications require collecting regular measurements, getting immediate fire alarms from nodes and querying instant measurements of a particular cluster from management office for analyzing. Tasks of cluster headers are to transfer raw messages among sink and sensor nodes, to prepare aggregate reports and to transfer them in the name of cluster. This use case is analyzed to extract the main scenarios of the system (see Figure 4). Sensor, cluster header and sink nodes in Figure 4 are considered by focusing on the internal node structure and communication of node groups.

Following the analysis, the system to be built is designed by using Contiki modeling tool previously introduced in Section 4. It is worth indicating that the modeling discussed in here takes into account the main process of the system and does not cover the setup procedure including clustering.

The instance model conforming to our metamodel for this case study is given in Figure 5. A *SensorNode* can measure environment temperature, relative humidity and smoke in every minute. *RegularProcess* in *SensorNode* gathers data of some consecutive measurements, e.g. 10 times, and posts their average to *SendProcess* to be sent to the respective cluster head. The node *ClusterHeader*, see Figure 5, calculates the weather index from the messages, which is held in *RR_Data* payload, in *SensorListenerProcess*. Then it sends the weather index which is encapsulated in process report called *PR_Data*, to *SinkNode* by the use of *ClientCon_Report* connection.

Each *SensorNode* can generate three classes of data packets:

1. Regular Report (*RR_Data*);
2. Query Response (*QR_Data*);
3. Emergence Report (*ER_Data*).

When a *SensorNode* detects an abnormal event in *RegularProcess*, e.g. smoke, it will immediately generate and send an *ER_Data* packet to the *ClusterHeader* containing the information related to the abnormal event without waiting for the rest of consecutive measurements, e.g. 10 times. *SensorListenerProcess* in *ClusterHeader*, uses a socket named *Pscok_ER* defined in *EmergencyThread* to transmit the emergency report by calling *AlarmCall* PT_Thread_Call.

The *QR_Data* packet is only transmitted to *SinkNode* when the sink asks *ClusterHeader* for the current measurement aspects defined in *QueryData*, e.g. temperature, which belongs to respective cluster. To implement this feature, *Sensor_Port* numbered '32900' is opened by *QueryProcesses* in *SensorNode* and *ClusterHeader* to accept instant data queries. *CurrentMeasurement* is sent back to *ClusterHeader* when a client connection is initiated. Then, *ClusterHeader* aggregates minimum, maximum, and average of the measurements from *SensorNodes* and constructs *QR_Data* to send the *SinkNode* via previously opened *ServerCon_Query*.

The resulting modeling artifact paves the way to do some analysis such as formal validation and verification of the system model based on the domain rules which can lead to less logical errors in the development of the system. Furthermore, the generation mechanism can be defined over this systematic modeling to generate the architectural code which can reduce the cost and number of errors for the embedded software development of IoT systems.

## 6 Related Work

Some MDE studies are provided in the literature to simplify the design, development, deployment and configuration of WSN and IoT systems. As there is considerable attention of researchers on applying MDE approaches on WSNs and IoT, two surveys ([11, 15]) studied systematic mapping of this domain to provide organized view of existing MDE approaches for WSN and IoT.

According to the above-mentioned surveys, among the reported studies, different MDE-based languages developed between the years 2007 and 2015. Furthermore, the modeling motivation of most of these studies is code-generation [11]. Among these studies, the code generation in nesC language for TinyOS is considered. However, none of these studies address Contiki OS.

LwiSSy is a Domain-specific Language (DSL) [6] to model Wireless Sensor and Actuators Network (WSAN) systems. LwiSSy allows the separation of responsibilities between domain expertise and network expertise. It also involves separation of structure, behavior, and optimization concerns by multiple views. In the study of [22], a model-driven architecture (MDA) is proposed which composes platform independent modeling (PIM), platform-specific modeling (PSM), and transformation rules for WSAN application development. PIM helps the domain experts to develop applications without knowing the programming details on WSAN platforms. PSM allows network experts to focus on the specific characteristics of their area of expertise without the need of knowing each specific application domain. Doddapaneni et al. [7] proposed a framework to model separately the software components and their interactions, the low-level and hardware specification of the nodes, and the physical

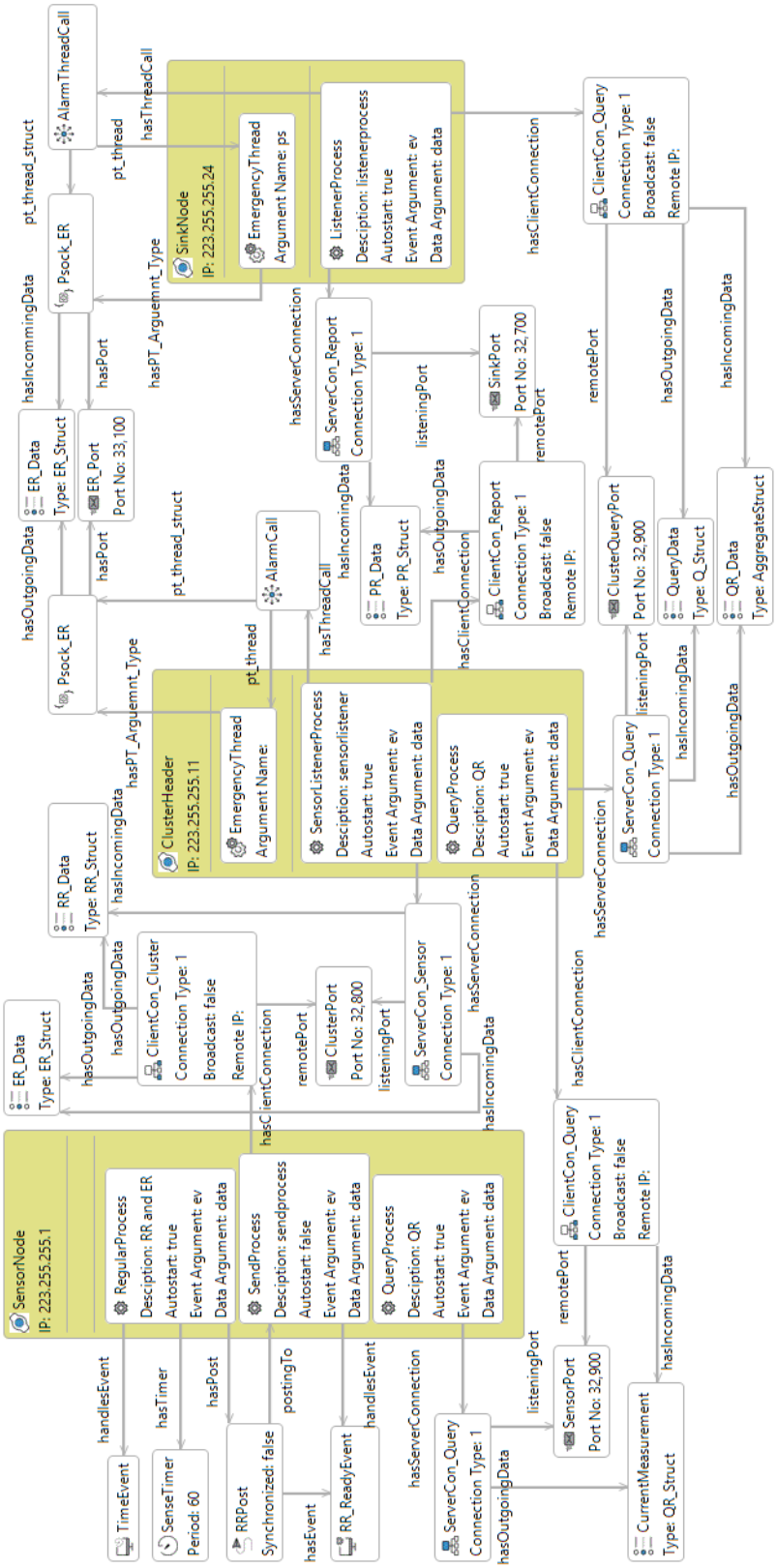



**Figure 5** The instance model for the forest fire detection system.

environment where the nodes are deployed. This multi-view architectural approach [8] requires linking the models together for mapping models.

Tei et al. [25] propose a process that enables stepwise refinement to separately address data processing-related and network-related concerns. Their approach is similar to the modeling purpose of Rodrigues et al. [21] which especially takes into consideration the separation of responsibilities between domain experts and network experts. Rodrigues et al. [21] propose PIM for domain experts and PSM for network experts which can be used for the MDE of systems working on TinyOS. However, Tei et al. [25] have limited support for experts. PSM is not supported in their study and the experts simply create templates over platform independent models.

As also indicated in Section 1, our work contributes to the aforementioned noteworthy studies in the way of providing an MDE for developing IoT systems based on Contiki OS. To the best of our knowledge, currently no study addresses modelling WSNs or IoT systems according to the specifications of Contiki platform. The unique lightweight thread structure of Contiki makes it more popular in the implementation of new IoT systems and conduces the developers preferring Contiki instead of many other existing platforms such as TinyOS and SOS. Hence, providing a modeling language as proposed in this study can facilitate the efficient development of IoT systems based on this fashionable OS.

## 7 Conclusion

A metamodel and its supporting graphical modeling environment for the MDE of IoT systems are discussed in this paper. The metamodel includes all entities and relations required for modeling systems according to the event-driven mechanism of Contiki OS. Modeling based on the Contiki protothread architecture is also possible with using this metamodel. Moreover, a concrete syntax has been derived from this metamodel. Developers can use the proposed modeling language inside a graphical modeling environment to design the IoT systems as described in the conducted use case study. Both the modeling editor and the instance model discussed in this paper are available online with including required installation and configuration instructions at: `http://serlab.ube.ege.edu.tr/Bundles/ContikiOS_Editor.zip`.

The work discussed herein is our initial effort towards providing a full-fledged MDE environment for the development of Contiki-based IoT systems. Our next work will include design and implementation of model-to-text transformations which enable the automatic code generation from the modeled systems. This facility will be built-in for the existing Eclipse-based graphical modeling environment and hence the developers quickly achieve the required codes for deploying the designed systems on the embedded devices running Contiki OS.

## References

1 Vahid Khalilpour Akram and Orhan Dagdeviren. Breadth-first search-based single-phase algorithms for bridge detection in wireless sensor networks. *Sensors*, 13(7):8786–8813, 2013.

2 Lan S. Bai, Robert P. Dick, and Peter A. Dinda. Archetype-based design: Sensor network programming for application experts, not just programming experts. In *2009 International Conference on Information Processing in Sensor Networks*, pages 85–96. IEEE, 2009.

[8] ISO/IEC/IEEE 42010:2011: `https://www.iso.org/standard/50508.html`.

3 Pruet Boonma, Yuthapong Somchit, and Juggapong Natwichai. A model-driven engineering platform for wireless sensor networks. In *Eighth International Conference on P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC)*, pages 671–676, 2013.

4 Moharram Challenger, Ferhat Erata, Mehmet Onat, Hale Gezgen, and Geylani Kardas. A model-driven engineering technique for developing composite content applications. In Marjan Mernik, José Paulo Leal, and Hugo Gonçalo Oliveira, editors, *5th Symposium on Languages, Applications and Technologies (SLATE'16)*, volume 51 of *OpenAccess Series in Informatics (OASIcs)*, pages 1–10, 2016.

5 Orhan Dagdeviren and Vahid Khalilpour Akram. An energy-efficient distributed cut vertex detection algorithm for wireless sensor networks. *The Computer Journal*, 57(12):1852–1869, 2014.

6 Priscilla Dantas, Taniro Rodrigues, Thais Batista, Flavia C. Delicato, Paulo F. Pires, Wei Li, and Albert Y. Zomaya. LWiSSy: a domain specific language to model wireless sensor and actuators network systems. In *4th International Workshop on Software Engineering for Sensor Network Applications (SESENA)*, pages 7–12. IEEE, May 2013.

7 Krishna Doddapaneni, Enver Ever, Orhan Gemikonakli, Ivano Malavolta, Leonardo Mostarda, and Henry Muccini. A model-driven engineering framework for architecting and analysing wireless sensor networks. In *Third International Workshop on Software Engineering for Sensor Network Applications (SESENA)*, pages 1–7. IEEE, June 2012.

8 Adam Dunkels, Niclas Finne, Joakim Eriksson, and Thiemo Voigt. Run-time dynamic linking for reprogramming wireless sensor networks. In *International conference on Embedded networked sensor systems – SenSys'06*. ACM Press, 2006.

9 Adam Dunkels, B. Gronvall, and Thiemo Voigt. Contiki – a lightweight and flexible operating system for tiny networked sensors. In *29th Annual IEEE International Conference on Local Computer Networks*, pages 455–462. IEEE, 2004.

10 Adam Dunkels, Oliver Schmidt, Thiemo Voigt, and Muneeb Ali. Protothreads: simplifying event-driven programming of memory-constrained embedded systems. In *4th international conference on Embedded networked sensor systems*, pages 29–42. ACM, 2006.

11 Fatima Essaadi, Yann Ben Maissa, and Mohammed Dahchour. MDE-based languages for wireless sensor networks modeling: A systematic mapping study. In *Advances in Ubiquitous Networking 2*, pages 331–346. Springer, 2017.

12 Jayavardhana Gubbi, Rajkumar Buyya, Slaven Marusic, and Marimuthu Palaniswami. Internet of Things (IoT): A vision, architectural elements, and future directions. *Future Generation Computer Systems*, 29(7):1645–1660, September 2013.

13 Chih-Chieh Han, Ram Kumar, Roy Shea, Eddie Kohler, and Mani Srivastava. A dynamic operating system for sensor nodes. In *3rd international conference on Mobile systems, applications, and services – MobiSys'05*, page 163. ACM Press, 2005.

14 Jason Hill, Robert Szewczyk, Alec Woo, Seth Hollar, David Culler, and Kristofer Pister. System architecture directions for networked sensors. *ACM SIGOPS operating systems review*, 34(5):93–104, 2000.

15 Ivano Malavolta and Henry Muccini. A study on MDE approaches for engineering wireless sensor networks. In *40th EUROMICRO Conference on Software Engineering and Advanced Applications*, pages 149–157. IEEE, August 2014.

16 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.

17 Daniele Miorandi, Sabrina Sicari, Francesco De Pellegrini, and Imrich Chlamtac. Internet of Things: vision, applications and research challenges. *Ad Hoc Networks*, 10(7):1497–1516, September 2012.

18 Luca Mottola and Gian Pietro Picco. Programming wireless sensor networks with logical neighborhoods. In *First international conference on Integrated internet ad hoc and sensor networks – InterSense'06*, page 8. ACM Press, April 2006.

19 Shayan Nalbandian. A survey on Internet of Things: Applications and challenges. In *International Congress on Technology, Communication and Knowledge (ICTCK)*, pages 165–169. IEEE, 2015.

20 Tobias Reusing. Comparison of operating systems tinyos and contiki. *Network Architectures and Services*, 7:7–13, 2012.

21 Taniro Rodrigues, Priscilla Dantas, Paulo F Pires, Luci Pirmez, Thais Batista, Claudio Miceli, Albert Zomaya, et al. Model-driven development of wireless sensor network applications. In *IFIP 9th International Conference on Embedded and Ubiquitous Computing*, pages 11–18. IEEE, 2011.

22 Taniro Rodrigues, Flávia C. Delicato, Thais Batista, Paulo F. Pires, and Luci Pirmez. An approach based on the domain perspective to develop WSAN applications. *Software & Systems Modeling*, pages 1–29, September 2015.

23 Douglas C Schmidt. Model-driven engineering. *IEEE COMPUTER*, 39(2):25, 2006.

24 Zach Shelby and Carsten Bormann. *6LoWPAN: The Wireless Embedded Internet*. John Wiley & Sons, Ltd, Chichester, UK, November 2009.

25 Kenji Tei, Ryo Shimizu, Yoshiaki Fukazawa, and Shinichi Honiden. Model-driven-development-based stepwise software development process for wireless sensor networks. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 45(4):675–687, April 2015.

26 Jon Whittle, John Hutchinson, and Mark Rouncefield. The state of practice in model-driven engineering. *IEEE software*, 31(3):79–85, 2014.

27 Ning Xu. A survey of sensor network applications. *IEEE communications magazine*, 40(8):102–114, 2002.

28 Liyang Yu, Neng Wang, and Xiaoqiao Meng. Real-time forest fire detection with wireless sensor networks. In *International Conference on Wireless Communications, Networking and Mobile Computing*, volume 2, pages 1214–1217. IEEE, 2005.

# Exercise Solution Check Specification Language for Interactive Programming Learning Environments

**Jakub Swacha**

**Institute of Information Technology in Management, University of Szczecin, Szczecin, Poland**
jakubs@uoo.univ.szczecin.pl

**Abstract**

Automatic checking of the correctness of students' solutions of programming exercises for generating appropriate feedback is a necessary component of interactive programming learning environments. Although there are multiple ways of specifying such a check, ranging from mere string patterns to code written in general-purpose programming language, they all have their deficiencies, with the check specification being too verbose, too complicated, difficult to reuse, or very limited in its expressive capabilities. In this paper, a new language designed especially for this purpose is described. It provides both extension and replacement for RegEx-based pattern specification so that checks typical for programming exercise verification can be expressed in a concise and highly-readable manner.

**1998 ACM Subject Classification** D.3.2 [Language Classifications] Specialized Application Languages

**Keywords and phrases** automatic programming exercise solution verification, source code pattern specification, RegEx extension, RegEx alternative

**Digital Object Identifier** 10.4230/OASIcs.SLATE.2017.6

## 1 Introduction

Learning programming is difficult (see [1] and works cited therein). A highly important element of this process is letting the students practice with writing their own code by providing them with adequately chosen programming exercises. It may lead to substantial learning progress if only the students receive relevant feedback after they submit their solutions of the exercises. In traditional learning environments, checking the students' solutions and giving them feedback belongs to the instructor. In interactive programming learning environments, most of this process is automated, so that the instructor can focus better on other types of teaching activities.

The downside is that the instructor who is preparing an exercise for students not only has to conceive it and write its description (the aim and rules, possibly also expected results), but also specify the automatic checks and feedback generation rules. From this author's experience, based on the development of an interactive course of Python [8], if the course is intended for small student groups, the overall time spent on specifying the checks is greater than time spent on traditional checking of the exercises. Much of the reason for it lies in the deficiencies in the form of specification of the automatic checks (see the following subsection). An obvious solution for this problem would be to use a better form of specification, designed especially for this purpose and thus free of the most frequent and onerous nuisances. The goal of this paper is to describe such a solution, in a form of a domain-specific language.


## 2 Problem setting

The most basic automatic checking of programming exercise solution takes into consideration only the output it produces after its execution (*black-box testing*). As the exercise description may impose restrictions on the programming language constructs or functions allowed in the solution, or require to apply certain constructs, functions, algorithms, programming techniques or design patterns, also the source code of the solution has to be checked to verify that, in addition providing valuable input for generating meaningful feedback to the students. Moreover, interpreter syntax check and execution environment error messages may be checked to transform them into a form that would be related to the actual exercise description and more intelligible to the students.

There are various ways that can be used to specify such checks [5, p. 44]. In the case of interactive programming learning environments (with limitations of their own), the two most obvious alternatives are to use a general-purpose programming language (not necessarily the language the solutions are written in) or RegEx-based patterns [9]. The general-purpose programming languages are designed for far more than exercise checking, therefore the check code, in all but trivial examples, is somewhat verbose, and it is not easy to tell after a single look at it what is actually checked. It makes them difficult to reuse and may cause a need for translation when transferring the exercises to another learning environment. Moreover, the actual match patterns are often expressed using RegEx expressions embedded in the general-purpose programming language used, which makes the issues specific to RegEx (see below) still valid.

RegEx (short from regular expressions, though it extends regular languages as known from theory [3]) is a lingua franca of text pattern specification in today's computing world. Nonetheless, as Lowell D. Thomas stated, the RegEx-defined patterns are often "hard to read, even harder to write, and hard to maintain" [11].

The intrinsic issues with RegEx readability are significantly augmented by the type of content that is searched through (i.e., especially, program source code) and the character of patterns to be matched. Selected examples of those are discussed below.

For instance, many programming languages allow optional whitespaces. Accommodating RegEx expressions for it results in single spaces being replaced with wildcards, obfuscating the pattern specification, e.g.:

```
/\(\s*limit\s*=\s*50\s*\)/
```

Similar issues are due to alternative notation of quotes (single or double) or certain operators (e.g. `<>` and `!=`) allowed in some languages (e.g. Python).

RegEx defines a number of special characters (e.g. `.`, `(`, `)`, `[` or `*` which have to be escaped in pattern specification. The same characters are very frequently used in source code of most programming languages, which leads to obfuscation of the pattern specification by insertion of backslashes, e.g.:

```
/\*\*|f\.a\s*\*f\.a/
```

The problem grows (along with the number of inserted backslashes) when the checked source code itself is expected to contain backslash-escaped characters (due to the requirements of the programming language notation), e.g.:

```
/1\.\.\.\\n\\t2\.\.\.\\n/
```

Most of the patterns looked for in the solution source code are expected to match valid program instructions, but they can as well match within literal string constants and comments.

In order to overcome that, the pattern actually sought has to be prepended with another one which ensures invalid code is ignored, increasing the total pattern complexity. The example below is intended for Python and it does not even cover multi-line strings and escaped backslashes:

```
/^(?:[^#"']*(?:'(?:[^']|\\')*'|"(?:[^"]|\\")*"))*[^#"']*\bfor\b/
```

Moreover, the patterns are often used to check the solution's structure, i.e., that specific code is found within specific logical block (e.g. an `if` statement). This requires a sophisticated RegEx in the case of languages forming logical blocks by indentation (such as Python), e.g.:

```
/if\s+x\s*<\s*0\s*:(?:\s*|\n(\s+)(?:.*\n\1))x\s*=\s*-\s*x/
```

The exercise solution often has to produce a specified, exact number of results. RegEx expressions checking non-occurrence of specific elements look awkward and contribute to low readability of pattern specification, e.g.:

```
/^(?:[^i]|i[^f]|\Bif|if\B)*\bif\b(?:[^i]|i[^f]|\Bif|if\B)*$/
```

A similar problem is when the exercise solution has to produce results in a specified order (e.g. sorted ascendingly). Ensuring that using RegEx expression is a nuisance, as it requires explicit listing of all the possible expected values, e.g.:

```
/^\D*(?:0\D+)*(?:1\D+)*(?:2\D+)*(?:3\D+)*(?:4\D+)*(?:5\D+)*
(?:6\D+)*(?:7\D+)*(?:8\D+)*(?:9\D+)*(?:10\D+)*$/
```

There is even bigger problem if more than one of alternative patterns has to be found in the source code, in unknown order. The example below is for an exercise which expected the student to use at least two different parameter orders when calling a function taking three parameters:

```
/\(\s*(?:(?:a_1\D+)?1\D+\d\D+(\d)[\s\S]+\(\s*(?:(?:a_1\D+)?
1\D+\1|\D+[23])|\D+([23])\D+(?:a_1\D+)?(\d)\D+(\d)[\s\S]+
\(\D*(?:a_1\D+)?(?:\2\D+\4|(?:\3|\4)))/
```

## 3 Related work

The unreadability of RegEx prompted search for alternative ways of pattern specification. A good example of such is *apg-exp* [11], using a superset of Augmented Backus-Naur Form (ABNF) [2]. The most notable extensions of the original ABNF are the positive and negative look ahead/behind operators which make specification of context-relevant patterns easy. Although *apg-exp* brings a visible improvement in readability of pattern specification, it is hardly concise. And while it fits well with programming language syntax definitions which are usually expressed in some form of BNF, *apg-exp* was designed as a general-purpose replacement for RegEx and thus provides no direct shortcuts for exercise solution checking.

Looking at the existing work in this area, however, there are no solutions that fit exactly such needs. The specification format for programming exercises, PExIL (Programming Exercises Interoperability Language) defines elements (within *specification* element) only for output check and feedback specification [6]. The automatic assessment solutions that transform solution code into abstract syntax trees or graphs use standard query languages, respectively, XPath and GReQL, to define the checks [7].

Perhaps the most promising of the existing solutions is the domain-specific language proposed by Hadiwijaya and Liem for an "automatic generation of the output, input, and

source code checkers" [4]. However, only the output- and input-checking is based entirely on the syntax native to that language, whereas source code checking (which demands the most sophisticated checks) relies on RegEx expressions.

# 4 Language Specification

## 4.1 Text pattern specification

The proposed language is designed as both a replacement and an extension of RegEx. Regarding the latter, it introduces a number of textual instructions and operators that allow to specify additional requirements for the specified patterns to match. Regarding the former, it provides four ways of pattern notation:

- `word`: a sequence of non-space characters (from a limited set, mostly alpha-numeric), intended to match single numbers, instruction names and identifiers; note there is no need for delimiters other than whitespace;
- `words`: a sequence of characters delimited with apostrophes; it has special properties: a space matches any whitespace sequence, a double quotation mark matches any language-defined literal string constant delimiter, code comments within matched text are ignored, it may contain variable references (they have to be delimited on both ends with `$` or `#` characters - the difference is explained below);
- `string`: a sequence of characters delimited with double quotation marks; they match literally the given string (no wildcards, no special properties);
- `regex`: a RegEx expression (JavaScript flavor) delimited with slashes.

## 4.2 Referencing and combining patterns

All the available pattern notations can be used interchangeably, and even combined to form compound pattern expressions, as well as labeled for later reuse using `->` operator, e.g. the following code will look for decimal digits after `width =` , and label the numeric pattern as `$width`:

```
'width = ',/\d+/ -> $width
```

The labeled patterns can be referenced both by definition (to look for similar patterns elsewhere) or by value (to look for repeating occurrences of the value matched by the pattern earlier), e.g. the following code references the pattern defining variable identifiers allowed in Python (`$pyvar`, assumed to be defined earlier; not a part of the specification language) and the value last matched by pattern `$width` (also assumed to be defined earlier):

```
'for $pyvar$ in range ( #width# ) :'
```

Note the spaces in the above listing will be matched by any whitespace combination.

The patterns can be combined using the following operators (and round brackets):

- `seq`: match all the elements in the given order (also applies if no operator is specified),
- `each`: match all the elements in any order,
- `none`: match none of the elements,
- `any`: match at least one of the elements,
- `select` *`num`*: match exactly *num* of the elements in any order.

For example, the following code will report match only if there is either *for* or *while*, but not a combination of these (as when using `any`):

```
select 1 (for,while)
```

## 4.3 Match quantity requirement specification

Using the following operators, the expected number of matches can be specified:

- `just` *num*: the pattern must match exactly *num* times,
- `atleast` *num*: the pattern must match *num* or more times,
- `atmost` *num*: the pattern must match *num* or less times,
- `between` *num1* `and` *num2*: the pattern must match at least *num1* and at most *num2* times,
- `multiply` *num*: the pattern must match *num***x* times, where *x* is any integer greater than 0.

For instance, the following code will report match only if the number of occurrences of two-digit numbers is even:

```
multiply 2 /{\b\d\d\b}/
```

## 4.4 Order and uniqueness requirement specification

Using the following operators, additional requirements in case of multiple matches can be specified:

- `distinct`: none of the fragments matched by the pattern can repeat,
- `same`: each fragment matched by the pattern must have the same content,
- `incr`: each subsequent fragment matched by the pattern must have greater value (lexicographic order is used for non-decimals);
- `decr`: each subsequent fragment matched by the pattern must have smaller value (lexicographic order is used for non-decimals).

For instance, the following code will report match only if there occurs at least one number and none of the matched numbers repeats:

```
distinct /\b\d+\b/
```

## 4.5 Position requirement specification

By default, the pattern is matched in the whole text (i.e., the input, output, or source code of the exercise solution). This can be changed by explicitly defining match position using one of the following operators:

- `in` *block*: the pattern will be matched inside the *block* (see below for details),
- `after` *block*: the pattern will be matched no sooner than the block ends,
- `follows` *block*: the pattern will be matched right after the block ends (only language-specific whitespaces are allowed in-between).

The *block* can be specified either by type only (the pattern will be matched in each block of that type) or also specified by pattern (the sought pattern will be matched only in those blocks of that type which also match the context pattern). There are five types of blocks defined:

- `bracket`: matches only within the specified kind of brackets (one of `()`, `<>`, `[]`, and `{}`); in source code, brackets outside of language-specific valid code (e.g. inside comments or literal string constants) are ignored;
- `line`: matches only within a single line (having source-language-specific boundaries, e.g. `\` may disable line end in some languages);

- `compound`: matches only in the specified compound statement, having source-language-specific boundaries; this is designed especially for languages which do not use brackets (like `{}` in C) to delimit compound statements, as e.g. Python; the context phrase matches from the beginning of the compound statement (i.e. including its header);
- `string`: matches only within a single string (having source-language-specific boundaries, e.g. `\"` may disable string end in some languages);
- `comment`: matches only within code comments.

Note that the context pattern can also have the position requirement specified, therefore the following code is correct and matches `5` as `range` function parameters in a line containing `=` within `if` statement block inside of `for` loop:

```
5 in bracket 'range (' in line '=' in compound if in compound for
```

## 4.6 Assessment result and feedback generation

The result of a match may trigger acceptance or rejection of the solution. The language defines two instructions serving this purpose:

- `req`: considers the solution incorrect and generates feedback if the pattern does not match;
- `forbid`: considers the solution incorrect and generates feedback if the pattern matches.

The appropriate feedback for the student is specified using the `=>` operator. For instance, the following code will report an incorrect solution and provide the specified feedback (`Incorrect number`) only if there is `1` or `2` or `3` (or any combination of these numbers):

```
forbid any(1,2,3) => Incorrect number
```

## 4.7 Conditional requirements

The result of a match can be stored in a variable. A match can be executed on a condition defined with a specified Boolean expression which can use three Boolean operators (`not`, `and`, `or`) and reference any variables set earlier. This allows for, e.g., performing checks and generating hints depending on a number of factors. For instance, the following code will require only one of the three specified patterns:

```
$v1 = 'x = -x' in compound 'if x < 0'
$v2 = 'x = -x' in compound 'else' follows compound 'if x >= 0'
$v3 = 'abs ( x )'
if not $v1 and not $v2 req $v3 => Calculate the absolute value
```

# 5 Implementation and validation

The proposed language has been defined using ABNF [2]. Its parser has been generated automatically in JavaScript using APG [12] and served as the basis for a proof-of-concept implementation of its interpreter, tuned for checking solutions written in Python – it can still be easily adapted to any other programming language by replacing a set of callback functions.

The language and its interpreter were validated using multiple pattern examples from real-world programming exercises from the course mentioned earlier [8]. No major issues were

encountered, and minor issues were used as a base for improvements in language specification and its interpreter. Among others, the eight RegEx patterns presented in Section 2 of this paper were successfully translated to the proposed language (note its conciseness and high readability):

1. `( limit = 50 )'`
2. `any('f.a ** 2', 'f.a * f.a', 'pow ( f.a , 2 )')`
3. `'1...nt2...n'`
4. `for`
5. `'x = -x' in compound 'if x < 0'`
6. `just 1 if`
7. `incr /b1?db/`
8. `distinct 2 each ( any ('( 1','a_1 = 1'), 2, 3) in bracket '(' )`

## 6 Conclusion

The instant feedback based on automatic checking of the correctness of students' solutions should be considered as a crucial component of every interactive programming learning environment. The specificity of such an intended use (large number of short exercises compared to small number of relatively long exercises used usually in programming contests) results in significant work effort needed to specify the acceptance and feedback rules even for a single course. The described deficiencies of existing forms of specification often turn even simple checks into verbose, complicated and difficult to reuse pattern definitions. The new language introduced in this paper provides ways to specify checks typical for programming exercise verification in a concise and highly-readable manner.

Currently, there is work undergoing on developing a new open specification format for interactive programming exercises which will feature the language described in this paper [10]. In the next step, the mentioned interactive Python course [8] will be converted to the new format, and the check code of all its exercises will be translated to the proposed language, which will confirm its usefulness and also provide statistical evidence on its conciseness.

## References


1 Yorah Bosse and Marco Aurélio Gerosa. Why is programming so difficult to learn?: Patterns of difficulties related to programming learning mid-stage. *ACM SIGSOFT Software Engineering Notes*, 41(6):1–6, 2017.

2 David H. Crocker and Paul Overell. Augmented BNF for syntax specifications: ABNF. RFC 5234, IETF, January 2008.

3 Cezar Câmpeanu, Kai Salomaa, and Sheng Yu. Regex and extended regex. In Jean-Marc Champarnaud and Denis Maurel, editors, *Implementation and Application of Automata*, volume 2608, pages 77–84. Springer, 2003.

4 Ryan Ignatius Hadiwijaya and M. M. Inggriani Liem. A domain-specific language for automatic generation of checkers. In *International Conference on Data and Software Engineering*, pages 7–12, 2015.

5 Hieke Keuning, Johan Jeuring, and Bastiaan Heeren. Towards a systematic review of automated feedback generation for programming exercises. In *Conference on Innovation and Technology in Computer Science Education*, pages 41–46, 2016.

6 Ricardo Queirós and José Paulo Leal. Making programming exercises interoperable with PExIL. In José Carlos Ramalho, Alberto Simões, and Ricardo Queirós, editors, *Innovations in XML Applications and Metadata Management: Advancing Technologies*, pages 38–56. IGI Global, 2013.

7 Michael Striewe and Michael Goedicke. A review of static analysis approaches for programming exercisesd. In Marco Kalz and Eric Ras, editors, *Computer Assisted Assessment. Research into E-Assessment*, volume 439, pages 100–113. Springer, 2014.

8 Jakub Swacha. An interactive Python course: development and evaluation. Forthcoming, 2017.

9 Jakub Swacha. Scripting environments of gamified learning management systems for programming education. In Ricardo Queirós and Mário Pinto, editors, *Gamification-Based E-Learning Strategies for Computer Programming Education*, pages 278–294. IGI Global, 2017.

10 Jakub Swacha. SIPE: a domain-specific language for specifying interactive programming exercises. Forthcoming, 2017.

11 Lowell D. Thomas. An alternative to regular expressions: apg-exp, July 2016. SitePoint. `http://www.sitepoint.com/alternative-to-regular-expressions`.

12 Lowell D. Thomas. JavaScript APG, 2017. Coast to Coast Research. `http://www.coasttocoastresearch.com/docjs2/apg/index.html`.

# Visualizing the Evaluation of Functional Programs for Debugging

**John Whitington[1] and Tom Ridge[2]**

1 **University of Leicester, Leicester, UK**
`jw642@le.ac.uk`
2 **University of Leicester, Leicester, UK**
`tr61@le.ac.uk`

## Abstract

In this position paper, we present a prototype of a visualizer for functional programs. Such programs, whose evaluation model is the reduction of an expression to a value through repeated application of rewriting rules, and which tend to make little or no use of mutable state, are amenable to visualization in the same fashion as simple mathematical expressions, with which every schoolchild is familiar. We show how such visualizations may be produced for the strict functional language OCaml, by direct interpretation of the abstract syntax tree and appropriate pretty-printing. We describe (and begin to address) the challenges of presenting such program traces in limited space and of identifying their essential elements, so that our methods will one day be practical for more than toy programs. We consider the problems posed by the parts of modern functional programming which are not purely functional such as mutable state, input/output and exceptions. We describe initial work on the use of such visualizations to address the problem of program debugging, which is our ultimate aim.

**1998 ACM Subject Classification** D.2.5 Testing and Debugging

**Keywords and phrases** Debugging, Functional, Visualization, OCaml

**Digital Object Identifier** 10.4230/OASIcs.SLATE.2017.7

## 1 Introduction

When we do mathematics on paper, we write an expression or equation and, through a series of legal transformations, produce a simpler one that, we hope, tells us what we want to know. It is the same with functional programming, but the semantics define more closely the order in which the expression is evaluated, choosing each transformation by inspection of the shape of the expression. Of course, this is not quite how the compiled code runs, but it is the mental model. So it is natural, when teaching students functional programming, to proceed by analogy to the mathematical model in which they are already well-practiced. Since computer languages must be more formal in their choice of evaluation order, we tend to underline the sub-expression being evaluated at each step. Given the function `f` for doubling a number, we might have:

```
      f 3 = 1 + 2 * 3
⟹    6 = 1 + 2 * 3
⟹    6 = 1 + 6
⟹    6 = 7
⟹    false
```





There are differences from mathematics, of course: the last step may be rather confusing to the schoolchild (our equals sign does not denote an equation as such, but a comparison operator). Such visualizations are longwinded to write on paper, for all but the least substantial programs. We should like to generate them by computer. In order to provide a tool useful to both learners and the everyday programmer, we will begin with a subset of a real language (rather than building our own toy one), extend it to work for the whole language, and integrate it properly with the toolchain as a first class citizen. We are writing, in essence, a step-by-step interpreter.

What is the relevance to debugging? The dream of debugging is this: having observed a misbehaviour caused by a bug, we assemble all relevant information, both about the program source and the full trace of the program's operation, and, describing ourselves concisely to the computer, we narrow the circumstances down, again and again reducing the search space, until we have the bug in our grasp, and understand it fully. The fix is then often easy. But we are very far from this dream, even today. In his introduction to a Special Issue of Communications of the ACM in 1997 "The Debugging Scandal and What to Do About It" [5], Lieberman writes: *"Today's commercial programming environments provide debugging tools that are little better than the tools that came with programming environments thirty years ago. It is a sad commentary on the state of the art that many programmers name* "inserting print statements" *as their debugging technique of choice."* We claim this is still largely true, twenty years later.

## 2 Related Work

Two useful surveys [16, 17] give a general overview of recent developments in this area, the first specific to functional programming, the second with wider scope. A very broad introduction [8] gives background. A comprehensive survey [11] of education systems for program visualization is useful too. We pick out a few recent systems for further discussion.

The *WinHIPE* system [7] is a recent incarnation of these ideas for the HOPE language. It uses a step-by-step evaluation system, and explicitly addresses the problems of scale by elision of information and a focusing mechanism. The emphasis, however, is on graphical (tree-based) representations, an approach we shall not take, being of the belief that trees can often be, in fact, harder to read than well-pretty-printed program representations. The *Visual Miranda Machine* [1] provides a trace of the evaluation of a lazy functional program, together with a commentary showing the reason for choosing each evaluation step. There is a discussion of granularity, taking the example of the "list comprehension" language feature. *DrScheme* [2] provides, amongst many other facilities, an "algebraic stepper" for the Scheme language that can print out steps of evaluation. The stepper, however, supports only a subset of the language. The implementation is interesting, though – it reuses some of the underlying Scheme implementation to ensure equivalent semantics. Touretsky describes a LISP-based system [14] that produces mainly textual traces, but with some use of graphical elements to indicate the different scoping mechanisms peculiar to LISP. The presentation of *ZStep95* [15] begins by noting that debugging is, essentially, a human interface problem. The authors concentrate on the concept of *immediacy* (temporal, spatial, and so on), which they see as essential, and exhibit a stepping debugger for a functional language which can go back and forth through time. There is a system bolted onto Haskell [3] which is not an interpreter as such, but allows the programmer to insert observation points in the code, within the syntax of the source language. At such points, the values are printed out. This has the advantage of explicitly dealing with at least part of the problem of scale – only what we wish to be printed

is printed. Another approach to this problem is as a special case of the more general concept of a *calculator* [9, 4], showing how it pertains to various logical systems with a mathematical basis, not just functional programs. *Prospero* [12] is a more fully-developed system, again for a lazy language. It includes methods for filtering the evaluation trace to elide information and a careful discussion of usability issues.

These systems are mostly concerned with program visualization for teaching; we wish to bias ourselves towards the task of general debugging, hoping that some of the educational uses will be subsumed by it (the authors of DrScheme [2] urge caution here, choosing instead to build a "tower" of syntactically restrictive variants of Scheme specifically for educational purposes. They say that, due to the fact that so many sequences of characters are syntactically valid in Scheme, error messages are less confusing when the dialect is restricted – we would prefer to avoid this in the name of universality).

It is worth pointing out that much research in software visualization concerns overtly graphical approaches. We take a simpler line, sticking to pretty-printing. We claim that the most important aspect of a successful visualization is elision – reducing the information visible to just what is required so that large datasets may be understood easily – whether interactively or not. Programmers are used to seeing their program as text, and visualizing its evaluation as, for example, a graphical tree structure, is less useful for debugging large programs (it can be useful, of course, for visualizing program source code structure as opposed to evaluation traces).

## 3 Rationale

We surveyed users of the strict functional programming language OCaml informally to ask whether they routinely used debuggers, and if not, why not. The overwhelming result was that debuggers are not widely used. The Haskell community has found the same [6]. Several respondents honed in on a theme: *"I use tools that I am familiar with when debugging because I don't want to focus on two things (learning a new tool and tracking down/fixing a bug)."* One coined this the "lack-of-use vicious circle": *"When you really need a debugger, you're not willing to learn a new tool. When you're willing to learn a new tool, you don't really want to learn a debugger."*

Wadler records the story [18] of the Standard ML debugger [13] that was deeply intertwined with the compiler and runtime of the SML/NJ Standard ML compiler. As the SML/NJ implementation evolved, the debugger fell out of step, and is no longer available.

The implementations described above all suffer, to a lesser or greater extent, from a lack of what Marlow [6] calls *accessibility*. They provide only for a subset of the language, or require changes to be made to build environments, or do not scale well. So there is often one or more fundamental impediments to their use – they are not *accessible*. A debugger must be as accessible as a compiler. Marlow claims that the most complete Haskell trace debugger, Hat [19], remains largely unused due to a lack of such accessibility – for example, it must be modified to support new third-party libraries. We intend, then, to bake in the correct design decisions to support widespread applicability (and thus adoption) from the beginning, even if it is at the expense of other desirable characteristics (such as speed). We aim for our system to (a) be able to support the whole language by design; (b) be suitable for any build environment where OCaml programs can already be built; and (c) be abstracted from the compiler, and thus be robust to advances in the language and runtime environment. Thus, instead of imagining the perfect visualization, writing a toy system, and worrying about how to extend it to a practical one later, we will make design decisions based on the practicalities,

and work backward from our goal. Even if our system is initially a toy in the sense that it does not support the full language, it is not a toy in terms of its integration with the language and runtime, and so extending it to the full language should be technologically straightforward (though a sizeable piece of work). Our litmus tests are these: (1) Can our system be used to debug any OCaml program where source is available, even if uses external libraries? (2) Can our system support development of a complex system such as the OCaml compiler itself? (3) Most importantly, of course, do people actually choose to use it?

We shall, therefore, take an extremist approach: we shall worry about Marlow's *accessibility* foremost, and everything else second. Wadler [18] writes: *"... there are few debuggers or profilers for strict [functional] languages, perhaps because constructing them is not considered research. This is a shame, since such tools are sorely needed, and there remains much of interest to learn about their construction and use."* We aim to right this wrong.

## 4 Simple Visualizations

As is traditional, we consider a program for calculating the factorial of a positive number, with `4` as our input:

```
let rec factorial n =
  if n = 1 then 1 else n * factorial (n - 1)
in
  factorial 4
```

The upper portion of Figure 1 shows a naive evaluation of this program. This is certainly not how we would write such an evaluation on paper. Although the evaluation shown is self-contained in the sense that each line of it is a valid program, which might seem a useful property, it is hard to see what is going on. It is large, both in width (how long the expression becomes) and length (how many lines are needed). Writing each evaluation step over multiple lines as we did with the original program above would not only increase the length, but make it difficult to visually compare adjacent lines. We must reduce the amount of information shown, even in this simple case.

Look now at the lower part of Figure 1, which shows the output of our prototype system. The following differences are apparent: (a) We have removed the definition of the factorial function itself. Since it is recursive, its name will appear in the expression anyway; (b) We have avoided printing any reduction step which leads to an expression such as `if false` or `if true`; (c) We have not shown the intermediate steps of simple arithmetic which reduce `4 * (3 * (2 * 1))` to `24`; (d) We have removed trivial arithmetic (like subtracting one), even when it involves variable names, such as reducing `n - 1` to `3` directly rather than via `4 - 1`; (e) We have removed `let` bindings which apply to the whole expression to the left hand side of the `=>` arrow to avoid too many `let n = ...` instances making the expression too wide; (f) We have used some simple syntax colouring in the form of bold for keywords; and (g) We have underlined the expression to be reduced at each step. All these changes have been made automatically. Each step is no longer a valid OCaml program, but the increase in readability is significant. Clearly, for larger programs, such elision will be even more important, since the focus needs to be on the currently-evaluating subexpression of a potentially huge expression representing the whole program. Note that all the intervening steps of the computation are performed, but certain lines are not printed. This means that the finer details of the computation may be inspected upon demand.

In the program trace we have already exhibited, it is clear that for realistic programs, the program trace (both its width and its length) may be significant. This issue is discussed

in some detail by Taylor [12] and Pajera-Flores [7]. A practical solution, we claim, must involve providing ways of (a) eliding information within a single step (reducing the width); (b) eliding whole steps (reducing the length); (c) searching the resultant trace, if it is still too large to spot the bug; and (d) moving backward and forward through the trace to connect cause and effect in the computation.

If we wish to be able to reduce traces on command to find a bug in a morass of data, we shall need a concise and powerful way for the programmer to describe the elisions and searches required. That is to say, we need to have a way for the programmer to translate *"I'm sure this bug has something to do with the* `tree` *data type, and I know it must happen after the tree has been populated, but before the last element is removed. I know the proximate cause is a* `Not_found` *exception being raised."* into something the computer can understand, and which results in a reduced, concise, useful trace.

## 5 Other Kinds of Computation

Although the primary challenge is one of scale, we must consider also non-functional computation (such as the use of input/output or mutable data). There follows a brief discussion of several of these, to give a flavour of the complications involved. There are plenty of others, of course. For example, we have yet to explore the visualization of concurrent or parallel execution.

### 5.1 Exceptions

Though exceptions can be explained using the same term-rewriting rules as any other functional construct, it is likely that some special treatment will be necessary, especially for the visualization of larger programs. Some of the complications of exception visualization for imperative languages are described by Shah [10], many of which will apply in our case too. Exceptions are important, of course, because they are used for dealing with illegal states, a common cause of problems which we end up debugging. Additionally, and somewhat unusually, exceptions are frequently used in OCaml programs not just for genuinely exceptional situations, but for local control flow. So it is especially important that the debugger's approach to exceptions is lightweight.

### 5.2 Input/Output and System Primitives

Consider the following program, which reads a line from standard input, and then prints it on standard output:

```
  print_string (input_line stdin)
```

Do we separate the output of the program from the output of the debugger, or is it better to show everything interleaved? In addition, how do we deal with `print_int` and `input_line`, which are members of the most basic parts of the Standard Library, themselves defined in terms of system primitives? In the present prototype, the output of the program and the debugger are interleaved. Here is a session, with the Standard Library and primitive operations elided, which is the default:

```
  print_string (input_line <in_channel>)
SLATE
SLATE=>  ()
```

```
    let rec factorial n = if n = 1 then 1 else n * factorial (n - 1) in factorial 4
=>  let rec factorial n = if n = 1 then 1 else n * factorial (n - 1) in let n = 4 in if n = 1 then 1 else n * factorial (n - 1)
=>  let rec factorial n = if n = 1 then 1 else n * factorial (n - 1) in let n = 4 in if false then 1 else n * factorial (n - 1)
=>  let rec factorial n = if n = 1 then 1 else n * factorial (n - 1) in let n = 4 in n * factorial (n - 1)
=>  let rec factorial n = if n = 1 then 1 else n * factorial (n - 1) in let n = 4 in 4 * factorial (n - 1)
=>  let rec factorial n = if n = 1 then 1 else n * factorial (n - 1) in 4 * factorial (4 - 1)
=>  let rec factorial n = if n = 1 then 1 else n * factorial (n - 1) in 4 * factorial 3
=>  let rec factorial n = if n = 1 then 1 else n * factorial (n - 1) in 4 * (let n = 3 in if n = 1 then 1 else n * factorial (n - 1))
=>  let rec factorial n = if n = 1 then 1 else n * factorial (n - 1) in 4 * (let n = 3 in if false then 1 else n * factorial (n - 1))
=>  let rec factorial n = if n = 1 then 1 else n * factorial (n - 1) in 4 * (let n = 3 in n * factorial (n - 1))
=>  let rec factorial n = if n = 1 then 1 else n * factorial (n - 1) in 4 * (let n = 3 in 3 * factorial (n - 1))
=>  let rec factorial n = if n = 1 then 1 else n * factorial (n - 1) in 4 * (3 * factorial (3 - 1))
=>  let rec factorial n = if n = 1 then 1 else n * factorial (n - 1) in 4 * (3 * factorial 2)
=>  let rec factorial n = if n = 1 then 1 else n * factorial (n - 1) in 4 * (3 * (let n = 2 in if n = 1 then 1 else n * factorial (n - 1)))
=>  let rec factorial n = if n = 1 then 1 else n * factorial (n - 1) in 4 * (3 * (let n = 2 in if false then 1 else n * factorial (n - 1)))
=>  let rec factorial n = if n = 1 then 1 else n * factorial (n - 1) in 4 * (3 * (let n = 2 in n * factorial (n - 1)))
=>  let rec factorial n = if n = 1 then 1 else n * factorial (n - 1) in 4 * (3 * (let n = 2 in 2 * factorial (n - 1)))
=>  let rec factorial n = if n = 1 then 1 else n * factorial (n - 1) in 4 * (3 * (2 * factorial (2 - 1)))
=>  let rec factorial n = if n = 1 then 1 else n * factorial (n - 1) in 4 * (3 * (2 * factorial 1))
=>  let rec factorial n = if n = 1 then 1 else n * factorial (n - 1) in 4 * (3 * (2 * (let n = 1 in if n = 1 then 1 else n * factorial (n - 1))))
=>  let rec factorial n = if n = 1 then 1 else n * factorial (n - 1) in 4 * (3 * (2 * (let n = 1 in if true then 1 else n * factorial (n - 1))))
=>  4 * (3 * (2 * 1))
=>  4 * (3 * 2)
=>  4 * 6
=>  24

    factorial 4
n = 4  =>  if n = 1 then 1 else n * factorial (n - 1)
n = 4  =>  n * factorial (n - 1)
       =>  4 * factorial 3
n = 3  =>  4 * (if n = 1 then 1 else n * factorial (n - 1))
n = 3  =>  4 * (n * factorial (n - 1))
       =>  4 * (3 * factorial 2)
n = 2  =>  4 * (3 * (if n = 1 then 1 else n * factorial (n - 1)))
n = 2  =>  4 * (3 * (n * factorial (n - 1)))
       =>  4 * (3 * (2 * factorial 1))
n = 1  =>  4 * (3 * (2 * (if n = 1 then 1 else n * factorial (n - 1))))
       =>  4 * (3 * (2 * 1))
       =>* 24
```

**Figure 1** A naive rendering of the evaluation of `factorial 4` showing each step of the evaluation, followed by an automatically trimmed one, eliding (a) parts of the evaluation of the `if` construct; (b) the definition of a recursive function mentioned in the expression; (c) the final portion of arithmetic and (d) trivial operations such as `3 - 1`. In addition, `let` expressions unique in the whole expression are written on the left, and simple syntax highlighting has been used. The expression to be reduced in each step is underlined.

(We typed the word “SLATE”). If the user really wants the gory details of the inside of the Standard Library, they can be shown instead:

```
    print_string (input_line <in_channel>)
=> print_string (let x = <in_channel> in <<input_line>>)
=> print_string <<input_line>>
SLATE
=> print_string "SLATE"
=> let x = "SLATE" in output_string <out_channel> x
=> let x = "SLATE" in (let x = <out_channel> in fun y -> <<output_string>>) x
=> let x = "SLATE" in (fun y -> let x = <out_channel> in <<output_string>>) x
=> (fun y -> let x = <out_channel> in <<output_string>>) "SLATE"
=> let y = "SLATE" in let x = <out_channel> in <<output_string>>
=> let y = "SLATE" in <<output_string>>
=> <<output_string>>
SLATE=>  ()
```

Abstract data types for standard input and output channels have been written `<thus>`. The invocation of a genuine system primitive (rather than just a standard library function) is written with double angle brackets around it `«thus»`. You can see why it is usually sensible to elide such information. We are not typically debugging libraries, but our own program. Until we are sure a library is at fault, we do not want to delve into its internals. This is just one example of the importance of mechanisms of elision in trace debugging.

## 5.3 Mutability

The OCaml language is what has been termed a “functional-first” language. That is to say, whilst we may tend to write in a pure functional style, we may also use mutable cells to hold changing values. Thus, it is necessary either to ensure that the contents of a cell is always displayed in the printed evaluation step, or that there is another way for the user to see it or request to see it. In the latter case, which is probably preferable, we may simply use techniques from the broad range available in traditional debuggers for imperative languages.

# 6 Implementation Notes

The present prototype provides visualization for a subset of the OCaml language, together with a number of methods for elision of information to produce more reasonable traces. Our interpreter is, thus far, only a thousand lines of code, due to the ease with which we can use parts of the OCaml compiler: in recent versions of OCaml the compiler is built not only in executable form, but in library form as compiler-libs. This means that one may write a program which uses types and functions from the OCaml compiler, for example the Abstract Syntax Tree type. Our `ocamli` interpreter is a program of this form: it uses the standard lexer, parser and type-checker direct from compiler-libs. And so, writing an interpreter for OCaml programs has required almost no duplication or modification of compiler code. This has reduced vastly the cognitive load of such an endeavour, and increased the likelihood that the interpreter will, with little modification, continue to build and function well into the future. Before compiler-libs, we would have had to copy reams of code from the OCaml compiler source code, or provide our interpreter as a patch to the OCaml compiler itself. Both are inadvisable from the perspective of Marlow’s accessibility – that a debugger should be available in a low-friction manner to users, for any project, at any time.

The prototype implementation is simplistic, using a number of tree-processing passes to perform the evaluation and elisions, so does not guarantee that the time or space complexity of interpreting a program is the same as the time or space complexity of compiling and running that program. We hope to discover in future to what extent such a guarantee

is possible (at least modulo the pretty-printing – clearly, printing out all the stages of a computation cannot help but increase the time complexity of interpreting it).

## 7 Conclusion

We have advanced an ambitious but, in our opinion, practical scheme for the implementation of an interpretive debugger for a popular functional language, and described aspects of the current prototype. There is much to be done. Will it fall by the wayside like so many other debuggers? Or have we found the right formula?

## References


1 Mikhail Auguston and Juris Reinfields. A visual Miranda machine. In *Software Education Conference*, pages 198–203, November 1994.

2 Robert Bruce Finder, John Clements, Cormac Flanagan, Matthew Flatt, Shriram Krishnamurthi, Paull Steckler, and Matthias Felleisen. DrScheme: A programming environment for Scheme. *Journal of Functional Programming*, 12(2):159–182, March 2002.

3 Andy Gill. Debugging Haskell by observing intermediate data structures. Technical report, University of Nottingham, 2000.

4 Doug Goldson. A symbolic calculator for non-strict functional programs. *The Computer Journal*, 37(3):177–187, 1993.

5 Henry Lieberman. The debugging scandal and what to do about it. *Communications of the ACM*, 40(4), April 1997.

6 Simon Marlow, José Iborra, Bernard Pope, and Andy Gill. A lightweight interactive debugger for Haskell. In *ACM SIGPLAN Workshop on Haskell Workshop*, pages 13–24. ACM, 2007.

7 Cristóbal Pareja-Flores, Jaime Urquiza-Fuentes, and J. Ángel Velázquez-Iturbide. WinHIPE: An IDE for functional programming based on rewriting and visualization. *ACM SIGPLAN Notices*, 42(3):14–23, March 2007.

8 Marian Petre and Ed de Quincey. A gentle overview of software visualization. *PPIG Newsletter*, pages 1–10, September 2006.

9 Steve Reeves, Doug Goldson, Pat Fung, Mike Hopkins, and Richard Bornat. The Calculator project – formal reasoning about programs. In *Software Education Conference*, pages 166–173, November 1994.

10 Hina Shah, Carsten Görg, and Mary Jean Harrold. Visualization of exception handling constructs to support program understanding. In *4th ACM Symposium on Software Visualization*, pages 19–28, 2008.

11 Juha Sorva, Ville Karavirta, and Lauri Malmi. A review of generic program visualization systems for introductory programming education. *Transactions on Computing Education*, 13(4):15:1–15:64, November 2013.

12 Jonathan Paul Taylor. *Presenting the Lazy Evaluation of Functions*. PhD thesis, Queen Mary, University of London, 1996.

13 Andrew Tolmach and Andrew W. Appel. A debugger for Standard ML. *Journal of Functional Programming*, 5(2):155–200, April 1995.

14 David S. Touretzky. Visualizing evaluation in applicative languages. *Communications of the ACM*, 35(10):49–59, October 1989.

15 David Ungar, Henry Lieberman, and Christopher Fry. Debugging and the experience of immediacy. *Communications of the ACM*, 40(4):38–43, April 1997.

16 Jaime Urquiza-Fuentes and J. Ángel Velázquez-Iturbide. A survey of program visualizations for the functional paradigm. In *3rd Program Visualization Workshop*, pages 2–9, 2004.

**17** Jaime Urquiza-Fuentes and J. Ángel Velázquez-Iturbide. A survey of successful evaluations of program visualisation and algorithm animation systems. *Transactions on Computing Education*, 9(2):9:1–9:21, June 2009.

**18** Philip Wadler. Why no one uses functional languages. *The SIGPLAN Notices*, 33(8):23–27, August 1998. doi:10.1145/286385.286387.

**19** Malcolm Wallace, Olaf Chitil, Thorsten Brehm, and Colin Runciman. Multiple-view tracing for Haskell: a new Hat. In *ACM SIGPLAN Haskell Workshop*, pages 151–170, 2001.

# A Survey on CSS Preprocessors


## Ricardo Queirós

**ESMAD, Department of Informatics, Polytechnic of Porto, Porto, Portugal**
ricardoqueiros@esmad.ipp.pt



## Abstract

In the Web realm, the adoption of Cascading Style Sheets (CSS) is unanimous, being widely used for styling web documents. Despite their intensive use, this W3C specification was written for web designers with limit programming background. Thus, it lack several programming constructs, such as variables, conditional and repetitive blocks, and functions. This absence affects negatively code reuse, and consequently, the maintenance of the styling code. In the last decade, several languages (e.g. Sass, Less) appeared to extend CSS, defined as CSS preprocessors, with the ultimate goal to bring those missing constructs and to foster stylesheets structured programming. The paper provides an introductory survey on CSS Preprocessors. It gathers information on a specific set of preprocessors, categorizes them and compares their features regarding a set of predefined criteria such as: maturity, coverage and performance.



**1998 ACM Subject Classification** D.3.4 Processors / Preprocessors

**Keywords and phrases** CSS, Preprocessors, Web formatting

**Digital Object Identifier** 10.4230/OASIcs.SLATE.2017.8


## 1 Introduction

Nowadays, when it comes to Web frontend development, we need to understand three languages with different purposes: HyperText Markup Language (HTML) for structuring content, Cascading Style Sheets (CSS) to style it and JavaScript to attach some behaviour. Regarding CSS, several surveys show that this W3C specification is used in more than 95% of the websites [2], revealing its enormous importance on the process of constructing a website. A CSS file organizes a set of styling rules in selectors which has the responsibility to select the elements of the target documents that should be styled. Although its power, it lacks many programming constructs such as variables, conditional blocks and functions. This absence leads CSS developers to copying style declarations from one selector to another (e.g. code cloning) fostering code duplication and harming code maintenance. In a previous work [3], the CSS code of 38 high traffic websites were analysed and it was found that, on average, more than 60% of the CSS declarations were duplicated across at least two selectors.

In order to address these kind of issues, CSS preprocessor languages were introduced in the last decade. The code written in a CSS preprocessor can include programming constructs and, thereafter, be compiled to pure standardise CSS. Currently, there are several CSS preprocessors (e.g. Sass[1], Less [2], Stylus [3], PostCSS [4]), and their use is becoming a fast growing trend in the industry. In 2012, an online survey [1] with more than 13,000 responses from Web developers, conducted by a famous website focusing on CSS development, showed

[1] Available at http://sass-lang.
[2] Available at http://lesscss.org/.
[3] Available at http://stylus-lang.com/.
[4] Available at http://postcss.org/.


that 54% of the respondents use a CSS preprocessor in their development tasks. Despite these promising numbers, there are many developers that understands the power of CSS preprocessors but do not know which preprocessor should use.

In this paper we make a simple survey on CSS preprocessors. For this study, we start by selecting active preprocessors with a reasonable amount of popularity and documentation. Then, we describe the selected preprocessors and we compare them regarding a set of predefined criteria such as: maturity, coverage and performance. For the first criteria we based our study on the preprocessors activity in the Git repository hosting service, called GitHub, where the number of commits, releases, contributors and forks are enumerated and analysed. For the coverage criteria, we rely on a set of features (e.g variables, mixins, conditional, loops) that developers expect to have in such type of tools and check if these features are supported in the selected preprocessors. Finally, in the performance criteria, we conducted a simple experiment to test the performance of the selected preprocessors measuring their compilation time for a specific set of styling rules.

The remainder of this paper is organized as follows: Section 2 enumerates the typical programming constructs in a preprocessor. In the following section we initiate the survey based on three criteria: maturity, coverage and performance. Finally, we conclude with a summary of the main contributions of this work and a perspective of future research.

## 2 Preprocessors

As previously stated, building a function, or inheritance are hard to achieve using solely CSS. When a webpage becomes more complex, we often see CSS files with a lot of rules and a reasonable level of redundancy. One way to save time, and to keep all these rules in a more flexible way, is through the use of CSS preprocessors. These tools make it easy to maintain large, complex style sheets, thanks to the use of features hitherto unavailable in the context of creating style sheets, such as variables, functions, and mixins. Using these constructs, code becomes more organized, allowing developers to work faster and make fewer mistakes. In this section we analysed several features typically included in CSS preprocessors. We demonstrate each feature through code examples with both preprocessor and CSS generation code. For the preprocessor language we use Sass syntax.

### 2.1 Variables

One of the main features of CSS preprocessors is the support for variables. This is a huge feature, since CSS developers lack the chance, for instance, to define a base color as a variable and reuse it all over on the CSS file. Instead they need hard-coded the hex or named color (or other property such as width, font-size, and others) as a property value for each rule. Using CSS preprocessors, in the case you need to change the color, you only have to do it in one place, in the variable inicialization. Using the traditional approach, you must change manually all occurrences of the color. In big and complex web apps this could be cumbersome and error-prone.

Variables work in a similar fashion to the those in any programming language, including the concepts for data types and scope. Also like every programming language, preprocessors have different syntax to represent variables' declaration and initialization. Despite their differences, their syntax are like classic CSS, so the learning curve for CSS developers is small. For instance, variables in Sass start with $ sign, in LESS with the @ sign and no prefix in Stylus. Both in Sass and Less, values are assigned with colon (:), and with equals sign (=) in Stylus. In the first example, we declare and initialise a variable and use it as a

property value for two rules. The generated CSS will override all the variable occurrences with its value.

**Listing 1** Sass code.

```
$primaryColor: #eeccff;

body {
  background: $primaryColor;
}
p {
  color: $primaryColor;
}
```

**Listing 2** Generated CSS code.

```
body {
  background: #eeccff;
}
p {
  color: #eeccff;
}
```

Beyond using variables in property values, it is also possible to use variables in selectors or property names through interpolation. The next example shows how to use variables interpolation to define a name for a selector and a property:

**Listing 3** Sass code.

```
$name: foo;
$attr: border;
p.#{$name} {
  #{$attr}-color: blue;
}
```

**Listing 4** Generated CSS code.

```
p.foo {
  border-color: blue;
}
```

## 2.2 Nesting

As opposed to HTML, CSS nesting structure it's hard to write and to understand. Using preprocessors, we can combine various CSS rules by creating composite selectors. Basic nesting refers to the ability to have a declaration within another declaration. With the preprocessors nesting syntax, developers can organise stylesheets in a way that resembles HTML more closely, thus reducing the chance of CSS conflicts.

A classic example is when we have to style an HTML list composed by a set of links. In this case we have a structure with several depth levels (<ul>, <li>, <a>) and that requires some care when styling it through CSS rules. Using preprocessors, we have a more intuitive and compact way of styling these types of scenarios through nested declarations. The following example illustrates this situation:

**Listing 5** Sass code.

```
ul {
  list-style: none;
  li {
    padding: 15px;
    display: inline-block;
    a {
      text-decoration: none;
      font-size: 16px;
      color: #444;
    }
  }
}
```

**Listing 6** Generated CSS code.

```
ul {
  list-style: none;
}
ul li {
  padding: 15px;
  display: inline-block;
}
ul li a {
  text-decoration: none;
  font-size: 16px;
  color: #444;
}
```

In the case we want to reference the parent element in nested declarations, the & symbol should be used. The next example shows how to add pseudo-selectors using this technique:

**Listing 7** Sass code.

```
a.myAnchor {
 color: blue;
 &:hover {
  text-decoration: underline;
 }
 &:visited {
  color: purple;
 }
}
```

**Listing 8** Generated CSS code.

```
a.myAnchor {
 color: blue;
}
a.myAnchor:hover {
 text-decoration: underline;
}
a.myAnchor:visited {
 color: purple;
}
```

## 2.3 Extend

Inheritance is one of the key concepts in the object oriented programming paradigm. In OOPs, the concept of inheritance provides the idea of reusability. This means that we can add new features to an existing class without modifying it. This is possible by deriving a new class from the existing one. In the CSS realm, inheritance should be used when we need similar styled elements, but requiring slight changes between them. In Sass, we use the keyword @*extend* following by the base rule name we want to inherit. A classic example is the definition of two buttons: one for confirmation and one for cancellation.

**Listing 9** Sass code.

```
.dialog-btn {
  box-sizing: border-box;
  color: #ffffff;
  box-shadow: 0 1px 1px 0
   rgba(0, 0, 0, 0.12);
}
.confirm {
  @extend .dialog-btn;
  background-color: #87bae1;
  float: left;
}
.cancel {
  @extend .dialog-btn;
  background-color: #e4749e;
  float: right;
}
```

**Listing 10** Generated CSS code.

```
.dialog-btn, .confirm, .cancel {
  box-sizing: border-box;
  color: #ffffff;
  box-shadow: 0 1px 1px 0
   rgba(0, 0, 0, 0.12);
}
.confirm {
  background-color: #87bae1;
  float: left;
}
.cancel {
  background-color: #e4749e;
  float: right;
}
```

In the previous example, we note the CSS generated by Sass combined the selectors instead of repeating the same statements systematically, saving us precious memory.

In certain situations, you need to define styles that you just want to be extended and never used directly. A good example is when writing a library (such as Sass), where you just want to provide styles for users to extend in case they need to or ignore otherwise. For this type of scenario, Sass supports placeholder selectors (or placeholders). This type of selectors resemble the class and id selectors (# or.), But in this case the % symbol is used. The selectors can be used anywhere and will prevent rule sets from being rendered to CSS, except by extension.

**Listing 11** Sass code.

```
%input style {
  font-size: 14px;
}
input {
  @extend %input style
  color: black;
}
```

**Listing 12** Generated CSS code.

```
input {
  font-size: 14px;
}
input {
  color: black;
}
```

In this example, the %input-style rule will not be generated for CSS. However if we extend this rule, we will see that it is already possible to obtain the rule in the generated CSS.

## 2.4 Mixins

Mixins can be seen as a simplified version of constructors in object-oriented programming languages. In fact, it may include styles in the same way that @extend does but as the ability to provide arguments, thus making it a valuable tool for dynamic code reuse and redundancy reduction. The @*mixin* directive is used to create mixins and the @*include* directive is used to invoke them. In the next example, a mixin is defined to render squares with colors and sizes passed as arguments to the mixin:

**Listing 13** Sass code.

```
@mixin square($size, $color) {
  width: $size;
  height: $size;
  background-color: $color;
}
.small-blue-square {
  @include square(20px,
   rgb(0,0,255));
}
.big-red-square {
  @include square(300px,
   rgb(255,0,0));
}
```

**Listing 14** Generated CSS code.

```
.small-blue-square {
  width: 20px;
  height: 20px;
  background-color: blue;
}
.big-red-square {
  width: 300px;
  height: 300px;
  background-color: red;
}
```

Another typical example is when a property requires prefixes to work in all browsers, such as the transform property:

**Listing 15** Sass code.

```
@mixin transform-tilt($tilt) {
  -webkit-transform: $tilt;
  ms-transform: $tilt;
  transform: $tilt;
}
table:hover {
  $tilt: rotate(15deg);
  @include transform-tilt($tilt);
}
```

**Listing 16** Generated CSS code.

```
table:hover {
  -webkit-transform:
    rotate(15deg);
  -ms-transform: rotate(15deg);
  transform: rotate(15deg);
}
```

## 2.5 Functions

Typically, preprocessors support a long list of predefined functions. They serve all sorts of purposes, including string manipulation, color-related operations, and some useful math methods such as random and round. The next example uses the darken function that dims a particular color by a certain percentage:

**Listing 17** Sass code.

```
$my-blue: #2196F3;

a {
  padding: 10px 15px;
  background-color: $my-blue;
}
a:hover {
  background-color:
    darken($my-blue,10%);
}
```

**Listing 18** Generated CSS code.

```
a {
  padding: 10px 15px;
  background-color: #2196F3;
}
a:hover {
  background-color: #0c7cd5;
}
```

You can also define your own functions and invoke them anywhere through role directives. In Sass, the role directives are similar to the mixins, but instead of returning a set of properties, they return values through the @*return* directive. The definition of a function is done with the @*function* directive. The next example shows how to define a function for calculating the width of a column based on the width of its parent, the number of columns, and the margins size:

**Listing 19** Sass code.

```
$ctn-width: 100%;
$col-count: 4;
$mgin: 1%;
@function getColW($w, $cols, $mgin){
  @return ($w / $cols) - ($mgin * 2);
}
.container {
  width: $ctn-width;
}
.column {
  width: getColW($ctn-width,$col-count,$mgin);
  height: 200px;
}
```

**Listing 20** CSS code.

```
.container {
  width: 100%;
}
.column {
  width: 23%;
  height: 200px;
}
```

## 2.6 Conditional and cyclic structures

Some preprocessors support decision and cyclic structures. You can actually use directives like @*if*, @*for*, @*each*, and @*while*. The next two examples show how to use these directives in two different contexts: decision making at a given value and iteration under variables to create similar statements. In the first example, the value of a variable is inspected and a set of actions is executed by its value. Note that if the condition in the @*if* directive evaluates to false, the set of expressions that follow the @*else* directive is executed.

**Listing 21** Sass code.

```
$boolean: true !default;
@mixin foo() {
  @debug "$boolean is #{$boolean}";
  @if $boolean {
    display: block;
  }
  @else {
    display: none;
  }
}
.some-selector {
  @include foo();
}
```

**Listing 22** CSS code.

```
.some-selector {
  display: block;
}
```

The following example uses the @*for* directive to define multiple style declarations named by interpolation:

**Listing 23** Sass code.

```
$squareC: 3;

@for $i from 1 through $squareC {
  #square-#{$i} {
    background-color: red;
    width: 50px * $i;
    height: 120px / $i;
  }
}
```

**Listing 24** CSS code.

```
#square-1 {
  background-color: red;
  width: 50px;
  height: 120px;
}
#square-2 {
  background-color: red;
  width: 100px;
  height: 60px;
}
#square-3 {
  background-color: red;
  width: 150px;
  height: 40px;
}
```

## 3 CSS preprocessors survey

Currently, there are several CSS preprocessors offering similar features with a different syntax. In order to select one, users typically start by reading some sites where a subset of them is described and compared based on a adhoc list of advantages/disadvantages. This approach could be dangerous since you could invest time in one preprocessor and discover later that is not well document or do not have an active developers community, or do not support a specific language binding or feature, or even, has poor performance while running in a production environment. This work aims to offer a consistent study to base a sustained choice of a CSS preprocessor. In order to accomplish this challenge, we start by selecting a specific set of preprocessores based on a very popular 2016 frontend tooling survey [4] where 4,715 developers (mostly people with 5 to 10 years of frontend technologies experience) answered questions about Web tools and standards. Analysing deeply the survey, we can find the responses to the question “What is your CSS Processing tool of choice?”. The results were conclusive as shown in Table 1.

**Table 1** CSS Processing tool.

| Preprocessor | # Votes | Percentage | % Diff (to 2015) |
|---|---|---|---|
| Sass | 2,989 | 63.39% | -0.56% |
| Less | 478 | 10.14% | -5.05% |
| Stylus | 137 | 2.91% | -0.84% |
| PostCSS | 392 | 8.31% | N/A |
| No Preprocessor | 643 | 13.64% | -1.4% |
| Other | 73 | 1.55% | -0.52% |

Analysing the results, Sass is still the CSS processing tool of choice for the majority of respondents with 63.39%. When compared to last years results, Less usage has dropped from 15.19% to 10.14% (down 5.05%).

Also we can state that the percentage of respondents using CSS processing tools grows to 86.36% (more 1.4% from 2015). This reenforces the importance of CSS processing tools in the Web development workflows.

We have included tools that are not pure preprocessors, such as PostCSS that is coined as a CSS postprocessor tool. These type of tools are becoming essential for frontend developers since they can apply specific actions after the CSS has been generated. Among the supported actions, the most popular are applying vendor prefixes automatically, creating pixel and IE8 media query fallbacks. The most notable example of this kind of tools is PostCSS. In this survey one can notice that PostCSS is used by 8.31% of respondents. Actually, it's usage is likely to be slightly higher, as this survey does not count for those respondents who use it in combination with another processing tool.

Based on this survey, we select Sass, Less and Stylus for the CSS Processors survey. In the next subsections the three preprocessors are described and compared based on three facets: maturity, coverage and performance.

## 3.1 Maturity

It is difficult to determine which preprocessor is most widely used or have more impact. In this study we made an effort to measure the popularity/impact of the CSS preprocessors in the Web. Popularity, is usually influenced by an active community. When choosing a specific language, library or framework for your project, consider first looking at the community behind and check if it is actively developed for bug fixes and new features. A large and active community will also make it easier to get support and will provide more learning advices and resources. Beyond that, verify how detailed, well-written and organized the documentation is.

Various methods of measuring tools popularity have been proposed, each subject to a different bias over what is measured. In this context, we will focus on two: counting the number of times the language name is mentioned in web searches (using Google Trends) and comparing the activity in GitHub where all the selected preprocessors are stored.

Firstly, we search in Google Trends for the three CSS preprocessors. The results are shown in Figure 1.

It appears there is much more activity with Sass and Less, rather than Stylus. From 2015, Sass is the CSS preprocessor most typed in Google searches.

The second method chosen to measure the impact/popularity of the selected preprocessors was the analysis of their Github activity. In fact, the three preprocessors chosen are pretty active on Github, as you can see in Table 2.

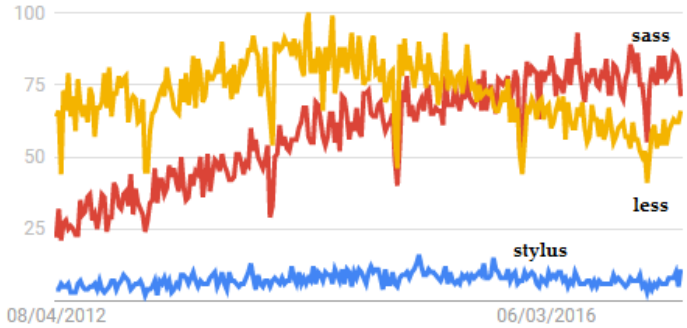



**Figure 1** Interest over time on Sass, Less and Stylus preprocessors – Google Trends.

**Table 2** CSS Processors Maturity.

| GitHub data | Sass | Less | Stylus |
|---|---|---|---|
| First release date | 0.1.0 (Oct/2006) | 1.0 (Apr/2010) | 0.02 (Jan/2011) |
| Last release date | 3.4.23 (Dec/2016) | 2.7.2 (Jan/2017) | 0.54.5 (Apr/2016) |
| # Open issues | 182 | 241 | 159 |
| # Pull requests | 7 | 28 | 17 |
| # Commits | 6241 | 2663 | 3881 |
| # Releases | 181 | 49 | 161 |
| # Contributors | 180 | 208 | 147 |
| # Stars | 9557 | 14590 | 7973 |
| # Forks | 1763 | 3353 | 949 |

Although relatively recent, these initiatives have been growing with the evolution of the HTML and CSS Web standards. Sass is the oldest and the one with greater update frequency of commits and releases. However, Less, despite being younger, is the most popular, with a larger number of stars and forks. The number of forks is relevant. A fork is a copy of a repository. Forking a repository allows you to freely experiment a repo (with changes) without affecting the original project. Thus, this means that most people are using Less base code to start their own projects. Regarding Stylus, it presents the lower values of the three in almost all the indicators.

## 3.2 Coverage

In the coverage criteria we will make a comparison of the three CSS preprocessors regarding the support for the most popular features, such as, variables, mixins, conditionals and loops.

In Table 3 we present the comparison on CSS preprocessors variable features.

All the preprocessores have the basic ability to declare variables and use them later. However, Less do not support the feature of default variables, that is, variables that are overwritten by regular ones, no matter when they are declared. Nevertheless, variable hosting (lazy) is only supported by Less, where variables can be declared after being used. The Stylus preprocessor is the only one that allows you to use the value of a previously declared property elsewhere (variables lookup). Finally, all the preprocessors support interpolation. This means that we can use variables as parts of selectors, properties, values, as parameters of the calc function – a CSS function that performs calculations to determine CSS property values – and even, place a set of selectors into a variable and reuse it.

**Table 3** Variables features comparison.

| Features | Sass | Less | Stylus |
|---|---|---|---|
| Basic | Yes | Yes | Yes |
| Default | Yes | No | Yes |
| Lazy | No | Yes | No |
| Lookup | No | No | Yes |
| Interpolation | Yes | Yes | Yes |

**Table 4** Mixins features comparison.

| Features | Sass | Less | Stylus |
|---|---|---|---|
| Basic | Yes | Yes | Yes |
| Params | Yes | Yes | Yes |
| Params-named | Yes | Yes | Yes |
| Arguments | Yes | Yes | Yes |

**Listing 25** Less loop example.

```
.generate-column(@i: 1) when (@i =< 4) {
  .col-@{i} {
    width: @i * (100% / 4);
  }
  .generate-column(@i + 1);
}
```

Mixins is another powerful feature of CSS preprocessors. The CSS preprocessors mixins support is shown in Table 4.

All the CSS preprocessors support the mixins main features, namely, the inclusion of mixins (basic), mixins that can receive parameters passed to them (params), mixins that have named placeholders for each parameter passed to them (params-named) and, finally, mixins with an unknown number of arguments passed to them (arguments).

Other important feature of CSS preprocessors is the support for conditionals. Conditionals are useful when we want that a part of our code to be executed only certain condition is evaluated as true (or false). Although Less uses a slightly unconventional syntax (*), all the preprocessors support if statements within a declaration. Ternary operators allow for a single-line if/else test in the format of $x > 0?true : false$. Less do not support this feature and Sass uses an unusual syntax. Regarding, the capacity to interpolate "if statements" inside property names, only Sass and Stylus support it. Table 5 summarises all the conditional supported features.

Regarding loops (Table 6), we only compared loops, that increment values (basic), and loops that iterate though items in a list (intermediate). All preprocessors support both, although, Less needs to call the function as shown in Listing 25. Thus, we can conclude that Sass and Stylus are the preprocessores with greater support for the most popular features. Less has support for conditional and loops, but relies mostly on unusual syntax to achieve it.

**Table 5** Conditional features comparison.

| Features | Sass | Less | Stylus |
|---|---|---|---|
| Basic | Yes | Yes * | Yes |
| Ternary | Yes * | No | Yes |
| Property | Yes | No | Yes |

**Table 6** Loop features comparison.

| Features | Sass | Less | Stylus |
|---|---|---|---|
| Basic | Yes | Yes * | Yes |
| Intermediate | Yes | Yes * | Yes |

**Listing 26** Benchmark script excerpt.

```
var path = require('path');
var fs   = require('fs');
var n = 999;

// Sass
var libsass = require('node-sass');

var scss = '$size: 100px;';
scss += '@mixin icon { width: 16px; height: 16px; }';
for ( i = 0; i < n; i++ ) {
    scss += 'body { h1 { a { color: black; } } }';
    scss += 'h2 { width: $size; }';
    scss += 'h1 { width: 2 * $size; }';
    scss += '.search { fill: black; @include icon; }';
}

var result =  libsass.renderSync({ data: scss });
console.log("Sass: ", result.stats.duration + "\t ms");

// Repeat for the other two preprocessors
...
```

## 3.3 Performance

In this subsection the three preprocessors are compared in terms of performance. This performance benchmark will be achived by measuring the compilation time of the preprocessors for different sizes of files.

For the experiment, we started by installing Node.js v6.10.2 (includes npm 3.10.10) in a machine running Windows 10 (64 bits), Intel Core i7-6700K – 4.00GHz, 16 GB RAM and SSD.

The next step was the creation of a JavaScript file with the test code. The aim of this test is to compare CSS preprocessors for parsings, nested rules, mixins, variables and math. An excerpt of the test is presented in Listing 26.

The test starts by the generation of the preprocessor code of three sizes (10/100/1000 KB). Then, it uses the internal function of each library for the compilation of the preprocessor code to the CSS format. The compilation process for each library is measured and the processing time are showned in the console. The results are presented in Table 7.

**Table 7** Performance benchmark.

| Preprocessors | 10KB | 100KB | 1000KB |
|---|---|---|---|
| Sass | 9ms | 91ms | 943ms |
| Less | 63ms | 287ms | 2067ms |
| Stylus | 133ms | 581ms | 3969ms |

Based on these results, the main conclusion is that Sass is the fastest of the three. Please note, that the official implementation of Sass is open-source and coded in Ruby. In these tests we didn't use it. Instead, we used libSass a high-performance implementation in C.

## 4 Conclusions

CSS preprocessors are very powerful and they can help streamline your Web development process, especially if you are coming from a programming background. In this paper we surveys a set of CSS preprocessors regarding a set of predefined criteria such as: maturity, coverage and performance.

While it appears that Sass is more widely used and have better performance, Stylus demonstrates that covers very well the main and typical features of a CSS preprocessor. In fact, there isn't really a preprocessor that is better than the other. It usually comes down to the developer and what they are comfortable with using. The most important thing is to use one in order to help you make your CSS more maintainable and extendable.

As future work, the main idea is to extend this survey to other CSS preprocessors and include also some postprocessors that are being used nowadays (such as PostCSS and Rework). Other important upgrade is to enrich the tests not only based on the size of the file but in the diversity of block types used in code. This will create more realistic tests since many of the CSS preprocessors features are often used in production [2].

## References


1 Chris Coyier. *Popularity of CSS Preprocessors*, 2012. CSS-Tricks. `http://css-tricks.com/poll-results-popularity-of-css-preprocessors`.

2 Davood Mazinanian and Nikolaos Tsantalis. An empirical study on the use of CSS preprocessors. In *23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, pages 168–178, 2016.

3 Davood Mazinanian, Nikolaos Tsantalis, and Ali Mesbah. Discovering refactoring opportunities in cascading style sheets. In *22nd International Symposium on Foundations of Software Engineering*, pages 496–506, 2014.

4 Ashley Nolan. The State of Front-End Tooling 2016 – Results, 2016. Personal blog. `https://ashleynolan.co.uk/blog/frontend-tooling-survey-2016-results`.

# XML Parsing in JavaScript

**Alberto Simões**

**Centro Algoritmi, University of Minho, Braga, Portugal; and**
**Instituto Politécnico do Cávado e do Ave, Barcelos, Portugal**
asimoes@ipca.pt

## Abstract

With Web 2.0 the dynamic web got to a reality. With it, some new concepts arived, like the use of asynchronous calls to receive missing data to render a website, instead of requesting a full new page to the server. For this task, and in the recent years, developers use mostly the JSON format for the interchange of data, than XML. Nevertheless, XML is more suitable for some kind of data interchange but, and even if the web is based in SGML/XML standards, processing XML using directly JavaScript is tricky.

In this document, a set of different approaches to parse XML with JavaScript will be described, and a new module, based on a set of translation functions, will be presented. At the end, a set of experiments will be discussed, trying to evaluate how versatile the proposed approach is.

**1998 ACM Subject Classification** I.7.2 Document Preparation / Markup languages, D.3.4 Processors / Parsing

**Keywords and phrases** XML, JSON, Parsing, JavaScript

**Digital Object Identifier** 10.4230/OASIcs.SLATE.2017.9

## 1 Introduction

The Internet is built in a set of standards. First, the SGML (Standard Generalized Markup Language) from which HTML (HyperText Markup Language) was derived. Despite the fact that a lot of problems arrived from the use of such a permissive standard, the XHTML version of HTML, built on top of XML (eXtensible Markup Language) did not stick. Example of it is the HTML5 standard that, although suggesting XML well formed documents, is still based in SGML.

On account of that, browsers needed to implement two different parsers, one for each of the standards[1](SGML and XML). Although not discussed in this document, the Cascading Style Sheets (CSS) standards started to appear. Since JavaScript entered the web world enabling dynamic web sites, the Internet is no longer the same.

To support JavaScript, browsers needed to add a new parser, for the language, together with its interpreter. As for other programming languages, JavaScript support constructs to define data structures, and soon JSON, the JavaScript Object Notation was defined to allow the serialization of data.

The simplicity of JSON compared to the verbosity of XML lead to the use of JSON for most of the Web 2.0 asynchronous calls. And this happened because most tools developed for the web deal with very structured data that is easily serializable in JSON. But XML and JSON has very different capacities, and when in the need of mixed content, XML stands out [6].

[1] In fact, each one of these parsers has a lot of tweaks in order to cope with some behaviors that were the default during the first browsers implementations, and for which compatibility was desired [12].


But, even if XML parsers were added a long time ago on browsers, the truth is that it is still quite hard to manipulate data obtained from an XML file. There are a couple of possibilities:

- taking the advantage of including XML directly in a web page, and just format it using Cascading StyleSheets [3], with the disadvantage that the structure manipulation is limited to the addition or generated content, or the hiding of non relevant data;
- another option is to use XSLT (Extensible Stylesheet Language Transformations) [5, 7, 8], but although browser support for its first version is available on all major browsers, newer versions not available natively on any browser. It also has the major drawback of requiring programmers to learn how to use it. If the syntax itself is basic, given it is a XML based language, the way transformations are declared (in a declarative way, not imperative as most programmers are used to use) can be quite challenging.
- using a JavaScript library, either the built-in DOMParser [10] or any other available, like jQuery. The usage of DOMParser would be the better approach, given its availability on all browsers, and even as a module for node.js. Nevertheless, the Document Object Model (DOM) structure is not trivial, and the methods available directly in JavaScript to manipulate it are not versatile.

This document will only focus on the usage of JavaScript to manipulate the XML file, given the limitations of CSS, and the lack of support of XSLT.

In the next section I will start by explaining what mixed content is, and why and when it matters. Section 3 will discuss the two main approaches used for processing XML in any language. Section 4 will restrict it to JavaScript, and present some of the available tools to manipulate the XML DOM available in Browsers. Section 5 describes the approach implemented and why it is useful. Before concluding, section 6 will show some uses of the tool.

## 2 XML, JSON and Mixed Content

As stated in the introduction, XML and JSON, both, can be used for serialization and, in that arena, JSON is more compact and, with its binary counterparts, like BSON, can be quite fast. Nevertheless, when the goal is to encode mixed content, XML is more adequate, as discussed in depth in [6].

In order to understand the problem, we will consider an example extracted from the project that motivated this problem [13], an on-line dictionary with a RESTful API: Dicionário-Aberto.[2]

One of the methods available in the API is the retrieval of entries from the dictionary (either randomly or by a specific word). Figure 1 shows a random entry from the dictionary as presented to the common user in the web interface.

As can be seen, there are multiple lines, with different definitions. Some of them can be of pure text, but other include quotes and examples, that need to be differentiated. Although this could be codified in JSON, it is not so natural.

The current API for Dicionário-Aberto can return records in both formats. Listing 1 shows how the entry in Figure 1 is encoded in JSON, while Listing 2 shows that same entry encoded in XML.

[2] `http://dicionario-aberto.net`

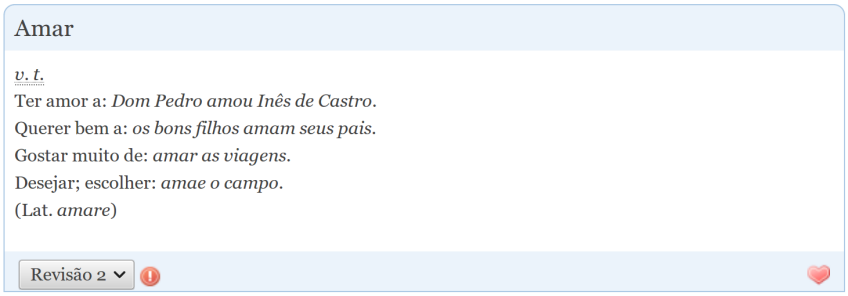



**Figure 1** Entry for the word 'amar' (*to love*).

**Listing 1** Entry for 'amar' encoded in JSON.

```
{ "entry" : {
    "@id" : "amar",
    "form" : { "orth" : "Amar" },
    "sense" : [ {
      "gramGrp" : "v. t.",
      "def" : [
        ["Ter amor a: ", { "quote": "Dom Pedro amou Inês de Castro" }],
        ["Querer bem a: ", { "example": "os bons filhos amam seus pais" }],
        ["Gostar muito de: ", { "example": "amar as viagens" }],
        ["Desejar; escolher: ", { "example": "amai o campo" }]
      ]},
    "etym" : { "@orig" : "Lat", "#text" : "amare" }
} }
```

**Listing 2** Entry for 'amar' encoded in XML.

```
<entry id="amar">
 <form><orth>Amar</orth></form>
 <sense>
  <gramGrp>v. t.</gramGrp>
  <def>Ter amor a: <quote>Dom Pedro amou Inês de Castro</quote>.</def>
  <def>Querer bem a: <example>os bons filhos amam seus pais</example>.</def>
  <def>Gostar muito de: <example>amar as viagens</example>.</def>
  <def>Desejar; escolher: <example>amai o campo</example>.</def>
 </sense>
 <etym orig="Lat">(Lat. <mentioned>amare</mentioned>)</etym>
</entry>
```

While the presented JSON is not too complicated as in some other entries, this example shows how to encode a simple entry a set of recursive lists and dictionaries are needed.

The best thing about the JSON approach, is how JavaScript makes it easy to manipulate. For the XML, there is a big advantage. New browsers allow the inclusion of this snippet directly inside an HTML page, able to be formatted through CSS. The problem is when there are minor details that need to be manipulated somehow.

## 3 XML Processors

There are two main approaches when parsing XML documents:

- SAX (Simple API for XML), that works mostly by defining callbacks to every element tag found in the document, or any entity [4]. Instead of processing the document in a structural fashion, SAX parsers will transform the document progressively, as it is parsed. While this approach is quite simple to implement, it is not versatile for complex XML

transformations. For that, a couple of external data structures would be needed, mostly as if the user create her own document tree structure.

- DOM (Document Object Model) parsing, in the other hand, works by creating an abstract syntax tree for the whole document, as created by mostly compilers of conventional programming languages [9]. The main disadvantage of this approach is the amount of memory needed when processing large documents, as it needs to be all loaded into memory. But, in the other hand, it is quite easy to traverse the tree and do changes, rearranging the branches, pruning them, or adding new ones.
  For the pointed disadvantage, there are two main approaches that have been used for large documents:
  - XML Pull Parsing allows the construction of DOM trees for specific parts of a document, that are retrieved as needed.
  - As most of the large XML documents have a repetitive structure, just as logs or collections of resources, there are parser implementations that chunk the large document on the repetitive element, and parses their contents using a standard DOM Parser.

  Nevertheless, usually the size of XML documents sent through the web during AJAX calls are small, and this is not a relevant problem.

Most programming languages have libraries or modules that use some of these approaches. The well known Expat[3] parser and LibXML[4] support both approaches, and have binding for most of the common programming languages. Unfortunately that is not true for JavaScript.

## 4 XML and JavaScript

Despite the fact that XML is parsed by browser for a long time, the amount of tools to process XML with JavaScript is quite limited. This might be a result of the arrival of JSON and the small number of users actually needing real mixed content.

Browsing the Internet for JavaScript libraries to manipulate XML there are two obvious answers:

- use the built-in DOMParser [10], and its DOM structure, navigating through each element top-down (from the root node to the leafs), looking for the relevant data;
- use jQuery [2, 11] and its selectors[5] (based on CSS selectors).

Both approaches are easy to use, but not very versatile. To explain this, consider the example shown in Listing 2, and a pair of simple tasks:

- **Task I.** Find the orthographic form of the entry (`orth` tag):
  **DOM** using the DOM tree is not too hard, specially when looking for a specific leaf of the DOM tree. Considering the variable `entry` to contain the XML fragment above, the following code would retrieve the orthographic form:

```
var parser = new DOMParser();
var doc = parser.parseFromString(entry , "text/xml");
var term = doc.activeElement.children[0]
                            .children[0].childNodes[0].data);
```

[3] https://libexpat.github.io/
[4] http://xmlsoft.org/
[5] Note that jQuery can be used to manipulate the DOM as well, but it just adds a couple of extra methods to make the tree traversal easier. Also, although jQuery uses CSS selectors, there is the possibility to add support for XPath as well.

Accordingly with the standard, the method `getElementsByName` should be available in an XMLDocument instance (`doc` in the code above). Nevertheless, it does not work correctly on all browsers. A recent Firefox would complain about a non existing function (even if the Mozilla Developers Network documents that an XMLDocument instance inherits methods from Document).

**jQuery** this is the simplest task for jQuery: as there is only one tag with that name, a simple selector can be used. Considering that the variable `entry` is a string containing the XML fragment, the following code would suffice:

```
var term = $(entry).find('orth')[0];
```

- **Task II.** Remove the `example` elements, and remove the colon before them.

  **DOM** giving the quite unstable API to manipulate an XMLDocument directly in the browser, no solution will be presented using directly the DOM. From the example before it could be seen that a traversal approach would take too long to write, and would be error prone (how many children levels?), and the lack of support for XMLDocument methods would make the resulting code work (or completely not work) accordingly with the used browser.

  **jQuery** using jQuery for this task is a little more tiresome. The first task is to remove the `example` elements, while the second is to cycle all `def` elements to remove the colon. While the whole jQuery syntax is based in the functional paradigm, for this task it is needed to remind that JavaScript is an object oriented programming language, and therefore, changes need to be done directly on the XML object. Also, notice that at the end the user gets a jQuery XML document, and not a string with the parsed XML[6].

```
$xml = $(entry);
$xml.find('example').remove();
$xml.find('def').map(function(i,val){
   val.innerText = val.innerText.replace(/:\s*\.\s*$/,".");
});
```

## 5 Traversing the DOM Tree

The implemented approach is based in a Perl module, named XML::DT [1], that uses a bottom-up approach to process the DOM tree. Following its brother name, the JavaScript implementation is named XML-DT-JS.

It works like a dispatch table where, for each element, a function is defined. The traversal algorithm will start with the leafs, and feed the function with the element name, its contents (the CDATA) and the associated properties. The function can do whatever is needed to this data, and must return a string (that can contain XML).

The non-leaf nodes' functions receive the element name, and the associated properties, as the leaf nodes' functions, but the content itself, is supplied as returned by the child nodes processors. In the case the element has more than one child, then their results are concatenated into a single string.

Listing 3 shows the code to convert from the following input XML document to the respective output string:

[6] This fact can be seen as an advantage or disadvantage, depending on the user goals.

**Input:**

```
<root><foo>zbr</foo><bar>Something</bar></root>
```

**Output:**

```
<root><zbr>foo</zbr>Hello</root>
```

**Listing 3** Simple XML-DT-JS code example.

```
xml$dt.process(input,
{ root: function(q,c,v) { return xml$dt.tag(q,c,v); }
  foo:  function(q,c,v) { return xml$dt.tag(c,v); },
  bar:  function(q,c,v) { return "Hello"; } });
```

The `xml$dt.process` function is the main method to call for the structural processing. First argument is the XML string to process. Second argument is a mapping from tag names to functions. Each function receive the tag name (`q` variable[7], the tag contents (`c` variable) and a map of attribute names to their values (`v` variable). The functions should return the processed node as a string.

The utility `xml$dt.tag` function allows the quick creation of a XML string, given the tag name, tag contents, and attributes (in the same order as they are received by the `process` function.

There are three special element names that can be defined:

- **#document** allows to define a function associated to the root node, without the need to know what is its name. It also allows to define a function to deal with the final tree, after the root node processing. By default it is the identity function.
- **#text** allows to define a function to process all the text nodes, before them being processed by the respective enclosing element. By default it is the identity function.
  Listing 4 shows the code to convert from the following input XML document to the respective output string:

  **Input:**

  ```
  <list><item qt="2">banana</item><item qt="5">pineapple</item></list>
  ```

  **Output:**

  ```
  <list>two bananas</item><item>five pineapples</item></list>
  ```

  **Listing 4** XML-DT-JS code using `#text` rule.

  ```
  var nrs = ['zero','one','two','three','four','five'];
  xml$dt.process(input, {
   '#text': function(q,c)   { return c + "s"; },
   item:    function(q,c,v) { return xml$dt.tag(q, nrs[v.qt] +" "+ c); },
   list:    function(q,c,v) { return xml$dt.tag(q,c,v); } });
  ```

- **#default** defines a function to process any element whose processing function is not defined. Therefore, in cases where the processing algorithm can be derived from the element name or its attributes, a simple default processing function can be enough.
  Listing 5 shows the code to prefix every tag with a namespace, as in the following example:

  **Input:**

  ```
  <list><item qt="2">banana</item><item qt="5">pineapple</item></list>
  ```

[7] These variable names can be changed, but are kept in our examples to keep the same variable names used by the Perl version that were, also, kept from Omnimark.

**Output:**

```
<ex:list><ex:item qt="2">banana</ex:item>
         <ex:item qt="5">pineapple</ex:item></ex:list>
```

**Listing 5** XML-DT-JS code using `#default` rule.

```
xml$dt.process(input, {
 '#default' : function(q,c,v) { return xml$dt.tag("ex:"+q, c, v); });
```

On top of this basic traversal algorithm, a few features were added, to allow more control of the processing functions, and to allow easier definition of markup converters:

- For markup conversion, where the goal is just to change the element name from one to another, a shortcut mapping can be defined.
  Listing 6 shows the code to convert some tags directly to HTML tags:

  **Input:**

  ```
  <list><item>bananas</item><item>pineapples</item></list>
  ```

  **Output:**

  ```
  <ul><li>bananas</li><li>pineapples</li></ul>
  ```

  **Listing 6** XML-DT-JS code using `#map` shortcut.

  ```
  xml$dt.process(input, { '#map' : { list: 'ul', item: 'li' }});
  ```

- In some situations, it is relevant to store some data from one node on its father, rather than just returning it as a string. For example, it can be handy when returning two distinct values, or when it is easier for the elements fathers to process a list rather than a concatenated string, as shown in the next example:
  Listing 7 shows the code to convert some tags directly to HTML tags:

  **Input:**

  ```
  <list><item qt="4">bananas</item><item qt="2">pineapples</item></list>
  ```

  **Output:**

  ```
  <list total="6''><item qt="4">bananas</item>
                  <item qt="2">pineapples</item></list>
  ```

  **Listing 7** XML-DT-JS code using `father` variable.

  ```
  xml$dt.process(input, {
    item: function(q,c,v){
      if ('total' in xml$dt.father) xml$dt.father.total += v.qt;
      else                          xml$dt.father.total = v.qt;
      return xml$dt.tag(q,c,v);
    },
    list: function(q,c,v) { return xml$dt.tag(q,c,v); }});
  ```

  Notice that the `father` shortcut accesses the attributes of the father element. Therefore, when calling the processing function for that element, the attributes defined using that shotcut will be available in the `v` variable.

- This section concludes with the implementation of the two tasks described in section 4: Obtaining the orthographic form for an entry:

  ```
  var term;
  xml$dt.process(entry, { orth: function(q,c,v) { term = c; } });
  ```

This is not a clean solution, as it is not a functional approach, doing the job using lateral effects. Nevertheless, it is not that easy to implement this same behavior using the functional paradigm.

- Removing examples from the definitions:

```
var result = xml$dt.process(entry,
{ example: function() { return ""; },
  def: function(q,c,v) {
     return xml$dt.tag(q, c.replace(/:\s*\.\s*$/, "."), v); }
});
```

The main advantage from this solution when compared with the jQuery solution presented before, is that it does not require the user to know how to use the `map` function, or to deal with the rather obscure `innerText` property.

## 6 Using XML-DT-JS

This section presents some real examples where XML-DT-JS is being used, in the context of Dicionário-Aberto. Fortunately, modern browsers allow the embedding of XML snippets inside of HTML documents, and their formatting with CSS rules. Therefore, everything that can be accomplished just by the definition of CSS rules has priority over the processing of the entries. Simple tasks, like changing the font weight or the block-formatting of tags are done directly in CSS.

- Extracting the orthographic form from the entry identifier and, if present, the sense number, formatting it properly in HTML (see Listing 8.

**Listing 8** Extract orthographic form and sense number from an entry.

```
function getEntryTerm(data) {
    return xml$dt.process(data, {
      entry: function(q,c,v) {
        var word = v.id;
        if (word.match(/:\d+$/) {
          word = word.replace(/:(\d+)/, "<sup>$1</sup>");
        }
        return word;
      }
    });
}
```

This task is quite similar to the extraction of the orthographic form presented before. In this case, only the `id` attribute from the `entry` tag is processed and extracted. Given this is the root node, it suffices to return it, obtaining a better functional approach.

- Other task currently being solved with XML-DT-JS is the formatting of an entry. Given some entries include definitions with old wiki-like markup (underscores instead of italic), some rules treat the element textual contents. In the other hand, some entries include common new-lines to mark different senses, and therefore, they need to be correctly formatted as line breaks. Finally, the `form` tag needs to be renamed, given that HTML already uses it (see Listing 9).

There are some other places where XML-DT-JS is handy, but those situations does not add any more to this document, and therefore, will not be presented.

**Listing 9** Formatting a dictionary entry, with some pre-processing.

```
function formatEntry(data) {
    return = xml$dt.process(data, {
        '#map' : { 'form': 'div' },
        '#default' : function(q,c,v) { return xml$dt.tag(q,c,v); },
        'def' : function(q,c,v) {
          var s = c.replace(/(^\n(\s*\n)*|\n(\s*\n)*$)/g,"")
                   .replace(/_([^_]+)_/g, "<i>$1</i>")
                   .replace(/\n(\s*\n)*/g,"<br/>");
          return xml$dt.tag(q,s,v);
        },
        'etym' : function(q,c,v) {
            return c.replace(/_([^_]+)_/g, "<i>$1</i>");
        },
      });
}
```

## 7 Conclusions

In this document a small library to process XML documents using JavaScript in the browser was presented[8]. The library uses a bottom-up approach to process the structure of an XML document. Given that the traversal algorithm is predefined, the user just needs to implement the rules of how each XML tag will be processed.

The tool was developed in the context of a project where the mixed content support of XML is relevant. While current usage in the context of the project is quite limited, the experience of using this kind of processors with the Perl programming language shows that this approach is very versatile.

In the future, it is intended to add to this tool the following functionality:

- support types: allow the definition of structural types for some tags (for example, specifying that a list is a collection of items) instead of always using the concatenation of strings;
- allow its usage in the server side, with node.js, where DOMParser is not available by default;
- support direct access to the root element of the tree (using a similar approach as the *father* attribute presented before);
- better support for entities and entities escaping;
- an analysis on the impact of the traversal time, versus the usage of CSS or XPath expressions.

## References


1 José João Almeida and José Carlos Ramalho. XML::DT a Perl Down-Translation module. In *XML-Europe'99*, Granada, Spain, May 1999.

2 Bear Bibeault and Yehuda Katz. *jQuery in Action, Second Edition.* Manning Publications, 2 edition, 2010.

3 Bert Bos. Descriptions of all CSS specifications. Technical report, World Wide Web Consortium (w3c), 2017. URL: `https://www.w3.org/Style/CSS/specs.en.html`.

4 David Brownell. *Processing XML Efficiently with Java.* O'Reilly Media, 2002.


[8] The current version of the library is available at the following GIT repository: `https://gitlab.com/ambs/xml-dt-js`

5 James Clark. XSL Transformations (XSLT) – version 1.0. Technical report, World Wide Web Consortium (w3c), 1999. URL: https://www.w3.org/TR/xslt.

6 Rúben Fonseca and Alberto Simões. Alternativas ao XML: YAML e JSON. In José Carlos Ramalho, João Correia Lopes, and Luís Carríço, editors, *XATA 2007 – 5th Conferência Nacional em XML, Aplicações e Tecnologias Associadas*, pages 33–46, February 2007.

7 Michael Kay. XSL Transformations (XSLT) – version 2.0. Technical report, World Wide Web Consortium (w3c), 2007. URL: https://www.w3.org/TR/xslt20/.

8 Michael Kay. XSL Transformations (XSLT) – version 3.0. Technical report, World Wide Web Consortium (w3c), 2017. URL: https://www.w3.org/TR/xslt30/.

9 Peter-Paul Koch. The document object model: an introduction. *Digital Web Magazine*, May 2001.

10 Travis Leithead. DOM parsing and serialization. Technical report, World Wide Web Consortium (w3c), 2016. URL: https://www.w3.org/TR/DOM-Parsing/.

11 Code Lindley. *jQuery Succinctly.* Syncfusion, 2012.

12 Mark Pilgrim. *HTML5: Up and Running.* O'Reilly Media, Inc., 1st edition, 2010.

13 Alberto Simões, Álvaro Iriarte, and José João Almeida. Dicionário-aberto – a source of resources for the portuguese language processing. *Computational Processing of the Portuguese Language, Lecture Notes for Artificial Intelligence*, 7243:121–127, April 2012.

# Indexing XML Documents Using Tree Paths Automaton*

**Eliška Šestáková[1] and Jan Janoušek[2]**

1 **Faculty of Information Technology, Czech Technical University in Prague, Prague, Czech Republic**
Eliska.Sestakova@fit.cvut.cz

2 **Faculty of Information Technology, Czech Technical University in Prague, Prague, Czech Republic**
Jan.Janousek@fit.cvut.cz

## Abstract

An XML document can be viewed as a tree in a natural way. Processing tree data structures usually requires a pushdown automaton as a model of computation. Therefore, it is interesting that a finite automaton can be used to solve the XML index problem. In this paper, we attempt to support a significant fragment of XPath queries which may use any combination of child (i.e., `/`) and descendant-or-self (i.e., `//`) axis. A systematic approach to the construction of such XML index, which is a finite automaton called Tree Paths Automaton, is presented. Given an XML tree model $T$, the tree is first of all preprocessed by means of its linear fragments called string paths. Since only path queries are considered, the branching structure of the XML tree model can be omitted. For individual string paths, smaller Tree Paths Automata are built, and they are afterwards combined to form the index. The searching phase uses the index, reads an input query $Q$ of size $m$, and computes the list of positions of all occurrences of $Q$ in the tree $T$. The searching is performed in time $\mathcal{O}(m)$ and does not depend on the size of the XML document. Although the number of queries is clearly exponential in the number of nodes of the XML tree model, the size of the index seems to be, according to our experimental results, usually only about 2.5 times larger than the size of the original document.

**1998 ACM Subject Classification** F.1.1 Models of Computation

**Keywords and phrases** XML, XPath, index, tree, finite automaton

**Digital Object Identifier** 10.4230/OASIcs.SLATE.2017.10

## 1 Introduction

XML plays an important role in many aspects of software development, often to simplify data storage and sharing. Therefore, efficient storing and querying XML data are key tasks which have been extensively studied during the past years. XML data is stored in a plain text format. This provides a software- and hardware-independent way of storing data. To be able to retrieve data from XML documents, various query languages such as XPath [2], XPointer [5], and XLink [6] have been designed.

However, without a structural summary, query processing can be quite inefficient due to an exhaustive traversal on XML data. To achieve fast searching and efficient processing of

* This research has been partially supported by grant CTU in Prague as project No. SGS17/209/OHK3/3T/18.





queries, we can preprocess the data subject and construct an index. This allows us to answer number of queries with low requirements for time complexity.

An XML document can be simply treated as a stream of plain text. Thus, stringology algorithms [3, 4] are applicable in this field. The theory of text indexing is well-researched and uses many sophisticated data structures, such as suffix tree, suffix array, or factor automaton.

However, the internal structure of XML documents can be also viewed as a tree in a natural way. The algorithmic discipline interested in processing tree data structures is called arbology, and it was officially introduced at London Stringology Days 2009 conference. Arbology solves problems such as tree pattern matching, tree indexing, or finding repeats in trees. For its algorithms, arbology uses a standard pushdown automaton as the basic model of computation, unlike stringology where a finite automaton is used.

Nowadays, many methods for indexing XML documents exist, but most of them lack clear references to a systematic approach of the standard theory of formal languages and automata. XML indexes usually work with the tree structure of an XML document, and according to their approaches, we can divide them as follows:

**Graph-based methods** construct a structural path summary; used to improve especially single path queries. See DataGuides [7], 1-Index [12], PP-Index [17], F&B-Index [8], or MTree [13].

**Sequence-based methods** transform both the source data and query into sequences. Therefore, querying XML data is equivalent to finding subsequence matches. See ViST [18], PRIX [14].

**Node coding methods** design codes for each node in order to evaluate the relationship among nodes by computation. See XISS [9].

**Adaptive methods** adapt their structure to suit the query workload. Therefore, adaptive methods index only the frequently used queries. See APEX Index [1].

Each of the methods listed above has its own advantages and disadvantages. Graph-based methods often do not support complex queries; sequence-based methods are likely to generate approximate solutions; node coding methods are difficult to be applied to ever changing data source; and adaptive methods perform low efficiency on non-frequent queries.

In [16], we discussed the automata-based approach for solving the XML index problem and presented Tree String Path Subsequences Automaton (TSPSA), an index for all linear XPath queries using descendant-or-self axis (i.e., `//`) only. In this paper, we introduce Tree Paths Automaton (TPA) which is designed to process more significant fragment of XPath queries. It is able to answer all queries with any combination of child (i.e., `/`) and descendant-or-self (i.e., `//`) axis, noted as $XP^{\{/,//,name-test\}}$.

Given an XML document $D$ with its corresponding XML tree model $T(D)$, the searching phase uses the index, reads an input query $Q$ of size $m$, and computes the list of positions of all occurrences of $Q$ in the tree $T(D)$. The searching is performed in time $\mathcal{O}(m)$ and does not depend on the size of the original document $D$. Although the number of distinct queries is exponential in the number of nodes of the XML tree model, our experiments suggest that determinisation will result in a smaller number of states.

Both TSPSA and TPA support only linear XPath queries. However, the techniques described here may also be relevant to the general XPath processing problem. First, processing linear expressions is a subproblem in processing more complex queries, as we can decompose them into linear fragments. Second, this can be seen as a building block for more powerful processors, such as pushdown automata, able to process branching queries. Moreover, it is


**Figure 1** XML Index Problem.

easy to combine the index presented in this paper with other automata-based indexes using standard methods of automata theory.

## 2 Basic Notions

An *alphabet* $A$ is a finite non-empty set whose elements are called symbols. A *nondeterministic finite automaton* (NFA) is a 5-tuple $M = (Q, A, \delta, q_0, F)$, where $Q$ is a finite set of states, $A$ is an alphabet, $\delta$ is a state transition function from $Q \times A$ to the power set of $Q$, $q_0 \in Q$ is an initial state, $F \subseteq Q$ is a set of final states. A finite automaton is *deterministic* (DFA) if $\forall a \in A, q \in Q : |\delta(q, a)| \leq 1$.

A *rooted and directed tree* $T$ is an acyclic connected directed graph $T = (N, E)$, where $N$ is a set of nodes and $E$ is a set of ordered pairs of nodes called directed edges. A *root* is a special node $r \in N$ with in-degree 0. All other nodes of a tree $T$ have in-degree 1. There is just one path from the root $r$ to every node $n \in N$, where $n \neq r$. A node $n_1$ is a *direct descendant* of a node $n_2$ if a pair $(n_2, n_1) \in E$.

A *labelling* of a tree $T = (N, E)$ is a mapping $N$ into a set of labels. $T$ is called a *labelled tree* if it is equipped with a labelling. $T$ is called an *ordered* tree if a left-to-right order among siblings in $T$ is given. Any node of a tree with out-degree 0 is called a *leaf*. A *depth* of a node $n$, noted as $depth(n)$, is the number of directed edges from the root to the node $n$.

## 3 Problem Statement

Basically, the XML index problem is to construct an effective data structure able to efficiently process XML query languages, such as XPath. There are two crucial issues connected with all indexing methods. First, the requirement for a small size of the index which, in the best case, should be linear in the size of the preprocessed subject. The second essential feature of the index is very fast query processing. Ideally, queries should be answered in time linear to the size of the query and should not depend on the size of the subject where the queries are located. If these requirements are fulfilled, the index structure allows us to answer number of queries with low requirements for both space and time complexity.

At first, indexing methods usually preprocess the data subject and then construct a structure (an index) that allows to efficiently answer queries related to the content of the subject. In other words, occurrences of input patterns in the subject can be located repeatedly and quickly. See the diagram of the XML index problem in Figure 1.

Among others, the theory of indexing a data structure and finding efficient solutions for particular indexing problems allow us to understand the problem better. Moreover, sometimes various indexes for particular problems can be combined to index, for instance, unions or concatenations. In this last aspect, especially the use of the theory of formal languages and automata could be helpful.

However, the XML index problem is a challenging area. Using only the two most commonly used XPath axes (child axis `/` and descendant-or-self `//` axis), the number of potential queries is exponential (e.g., $\mathcal{O}(2.62^n)$ for a simple linear XML tree with $n$ nodes [10]).

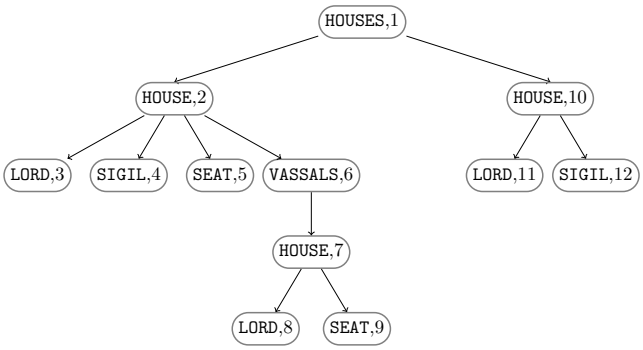



**Figure 2** XML tree model $T(D)$ of the XML document $D$ from Example 2.

In this paper, we attempt to support exactly this significant fragment of XPath queries, noted as $XP^{\{/,//,name-test\}}$.

## 4 XML Tree Model

We model an XML document as an ordered labelled tree where nodes correspond to elements, and edges represent element inclusion relationships. Hence, we only consider the structure of XML documents and, therefore, ignore attributes and the text in leaves.

A node in an XML tree model is represented by a pair $(label, id)$ where $label$ and $id$ represent a tag name and an identifier, respectively. We use a preorder numbering scheme to uniquely assign an identifier to each of the tree nodes. Unique tag names of an XML document form its XML alphabet, formally defined as follows.

▶ **Definition 1** (XML alphabet)**.** Let $D$ be an XML document. An *XML alphabet* $A$ of $D$, represented by $A(D)$, is an alphabet where each symbol represents a tag name (label) of an XML element in $D$.

▶ **Example 2.** Let $D$ be the following XML document. The corresponding XML alphabet $A$ is $A(D) = \{$`HOUSES, HOUSE, LORD, SIGIL, SEAT, VASSALS`$\}$. Figure 2 shows its corresponding XML tree model $T(D)$.

```
<HOUSES>
    <HOUSE name="Stark">
        <LORD>Eddard Stark</LORD>
        <SIGIL>Direwolf</SIGIL>
        <SEAT>Winterfell</SEAT>
        <VASSALS>
           <HOUSE name="Karstark">
              <LORD>Rickard Karstark</LORD>
              <SEAT>Karhold</SEAT>
           </HOUSE>
        </VASSALS>
    </HOUSE>
    <HOUSE name="Targaryen">
        <LORD>Daenerys Targaryen</LORD>
        <SIGIL>Dragon</SIGIL>
    </HOUSE>
</HOUSES>
```

## 5 Tree Paths Automaton

Tree Paths Automaton (TPA) is a finite automaton designed to process a significant fragment of XPath queries which may use any combination of child (i.e., /) and descendant-or-self (i.e., //) axis, noted as $XP^{\{/,//,name-test\}}$. Formally, we can represent such fragment of XPath queries over an XML document $D$ by the following context–free grammar:

$$G = (\{S\}, A(D), \{S \rightarrow SS \mid /a \mid //a\} \wedge a \in A(D), S)\,.$$

This section describes a systematic approach to the construction of TPA and demonstrates it by several examples. Hence, the index is simple and well understandable for anyone who is familiar with the automata theory.

Given an XML tree model $T$, the tree is first of all preprocessed by means of its linear fragments called string paths. Since only path queries are considered, the branching structure of the XML tree model can be omitted. For individual string paths, smaller Tree Paths Automata are built, and they are afterwards combined using product construction (union) to form the index.

▶ **Definition 3** (String path)**.** Let $T$ be an XML tree model of height $h$. *A string path* $P = n_1 n_2 \dots n_t$ $(t \leq h)$ of $T$ is a linear path leading from the root $r = n_1$ to the leaf $n_t$.

▶ **Definition 4** (String path alphabet)**.** Let $P$ be a string path of some XML tree model. *A string path alphabet* $A$ of $P$, represented by $A(P)$, is an alphabet where each symbol represents a node label in $P$.

▶ **Definition 5** (String paths set)**.** Let $T$ be an XML tree model with $k$ leaves. A set of all string paths over $T$ is called a *string paths set*, denoted by $P(T) = \{P_1, P_2 \dots P_k\}$.

▶ **Example 6.** Consider the XML tree model $T$ illustrated in Figure 2. We show the content of the corresponding string paths set $P(T)$ below. Each node $n$ of $T$ is represented by its label and identifier, which is shown in parenthesis.

- $P_1$ = HOUSES(1) HOUSE(2) LORD(3),
- $P_2$ = HOUSES(1) HOUSE(2) SIGIL(4),
- $P_3$ = HOUSES(1) HOUSE(2) SEAT(5),
- $P_4$ = HOUSES(1) HOUSE(2) VASSALS(6) HOUSE(7) LORD(8),
- $P_5$ = HOUSES(1) HOUSE(2) VASSALS(6) HOUSE(7) SEAT(9),
- $P_6$ = HOUSES(1) HOUSE(10) LORD(11),
- $P_7$ = HOUSES(1) HOUSE(10) SIGIL(12).

The corresponding string path alphabets are as follows:

- $A(P_1) = A(P_6) = \{$HOUSES, HOUSE, LORD$\}$,
- $A(P_2) = A(P_7) = \{$HOUSES, HOUSE, SIGIL$\}$,
- $A(P_3) = \{$HOUSES, HOUSE, SEAT$\}$,
- $A(P_4) = \{$HOUSES, HOUSE, VASSALS, LORD$\}$,
- $A(P_5) = \{$HOUSES, HOUSE, VASSALS, SEAT$\}$.

XPath queries containing only child axis (i.e., /) are basically prefixes of individual string paths. Therefore, to support only $XP^{\{/,name-test\}}$ fragment of XPath queries, we can use a prefix automaton constructed for a set of strings (a string paths set). At first, for each string path $P$, a deterministic prefix automaton accepting all $XP^{\{/,name-test\}}$ queries of $P$ can be constructed. Afterwards, individual automata can be combined using product construction (union).

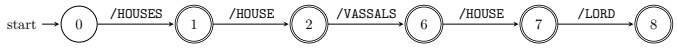



**Figure 3** Deterministic prefix automaton for the string path $P =$ `HOUSES`(1) `HOUSE`(2) `VASSALS`(6) `HOUSE`(7) `LORD`(8) from Example 7.

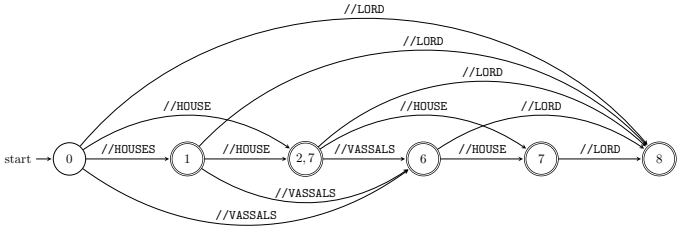



**Figure 4** Deterministic subsequence automaton for the string path $P =$ `HOUSES`(1) `HOUSE`(2) `VASSALS`(6) `HOUSE`(7) `LORD`(8) from Example 7.

**Data:** A string path $P = n_1 n_2 \dots n_{|P|}$.
**Result:** DFA $M = (Q, A, \delta, 0, F)$ accepting all $XP^{\{/,name-test\}}$ queries of $P$.

1. $Q \leftarrow \{0, id(n_1), id(n_2), \dots, id(n_{|P|})\}$,
2. $F \leftarrow Q \setminus \{0\}$,
3. $A = \{/a : a \in A(P)\}$,
4. $\delta(0, /label(n_1)) \leftarrow id(n_1)$, $\forall i \in \{1, 2, \dots, |P| - 1\} : \delta(id(n_i), /label(n_{i+1})) \leftarrow id(n_{i+1})$.

**Algorithm 1:** Construction of a deterministic prefix automaton for a single string path.

**Data:** A string path $P = n_1 n_2 \dots n_{|P|}$.
**Result:** DFA $M = (Q, A, \delta, 0, F)$ accepting all $XP^{\{//,name-test\}}$ queries of $P$.

1. Construct DFA $M_1 = (Q_1, A, \delta_1, 0, F_1)$ accepting all non-empty "prefixes" of $P$ as follows:
   a. $Q_1 \leftarrow \{0, id(n_1), id(n_2), \dots, id(n_{|P|})\}$,
   b. $F_1 \leftarrow Q_1 \setminus \{0\}$,
   c. $A = \{//a : a \in A(P)\}$,
   d. $\delta_1(0, //label(n_1)) \leftarrow id(n_1)$,
      $\forall i \in \{1, 2, \dots, |P| - 1\} : \delta_1(id(n_i), //label(n_{i+1})) \leftarrow id(n_{i+1})$.
2. Insert $\varepsilon$-transitions into the automaton $M_1$ leading from each state to its next state. Resulting automaton $M_2 = (Q_2, A, \delta_2, 0, F_2)$ where
   a. $Q_2 \leftarrow Q_1$, $F_2 \leftarrow F_1$,
   b. $\delta_2 \leftarrow \delta_1 \cup \delta'$ and $\delta'(0, \varepsilon) \leftarrow id(n_1)$,
      $\forall i \in \{1, 2, \dots, |P| - 1\} : \delta'(id(n_i), \varepsilon) \leftarrow id(n_{i+1})$.
3. Eliminate all $\varepsilon$-transitions. The resulting automaton is $M_3$.
4. Construct a deterministic finite automaton $M$ equivalent to $M_3$ using standard determinisation algorithm based on the subset construction (see [11], Algorithm 1.40).

**Algorithm 2:** Construction of a deterministic subsequence automaton for a single string path.

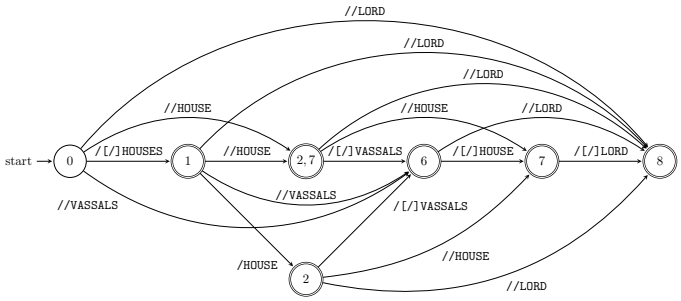



**Figure 5** Deterministic Tree Paths Automaton for the string path $P$ = `HOUSES`(1) `HOUSE`(2) `VASSALS`(6) `HOUSE`(7) `LORD`(8) from Example 7.

To satisfy XPath queries $XP^{\{//,name-test\}}$ containing descendant-or-self-axis (i.e., `//`) only, we are interested in subsequences of a string path rather than its prefixes. For each string path $P$, we can construct a deterministic subsequence automaton, in this case, accepting all $XP^{\{//,name-test\}}$ queries of $P$. Afterwards, by product construction, we get Tree String Path Subsequences Automaton, which we presented earlier in [16].

To provide a solution for XPath queries $XP^{\{/,//,name-test\}}$ containing any combination of child and descendant-or-self axis, we first propose a building algorithm that combines prefix and subsequence automata for a single string path $P$ to answer all $XP^{\{/,//,name-test\}}$ queries of $P$. See Algorithm 3 and Example 7.

▶ **Example 7.** Let $D$ and $T(D)$ be the XML document and its corresponding XML tree model from Example 2 and Figure 2, respectively. Given $P$ = `HOUSES`(1) `HOUSE`(2) `VASSALS`(6) `HOUSE`(7) `LORD`(8) as the input string path, Algorithm 3 conducts these steps:

1. constructing a deterministic prefix automaton for $P$ as shown in Figure 3,
2. building a deterministic subsequence automaton for $P$ as shown in Figure 4,
3. combining these two automata as described in step 3 of the algorithm. See resulting Tree Paths Automaton for $P$ in Figure 5. Note, that transition rules $\delta(p,$`/[/]LABEL`$) = q$ represent two transitions leading from the state $p$ to the state $q$: $\delta(p,$`/LABEL`$) = q$ and $\delta(p,$`//LABEL`$) = q$.

To obtain the final index for an XML document, we again use the product construction (union) of automata that were constructed for individual string paths by Algorithm 3. Algorithm 4 describes the whole process in detail and Example 8 demonstrates the result.

▶ **Example 8.** Let $D$ be the XML document from Example 2. The corresponding Tree Paths Automaton accepting all $XP^{\{/,//,name-test\}}$ queries, constructed by Algorithm 4, is shown in Figure 6. Again, we note that transition rules $\delta(p,$`/[/]LABEL`$) = q$ represent two transitions leading from the state $p$ to the state $q$: $\delta(p,$`/LABEL`$) = q$ and $\delta(p,$`//LABEL`$) = q$.

## 5.1 Evaluation of Input Queries

This section describes the searching phase using the index. To compute positions of all occurrences of an input query $Q$ in an XML tree model $T(D)$ of given XML document $D$, we simply run Tree Paths Automaton on the input query. Eventually, the answer for the input query is given by the d-subset contained in the terminal state of the automaton. If there is no transition that matches the input symbol, the automaton stops and rejects the input. Therefore, there are no elements in the XML document satisfying the query.

**Data:** A string path $P = n_1 n_2 \dots n_{|P|}$ of an XML tree model using preorder numbering scheme.
**Result:** DFA $M = (Q, A, \delta, 0, F)$ accepting all $XP^{\{/,//,name-test\}}$ queries of $P$.

1. Construct a deterministic finite automaton $M_1 = (Q_1, A_1, \delta_1, 0, F_1)$ accepting all $XP^{\{/,name-test\}}$ queries of $P$ using Algorithm 1.
2. Construct a deterministic finite automaton $M_2 = (Q_2, A_2, \delta_2, 0, F_2)$ accepting all $XP^{\{//,name-test\}}$ queries of $P$ using Algorithm 2.
3. Construct a deterministic finite automaton $M = (Q, A_1 \cup A_2, \delta, 0, Q \setminus \{0\})$ accepting all $XP^{\{/,//,name-test\}}$ queries of $P$ as follows:

initialize $Q = Q_1 \cup Q_2$;
create a new queue $S$ and initialize $S = Q$;
**while** *$S$ is not empty* **do**
  State $q \leftarrow S.pop$;
  **forall** $a \in A_1$ **do**
    create a new d-subset $d$;
    **forall** *numbers $n$ in the d-subset of $q$* **do**
      **if** $\delta_1(n, a) \neq \emptyset$ **then**
        add $n$ into $d$;
      **end**
    **end**
    **if** $d \notin Q$ **then**
      $Q = Q \cup \{d\}$;
      $S.push(d)$
    **end**
    $\delta(q, a) \leftarrow d$ ; ▷ `add / transitions`
  **end**
  find the smallest number $m$ in the d-subset of $q$;
  find a matching state $q_2 \in Q_2$ containing $m$ as the smallest number in its d-subset;
  $\forall a \in A_2 : \delta(q, a) \leftarrow \delta_2(q_2, a)$ ; ▷ `add // transitions`
**end**

**Algorithm 3:** Construction of Tree Paths Automaton for a single string path.

**Data:** A string paths set $P(T) = \{P_1, P_2, \dots, P_k\}$ of an XML tree model $T(D)$ with $k$ leaves.
**Result:** DFA $M$ accepting all $XP^{\{/,//,name-test\}}$ queries of the XML document $D$.

1. For all $P_i \in P(T)$ construct a finite automaton $M_i = (Q_i, A_i, \delta_i, 0, F_i)$ accepting all $XP^{\{/,//,name-test\}}$ queries of $P_i$ using Algorithm 3.
2. Construct a deterministic Tree Paths Automaton $M = (Q, \{/a, //a : a \in A(D)\}, \delta, 0, Q \setminus \{0\})$ accepting all $XP^{\{/,//,name-test\}}$ queries of the XML document $D$ using product construction (union).

**Algorithm 4:** Construction of Tree Paths Automaton for an XML document $D$.

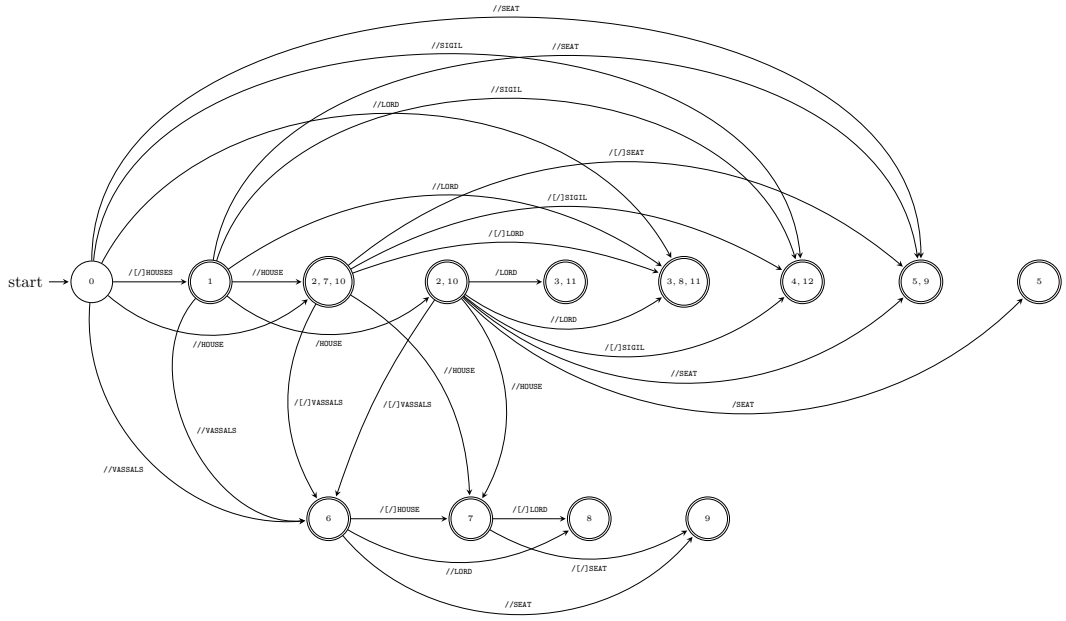



**Figure 6** Deterministic Tree Paths Automaton from Example 8.

## 6 Time and Space Complexities

Tree Paths Automaton is designed to efficiently evaluate all $XP^{\{/,//,name-test\}}$ queries of an XML document $D$. The number of such queries is exponential in the number of nodes of the XML tree model $T(D)$. However, the running time for answering a query is clearly linear in the size of the query and does not depend on the size of the XML document, i.e., $\mathcal{O}(m)$ for a query of size $m$.

The size of the index, i.e., the number of states of the deterministic Tree Paths Automaton, is closely related to the equivalence problem of XPath queries. Two XPath queries are equivalent if and only if their result sets of elements are equal. Each state of TPA corresponds to an answer of a single query or a collection of queries. Although the number of different queries accepted by TPA is exponential, usually a lot of them is equivalent.

That is, if we know the number of unique query answers, we can construct a deterministic automaton answering all queries using exactly this number of states. On the other hand, we can obviously use TPA to decide the equivalence of two queries and even determine equivalence classes.

From another point of view, we can examine the number of states of TPA by using our knowledge about smaller automata $M_i$ constructed for individual string paths. Assume $|M_i|_{max}$ is the maximum possible size of TPA for a single string path. Therefore, the number of states of TPA for the XML document can by trivially bounded by $\mathcal{O}(|M_i|^k_{max})$ (size of a product of $k$ automata with maximum of $\mathcal{O}(|M_i|_{max})$ states). However, this is the asymptotic upper bound. The size of the index seems to be according to our experimental results much smaller.

For a common XML document (XML with l-property), in which nodes with the same label can only appear at the same level of the XML tree model, the asymptotic upper bound is $\mathcal{O}(h \cdot 2^k)$.

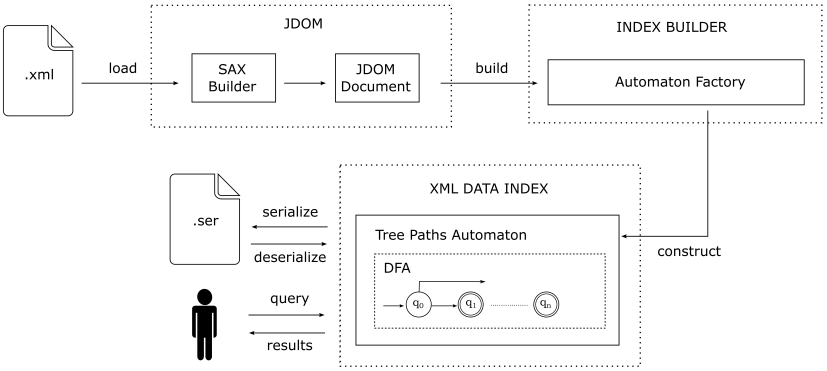



**Figure 7** System Architecture of `tpalib`.

▶ **Definition 9** (Level property). Let $T = (N, E)$ be a labelled directed rooted tree. Level property (l-property):

$$\forall n_1, n_2 \in N \wedge n_1 \neq n_2 : label(n_1) = label(n_2) \implies depth(n_1) = depth(n_2)\,.$$

▶ **Definition 10** (State level). Let $M = (Q, A, \delta, q_0, F)$ be an acyclic deterministic finite automaton. A state level $s$ of a state $q$ is a maximal number of transitions leading from the initial state $q_0$ to $q$.

▶ **Theorem 11.** *Let $D$ be an XML document and $T(D)$ be its XML tree model satisfying l-property with height $h$ and $k$ leaves. The number of states of the deterministic TPA constructed for the XML document $D$ by Algorithm 4 is $\mathcal{O}(h \cdot 2^k)$.*

**Proof.** There is $k$ string paths in $T(D)$, for which we construct a set $S$ of $k$ deterministic TPA of no more than $h$ states each (due to l-property). We can run all automata "in parallel", by remembering the states of all automata by constructing $k$-tuples $q$ while reading the input. This is achieved by the product construction. This way we construct the Tree Paths Automaton $M$ for $T(D)$.

Due to l-property of $T(D)$ it holds that: The target state of a transition labelled with $l$ is either a sink state or its state level is the same in each automaton in $S$. Hence, the $k$-tuples $(q_1, q_2, \ldots, q_k)$ are restricted as follows: If state level of $q_1$ is $s$, then each of $q_2, \ldots, q_k$ is either a sink state or of state level $s$. If $q_1$ is a sink state, then $q_2$ is arbitrary, but each of $q_3, \ldots, q_k$ is either a sink state or the same state level as $q_2$. In addition, the $k$-tuples of levels 0 and 1 are always $(0_1, 0_2, \ldots, 0_k)$ and $(1_1, 1_2, \ldots, 1_k)$, respectively. Therefore, the maximum number of states of $M$ is $2 + 2^{k-1} \cdot (h-1) + 2^{k-2} \cdot (h-2)$. ◀

## 7 Experimental Evaluation

This section explores the performance of Tree Paths Automaton. We first present TPA System Architecture. Then, we introduce the testing environment for our experiments and characteristics of selected XML data sets. Afterwards, we study space requirements of TPA and finally present a performance study over XPath queries that are supported by TPA.

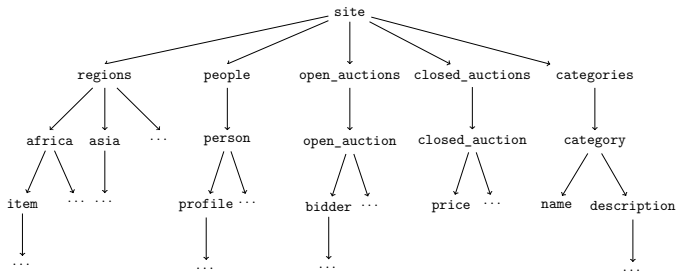



**Figure 8** Partial scheme of XMark data sets.

**Table 1** Characteristics of XMark benchmark files.

| Key | XML File | Xmark `xmlgen` Scaling Factor | # Elements | File size [MB] |
|---|---|---|---|---|
| $D_1$ | XMark-f0 | 0 | 382 | 0.03 |
| $D_2$ | XMark-f0.001 | 0.001 | 1,729 | 0.10 |
| $D_3$ | XMark-f0.005 | 0.005 | 8,518 | 0.60 |
| $D_4$ | XMark-f0.01 | 0.01 | 17,132 | 1.20 |
| $D_5$ | XMark-f0.5 | 0.5 | 832,911 | 58.00 |

### 7.0.1 System Architecture and Testing Environment

The XML index software was developed using Java SE, JDK 8u45 in the NetBeans IDE 8.0.2 and was designed as *Java Class Library* called `tpalib`. The system architecture of the `tpalib` is illustrated in Figure 7. The library consists of three virtual parts called JDOM, Index Builder and XML Data Index.

The experiments were conducted under the environment of Intel Core i7 CPU @ 2.00 GHz, 8.0 GB RAM and 240 GB SSD disk with Windows 8.1 operation system running.

## 7.1 XML Data Sets

For our experimental evaluation, we selected XML benchmark XMark data sets generated by `xmlgen` [15]. The XMark data set is a single record with a very large and fairly complicated tree structure with a maximal depth of 11 and average depth of 4.5. The XML data models an on-line auction site. Some of element relationships are illustrated in Figure 8.

Table 1 describes relevant characteristics of generated data sets. the first column defines data set keys. The second column shows names of generated XML files. The next column contains XMark `xmlgen` document scaling factors,[1] float values where 0 produces the "minimal document." The fourth column shows numbers of element nodes in generated files and, finally, the last column contains the size of files in megabytes.

## 7.2 Index Size and Performance on Query Processing

Table 2 shows the experimental results on the index size for generated XMark data sets. The space requirements of the index structure was measured using the size of the file with serialized `TreePathsAutomaton` Java object. The results suggest that the ratio of the index

[1] `http://www.xml-benchmark.org/faq.txt`

**Table 2** Experimental results on index size.

| Key | Index Size [MB] | Index Size / XML File Size |
|---|---|---|
| $D_1$ | 0.08 | 2.60 |
| $D_2$ | 0.30 | 3.00 |
| $D_3$ | 1.35 | 2.25 |
| $D_4$ | 2.68 | 2.23 |
| $D_5$ | 129.00 | 2.22 |

**Table 3** Set of queries used in performance analysis.

| Key | XPath Query |
|---|---|
| $Q_1$ | `/site/open_auctions` |
| $Q_2$ | `/site/people/person/name` |
| $Q_3$ | `/site/regions/europe/item/description/parlist/listitem/text/emph` |
| $Q_4$ | `//person//watch` |
| $Q_5$ | `//regions//mail//date` |
| $Q_6$ | `//site//regions//europe//description//listitem//text//emph` |
| $Q_7$ | `/site//open_auction` |
| $Q_8$ | `//people/person//watch` |
| $Q_9$ | `//regions/europe//item//parlist/listitem//text/emph` |

size to original XML data size stays linear since the second column shows that the size of TPA data is only about 2.5 times larger than the size of the original document size.

The analysis on performance of query processing was conducted in comparison with a well-known reference implementation called Saxon.[2] The Saxon package is a collection of Java tools for processing XML documents. One of the main components is an XPath processor accessible to applications via its supplied API. Our measurements reflect query processing time only. Hence, document loading cost and query parsing cost have been excluded from the measurements.

Table 2 lists 9 sample queries we used for the experiments. The queries are split into categories depending on the type of axis used: $Q_1$–$Q_3$ queries contain child axis only, $Q_4$–$Q_6$ include descendant-or-self axis only, and $Q_7$–$Q_9$ queries use combination of both axes. Numbers of elements satisfying individual queries in each of generated data sets are shown in Table 4.

Figure 9 summarizes the experimental results of Tree Paths Automaton and Saxon. The graph is plotted using logarithmic scale. The $x$-axis represents the data sets, while the $y$-axis shows the response time in milliseconds. We used light blue dashed lines to display Saxon results, whereas TPA score is depicted as dark blue solid lines.

As for Saxon, there appears to be a clear upward pattern in the query processing time with growing size of data sets. We can also see that queries $Q_1$–$Q_3$ that use only child axis are easier to evaluate than more complex queries including also descendant-or-self axis. However, TPA results remain stable with processing time around 1 to 3 milliseconds. That is since the searching phase of all elements satisfying the query depends only on the size of a query and does not depend on the size of a data set. Overall, the sample queries achieve better response time using our proposed indexing method.

[2] Available from `http://saxon.sourceforge.net/`

**Table 4** Number of elements satisfying queries in the generated data sets.

| | $Q_1$ | $Q_2$ | $Q_3$ | $Q_4$ | $Q_5$ | $Q_6$ | $Q_7$ | $Q_8$ | $Q_9$ |
|---|---|---|---|---|---|---|---|---|---|
| $D_1$ | 1 | 1 | 2 | 1 | 5 | 2 | 1 | 1 | 2 |
| $D_2$ | 1 | 25 | 2 | 50 | 20 | 4 | 12 | 50 | 2 |
| $D_3$ | 1 | 127 | 5 | 247 | 124 | 6 | 60 | 247 | 5 |
| $D_4$ | 1 | 255 | 17 | 488 | 205 | 50 | 120 | 488 | 43 |
| $D_5$ | 1 | 12,750 | 1,235 | 25,414 | 10,455 | 2,357 | 6,000 | 2,5414 | 2,099 |

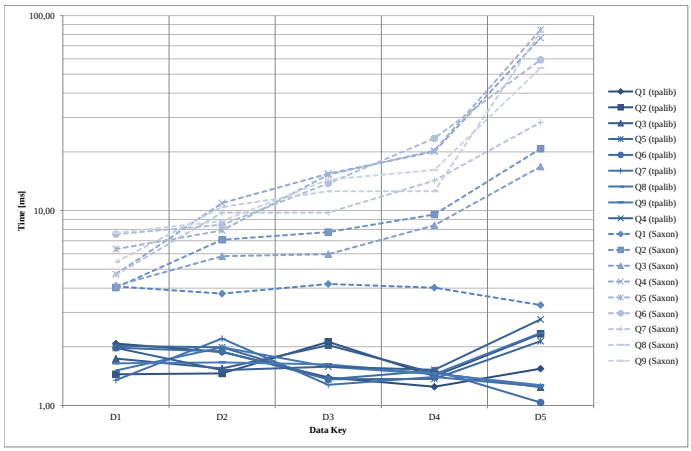



**Figure 9** Performance comparison of TPA and Saxon (logarithmic scale).

## 8 Conclusion and Future Work

A simple method for indexing XML documents using the theory of formal languages and automata was presented. Tree Paths Automaton is able to answer all queries which may use any combination of child (i.e., `/`) and descendant-or-self (i.e., `//`) axis, noted as $XP^{\{/,//,name-test\}}$.

Given an XML document $D$ with its corresponding XML tree model $T(D)$, the tree is preprocessed and an index, which is a finite automaton, is constructed. The searching phase uses the index, reads an input query $Q$ of size $m$, and computes the list of positions of all occurrences of $Q$ in the tree $T(D)$. The searching is performed in time $\mathcal{O}(m)$ and does not depend on the size of the original XML document.

Although the number of distinct queries is exponential in the number of nodes of the XML tree model, the size of the index seems to be according to our experimental results only about 2.5 times larger than the size of the original document. There is also a number of interesting open problems that we hope to explore in the future:

- develop an incremental building algorithm for our automata-based indexes to efficiently adapt their structure to ever changing XML data sources,
- adapt our indexing methods to be able to support multiple XML documents,
- extend our methods to support more complex queries (e.g., including attributes, wildcards, branching etc.).

## References


1. Chin-Wan Chung, Jun-Ki Min, and Kyuseok Shim. APEX: an adaptive path index for XML data. In *SIGMOD International Conference on Management of Data*, pages 121–132, 2002.
2. James Clark and Steve DeRose. *XML Path Language (XPath) Version 1.0*, 1999. URL: `http://www.w3.org/TR/xpath`.
3. Maxime Crochemore, Christophe Hancart, and Thierry Lecroq. *Algorithms on strings*. Cambridge University Press, 2007.
4. Maxime Crochemore and Wojciech Rytter. *Text Algorithms*. Oxford University Press, 1994.
5. Steve DeRose, Ron Daniel Jr., Paul Gross, Eve Maler, Jonathan Marsh, and Norman Walsh. *XML Pointer Language (XPointer)*, 2002. URL: `http://www.w3.org/TR/xptr`.
6. Steve DeRose, Eve Maler, and David Orchard. XML linking language (XLink) version 1.0. Technical report, World Wide Web Consortium, 2001. URL: `http://www.w3.org/TR/xlink`.
7. Roy Goldman and Jennifer Widom. DataGuides: Enabling query formulation and optimization in semistructured databases. In *23rd International Conference on Very Large Data Bases*, pages 436–445, 1997.
8. Raghav Kaushik, Philip Bohannon, Jeffrey F. Naughton, and Henry F. Korth. Covering indexes for branching path queries. In *SIGMOD International Conference on Management of Data*, pages 133–144, 2002.
9. Quanzhong Li and Bongki Moon. Indexing and querying XML data for regular path expressions. In *27th International Conference on Very Large Data Bases*, pages 361–370, 2001.
10. Bhushan Mandhani and Dan Suciu. Query caching and view selection for XML databases. In *31st International Conference on Very Large Data Bases*, pages 469–480, 2005.
11. Bořivoj Melichar, Jan Holub, and Tomáš Polcar. *Text Searching Algorithms*. Czech Technical University in Prague, 2005. Available at `http://www.stringology.org/athens/TextSearchingAlgorithms`.
12. Tova Milo and Dan Suciu. Index structures for path expressions. In Catriel Beeri and Peter Buneman, editors, *7th International Conference on Database Theory*, pages 277–295, 1999.
13. P. Mark Pettovello and Farshad Fotouhi. MTree: An XML XPath graph index. In *ACM Symposium on Applied Computing*, pages 474–481, 2006.
14. Praveen Rao and Bongki Moon. PRIX: indexing and querying XML using prufer sequences. In *20th International Conference on Data Engineering*, pages 288–299, March 2004.
15. Albrecht Schmidt. XMark: an XML benchmark project. `http://www.xml-benchmark.org/`.
16. Eliška Šestáková and Jan Janoušek. Tree string path subsequences automaton and its use for indexing xml documents. In *International Symposium on Languages, Applications and Technologies (SLATE)*, pages 171–181, 2015.
17. Nan Tang, Jeffrey Xu Yu, M. Tamer Ozsu, and Kam-Fai Wong. Hierarchical indexing approach to support XPath queries. In *IEEE 24th International Conference on Data Engineering*, pages 1510–1512, April 2008.
18. Haixun Wang, Sanghyun Park, Wei Fan, and Philip S. Yu. ViST: a dynamic index method for querying XML data by tree structures. In *SIGMOD International Conference on Management of Data*, pages 110–121, 2003.

# Enhancing Feedback to Students in Automated Diagram Assessment*

**Helder Correia[1], José Paulo Leal[2], and José Carlos Paiva[3]**

1 **CRACS & INESC-Porto LA, Faculty of Sciences, University of Porto, Porto, Portugal**
up201108850@fc.up.pt
2 **CRACS & INESC-Porto LA, Faculty of Sciences, University of Porto, Porto, Portugal**
zp@dcc.fc.up.pt
3 **CRACS & INESC-Porto LA, Faculty of Sciences, University of Porto, Porto, Portugal**
up201200272@fc.up.pt

## Abstract

Automated assessment is an essential part of eLearning. Although comparatively easy for multiple choice questions (MCQs), automated assessment is more challenging when exercises involve languages used in computer science. In this particular case, the assessment is more than just grading and must include feedback that leads to the improvement of the students' performance.

This paper presents ongoing work to develop Kora, an automated diagram assessment tool with enhanced feedback, targeted to the multiple diagrammatic languages used in computer science. Kora builds on the experience gained with previous research, namely: a diagram assessment tool to compute differences between graphs; an IDE inspired web learning environment for computer science languages; and an extensible web diagram editor.

Kora has several features to enhance feedback: it distinguishes syntactic and semantic errors, providing specialized feedback in each case; it provides progressive feedback disclosure, controlling the quality and quantity shown to each student after a submission; when possible, it integrates feedback within the diagram editor showing actual nodes and edges on the editor itself.

**1998 ACM Subject Classification** D.2.6 [Programming Environments] Interactive Environments

**Keywords and phrases** automated assessment, diagram assessment, feedback generation,language environments, e-learning

**Digital Object Identifier** 10.4230/OASIcs.SLATE.2017.11

## 1 Introduction

Automated assessment is essential for effective eLearning. Both in formative and summative assessment, eLearning students need to have their exercises compared with standard solutions, so that they know if they are achieving the expected result. Any form of assessment, even if it is just a grade, is already *feedback* to the student. However, feedback should be more than just a distance to the correct solution. Students need to be guided, see evidence of their mistakes and receive suggestions to improve their performance [8].

* This work is financed by the ERDF – European Regional Development Fund through the Operational Programme for Competitiveness and Internationalisation – COMPETE 2020 Programme, and by National Funds through the FCT – Fundação para a Ciência e a Tecnologia (Portuguese Foundation for Science and Technology) within project POCI-01-0145-FEDER-006961.





When the set of all possible answers to an exercise is small, grading and feedback are fairly easy to automate. This is arguably the reason why multiple choice questions (MCQs) are so popular on eLearning. In fact, simple skills and superficial knowledge can be completely assessed using MCQs, but for some cases they are insufficient. For instance, it is impossible to assess a student proficiency on a language using only MCQs. This is obviously true in natural languages, such as English and Portuguese, and is also the case for the artificial languages used in computer science. This fact lead to the development of several systems for assessing both programming languages [4] and diagrammatic languages [11, 2]. Nevertheless, feedback in these systems is still largely an open issue [5].

Diagram assessment has been less researched than program assessment, which is understandable since programs are more relevant than diagrams in computer science. However, programs are much more difficult to assess than diagrams, since their semantics is more complex. That is, a program has an operational semantics that needs to be checked with test data but diagrams have only a declarative semantics. Feedback on a diagram exercise can be solely based on the differences between the student's diagram and a solution. The relevance of the research on diagram assessment feedback is twofold: diagram languages are studied in several computer science disciplines, such as theory of computation (DFA), databases (EER) and software modeling (UML), thus it is useful for teaching those subjects; the tools and techniques developed for diagrammatic languages may later on be extended to more complex languages such as programming languages.

The research presented in this paper builds on previous work to develop the components for diagram assessment, namely a computer language learning environment [9], a diagram editor [6] and a a graph comparator [12]. Diagrams are modeled by graphs, hence it is possible to compare two diagrams by computing the differences between their model graphs. For large graphs, the computational complexity is prohibitive, but using heuristics and for graphs of the size typically used in programming exercises, this method is effective. However, the validation of the graph comparator revealed several issues related to feedback generation.

This paper presents Kora, a component for assessing diagrams with enhanced feedback. It was designed to support any diagrammatic language used in computer science. Nevertheless, some of its features were inspired by existing UML editors and it will validated with class and use case diagrams. Hence, Section 2 surveys in particular the existing editors and assessment systems for that language. Kora relies on components resulting from previous research that are described in Section 3. One the contribution of this work is the Diagrammatic Language Definition Language (DL2) that is presented in Section 4. The enhanced feedback features provided by Kora required a redesign of diagram editor Eshu, as explained in Section 5. The design and implementation of the Kora component is presented in Section 6. The final section addresses the work currently being done and the planned validation of Kora.

## 2 Related work

Kora supports the creation and assessment of diagram exercises of any type, with visual and textual feedback. To the best of authors' knowledge, no other tool described in the literature includes all these features. Hence, this section reviews several works including some of these features.

Most of the existent automatic diagram assessment systems are designed for a specific diagram type. Some examples of these system are deterministic finite automata (DFA) [2, 10], UML class diagrams [1, 11, 14], Entity-Relationship diagrams [3], among others.

There are many diagram editing tools targeted to UML and most of them are commercial.

Because they are developed for companies that normally use them for the modeling of complex systems, they present more features and functionalities (e.g forward engineering,reverse engineering). Most of these tools are visual tools defined for multi-domain modeling (e.g computational modeling) that support UML diagram modeling or drawing. Examples of this kind of tools are *MagicDraw*[1] and *Modelio*[2].

Nevertheless, there are a few non-commercial UML tools such as *ArgoUML*[3], and *Dia*[4]. These are usually developed by research groups with pedagogical scope, they usually have fewer features and functionality than commercial tools. However, in general, they are tools developed for the domain of modeling UML diagrams and present models that faithfully follow the specification UML.

A growing number of UML editing tools are deployed on the web, such as *Cacoo*[5]. These tools typically allow real-time multi-user collaboration in diagram editing, with specific features that facilitate this mode of editing (chat and version control). They are tools for drawing and not modeling, they have few features and functionalities and it is mandatory to have an account.

From the diagram assessment viewpoint, critiquing systems are a relevant feature of many UML editing and modeling tools. A critiquing system acts on modeling tools to provide corrections and suggestions on the models to be designed. Much research has been devoted to critiquing tools and they are incorporated in systems such as *ArgoUML*, *ArchStudio5*[6] *ABCDE-Critic* [13].

## 3 Background

In project Eshu [6], we develop an extensible diagram editor, embeddable in Web applications that require diagram interaction, such as modeling tools or e-learning environments. Eshu is a JavaScript library with an API that supports its integration with other components, including importing/exporting diagrams in JSON. In order to validate the API of Eshu we created an EER diagram editor in *Javascript* using the library provided by Eshu and HTML5 canvas. The editor allows to edit ERR diagram, import / export diagram into JSON format, apply ERR language restrictions in diagram editor (constraints on links), display visual feedback on EER diagram submissions. The editor has been integrated into the Enki [9] with a diagram evaluator and was used in database course to edit and evaluate EER diagrams.

Diagrams are schematic representations of information that, ignoring the positioning of its elements, can be abstracted in graphs. Based on this, structure driven approach to assess graph-based exercises was proposed [12]. Given two graphs, a solution and an attempt of a student, this approach computes a mapping between the node sets of both graphs that maximizes the students grade, as well as a description of the differences between the two graph. Then, it uses an algorithm with heuristics to test the most promising mappings first and prune the remaining when it is sure that a better mapping cannot be computed.

Enki [9] is a web-based IDE for learning programming languages, which blends assessment (exercises) and learning (multimedia and textual resources). It integrates with external

[1] https://www.nomagic.com/products/magicdraw.html
[2] https://www.altova.com/umodel.html
[3] http://argouml.tigris.org/
[4] http://dia-installer.de/
[5] https://cacoo.com
[6] https://basicarchstudiomanual.wordpress.com/

services to provide gamification features and to sequence educational resources at different rhythms according to students' capabilities. The assessment of exercises is provided by the new version of Mooshak [7] – Mooshak 2.0 –, which, among other features, allows the creation of special evaluators for different types of exercises.

## 4 Language configuration-DL2

Kora was designed to be extensible, to be able to incorporate new diagrammatic languages defined by an XML configuration file. This file includes configurations for syntactic feedback and editor. It configures types of nodes, types of edges, restrictions of the language, among others, that are used while validating the syntax. It includes configurations of the editor and toolbar style that applied on Eshu.

The configuration file consists of two top elements `Style` and `Diagram`. Type `Style` contains information such height, width, background and grid of the editor and the toolbar. Type `Diagram` configures the syntax of the language (nodes, edges and constraints), and has two types of elements, `nodeTypes` and `edgeTypes`, and two attributes, `name` and `pathFile`. The attribute `name` contains the name of the language, `pathFile` contains the path of the configuration file. Elements of *nodeTypes* contain a set of `nodeInfo` and each `nodeInfo` is a configuration of a node (type, svg image, label, URL for node type information, visible properties in the configuration window, type of connections of the node and degree in/out of this node). Elements of `edgeTypes`, similar to `nodeTypes`, contain a set of `edgeInfo`, and each `edgeInfo` contains the configuration of an edge type.

## 5 Eshu 2.0

A diagram is composed of a set of Node and a set Edge; Nodes have a position and dimension; Edges connect a source and a target node. Although Eshu 2.0, similarly to Eshu 1.0 [6], follows an object-oriented approach for *Javascript*, it separates the data part from the visualization and editing part.

Eshu 2.0 consists of three packages: `eshu`, `graph` and `commands`. The package `graph` has the classes responsible for creating nodes and edges, storing the graph (`Quadtree`) and operating on the data of the graph (insert, remove, save changes and select an element). Package `eshu` contains the classes responsible for the user interface, including handlers for user interaction, methods to export and import the graph of the diagram in JSON format, methods to present visual feedback in the diagram editor, among many others. The package `commands` contains the classes that are responsible for the implementation of operations, such as undo, redo, paste, remove or resize.

One of the main improvements of Eshu 2.0 is the extensibility of nodes and edges. In Eshu 1.0, the creation of a new type of node (or edge) involves the creation of a new class that extends `Vertice` (or `Edge` for edges) and defines the method `draw`. With Eshu 2.0, a new type of node (or edge) can be inserted by only adding a `nodeConfig` (or `edgeConfig`) element to `nodeTypes` (or `edgeTypes`), in the configuration file. This element contains general information for a node (or edge), such as its SVG image path (used to represent it in the UI), type name, constraints on connections, among others.

Eshu is a pure JavaScript library, hence it can be integrated in most web applications. However, some frameworks, such as Google Web Toolkit (GWT), use different languages to code the web interfaces, in this case Java. To enable the integration of Eshu in GWT applications, a binding to this framework was also developed. The binding is composed

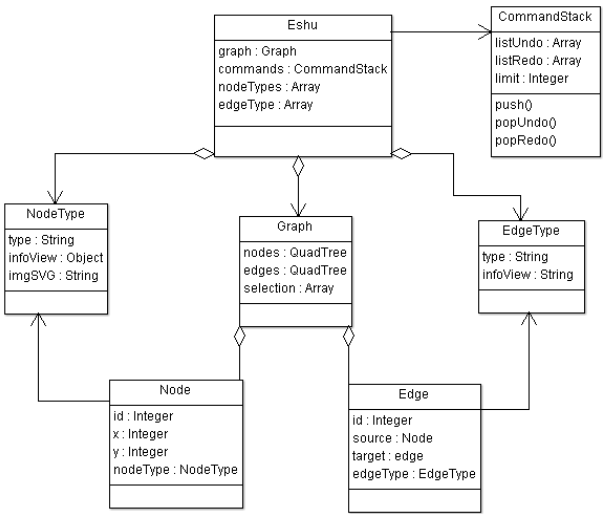



**Figure 1** Diagram Class Eshu.

of a Java class (that is converted to JavaScript by GWT) with methods to use the API, implemented using the JavaScript Native Interface (JSNI) of GWT.

The undo and redo commands are very important to the user while editing the graph. These two operations were not included in the first version of Eshu [6], but were now added. To facilitate the integration of these operations, a set of classes that implement the command design pattern were developed. Now, operations, such as insert, delete and paste, are encapsulated as an object allowing to register them in a stack, and thus pop or push them.

Also, the API allows the host application to send feedback in the form of changes to the existing diagram. If these changes are deletions or modifications, they can be rendered by displaying the existing nodes and edges with a different color (blue – insert, red – delete). However, if the difference is a node insertion then it has to be positioned by Eshu. The layout of these new nodes is computed using a force-directed algorithm. In this approach, nodes repel each other according to Coulomb's law, as if they were electrically charged particles with the same signal, and edges bind them together as springs following Hooke's law.

## 6 Kora component

The Kora component is divided into two parts, `client` and `server`. The `client` part is integrated on the web interface and is responsible for running the Eshu editor, as well as handling user actions and presenting the feedback. The `server` part is responsible for evaluating diagrams, generating feedback, and exchanging information with the client side, such as language configurations.

A diagram is a schematic representation of information. This representation has associated to itself elements that have certain characteristics and a positioning in the space. By abstracting the layout (the position of the elements), the diagrams can be represented as graphs. The approach that is intended to follow for the assessment of the diagrams is the comparison of the graphs. Thus, it is possible to analyze the contents of the diagram without giving relevance to its positioning or graphic formatting.

In Eshu 1.0, types of connections are checked during creation editing, that is, if a source and target nodes could not be connect it would be reported immediately. However, during

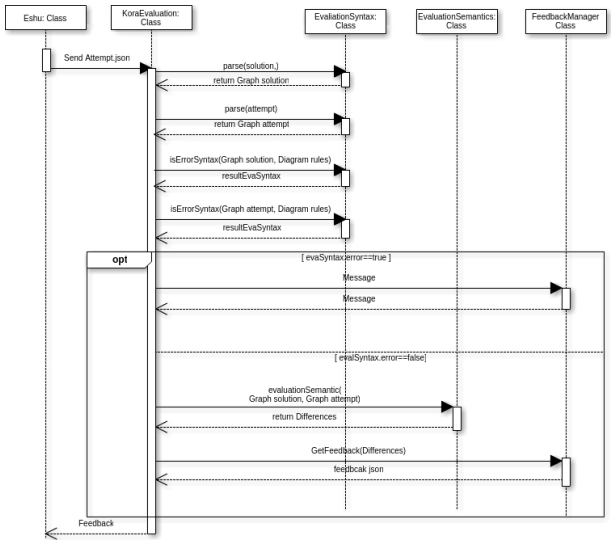



**Figure 2** Assessment diagram in Kora system.

the validation of Eshu 1.0, it was noticed that the editor was getting slower as the number of nodes increased, although not all syntactic issues were actually covered. Also, syntactically incorrect graphs were causing problems in the generation of feedback by the evaluator. Due to these issues, syntactic verification was moved to Kora.

The diagram assessment in the system is split in two parts: syntactic assessment and semantic assessment. The syntactic assessment involves the conversion of the JSON file to a graph structure, and validation of the language syntax. It consists of validating the structural organization of the language, based on the set of rules, defined in the configuration file, for the types of nodes and edges. In this phase, the following tasks are done: validation of the types for the language; validation of the edges – for each edge it is checked if the type, source and target are valid; validations of the nodes – the degree of in and out are valid; validation of the number of connected components in the graph. The semantic assessment has to do with the comparison of the diagrams and follows graph assessment algorithm [12]. The evaluator receives a graph as an attempt to solve a problem and compares it with a graph solution, aims to find out which mapping of the solution nodes in nodes attempt to minimize the set of differences and therefore maximize the classification. For this, it is necessary to find out which solution node corresponds to the attempt node.

Figure 2 presents the UML sequence diagram of the diagram assessment in the Kora system. The Kora client gets the graph of the diagram in JSON format through a function of the Eshu API – `Eshu exportGraph()`. It parses the JSON graphs of the solution and the attempt, gathering the information necessary, in the form of graph, to represent them in the next assessments of these diagrams. Then, the *Kora* system performs syntactic validation and reports the existence of any syntax error, aborting the evaluation if a syntax error exists. If it does not contain any syntax error, it proceeds to semantic evaluation. This evaluator receives two graphs, the attempted graph and a solution graph. In the wrong answers, the errors are located and inserted into the lists of differences. Based on this list, the respective feedback is generated and a classification is calculated, so that the diagram can be improved.

The semantic assessment provided by Kora is based on the differences computed by a graph evaluator. The graph evaluator compares two graphs (attempt and solution), returns a set of differences and based on these differences is generated a feedback that is presented

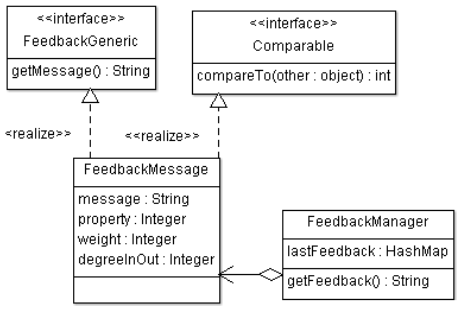



**Figure 3** Feedback Manager.

in Eshu, both in visual and textual form. However, when the student's attempt is far from the solution, it reports too many differences.

To cope with this problem Kore uses an incremental feedback generator. The generator uses several strategies to summarize a list of differences in a single message. The most general message that was not yet presented to the user is then selected as feedback.

Kora uses a repertoire of strategies to summarize a list of differences. Some strategies manage to condense several differences. For instance, several differences reporting a missing node of the same type may be condensed in the message "$n$ missing nodes of type $T$." Another strategy may select one of these nodes and show its label. An even more detailed strategy may show the actual missing node on the diagram. A particular strategy may not be applicable to some list of differences. In this case no message is produced.

The resulting collection of feedback messages is sorted according to generality. General messages have precedence over specific messages. However, if a message was already provided as feedback than it is not repeated. The following message is reported instead. Using this approach, messages of increasing detail are provided to the student if she or he persist on the same exact error.

Figure 3 presents the UML class diagram of the feedback implementation. The class `FeedbackMessage` contains the feedback information, including message, property number, weight, and in / out degrees (if it is a node). The property number indicates the property to which the message refers, the weight defines how much important is the mistake of the student, the degree of input/output allows to determine the importance of the node comparing to other nodes (i.e. higher degree, generally, means higher importance), and the message is the message itself. The class `FeedbackManager` generates and selects the feedback to be sent to the student. From the list of differences that is returned by the graph evaluator, it is generated a list of `FeedbackMessage`. From this list, the feedback already sent to the student is removed, and the remaining is sorted based on the fields of the `FeedbackMessage` class. The first `FeedbackMessage` from the list is selected and sent to the student.

## 7 Current and Future Work

Kora is work in progress. The project is in the final stage of development, just before validation. The design of Kora, including the diagrammatic language definition language, is already concluded. The implementation of the components described in this paper is in an advanced stage. Currently, most of the development effort is in the integration of Kora in Eshu, the learning environment where it will be deployed. In parallel, the definitions for two types of UML diagrams, namely class and use case, are also in development. Exercises for these diagram types will be used in the validation of Kora.

The research question that Kora aims to answer is: can feedback be enhanced by processing the output of an evaluator? Thus, the validation of the proposed approach will compare the efficacy of feedback with and without Kora. An experiment on the effect of Kora as a "treatment" to improve the efficacy of feedback is also being designed. Two groups of randomly chosen students will solve the same exercises, one group will receive raw feedback (just a grade and a list of differences) and the other will receive feedback processed by Kora. Both quantitative and qualitative differences are expected in the outcomes of the two groups. A number of variables will be measured to quantify those differences, including: the percentage of solved exercises, the number of submissions per problem and the time spent per exercise. To estimate the qualitative difference, students of both groups will be asked to fill in a questionnaire on their experience using diagram assessment tool.

## References


1. Noraida Haji Ali, Zarina Shukur, and Sufian Idris. A design of an assessment system for UML class diagram. In *International Conference on Computational Science and its Applications*, pages 539–546, 2007.
2. Rajeev Alur, Loris D'Antoni, Sumit Gulwani, Dileep Kini, and Mahesh Viswanathan. Automated grading of DFA constructions. In *International Joint Conference on Artificial Intelligence (IJCAI)*, volume 13, pages 1976–1982, 2013.
3. Firat Batmaz and Chris J. Hinde. A diagram drawing tool for semi-automatic assessment of conceptual database diagrams. In *10th CAA International Computer Assisted Assessment Conference*, pages 71–84, 2006.
4. Christopher Douce, David Livingstone, and James Orwell. Automatic test-based assessment of programming: A review. *Journal on Educational Resources in Computing (JERIC)*, 5(3):4, 2005.
5. Hieke Keuning, Johan Jeuring, and Bastiaan Heeren. Towards a systematic review of automated feedback generation for programming exercises. In *Conference on Innovation and Technology in Computer Science Education*, pages 41–46, 2016.
6. José Paulo Leal, Helder Correia, and José Carlos Paiva. Eshu: An extensible web editor for diagrammatic languages. In *5th Symposium on Languages, Applications and Technologies (SLATE)*, pages 12:1–12:13, 2016.
7. José Paulo Leal and Fernando Silva. Mooshak: A web-based multi-site programming contest system. *Software: Practice and Experience*, 33(6):567–581, 2003.
8. Robin Mason and Frank Rennie. *Elearning: The key concepts*. Routledge, 2006.
9. José Carlos Paiva, José Paulo Leal, and Ricardo Queirós. Enki: A pedagogical services aggregator for learning programming languages. In *Conference on Innovation and Technology in Computer Science Education*, pages 332–337, 2016.
10. Zarina Shukur and Nurul F. Mohamed. The design of ADAT: A tool for assessing automata-based assignments. *Journal of Computer Science*, 4(5):415, 2008.
11. Josep Soler, Imma Boada, Ferran Prados, Jordi Poch, and Ramon Fabregat. A web-based e-learning tool for UML class diagrams. In *Education Engineering Conference (EDUCON)*, pages 973–979, 2010.
12. Rúben Sousa and José Paulo Leal. A structural approach to assess graph-based exercises. In *International Symposium on Languages, Applications and Technologies (SLATE)*, pages 182–193, 2015.
13. Cleidson R. B. Souza, J. S. Ferreira, Kléder Miranda Gonçalves, and Jacques Wainer. A group critic system for object-oriented analysis and design. In *The Fifteenth International Conference on Automated Software Engineering (ASE)*, pages 313–316, 2000.
14. Vinay Vachharajani and Jyoti Pareek. A proposed architecture for automated assessment of use case diagrams. *International Journal of Computer Applications*, 108(4), 2014.

# A REST Service for Poetry Generation

**Hugo Gonçalo Oliveira**

**CISUC, Department of Informatics Engineering, University of Coimbra, Coimbra, Portugal**
hroliv@dei.uc.pt

## Abstract

This paper describes a REST API developed on the top of PoeTryMe, a poetry generation platform. This API exposes several functionalities, from the production of full poems, to narrower tasks, having in mind their utility for poetry composition, including the acquisition of well-formed lines, or semantically-related words, possibly constrained by the number of syllables, rhyme, or polarity. Examples that illustrate the endpoints and what they can be used for are also revealed.

**1998 ACM Subject Classification** I.2.7 Natural Language Processing

**Keywords and phrases** REST, creative web services, poetry generation, computational creativity

**Digital Object Identifier** 10.4230/OASIcs.SLATE.2017.12

## 1 Introduction

The topic of poetry generation, popular among the research community of Computational Creativity [3], is a kind of knowledge-intensive natural language generation that deals with several levels of language (e.g. lexical choice, syntax, semantics) as well as with formal features (e.g. metre and rhyme), towards the production of aesthetically-pleasing text, with a creative value. PoeTryMe [6, 7, 10] is one of many poetry generation systems reported in the literature (e.g. [4, 13, 15, 14, 19]), with the particularity of being focused on semantic relations and, more relevant for this work, having a modular architecture. The latter enabled its instantiation for producing poetry in different languages [10] and forms, including song lyrics [8], following different strategies [7], and from different stimuli, usually a list of seed words, but also Twitter trends [9] or concept maps extracted from text [11].

Though not designed with this specific purpose, PoeTryMe's architecture is friendly towards the vision of a Creative Web, where creative applications are deployed as web services [16], and in line with Gervás's [5] view on the deconstruction of poetry generation systems. Therefore, we decided to embrace this vision and develop a REST API for PoeTryMe, thus enabling the interaction of third-party applications, such as poetry-composition support tools, with this system.

This paper describes the state of this API, which currently enables the autonomous generation of full poems; the acquisition of well-formed lines, or semantically-related words, possibly constrained by the number of syllables, rhyme, or polarity; and to compute a score for the metre of a piece of text, according to a given poetry form. It starts with a brief reference to related work, followed by a contextualization of PoeTryMe and its architecture. After this, some details are presented about the API's implementation, followed by an enumeration of the available endpoints, together with some illustrative examples.


## 2 Related Work

Veale [16] describes the vision of creativity as set of web of services, a vision that promotes interoperability between different services that can be exploited and combined by different researchers for different purposes or for the creation of new applications, also saving development time, because the services are ready to use. Following this vision, several web services for metaphor generation were developed and used by different creative systems [17].

This vision is further materialised by creativity platforms that allow users to combine different creative or creativity-support services in the development of novel creative workflows that can be tested right away. These platforms include a wide range of tools, available as services, in a web interface that enables the selection and connection of services through drag-and-drop, setting the initial parameters and enabling the inspection of the state of each service. Produced workflows may include applications from different authors and can be easily shared with other users. ConCreTeFlows [18] and FloWr [2] are notable examples of such platforms, which have, among many other tasks, been used for poetry generation. The former includes a PoeTryMe service that uses one of the endpoints described in this paper for producing poems, given a small set of parameters. This has been used for generating poems inspired by conceptual blends. As for FloWr, it has been used for producing poetry by the recombination of lines extracted from Twitter, towards predefined poetic features [1].

Specifically speaking of poetry and web services, Gervás's [5] discusses the deconstruction of poetry generation systems and argues that abstractions of the various functionalities involved in such a system should be available as services that may be later invoked by other systems. Even if inspired by different stimuli or following different generation procedures, it makes sense that poetry generators share some of their modules. The most obvious would be to share the module for metric scansion, but other modules would be useful for different systems. PoeTryMe's REST API takes advantage of PoeTryMe's's modular architecture and embraces the aforementioned vision.

## 3 PoeTryMe's modular architecture

PoeTryMe has a modular architecture, explained with detail elsewhere [10], with several independent modules that might be combined for poetry generation. Those include two core modules – a Generation Strategy and a Lines Generator – and some complementary ones.

The Generation Strategy implements a plan for producing poems according to user-given parameters. It can have different implementations and interact with the Syllable Utils to perform syllable-related operations, such as syllable division or rhyme identification.

The Lines Generator interacts with the Relations Manager, Morphology Handler and Grammar Processor for producing semantically-coherent fragments of text, to be used as lines of a poem. The Relations Manager and the Grammar Processor are interfaces to a semantic network and to a context-free grammar, respectively. The former may cover any kind of labelled relation, and the latter has rules for rendering relations as text, given the relation label. The Lines Generator may also resort to a Contextualizer for explaining the selection of words and lines through lists of relations and the connection of their arguments to the initial parameters.

There is also a module for expanding a set of provided words with structurally-relevant words, possibly constrained by a target polarity (positive or negative). Poetry can be generated in Portuguese, Spanish or English, depending on the underlying linguistic resources, namely the semantic network, the lexicons and the grammars.

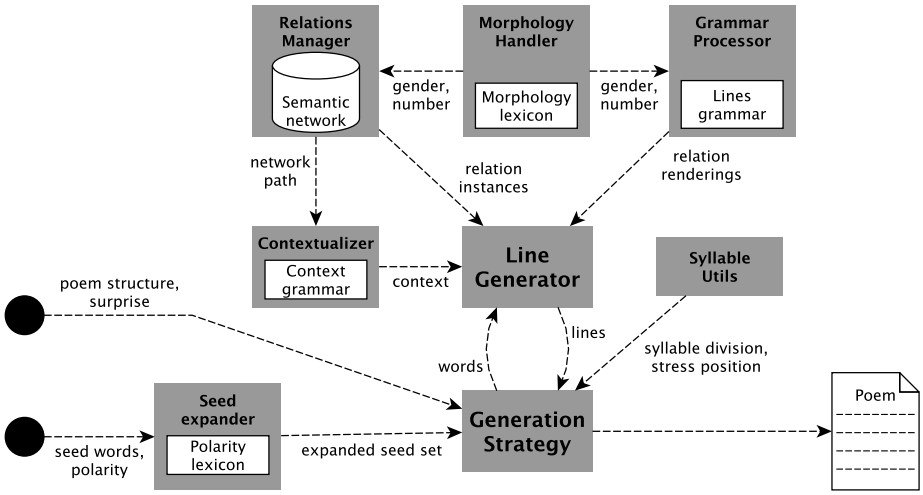



**Figure 1** PoeTryMe's modular architecture.

Figure 1 depicts PoeTryMe's architecture with dashed lines representing the flow involved in the autonomous generation of a poem. Initially, the user provides a set of parameters, including the structure of the poem, the set of seed words, the target polarity and a surprise factor. The structure file indicates the number of lines, syllables per line and rhyme scheme to follow. The seeds constrain the Relations Manager to use only these words, directly-related and some indirectly-related words, depending on the surprise. A higher surprise factor will increase the number of indirectly-related words. The set of seeds can be further augmented with the Seed Expander.

In addition to the presented flow, this modular architecture enables different combinations of the available modules, which work independently of each other. In other words, different poetry generation systems can arise from a different combination of PoeTryMe's modules, or existing computational applications can be augmented by exploiting only one or two modules. This was our main motivation to expose the access to some of these modules, through a REST API. Details about this API, with a focus on the available endpoints, are described in the following section.

## 4 API: implementation and endpoints

PoeTryMe is implemented in Java, so it made sense to implement its REST API also in this language. To ease the process, the Jersey RESTful Web Services framework[1], an open-source, standard and portable JAX-RS API, was used. The service runs in a Tomcat server, installed in `http://poetryme.dei.uc.pt:8080`, and has several HTTP endpoints, all returning HTTP responses with JSON objects. This section describes the endpoints defined for this service, together with usage examples. Endpoints were created with PoeTryMe's architecture in mind, as well as their utility for poetry composition support tools.

Full poems can be generated from the following endpoint:

- `http://poetryme.dei.uc.pt:8080/PoetrymeWeb/rest/poetry`

[1] Check `https://jersey.java.net/`.

- **Request:**
  `http://poetryme.dei.uc.pt:8080/PoetrymeWeb/rest/poetry?lang=en&form=10-4&seeds=computer+language&surp=0.002`
- **Response:**


```
{"form":"quadra_10.est","language":"en","score":"-2.0",
"seeds":["computer","language"],"surprise":"0.002",
"text":"computer slots have many other names\n
  the focus accents swaying in the wind\n
  the packs of heaven content over me\n
  resurrect my vocal through potpourri\n\n"}
```


**Figure 2** Using the API for the generation of a full poem.

Internally, this endpoint triggers the workflow of Figure 1, without a target polarity or expansion, because the latter could overload the server. It thus supports the following set of parameters:

- Language: `lang=[en|pt|es]`
- Form: `form=[id of the form]` (currently limited to a pre-defined list that includes, e.g. `10-2` for 10-syllable couplets, `10-4` for 10-syllable blocks-of-four, or `sonnet` for sonnets, and also some children and pop songs.)
- Seeds: `words=[comma-separated list of words]`
- Surprise: `surp=[0-1]`

Figure 2 illustrates how the previous endpoint can be used for generating a block of four lines in English, using the seeds *computer* and *language*, with a surprise of 0.002. The response is a JSON object with the properties form (internal file with that represents the target form), language (the language of the poem), score (an internal score based on the metre), seeds (an array with the seed words), surprise (the surprise factor), and text (the resulting poem).

Besides the previous, there are currently three other endpoints: one focused on single lines; another on words; and one for scoring the metre of a piece of text, according to a target poetry form. The following endpoint can be used for generating a single line, given a list of seeds and a surprise factor, selected from the best $n$ generated lines, according to the target number of syllables:

- `http://poetryme.dei.uc.pt:8080/PoetrymeWeb/rest/poetry/line`
  - Language: `lang=[en|pt|es]`
  - Seeds: `seeds=[comma-separated list of words]`
  - Surprise: `surp=[0-1]`
  - Target number of syllables: `nsyl=[1-n]`
  - Generations: `bestof=[1-n]`

Figure 3 illustrates this endpoint with an example in Portuguese and another in Spanish.

Words related to a target word can be retrieved with the following endpoint. The relation can be semantic, same number of syllables, same rhyme, or a combination of the previous. Resulting words might be further constrained with a target polarity[2]:

- `http://poetryme.dei.uc.pt:8080/PoetrymeWeb/rest/poetry/words`
  - Language: `lang=[en|pt|es]`
  - Word: `word=[word]`

[2] Check Gonçalo Oliveira et al. [10] for a reference of the current underlying resources.

- **Request 1:**
  `http://poetryme.dei.uc.pt:8080/PoetrymeWeb/rest/poetry/line?lang=pt&nsyl=10&seeds=criatividade&bestof=10`
- **Response 1:**
  `{"form":"1-liner","language":"pt","score":"3.0","seeds":"criatividade", "surprise":"0.0","text":"aptidões mortas , criatividade escura"}`
- **Request 2:**
  `http://poetryme.dei.uc.pt:8080/PoetrymeWeb/rest/poetry/line?lang=es&nsyl=7&seeds=crear&bestof=100`
- **Response 2:**
  `{"form":"1-liner","language":"es","score":"0.0","seeds":"crear", "surprise":"0.0", "text":"mi dar no quiere crear"}`

**Figure 3** Using the API for the generation of single lines.

- Relation: `rel=[id for relation type]` (supported types are: `synon` for synonymy, `anton` for antonymy, `hyper` for hypernym, `hypon` for hyponymy, `cohyp` for co-hyponymy, `other` for any other type, or `any` for any type)
- Rhyme: `syl=[word with the target rhyme]`
- Target number of syllables: `syl=[word with the target number of syllables]`
- Polarity: `pol=[-1,0,1]`

Figure 4 illustrates this endpoint with four examples: the first retrieves English words that rhyme with *state* and have a negative polarity; the second retrieves English words related to *creativity* that rhyme with *nation*; the third example retrieves Portuguese words that are hyponyms of *pessoa* and rhyme with *gerar*; the final example retrieves Portuguese words that are synonyms of *plano* and have the same number of syllables as *amigo*.

The remaining endpoint scores the metre of a piece of text according to a poetry form:

- `http://poetryme.dei.uc.pt:8080/PoetrymeWeb/rest/poetry/score`
  - Language: `lang=[en|pt|es]`
  - Form: `form=[id of the form]` (currently limited to a small set of forms)
  - Text: `text=[full text]` (lines split with a '`\n`')

The score is computed the same way as for the automatically-generated poems. More precisely, score is the sum of the absolute difference between the target number of syllables each line should have and the actual number. On the top of this, a bonus of $-2$ is given for every pair of rhyming lines, meaning that, the lower the score, the better the metre is matched. Figure 5 illustrates this endpoint when scoring text as 10-syllable couplets. In the three examples, couplets are given, yet, in the first, both lines rhyme; in the second they do not, but the number of syllables is still matched; while in the third the second line has only eight syllables.

## 5 Concluding remarks

A REST service was presented for autonomous poetry generation, with additional features that might be useful for poetry composition support tools. It is built on the top of PoeTryMe, a poetry generation platform with a modular architecture, and besides the autonomous generation of full poems, it enables the acquisition of well-formed lines, or semantically-related words, possibly constrained by the number of syllables, rhyme, or polarity, and the computation of a score for the metre of a piece text, according to a target poetry form.

- **Request 1:**
  http://poetryme.dei.uc.pt:8080/PoetrymeWeb/rest/poetry/words/?lang=en&rhyme=state&pol=-1
- **Response 1:**

```
{"language":"en","polarity":"-1","relation":"N/A",
"rhymesWith":"state","word":"state",
"related":["desecrate","bait","irate","negate","grate","hate",
  "berate","overweight"]}
```

- **Request 2:**
  http://poetryme.dei.uc.pt:8080/PoetrymeWeb/rest/poetry/words/?lang=en&word=creativity&rel=any&rhyme=nation
- **Response 2:**

```
{"language":"en","polarity":"N/A","relation":"any",
"rhymesWith":"nation","word":"creativity",
"related":["innovation","imagination"]}
```

- **Request 3:**
  http://poetryme.dei.uc.pt:8080/PoetrymeWeb/rest/poetry/words/?lang=pt&word=pessoa&rhyme=gerar&rel=hypon
- **Response 3:**

```
{"language":"pt","polarity":"N/A","relation":"hypon",
"rhymesWith":"gerar","word":"pessoa",
"related":["par","secular","militar","familiar","titular"]}
```

- **Request 4:**
  http://poetryme.dei.uc.pt:8080/PoetrymeWeb/rest/poetry/words/?lang=pt&word=plano&rel=synon&syl=amigo
- **Response 4:**

```
{"language":"pt","polarity":"N/A","relation":"synon",
"syllablesMatch":"amigo","word":"plano",
"related":["prospecto","traçado","intento","sistema","esquema",
  "macio","programa","direito","desenho","projecto"]}
```

**Figure 4** Using the API for the retrieval of words.

This API is still in development and, in the future, we should deal with minor internal issues and will consider the inclusion of additional functionalities, such as the contextualization of given lines, or the production and scoring of poems according to any given metre, not limited to the set of covered forms. For the latter purpose, we would use the internal representation for poetry forms, which includes the number of lines of a poem, their metre, and, optionally, a rhyming scheme. Yet, this representation is not URL friendly, so a simpler but equivalent format will have to be designed.

The poetry generation module of this API is currently integrated in ConCreTeFlows [18], and may thus be included in any workflow of this platform. Furthermore, Co-PoeTryMe[3], a web application built on the top of this API is currently being developed. Co-PoeTryMe can be seen as a co-creative application, like others of such kind targeting poetry generation [12]. It enables the collaboration between the human creator and PoeTryMe in the composition of poetry, hopefully better than poems autonomously produced by PoeTryMe, or more in line with the user intents. In the future, it should also be used by the Twitter bot *@poetartificial*, which is currently based on an independent instantiation of PoeTryMe.

[3] Check `http://poetryme.dei.uc.pt/~copoetryme/`.

- **Request 1:**
  `http://poetryme.dei.uc.pt:8080/PoetrymeWeb/rest/poetry/score?lang=en&form=10-2&text=Be%20not%20the%20first%20by%20whom%20the%20new%20are%20tried%2C%0ANor%20yet%20the%20last%20to%20lay%20the%20old%20aside.`
- **Response 1:**

```
{"form":"10-2","language":"en","score":"-2.0",
"text":"Be not the first by whom the new are tried,\n
  Nor yet the last to lay the old aside."}
```

- **Request 2:**
  `http://poetryme.dei.uc.pt:8080/PoetrymeWeb/rest/poetry/score?lang=en&form=10-2&text=Be%20not%20the%20first%20by%20whom%20the%20new%20are%20tried%2C%0ANor%20yet%20the%20last%20to%20lay%20the%20old%20aside.`
- **Response 2:**

```
{"form":"10-2","language":"en","score":"0.0",
"text":"Be not the first by whom the new are tried,\n
  Nor yet the last to lay the old behind."}
```

- **Request 3:**
  `http://poetryme.dei.uc.pt:8080/PoetrymeWeb/rest/poetry/score?lang=en&form=10-2&text=Be%20not%20the%20first%20by%20whom%20the%20new%20are%20tried%2C%0AThis%20line%20has%20eight%20metrical%20syllables`
- **Response 3:**

```
{"form":"10-2","language":"en","score":"2.0",
"text":"Be not the first by whom the new are tried,\n
  This line has eight metrical syllables"}
```

**Figure 5** Using the API for scoring the metre of couplets.

## References


1 John Charnley, Simon Colton, and Maria Teresa Llano. The FloWr framework: Automated flowchart construction, optimisation and alteration for creative systems. In *International Conference on Computational Creativity (ICCC)*, pages 315–323, June 2014.

2 John Charnley, Simon Colton, Maria Teresa Llano, and Joseph Corneli. The FloWr online platform: Automated programming and computational creativity as a service. In *7th International Conference on Computational Creativity (ICCC)*, pages 363–370, 2016.

3 Simon Colton and Geraint A. Wiggins. Computational creativity: The final frontier? In *20th European Conference on Artificial Intelligence (ECAI)*, pages 21–26, 2012.

4 Pablo Gervás. WASP: Evaluation of different strategies for the automatic generation of spanish verse. In *AISB'00 Symposium on Creative & Cultural Aspects and Applications of AI & Cognitive Science*, pages 93–100, 2000.

5 Pablo Gervás. Deconstructing computer poets: Making selected processes available as services. *Computational Intelligence*, 33(1):3–31, 2017.

6 Hugo Gonçalo Oliveira. PoeTryMe: a versatile platform for poetry generation. In *Proceedings of the ECAI 2012 Workshop on Computational Creativity, Concept Invention, and General Intelligence*, C3GI 2012, Montpellier, France, August 2012.

7 Hugo Gonçalo Oliveira and Amílcar Cardoso. Poetry generation with PoeTryMe. In T. R. Besold, M. Schorlemmer, and A. Smaill, editors, *Computational Creativity Research: Towards Creative Machines*, chapter 12, pages 243–266. Atlantis-Springer, 2015.

8 Hugo Gonçalo Oliveira. Tra-la-Lyrics 2.0: Automatic generation of song lyrics on a semantic domain. *Journal of Artificial General Intelligence*, 6(1):87–110, 2015. Special Issue: Computational Creativity, Concept Invention, and General Intelligence.

9 Hugo Gonçalo Oliveira. Automatic generation of poetry inspired by Twitter trends. In *Knowledge Discovery, Knowledge Engineering and Knowledge Management*, volume 631 of *CCIS*, pages 13–27. Springer, 2016.

10 Hugo Gonçalo Oliveira, Raquel Hervás, Alberto Díaz, and Pablo Gervás. *Multilanguage Extension and Evaluation of a Poetry Generator*, 2017. (in press).

11 Hugo Gonçalo Oliveira and Ana Oliveira Alves. Poetry from concept maps: Yet another adaptation of PoeTryMe's flexible architecture. In *7th International Conference on Computational Creativity*, 2016.

12 Anna Kantosalo, Jukka M. Toivanen, Ping Xiao, and Hannu Toivonen. From isolation to involvement: Adapting machine creativity software to support human-computer co-creation. In *5th International Conference on Computational Creativity*, 2014.

13 H. M. Manurung. *An evolutionary algorithm approach to poetry generation.* PhD thesis, University of Edimburgh, Edimburgh, UK, 2003.

14 Joanna Misztal and Bipin Indurkhya. Poetry generation system with an emotional personality. In *5th International Conference on Computational Creativity (ICCC)*, 2014.

15 Jukka M. Toivanen, Matti Järvisalo, and Hannu Toivonen. Harnessing constraint programming for poetry composition. In *4th International Conference on Computational Creativity (ICCC)*, pages 160–167, 2013.

16 Tony Veale. Creativity as a web service: A vision of human and computer creativity in the web era. In *Creativity and (Early) Cognitive Development: A Perspective from Artificial Creativity, Developmental AI, and Robotics*, pages 73–78. AAAI Press, 2013.

17 Tony Veale. A service-oriented architecture for metaphor processing. In *Second Workshop on Metaphor in NLP*, pages 52–60. ACL Press, 2014.

18 Martin Žnidaršič, Amílcar Cardoso, Pablo Gervás, Pedro Martins, Raquel Hervás, Ana Oliveira Alves, Hugo Gonçalo Oliveira, Ping Xiao, Simo Linkola, Hannu Toivonen, Janez Kranjc, and Nada Lavrac. Computational creativity infrastructure for online software composition: A conceptual blending use case. In *7th International Conference on Computational Creativity (ICCC)*, Paris, France, 2016.

19 Rui Yan. i, Poet: Automatic poetry composition through recurrent neural networks with iterative polishing schema. In *25th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 2238–2244, 2016. URL: `http://www.ijcai.org/Abstract/16/319`.

# SOS – Simple Orchestration of Services

**Ricardo Queirós[1] and Alberto Simões[2]**

1 **ESMAD, Polytechnic of Porto, Porto, Portugal**
ricardoqueiros@esmad.ipp.pt
2 **Centro Algoritmi, University of Minho, Braga, Portugal; and Instituto Politécnico do Cávado e do Ave, Barcelos, Portugal**
asimoes@ipca.pt

## Abstract

Nowadays, we continue to write redundant code which can often be reused from the Web. Reusing programming tasks is beneficial since it speeds up the process of creating applications and reduces the errors related with the task creation from scratch. At the same time, the demands of our applications are increasing, leading to a simple problem having to be solved through several tasks. With the advent of the cloud, there are countless Web services that proliferate on the Web. One solution for developers is to use these Web Services. However, the process of mastering and coordinating all these services manually is time-consuming and error-prone.

This paper presents SOS, a Simple Orchestration of Services. The ultimate goal of this tool is to act as a service composer while promoting the separation of concerns for two typical actors in this realm: the developer and the business analyst. The developer must define a service as a SOS task based on a JSON schema and submit it in a Web specialized editor. The business analyst uses the SOS editor, in an interactive way, to chain the required tasks to solve a specific problem. Then, the developer, uses a a simple client API – a SOS engine wrapper – to load a SOS manifest and to iterate over all tasks, without the need to dominate any bureaucratic aspects related with HTTP clients and messages. As a case study, several tasks are instantiated and aggregated in order to generate a composite service for a mobile app whose goal is to give an translated description of a picture taken with a mobile phone.

**1998 ACM Subject Classification** H.3.5 [Online Information Services] Web-based Services

**Keywords and phrases** Web services, Service Composition, Orchestration

**Digital Object Identifier** 10.4230/OASIcs.SLATE.2017.13

## 1 Introduction

Service Oriented Architecture (SOA) emerged in the early 2000s, offering a way to develop new business services by reusing components from existing programs rather than writing redundant code from scratch and developing new infrastructures to support them [3].

SOA can be defined as an approach to develop enterprise systems by loosely coupling interoperable services – small units of software that perform discrete tasks – from separate systems across different business domains [2].

A crucial aspect of SOA is service composition. Service composition is the process of create a composite service using a set of available Web services in order to satisfy a user request or a problem that cannot be satisfied by any individual Web service [2]. The service composition can be defined from a global perspective (choreography) or using a central component which coordinates the entire process (orchestration).

However, despite all the software that implements those concepts, there are few that can be used, in a very simple way, and focused on the new paradigm of cloud services, where the


JSON specification increasingly assumes a prominent role in the data exchange formalization within the client-server model.

In this paper we present SOS – Simple Orchestration of Services – as a tool for services composition. The tool is composed by two components: a Web editor and a client engine. The former allows the submission, aggregation, sharing and grading of small services defined as SOS tasks. The later, allows developers to iterate over all the service tasks, through a simple API.

The main advantage of this approach regarding the existent approaches is the simplicity and the separation of concerns. Firstly, a SOS task is formalized with a simple JSON schema. The aggregation is materialized on a JSON manifest that can be loaded from the cloud. Then, developers through a simple API can manage the execution flow of the process without worrying about HTTP clients and messages. Secondly, the SOS network fosters the separation of concerns by giving to the developer the mission to formalize and submit tasks and interact with the engine and to the business analysts the chance to compose the tasks for the creation of a single service. The chaining process will be very simple, using drag-and-drop techniques, and will automatically inform of invalid pairings based on the matching of the tasks' response/request types.

As a case study we present a composite service that receives a photo taken by a smartphone and returns a translation definition of the photographed object.

## 2 Services Composition

Service composition encourages the design and aggregation of services that can be reused in multiple scenarios. In this realm, two techniques are used: orchestration and choreography.

### 2.1 Orchestration vs. Choreography

Web service orchestration is a type of service composition where specific web service business processes are controlled by a central component. This component coordinates asynchronous interactions, flow control and business transaction management [3]. Typically, Business process modelling notation[1] (BPMN), maintained by the Object Management Group, is used to define a visual representation of the flow and business process execution language (BPEL) is used to write the code that executes the services.

Service choreography is a form of service composition in which the interaction protocol between several partner services is defined from a global perspective. This could be mapped for the dance domain, where "dancers dance following a global scenario without a single point of control" [4]. In other words, at run-time each participant in a service choreography executes its role according to the behaviour of the other participants [2]. Several languages specifications appeared to model service choreographies: the Web Service Choreography Description Language[2] (WS-CDL) and the Web Service Choreography Interface[3] (WSCI). Both are XML-based specifications from the W3C for modelling choreographies. The BPMN version 2.0 includes diagrams to model service choreographies. Other academic proposals for service choreography languages include: Let's Dance [5], BPEL4Chor [1] and Chor[4].

[1] Link: `http://www.bpmn.org/`.
[2] Link: `https://www.w3.org/TR/ws-cdl-10/`.
[3] Link: `https://www.w3.org/TR/wsci/`.
[4] Link: `http://www.chor-lang.org/`.

A distinction is often made between orchestration (a local view from the perspective of one participant) and choreography (coordination from a global multi-participant perspective, without a central controller). Although, the service orchestration is the most used and plays an important part in a service-oriented architecture (SOA), the web service choreography, is also often used to address the typical issues of single point-of-failure that are often found using the orchestration paradigm [3].

## 2.2 Related work

The SOS tool does not intend to compete with similar tools. If we want to catalog the proposed tool, it can be defined as an automation tool made up of small services formalized through light specifications such as JSON. In the field of service automation tools several tools appeared in recent years. The best examples are IFTTT[5], Zapier[6] and Microsoft Flow[7]. IFTTT is a free web-based service that people use to create chains of simple conditional statements, called applets. Zapier connects existent apps for the automation of workflows. Lastly, Microsoft Flow is an automated actions service. All of them are more focused on tasks automation and are not focused on the composition and execution flow management of HTTP REST services.

# 3 SOS – Simple Orchestration of Services

This paper presents SOS as a simple system to help developers and business analysts to submit services as tasks and chain them in order to create composite services. The architecture of SOS is straightforward and is composed by the following components:

- The editor – Web-based component with a GUI for the submission, chaining and generation of composite services.
- The engine – client component responsible for the orchestration of the composite service.

## 3.1 The editor

The editor is a Web-based component for the submission of tasks and their aggregation in order to obtain a composite service. The next section will detail the main aspects of the editor, more precisely, the GUI component, the task schema and the service manifest.

### 3.1.1 The GUI component

The SOS editor is a Web-based component, based on HTML5 Canvas and D3.js[8], that will help users in the submission and aggregation of tasks. The final result is a new composite service as a SOS manifest that can be stored in the cloud or saved in the user's computer. A user can perform the following operations in the editor:

- Submit a new task.
- Create a new composite service (by aggregating tasks).
- Share and grade tasks and services.

[5] Link: `https://ifttt.com/discover`.
[6] Link: `https://zapier.com/`.
[7] Link: `https://flow.microsoft.com/pt-pt/`.
[8] Link: `https://d3js.org/`.

After the registration and, before any submission, developers can search for desirable tasks using tags. For instance, if we want to use a service for weather consumption, we insert the tag "weather" and the editor shows all the tasks with the related tag. If no services are returned, the developer can submit a new task. The submission of a task can be performed interacting with the GUI component or by submitting a valid JSON document.

After the selection/submission of a task, business analysts can aggregate new ones in order to compose a service. The aggregation of two tasks (for instance, Task A and Task B) is only possible if the type of the response of Task A complies with one of the types of the request of Task B. Outside of this rule are the initial and the final tasks. In the end, we will have a composite service as a SOS service manifest that can be stored in the private cloud of the user or downloaded for its computer.

Other feature of the editor is the capacity for sharing and grading tasks. This feature will allow a user to share a previously created tasks in the public space of the SOS community. With the grade feature, developers could score a given task taken into account the experience that they have with it. This grading will influence the results list after searching.

### 3.1.2 The Task schema

Developers should create a task as a JSON document. In order to submit a new task, developers should comply with the SOS official task schema formalized by the following JSON Schema:

**Listing 1** SOS schema for a task.

```
{
 "$schema": "http://json-schema.org/draft-04/schema#",
  description: "A schema to formalize a SOS task",
 type: "object",
 properties: {
   id: { description: "task identifier", type: "integer" },
   tags: { type: "array", items: {type: "string"},
           minItems: 1, uniqueItems: true },
   placeholders: { description: "Dynamic values in the request",
                   type: "array", items: {type: "string"},
                   uniqueItems: true },
   endpoint: { description: "Task endpoint", type: "string" },
   request: { description: "The request (template with placeholders)",
              type: "object" },
   pathQueryResponse: { description: "Queries for the response",
                        type: "array", items: {type: "string"} } },
   required: ["id", "tags", "endpoint", "pathQueryResponse"]
}
```

Using schemata is always a good approach for validating client-submitted data, specially with the absence of a GUI interface. A JSON Schema has several purposes, one of which is JSON instance validation. The JSON schema is composed by six important elements that must be included in a SOS task instance:

- id – is a numeric value that uniquely identifies a task;
- tags – is an array with keywords that characterise the task. Their inclusion is crucial for the task discovery process;
- placeholders – specifies places in the request where substitution may take place;

- endpoint – the task URL endpoint;
- request – a request template sent to the task implementer. The template can include placeholders that will be changed automatically by the previous task response in the chain or, manually, through the client API;
- pathQueryResponse – a set of queries executed against the task implementer response in order to obtain specific results. These results can be used as input for the following tasks.

One final note for the description property. This property can serve multiple purposes, such as, to assist users with the editor GUI operations and to automatically generate the API code documentation.

As an example, the next code shows the JSON document for the Ipify service[9]. This service allows to get your public IP address programmatically.

**Listing 2** Task instance for Ipify service.

```
{
    "id": 1,
    "tags": ["ip", "machine"],
    "placeholders": ["json"],
    "endpoint": "https://api.ipify.org?format=$1",
    "request": {},
    "pathQueryResponse": ["ip"]
}
```

The JSON document for the Ipify service is very simple. We included two strings for the tags property. The property placeholders is fulfilled with one value which will be used by default. However this default value will only be used if the developer does not associate another value through the client API. A placeholder is marked with a sequential number preceded by an $. In this example, there isn't any request message to send to the service implementer, thus the request property is empty. Finally, the pathQueryResponse defines one query to be performed in the task response: {"*ip*" : "98.207.254.136"}. The value of this property adheres to the public specification json-query[10]. After submitting a task as a JSON document, it is validated through a validator. For the validation process, the editor uses the jsonschema validator[11] for Node.js. The validator uses the previous JSON schema and produces a result. The following code validates a specific submitted task instance:

**Listing 3** SOS schema for a task.

```
var Validator = require('jsonschema').Validator;
var v = new Validator();
v.addSchema(mainSchema, '/SOSTaskSchema');
var result = v.validate(myInstance, mainSchema);
if(result.valid === true) { // actions for successful validation }
else { // iterate result.errors }
```

In case of validation errors, they will be appended to the result.errors array which also contains the success flag result.valid.

[9] Link: `https://api.ipify.org?format=json`.
[10] Link: `https://github.com/mmckegg/json-query`
[11] Link: `https://github.com/tdegrunt/jsonschema`.

**Table 1** API functions of the SOS engine.

| Function | Description |
|---|---|
| `SOService SOService.createService(Url manifest)` | Creates a new service based on a SOS manifest |
| `SOService.start()` | Init the service |
| `Task SOService.getCurrenatTask()` | Gets the current task |
| `List<PlaceHolder> task.getPlaceholders()` | Obtains the placeholders defined for the current task |
| `PlaceHolder.setValue(int index, Object value)` | Assigns a value for a specific placeholder |
| `Object task.run()` | Run a specific task |
| `Object SOService.executeAll()` | Execute all tasks automatically and returns the response object of the last task |

### 3.1.3 The SOS manifest

The aggregation of tasks should be done through the GUI component (using drag-and-drop). The aggregation of two tasks should fail if the tasks are not eligible for pairing. The eligibility is granted only if the two tasks (for instance, Task A and Task B) have response and request types similar. In other words, we cannot pair task A that returns a date with task B that are expecting an integer as input.

After the chaining process, the editor will produce a manifest (a new JSON document) with all the selected task instances included. The manifest structure is very simple and can be accessed by the SOS engine through an URL endpoint or with a local reference.

## 3.2 The engine

The engine is a software component that acts as the orchestrator in the SOS realm. The main goal of the engine is to manage the execution flow of the service and to free the developer of the need to handle HTTP clients and parsing request and response messages.

In practical terms, the engine receives a SOS manifest (from a local reference or a cloud location), maps the JSON data to a set of objects and iterates over them. At the same time, it provides a simple API so the developer can manipulate all the tasks in the composite service. In Table 1, we present some of the API functions of the SOS engine.

The engine support two run modes:

- **Automatic mode** – executes all the tasks automatically, picking the response of the current task and inject it in the request of the following task. In the end, it returns the response of the last task. In order to trigger this mode, the developer must use the executeAll method;
- **Iterative mode** – performs one task at a time. For each task, the developer can inject values to map with placeholders. Use this mode when the inputs of the app users are crucial for the flow of the service. In order to trigger this mode, the developer must use the run method of the specific task, and then, use the setValue method of the PlaceHolder object to inject new values from the user's input;

The engine uses the Fuel library[12] as the HTTP client. In short, with Fuel we can make HTTP GET requests to specific endpoints and use the responseString() method to

[12] Link: `https://github.com/openstack/fuel-library`.

asynchronously get the responses. At the time of writing, a Java binding implementation of the engine is being created and can be used, in Android platforms (for instance), as a Gradle dependency.

## 4 Case Study: a Photo App

In this section we briefly detail a Photo mobile app as a case study for the SOS tool. The main goal of the app is to allow the user to take a photo and, automatically, receive a description of the photographed object. In order to accomplish these requirements we need three services: (1) a service to analyse an image and get the name of the photographed object; (2) a service that translate the given name to another language (e.g. Portuguese); (3) a service that takes the translated name and returns its definition.

After some research, we found three services to address these requirements, namely: Google Cloud Vision[13], Google Cloud Translation[14] and Dicionario-Aberto[15] services. The first two are part of the Google Cloud Machine Learning services that provides fast, large scale and easy access to machine learning services. The last one consists of a project with more than 100,000 entries of the Portuguese dictionary providing a RESTful "user-friendly" API which returns results in XML and JSON formats.

The first step was the submission of each service as a task in the Editor. In this case, we need to create a JSON document that complies with the JSON schema previously described. For reasons of compactness and readability, we only expose the JSON instance for the first service, the Cloud Vision service.

**Listing 4** Task instance for Google Cloud Vision API.

```
{
  "id": 1,
  "tags": ["vision", "image manipulation"],
  "placeholders": ["DEFAULT_KEY", "<BASE64-ENCODED IMAGE DATA>"],
  "endpoint": "https://vision.googleapis.com/images:annotate?key=$1",
  "request": {
      {"requests": [{
        "image": {"content": $2},
        "features": [{"type": "LABEL_DETECTION"}]
      }]}
  },
  "pathQueryResponse": "responses/labelAnnotations[0]/description"
}
```

The request property of this task deserves an explanation. In order to send data to the API, one should create an HTTP POST request. The body of the request must be a JSON document containing the Base64-encoded image data. For label detection, the document must also have a features array containing the value LABEL_DETECTION. When we use the label detection feature, the Vision API returns a JSON document containing labels. Along with each label, we also get a score specifying how accurate the label is. In this case, we pick the first label since it will be the most accurate.

[13] Link: `https://cloud.google.com/vision/`.
[14] Link: `https://cloud.google.com/translate/`.
[15] Link: `http://dicionario-aberto.net`.

After all the tasks submitted, the business analyst can select the tasks and create a composite service. The SOS Editor will generate a cloud endpoint for the manifest. Then, the developer, using the SOS library, can load the manifest through the engine API and iterate all over the tasks of the composite service. In this example, the developer must inject the Vision API key (previously obtained) and a Base64 encoded string based on the photo taken. The following code shows how to handle the engine API in order to inject user's value in tasks' placeholders and to manage the execution flow of the composite service.

**Listing 5** SOS engine execution flow.

```
Process process = SOS.createProcess("http://sos.com/manifests/123)
ArrayList<PlaceHolder> placeHolders = new ArrayList<PlaceHolder>();
placeHolders.get(0).setValue(1, VisionApiKey);
placeHolders.get(1).setValue(2, base64Data);
process.getCurrentTask().run();
```

In short, the user takes a photo. Then, the generated Base64 encoded string is used as input for the Cloud Vision service. This service returns a label that represents the photographed object. Then, the label is translated using the Translation service. With the label translated, we use the last service to obtain a definition of the label. This label is presented in a Snackbar GUI component in the bottom of the device screen.

## 5 Conclusions

In this paper we present SOS as a tool for service composition. The main idea is to use an intelligent Web editor to aggregate small services as tasks and package them in a composite service. These composite services can be loaded in an engine library that will manage the execution flow of the service and abstract the developer of all the bureaucratic aspects related to HTTP messaging management.

The main contributions of this work is the JSON schema to describe a task and the a client API that will be exposed by a service engine.

As future work we intend to create a prototype by choosing a specific domain and by implementing all these components. In the editor we intend to maintain it simple (to justify its name), but the inclusion of conditional and cyclic blocks are being considered.

## References


1 Gero Decker, Oliver Kopp, Frank Leymann, and Mathias Weske. BPEL4Chor: extending BPEL for modeling choreographies. In *International Conference on Web Services*, 2007.

2 Howard Foster, Sebastian Uchitel, Jeff Magee, and Jeff Kramer. Model-based analysis of obligations in web service choreography. In *Advanced Int' Conference on Telecommunications and Int' Conference on Internet and Web Applications and Services*, page 149, 2006.

3 Yang Hongli, Zhao Xiangpeng, Cai Chao, , and Qiu Zongyan. Exploring the connection of choreography and orchestration with exception handling and finalization/compensation. In John Derrick and Jüri Vain, editors, *Formal Techniques for Networked and Distributed Systems*, pages 81–96, 2007.

4 Ashley McNeile. Protocol contracts with application to choreographed multiparty collaborations. *Service Oriented Computing and Applications*, 4(2):109–136, 2010.

5 Johannes Maria Zaha, Alistair P. Barros, Marlon Dumas, and Arthur ter Hofstede. Let's dance: A language for service behavior modeling. In Robert Meersman and Zahir Tari, editors, *On the Move to Meaningful Internet Systems*, pages 145–162, 2006.

# Visualization of Ontology Evolution Using OntoDiffGraph*

**André Lara[1], Pedro Rangel Henriques[2], and Alda Lopes Gançarski[3]**

1 **Centro ALGORITMI, Universidade do Minho, Braga, Portugal**
a64362@alunos.uminho.pt
2 **Centro ALGORITMI, Universidade do Minho, Braga, Portugal**
prh@di.uminho.pt
3 **Institut Telecom, Telecom SudParis, CNRS Samovar, Evry, France**
alda.gancarski@telecom-sudparis.eu

## Abstract

Ontologies evolve with the passing of time due to improvements, corrections or changes in requirements that need to be made. In this paper we describe a thesis work aiming at the creation of a visualization technique with the objective of allowing the viewer to easily identify changes made in an ontology. With the use of a specification based on the already existing Visual Notation for OWL Ontologies (VOWL) it is possible to display the differences that exist between two versions of an ontology. The proposed approach will be implemented in an application, that is also discussed in the paper.

**1998 ACM Subject Classification** H.3.1 Content Analysis and Indexing

**Keywords and phrases** Ontology Evolution, Ontology Visualization

**Digital Object Identifier** 10.4230/OASIcs.SLATE.2017.14

## 1 Introduction

Through the use of ontologies it is possible to store entities and their relations with each other. However with the increase in complexity of an ontology, the risk of the user becoming unable to keep up with the changes that are made might make it necessary to change the method used to visualize the ontology. One of the most intuitive forms of displaying an ontology is through the use of a graph, however there are various different forms to display the same information in a graph. Graphs can be presented through the use of force-directed, orthogonal, radial, trees, and many other types of layouts as it can be seen in [3]. In order to map ontology elements to graphical entities there are already existing notations, one example of this is the Visual Notation for OWL Ontologies (VOWL) notation [9].

The Friend of a Friend (FOAF) Ontology[1], is an ontology that contains information about people and the connections they have between each other. Since the year this ontology was created (2000) until the release of the most recent version (2014), this ontology went through several different versions. Tools to analyze this evolution in the ontology already exist [7, 10, 5] however the visualization aspect of these tools can still be greatly improved. The existence of these tools confirms that there is a problem that needs to be solved. The creation of a visualization technique that allows users to easily identify changes would help

* This work has been supported by COMPETE: POCI-01-0145-FEDER-007043 and FCT – Fundação para a Ciência e Tecnologia within the Project Scope: UID/CEC/ 00319/2013.

[1] Available from http://xmlns.com/foaf/spec/.





users analyze how ontologies are evolving and quickly determine what has been changed between different versions of an ontology.

With the use of a graph to display an ontology it is possible to display the results of a structural *diff* inside the graph. If this is successfully executed it is possible to create a way to easily display the evolution of an ontology. The existence of an application that implements these features will allow users to analyse the structural changes made in different versions of an ontology and visualize the impact that these changes made in the graph displaying all concepts and relationships of the ontology.

The project here discussed has the following objectives:

- Proposal of a visualization technique that allows the users to easily identify changes in an ontology. Our contribution focuses on the possibility for the users to visualize the changes made on an ontology without having to see the entire ontology, which is difficult to manage. They can easily select the desired changes to visualize them directly in the ontology display window and a textual description.
- Implementation of this visualization technique in an application using Java.

This thesis research hypothesis is that through the use of the proposed technique, a visual *diff* using a graph to display the ontology and the changes made to it, it will be easier and faster to identify and locate the differences in versions of an ontology and help the user comprehend their meaning and impact.

This paper is divided in the following sections:

- Introduction: In this section we introduce the reader to the context of this paper and what it aims to achieve.
- Related Work: This section identifies and discusses work related to the proposal here reported. A broad and deep research was made on the topics concerned with this project: visualization of ontology evolution, and change detection. However for the sake of space it is not possible to include here all the material discovered; to read all the information collected and organized, the reader should see [8].
- OntoDiffGraph: The Proposal: In this section we discuss the proposed technique and the architecture of the application where it is going to be implemented, OntoDiffGraph.
- OntoDiffGraph: Development: This section contains information related to features and decisions that were taken during the development of the application.
- Conclusion: In this section, we analyse what has been described in this paper and the future work that still needs to be done.

# 2 Related Work

## 2.1 Ontology Visualization

There are languages that can be used to serialize the content and structure of an ontology. One of these languages is the Web Ontology Language (OWL [1]), built on the already existing Resource Description Framework (RDF [2]) and RDF Schema (RDFS [4]) specifications.

Visual Notation for OWL Ontologies (VOWL) [9] is a visual language for ontologies with the aim to help users understand the structure of an ontology intuitively. The VOWL notation provides a graphical version of the various existing ontology elements and has been improved since its initial release. In Figure 1 it is possible to observe the representation of the various elements of an ontology. This notation can be read in more detail in [9].

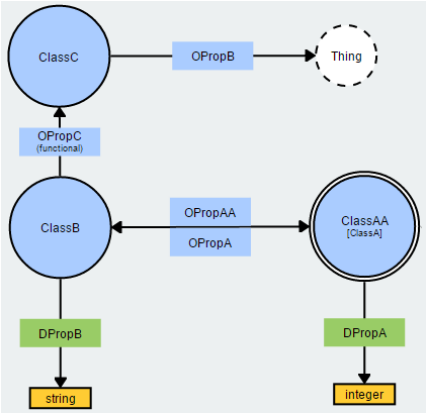



**Figure 1** Visualization of an ontology with the VOWL notation.

## 2.2 Change Visualization

PROMPT-Viz [11] is a Protégé plugin that extends the PROMPTDiff plugin so that the differences between ontologies can be displayed intuitively. This plugin uses a zoomable treemap layout to optimize the utilization of the screen, facilitating the visualization of large ontologies.

Arcs are used to display changes in the location of classes in the treemap and its connections to the other nodes. The selection of a node reveals its arcs to other nodes, these arcs are coloured depending on how the destination class has been changed. This feature cluttered the visualization if it was active for all the nodes all the time, therefore this information is only displayed on a selection event. However the users that evaluated this plugin still had difficulty understanding the information the arcs were trying to convey.

AberOWL [12] is an ontology repository and framework for ontology-based data access through the use of a web interface. In AberOWL it is possible to visualize ontologies as directed graphs where nodes represent classes and edges represent axioms. It is also possible to visualize the difference between several versions of an ontology, to achieve this AberOWL uses a different colour for each version of the ontology to differentiate them. However the visualization of the evolution of an ontology is not the main feature of this system. It is not possible to search for the changes between two different versions, the user must expand the tree manually to visualize the entire ontology and be able to detect the evolution, this is not a practical solution because if the user needs to visualize a large ontology with a large number of nodes it will take a very long time to get the desired results.

# 3 OntoDiffGraph: The Proposal

We propose a system, called OntoDiffGraph, exploring a new visualization technique to easily identify the differences between two versions of an ontology. This technique uses a visual notation that is based on the already existing VOWL specification [9], used to map ontology elements to visual graph elements in order to easily display an ontology, with the objective of displaying the differences between two versions of an ontology.

To make the user able to easily identify what has been changed and the type of change made to an element we decided to add borders to the visual representation of the various ontology elements. The types of changes that will be identified are creations, modifications and deletions. Each one of these types will have a colour associated with them: creation will

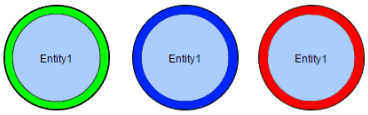



**Figure 2** Class Creation, Modification and Deletion.

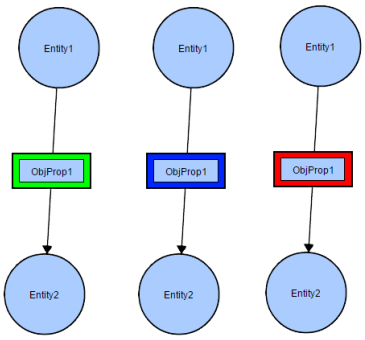



**Figure 3** Relation Creation, Modification and Deletion.

have a green border, modification will have a blue border and deletion will have a red border. Figure 2 shows what a owl:Class node would look like when displayed with this notation.

Figure 3 shows what the edges would look like when the notation is applied on a owl:ObjectProperty.

Changes to the domain or range of the object property will make this element have a blue border, because these changes are considered a modification of the property. The change in domain and range will also be seen in colours of the edges connecting the property to its targets. Figure 4 displays a change in the object property domain and range, the domain and the range used to be **Resource** but in the new version it is **Thing**.

In Figure 5 it is possible to observe what the edges would look like when the notation is applied on a owl:DatatypeProperty.

However, due to the large size of some ontologies, it can still be difficult to locate the changes existing from one version to the other one, even with the added visual information. To solve this, the user will also have access to the textual version of the changes made in the ontology. When the user selects one of the changes in the textual version the viewable area in the application will reposition itself so that the change in centered on that area.

## OntoDiffGraph: Architecture

The previously stated features are being implemented in a Java program, using JavaFX to draw the graph and OWL API [6] to extract information from the ontology.

Figure 6 contains the architecture of the proposed system. This system will be able to load ontologies and display the differences between them through the use of a graph.

To load ontologies, we use of the OWL API [6], a Java API used to create, parse, manipulate and serialize OWL ontologies. This process can be observed in Figure 6 process named *Load Ontologies*. After the loading task the system compares the ontologies that were loaded and identifies the differences that exist between them, in the *Calculate Differences* process. With this information, the system will be able to generate a graph in the *Generate Graph* process. This graph contains the ontology information and the differences between the ontologies. After creating the graph structure, the only step left to do is drawing the

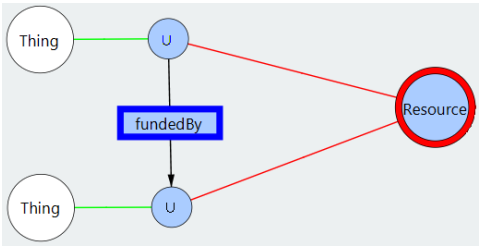



**Figure 4** Example of a change in domain and range.

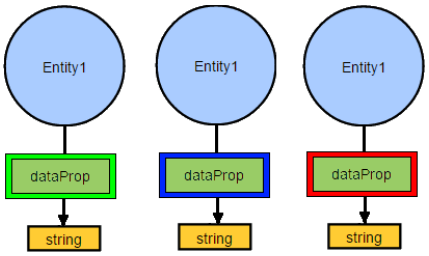



**Figure 5** Datatype Creation, Modification and Deletion.

graph so that the results can be displayed to the user. This will be done with the use of a JavaFX Canvas in the *Draw Graph* process.

## 4 OntoDiffGraph: Development

This section contains detailed information about the features and decisions made in the development of the application. It does not only contain the implementation of the technique but all the necessary features for a complete ontology visualization application, such as manipulation of graph elements, layout customization or the visualization of the ontology.

### 4.1 Basic features

Some important features need to be added to improve the usability of the application. The following features were considered must-have and have been implemented in the application:

- Zooming, allows the users to zoom in and out so that the graph can be seen from different distances.
- Panning, allows the users to move the visible graph area of the application.
- Dragging, allows the users to drag nodes of the graph to their desired position.

It will also be possible to easily find the location of an element in the graph by the use of the menu containing the list of classes, object properties and data properties of the ontology. By selecting an element of the list, the node in the visualization will be centered in the viewable area. This menu is shown in the following figure:

In this menu it is also possible to see the elements of the ontology that were added, modified and removed.

The background colour of each cell shows what happened to that element between the different versions, the possible colours are:

- White, the element did not change from one version to the other.
- Green, the element was added to the ontology.
- Blue, the element was modified.
- Red, the element was removed from the ontology.

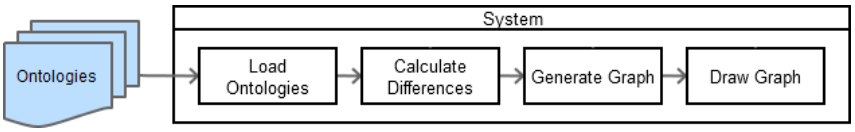



**Figure 6** Architecture of the OntoDiffGraph system.

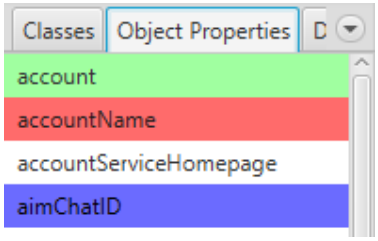



**Figure 7** Menu with all ontology elements.

## 4.2 Layout Customization

In the application it is possible to change general and layout specific values during runtime. This makes the user able to easily learn the impact of each parameter in the graph visualization and is a better option than reloading the entire graph when changes are made to these.

The currently available variables that can be changed in the force-directed layout are the following:

- Edge spacing, controls the distance between edges with the same node ends.
- Repulsion force, controls the repulsion force of the nodes of the graph.
- Attraction force, controls the attraction force of the edges of the graph.
- Spring length, controls the length of the edges.
- Damping, controls the damping of the forces of the graph.
- Timestep, controls the seconds that pass in every iteration of the algorithm.

## 4.3 Change detection

In order to find all the differences between two versions of an ontology it is necessary to compare its contents. All ontology elements contain an unique IRI that identifies them, this will help identify elements that were added or removed. The following list contains the change made in an ontology and how it is possible to determine that change:

- Element was added, an element is classified as added to the ontology if its IRI does not exist in the initial version but exists in the final version.
- Element was removed, an element is classified as removed from the ontology if its IRI exists in the initial version but does not exist in the final version.
- Element was modified, an element is classified as modified if information contained in this element is modified. An example of this is a change in the element domain or range as seen in Figure 4.

However this method has some weaknesses, one of those is that it does not classify a change in an element name as a modification. The renaming of an element will change its IRI, therefore this change will be displayed as the removal of the element with the old name and the addition of a new element with the new name.

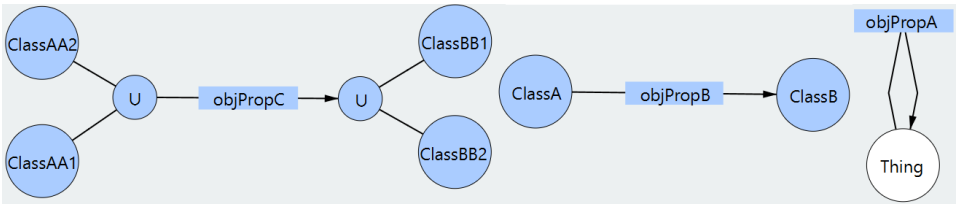



**Figure 8** Visualization of classes and object properties in the application.

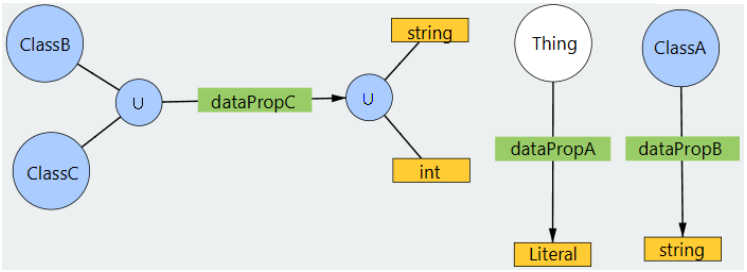



**Figure 9** Visualization of classes and data properties in the application.

## 4.4 Visualization

In order to create the visualization it is necessary to obtain all the elements of the ontology and transform these into nodes and edges with their respective visual representation. The following figures displays classes, object and data properties as seen in the developed application.

In the graph it is also possible to observe the union node that is created when the property domain or range contains more than one element.

# 5 Conclusion

In this paper the context, motivation and objectives of the underlining work were presented and discussed. The outcomes concerning the research done on the topics involved in the state of the art of our working area were reported. Concerning the topic of ontology visualization we justified the adoption of some of the visual elements from VOWL with the fact that it is a good notation to display ontologies and that it is already being used by several applications. Tools that allow the visualization of the difference between ontologies and their main strengths and weaknesses were also discussed – such survey supported our decisions in the creation of OntoDiffGraph. To solve this problem we propose a solution that consists of a visualization technique that will be implemented in an Java application. The architecture, features and development decisions of this application were detailed in this paper.

The application that was proposed is still under development, however the visualization of an ontology is almost complete. There is still work that needs to be done in the detection of differences in two versions of an ontology and the representation of this information however it is already possible to determine the ontology elements that were added and removed from the different versions. After the implementation of these features in the application it will be evaluated to verify if it needs to be modified or improved.

## References


1. Sean Bechhofer, Frank van Harmelen, Jim Hendler, Ian Horrocks, Deborah L. McGuinness, Peter F. Patel-Schneider, and Lynn Andrea Stein. *OWL Web Ontology Language Reference*. World Wide Web Consortium, 2004. `http://www.w3.org/TR/2004/REC-owl-ref-20040210/`.
2. Dave Beckett. *RDF/XML Syntax Specification (Revised)*. World Wide Web Consortium, 2004. URL: `http://www.w3.org/TR/2004/REC-rdf-syntax-grammar-20040210/`.
3. Giuseppe Di Battista, Peter Eades, Roberto Tamassia, and Ioannis G. Tollis. Algorithms for drawing graphs: an annotated bibliography. *Computational Geometry: theory and applications*, 4(5):235–282, 1994.
4. Ramanathan Guha and Dan Brickley. *RDF Schema 1.1 Recommendation*. World Wide Web Consortium, 2014. URL: `http://www.w3.org/TR/2014/REC-rdf-schema-20140225/`.
5. Michael Hartung, Anika Gross, and Erhard Rahm. CODEX: exploration of semantic changes between ontology versions. *Bioinformatics*, 28(6):895–896, 2012.
6. Matthew Horridge and Sean Bechhofer. The OWL API: A Java API for OWL Ontologies. *Semantic Web*, 2(1):11–21, January 2011.
7. Petr Kremen, Marek Smid, and Zdenek Kouba. OWLDiff: a practical tool for comparison and merge of OWL ontologies. In *22nd International Workshop on Database and Expert Systems Applications*, pages 229–233, 2011.
8. André Lara. Visualization of Ontology Evolution using OntoDiffGraph. Master's thesis, University of Minho, 2017.
9. Steffen Lohmann, Stefan Negru, Florian Haag, and Thomas Ertl. VOWL 2: user-oriented visualization of ontologies. In Krzysztof Janowicz, Stefan Schlobach, Patrick Lambrix, and Eero Hyvönen, editors, *International Conference on Knowledge Engineering and Knowledge Management*, pages 266–281, 2014.
10. Natalya Fridman Noy, Mark A. Musen, et al. PromptDiff: A fixed-point algorithm for comparing ontology versions. In *Association for the Advancement of Artificial Intelligence (AAAI) Conference*, pages 744–750, 2002.
11. David Stephen John Perrin. *Prompt-viz: Ontology version comparison visualizations with treemaps*. PhD thesis, University of Victoria, Canada, 2004.
12. Miguel Ángel Rodrıguez-Garcıa, Luke Slater, Keiron O'Shea, Paul N. Schofield, Georgios V. Gkoutos, and Robert Hoehndorf. Visualizing ontologies with AberOWL. In James Malone, Robert Stevens, Kerstin Forsberg, and Andrea Splendiani, editors, *Semantic Web Applications and Tools for the Life Sciences*, pages 183–192, December 2015.

# Socii: A Tool to Analyze and Visualize Dynamic Social Networks*

**Jorge Daniel Caldas[1], Alda Lopes Gançarski[2], and Pedro Rangel Henriques[3]**

1 **Centro Algoritmi/Dpt.Informática, Universidade do Minho, Braga, Portugal**
a67691@alunos.uminho.pt
2 **Institut Télécom, Télécom SudParis, CNRS Samovar, Evry, France**
alda.gancarski@telecom-sudparis.eu
3 **Centro Algoritmi/Dpt.Informática, Universidade do Minho, Braga, Portugal**
prh@di.uminho.pt

## Abstract

Social media network analysis represents a major challenge for data scientists in every aspect, since the extraction all the way to the visualization. Despite representing a major technological challenge, social media data analysis has an additional motivation, that is the daily usage in every country across the planet, making Online Social Network (OSN) a universal tool for communication, such as radio or TV, but with the technological flavor of the 21st century. In the present article, we propose a system, called Socii, for social networks analysis and visualization, as part of an ongoing work under a master's dissertation. This system overlaps two main scientific fields, sociology (more concisely social networks) and computer science. Socii aims at helping OSNs users to know and understand social structures through a user friendly interface. The system relies in four main principles, namely simplicity, accessibility, OSNs integration and contextual analysis.

**1998 ACM Subject Classification** D.3.4 Processors

**Keywords and phrases** Social Networks, Online Social Networks, Social Network Analysis, Data Analytics

**Digital Object Identifier** 10.4230/OASIcs.SLATE.2017.15

## 1 Introduction

In the mid 1950s sociologists introduced the term Social Network (SN), that despite being a familiar term for today general public because of the Online Social Networks (OSNs) platforms such as Facebook, Instagram or Twitter, it is a deeper and more mature concept. It was in the 2000s that much of the OSNs we know today start emerging, so it took at least ten years to people to adopt the concept and the new way of living, so today billions of people use these online platforms as channels for socializing, connect with each other and share their daily lives.

From the user's point of view we may consider that all the platforms offer a microscopic perspective from within the network, people have a public profile, and they can visualize their friend's profile (this is a typical scenario that we observe today in the majority of the OSNs), and normally have access to a timeline that displays friends activity. The point is

* This work has been supported by COMPETE: POCI-01-0145-FEDER-007043 and FCT – Fundação para a Ciência e Tecnologia within the Project Scope: UID/CEC/ 00319/2013.





that, to the users of these online platforms, it is not provided a mean to visualize and analyze their network structure in a more abstract and generalized sense, where users are given the opportunity to observe their social network from a macroscopic perspective, and, with that, all the metrics for measuring nodes and relationships within the network.

This work is being developed under the master's dissertation which aims to prove that, a software tool may be designed and implemented, in order to actually improve the analysis of social phenomena, allowing not only sociologists, but also the public in general, to explore with greater detail the connections of individuals within a network, being OSNs the base of analysis for such a tool.

## 2 Origins of Social Networks (a Sociology Perspective)

Event if today people automatically think in Social Networks (SNs) as websites (or web applications), they are not aware that when talking about SNs, we refer to a much more broader term, that said, we may consider a SN as the following:

> *A social structure made of nodes that are generally individuals or organizations. A social network represents relationships and flows between people, groups, organizations, animals, computers or other information/knowledge processing entities. The term itself was coined in 1954 by J. A. Barnes.* [2]

The network concept is broadly used across multiple fields of study, including, physics, biology, linguistic, anthropology, mathematics, computer science and more recently computer networks, we may certainly say that *"the network concept is one of the defining paradigms of the modern era."* [8].

## 3 Online Social Networks

People need to connect other people, and the urge for connection brings to us what today are known as OSNs. These Web sites allow us to define a profile as an individual, and to share and visualize content with other individuals in the network, therefore connecting.

> *We define Online Social Networks as Web-based services that allow individuals to construct a public or semi-public profile within a bounded system, articulate a list of other users with whom they share a connection, and view and traverse their list of connections and those made by others within the system (...)* [6]

OSNs have been around for more than a decade now, but these systems have gain world wide popularity since the global adoption of platforms such as Facebook, Youtube or Twitter, which are platforms that are today massively used across all cultures and age groups, and represent a paradigm shift on social interaction that we not yet fully understand.

## 4 Social Network Analysis

Social Network Analysis (SNA) is the study of how people are connected to each other, basically it studies a set of relations among a set of entities, these entities may be individuals, organizations, or even countries.

The common analysis procedure consists in mapping the network and then creating metrics to characterize the network. Then one tries to figure what is the structure of the

network and why does it have that structure. Social Network Analysis (SNA) is also about looking at the individuals inside the network and where are those individuals located.

We try to explore social network analysis in order to understand the OSNs reality, for that purposes we explored concepts such as clustering or centrality measures.

## 4.1 Social Network Analysis Software

In this section we talk about software tools that we analyzed with greater detail in the master's pre-dissertation [4], but that we found relevant to refer in the present article.

We reference tools such as UCINET[1] that are very mature and well known among SNA community, then we looked into more modern tools with a set of advanced features and more modern user interfaces, Gephi [1] and Social Network Visualizer[2] (SocNetV). We look also into more specific purpose tools such as Structure[3] which focus on trying to understand population flow. Investigating more deeply academic work on this field, we found Vizster [7], a visualization system for playful end-user exploration and navigation of large-scale online social networks.

# 5 Socii: our proposal

Given the complexity of SNA for a common user, we propose a software tool, called Socii, that allows the user easily understand and interpret his network structure. For that, not only useful metrics are computed, but also they are visualized in a convenient way, together with the network structure.

Before diving into the architectural details of the system, we will present Socii positioning. This is result of the state of the art summary, that concisely describes what is the positioning of this project in light of the existent explored SNAs tools. This comparative study can be consulted in the master's pre-dissertation document [4].

### Simplicity

Aside of **Vizster** [7], the majority of the previously presented tools such as Gephi or Social Network Visualizer, are very complex tools with very heavy interfaces, that have a big learning and are meant for users that have particular advanced knowledge in SNs and SNAs. The system we are developing could also serve for less expert users, providing a set of core basic functionalities (e.g only allow users to load and visualize their networks), and then, allow the user to build complexity from there enabling and disabling other features.

### Accessibility

All the software that we presented above exist in the form of desktop applications. This applications need to be downloaded, and installed in a compatible machines (sometimes with dependencies on other software that is not installed by default). Nowadays almost every application is Web based, this allows users to access them every where trough a browser, making Web apps a solution that is Operating System and device agnostic. This said,

[1] Available from `https://sites.google.com/site/ucinetsoftware/home`.
[2] Available from `http://socnetv.org/`.
[3] Available from `http://web.stanford.edu/group/pritchardlab/structure.html`.

building a Web based social networks analysis tool could be a way of tackle the accessibility of such tools.

A Web based application, it is good for sake of accessibility but in another hand it is a performance culprit when it comes to performance. This is a decision to take into account, **but always having in mind that tackling performance is not the main goal this master's thesis**, also, the mentioned tools are mature projects that are highly performant and are capable of rendering huge networks.

### Online Social Network (OSN) integration

Social Network Visualizer, allows to *scrap* Web sites to build networks, but for this feature relays only on links to build the network (it blindly scraps recursively some url to build the network). By allowing the user the power to the user to analyze networks that are directly reporting their social network status would be a differentiation factor from the other tools, and would certainly be a more meaningful and valuable analysis for the end user. This feature of OSNs integration may be simulated by data generators that are specially deliver fictitious data sets within the domain of a given OSN.

### Drawing Accurate Conclusions

As we state before when talking about simplicity, the mentioned SNAs tools provide generic metrics on networks such as network density or actor centrality. The values outputted from these tools are the result of running generic formulas and algorithms against some networks, so it is very common for current SNAs researchers to be worried about the size of the network, being their focus on **quantitive analysis**.

In a hypothetical analysis scenario where some researcher has a network with a few thousand nodes, **what is the meaning of his assumptions when analyzing the network**? Since this is a pure quantitive analysis the numbers will seem reasonable for the given network, but this will not allow him to extract contextual conclusions, because in this case analyzing data form Facebook or analyzing data from LinkedIn will sound just like the same, because deep down it all comes down to the network. A better approach for drawing conclusions would be to have a mixture between **quantitive analysis** and **quality analysis**, the tool could do some content and context analysis to help the end user to get a more meaningful conclusion rather than just simple metrics.

## 6 System Architecture Proposal and Implementation Details

In this section we present a more concrete image of the overall system. In Figure 1 we may observe an abstract diagram that represents the system architecture.

As the interaction of the software components may be understood from simply observing the diagram in Figure 1, a description of the specific role per component is needed in order to explain the system overall functionality.

### Database

We use a **MongoDB**[4] (a document oriented database) to store data, this gives us more flexibility in manipulating complex JSON structures that are persisted in documents. These

[4] Available from `http://networkx.github.io/`.

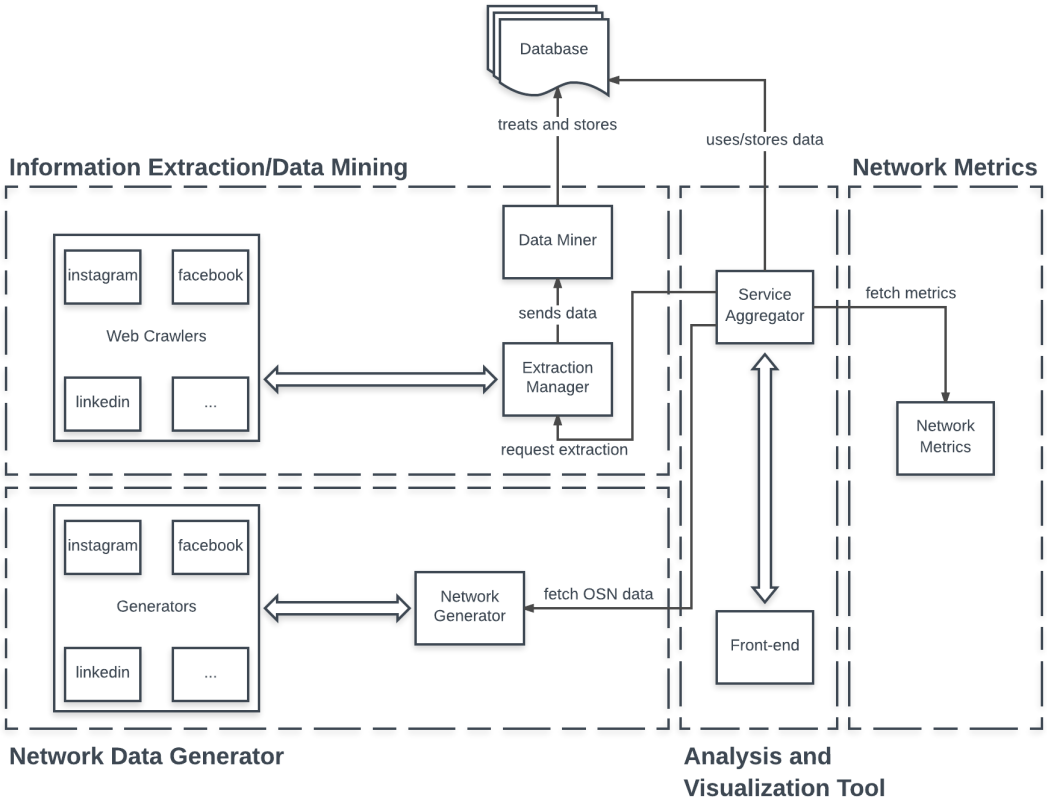



**Figure 1** System architecture proposal.

flexibility and interoperability would be considerably more complex to achieve using relational databases.

## Information Extraction and Data Mining

The purpose of this component is to actually extract some user network from a given OSNs. This is achieved by calling the web crawler modules (each one dedicated to a specific OSNs), that return the extracted information to the extraction manager which then sends the data through a simple data mining process in order to get data normalized before it is stored into the database. The web crawlers are implemented in Python that plays nicely along side with Selenium[5] and PhantomJS[6] to perform crawling operations using **XPath** [5] selectors to extract the information that we need to build the network.

We are aware that this kind of work is not easily achieved in a reasonable amount of time, but still it is the only way to fully access the data since majority of today OSNs have limited Application Program Interfaces (APIs) that do not allows us to extract the data to build the network. We use a **Flask**[7] APIs that executes background processes to perform parallel extractions (run web crawlers in parallel) to mitigate the slowness of web crawling, still it is a slow and not scalable solution, this is the motivation for building the network

[5] Available from `http://www.seleniumhq.org/`.
[6] Available from `http://phantomjs.org`.
[7] Avaialble from `http://flask.pocoo.org/`.

generator module that we present in a following section, where we retrieve generated mocks that are within a domain of some OSNs.

### Network Data Generator

This component allows Socii aggregator to be fed with mock objects that follow pre-defined data schema specially suited for each OSNs. This is a replacement for the network extraction component, in order to validate Socii instead of having extracted data we can have several fake networks and test several different scenarios. This is also a proof of the architecture interoperability since if in the future some OSNs decide to make their APIs public we can just implement a different extraction component that instead of crawling the OSNs uses data from its APIs. This said we consider the extraction component a pluggable part of the system this architecture is designed to consume any other data retrieval component that returns the same data schema. This generators are developed with NodeJS,[8] and are available as a microservice, one just need to ask for a specific number of users for a specific OSNs.

### Network Metrics

Network metrics is a microservice available trough a web API that performs calculations and runs algorithms against a given network. To consume this microservive, a client (let's say the service aggregator) only sends one HTTP request with the graph and the metrics to calculate upon the given graph. The microservice returns the computed metrics to the client. This microservice is implemented Python (again with Flask), this allows us to use the powerful library NetworkX[9] for performing graph calculations that we refer in Section 4.

### Analysis and Visualization Tool

The tool that directly interacts with the end user is composed by a **Service Aggregator** that communicates with all the available services (extraction, generator and metrics api), and the front end application that provides the visualization and interaction features and it only communicates with the aggregator that it plays a facade/interface role for communication with the other services. Among the usual suspects of front end development (HTML, CSS and Javascript through the React framework,[10] in the context of Socii front end it made sense to develop a flexible and configurable graph visualization abstraction, this work originated a open source React component called **react-d3-graph**[11], which focuses on providing easy graph rendering, interaction and visualization using **D3.js** [3] for core functionalities such as rendering and node positioning calculation, it also allows the consumer to register to graph interaction events via callback functions that allowing custom application reactions upon simple graph interactions such as node and link click.

The Socii tool has a simple interface, where a user is capable of choosing between generating Facebook or LinkedIn networks, and then observe the social structure with react-d3-graph and with the metrics and other relevant data that Socii provides in a convenient way. Socii has two main areas. The configuration area where the user chooses which social network wants to generate, what metrics he wants to analyze. From here the user enters in the main area, the network vizualizer, where the user has in the center the network rendered

[8] Available from `https://nodejs.org/en/`.
[9] Available from `https://networkx.github.io/`.
[10] Available from `https://facebook.github.io/react/`
[11] Available from `https://danielcaldas.github.io/react-d3-graph/docs/`.

with react-d3-graph, on the bottom the user has some main interactions and global network metrics are displayed. When the user interacts with some node in the network, a panel is open where some metrics are shown as well the OSNs data so that the user is able to cross that data in order to derive conclusions. Some particular features were implemented for each OSNs, in Facebook we have a simple sentimenta analysis where user's posts reactions (reactions such as "*likes*") are used to determine users' sentiments in the network, for that the graph assumes a color schema that maps nodes to reactions. In LinkedIn we implemented human resources discovery where we try to find nodes with particular professional skillset and what are the best ways to approach a certain individual through interpersonal relationships within the network.

## 7 Conclusion

In this paper, we propose a new system, Socii, for OSNs analysis and visualization, which allows for contextual network analysis, with a relatively low complexity. Socci is available through the Web and it intends to integrate data from different OSNs. As Socii is based in OSNs platforms that are prune to change, Socii architecture was designed with interoperability in mind in order to allow different data sources to be consumed, being this operations transparent to the end user, not regarding the data that one observers but regarding the behavior and functionality of the application.

## References


1 Mathieu Bastian, Sebastien Heymann, and Mathieu Jacomy. Gephi: an open source software for exploring and manipulating networks. In *Third International Conference on Weblogs and Social Media*, pages 361–362, 2009.

2 Vangie Beal. Definition for 'social network', 2016. Webopedia.

3 Michael Bostock, Vadim Ogievetsky, and Jeffrey Heer. D3 Data-Driven Documents. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2301–2309, 2011.

4 Jorge Caldas. Analysis and visualization of dynamic social networks, 2017. Pre-Dissertation Report, Master Degree in Computer Science at University of Minho.

5 James Clark and Steve DeRose. *XML Path Language (XPath) Version 1.0*. World Wide Web COnsortium, 1999. URL: `https://www.w3.org/TR/xpath/`.

6 Nicole B. Ellison and danah m. boyd. Social network sites: Definition, history, and scholarship. *Journal of Computer-Mediated Communication*, 13(1):210–230, 2007.

7 Jeffrey Heer and danah m. boyd. Vizster: Visualizing Online Social Networks. In *IEEE Symposium on Information Visualization*, page 5, 2005.

8 Martin Kilduff and Wenpin Tsai. *Social networks and organizations*. SAGE Publications, 2003.

# Comparing and Combining Portuguese Lexical-Semantic Knowledge Bases

**Hugo Gonçalo Oliveira**

**CISUC, Department of Informatics Engineering, University of Coimbra, Coimbra, Portugal**
hroliv@dei.uc.pt

## Abstract

There are currently several lexical-semantic knowledge bases (LKBs) for Portuguese, developed by different teams and following different approaches. In this paper, the open Portuguese LKBs are briefly analysed, with a focus on size and overlapping contents, and new LKBs are created from their redundant information. Existing and new LKBs are then exploited in the performance of semantic analysis tasks and their performance is compared. Results confirm that, instead of selecting a single LKB to use, it is worth combining all the open Portuguese LKBs.

**1998 ACM Subject Classification** I.2.7 Natural Language Processing

**Keywords and phrases** Lexical Knowledge Bases, Portuguese, WordNet, Redundancy, Semantic Similarity

**Digital Object Identifier** 10.4230/OASIcs.SLATE.2017.16

## 1 Introduction

Lexical-semantic knowledge bases (LKBs) are computational resources that organize words according to their meaning, typically used in natural language processing (NLP) tasks at the semantic level. Princeton WordNet [11] is the paradigmatic resource of this kind, for English, with a model adapted to many languages, including Portuguese. However, the first Portuguese WordNet [21] was not available to be used by the research community and the first open alternatives were only developed in the last decade.

Several open Portuguese LKBs are currently available, developed by different teams, following different approaches. Due to the difficulties inherent to crafting such a broad resource manually, most LKBs have some degree of automation in their creation process, which increases the chance of noise. Furthermore, none of them is as consensual as WordNet, created manually and with a large community of users, is for English. In fact, while some Portuguese LKBs are not large enough, others have an interesting size but include several incorrect, unfrequent or unuseful relations or lexical items.

In this paper, nine open Portuguese LKBs are characterised in terms of covered lexical items and relations. The redundancy across them is then analysed, towards the creation of (potentially) more useful LKBs. All the LKBs, including the new ones, are finally compared indirectly, when exploited in semantic similarity tasks with available benchmark datasets for Portuguese. Besides confirming our intuition that there are advantages in combining different LKBs, this can be seen as the first systematic comparison of the Portuguese LKBs.

## 2 Related Work

The current scenario for Portuguese LKBs can be seen as atypical. There are currently many open LKBs for this language, but none is as consensual as Princeton WordNet [11] is for


English. This includes several wordnets [8] and other simpler LKBs that, in some cases, may replace a wordnet. For many languages, there is generally one "main" LKB used by the NLP community, possibly further enriched or aligned with different knowledge bases in specific domains or kinds of knowledge. For instance, there are several extensions for Princeton WordNet (e.g. subject field codes [20]), as well as alignments with other lexical resources (e.g. FrameNet and VerbNet [29], or Wikipedia and Wiktionary [17]). WordNet is also the "core" of most multilingual wordnets (e.g. EuroWordNet [32], MultiWordNet [25], Open Multilingual WordNet [4]) and of multilingual knowledge bases that cover linguistic and encyclopaedic knowledge (e.g. Universal WordNet [6], BabelNet [24]). Furthermore, authors working on the automatic acquisition of semantic relations from English text often mention their utility for enriching WordNet [18].

This is probably why there is not much work similar to what is presented here, where LKBs that aim at covering more or less the same kind of knowledge are combined. On the other hand, redundancy models have been proposed for assessing the confidence of relations automatically extracted from corpora [10]. The main intuition is that relation instances extracted more often, from different sources, are more plausible to be correct or useful.

## 3 Open Portuguese LKBs

Nine open Portuguese LKBs were explored in this work, namely:

- Three wordnets: WordNet.Br [9], OpenWordNet-PT (OWN.PT) [7] and PULO [30];
- Two synset-based thesauri: TeP [22] and OpenThesaurus.PT[1] (OT.PT);
- Three lexical-semantic networks extracted from Portuguese dictionaries: PAPEL [15] and relations from Dicionário Aberto (DA) [31] and Wiktionary.PT[2];
- The semantic relations available in Port4Nooj [3], a set of linguistic resources.

As these resources do not share exactly the same structure, to enable comparison and integration, they were all reduced to a set of relation instances of the kind "*x related-to y*", where *x* and *y* are lexical items and *related-to* is a relation name. For synset-based LKBs, synsets had to be deconstructed. For example, the instance

{*porta, portão*} partOf {*automóvel, carro, viatura*}

resulted in:

(*porta* synonymOf *portão*), (*automóvel* synonymOf *carro*)
(*automóvel* synonymOf *viatura*), (*carro* synonymOf *viatura*)
(*porta* partOf *automóvel*), (*porta* partOf *carro*)
(*porta* partOf *viatura*), (*portão* partOf *automóvel*)
(*portão* partOf *carro*), (*portão* partOf *viatura*)

Adopted relation names were those defined in the project PAPEL [15], which covered the relation types in all the LKBs, though some names had to be normalized. Table 1 characterises each explored LKB according to the number of lexical items – for each part-of-speech (POS) and total distinct (not considering POS) – and relation instances, grouped by their broader type.

[1] `http://paginas.fe.up.pt/~arocha/AED1/0607/trabalhos/thesaurus.txt` (April 2017)
[2] `http://pt.wiktionary.org` (2015 dump)

**Table 1** Number of lexical items and triples extracted from each LKB.

| Lexical items | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| **POS** | **PAPEL** | **DA** | **Wikt.PT** | **TeP** | **OT.PT** | **OWN.PT** | **PULO** | **WN.Br** | **Port4Nooj** |
| Nouns | 56,660 | 61,334 | 30,170 | 17,244 | 6,110 | 32,509 | 5,149 | 0 | 8,109 |
| Verbs | 21,585 | 16,429 | 8,918 | 8,343 | 2,856 | 3,626 | 1,573 | 5,857 | 3,161 |
| Adjectives | 22,561 | 18,892 | 9,536 | 14,979 | 3,747 | 4,401 | 1,316 | 0 | 1,055 |
| Adverbs | 1,376 | 3,160 | 610 | 1,138 | 143 | 1,120 | 153 | 0 | 475 |
| **Distinct** | 94,165 | 95,188 | 45,345 | 40,499 | 12,782 | 40,940 | 7,943 | 5,857 | 12,641 |
| Relations | | | | | | | | | |
| Synonymy | 83,432 | 52,278 | 35,330 | 388,698 | 51,410 | 35,597 | 9,189 | 88,488 | 559 |
| Antonymy | 388 | 440 | 1,263 | 92,234 | – | 5,774 | 2,818 | – | – |
| Hypernymy | 49,210 | 46,079 | 22,931 | – | – | 78,854 | 26,596 | 73,302 | 15,303 |
| Part | 5,491 | 4,367 | 1,574 | – | – | 14,275 | 1,146 | – | – |
| Member | 6,585 | 1,057 | 1,578 | – | – | 5,153 | 259 | – | – |
| Material | 336 | 518 | 192 | – | – | 958 | 67 | – | – |
| Contains | 391 | 263 | 120 | – | – | – | – | – | – |
| Cause | 7,700 | 7,211 | 3,278 | – | – | 295 | 291 | – | 3,325 |
| Producer | 1,336 | 913 | 500 | – | – | – | – | – | – |
| Purpose | 9,144 | 5,220 | 4,227 | – | – | – | – | – | 303 |
| Property | 23,354 | 15,732 | 7,020 | – | – | 10,825 | 3,327 | – | – |
| State | 394 | 237 | 79 | – | – | – | 505 | – | – |
| Quality | 1,636 | 1,221 | 381 | – | – | – | – | – | – |
| Manner | 1,268 | 3,381 | 439 | – | – | – | – | – | 850 |
| Place | 832 | 487 | 1,159 | – | – | – | – | – | – |
| **Total** | 191,497 | 139,404 | 80,071 | 480,932 | 51,410 | 151,731 | 44,198 | 161,790 | 20,340 |

**Table 2** Occurrences of the same triples in different resources, per type.

| Relation | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| Synonymy | 230,030 | 65,778 | 17,592 | 7,506 | 3,212 | 1,166 | 377 | 76 |
| Antonymy | 48,444 | 1,257 | 345 | 96 | 22 | 7 | – | – |
| Hypernymy | 247,349 | 25,145 | 4,050 | 516 | 82 | 2 | – | – |
| Part | 22,620 | 1,883 | 146 | 6 | 1 | – | – | – |
| Member | 13,200 | 638 | 48 | 3 | – | – | – | – |
| Material | 1,735 | 159 | 6 | – | – | – | – | – |
| Contains | 635 | 65 | 3 | – | – | – | – | – |
| Cause | 10,668 | 3,115 | 1,158 | 432 | – | – | – | – |
| Producer | 2,216 | 217 | 33 | – | – | – | – | – |
| Purpose | 15,938 | 1,276 | 132 | 2 | – | – | – | – |
| Property | 45,431 | 6,057 | 798 | 76 | 3 | – | – | – |
| State | 1,031 | 81 | 6 | 1 | – | – | – | – |
| Quality | 1,760 | 631 | 72 | – | – | – | – | – |
| Manner | 4,274 | 683 | 98 | 1 | – | – | – | – |
| Place | 1,609 | 286 | 99 | – | – | – | – | – |
| **Total** | 646,940 | 107,271 | 24,586 | 8,639 | 3,320 | 1,175 | 377 | 76 |
| | (81.6%) | (13.5%) | (3.1%) | (1.1%) | (0.4%) | (0.1%) | (0.0%) | (0.0%) |

Although the LKB with more lexical items is the one extracted from DA (≈95,000 distinct items), it contains substantially less relation instances than TeP, which covers ≈490,000 synonymy and antonymy instances but no other relation type. PAPEL, DA, OWN-PT and WN.Br all contain more than 100,000 relation instances. All LKBs cover synonymy; antonymy is not covered by OT.PT, WN.Br and Port4Nooj; and hypernymy is not covered by TeP and OT.PT, because the latter are originally synset-based thesauri. WN.Br only covers verbs and relations between them, but the other wordnet-based LKBs cover all four open POS. Besides synonymy, antonymy and hypernymy, they also cover additional relation types (e.g. part, cause, property), but some types are only found in the LKBs extracted from dictionaries.

## 4 Redundancy in Portuguese LKBs

Despite originally organised in different models, LKBs were created with different approaches, most of which involving automatic or semi-automatic steps. Therefore, although they try to cover the whole language, they end up having different granularities and contents, in terms of covered relations and lexical items, some of which less useful for some tasks, or even incorrect. Table 2 shows the number of relation instances grouped by relation type and number of LKBs they were found in.

The majority of relation instances (≈81%) is in only one LKB, ≈13% is in two, ≈3% in three and just ≈1% in four. Only synonymy, and a residual number of antonymy and hypernymy instances, are in six or more LKBs, expectable because those also happened to be the types covered by more LKBs. Our intuition is that the more resources an instance is in, the more likely it is to transmit a consensual, frequent and useful relation. This is confirmed by observed examples, including those in Table 3, which contains relation instances that are in eight to three LKBs. Each redundancy level includes only instances of types that were not present in the previous level, or were but with arguments with a different POS.

**Table 3** Examples of redundant relation instances.

| # | Examples of relation instances |
|---|---|
| **8** | *agarrar* `synonymOf` *pegar* (grab, catch) |
| | *apressar* `synonymOf` *acelerar* (rush, hasten) |
| | *punir* `synonymOf` *castigar* (punish, discipline) |
| **7** | *pedinte* `synonymOf` *mendigo* (beggar, mendicant) |
| | *espesso* `synonymOf` *grosso* (thick) |
| | *porventura* `synonym` *talvez* (perhaps, possibly) |
| **6** | *público* `antonymOf` *privado* (public, private) |
| | *fácil* `antonymOf` *difícil* (easy, hard) |
| | *árvore* `hypernymOf` *carvalho* (tree, oak) |
| **5** | *degrau* `partOf` *escada* (step, stairs) |
| | *sexual* `propertyOf` *sexo* (sexual, sex) |
| **4** | *investir* `causes` *investimento* (invest, investment) |
| | *feliz* `stateOf` *felicidade* (happy, happiness) |
| | *carta* `memberOf` *baralho* (card, deck) |
| | *votar* `purposeOf` *voto* (vote, vote) |
| | *habilmente* `mannerOf` *habilidade* (ably, ability) |
| | *dependente* `propertyOf` *depender* (dependable, depend) |
| **3** | alterar `hypernymOf` afetar (change, affect) |
| | *impertinente* `qualityOf` *impertinência* (impertinent, impertinence) |
| | *vinho* `containedIn` *galheta* (wine, cruet) |
| | *coqueiro* `producerOf` *coco* (coconut tree, coconut) |
| | *fio* `materialOf` *meada* (thread, hank) |
| | *Brasil* `placeOf` *brasileiro* (Brazil, Brazilian) |

On the other hand, instances that only occur in one LKB are more likely to either be incorrect, resulting from noise on the automatic process, or to involve very specific meanings, thus less useful. Observed examples also confirm this. Some of them are presented in Table 4, which shows a list of relation instances that are in a single LKB, selected randomly for different relation types.

Following the aforementioned intuition, new LKBs were created, based on the redundancy level: one with all the relation instances in all LKBs (*All*) and seven more with the relation instances in at least two to eight LKBs (*Redun2-8*). The resulting LKBs are characterized in Table 5. From those, the largest three (*All*, *Redun2*, *Redun3*) were used to perform the same tasks as the original LKBs, as reported in the following section.

## 5 Comparing Portuguese LKBs indirectly

Our first attempt to compare the Portuguese LKBs relied on their extrinsic evaluation, when exploited to solve semantic similarity-related tasks, for which datasets, here used as benchmarks, are available. This section reports this attempt, which covers four different tasks: selecting the most similar word from a small set (Section 5.1); computing the semantic similarity between pairs of words (Section 5.2); selecting the most suitable word for a blank in a sentence (Section 5.3); and computing the semantic similarity between pairs of sentences (Section 5.4). Table 6 organizes the benchmark tests according to their type.

**Table 4** Examples of relation instances in only one LKB.

| |
|---|
| olorado `synonymOf` aromal (smelt, aromal?), economicamente `synonymOf` regradamente (economically, ordely), saltão `synonymOf` salta-paredes (locust, wall-jumper?), coisa `hasState` clima (thing, climate), lugar-tenente `hasQuality` lugar-tenência (lieutenant, lieutenancy?), satanizar `causes` satanização (demonize, demonization), pressão `causes` depressão (pressure, depression), cobre `containedIn` hemocianina (copper, hemocyanin), despropositado `antonymOf` razoável (inopportune, reasonable), em_definitivo `antonymOf` temporariamente (definitively, temporarily), crueza `antonymOf` clemência (crudeness, mercy), desgarrar `antonymOf` aprochegar (tear apart, approach?), despigmentado `propertyOf` perder_cor (depigmented?, lose_color), diluviano `propertyOf` aluvião (diluvial, alluvium), alfitomancia `purposeOf` farinha (alphitomancy, flour), guarnecer `purposeOf` cacundê (garnish, ?), transformar `hypernymOf` colorir (transform, coloring), atitude `hypernymOf` anticomunismo (attitude, anticomunism), Abissínia `placeOf` abissínio (Abyssinia, Abyssinian), parabolicamente `mannerOf` parábola (paraborically?, parable), imunoglobina `materialOf` plasma (immunoglobulin, plasma), pessoa `memberOf` lobby (person, lobby), kibibyte `partOf` megabyte, caju `producerOf` castanha (cashew, chestnut) |

**Table 5** Size of the redundancy-based LKBs.

| **Redundancy** | **1** (All) | **2** | **3** | **4** | **5** | **6** | **7** | **8** |
|---|---|---|---|---|---|---|---|---|
| **Lexical items** | 178,903 | 56,565 | 23,468 | 11,431 | 5,557 | 2,292 | 764 | 144 |
| **Relation instances** | 791,182 | 145,429 | 38,173 | 13,587 | 4,948 | 1,628 | 453 | 76 |

**Table 6** Characterization of the benchmark tests.

| | **Word level** | **Sentence level** |
|---|---|---|
| **Multiple choice** | $B^2SG$ | Cloze questions |
| **Similarity score** | SimLex-999 | ASSIN |

**Table 7** First entries of each file of the $B^2SG$ test.

| **Relation** | **Target** | **Candidates** | | | |
|---|---|---|---|---|---|
| Synonym (noun) | concorrente | **competidor** | cortina | amurada | carmesim |
| Synonym (verb) | trancar | **barrar** | aviar | alienar | progredir |
| Hypernym (noun) | matemática | **ciência** | célula | pulseira | libertação |
| Hypernym (verb) | segar | **ceifar** | anexar | concentrar | desembrulhar |
| Antonym (noun) | esquerda | **direita** | repressão | sétimo | diácono |
| Antonym (verb) | trancar | **abrir** | praticar | dragar | empenhar |

## 5.1 Selecting the most similar word from a small set

The $B^2SG$ [33] test is similar to the WordNet-Based Synonymy Test [13], but based on the Portuguese part of BabelNet [24] and partially evaluated by humans. It contains frequent Portuguese nouns and verbs (target), each followed by four candidates, from which only one is related, and is organized in six files: two for synonymy, two for hypernymy, and two for antonymy, for nouns and for verbs. Table 7 illustrates the $B^2SG$ test with the first line of each file. The correct answer is always the first candidate, followed by three distractors.

Although created for evaluating less structured resources, such as distributional thesauri, we analysed how many correct relations of this test are covered by the Portuguese LKBs. Furthermore, for the uncovered instances, the correct alternative was guessed from the top-ranked candidate, after running the Personalized PageRank [1] algorithm in each LKB, for 30 iterations, using the target word as context.

Table 8 presents the number of covered (In) and guessed (Guess) relations for each LKB. Coverage numbers highlight known limitations of some LKBs, e.g.: antonymy relations extracted from dictionaries are mostly between adjectives; synset-based thesauri do not cover hypernymy; only the wordnet-based LKBs cover hypernymy between verbs and WN.Br only covers verbs. However, for this specific test, some limitations could be minimized by exploiting the structure of the LKB. As expected, the highest coverage and proportion of guessed relations is obtained for the *All* LKB, for which 97.4% of the instances are guessed. It is followed by OWN-PT on both coverage and guesses, except for hypernymy and antonymy between nouns, for which *Redun2* gets the second highest number of guesses. Yet, we suspect that these numbers are positively biased towards OWN-PT, because it is currently integrated in BabelNet.

## 5.2 Computing the similarity between word pairs

SimLex-999 [19] is a recent benchmark for assessing methods for computing semantic similarity. It contains 999 pairs of words, with the same POS, and their similarity score, given by human subjects who followed strict guidelines to differentiate between similarity and relatedness. No multiword expressions nor named entities are included. This dataset was originally made available for English but has been translated to other languages. The Portuguese translation was originally made to assess the LX-DSemVectors [27], word embeddings learned from Portuguese corpora, and is available online[3]. Table 9 shows the first two adjectives, nouns and verbs of the Portuguese SimLex-999.

In order to exploit the LKBs in this task, two different algorithms were applied to compute the similarity between the words of each pair, namely:

- Similarity of the adjacencies of each word in the LKB, using measures such as the Jaccard coefficient (Adj-Jac) or the cosine similarity (Adj-Cos);
- PageRank vectors, inspired by Pilehvar et al. [26]. For each word of a pair, Personalized PageRank was first run in the target LKB, for 30 iterations, using the word as context; a vector was then created where each position contained the resulting rank of each other word in the LKB. Finally, the similarity between the vectors for each word was computed, using: the Jaccard coefficient between the sets of words in these vectors (PR-Jac) or the cosine of the vectors (PR-CosV). Given the large vector sizes, vectors were trimmed to the top$-N$ ranked words. Different sizes $N$ were tested, from 50 to 3,200.

[3] `http://metashare.metanet4u.eu/` or `https://github.com/nlx-group/lx-dsemvectors/` (April 2017)

**Table 8** Relation instances in and guessed from the B$^2$SG test. Highest and second highest numbers are in bold.

| | LKB | Synon (1,171) | | Hypern (758) | | Anton (145) | |
|---|---|---|---|---|---|---|---|
| | | In | Guess | In | Guess | In | Guess |
| Nouns | **PAPEL** | 28.9% | 84.0% | 5.0% | 78.2% | 0.0% | 63.4% |
| | **DA** | 16.5% | 71.7% | 4.6% | 66.1% | 0.0% | 59.3% |
| | **Wikt.PT** | 16.6% | 66.2% | 5.0% | 67.9% | 8.3% | 74.5% |
| | **OWN-PT** | **62.8%** | 80.1% | **59.0%** | 82.5% | **60.0%** | 82.8% |
| | **PULO** | 13.2% | 30.2% | 18.3% | 38.8% | 27.6% | 49.7% |
| | **TeP** | 33.2% | 63.9% | 0.0% | 52.9% | 32.4% | 69.7% |
| | **OT.PT** | 17.7% | 35.0% | 0.0% | 30.2% | 0.0% | 31.7% |
| | **Port4Nooj** | 0.1% | 17.1% | 0.3% | 20.4% | 0.0% | 26.2% |
| | **WN.Br** | 0.0% | 0.0% | 0.0% | 0.0% | 0.0% | 0.0% |
| | **Redun2** | 45.9% | **85.9%** | 20.1% | **84.7%** | 39.3% | **85.5%** |
| | **Redun3** | 28.4% | 66.6% | 5.3% | 59.4% | 13.8% | 73.8% |
| | **All** | **80.7%** | **97.6%** | **64.6%** | **95.8%** | **71.0%** | **97.9%** |
| Verbs | **PAPEL** | 37.0% | 82.8% | 0.0% | 78.8% | 0.0% | 46.7% |
| | **DA** | 24.8% | 74.0% | 0.0% | 71.7% | 0.0% | 37.7% |
| | **Wikt.PT** | 18.9% | 60.9% | 0.0% | 55.1% | 3.6% | 52.7% |
| | **OWN-PT** | **84.8%** | **95.4%** | **88.4%** | **97.5%** | **86.8%** | **97.6%** |
| | **PULO** | 24.4% | 41.6% | 24.7% | 46.0% | 40.1% | 59.9% |
| | **TeP** | 53.1% | 76.8% | 0.0% | 69.7% | 47.9% | 79.0% |
| | **OT.PT** | 25.1% | 43.0% | 0.0% | 35.4% | 0.0% | 24.6% |
| | **Port4Nooj** | 0.0% | 17.7% | 0.0% | 19.2% | 0.0% | 22.8% |
| | **WN.Br** | 47.6% | 73.1% | 32.3% | 74.2% | 0.0% | 44.9% |
| | **Redun2** | 63.4% | 88.0% | 43.9% | 86.9% | 56.9% | 84.4% |
| | **Redun3** | 53.6% | 82.1% | 9.6% | 79.8% | 25.7% | 65.9% |
| | **All** | **92.9%** | **98.4%** | **91.9%** | **99.0%** | **94.6%** | **97.6%** |

In addition, since SimLex-999 is a similarity test, the previous methods were tested using all the relations of each LKB, or only synonymy and hypernymy relations, which are more connected with this phenomena.

The obtained results were evaluated with the Spearman correlation between the similarities in SimLex-999 and the similarities computed from each of the previous methods in each LKB. Table 10 shows the best results for each combination of method, relations used, and LKB, as well as different methods for the LKB with the best results (*All*).

Results show that LKBs extracted from dictionaries have better results with PageRank-based algorithms, using all relations, while LKBs extracted from wordnets have better results with adjacency-based algorithms, using only synonymy and hypernymy relations. The best results are clearly obtained with the combination of all LKBs, using different configurations (0.56–0.60). The original LKB with the best performance is PAPEL (0.49), which performed slightly better than *Redun2* (0.48). PAPEL was followed by OWN-PT (0.44) and Wiktionary.PT (0.42), both better than *Redun3*.

Although the top result is obtained with a PageRank-based algorithm, adjacency-based similarity is close, and even higher for some LKBs. It should thus be seen as a valuable alternative, especially because PageRank-based algorithms are either time (complexity of running PageRank) or memory-expensive (ranks can be pre-computed, but large matrices

**Table 9** First two adjectives, nouns and verbs of the Portuguese SimLex-999.

| Word 1 | Word 2 | POS | Similarity |
|---|---|---|---|
| velho | novo | A | 0.00 |
| esperto | inteligente | A | 8.33 |
| esposa | marido | N | 5.00 |
| livro | texto | N | 5.00 |
| ir | vir | V | 3.33 |
| levar | roubar | V | 6.67 |

are required). As for the size of the vectors, there is no clear trend, except that the best result is never obtained with the larger sizes tested (1,600 and 3,200). Further discussion of the best methods is out of the scope of this paper.

Although languages are different and so are the available resources, a final word should be given on the comparison of our results with the top state-of-the-art results for English, as reported in the ACL Wiki[4]. By combining distributional vectors with knowledge from Princeton WordNet, a Spearman coefficient of 0.642 was obtained for the English SimLex-999 [2], which is not very far from the results of our best configuration (0.60).

## 5.3 Answering Cloze Questions

Open domain cloze questions have been generated in the scope of REAP.PT [5], an assisted language learning tutoring system for European Portuguese. Those consist of sentences with a blank, to be filled with a word from a shuffled list of candidates, of which only one is correct and the other are distractors. Some of the Portuguese LKBs have previously been exploited [14] to answer a set of 3,890 of those questions, provided by the researchers involved in the REAP.PT project. Table 11 illustrates the contents of this dataset with the first two questions and the respective set of candidate words, with the correct answer in bold.

The experiment reported here used the same dataset, this time answered with each of the LKBs explored in this work. The selection method was similar to the one used for the $B^2SG$ test (Section 5.1): for each question, answers were guessed from the top-ranked candidate, after running Personalized PageRank, this time using all the open-class words as context.

Table 12 shows the accuracy of selecting the correct answer, for each LKB, and with a baseline that selects the most frequent alternative, based on the frequency lists of the AC/DC corpora [28]. When no alternative was covered by the LKB, the answer would contain all the alternatives (25% correct).

Although all LKBs performed better than random chance (25%), this revealed to be a challenging task. WN.Br was just slightly higher than this number, possibly because it only covers verbs. Other LKBs were around the frequency baseline and the highest rate of correct answers (≈40%) was obtained with the *All* LKB. If using such a large LKB (≈791,000 relation instances) is not an option, PAPEL (≈191,000) or *Redun2* (≈145,000) answer ≈38% of the questions correctly.

[4] `https://www.aclweb.org/aclwiki/index.php?title=SimLex-999_(State_of_the_art)` (April 2017)

**Table 10** Selection of results for the SimLex-999 test.

| LKB | Relations | Algorithm | Spearman |
|---|---|---|---|
| **PAPEL** | All | PR-Jac$_{800}$ | 0.49 |
| **DA** | All | PR-Jac$_{400}$ | 0.38 |
| **Wikt.PT** | All | PR-Jac$_{1600}$ | 0.42 |
| **OWN-PT** | Syn+Hyp | Adj-Cos | 0.44 |
| **PULO** | Syn+Hyp | Adj-Cos | 0.29 |
| **TeP** | Syn+Hyp | Adj-Jac | 0.36 |
| **OT.PT** | Syn+Hyp | Adj-Cos | 0.34 |
| **Port4Nooj** | All | Adj-Jac | 0.19 |
| **WN.Br** | Syn+Hyper | Adj-Jac | 0.04 |
| **Redun2** | Syn+Hyper | PR-Jac$_{50}$ | 0.48 |
| **Redun3** | Syn+Hyper | Adj-Jac | 0.41 |
| **All** | Syn+Hyper | PR-CosV$_{400}$ | **0.60** |
| **All** | Syn+Hyper | PR-CosV$_{50}$ | 0.56 |
| **All** | Syn+Hyper | PR-CosV$_{100}$ | 0.58 |
| **All** | Syn+Hyper | PR-CosV$_{200}$ | 0.59 |
| **All** | Syn+Hyper | PR-CosV$_{800}$ | 0.59 |
| **All** | Syn+Hyper | PR-CosV$_{1600}$ | 0.59 |
| **All** | Syn+Hyper | PR-CosV$_{3200}$ | 0.59 |
| **All** | Syn+Hyper | Adj-Cos | 0.57 |
| **All** | Syn+Hyper | Adj-Jac | 0.56 |
| **All** | All | PR-CosV$_{400}$ | 0.57 |

**Table 11** First two cloze questions of the dataset used.

| | | | | |
|---|---|---|---|---|
| **Sentence** | *A instalação de «superpostos» nas entradas e saídas dos grandes _____ urbanos levanta, por outro lado, algumas dúvidas à Anarec.* | | | |
| **Candidates** | **centros** | mecanismos | inquéritos | indivíduos |
| **Sentence** | *O artista _____ uma verdadeira obra de arte.* | | | |
| **Candidates** | **criou** | emigrou | requereu | atribuiu |

## 5.4 Textual Similarity and Entailment

The ASSIN shared task targeted semantic similarity and textual entailment in Portuguese [12]. Its training data comprises 6,000 sentence pairs ($t$, $h$), half of which in Brazilian Portuguese (PTBR) and the other half in European Portuguese (PTPT). Test data comprises 4,000 pairs, 2,000 in each variant. Data is available in the task's website[5], together with the gold annotations of the test data and evaluation scripts. Similarity values range from 1 (completely different sentences, on different subjects) to 5 ($t$ and $h$ mean essentially the same). Entailment can have the value *Paraphrase*, *Entailment* or *None*. Table 13 shows a selection of sentence pairs in the ASSIN training collection.

LKBs were exploited to compute similarity according to equation 1. After preprocessing the sentences and computing the cosine of their stems, a bonus ($\gamma$) was added for each additional word from $t$ directly related to a word in $h$ ($\gamma$+=0.75) or related to a common word ($\gamma$+=0.05).

$$Sim(S_1, S_2) = \frac{|S_1 \cap S_2| + \gamma}{\sqrt{|S_1|}\sqrt{|S_2|}}\,. \tag{1}$$

[5] `http://nilc.icmc.usp.br/assin/` (April 2017)

**Table 12** Accuracy for answering cloze questions.

| | **Accuracy** |
|---|---|
| *Baseline* (frequency) | 32.83% |
| **PAPEL** | **38.53%** |
| **DA** | 34.77% |
| **Wikt.PT** | 36.13% |
| **OWN-PT** | 33.25% |
| **PULO** | 33.25% |
| **TeP** | 35.53% |
| **OT.PT** | 30.24% |
| **Port4Nooj** | 31.93% |
| **WN.Br** | 26.07% |
| **Redun2** | 38.05% |
| **Redun3** | 35.35% |
| **All** | **40.57%** |

A very simple approach was followed for the entailment task. Common words and synonyms were first removed from the longer sentence. If the proportion of remaining words was below $\alpha = 0.1$, the pairs would be classified as a Paraphrase. After this, words from the first sentence in an hypernymy relation with words from the second were also removed. If the proportion of remaining words was below $\beta = 0.45$, the pair would be classified as Entailment. Parameters $\alpha$ and $\beta$ were set after several experiments in the training collection.

Table 14 shows the obtained results for the PTPT and PTBR variants, with each LKB, plus a baseline that does not use a LKB ($\alpha = \beta = 0$), and the best official results of ASSIN. Entailment performance is scored in terms of accuracy and Macro-F1, while similarity resorts to the Pearson correlation and the mean square error (MSE).

In this task, the performance of using different LKBs does not vary significantly and no strong conclusions can be taken, as the cosine seems to play a greater role. To reach the best performances, LKB features would have to be combined with other, possibly in a supervised approach, where the weights for each feature would be learned during the training phase. This is how most participating systems approached ASSIN, including the best results.

Despite the previous remark, in opposition to the cloze questions, in this case, using the *All* LKBs leads to the lowest results is most scores, possibly due to the noise in such a large LKB, and also due to the different method applied.

## 6 Concluding remarks

Open Portuguese LKBs were briefly overviewed in this paper, with a focus on size and redundancy across them. Despite sharing a similar goal, these LKBs were created by different teams, following different approaches, and there are significant differences in the covered lexical items, relations, their correctness or utility. The creation of new LKBs by combining the existing ones was described and all LKBs were then compared indirectly, when exploited in different computational semantics tasks.

This comparison confirmed the limitations of some LKBs, especially those with a limited size (Port4Nooj, OT.PT), or the ones focused on a single POS (WN.Br) or relation (OT.PT). Except for the expected impact of those limitations, obtained results are positive for every LKB, especially in the word-based similarity tests. This is a preliminary comparison and

**Table 13** Selected examples from the ASSIN training collection.

| Variant | Id | Pair | | Sim | Entailment |
|---|---|---|---|---|---|
| PTPT | 2675 | **t** | *O Chelsea só conseguiu reagir no final da primeira parte.* | 1.25 | None |
| | | **h** | *Não podemos aceitar outra primeira parte como essa.* | | |
| PTBR | 319 | **t** | *Cerca de 10% da Grande Muralha da China já desapareceu.* | 2.5 | None |
| | | **h** | *Em 2006, a China estabeleceu regulamentos para a proteção da Grande Muralha.* | | |
| PTPT | 315 | **t** | *Todos que ficaram feridos e os mortos foram levados ao hospital.* | 3.0 | None |
| | | **h** | *Além disso, mais de 180 pessoas ficaram feridas.* | | |
| PTBR | 2982 | **t** | *Maldonado disse ainda que cerca de 125 casas foram afetadas pelo deslizamento.* | 4.0 | Entailment |
| | | **h** | *Segundo Maldonado, mais de 100 casas podem ter sido atingidas.* | | |
| PTBR | 1282 | **t** | *As multas previstas nos contratos podem atingir, juntas, 23 milhões de reais.* | 5.0 | Paraphrase |
| | | **h** | *Somadas, as multas previstas nos contratos podem chegar a R$ 23 milhões.* | | |

**Table 14** Exploiting LKBs in the ASSIN test set.

| | PTPT | | | | PTBR | | | |
|---|---|---|---|---|---|---|---|---|
| **Config** | **Entailment** | | **Similarity** | | **Entailment** | | **Similarity** | |
| | **Acc** | **F1** | **Pearson** | **MSE** | **Acc** | **F1** | **Pearson** | **MSE** |
| *Baseline (cosine)* | 74.10% | 0.43 | 0.66 | 0.66 | 78.60% | 0.43 | 0.65 | 0.445 |
| *Best PTPT* | 83.85% | 0.70 | 0.73 | 0.61 | – | – | – | – |
| *Best sim PTBR* | – | – | 0.70 | 0.66 | – | – | 0.70 | 0.38 |
| *Best entail PTBR* | 77.60% | 0.61 | 0.64 | 0.72 | 81.65% | 0.52 | 0.64 | 0.45 |
| **PAPEL** | 74.30% | 0.45 | 0.67 | 0.70 | 78.25% | 0.45 | 0.66 | 0.44 |
| **DA** | 74.10% | 0.44 | 0.67 | 0.69 | **78.50%** | 0.44 | 0.66 | **0.43** |
| **Wikt.PT** | 74.00% | 0.44 | 0.67 | **0.68** | 77.55% | 0.43 | 0.66 | **0.43** |
| **OWN-PT** | 73.80% | 0.45 | 0.67 | 0.71 | 77.30% | 0.43 | 0.66 | **0.43** |
| **PULO** | 74.00% | 0.45 | 0.66 | 0.74 | 76.80% | 0.45 | 0.66 | 0.45 |
| **TeP** | 74.55% | **0.47** | 0.67 | 0.71 | 77.90% | 0.47 | **0.67** | 0.45 |
| **OT.PT** | 74.05% | 0.44 | 0.67 | **0.68** | 78.40% | 0.44 | 0.66 | **0.43** |
| **Port4Nooj** | 73.85% | 0.43 | 0.66 | **0.68** | 78.10% | 0.43 | 0.66 | 0.44 |
| **WN.Br** | 74.20% | 0.45 | 0.66 | 0.71 | 77.50% | 0.44 | 0.66 | 0.45 |
| **Redun3** | **74.70%** | **0.47** | **0.68** | 0.69 | 78.05% | 0.46 | **0.67** | 0.45 |
| **Redun2** | 74.20% | **0.47** | 0.67 | 0.72 | 77.65% | 0.47 | **0.67** | 0.44 |
| **All** | 73.00% | **0.47** | 0.66 | 0.69 | 75.90% | **0.48** | 0.65 | 0.45 |

further analysis is needed for stronger conclusions. But results suggest that using a LKB with knowledge from all the others is generally the best solution. Due to the large size of this solution, in some cases, it might be worth using a LKB containing only relations in two or three LKBs, depending on the task. With the later solution, the negative impact on performance is higher for algorithms based on the structure of the network, such as PageRank, and not so much on approaches that do not go one level further than the direct adjacencies. This happens because PageRank exploits every link in the network structure, some of which are not redundant and thus missing from the redundancy-based LKBs. Even though the aforementioned conclusions are still valid for the sentence-oriented tests, additional features and more sophisticated approaches would be required for a higher performance.

All the nine LKBs compared in this work were exploited in the creation of new version of the fuzzy Portuguese wordnet CONTO.PT [16], to be released in the future, and the redundancy-based LKBs are freely available for anyone to use[6]. In the future, we aim at using these LKBs in additional tasks, or in the same but focusing on certain aspects, such as the POS. Yet, a manual evaluation might be required for stronger conclusions. It is also in our plans to compare the performance of some of these LKBs and of the algorithms used here with the performance of models of distributional similarity for Portuguese. Although created from different methods – theoretical views on the mental lexicon *vs* distribution of words in a corpus – models such as word embeddings [23] are a recent trend in many NLP tasks, including computing semantic similarity.

## References


1. Eneko Agirre and Aitor Soroa. Personalizing PageRank for word sense disambiguation. In *12th Conference of the European Chapter of the Association for Computational Linguistics*, pages 33–41, 2009.
2. Rajendra Banjade, Nabin Maharjan, Nobal B. Niraula, Vasile Rus, and Dipesh Gautam. Lemon and tea are not similar: Measuring word-to-word similarity by combining different methods. In *16th International Conference on Computational Linguistics and Intelligent Text Processing (CICLing)*, pages 335–346, 2015.
3. Anabela Barreiro. Port4NooJ: an open source, ontology-driven portuguese linguistic system with applications in machine translation. In *2008 International NooJ Conference*, pages 19–47, 2010.
4. Francis Bond and Ryan Foster. Linking and extending an open multilingual Wordnet. In *51st Annual Meeting of the Association for Computational Linguistics*, volume 1, pages 1352–1362, August 2013.
5. Rui Correia, Jorge Baptista, Maxine Eskenazi, and Nuno Mamede. Automatic generation of cloze question stems. In *10th International Conference on Computational Processing of the Portuguese Language (PROPOR)*, pages 168–178, April 2012.
6. Gerard de Melo and Gerhard Weikum. Towards a universal wordnet by learning from combined evidence. In *18th ACM Conference on Information and Knowledge Management (CIKM)*, pages 513–522, 2009.
7. Valeria de Paiva, Alexandre Rademaker, and Gerard de Melo. OpenWordNet-PT: An Open Brazilian WordNet for Reasoning. In *24th International Conference on Computational Linguistics*, pages 353–360, 2012.


[6] Check `http://ontopt.dei.uc.pt/index.php?sec=download_outros`.

8 Valeria de Paiva, Livy Real, Hugo Gonçalo Oliveira, Alexandre Rademaker, Cláudia Freitas, and Alberto Simões. An overview of Portuguese wordnets. In *8th Global WordNet Conference*, pages 74–81, 2016.

9 Bento C. Dias-da-Silva. Wordnet.Br: An exercise of human language technology research. In *3rd International WordNet Conference (GWC)*, pages 301–303, January 2006.

10 Douglas Downey, Oren Etzioni, and Stephen Soderland. A probabilistic model of redundancy in information extraction. In *19th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1034–1041, 2005.

11 Christiane Fellbaum, editor. *WordNet: An Electronic Lexical Database*. Language, Speech, and Communication. The MIT Press, 1998.

12 Erick Rocha Fonseca, Leandro Borges dos Santos, Marcelo Criscuolo, and Sandra Maria Aluísio. Visão geral da avaliação de similaridade semântica e inferência textual. *Linguamática*, 8(2):3–13, 2016.

13 Dayne Freitag, Matthias Blume, John Byrnes, Edmond Chow, Sadik Kapadia, Richard Rohwer, and Zhiqiang Wang. New experiments in distributional representations of synonymy. In *9th Conference on Computational Natural Language Learning*, pages 25–32, 2005.

14 Hugo Gonçalo Oliveira, Inês Coelho, and Paulo Gomes. Exploiting Portuguese lexical knowledge bases for answering open domain cloze questions automatically. In *9th Language Resources and Evaluation Conference (LREC)*, May 2014.

15 Hugo Gonçalo Oliveira, Diana Santos, Paulo Gomes, and Nuno Seco. PAPEL: A dictionary-based lexical ontology for Portuguese. In *8th International Conference on Computational Processing of the Portuguese Language (PROPOR)*, pages 31–40, September 2008.

16 Hugo Gonçalo Oliveira. CONTO.PT: Groundwork for the automatic creation of a fuzzy portuguese wordnet. In *12th International Conference on Computational Processing of the Portuguese Language (PROPOR)*, pages 283–295, July 2016.

17 Iryna Gurevych, Judith Eckle-Kohler, Silvana Hartmann, Michael Matuschek, Christian M. Meyer, and Christian Wirth. UBY – a large-scale unified lexical-semantic resource. In *13th Conference of the European Chapter of the Association for Computational Linguistics*, pages 580–590, April 2012.

18 Marti A. Hearst. Automated discovery of WordNet relations. In Christiane Fellbaum, editor, *WordNet: An Electronic Lexical Database*, Language, Speech, and Communication, pages 131–151. The MIT Press, 1998.

19 Felix Hill, Roi Reichart, and Anna Korhonen. Simlex-999: Evaluating semantic models with genuine similarity estimation. *Computational Linguistics*, 41(4):665–695, December 2015.

20 Bernardo Magnini and Gabriela Cavaglià. Integrating subject field codes into WordNet. In *2nd International Conference on Language Resources and Evaluation (LREC)*, pages 1413–1418, 2000.

21 Palmira Marrafa. Portuguese WordNet: general architecture and internal semantic relations. *DELTA*, 18:131–146, 2002.

22 Erick Maziero, Thiago Pardo, Ariani Di Felippo, and Bento Dias-da-Silva. A base de dados lexical e a interface web do TeP 2.0 – Thesaurus Eletrônico para o Português do Brasil. In *VI Workshop em Tecnologia da Informação e Linguagem Humana*, pages 390–392, 2008.

23 Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. In *Workshop Track of the International Conference on Learning Representations (ICLR)*, 2013.

24 Roberto Navigli and Simone Paolo Ponzetto. BabelNet: The automatic construction, evaluation and application of a wide-coverage multilingual semantic network. *Artificial Intelligence*, 193:217–250, 2012.

**25** Emanuele Pianta, Luisa Bentivogli, and Christian Girardi. MultiWordNet: developing an aligned multilingual database. In *1st International Conference on Global WordNet*, pages 293–302, 2002.

**26** Mohammad Taher Pilehvar, David Jurgens, and Roberto Navigli. Align, disambiguate and walk: A unified approach for measuring semantic similarity. In *51st Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 1341–1351, 2013.

**27** João António Rodrigues, António Branco, Steven Neale, and João Ricardo Silva. LX-DSemVectors: Distributional semantics models for Portuguese. In *12th International Conference on the Computational Processing of the Portuguese Language (PROPOR)*, pages 259–270, 2016.

**28** Diana Santos and Eckhard Bick. Providing internet access to Portuguese corpora: the AC/DC project. In *2nd International Conference on Language Resources and Evaluation (LREC)*, pages 205–210, 2000.

**29** Lei Shi and Rada Mihalcea. Putting pieces together: Combining FrameNet, VerbNet and WordNet for robust semantic parsing. In *Computational Linguistics and Intelligent Text Processing (CICLing)*, pages 100–111, 2005.

**30** Alberto Simões and Xavier Gómez Guinovart. Bootstrapping a Portuguese wordnet from Galician, Spanish and English wordnets. In *Advances in Speech and Language Technologies for Iberian Languages*, volume 8854 of *LNCS*, pages 239–248, 2014.

**31** Alberto Simões, Álvaro Iriarte Sanromán, and José João Almeida. Dicionário-Aberto: A source of resources for the Portuguese language processing. In *10th International Conference on Computational Processing of the Portuguese Language (PROPOR)*, pages 121–127, April 2012.

**32** Piek Vossen. EuroWordNet: a multilingual database for information retrieval. In *DELOS workshop on Cross-Language Information Retrieval*, 1997.

**33** Rodrigo Wilkens, Leonardo Zilio, Eduardo Ferreira, and Aline Villavicencio. B2SG: a TOEFL-like task for Portuguese. In *10th International Conference on Language Resources and Evaluation (LREC)*, pages 3659–3662, May 2016.

# An Emotional Word Analyzer for Portuguese*

**Maria Inês Maia[1] and José Paulo Leal[2]**

1 **CRACS & INESC-Porto LA, Faculty of Sciences, University of Porto, Porto, Portugal**
up201101593@fc.up.pt

2 **CRACS & INESC-Porto LA, Faculty of Sciences, University of Porto, Porto, Portugal**
zp@dcc.fc.up.pt

## Abstract

The analysis of sentiments, emotions and opinions in texts is increasingly important in the current digital world. The existing lexicons with emotional annotations for the Portuguese language are oriented to polarities, classifying words as positive, negative or neutral. To identify the emotional load intended by the author it is necessary also to categorize the emotions expressed by individual words.

EmoSpell is an extension of a morphological analyzer with semantic annotations of the emotional value of words. It uses Jspell as the morphological analyzer and a new dictionary with emotional annotations. This dictionary incorporates the lexical base EMOTAIX.PT, which classifies words based on three different levels of emotions – global, specific and intermediate.

This paper describes the generation of the EmoSpell dictionary using three sources, the Jspell Portuguese dictionary and the lexical bases EMOTAIX.PT and SentiLex-PT. Also, this paper details the web application and web service that exploit this dictionary. It presents also a validation of the proposed approach using a corpus of student texts with different emotional loads. The validation compares the analyses provided by EmoSpell with the mentioned emotional lexical bases on the ability to recognize emotional words and extract the dominant emotion from a text.

**1998 ACM Subject Classification** H.3.1 [Content Analysis and Indexing] Dictionaries

**Keywords and phrases** Sentiment Analysis, Opinion Mining, Emotion API

**Digital Object Identifier** 10.4230/OASIcs.SLATE.2017.17

## 1 Introduction

Sentiment Analysis, also known as Opinion Mining, can be classified as the identification of opinions, emotions and evaluation in texts [22] toward topics, events, entities or individuals. A frequent approach to perform this kind of analysis is to use dictionaries containing words annotated with semantic or polarity orientation. These dictionaries can be created manually or by using seed words, thus expanding automatically the list of words [20].

This paper describes the implementation of EmoSpell[1], an emotional word Analyzer for the Portuguese language. The motivation for EmoSpell comes from the research on the

* This work is financed by BIAL, trough project M-BW, BIAL 312/16, the ERDF – European Regional Development Fund through the Operational Programme for Competitiveness and Internationalisation – COMPETE 2020 Programme, by National Funds through the FCT – Fundação para a Ciência e a Tecnologia (Portuguese Foundation for Science and Technology) within project POCI-01-0145-FEDER-006961, and by FourEyes, a Research Line within project TEC4Growth – Pervasive Intelligence.

[1] Available at `http://daar.up.pt:8080/EmoSpell/`.





cognitive processes in writing. This kind of research is based on a model that views writing as the interplay of several cognitive processes [14]. Examples of these processes are: planning the text, selecting grammar constructions, checking word spelling. If one of more of these processes take that too much time then writing pauses will occur.

The goal of the research presented in this paper is to integrate EmoSpell with HandSpy[2], a web based collaborative environment for researchers working on cognitive processes in writing [12]. This system, currently in its version 2.0, is being used by 13 research teams from 10 different countries. HandSpy is a tool to detect pauses in the writing productions collected with smartpens and investigate what may have cause them. EmoSpell integrated in HandSpy will enable researchers to relate pauses in writing with the grammatical categories and emotional load of the words where these pauses occurred.

EmoSpell contains a dictionary with semantic annotations regarding the emotional value of words. It is based on a morphological analyzer named Jspell [1] and EMOTAIX.PT [7], a lexical base that catalogs a set of Portuguese words based on their emotional load. SentiLex-PT was used to compare the polarities and extend the emotional classification of words.

Jspell uses a dictionary of lemmas with a set of morphological rules. It avoids the enumeration of all words in a language by creating rules that associate them to their lemmas. Moreover, this dictionary of lemmas can be extended to contain new categories, either syntactic or semantic. Hence, when the lemmas are annotated with their emotional load as semantic categories, their flexed forms also inherit these properties. Also, it is possible to reverse the emotional category of a word, when a negation prefix is added, such as “in”, “des”, “de”. For example, for the word, “feliz” (“happy”) and “infeliz” (unhappy), it will be annotated that this emotional classification will be the opposite.

EMOTAIX.PT is a lexical base with 3992 collected words. These words are classified according to their valence, that is, positive, negative and neutral and semantic nature, that consists in a hierarchical classification of emotional categories. There are other Portuguese lexicons with sentimental annotations such as SentiLex-PT [18]. This one in particular, provides only word polarity, by classifying them as positive, negative or neutral. Therefore, EmoSpell will allow a more detailed analysis of sentiments for the Portuguese language.

EmoSpell will be useful for other purposes than the research in cognitive processes in writing and its planed integration with HandSpy. To reach wide range of potential users, EmoSpell has two kinds of interfaces: a web GUI and an API. The former is intended for end users and was instrumental for the validation of the proposed approach. The latter was designed for the integration with applications, in particular HandSpy. To increase its interoperability, the API build on open XML standards for natural language processing and emotional analysis, namely the *Text Encoding Initiative* (TEI)[3] and *Emotion Markup Language* (EmotionML) [4].

The remainder of this article is organized as follows. Section 2 reviews sentiment analysis APIs that detect emotion on texts. EmoSpell was created based on the Jspell and EMOTAIX.PT, also using the Sentilex-PT. The analyzer and the two lexical bases are detailed in Section 3. The design choices and implementation of the EmoSpell dictionary, as well as the system created to validate this analyzer, are presented in Section 4. Section 5 presents a validation of the proposed approach and the final section summarizes the contribution of this research.

[2] Available at `http://daar.up.pt/index.php/pt/handspy`.
[3] `http://www.tei-c.org/index.xml`

## 2 State of the Art

Opinion analysis has always been an important topic, either for individuals who want to know the general opinion regarding a certain products of their interest, or for businesses and organizations that want to know their customers' opinions regarding their products and services.[11]

The availability of information from *Microblogging Web-Sites* users turns this into the most famous source of data in this area. Application-oriented research papers referring to *Microblogging Web-Sites*, such as *Twitter*, are based on consumers' opinions regarding products or services, which is valuable information for companies [6]. It is also useful to gauge the political opinion of the population on a certain subject [13].

There are two main approaches to Sentiment Analysis: the classifier-based and the lexicon-based approaches. The former uses machine learning techniques, considering this analysis as a problem of text categorization. The latter uses sentiment lexicons to analyze the text and perceive the sentiments that it contains.

A sentiment lexicon can be constructed either manually or automatically. In the manual creation, there are no algorithms. It is based on the enumeration of a list of words and the annotation of the sentiment classification, using a dictionary or a corpus as resource. This annotation is made by humans so, besides the possibility of human errors, the creation time and lexicon size are also not as expected. The automatic creation of sentiment lexicon uses a set of seed words and rules or methods to expand these words.

Sentiment lexicons are created to analyze the emotional context of texts. They assign an emotional classification to words, so that when they are found in a text they provide information regarding their emotional load. This emotional load can be represented by polarities (positive, negative and neutral), values representing the emotion strength or emotional categories.

Through words, an individual can express his emotions and feelings towards a certain subject. For example, if a sentence contains the words "*like*", "*approved*" and "*good*", we understand that the position of the author towards the matter is positive. On the other hand, if the words are "*hated*", "*disapproved*" and "*bad*", the position is negative.

However, understanding the exact meaning intended by author is a challenge. Words can be used with different meanings, thus conveying different emotions. It is not as simple as to classify some words as positive and others as negative.

When writing a text or a review, it is possible that the writer uses words of an opposite emotional category opposite to the context of the text. When classifying singles words with specific emotional classification, the phrase that incorporates those words can express a different sentiment. The simplest example is the use of a positive word in a negative sentence – "*It was not great*" – despite being a negative opinion, the annotated lexicon translates this as a sentence with a positive word "*great*" [21].

Also, authors are not express their emotions and feeling straightforwardly. For example, the use of sarcasm in sentences is a problem in the emotional analysis of texts. In the sentence "*I Absolutely adore it when the bus is late*", it is used the positive word "*adore*", but sarcastically, to highlight a negative situation [17].

There are lexicons available for several languages. For Portuguese, the already mentioned SentiLex-PT is explained in more detail in the next section, since it was used to create EmoSpell. Examples of these lexicons are the following:

- **Linguistic Inquiry and Word Count (LIWC)** – uses a dictionary of words and word stems, each filed into one or more sub-dictionaries. It classifies words in psychological relevant categories. [15] A similar tool is **EMOTAIX**, but for the French language. [16]

- **SentiWordNet** – it is an extension of **WordNet**. WordNet uses a dictionary containing nouns, adjectives, verbs and adverbs that can be called *"synsets"*. [9] SentiWordNet added three sentiment values to each "*synset*", that is, a "*synset*" will have a positive, negative and neutral score. [8]
- **Sentiment Orientation CALculator (SO-CAL)** – is a system that contains a dictionary of annotated words with semantic orientation. SO-CAL have two assumptions – the first is that the individual words have a "prior polarity", which is a semantic orientation independent of context. The second is that this orientation can be identified with a numerical value, the strength. [20]

The lexicons are usually made available as *Sentiment Analysis API*, frequently in complement to other natural language processing features. Examples of these APIs are the following.

- **TweetSentiments** – returns the sentiment of Tweets using the supervised learning algorithm Support Vector Machine (SVM). It has two online APIs that analyze Tweets from Twitter API calls, returned by a Twitter search query[4].
- **ML Analizer** – provides several text analyzes, including feelings, text classification, language detection, locations extractor, adult content analyzer and article summarization[5].
- **WebKnox Text-Processing** – natural language processing of texts such as determining the feeling, identification of the language, the classification of the quality of writing, the auto-correction of a text, the extraction of data and locations and the tagging of a text with part-of-speech tags[6].
- **Skyttle** – provides services to extract patterns from text such as sentiment terms, constituent terms (meaningful expressions) and entities such as names of people, place and things. Supported languages are English, French, German and Russian[7].
- **Sentiment Analysis Spanish** – Sentiment analysis for tweets in Spanish[8].
- **nlpTools** – text classification and sentiment analysis for Natural language. It is an API focused on online news media[9].
- **Yactraq Speech2Topics** – converts audiovisual content into topic metadata. This conversion is done through speech recognition and natural language processing[10].

## 3 Background

EmoSpell is based on the extension of a dictionary of the morphological analyzer Jspell using the lexical base of emotional words, EMOTAIX.PT. Jspell has features that improve the efficiency of classifying words by providing a set of rules to generate words from radicals. The development of EmoSpell used also a Sentiment Lexicon called SentiLex-PT, mostly to compare the polarities of EMOTAIX.PT and to identify relevant missing words in EMOTAIX.PT. This section details these three different tools used in the development of EmoSpell.

---

[4] Available from `https://www.programmableweb.com/api/tweetsentiments`.
[5] Available from `https://market.mashape.com/mlanalyzer/ml-analyzer`.
[6] Available from `http://webknox.com/`.
[7] Available from `http://www.skyttle.com/`.
[8] Available from `https://market.mashape.com/molinodeideas/sentiment-analysis-spanish#!documentation`.
[9] Available from `http://nlptools.atrilla.net/web/`.
[10] Available from `http://yactraq.com/`.

## 3.1 Jspell

The morphological analyzer Jspell is an extension of the Ispell spell checker. Although not itself a morphological analyzer, Ispell already includes the possibility of definition and usage of elementary morphological rules [1].

Since natural language applications needed to be able to handle grammatical and semantic information of words, it is important to have a lexical classifier able to provide information on a given word. This information can be based in its origin and grammatical category. The lexical is fundamental in the parsing of these types of applications.

For that purpose, Jspell uses a dictionary, which is a list of words that are classified based on a set of formation rules. These rules functionally bring an important advantage for analyzers, which simplifies the exhaustive enumeration of all the dictionary words, thus creating a list with the lemmas of words and morphological rules. Consequently, this will associate flags to each word in the list that contains the rules that may be applied to the word [1].

The dictionary structure consists basically of entries, each one containing [19]:

- a **lemma**, which is a word from where you can get other words, by derivation or inflection and that cannot be obtained by any other lemma;
- **morphological description**, that is, a list of morphological properties that are key-value pairs of grammatical classification of lemmas that may contain macros to simplify, which will be explained below;
- **derivation rules** which is a set of identifiers of derivation or inflection rules (flags) that are defined in a separate file called affix rules.

Therefore, a typical entry in the dictionary is a line of the form

*word/classification/flags[/comment]*

The Portuguese dictionary of Jspell contains about 400 000 entries and the rules associated to it. Since dictionaries are in text format they can be easily modified. Thus, it is fairly simple to expand and to create new dictionaries with this analyzer.

To conclude, Jspell can be used in ways, ranging from an interactive web application with menus and options, to a library, It includes also a line interpreter where the user can write a word and receive the corresponding information. This interpreter can be used other programs interacting with Jspell trough pipes. Besides that, Jspell can be used as a standard library (dll/so/dylib) which can be an advantage in efficiency manners.

## 3.2 SentiLex-PT

SentiLex-PT, as the name suggests, is a sentiment lexicon which was designed to analyze the sentiment and opinion about human entities in texts written in Portuguese[11]. It contains 6.321 adjectival lemmas and 25.406 inflected forms. The lexicon entries correspond to human predicates – adjectives, nouns, verbs and idiomatic expressions. In a sentence, to classify a word based on its polarity, the target of sentiment is verified in order to identify whether it has a subject or complement function. For example, in the case of the word “fat”, as a modifier of a name of human nature, (e.g. “fat guy”), it has a negative polarity, but it has the opposite polarity when combined with a name such as salary (e.g. “fat salary”). [18]

[11] Available from `http://dmir.inesc-id.pt/project/SentiLex-PT_02`.

SentiLex-PT includes two dictionaries: *SentiLex-lem*, which describes the lemmas and the *SentiLex-flex* which is the dictionary that corresponds to the inflected forms. In the Sentilex-lem [5], each line includes:

- **Lemma;**
- **Grammar Category** (adjective, noun, verb or idiom);
- **Sentiment Attributes** (polarity and target of polarity which corresponds to a human subject, being N0 the subject and N1 the complement).

For instance, one entry of SentiLex-Lem is:

```
enganar.PoS=V;TG=HUM:N0:N1;POL:N0=-1;POL:N1=0;ANOT=MAN
```

In *SentiLex-flex*, the entries are associated to the lemma and in addition to the information contained in the *SentiLex-lem*, in this format it also describes information about inflection like gender and number.

Although it was an important step in the sentiment analysis of Portuguese texts, the SentiLex-PT classifies each word through polarity – the word can be negative, positive or neutral. So, as will be detailed in the description of EMOTAIX.PT, the last dictionary will allow a much more detailed sentiment analysis, because each word will not only be classified based on three polarities but by categories of sentiment as well.

## 3.3 EMOTAIX.PT

Tools have been created in order to classify words according to their emotional load and to automate the vocabulary analysis process used in the writing of texts.

In 2001, Pennebaker, Francis and Booth developed the *Linguistic Inquiry and Word Count* (LIWC) [15], that divides English words in psychological relevant categories. In 2009, *Piolat and Bannour* created a similar tool – EMOTAIX [16] – for the French lexicon, which were explained in more detail in the previous section. Due to the relevance of these tools and the absence of a lexical database with emotional words in European Portuguese, Sara Costa created an EMOTAIX adaptation for this lexicon/language – the EMOTAIX.PT. [7]

EMOTAIX.PT is based on a database of 3992 words, classified according to their valence (positive/negative) and semantic nature. Therefore, there is a main division of words in positive and negative. Negative emotions are divided in three broad categories: "*Malevolência*" (Malevolence), "*Mal-estar*" (Malaise) and "*Ansiedade*" (Anxiety). These categories are further divided in basic categories. In addition to categories with positive and negative valence, EMOTAIX.PT is still constituted by three additional categories of free valence: *Surprise*, *Indifference* and *Non-Specified*. For a better understanding, the figure 1 below represents graphically the different levels of organization:

With this, the EMOTAIX.PT, consists in 2*25 basic categories (center) organized in three hierarchical levels, on each side of a hedonic axis (positive and negative valence), that is, for a given category, if we draw a diagonal line, we can obtain the opposite category at the end of that same line.

# 4 Design and Implementation

The core of EmoSpell is an extension of the Jspell dictionary with the emotional annotations, accessible via two interfaces for word and text analysis. This section explains the dictionary generation procedure, presents the design of the generator, illustrates the kind of analysis provided by EmoSpell, and describes the main features of its user and application interfaces.

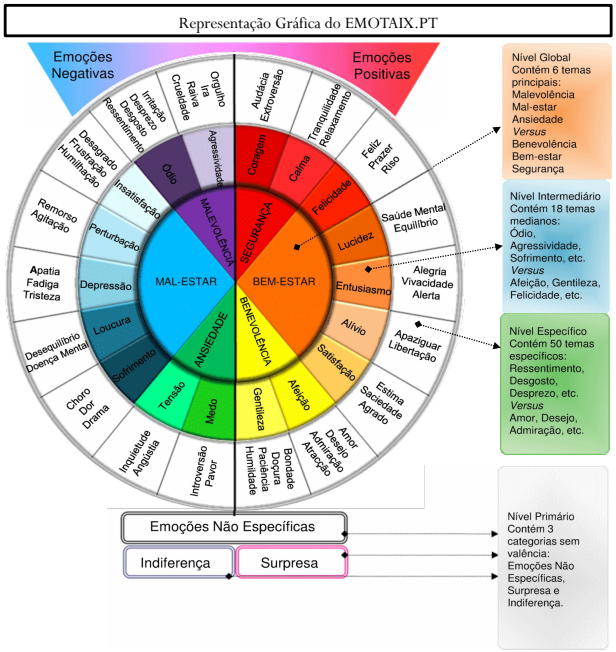



**Figure 1** EMOTAIX.PT – Levels of organization (source: [7], in Portuguese).

## 4.1 Dictionary generation procedure

The EmoSpell dictionary was developed using a Java program that imports three different dictionaries – Jspell, EMOTAIX.PT and SentiLex-PT – and stores them in memory using a common format. These sources are processed to generate the EmoSpell dictionary.

Since the EmoSpell dictionary is created from multiple sources, inconsistencies should be expected. The words that occur in multiples sources where verified for common features. In particular, EMOTAIX.PT and SentiLex-PT were checked against each other looking for inversions in polarity. Also, words in SentiLex-PT missing in EMOTAIX.PT are reported for future inclusion in this lexical base.

The EmoSpell dictionary extends the Jspell dictionary with a new type of classification of words for emotional value. This classification comes from the EMOTAIX.PT that classifies a word in three emotional category levels: global, intermediate and specific.

To add this emotional classification,there were created new 52 macros in the new Jspell dictionary, representing all the possible emotional classifications. A macro entry example is: `"#E26/EmoGlobal=bem estar,EmoIntermediate=entusiasmo,EmoSpecific=alegria`" which corresponds to the classification "*well being, enthusiasm, joy*".

This Java program can be represented by the UML activity diagram of the figure 2.

All EMOTAIX.PT words were compared with Jspell words, both the lemmas and the flexed words. If an emotional word from EMOTAIX.PT is a lemma in the Jspell dictionary then it is only necessary to add the emotional classification for this entry. Otherwise, there are several cases to consider.

If the word being analyzed is not a lemma, this means that either the word does not exist for Jspell or it is flexed from a lemma. In the latter, the generated words inherit

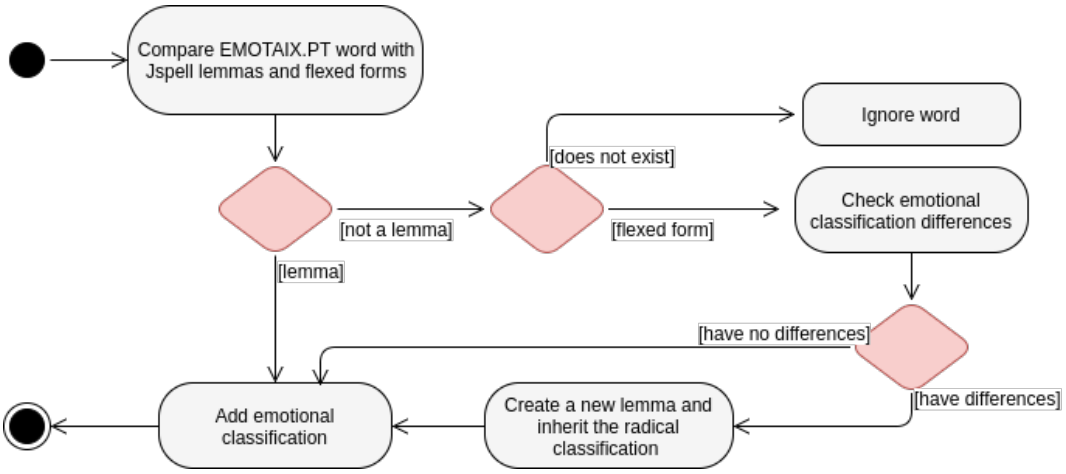



**Figure 2** Dictionary Generation – UML Activity Diagram.

the classifications from their radicals. However, if the emotional classification from the EMOTAIX.PT is different between the radical and the flexed forms, then these differences have to be added. A new lemma is created in the Jspell dictionary with the EMOTAIX.PT classification. For example, in the case of the radical "*terror*" and the flexed form "*terrorismo*" ("terrorism"), it was necessary to create a new lemma for the generated word. The emotional classification of the radical is "dread", but for the generated word "*terrorism*" is "cruelty".

For these flexed forms, instead of creating a new lemma, it would be possible to change the Jspell affix file, adding a new classification for the flexed forms. This would be the best approach if we only classified words as polarities. Since we have various classifications, this option would depend on a post-processing. So, when creating this new flexed form entry it is necessary to remove the flag from the copied lemma entry due to the various forms that it generates.

A particular case is when the emotional classification of the radical is the opposite of the generated word. For instance "*feliz*" ("*happy*") and "*infeliz*" ("*unhappy*"). In Jspell, "*feliz*" is the lemma, an entry in the dictionary, and "*infeliz*" the generated word. This is possible due to the existence of a rule that associates all the prefixes that can be added to a word to create their opposite. Being these two words associated, their classifications are the same, so, if we had the emotional classification to "*happy*", "*unhappy*" would also inherit this emotional value which is not true. This problem was solved by adding a new lemma for the opposite word in the dictionary. The new lemma has the categories of its opposite in the original Jspell dictionary and the emotional categories for that word in EMOTAIX.PT.

Another particular case are the irregular verbs. The Jspell dictionary contains entries with the each verbal forms since, by definition, it is not possible to flex them from the radical. If the word from EMOTAIX.PT is a verbal form of an irregular verb, it is necessary to also add to the other forms the emotional classification.

Given that the file to be used in the Jspell is a dictionary of lemmas, after building the new analyzer with the generated file and the affix file of morphological rules, the flexed forms of the annotated words also inherit the emotional classification.

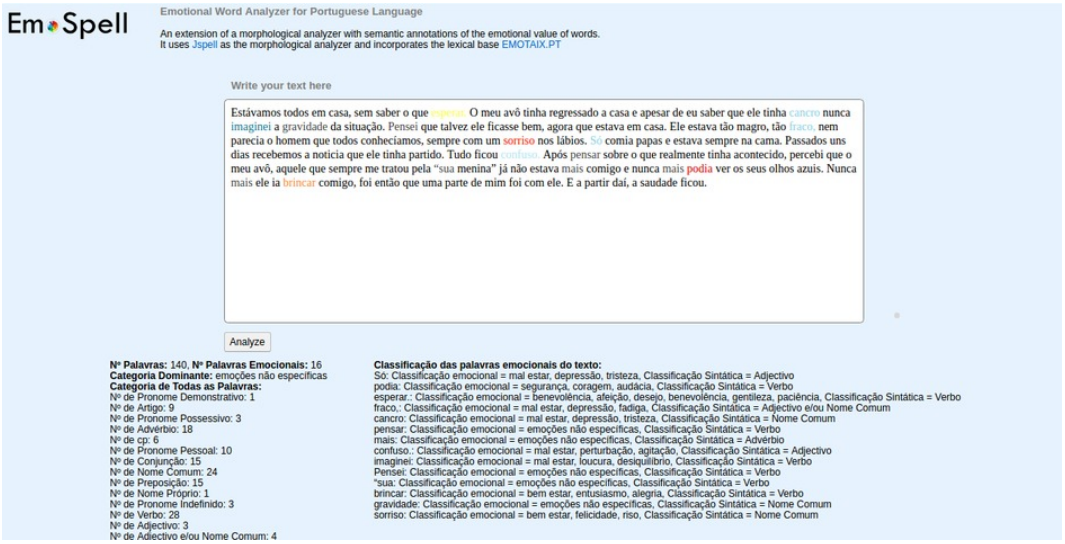



**Figure 3** GUI Screenshot – Analyzed text with classification information.

## 4.2 Text Analysis

The extended dictionary enables EmoSpell to analyze words and texts both with syntactic and emotional categories. This subsection presents an example of the output of analyzer with the extended dictionary, and explains the morphological and emotional description.

When a word is analyzed by EmoSpell, the syntactic information from Jspell is mixed with emotional classification from EMOTAIX.PT. Herewith, an example of this word analysis for the "*medo*" (fear) word, using the command line interface of Jspell.

```
medo

* medo 0 :lex(medo [CAT=nc,G=m,N=s,EmoGlobal=ansiedade,
EmoIntermediate=medo,EmoSpecific=pavor], [], [], [])
```

The first line echoes the user's input, the word to be analyzed. The following are EmoSpell's classification of the word. The first three classifications ( "`CAT=nc,G=m,N=s`") are based on the morphological description of the word and, in this case, "`medo`" is a common name, male gender and singular name. The following are the emotional categories levels, "`EmoGlobal=ansiedade`" correspond to the global emotional category (anxiety), "`EmoIntermediate=medo`" is the intermediate level (fear) and "`EmoSpecific=pavor`" to the most specific level is "`pavor`" (dread).

## 4.3 Interfaces

Although accessible from the command line as a Jspell dictionary, as shown in the previous subsection, EmoSpell has two interfaces for word and text analysis: a GUI (*Graphical User Interface*) and an API (*Application Programming Interface*). The GUI offers direct interaction with end user via a web browser, and the API enables the integration with remote applications using web services.

The EmoSpell Web Application was developed with GWT (*Google Web Toolkit*). Client server communication relies on the GWT RPC (*Remote Procedure Calls*) framework. Figure 3 is a screen shot of the web application. It contains a rich text area where users can write the

text they want to have analyzed. The submitted text is classified as a "bag of words" doing the linguistic and emotional analysis of each word. Also, the text is annotated in place with the result of words by word analysis, followed by a summary of the text analysis.

In the analyzed text, emotional words are formatted with the colors of the categories assigned by EMOTAIX.PT, shown in Figure 1. This provides a simple and immediate understanding of the text emotional load.

The panel below the text displays the general information such as number of emotional words, the dominant category, the morphological and emotional classification of the emotional words.

The API exposes as a web service the functions of EmoSpell's server that are invoked by RPC from the web client. The API follows a RESTfull architectural model, with a single function for analyzing texts. The request is implemented as an HTTP POST method that receives the text an HTTP parameter. The response is a XML document with EmoSpell's analysis.

The document type of the response builds on EmotionML (*Emotion Markup Language*) [4] and TEI (*Text Encoding Initiative*) [10]. EmotionML is a W3C recommendation to represent emotions [3]. This markup language consists of a root document, with `<emotionml>` annotation that contain one or more `<emotion>` elements that represent the emotional classification and can have more elements like category, action-tendency and dimension. EmotionML fragments can also be embed in documents of other languages. TEI is a much more complex XML norm than EmotionML and only a small part of it is actually used by EmoSpell, namely feature structures (`<fs>`), features (`<s>`), segments (`<seg>`) and choices (`<choice>`), which are enough to represent the syntactic categories of words. These TEI elements can also be mixed with elements from other types.

The XML Schema definition (XSD) of EmoSpell's API responses combines elements from the EmotionML and TEI. It defines a minimal structure to bind elements from the imported types: a sequence of `<word>` elements and a `<summary>`. The former combines elements of TEI with the `<emotion>` element from EmotionML. The summary is an element of summarization to represent the analysis information of words.

In summary, EmoSpell provides the GUI interface for end users, making available the text and words analysis for any user, as well as the API interface, thus providing a service to other systems to implement and communicate with EmoSpell.

## 5 Validation

The validation of the proposed approach compared the results obtained on the analyzes of the same corpus with EmoSpell, EMOTAIX.PT and SentiLex-PT. The corpus used for validation consists of texts written by university students. Each student was asked to write 3 texts describing: a traumatic moment, a happy experience and their daily routine.

Table 1 compares the results obtained with EMOTAIX.PT and EmoSpell for the same texts. In this table, the columns with headers labeled **EX** and **ES** refer respectively to EMOTAIX.PT and EmoSpell results. It lists results of texts analysis from three participants; one positive, one negative and one neutral for each participant. For each text, it lists the number of emotional words given by EMOTAIX.PT and EmoSpell, distinguishing the positive, negative and neutral words. As it can be seen, in all the texts, EmoSpell can detect a larger number of emotional words.

For example, if we observe the positive text from the third participant, the line 3-Pos, we can verify that EMOTAIX.PT detects 10 emotional words and EmoSpell 17 words. Also, it

**Table 1** Comparison between EMOTAIX.PT (EX) and EmoSpell (ES).

| Participant – Texts | Words | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Emotional | | | Positive | | | Negative | | | Neutral | | |
| | **EX** | **ES** | **Δ%** | **EX** | **ES** | **Δ%** | **EX** | **ES** | **Δ%** | **EX** | **ES** | **Δ%** |
| **1-Pos** | 10 | 15 | 50 | 4 | 6 | 50 | 1 | 2 | 100 | 5 | 7 | 40 |
| **1-Neg** | 11 | 18 | 60 | 1 | 5 | 400 | 6 | 8 | 30 | 4 | 5 | 25 |
| **1-Neut** | 2 | 3 | 50 | 2 | 3 | 50 | 0 | 0 | 0 | 0 | 0 | 0 |
| **2-Pos** | 10 | 14 | 40 | 4 | 5 | 25 | 4 | 5 | 25 | 2 | 4 | 100 |
| **2-Neg** | 11 | 16 | 40 | 2 | 4 | 100 | 3 | 5 | 60 | 6 | 7 | 20 |
| **2-Neut** | 3 | 9 | 200 | 1 | 2 | 100 | 0 | 5 | 0 | 2 | 2 | 0 |
| **3-Pos** | 10 | 17 | 70 | 5 | 12 | 140 | 0 | 0 | 0 | 5 | 5 | 0 |
| **3-Neg** | 12 | 18 | 50 | 4 | 7 | 75 | 7 | 7 | 0 | 1 | 4 | 300 |
| **3-Neut** | 1 | 3 | 200 | 0 | 1 | $\infty$ | 1 | 2 | 100 | 0 | 0 | 0 |
| **Average** | 7.8 | 12.6 | 84 | 2.6 | 5 | 104 | 2.4 | 3.8 | 35 | 2.8 | 3.8 | 54 |

detects that this is a positive text taking into account the large number of positive words and the absence of negative words.

One of the aims of this experiment was to verify if it is possible to categorize between positive, neutral and negative texts. From the results of 81 written texts obtained by the university students, it was verified that they write more emotional words in the negative and positive condition than if in the neutral condition. Also, it was observed that there were more positive words in the positive texts, and more negative words in the negative texts, as expected. In Table 1, it is also possible to observe that ability to synthesize the polarity of texts. Also, in general, all participants used at least one emotional word of the opposite category.

The existence of a large number of words of contrasting polarity, in particular in negative texts, has several explanations. One explanation for the high number positive word in the negative and neutral texts is the way EMOTAIX.PT categorizes words. Words such as "boyfriend" / "girlfriend" and "friend" are defined in EMOTAIX.PT as positive. When participants write about an experience they typically use these words to refer to the persons involved, without a emotional connotation.

Another explanations is the fact that the participants use contrasting words to intensify the emotional experience. One example is "*we were leaving a birthday party at 5am, we were all very happy and cheerful because the party had been incredible. My friend got in the car ... a few meters ahead the car capsized, they were moments of panic and deep fear*".

As mentioned before, EmoSpell was also validated against SentiLex-PT. Table 2 compares the results obtained with SentiLex-PT and EmoSpell for the same texts. In this table, the columns with headers labeled **S** and **ES** refer respectively to SentiLex-PT and EmoSpell.

The emotional words detection was based not just on the SentiLex-PT lemmas but also on their flexed forms, with the SentiLex-flex file. This increases emotional word detection, just as Jspell does for EmoSpell.

In contrast with the EmoSpell and EMOTAIX.PT comparison, SentiLex-PT can verify a large number of emotional words and in some cases SentiLex-PT detects more emotional words than EmoSpell. This was expected since a number of emotional words in SentiLex-PT were found to be missing in EMOTAIX.PT and reported as part of the generation process, as explained in subsection 4.1. These words are currently being categorized by the authors of EMOTAIX.PT and will be available on the next version of EmoSpell.

**Table 2** Comparison between SentiLex-PT (EX) and EmoSpell (ES).

| Participant – Texts | Words | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Emotional | | | Positive | | | Negative | | | Neutral | | |
| | S | ES | Δ% | S | ES | Δ% | S | ES | Δ% | S | ES | Δ% |
| **1-Pos** | 13 | 15 | 15.4 | 5 | 6 | 20 | 5 | 2 | −60 | 3 | 7 | 133 |
| **1-Neg** | 13 | 18 | 38.5 | 4 | 5 | 25 | 6 | 8 | 33.3 | 3 | 5 | 66.6 |
| **1-Neut** | 8 | 3 | −62.5 | 5 | 3 | −40 | 1 | 0 | −100 | 2 | 0 | −100 |
| **2-Pos** | 12 | 14 | 16.6 | 3 | 5 | 66.6 | 5 | 5 | 0 | 4 | 4 | 0 |
| **2-Neg** | 9 | 16 | 77.7 | 0 | 4 | 0 | 6 | 5 | −16.6 | 3 | 7 | 133.3 |
| **2-Neut** | 7 | 9 | 28.6 | 2 | 2 | 0 | 5 | 5 | 0 | 0 | 2 | 0 |
| **3-Pos** | 15 | 17 | 13.3 | 9 | 12 | 33.3 | 3 | 0 | −100 | 3 | 5 | 66.6 |
| **3-Neg** | 15 | 18 | 20 | 7 | 7 | 0 | 6 | 7 | 16.6 | 2 | 4 | 100 |
| **3-Neut** | 7 | 3 | −57.1 | 1 | 1 | 0 | 2 | 2 | 0 | 4 | 0 | −100 |
| **Average** | 11 | 12.6 | 21.16 | 4 | 5 | 11.6 | 4.3 | 3.8 | −25.19 | 2.6 | 3.8 | 33.28 |

In any event, this comparison shows that EmoSpell detects, on average, as many emotional words as the SentiLex-PT. However, the main advantage of EmoSpell is that it provides more information on emotional words. Besides presenting a hierarchy of emotional classification, EmoSpell also detect the dominant emotional category of each text.

As part of the validation, the top emotional categories of the texts shown in Tables 1 and 2 were also synthesized. For example, the dominant categories of the three negative is "*benevolence – affection – love*", and "*non-specific emotions*". The three positives texts have the same dominant categories as the negative ones. This can be explained by the fact that the participants used several times words such as "friend", "girlfriend/boyfriend", "to feel/feeling" that are words of "*benevolence*" category. For the "*non-specific words*" it was also noticed that the participants usually write words to intensify their feelings like "think", "more", "larger", "great" and "hard", as already explained.

## 6 Conclusion

Emotional analyzers have several applications, in research, business and governance. For instance, the opinions expressed on social media regarding a particular product or subject provide important information for companies, organizations and government. Nevertheless, the motivation for this research work came from the application of sentiment analysis to the research on cognitive processes in writing.

Sentiment analysis requires an analyzer capable of detecting a wide range of emotional words in texts, classifying their emotional value and synthesizing the writers' emotional state. EmoSpell improves sentiment analysis for the Portuguese language by classifying words according to emotional categories, not just discovering their polarity or valence. To achieve it, EmoSpell uses EMOTAIX.PT, a lexical base structured in several hierarchical levels of emotional value.

The proposed system build on the lexical analyzer Jspell to enhance the recognition power of EMOTAIX.PT. The main contribution of this work is a procedure for generation of a new Jspell dictionary integrating EMOTAIX.PT. A secondary contribution is the two interfaces to this dictionary, an interactive web application and a text analysis web service. The former was used to validate the proposed approach and is available for experiments in emotional text analysis. The latter will be integrated in the project HandSpy – a web environment

for managing experiments on cognitive processes in writing – and is also available to other systems requiring a syntactic and emotional analyzer of Portuguese texts.

As future work, this emotional analyzer can be improved to better support sentiment analysis in Portuguese. As previously mentioned, there are challenges to overcome such as multiple word analysis and the differentiation of the polarity.

Multiple word analysis would be an important feature to this project. The problem with the use of positive words in a negative context, such as "*I don't like*", could be solved with this addition. Some lexicons already overcame this challenge [2]. One approach is to analyze the phrases not only word by word but in a multi-level way, calculating the sentence polarity by verifying the noun and verb in phrases and identifying their polarities.

As SentiLex-PT, the differentiation of the polarity target can accomplish a more accurate analysis. In EmoSpell, the division of the polarity target into subject and complement would allow a word analysis with different meaning words. That is, it would solve the problem that a word can have opposite polarity when combined with another word.

**Acknowledgements.** The authors wish to thank to Alberto Simões and José João Almeida, the authors of Jspell, for their help.

## References

1 José João Almeida and Ulisses Pinto. Jspell – um módulo para análise léxica genérica de linguagem natural. In *X Encontro da Associaçao Portuguesa de Linguistica*, pages 1–15, 1994.

2 Amna Asmi and Tanko Ishaya. Negation identification and calculation in sentiment analysis. In *Second International Conference on Advances in Information Mining and Management*, pages 1–7, 2012.

3 Felix Burkhardt, Catherine Pelachaud, Björn W. Schuller, and Enrico Zovato. EmotionML. In Deborah A. Dahl, editor, *Multimodal Interaction with W3C Standards: Toward Natural User Interfaces to Everything*, pages 65–80. Springer, 2017.

4 Felix Burkhardt and Marc Schröder. Emotion markup language (EmotionML) 1.0. W3c recommendation, World Wide Web Consortium, 2014.

5 Paula Carvalho and Mário J. Silva. SentiLex-PT: Principais características e potencialidades. *OSLa, Oslo Studies in Language*, 7(1):425–438, 2015.

6 Wilas Chamlertwat, Pattarasinee Bhattarakosol, Tippakorn Rungkasiri, and Choochart Haruechaiyasak. Discovering consumer insight from twitter via sentiment analysis. *Journal of Universal Computer Science*, 18(8):973–992, 2012.

7 Sara Filipa Oliveira Costa. Adaptação e teste de uma base lexical de palavras emocionais para o português europeu: (EMOTAIX.PT). Master's thesis, Universidade do Porto, 2012.

8 Andrea Esuli and Fabrizio Sebastiani. SentiWordNet: a high-coverage lexical resource for opinion mining. *Evaluation*, pages 1–26, 2007.

9 Christiane Fellbaum. *WordNet*. Wiley Online Library, 1998.

10 Nancy Ide and Jean Véronis. *Text encoding initiative: Background and contexts*, volume 29. Springer Science & Business Media, 1995.

11 Bing Liu. Sentiment analysis and opinion mining. *Synthesis Lectures on Human Llanguage Technologies*, 5(1):1–167, 2012.

12 Carlos Monteiro and José Paulo Leal. Managing experiments on cognitive processes in writing with HandSpy. *Computer Science and Information Systems*, 10(4):1747–1773, 2013.

13 Brendan O'Connor, Ramnath Balasubramanyan, Bryan R. Routledge, and Noah A. Smith. From tweets to polls: Linking text sentiment to public opinion time series. In *International AAAI Conference on Web and Social Media*, volume 11, pages 122–129, 2010.

**14** Thierry Olive, Rui Alexandre Alves, and São Luís Castro. Cognitive processes in writing during pause and execution periods. *European Journal of Cognitive Psychology*, 21(5):758–785, 2009.

**15** James W. Pennebaker, Martha E. Francis, and Roger J. Booth. *Linguistic inquiry and word count: LIWC2001*, 2001.

**16** Annie Piolat and Rachid Bannour. EMOTAIX: un scénario de tropes pour l'identification automatisée du lexique émotionnel et affectif. *L'Année psychologique*, 109:655–698, 2009.

**17** Ellen Riloff, Ashequl Qadir, Prafulla Surve, Lalindra De Silva, Nathan Gilbert, and Ruihong Huang. Sarcasm as contrast between a positive sentiment and negative situation. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 704–714, 2013.

**18** Mário J. Silva, Paula Carvalho, and Luís Sarmento. Building a sentiment lexicon for social judgement mining. In *International Conference on Computational Processing of the Portuguese Language (PROPOR)*, pages 218–228, 2012.

**19** Alberto Simões and José João Almeida. jspell.pm: um módulo de análise morfológica para uso em processamento de linguagem natural. In *Associação Portuguesa de Linguística*, pages 485–495, 2001.

**20** Maite Taboada, Julian Brooke, Milan Tofiloski, Kimberly Voll, and Manfred Stede. Lexicon-based methods for sentiment analysis. *Computational linguistics*, 37(2):267–307, 2011.

**21** G. Vinodhini and R. M. Chandrasekaran. Sentiment analysis and opinion mining: a survey. *International Journal of Advanced Research in Computer Science and Software Engineering*, 2(6):282–292, 2012.

**22** Theresa Wilson, Janyce Wiebe, and Paul Hoffmann. Recognizing contextual polarity in phrase-level sentiment analysis. In *Conference on Human Language Technology and Empirical Methods in Natural Language Processing*, pages 347–354, 2005.

# Information Extraction for Event Ranking*

**José Devezas[1] and Sérgio Nunes[2]**

1 **INESC TEC and Faculdade de Engenharia, Universidade do Porto, Porto, Portugal**
`jld@fe.up.pt`
2 **INESC TEC and Faculdade de Engenharia, Universidade do Porto, Porto, Portugal**
`ssn@fe.up.pt`

## Abstract

Search engines are evolving towards richer and stronger semantic approaches, focusing on entity-oriented tasks where knowledge bases have become fundamental. In order to support semantic search, search engines are increasingly reliant on robust information extraction systems. In fact, most modern search engines are already highly dependent on a well-curated knowledge base. Nevertheless, they still lack the ability to effectively and automatically take advantage of multiple heterogeneous data sources. Central tasks include harnessing the information locked within textual content by linking mentioned entities to a knowledge base, or the integration of multiple knowledge bases to answer natural language questions. Combining text and knowledge bases is frequently used to improve search results, but it can also be used for the query-independent ranking of entities like events. In this work, we present a complete information extraction pipeline for the Portuguese language, covering all stages from data acquisition to knowledge base population. We also describe a practical application of the automatically extracted information, to support the ranking of upcoming events displayed in the landing page of an institutional search engine, where space is limited to only three relevant events. We manually annotate a dataset of news, covering event announcements from multiple faculties and organic units of the institution. We then use it to train and evaluate the named entity recognition module of the pipeline. We rank events by taking advantage of identified entities, as well as *partOf* relations, in order to compute an entity popularity score, as well as an entity click score based on implicit feedback from clicks from the institutional search engine. We then combine these two scores with the number of days to the event, obtaining a final ranking for the three most relevant upcoming events.

**1998 ACM Subject Classification** I.2.7 Natural Language Processing, H.3.3 Information Search and Retrieval

**Keywords and phrases** Named Entity Recognition, Relation Extraction, Knowledge Base Population, Entity-Based Ranking, Academic Events

**Digital Object Identifier** 10.4230/OASIcs.SLATE.2017.18

## 1 Introduction

Semantic search is not yet a reality. Nonetheless search engines are continuously striving to become more semantic-aware [14, 2]. Most modern search engines already rely on a well-

* José Devezas is supported by research grant PD/BD/128160/2016, provided by the Portuguese funding agency, Fundação para a Ciência e a Tecnologia (FCT). This work is partially supported by Project "TEC4Growth – Pervasive Intelligence, Enhancers and Proofs of Concept with Industrial Impact/NORTE-01-0145-FEDER-000020", financed by the North Portugal Regional Operational Programme (NORTE 2020), under the PORTUGAL 2020 Partnership Agreement, and through the European Regional Development Fund (ERDF).





curated knowledge base, having a basic understanding of telegraphic natural language queries. It is query understanding [12] that supports the "decoration" of results with contextual widgets that directly provide answers to the users questions by either displaying lists of entities (e.g., "sci-fi movies from 1985") or information about a single entity (e.g., "back to the future"), usually pointing to related queries (e.g., "Directed by Robert Zemeckis" or "Time travel movies") or even to queries leading to other relevant entities (e.g., "Michael J. Fox" or "Christopher Lloyd", which are part of the cast). Longterm solutions to support this kind of entity-oriented search require increasingly better automatic approaches for the extraction of entities and relations, as well as the improved integration of heterogeneous data sources, such as textual content and multiple knowledge bases that potentially need to be aligned and merged.

There are several challenges that need to be tackled in order to enable the creation of a truly semantic search engine. In this work, however, we cover the basics, by providing an overview of a complete information extraction pipeline, from data acquisition to knowledge base population. We then present an actual application of the extracted information, where we take advantage of identified entities and relations to improve a widget from an entity-oriented institutional search engine. In particular, we focus on a query-independent ranking problem, where the goal is to rank news that cover event announcements, in order to display the three most relevant upcoming events of general interest to the local academic community. This is illustrated in Figure 1, where we show the landing page for the ANT search engine[1], highlighting the upcoming events widget. Our approach to event relevance scoring is based on three factors: (*i*) the overall popularity of the mentioned entities, (*ii*) the overall popularity of the mentioned entities from clicked event news, and (*iii*) the number of days left until the event starts. Despite lacking an innovative facet, we present a complete implementation of an information extraction pipeline, for the Portuguese language, describing the whole process from data acquisition to the final application, where we solve a ranking problem from a real-world search engine with the knowledge base we automatically constructed.

## 2 Reference Work

In order to better grasp the full scope of the problem we tackle here, we require an understanding of the typical information extraction approaches, as well as available tools and corpora. We also require a bit of context regarding the integration of information extraction (IE) and information retrieval (IR), in particular with a focus on applications to semantic search.

### 2.1 Information Extraction

Information extraction consists of uncovering information from unstructured data. While it can be applied to different types of data, we only focus on textual content. Natural language expressed as text is an extremely rich source of information and the most common means of sharing knowledge among humans. Machines, however, rely on the identification of structure within this kind of unstructured data in order to be able to find answers to increasingly complex questions. Therein lies the goal of information extraction. Given a text, a typical pipeline would consist of named entity recognition (e.g., "José Devezas" or "InfoLab") and classification (e.g., "Person" or "Organization"), followed by the extraction

[1] `http://ant.fe.up.pt`

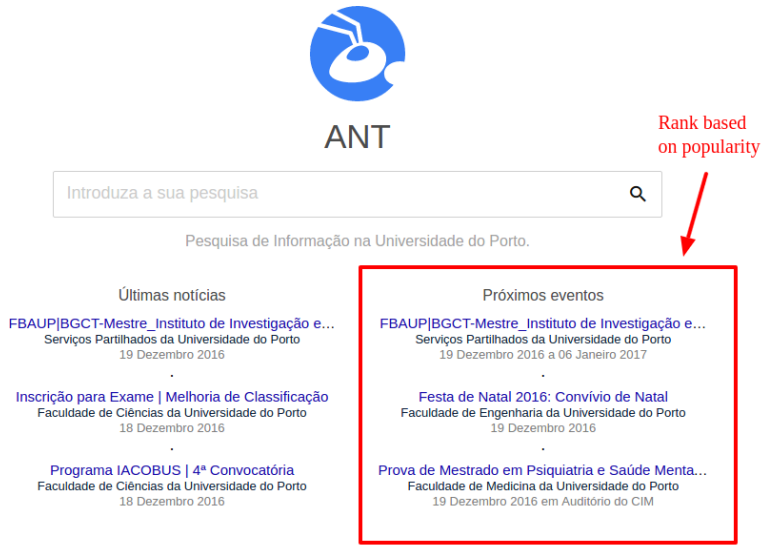



**Figure 1** ANT – Entity-Oriented Search Engine for the University of Porto. Upcoming events widget is highlighted, illustrating the target application of the entity-based ranking strategy.

of relations between identified entities, usually based on patterns centered around verbs (e.g., "works at"). So, given a text with the sentence "José Devezas works at InfoLab.", an information extraction pipeline would be able to build a machine-understandable statement for the "[Person] works at [Organization]" relation, identifying "José Devezas" as a "Person", "InfoLab" as an "Organization" and "works at" as the relation between "José Devezas" and "InfoLab". This type of relations is usually stored in a triple store. A triple store contains statements (or triples) in the format *(subject, predicate, object)*, which, as a whole, form a knowledge base. The knowledge base is a semantic network that links concepts and provides a structured way of accessing information – individual statements represent information that, when combined, can provide knowledge.

Information extraction can be classified as closed or open, regarding the domain of application. When the domain is closed, we can easily take advantage of handcrafted rules or gazetteers (dictionaries of entities) in order to link chunks of words to a particular entity, via a regular expression or through string matching, respectively. Open information extraction is used when the domain is unknown or too extensive to cover manually, as is the case of the web. Banko et al. [1] focus on the extraction of relational tuples from the web, without any human input, taking into account issues like automation, corpus heterogeneity and efficiency. They present TextRunner, which uses a self-supervised learner to measure the trustworthiness of candidate tuples, along with a single-pass extractor to identify each candidate tuple, using a redundancy-based assessor module to decide on which trustworthy tuples to keep, with a given probability. Their system is also able to be queried, in a distributed manner, supporting complex relational queries that go beyond a traditional search engine.

The typical result of an information extraction pipeline is a knowledge base. Google is currently supported on their own well-curated knowledge base, Knowledge Graph, which evolved from Freebase. However, while perhaps less well-known, they are also working on Knowledge Vault [8], an automated approach to build the automated equivalent of Knowledge Graph. Knowledge Vault does information fusion based on supervised learning

and prior knowledge derived from existing knowledge bases, associating a probability with each extracted statement. Their goal is to enable question answering and entity-oriented search systems, with fewer dependence on manual labor.

After building such a knowledge base, and in order to potentiate semantic search, one of the main relevant techniques is entity linking [22]. First, there is the need to identify entities in a text, for instance with resource to techniques like Conditional Random Fields [13], which is the approach we use in our pipeline. Then, named entity disambiguation is usually required to be able to identify the entity associated with a given word or sequence of words. While we did not implement a disambiguation step within our pipeline, given the relevance of this technique in general and for future work, we still present an interesting example. Nebhi [18] proposed a named entity disambiguation strategy based on entity popularity and syntactic features, trained using a Support-Vector Machine classifier. Entities extracted from text were matched with entities in a knowledge base (in this case Freebase was used) and the disambiguation module determined which candidate entity was more likely to be associated with that particular textual representation. Entity popularity was used as fallback (when everything else fails, it's probably the most frequent), while syntactic features were responsible for providing richer indicators than a bag-of-words approach, being able to capture dependencies between words in a sentence and, in a way, providing additional context.

### 2.1.1 Tools

There are several tools for natural language processing, implementing named entity recognition, as well as relation extraction. The tool we decided to use was the Natural Language ToolKit (NLTK) [3]. NLTK is a Python library that integrates well with other tools such as MaltParser [19], a dependency parser written in Java, or Stanford NER[2], a named entity recognition Java command line tool and library, based on Conditional Random Fields (CRFs). NLTK directly provides access to corpora in multiple languages, to train for instance a Part-Of-Speech (POS) tagger. POS tagging can be used as a feature for named entity recognition or relation extraction. In this work, we use Stanford NER without POS tags to extract entities, but build regular expressions based on words and their POS tags to extract relations.

In order to learn a CRF for the Portuguese language, able to support a custom set of entity types, we first needed to annotate our own dataset. One of the easiest ways to annotate a dataset is to use the Brat rapid annotation tool [23]. Brat is a web application, built in Python, that requires little configuration and a corpus of plain text documents to annotate. We will provide additional detail on the configuration and usage of Brat in Section 3.

Other relevant tools include LemPORT [21], a lemmatizer for the Portuguese language. Lemmatization can be useful for instance to resolve multiple verbal forms of the same verb into a single word, enabling better relation extraction. This is less efficient but more effective than stemming, where the suffix of the word is trimmed with the goal of obtaining common prefixes for similar words. Also regarding the Portuguese language, a well-known tool is REMBRANDT [4], a framework for semantic information retrieval, trained with the Second HAREM Portuguese corpus [16] and achieving first place in the competition, with an F-measure of 0.625. Finally, while not specifically relevant for the Portuguese language, we also cite GATE [6], a General Architecture for Text Engineering and, in particular Mímir [5], a multiparadigm indexing and retrieval framework, which is capable of searching

[2] `http://nlp.stanford.edu/software/CRF-NER.shtml`

over heterogeneous data sources, like text, annotations, ontologies and knowledge bases. GATE Mímir shares many of the goals of the ANT search engine, combining information extraction and retrieval in a single framework.

### 2.1.2 Corpora

There are at least two Portuguese corpora, available through NLTK, with annotated POS tags: Mac-Morpho [9] and Floresta [10]. These datasets are also called treebanks, which are corpora annotated with syntactic or semantic sentence structure. In this particular work, we only rely on Floresta treebank to train a POS tagger based on its *(word, tag)* tuples. The POS tags can be useful for named entity recognition and for relation extraction, but we only use this feature during relation extraction to build regular expressions.

While there are several tools for the English language, with pre-trained models for multiple tasks, the Portuguese language has been less explored, despite having its own particular challenges. These include dealing with European and Brazilian Portuguese, or even capturing the changes from the latest Portuguese orthography reform (1990)[3]. More importantly, the lack of resources for the Portuguese language is of great concern. One of the main, or perhaps the only publicly available dataset with annotated entities and relations for the Portuguese language is HAREM [16]. There are two versions of this dataset, but only the second version includes annotations regarding semantic relations between entities. While there are other datasets, their availability is a challenge, as they are held hostage as a valuable resource in a competitive world of science. One way to provide a larger amount of annotated data is simply to push for the semantic web and to motivate people to annotate their web sites using RDFa or Microdata. When this happens, hopefully in a near future, it will definitely be a valuable resource for the information extraction community, but until then, we can either annotate our own dataset or use HAREM. In this work, we will describe a way to easily annotate our own dataset, with a set of entity types of our choice, based on Brat.

## 2.2 Semantic Ranking and Search

Until now, we have been surveying information extraction techniques, with the final goal of automatically building a knowledge base to support a semantic ranking task. Knowledge bases can be used to empower search engines or to generically improve ranking, even in query-independent tasks. Next, we provide a bit of context regarding some of the central tasks of a semantic search engine that are representative of the integration of IE and IR. We cover query understanding, semantic matching, semantic search on text and knowledge bases and ontology-based ranking.

Query understanding involves conferring meaning to a keyword query. This is frequently done by segmenting the query and associating entities, types, or attributes to keyword chunks, through what is usually called semantic tagging [7]. Hu et al. [12] present a method for understanding query intent based on an index of Wikipedia concepts, supported on the idea of a graph of linked concepts. Given a query, the Wikipedia index is searched, returning the best matches to Wikipedia pages representing specific concepts. Whenever there is no match, the original query can be fed to a traditional *ad hoc* document retrieval search engine to build new queries from related text – in particular, the authors used Live Search[4], which has,

[3] http://www.portaldalinguaportuguesa.org/?action=acordo&version=1990
[4] http://search.live.com

at the time of this writing, been replaced with Bing[5]. The title and the description of the top-$K$ results is used to emit additional queries to the Wikipedia index, giving more weight to the title. The final result is a ranked list of Wikipedia concepts that better illustrates query intent, covering not only concepts that are directly mentioned in the original query, but also concepts related to those, that are theoretically close in the graph of linked concepts. Their idea is not very different from the idea of community structure in a graph [20] – dense subgraphs, where member nodes have more connections between themselves than to the remainder of the network, usually represent similar or related concepts.

A bit different from query understanding, semantic matching [14] is concerned with solving mismatch issues between query and documents (e.g., searching for "ny times" should also match documents containing "New York Times"). While semantic matching is different from semantic search, some semantic matching solutions can still take advantage of techniques like named entity recognition, or knowledge bases like ConceptNet [11], for query expansion. Nevertheless, semantic search on text and knowledge bases [2] is the main stage for the combination of information extraction and information retrieval. In particular, the two areas meet when doing keyword search or question answering on combined data (text + knowledge bases). While semantic search can exist solely over knowledge bases, it can seldom exist solely over text, even though some semantics can be captured from text with techniques like word embeddings (e.g., word2vec [15]). Thus, information extraction plays a central role in search engines of the future, already supporting well-known search engines like Google, through knowledge bases like Knowledge Graph[6].

Finally, there is also work by Vallet et al. [24] showcasing the potential of ontology-based ranking, where a corpus is already linked to instances in a knowledge base. They associate a weight to each annotation, inspired by TF-IDF, but applied to knowledge base instances as opposed to terms, over a text document. The approach is to convert the user query into an RDQL query (a SPARQL predecessor), which is used to retrieve tuples from the knowledge base. Documents annotated with the instances from the obtained tuples are then retrieved as well and presented to the user sorted by a ranking function that measures semantic similarity. Semantic similarity is computed based on the cosine similarity between a query vector $\vec{q}$ and a document vector $\vec{d}$. The document vector $\vec{d}$ is defined by the analogous TF-IDF weights of the instances in the documents, while the query vector $\vec{q}$ is defined by weights assigned to the variables in the RDQL query, which can be either selected by the user, or obtained through personalization or frequency analysis.

### 2.2.1 Our Approach in Context

Despite different from semantic search, in the sense that there is no issued query, the problem of semantic ranking we present here is closely related to the approaches we covered so far. We rank academic events based on the news articles that announce them. In particular, we use an information extraction pipeline for named entity recognition, as well as relation extraction, over institutional news. We then build a knowledge base, containing event information, but also *partOf* relations between pairs of organizations and locations. This enables us to rank events based on historical information on entity popularity and news article clicks, while also propagating popularity through *partOf* relations. It means that our ranking function captures the following information: if events about "Information Retrieval" are popular and

[5] `http://www.bing.com`
[6] `https://www.google.com/intl/es419/insidesearch/features/search/knowledge.html`

**Table 1** Entity types used to annotate the SIGARRA News Corpus, along with examples aligned with Figure 2. Listed entity types were used to populate the **[entities]** section of the `annotations.conf` file in Brat.

| Entity Type | Example |
|---|---|
| Person | *Liliana de Jesus Duarte da Mota* |
| Organization | *Faculdade de Direito da Universidade do Porto* |
| Event | *Provas de Mestrado em Direito - Licenciada Liliana de Jesus Duarte da Mota* |
| EventType | *Provas de Mestrado* |
| Topic | *Direito* |
| Location | *sala 228* |
| Date | *16 de dezembro de 2016* |
| Time | *11h00* |

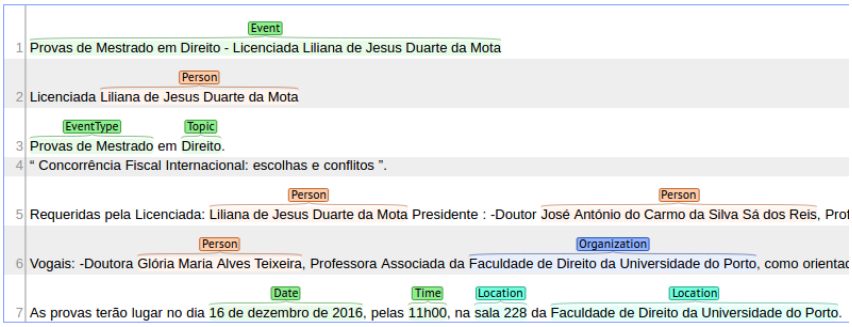



**Figure 2** Example of an annotated fragment of text, for a SIGARRA news, in the Brat rapid annotation tool.

events at "Faculdade de Engenharia da Universidade do Porto" are also popular, then an event at "Departamento de Engenharia Informática" about "Information Retrieval" will have a high probability of being relevant to the community and should have a higher rank. While the link to "Information Retrieval" was directly established, "Departamento de Engenharia Informática" indirectly received a high popularity weight because it is *partOf* "Faculdade de Engenharia da Universidade do Porto".

Since our system does not do entity disambiguation, whenever the same department naming is used across different universities, popularity will be propagated through all *partOf* links. In order to improve this, entity disambiguation is, in fact, one solution, but we can also assign an uncertainty weight proportional to the number of target nodes in the *partOf* relation, similar to IDF – the more target nodes are reached, the less reliable is the information. For our particular domain, however, we predict this to be of low impact – in fact, considering this type of relations without disambiguation might be a way of avoiding popular venues to control the ranks, giving a chance of similar events, happening at different venues, receiving a higher rank.

## 3 Data Acquisition, Model Training and Evaluation

The ANT search engine periodically collects and processes data from SIGARRA, the information system at the University of Porto. Data is retrieved using a combination of XPath and CSS selectors and stored in a relational database. While most of the data, like student or staff profiles, is structured, there is also some unstructured data from SIGARRA news.

**Table 2** Evaluation of the named entity recognition module, based on the test set for the SIGARRA News Corpus.

| Avg. Precision | Avg. Recall | Macro F1 |
|---|---|---|
| 0.633220 | 0.371867 | 0.468563 |

In this work, we are particularly interested in extracting information about events, locked inside SIGARRA news as natural language text. SIGARRA news announcing events can be identified based on whether there is an associated event start or end date in the database. We only retrieve and process news articles with at least one event date associated and skip articles in languages other than Portuguese. Steps from data acquisition to model training and evaluation are as follows:

1. Query the relational database for a subset of news articles that announce events and store them as a CSV file (one article per row).
   *We retrieved the news articles for the* 1,000 *closest upcoming events, starting from December 23, 2016, and containing contributions from all faculties and organic units of the University of Porto.*
2. For each row in the CSV, prepare a `txt` file containing title, subtitle and content, as well as an empty `ann` file.
   *SIGARRA news articles communicate a lot of relevant information in the title and subtitle. In particular, we used the title to build* `Event` *entities and the subtitle to extract information like the location or the date of the event. The content was provided as an HTML fragment that we converted into text using* `bs4.BeautifulSoup`*, after removing* `script` *and* `style` *tags.*
3. Put individual files within a subdirectory in the `data/` directory of the Brat rapid annotation tool and create an `annotations.conf` file with the list of entity types to annotate (shown in Table 1, along with examples).
4. Run `./standalone.py` in Brat's root directory and manually annotate the corpus, as shown in Figure 2 (we annotated 25 out of the 1,000 retrieved documents).
5. Split the annotated corpus into two separate directories, one for training (70%; 18 news articles) and another one for testing (30%; 7 news articles).
   *While we admit the size of the annotated corpus is not ideal, surprisingly 18 annotated documents were enough to build a working system. Additionally, the annotated corpus can be extended over time, in order to retrain and re-evaluate the system.*
6. Convert training documents into a single `col` file (tab-separated format supported by Stanford NER), and testing documents into individual `col` files to enable per-document evaluation.
7. Train a Conditional Random Field (CRF) using Stanford NER command line interface and obtain a model for named entity recognition (NER).
8. Evaluate the NER module. Extract entities from the original, non-annotated documents of the test set, using `StanfordNERTagger` from NLTK along with the learned CRF model. Compare extracted entities with annotated entities based on the `col` files, computing metrics like precision, recall and F-score.
   *There are several evaluation methods for NER [17]. In this particular case, we used the CoNLL evaluation scheme, where only exact matches are considered as true positives. We calculate the precision and recall for each document and then the overall averages for the test set. We also compute the macro-averaged F1-score.*

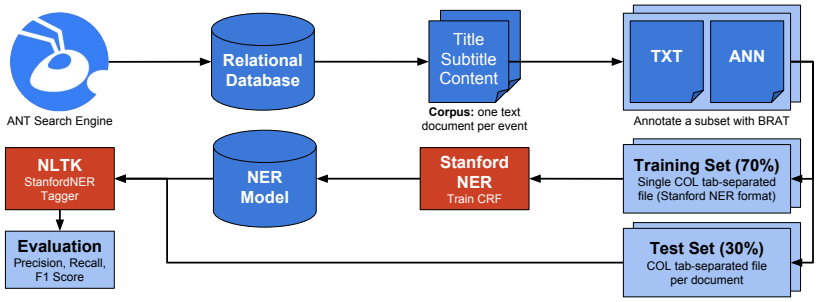



**Figure 3** Data acquisition, named entity recognition and evaluation. Data is obtained from the ANT search engine database and pre-processed to enable CRF training and testing.

To summarize, Figure 3 presents a complete overview of the named entity recognition module, from data acquisition to evaluation. As we can see, starting from the ANT search engine, we access the relational database, in order to obtain a corpus, initially stored as a CSV file. We then convert each document into a text file, with an associated empty annotations file. The corpus is then annotated using Brat rapid annotation tool and split into a training set and a test set. The training set is used to train a CRF[7], obtaining a NER model. The NER model is then used by the `StanfordNERTagger` from NLTK to evaluate the effectiveness of the named entity recognition module based on the test set. Evaluation results are shown in Table 2 – we obtained an average precision of 63%, which is acceptable given the reduced size of the annotated corpus, as well an average recall of 37% and a macro-averaged F1-score of 47%.

## 4 Information Extraction

When running the information extraction pipeline, we simply query the relational database, iteratively processing each individual news article, until a set of triples with the identified entities and relations in the document is constructed and loaded into the triple store. The pipeline we defined is mostly based on NLTK and includes the following steps, that are applied to each text sequentially:

**detect_language()** Our pipeline was trained using a Portuguese corpus. In order to ensure we only process Portuguese documents, we use the Python wrapper for the well-known `langdetect` Java library[8]. Whenever a text is not identified as 'pt', it is simply skipped. The associated event might still be featured in ANT's event widget, but only when it's extremely close to the date and no other, more relevant events, are held simultaneously.

**segment_sentences()** The first step in processing a SIGARRA news article is to split the text into individual sentences. We used the pre-trained Portuguese model for the Punkt sentence tokenizer provided by NLTK. Challenges associated with this task are frequently associated with distinguishing actual sentence ends from periods that do not end a sentence (e.g., 'Mr.' or 'Dr.'). While this is a task that is essentially solved and already achieves a high precision in the state of the art, given the quality of our text, many times sentences were incorrectly identified. However, we did not find this to be particularly

[7] We used a default configuration for Stanford NER, based on the example in the FAQ: `http://nlp.stanford.edu/software/crf-faq.shtml#a`.

[8] `https://github.com/Mimino666/langdetect`

impactful in the end result, as we did not rely too much on syntactic or dependency parsing or even on POS tags, which we only used for relation extraction.

**tokenize_sentences()** Given a list of strings, obtained from the previous step and corresponding to a sentence each, we split each string into individual words, as well as punctuation. For this, we used `WordPunctTokenizer` from NLTK, instead of the default tokenizer implemented in `nltk.tokenize.word_tokenize`, which was `TreebankWordTokenizer`. While this tokenizer reliably identified words, without splitting for instance compound words by dashes, it did not implement the `span_tokenize()` method, which was fundamental to simplify the conversion process from the Brat standoff format `ann` files into the Stanford NER tab-separated format `col` files. During tokenization, we also replaced any slash characters by a dash, since slashes are removed by `StanfordNERTagger` and, as we will explain next, we will need to match tagged sentences, word by word, in order to build a tree combining POS tags and named entity tags.

**pos_tag_sentences()** Given a list of lists of words and punctuation, from the previous step, we assigned a POS tag to each word, obtaining a list of lists of *(word, pos_tag)* tuples. Since there is no pre-trained model for POS tagging Portuguese sentences in NLTK, we used the provided Floresta treebank corpus to learn our own model. For training, we used a `nltk.BigramTagger`, falling back to a `nltk.UnigramTagger` and then to a `nltk.DefaultTagger` which always identifies a word as a noun (the most common), in case all else fails. An evaluation of a similar POS tagger based on Floresta treebank was already available[9], showing an accuracy of 89%, 87% and 18%, respectively for each tagger.

**ne_tag_sentences()** Given a list of lists of words and punctuation (without POS tags), we assigned a named entity to each word, obtaining a list of lists of *(word, ne_tag)* tuples. In order to do this, we used the model we had trained for Stanford NER, feeding it to the `StanfordNERTagger` (see step 8 in Section 3 for more details).

**build_sentence_trees()** Given the lists generated by `pos_tag_sentences()` and `ne_tag_sentences()`, we generate a list of `nltk.tree.Tree` (one per sentence). In order to do this, we use the `nltk.chunk.util.conlltags2tree()` function, which takes a sentence argument as a list of *(word, pos_tag, ne_tag)* tuples. In order to build such tuples, we assume that an exact equivalence can be established between the two lists, otherwise the system will fail, raising an exception, which results in the skipping of the news article being processed. The resulting tree has three levels, a root level (the sentence), a mid-level (the entity types) and a bottom-level (leaves corresponding to chunks of words belonging to a named entity, as aggregated by the mid-level nodes).

**extract_entities()** In order to extract entities, we simply iterate over the trees constructed in the previous step, aggregating tokens per entity type. This is one of the outputs of our system, which is mainly used for evaluation purposes, regarding the effectiveness of the named entity recognition module. Results are saved as a human-readable `ent` text file (one per document), containing lists of entities, grouped by type, with the frequency of the entity in the document. We also save the same information as a `col` file, in order to compare predicted entities with the ground truth established in the test set. Results have already been shown in Table 2 and commented at the end of Section 3.

**extract_relations()** Relation extraction consists of identifying links between pairs of entities. In order to do this, we defined a set of rules, using regular expressions to identify relations between two types of entities, based on `nltk.sem.extract_rels()`. We then defined

[9] `http://www.nltk.org/howto/portuguese_en.html`

an associated list of rules to map the extracted relations into one or more triples in the knowledge base, possibly in reverse order. For example, the rule *('Location', '(da|do)/n', 'Location')* was used to identify *partOf* relations between two locations. Each Location entity was then mapped to a `dul:Place` class, from the DOLCE+DnS Ultralite ontology (DUL), and the *partOf* relation was mapped to the `dul:isPartOf` property from the same ontology. Event information was modeled using the Linked Open Descriptions of Events ontology (LODE), which focuses on defining a `lode:Event` class and a set of properties like `lode:atPlace`, that links to `dul:Place`, or `lode:involvedAgent`, that links to `dul:Organization` or `dul:Person`. The Time ontology was also used to describe date and time, using `lode:atTime` to link to a `time:TemporalEntity` and, in particular, a `time:Instant` with the property `time:inXSDDateTime` linking to a `xsd:dateTime` literal.

**build_default_relations()** Since many of the extracted entities were not featured in any relation, and in order to expand our knowledge base to better support the ranking task, we decided to generate some default triples that linked identified entities to the corresponding `lode:Event`. With this step, we compromised the quality of the information, in the sense that we considered for instance all dates as part of the `lode:atTime` relation and we know that some dates were in fact deadlines associated with events. The same is true for locations, as different places were sometimes referenced in news articles, besides the venue of the event. This is something that we intend to improve over time, either by using different properties like `dul:associatedWith`, or by improving automatic relation extraction.

**load_relations_into_virtuoso()** Finally, we generate an N-Triples (`nt`) file with the relations identified for each document, including relations of type *isA*, which annotate each entity with its class. This `nt` file is then loaded into OpenLink Virtuoso, through a POST request to the `/sparql-graph-crud-auth` endpoint[10], storing this information in a separate `ant:EventsKnowledgeBase` graph, using the same naming convention for events and their associated news article, already part of an existing ANT ontology[11].

## 5 Event Ranking

Loading the extracted information, in the form of triples, into the search engine's knowledge base, completed the process of annotating existing SIGARRA news articles. This included links to the extracted entities, as well as the capturing of interesting relations, like *partOf*, that were previously locked inside the news body as natural language. Combining this information can confer a degree of knowledge to our search engine and, in particular, it can be applied to the task of event ranking.

Prior to the approach we present here, ANT simply displayed the three closest upcoming events, without any particular ranking strategy. Equipped with a knowledge base we can do much more. In particular, we propose two scores based on entities associated with events: the entity popularity score, $score_{pop}(e, E_e)$, and the entity click score, $score_{clk}(e, E_e)$. The entity popularity score is based on the popularity of individual entities, as defined by the frequency an entity appears in distinct news articles over the whole corpus. The entity click score is similar to the popularity score, but is limited to the entities that are linked to clicked

[10] `https://virtuoso.openlinksw.com/dataspace/doc/dav/wiki/Main/VirtGraphProtocolCURLExamples`
[11] Example of URI for `lode:Event` / `ant:NewsArticle`: `http://infolab.fe.up.pt/ontologies/ant#Sigarra_News_Article_53369_From_Faculdade_de_Engenharia_da_Universidade_do_Porto`.

**Listing 1** SPARQL query to compute $score_{clk}(e, E_e)$ based on `lode:`.

```
1  SELECT ?event ?code ?school (SUM(?count) AS ?score)
2  WHERE {
3    {
4      SELECT ?agent (COUNT(?agent) AS ?count)
5      FROM ant:EventsKnowledgeBase
6      WHERE {
7        ?event a lode:Event .
8        ?event ant:wasClicked "true"^^xsd:boolean .
9        ?event lode:involvedAgent ?agent .
10     }
11     GROUP BY ?agent
12   }
13   ?event a lode:Event .
14   ?event lode:involvedAgent ?involved_agent .
15   ?agent dul:partOf* ?involved_agent .
16   OPTIONAL {
17     ?event ant:hasCode ?code .
18     ?event ant:hasFaculty ?school .
19   }
20 }
21 GROUP BY ?event ?code ?school
```

event news – this click status is transfered to the knowledge base daily, when the IE pipeline is ran, and is stored using the `ant:wasClicked` property, linking to a `xsd:boolean` literal.

The two scores are computed using a SPARQL query and then stored in the relational database, each in their own column of the news articles table. Events are then retrieved by combining the number of days remaining to the event start date and the entity popularity and click scores. Listing 1 shows the SPARQL query used to calculate $score_{clk}(e, E_e)$ for all `lode:Event` instances in the system. The $score_{pop}(e, E_e)$ is calculated using a similar query, where we remove the statement in line 8, discarding the constraint for clicked events. As we can see, each entity score is calculated, per event, based on the total number of links to the entities that are associated with the event. We illustrate this by using `lode:involvedAgent` property for classes `dul:Person` and `dul:Organization`.

After computing and storing the two entity scores for each event, we calculate the final score, for event ranking, as shown in Equation 1. Given event $e$ and entities $E_e$, associated with event $e$, the final score is calculated based on a weighted average of three factors: days to event, $\Delta T_e$, entity popularity score, $score_{pop}(e, E_e)$, and entity click score, $score_{clk}(e, E_e)$.

$$score(e, E_e) = w_1 \times \frac{1}{\Delta T_e + 1} + w_2 \times \frac{score_{pop}(e, E_e)}{\max_e\{score_{pop}(e, E_e)\}} + w_3 \times \frac{score_{clk}(e, E_e)}{\max_e\{score_{clk}(e, E_e)\}} \quad (1)$$

We have deployed this ranking strategy in the ANT search engine, manually tuning the weights in order to prioritize the number of days to the event, since close events are more relevant, then considering the implicit feedback based on the entity click score and only then the entity popularity score. The version we deployed only relies on entities of type Person and Organization and uses $w_1 = 0.5$, $w_2 = 0.2$ and $w_3 = 0.3$. We will, in the future, measure the impact of this change in ranking strategy through implicit feedback from news event clicks, in particular looking for an increase in the number of clicks, and we will also experiment with the automatic tuning of the weights.

## 6 Conclusions

We presented a complete implementation of an information extraction pipeline, from data acquisition to knowledge base population. During this process, we covered different approaches, either based on machine learning or handcrafted rules. We took advantage of corpora that was directly integrated into the NLTK library, but also of our own manually annotated documents. We also built regular expressions for relation extraction, and ontology mapping rules for building triples to load into a knowledge base. We then described a practical application of the extracted information, to the area of information retrieval, where we tackled the problem of query-independent ranking for a corpus of news articles announcing events. In particular, we proposed a temporal ranking approach combined with automatically compiled knowledge about the events being ranked. Through this, we showcased how *partOf* relations can be harnessed to influence the ranking of documents beyond directly mentioned entities, by taking advantage of related entities, mentioned in different documents.

The future of search engine intelligence highly depends on the unified efforts of information extraction and information retrieval. While we already have high quality machine learning techniques to support search by modeling "thought through numbers", we still lack the ability to effectively model "thought through language" in order to build search engines that can better assist users, not only by better understanding them, but also by helping them sort through a large amount of information locked within textual documents in natural language.

## References


1 Michele Banko, Michael J. Cafarella, Stephen Soderland, Matthew Broadhead, and Oren Etzioni. Open information extraction from the web. In *20th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 2670–2676, 2007.

2 Hannah Bast, Björn Buchhold, and Elmar Haussmann. Semantic search on text and knowledge bases. *Foundations and Trends in Information Retrieval*, 10(2–3):119–271, 2016.

3 Steven Bird, Ewan Klein, and Edward Loper. *Natural Language Processing with Python*. O'Reilly, 2009.

4 Nuno Cardoso. REMBRANDT - reconhecimento de entidades mencionadas baseado em relações e análise detalhada do texto. In Cristina Mota and Diana Santos, editors, *Desafios na avaliação conjunta do reconhecimento de entidades mencionadas: O Segundo HAREM*, pages 195–211. Linguateca, 2008.

5 Hamish Cunningham, Valentin Tablan, Ian Roberts, Mark A Greenwood, and Niraj Aswani. Information extraction and semantic annotation for multi-paradigm information management. In Mihai Lupu, Katja Mayer, Noriko Kando, and Anthony Trippe, editors, *Current Challenges in Patent Information Retrieval*, pages 307–327. Springer, 2011.

6 Hamish Cunningham, Yorick Wilks, and Robert Gaizauskas. GATE: a general architecture for text engineering. In *16th Conference on Computational Linguistics*, pages 1057–1060, 1996.

7 José Devezas and Sérgio Nunes. Index-based semantic tagging for efficient query interpretation. In *International Conference of the Evaluation Forum (CLEF)*, pages 208–213, 2016.

8 Xin Dong, Evgeniy Gabrilovich, Geremy Heitz, Wilko Horn, Ni Lao, Kevin Murphy, Thomas Strohmann, Shaohua Sun, and Wei Zhang. Knowledge vault: A web-scale approach to probabilistic knowledge fusion. In *20th International Conference on Knowledge Discovery and Data Mining (SIGKDD)*, pages 601–610, 2014.

**9** Erick Rocha Fonseca and João Luís G. Rosa. Mac-morpho revisited: Towards robust part-of-speech tagging. In *9th Brazilian Symposium in Information and Human Language Technology*, pages 98–107, 2013.

**10** Cláudia Freitas and Susana Afonso. *Bíblia Florestal: Um manual lingüístico da Floresta Sintá(c)tica*. Linguateca, 2007.

**11** Catherine Havasi, Robert Speer, and Jason Alonso. ConceptNet 3: a flexible, multilingual semantic network for common sense knowledge. In *Recent Advances in Natural Language Processing (RANLP)*, pages 27–29, 2007.

**12** Jian Hu, Gang Wang, Fred Lochovsky, Jian-tao Sun, and Zheng Chen. Understanding user's query intent with Wikipedia. In *18th International Conference on World Wide Web*, pages 471–480, 2009.

**13** John Lafferty, Andrew McCallum, and Fernando Pereira. Conditional Random Fields: Probabilistic models for segmenting and labeling sequence data. In *Eighteenth International Conference on Machine Learning (ICML)*, pages 282–289, 2001.

**14** Hang Li and Jun Xu. Semantic matching in search. *Foundations and Trends in Information Retrieval*, 7(5):343–469, 2014.

**15** Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S. Corrado, and Jeff Dean. Distributed Representations of Words and Phrases and their Compositionality. In *Advances in Neural Information Processing Systems*, pages 3111–3119, 2013.

**16** Cristina Mota and Diana Santos. *Desafios na avaliação conjunta do reconhecimento de entidades mencionadas: O Segundo HAREM*. Linguateca, 2008.

**17** David Nadeau and Satoshi Sekine. A survey of named entity recognition and classification. *Lingvisticae Investigationes*, 30(1):3–26, 2007.

**18** Kamel Nebhi. Named entity disambiguation using freebase and syntactic parsing. In *First International Conference on Linked Data for Information Extraction*, pages 50–55, 2013.

**19** Joakim Nivre, Johan Hall, and Jens Nilsson. MaltParser: A data-driven parser-generator for dependency parsing. In *The Fifth International Conference on Language Resources and Evaluation (LREC)*, pages 2216–2219, 2006.

**20** Pascal Pons and Matthieu Latapy. Computing communities in large networks using random walks. *Journal of Graph Algorithms and Applications*, 10(2):191–218, 2006.

**21** Ricardo Rodrigues, Hugo Gonçalo Oliveira, and Paulo Gomes. LemPORT: a high-accuracy cross-platform lemmatizer for portuguese. In *3rd Symposium on Languages, Applications and Technologies (SLATE)*, pages 267–274, 2014.

**22** Wei Shen, Jianyong Wang, and Jiawei Han. Entity linking with a knowledge base: Issues, techniques, and solutions. *Transactions on Knowledge and Data Engineering*, 27(2):443–460, 2015.

**23** Pontus Stenetorp, Sampo Pyysalo, Goran Topić, Tomoko Ohta, Sophia Ananiadou, and Jun'ichi Tsujii. BRAT: a web-based tool for NLP-assisted text annotation. In *13th Conference of the European Chapter of the Association for Computational Linguistics (EACL)*, pages 102–107, 2012.

**24** David Vallet, Miriam Fernández, and Pablo Castells. An ontology-based information retrieval model. In *European Semantic Web Conference*, pages 455–470, 2005.

# A Method for Proper Noun Extraction in Kurdish

**Hossein Hassani**

**University of Kurdistan Hewlêr, Erbi, Kurdistan Region, Iraq; and**
**Sarajevo School of Science and Technology, Sarajevo, Bosnia and Herzegovina**
**hosseinh@ukh.edu.krd, hossein.hassani@stu.ssst.edu.ba**

## Abstract

This paper suggests a method for proper noun identification in Kurdish texts. Kurdish proper nouns are not capitalized and they also assume other part-of-speech roles, which leads to a broad ambiguity that should be addressed in Kurdish proper noun recognition applications. Kurdish is also among less-resourced languages. We developed an application based on an architecture which includes a number of name lists, a set of rules, and a set of processes that recognizes Kurdish person names. This can help the study of Information Retrieval (IR) in Kurdish to advance and can also be used in Kurdish machine translation. We conducted several experiments which showed that the precision of the method is more than 95%, the recall is between 40% to 80%, and the $F$-measure is close to 60% to more than 80%. The reason for the low recall precision was because our name lists were not exhaustive enough to cover the vast majority of the Kurdish names.

**1998 ACM Subject Classification** I.2.7 Natural Language Processing

**Keywords and phrases** Proper Noun Recognition, Named Entity Recognition, Information Extraction, Natural Language Processing, Kurdish

**Digital Object Identifier** 10.4230/OASIcs.SLATE.2017.19

## 1 Introduction

This paper suggests a method for proper noun recognition in Kurdish texts. Proper nouns are not capitalized in the vast majority of Kurdish texts. Despite recent intentions for using capitalization which are written in Latin script, this is technically not possible with Persian/Arabic script because it does not support capitalization formats. Also many proper nouns might have other grammatical forms such as being used as a verb, adjective, and object name. This causes word sense ambiguity in processes such as machine translation, information retrieval, and semantic analysis [4]. Kurdish is also considered a less-resourced [6, 14, 24, 5]. For example, the language does not have annotated corpora to be used as the required resources for most of Natural Language Processing (NLP) activities [4, 24]. We suggest a method that uses two name dictionaries, a gazetteer, a set of trigrams extracted from an untagged corpus, and a small set of hand-crafted rules.

Proper nouns recognition is part of Information Extraction (IE) and a subcategory of Named Entity Recognition (NER). NER is an application in NLP which extracts person names, locations, organizations, and generally, named entities from a text. This application is related to several other NLP and Computational Linguistics (CL) applications such as Machine Translation (MT) and Information Retrieval (IR).

As far as we are aware, there is no NER for Kurdish at the time of writing this paper. Sheykh Esmaili et al. [25] are the only scholars who have provided a significant research in Kurdish IR and have recognized several issues in this context [25]. In this research, among different categories of named entities, we have focused on person names detection.


Kurdish is a language that is mainly spoken by native Kurds in Iran, Iraq, Turkey, and Syria, alongside Kurdish communities in other countries such as Armenia, Lebanon, Egypt [6]. The population who speak the language is estimated about 30 million [11, 6]. Kurdish is multi-dialect from the Indo-European root [7]. Although different scholars have categorized its dialects differently, a considerable majority refer to it as Northern Kurdish (Kurmanji), Central Kurdish (Sorani), Southern Kurdish, Gorani, and Zazaki that include several sub-dialects [3, 6, 14]. Kurdish is written using four different scripts, which are modified Persian/Arabic, Latin, Yekgirtû(unified), and Cyrillic. The popularity of the scripts differ according to the geographical and geopolitical situations. Latin script uses a single character while Persian/Arabic and Yekgirtû in a few cases use two characters for one letter [6] (e.g., "ڕ" in Persian/Arabic and "sh" in Yekgirtû for "û" and "ş" in Latin, respectively). The Persian/Arabic script is even more complex with its RTL and concatenated writing style auto[6].

Although capitalization is possible in Latin scripts, this method neither has widely been practiced nor it has been standardized in Kurdish. However, there are evidence of attempts of practicing capitalization in Latin based texts, though it mainly covers Kurmanji texts but not in the texts written in Sorani (see [20, 30]). There are other languages such as Persian, Arabic, and Chinese that face this issue in a similar manner, for which a number of studies have been conducted, which would be helpful in the Kurdish proper noun computational study.

## 1.1 Kurdish Person Names

From the semantic perspective, Kurdish person names are intended to carry a meaning related to tangible objects such as waterfall, spring, and rain. This characteristics can be seen in many languages, though to a different extent. For instance, in Native American (indigenous American) languages this is a very popular case [27]. A large number of person names in Kurdish are Islamic (Arabic) nouns. However, the usage of different category of names, rooted in Kurdish history or culture has been growing since a few decades ago. In this approach to naming, historical names such as mythical or legendary Kurdish names, and the name of entities in nature are used as person names [1]. Many of these names have multiple linguistics categories, for example they can also appear as nouns or adjectives.

To illustrate, Table 1 shows several examples Kurdish names alongside their possible part-of-speech (POS) formats. We illustrate how these examples cause word sense ambiguity in MT and IR through a few examples. As the first example, "wenewşe hatwe.", might be interpreted as "The violet has come.", as a sign of spring, or "Violet has come.", meaning that a female whose name is Violet has just arrived. As another example, "çaw le baran ke!" could be interpreted as "Look at Baran!", which intends to ask for looking at a female whose name is "Baran". Also it can be interpreted as "Look at the rain!". Again, "înca baran degêrêtewe" can be interpreted as "then it protects against rain" or as "then Baran is telling the story".

Another issue is unusual homographs for names that are originally non-Kurdish, but have been transformed in a way that have made them a homograph to Kurdish words. For instance, "brayim" or "birayim" as a nickname for "îbrahîm", which is a homograph that might be interpreted as an Arabic name or might be interpreted as "I am her brother" or "I am his brother". Many Arabic names have been transformed into nicknames in the Kurdish context that sometimes officially appear in legal documents and ordinary texts as well. For example, "xule" for "xizir", "ebe" and "ewla" for "abudullah", and "bile" for "îbrahîm". This diversity can be seen for other foreign names such as English as well for which there is no standard that should be followed.

**Table 1** Examples of Person Names in Kurdish – The first column shows the Kurdish name, the second shows the meaning of the name, and third shows the possible POS that the name might have in a sentence.

| Name | Sense | Other POS formats |
|---|---|---|
| akam | result | Noun |
| amanj | goal | Noun |
| aso | horizon | Noun |
| azade | liberate, free | Noun, PP Verb |
| baran | rain | Noun |
| hawkar | colleague | Noun, Verb |
| sakar | simple, basic | Noun |
| tavge | waterfall | Noun |
| vareen | rain | Verb, Noun |
| wenewşe | violet | Noun, Adjective |

**Table 2** Examples of Arabic names in Kurdish – The first column shows the names in Arabic, the second shows the different orthographic appearance of the names, and the third shows different abbreviations of the names.

| Arabic | Kurdish (orthographic formats) | Nickname |
|---|---|---|
| احمد | ehmed, ahmad | ehe, aha |
| حسين | husên, husen, hussên | wuze |
| محمد | mohammad, mohamed, muhemed, muhemmed | heme, mihe |
| قادر | kadir, qadir, ghader | kale, gale, ghalah |
| عثمان | othman, usman, osman | wetman |

Table 2 shows a number of samples of Arabic names and some of their different formats in Kurdish alongside the nicknames that are used for the names. We have only showed the Latin version of Kurdish in this table. One can realize the issues that NER might face if one considers all these varieties.

Furthermore, another issue raises when many Arabic names that have multiple POS formats and their non-proper-name formats are also used in Kurdish. For instance, "جمال" which in Kurdish Latin script is written as "cemal" might mean "beauty" or "face" according to the context. In our approach, we have listed this kind of names in the Kurdish names dictionary in order to let the algorithm to apply the rules based on the words around the name (trigrams) and then make a decision about whether it should be taken as a proper name or not. These could have been done by adding features to a single dictionary and labeling each entity with appropriate category as Arabic, Kurdish, and such. However, we preferred to keep the lists in different dictionaries in this development stage. The way that this case is tackled might affect the performance of the devised algorithm either positively or negatively, but according to our project schedule and objectives, we did not implement more than one version of this application.

Among the other issues, we would like to address the problem with geographical names coming from the national or formal languages of the countries were Kurds located. For example, China and Austria are called 'sîn" and "nemsa" respectively, in Iraqi Kurdistan

that have been borrowed from Arabic. They are called "çîn" and "otrîş" in Iranian Kurdistan that are the way that these countries are called in Persian.

The rest of this paper is organized in the following sections. Section 2 reviews the related work. Section 3 discusses the methodology and presents the suggested method including an architecture, data collection steps, and a devised algorithm to implement the method. Section 4 presents the evaluation method and reports on the conducted experiments. Section 5 discusses the outcome of the experiments. Finally, Section 6 provides the conclusion and addresses the future work.

## 2 Related Work

Similar to other topics in NLP and CL, NER is significantly language and application specific. NER has been a focal point for many years and it has well-established architectures, practical approaches, and solutions for diverse applications in widely-studied languages such as English [10]. A large population of NLP and CL researchers worked on NER and have suggested practical approaches to the subject [15, 18, 26, 16, 2, 8].

Although all of these resources are helpful, most of them are working based on existence of proper computational resources. This is not the case in Kurdish NLP and CL, therefore, we review the works which have targeted the languages such as Chinese and Arabic which have similar proper nouns characteristics to Kurdish, for example, lack of capitalization. We also review the related work on languages such as Urdu which not only do not apply capitalization but also considered as less-resourced languages.

Sun et al. [26] provide an NER application based on statistical approach for Chinese [26]. Chinese NER faces another problem which is lack of space for marking word boundaries. However, researchers of this work rely on manually tagged data sets large enough for statistical/probabilistic approach, which is currently not applicable for Kurdish.

Tsai et al. [29, 28] report on "Mencius" an NER for Chinese in which they have used a hybrid model by combining rule-based and Machine Learning (ML) based approaches [28, 29]. They perform three experiments one of which is a rule-based method which uses name list and gazetteers with 32000 entries. The second one is an ML based experiment and the third is a hybrid approach that combines the two previous methods. The results show different level of accuracy for different type of entities. The ML approach is not applicable in the current situation of Kurdish hence we are interested in the first method of this study which is applicable in the absence of an established Language Model (LM) and corpora.

Shaalan and Raza [22] have developed a proper noun recognizer for Arabic using rule-based methods. They have suggested an architecture that has three major blocks. A gazetteer that includes several dictionaries, a grammar configuration that recognizes person names using regular expression patterns, and a filtration mechanism to reject invalid person names. They have reported that their system achieved 85.5% for the precision and 89% for recall measures [22].They have expanded and slightly modified their work in [23]. A recent survey on Arabic NER, [21] reports that both rule-based and ML based approaches have been successful hence a hybrid method has been suggested that utilizes the advantaged of each method in a single one.

Riaz has suggested a rule-based NER for Urdu [19]. The author provides a list of challenges that Urdu NER faces, several of which are common with Kurdish such as capitalization, ambiguity, spelling variations, loan words, and resource challenges. The suggested method uses the hand-crafted rules to form a Finite State Automata (FSA) based on lexical indications. The rules are categorized as corpus-based, heuristic-based, and grammar-based.

The approach also utilizes a 6-gram that have been extracted form available Urdu corpora. The paper reports that the results showed promising performance when the method is compared with other NER methods.

## 3 Methodology

The "practical architecture"s which have been applied by the mentioned researchers in Section 2 are not practical in our research for the lack of required resources. As a result, we have developed a revised architecture that fits the current situation of Kurdish NLP and CL which is in its infancy.

Our approach to the person name recognition is benefited from the work by Shaalan and Raza that suggest a rule-based person name recognition in Arabic [22]. It is also based on the work by Tsai et al. who proposes a hybrid model by combining rule-based and Machine Learning [28]. Furthermore, we consider the work by Riaz which provides a rule-based NER for Urdu [19].

However, our approach differs, in different ways, from what have been proposed in the mentioned studies. First, in our architecture the arrangement and type of dictionaries are different. This rearrangement and categorization enables the method to accept other types than person names and also simplifies the arrangement of dictionaries. For example, having locations as a separate list, allows the application to be expanded to recognize the location in the future. In fact, because no annotated corpus currently exists for Kurdish, we cannot label the entity types by using probabilistic methods. Therefore, in our architecture we have separated the name lists. Obviously, the hand-crafted rules also differ from the suggested methods because of these rules are mainly language-specif. In addition, we have devised and presented an algorithm that shows the implementation configuration.

### 3.1 Architecture

Figure 1 shows the proposed architecture for Kurdish proper nouns recognition.

Below the components of the architecture are explained.

- A Gazetteer- This includes a subset of proper names as it is used in most of more established NER approaches.
- A dictionary of Kurdish names- It includes person names that mainly have Kurdish origins.
- A dictionary of Arabic names- It includes Arabic names that can certainly be identified in Kurdish texts. The nickname/moniker equivalents are kept in item 2 (The dictionary of Kurdish names).
- A set of rules- The rules are hand-crafted and include words that could appear before or after a proper noun allowing the candidate names to be taken as proper nouns with a higher probability. For example, if any member of the subset ("aga", "axa", "beg", "xatûn", "xanim") appears after a candidate name, the candidate name is considered as a found name.

### 3.2 Data Collection

We collected data in several steps and organized them in different data containers as we will present them below.

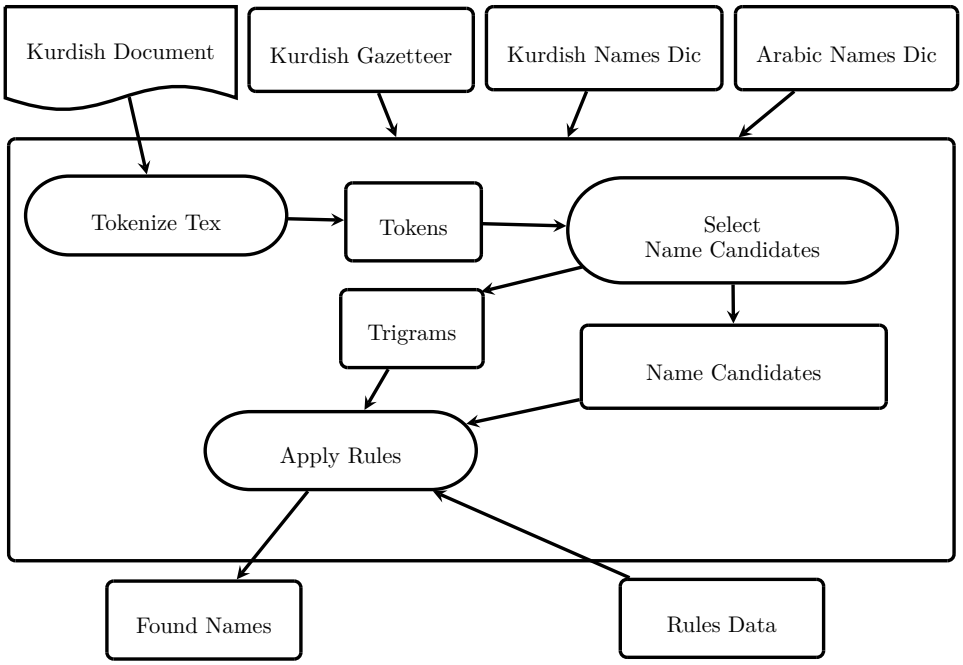



**Figure 1** Kurdish PNR Architecture - The architecture is based on three dictionaries and a set of rules which is used by the internal part of the system to find out the Proper Names.

### 3.2.1 Gazetteer

The gazetteer was manually created. It includes geographical names and famous person names including a number of well-known people such as poets, philosophers, and leaders. It is a developing list and was not intended to be complete for this research because the aim of this research is to recognize person names. The data was collected from different online resources such as GeoNames[1] and wherever needed the transliterated and their diacritics were unified.

However, at this stage we have not intended to recognize location entities hence we have not focused on data collection for this item. We will address this case again when we discuss the future work in Section 6.

### 3.2.2 Kurdish Names Dictionary

The data for Kurdish names dictionary was collected from various available online resources. The names written in Persian/Arabic were transliterated to Latin and transformed into their lower case before saved. Also the names that were not Kurdish or Arabic, for example, Assyrian and Chaldean names were inserted into this dictionary at this version of Kurdish PNR.

Because diacritics are not unified in Kurdish texts, for example, “ ‘ ” is used instead of “ˆ” in some texts, therefore, the diacritics were also unified (harmonized) according to the more accepted version, which is “ˆ”. Afterwards, duplicates were removed from the list and the list alphabetically sorted and it was written into a text file. Finally, although Arabic

[1] Available from `http://www.geonames.org/`.

names are popular in Kurdish communities, we have separated these names from the original Kurdish names.

We used [13, 12, 9] for the name lists construction. We also used name entries from the list of accepted students to the universities of Iraqi Kurdistan region, which have been announced by Ministry of Higher Eduction and Scientific Research (Kurdistan Regional Government) [17]. Although this latter case could have improved our lists in terms of the number of entries, for various technical reasons we were not able to automatically convert the list to the format that our application could handle, therefore, we manually processed the list and added some of the names that we did not have in our names lists.

### 3.2.3 Arabic Names Dictionary

Many Arabic names are popular in Kurdish communities. There are several historical reasons for this phenomenon, the discussion of which is beyond the scope of this dissertation. However, since several decades ago the usage of Kurdish names has significantly increased. Nevertheless, Arabic names are quite popular and hence they appear in many texts very frequently. This dictionary includes these names.

Despite the popularity of Arabic names in Kurdish, they do not follow the same spelling and orthography as they are used in Arabic. Also there is no orthographic standard for writing these names, hence there might be several versions of a single name no matter if Latin or Persian/Arabic script has been used (see Table 2).

We created Arabic names list based on our knowledge with regard to the Kurdish culture and familiarity with both Arabic and Kurdish languages.

### 3.2.4 Rules Set

The rule set consists of four lists as below:

- List of salutations that appear before a proper noun such as “mamosta” that is used to call a teacher or lecturer or a Muslim cleric or “kak’ that is used to friendly and respectfully call a male person, literally meaning “big-brother”.
- List of salutations that appear after a proper noun such “axa” or “xan”, which appear after a male and female person, literally meaning “sir” and “madam”. The second salutation is used for male person in Iranian Kurdistan.
- List of words that appear before a proper noun.
- List of words that appear after a proper noun.

We hand-crafted these rules based on our familiarity with Kurdish. We do not claim that this rules are complete rather we have suggested them to show that our approach is practical.

## 3.3 Algorithm

An algorithm has been developed in order to recognize proper nouns in Kurdish texts. The algorithm has been presented below. It reads the required dictionaries as its knowledge base, defines a number of lists to hold the candidate names, trigrams, and found names, and reads the input text. The input text will be tokenized and then processed by matching each token with the dictionaries. The algorithm is able to apply a stemming process, by calling a stemmer, on the token under investigation if it did not find it in the name lists. The stemmer acts as a suffix stripper in this case. Similar to many other languages, proper nouns can have suffixes in Kurdish. When a proper noun has a suffix, it cannot be found in

name dictionaries. We need a stemmer that is able to strip the suffix of the token in order to produce a proper searching item for the name dictionaries. For example, in the sentences "baran**im** dît." if we strip the bold letters, we might have a person name. As another example, in the expression "le kurdistan**ê**" if we strip the bold letter, we will definitely have a proper noun. The proper nouns are shown by underlining them in both mentioned cases.

**Algorithm 1** Kurdish Person Name Recognition.

```
function findKurdishPersonName(inputText)
    nameCandidates ← null
    nameCandidatesRuleApplied ← null
    trigramsAll ← null
    trigramsRuleApplied ← null
    isNameCandidate ← false
    isCheckedCandidate ← false

    read KurdishNames, KurdishGazetteer, ArabicNames, Rules
      ▷ Notice: The order of evaluation is important. It should be ordered according to the number
        of entries in each list.
    for token in inputText do

        if token is in KurdishGazetteer or token is in ArabicNames then
            isCheckedCandidate ← true
        ▷ Notice: If the token is not found as it is, stem the token and compare the result.
        else if token is in KurdishNames or stemmed-token is in KurdishNames then
            isNameCandidate ← true
            if Rules Apply or (either predecessor-token or successor-token) is in
                                (KurdishGazetteer or ArabicNames or KurdishNames) then
                isCheckedCandidate ← true
            end if
        end if

        if isCheckedCandidate then
            append token to nameCandidates
            append token to nameCandidatesRuleApplied
            trigram ← concat(predecessor, token, successor)
            append trigram to trigramsAll
            append trigram to trigramsRuleApplied
            isCheckedCandidate ← flase
        else if isNameCandidate then
            append token to nameCandidates
            trigram ← concat (predecessor, token, successor)
            append trigram to trigramsAll
            isCheckeCandidate ← flase
        end if

    end for

    return nameCandidates, nameCandidatesRulesApplied, trigramsAll, trigramsRulesApplied
end function
```

## 4 Evaluation

We used 15 documents of different sizes to test the accuracy of the method, ranging from several hundred to several thousand tokens, of which 8 documents were in Kurmanji and 7 were in Sorani. We ran the suggested algorithm once without and once with the stemming process on each document and saved the results separately. We also manually extracted the person names from each input text. The outputs of the algorithm, the suggested names, were compared against the manually extracted names whereby we calculated the Precision and Recall parameters and the $F$-measure for each document.

**Table 3** Kurdish PNR in Kurmanji Texts. TP: True Positive, FP: False Positive, and FN: False Negative were extracted by examining the results against the texts. P: Precision, R: Recall, and F: *F*-measure were calculated using Equations 2, 3, and 1 respectively. The last row shows the average performance of the experiments.

| No | Size (Words) | **TP** | **FP** | **FN** | **P** | **R** | ***F*** |
|---|---|---|---|---|---|---|---|
| 1 | 1605 | 7 | 0 | 3 | 1 | 0.7 | 0.82 |
| 2 | 1798 | 4 | 0 | 6 | 1 | 0.4 | 0.57 |
| 3 | 1910 | 11 | 1 | 6 | 0.92 | 0.65 | 0.76 |
| 4 | 2200 | 8 | 0 | 6 | 1 | 0.57 | 0.73 |
| 5 | 2300 | 10 | 0 | 9 | 1 | 0.53 | 0.69 |
| 6 | 2112 | 11 | 1 | 6 | 0.92 | 0.65 | 0.76 |
| 7 | 2520 | 9 | 0 | 5 | 1 | 0.64 | 0.78 |
| 8 | 2400 | 11 | 2 | 7 | 0.85 | 0.61 | 0.71 |
| | | | | **Average** | 0.96 | 0.59 | 0.73 |

The classic evaluation method for NER is based on *Precision, Recall*, and $F_1$ measure as shown in Eq. (1). In our case, *Recall* is the ratio of the number of correctly found names to the total names in the sample. *Precision* is the ratio of the number of correctly found names to the total found names.

$$F_1 = \frac{2PR}{P+R} \tag{1}$$

$P$ and $R$ are calculated by the following formulas:

$$P = \frac{T_P}{T_P + F_P} \tag{2}$$

$$R = \frac{T_P}{T_P + F_N} \tag{3}$$

In Eq. (1), $P$ stands for Precision and $R$ for Recall. In Equations (2) and (3), $T_P$ stands for *True Positive*s, which in our case are correctly recognized Person Names; in Eq. (2), $F_P$ stands for *False Positive*s, which are wrongly recognized Person Names; and in Eq. (3) $F_N$ stands for *False Negative*s, which are unrecognized Person Names.

Table 3 shows the result of experiments for Kurmanji texts. Table 4 shows the result of experiments for Sorani texts.

Currently there is no NER for Kurdish, hence we compared the results with the outcome of the works which we have addressed in Section 2. Shaalan and Raza report a *Precision* between 84.2% to 94%, a *Recall* between 84.7% to 96.8%, and an *F-measure* between 84.4% to 95.1% [22]. Riaz reports a *Precision* of 91.5%, a *Recall* of 90.7%, and an *F-measure* of 91.1% for one data set and an *F-measure* between 72.4% to 81.6% for another one [19]. Our method shows a *Precision* between 80% to 100%, a *Recall* between 40% to 77%, and an *F-measure* between 57% to 87%. Close investigations of the results, by looking into the manually extracted names, showed that the majority of unrecognized Person Names were non-Kurdish names.

## 5 Discussion

As Tables 3 and 4 show, the proposed architecture and the devised algorithm work with a high precision in most of the cases tested. However, they also show that the recall ratio is

**Table 4** Kurdish PNR in Sorani Texts. TP: True Positive, FP: False Positive, and FN: False Negative were extracted by examining the results against the texts. P: Precision, R: Recall, and F: *F*-measure were calculated using Equations (2), (3), and (1) respectively. The last row shows the average performance of the experiments.

| No | Size (Words) | **TP** | **FP** | **FN** | **P** | **R** | ***F*** |
|---|---|---|---|---|---|---|---|
| 1 | 1805 | 7 | 0 | 3 | 1 | 0.7 | 0.82 |
| 2 | 1994 | 4 | 0 | 6 | 1 | 0.4 | 0.57 |
| 3 | 2507 | 12 | 0 | 7 | 1 | 0.63 | 0.77 |
| 4 | 2302 | 10 | 0 | 3 | 1 | 0.77 | 0.87 |
| 5 | 2105 | 10 | 0 | 3 | 1 | 0.77 | 0.87 |
| 6 | 2900 | 11 | 1 | 4 | 0.92 | 0.73 | 0.81 |
| 7 | 2320 | 8 | 2 | 3 | 0.8 | 0.73 | 0.76 |
| | | | **Average** | | 0.96 | 0.68 | 0.78 |

very low in several cases and low in majority of them. The values show that the method is performing in the acceptable boundary providing a better upper-bound for *Precision* when it is compared to [22] and [19]. However, its *Recall* does not perform as its *Precision* which causes the *F-measure* to have a low upper-bound. This is because the name lists, the dictionaries, should be expanded to cover more names. Our dictionaries mainly cover popular names in Iraqi Kurdistan, while person names in other Kurdish speaking regions have been influenced by other countries culture such as Iranian, Turkish, and Syrian culture. Moreover, the same name might be spelled differently, which means that all formats of the names must be found and recorded. < This shows that although the proposed method works properly, the underlying data that are based on the fundamental components of the proposed architecture in Section 3 must be expanded and enriched more in order to cover foreign names as well. The investigation also showed that a number of unrecognized names has been captured and written in different format than what could have been found in the supporting dictionaries.

Importantly, we assessed the case of stemming and trigrams. The results revealed stemming has a significant impact on recall and trigrams considerably affect the precision. Figures 2 and 3 below show these findings for Kurmanji and Sorani dialects respectively. Differences between Kurmanji and Sorani results are mainly coming from two aspects. The first aspect is the usage of Turkish names in the Kurdish community in Turkey and the second is the stemming accuracy level. For the Sorani dialect the usage of other Iranian names, such as those that are mainly considered as Persian names affects texts that are written based on the Iranian sub-dialect of Sorani. However, whether increasing the order of n-grams to a higher order, for example four or five, leads to an improved recall or precision is yet to be investigated.

Finally, it is worth mentioning that although we have incorporated gazetteer into the architecture, geographical and place names have not been our target, hence their recognition has not participated in the evaluation process.

## 6 Conclusion

In this paper we presented the challenges that exist in Kurdish Named Entity Recognition. The research focused on Person names recognition. The result showed that the presented

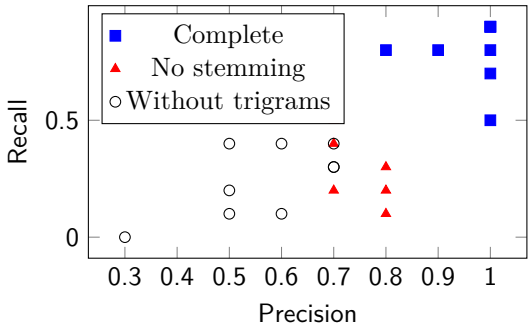



**Figure 2** The Impact of Stemming and Trigrams on Precision-Recall (Kurmanji).

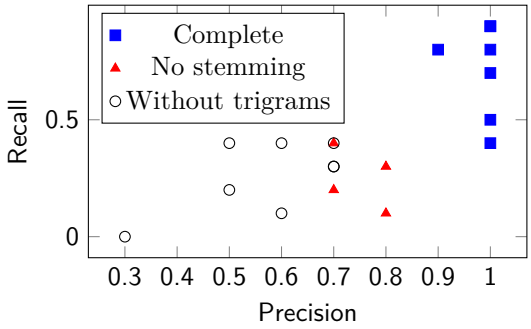



**Figure 3** The Impact of Stemming and Trigrams on Precision-Recall (Sorani).

method performed well in the absence of underlying data that is necessary for other well-established approaches to NER. The precision of the method is between 80% to 100% with the average more than 95%, the recall is between 40% to 80% with the average of more than 60%, and the $F$-measure is close to 60% to more than 80%.

## 6.1 Future Work

There are areas that should be studied further with regard to PNR, and in a general sense NER, in Kurdish. Most importantly, the method should be improved to be able to recognize not only person names but also entities in the general sense such as locations and organizations. For the geographical names, we expect that the expansion of Gazetteer and enhancing the rule-set would do the purpose. However, for the entities of organization type the case must be properly investigated. Moreover, the gender categorization should be implemented in person names recognition. Also the efficiency of trigrams should be tested and compared to other orders of n-grams in order to find the optimum format. Furthermore, the multi-segment names should also be considered and recognized properly. In addition, the dictionaries must be expanded to cover names from other Kurdish speaking regions in other countries. Similarly, the dictionaries should be augmented by different spellings of names.

Equally important, the method should be revisited when the Kurdish NLP and CL are matured enough by having proper language models that allow researchers to apply probabilistic approaches and ML-based NER. A comparison of combination of the methods, alongside the application of each method independently, could be the subject of another series of research.

**Acknowledgements.** We would like to express our warm appreciations to Dr. Dzejla Medjedovic an Assistant Professor and Vice Dean of Graduate Program at the University Sarajevo School of Science and Technology (SSST) for reviewing this paper and providing influential recommendations. We would also like to thank the anonymous reviewers for their constructive suggestions.

## References

1 Lazgin Al-Barany, Asma Albamarni, and Dilggash M. Shareef. Kurdish personal names in Kurdistan of Iraq: A sociolinguistic perspective, 2014. https://www.academia.edu/9662401/Kurdish_Personal_Names_in_Kurdistan_of_Iraq_A_Sociolinguistic_Perspective.

2 Silviu Cucerzan. Large-scale named entity disambiguation based on Wikipedia data. In *Conference on Empirical Methods on Natural Language Processing and Computational Natural Language Learning*, volume 7, pages 708–716, 2007.

3 Geoffrey Haig and Ergin Öpengin. Introduction to special issue-Kurdish: A critical research overview. *Kurdish Studies*, 2(2):99–122, 2014.

4 Hossein Hassani. Kurdish interdialect machine translation. In *VarDial Workshop*, pages 63–72, April 2017.

5 Hossein Hassani and Rahel Kareem. Kurdish text to speech (KTTS). In *Tenth International Workshop on Internationalisation of Products and Systems*, pages 79–89, 2011.

6 Hossein Hassani and Dzejla Medjedovic. Automatic Kurdish dialects identification. *Computer Science & Information Technology*, 6(2):61–78, 2016.

7 Amir Hassanpour. *Nationalism and language in Kurdistan, 1918-1985*. Edwin Mellen Pr, 1992.

8 Ulf Hermjakob, Kevin Knight, and Hal Daumé III. Name translation in statistical machine translation-learning when to transliterate. In *Association for Computational Linguistics*, pages 389–397, 2008.

9 Hesami. Kurdish definition, origin and usage of names, 2016. http://www.hesami.com/names/kurdish/.

10 Daniel Jurafsky and James H. Martin. *Speech and Language Processing*. Prentice Hall, 2 edition, 2008.

11 Kurdish Academy of Languages. The Kurdish Population, 2016. http://www.kurdishacademy.org/?q=node/199.

12 Kurdish Daily. Kurdish names for your baby, 2016. http://ekurd.net/mismas/kurdishnames.htm.

13 Kurdish Institute of Paris. Kurdish Names, 2016. http://www.institutkurde.org/en/kurdorama/kurdish_baby_names.php.

14 Shervin Malmasi. Subdialectal differences in Sorani Kurdish. In *Third Workshop on NLP for Similar Languages, Varieties and Dialects*, pages 89–96, 2016.

15 Inderjeet Mani, T Richard MacMillan, Susann Luperfoy, Elaine Lusher, and Sharon Laskowski. Identifying unknown proper names in newswire text. In *Workshop on Acquisition of Lexical Knowledge from Text*, pages 44–54, 1993.

16 Gideon S. Mann and David Yarowsky. Unsupervised personal name disambiguation. In *Seventh conference on Natural language learning at HLT-NAACL*, volume 4, pages 33–40, 2003.

17 Minstray of Higher Education and Scientific Research. Admitted students in 2010, 2016. http://www.mhe-krg.org/ku/node/698.

18 Thierry Poibeau and Leila Kosseim. Proper name extraction from non-journalistic texts. *Language and Computers*, 37(1):144–157, 2001.

**19** Kashif Riaz. Rule-based named entity recognition in Urdu. In *Named Entities Workshop*, pages 126–135, 2010.

**20** Rudaw, 2015. `http://rudaw.net/sorani`.

**21** Khaled Shaalan. A survey of arabic named entity recognition and classification. *Computational Linguistics*, 40(2):469–510, 2014.

**22** Khaled Shaalan and Hafsa Raza. Person name entity recognition for Arabic. In *Workshop on Computational Approaches to Semitic Languages: Common Issues and Resources*, pages 17–24, 2007.

**23** Khaled Shaalan and Hafsa Raza. NERA: Named entity recognition for arabic. *Journal of the American Society for Information Science and Technology*, 60(8):1652–1663, 2009.

**24** Kyumars Sheykh Esmaili. Challenges in Kurdish text processing. *arXiv preprint arXiv:1212.0074*, 2012.

**25** Kyumars Sheykh Esmaili, Shahin Salavati, and Anwitaman Datta. Towards Kurdish information retrieval. *Transactions on Asian Language Information Processing (TALIP)*, 13(2):7, 2014.

**26** Jian Sun, Jianfeng Gao, Lei Zhang, Ming Zhou, and Changning Huang. Chinese named entity identification using class-based language model. In *19th International Conference on Computational Linguistics*, pages 1–7, 2002.

**27** Tribal Directory. American Indian Names, 2016. `http://tribaldirectory.com/information/american-indian-names.html`.

**28** Tzong-Han Tsai, Shih-Hung Wu, and Wen-Lian Hsu. Mencius: A Chinese named entity recognizer using hybrid model. In *Research on Computational Linguistics Conference XV*, pages 193–209, 2003.

**29** Tzong-Han Tsai, Shih-Hung Wu, Cheng-Wei Lee, Cheng-Wei Shih, and Wen-Lian Hsu. Mencius: A chinese named entity recognizer using the maximum entropy-based hybrid model. *International Journal of Computational Linguistics and Chinese Language Processing*, 9(1), 2004.

**30** Wikipedia. Hûn bi xêr hatin Wîkîpediyaya kurdî, 2016. `https://ku.wikipedia.org/wiki/Destp%C3%AAk`.

# Natural Transmission of Information Extraction Results to End-Users – A Proof-of-Concept Using Data-to-Text*

**José Casimiro Pereira[1], António J. S. Teixeira[2], Mário Rodrigues[3], Pedro Miguel[4], and Joaquim Sousa Pinto[5]**

1 **Politecnic Institute of Tomar (IPT), Tomar, Portugal**
casimiro@ipt.pt
2 **Department of Electronics, Telecommunications and Informatics/IEETA, University of Aveiro, Aveiro, Portugal**
ajst@ua.pt
3 **ESTGA/IEETA, University of Aveiro, Aveiro, Portugal**
mjfr@ua.pt
4 **Department of Electronics, Telecommunications and Informatics/IEETA, University of Aveiro, Aveiro, Portugal**
5 **Department of Electronics, Telecommunications and Informatics/IEETA, University of Aveiro, Aveiro, Portugal**

## Abstract

Information Extraction from natural texts has a great potential in areas such as Tourism and can be of great assistance in transforming customers' comments in valuable information for Tourism operators, governments and customers. After extraction, information needs to be efficiently transmitted to end-users in a natural way. Systems should not, in general, send extracted information directly to end-users, such as hotel managers, as it can be difficult to read.

Naturally, humans transmit and encode information using natural languages, such as Portuguese. The problem arising from the need of efficient and natural transmission of the information to end-user is how to encode it. The use of natural language generation (NLG) is a possible solution, for producing sentences, and, with them, texts.

In this paper we address this, with a data-to-text system, a derivation of formal NLG systems that use data as input. The proposed system uses an aligned corpus, which was defined, collected and processed, in about approximately 3 weeks of work. To build the language model were used three different in-domain and out-of-domain corpora. The effects of this approach were evaluated, and results are presented.

Automatic metrics, BLEU and Meteor, were used to evaluate the different systems, comparing their values with similar systems. Results show that expanding the corpus has a major positive effect in BLEU and Meteor scores and use of additional corpora (in-domain and out-of-domain) in training language model does not result in significantly different performance.

The scores obtained, combined with their comparison with other systems performance and informal evaluation by humans of the sentences produced, give additional support for the capabilities of the translation based approach for fast development of data-to-text for new domains.

**1998 ACM Subject Classification** I.2.7 [Natural Language Processing] Language Generation, Machine Translation, H.5.2 [User Interfaces] Natural Language

**Keywords and phrases** Data-to-Text, Natural Language Generation, Automatic Translation, opinions, Tourism, Portuguese

**Digital Object Identifier** 10.4230/OASIcs.SLATE.2017.20

* Research partially funded by IEETA Research Unit funding (UID/CEC/00127/2013) and Marie Curie Actions IRIS (ref. 610986, FP7-PEOPLE-2013-IAPP).



6th Symposium on Languages, Applications and Technologies (SLATE 2017).
Editors: R. Queirós, M. Pinto, A. Simões, J. P. Leal, and M. J. Varanda; Article No. 20; pp. 20:1–20:14

Open Access Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

**To:** *Director of HOTEL XYZ Aveiro*
**Subject:** March 2017 evaluations

Your hotel had a mixed review in restaurant service AND bad at room service. The price is high. Your competitor, Hotel YYY, had good evaluation at restaurant service AND very good at room service. Their price was considered good.
2 April 2017 *[Automatically generated by IE + NLG]*

**Para:** Director do Hotel XYZ de Aveiro
**Assunto:** Avaliacões do mês de março 2017

O seu Hotel teve avaliação mista nos serviços de restauração E péssima nos quartos. O preço é alto. O seu concorrente, Hotel YYY, teve avaliação boa nos serviços de restauração E muito boas nos quartos. O preço dele foi classificado como bom.
2 de Abril 2017
*[Gerado automaticamente por EI + GLN]*

**Figure 1** Example of hypothetical email with hotels' evaluations. English version at the top.

## 1 Introduction

User comments about service experiences have a key role in influencing the consumption decisions of other customers. The widespread adoption of Internet technologies expanded enormously the volume of such comments and the potential impact of such customer generated content. An area where user opinions are particularly influential is Tourism, as hotels, restaurants, and attractions are constantly commented by users.

For service providers, such as hotel owners and managers, this feedback provided by customers is very important, but it is impossible to even grasp the information in all this comments by manual processes. Information Extraction (IE) from natural texts (e.g. [12]) can be of great assistance in processing customers comments and extracting relevant information, but systems cannot, in general, send extracted information unfiltered to an end-user, such as an hotel manager, as the raw information can be hard to understand. Gathered information needs to be efficiently transmitted to end-users in a natural way.

In this paper we address this problem, of efficient and natural transmission, with a data-to-text system, a derivation of formal Natural Language Generation (NLG) systems that use data as input (e.g. log events). The application scenario, selected for the development of a proof-of-concept, consists of having the data-to-text system as a "translator" to Portuguese sentences of the information extracted by IE from comments to hotels' services, made in Portuguese by their customers. IE provides information such as names of hotels, services, classifications given to services and the amount of people who classified services. The target, when system becomes completed, is the production of short text, expressing the opinions that customers give about hotels' services. This text, for example, could be sent by email to hotel managers each morning. Figure 1 presents a vision of such email.

This paper is organized as follows: in the next section, related work is presented, followed by an overview of the structure of proposed Data-to-Text system. Section 4 presents the scenario of application, and how the system was adapted to the domain. In Section 5 several examples are presented of system output and the evaluation of system variations. A comparison with similar systems was made too. The paper ends with conclusions, and acknowledgements.

## 2 Related Work (in Data-to-Text)

Reiter [10] defined *Data-to-Text* systems as *systems that generate texts from non-linguistic input data, which is typically numerical.* The major difference between 'classic' NLG systems

and Data-to-Text systems is that this last category must analyse and interpret their input data, whereas the other category does not need. Of course, they need, as well, to decide how to linguistically communicate the produced utterance. This section presents, briefly, some representative Data-to-Text systems, by chronological order, with focus on the ones employing data driven approaches.

**Pollen Forecast for Scotland system.** Pollen Forecast for Scotland system [15] was one of first Data-to-Text systems to be reported. Its aim is to report, on text, the prediction of pollen concentration on different regions of Scotland. It was made of two related sub-tasks: the prediction itself, and the translation of numerical data to text, provided by prediction. The translation task is based on a parallel corpus of 69 data-text pairs. Each pair corresponds to a pollen concentration data and the corresponding forecast. All forecasts were written by expert meteorologists.

To be able to generate correct texts, from meteorological data, all human written forecasts were analysed in respect to three dimensions: Message type; Data dependency and Corpus coverage. In addition, input data was analysed in three steps: segmentation of geographic regions by their non-spatial attributes (pollen values); segmentation of each segmented geographic regions by their spatial attributes (geographic proximity); and, detection of trends in the generalized pollen level for the whole region over time. With these segmented data, Pollen Forecast system produces vectors of trends for pollen prediction. These vectors were used to determine the correct forecast text to be produced.

**BabyTalk.** BabyTalk project [9] works with data from a Neonatal Intensive Care Unit (NICU), from a Scottish hospital. It tends to join two concepts: raw medical data and recommendation of specific actions to the medical staff. The aim is to produce a text summary, produced in Natural Language, where data and instructions should be put together.

BabyTalk has five different sub-projects, each one related with a particular issue of NICU centre, including: *BT-Nurse*, designed to automatically generate English summaries of the electronically recorded patient data over a twelve hour nursing shift; and *BT-45*, intended to present reports to nurses and doctors that summarize 45 minutes of patients' data.

BT-45 was the first to be implemented and is a truly data-to-text system. However, instead of only having numerical data as input, as weather reporting systems, BT-45's input is more heterogeneous. Most data comes from a NICU database, and includes an ontology that is used both to represent domain knowledge and to support linguistic processing. The BT-45 follows the data-to-text architecture suggested by [10], hence the presence of a Document Planning, Micro-Realization and Realization modules.

**Mountain.** Mountain was developed by Langner for his PhD [4]. Mountain was the only of the analysed systems that uses the Moses[1] tool. This statistical tool is used to train and process a parallel aligned corpora. The translation model built by Moses is used by Mountain to generate an output sentence, translated from input language.

Mountain's scenario was reservations of tennis court, where users make reserves from one to several hours. An aligned corpus was produced, with two languages. One with answers given to reservations, by English native speakers, and another with a set of three tokens expressing the requested scheduling. On the second language, the first token expresses the

[1] http://www.statmt.org/moses/

**Table 1** Example of MOUNTAIN's output. Left column: *input* sentence; right column: generated sentences, from [4, pag.71].

| | |
|---|---|
| 000000 d5 t3 | friday evening is completely closed |
| 100000 d2 t2 | the only time available is noon |
| 111111 d4 t1 | the court is open all morning |
| 111111 d1 t3 | you can reserve a court anytime on monday evening |
| 100011 d5 t3 | six , ten or eleven |

court availability. Each number represents one hour, over a day. `1` is for 'vacancy', and `0` is for 'taken'. Second token expresses the day of week, and last token represents the day period – morning, afternoon or evening. `111111 d2 t3` and `001110 d5 t1` are examples of it.

The collected corpus was expanded, from 800 to almost 4500 sentences, because it was realized that most sentences, produced by humans, with some minimal changes, could be reused, without losing their original meaning. After that, the corpus was trained with MOSES tools. The training model produced was responsible for translations carried out by MOUNTAIN.

Table 1 presents a sample of produced output sentences by MOUNTAIN. Some produced sentences were directly translated from *output corpus*, likely due to the presence of direct input–output pair on corpus. Others were the combination of two or more original sentences. Overall, more than three quarters of the generated responses are not present in the original corpus.

**SINotas.** SINotas [5] is one of few, for the best of our knowledge, data-to-text system that uses Portuguese language. In this case Brazilian Portuguese. The goal of SINotas is the production of short textual reports of students grades, weekly attendance rates, and other related academic information.

SINotas has a small corpus of 241 paired data-text records of students' academic performance [5]. All pairs were related to students of a single professor. The same professor was responsible for writing the text that expresses the contents in the the data. The use of a single professor, defined as 'domain expert', was justified with the need of coherence on meanings from raw data (e.g., students' grades) to semantics (i.e., the interpretation of the data according to a professor). The use of different 'domain experts' could generate contradictory description for the same data.

All output texts, that describe raw data, are up to five sentences long, offering students a description about their grades in relation with other students' grades.

**PortNLG.** PortNLG [14] is a recent tool that processes the last phase of NLG pipeline defined by Reiter [11] – the Surface Realization, on Portuguese language. PortNLG is a Java library, developed to be integrated on data-to-text systems, hence is not itself a real NLG system. It works similarly to SimpleNLG [3]. Like SimpleNLG, it uses grammar rules, but, since Portuguese grammar is more complex than English grammar, PortNLG needs an extra component: a lexicon module. The application where PortNLG is integrated, executes a sequence of calls to PortNLG methods, constructing, this way, the desired sentence.

**Medication2PT.** MEDICATION2PT [7] is a recent Data-to-Text work made to provide instructions in European Portuguese language about taking medicines. This system was

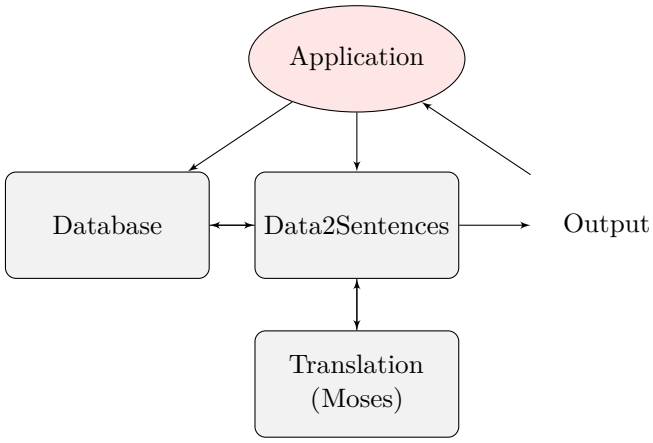



**Figure 2** Architecture of the Data-to-Text developed system.

integrated in a larger system, the Medication Assistant [2] designed to take advantage of Smartphones, and targeting elderly users.

The MEDICATION2PT was inspired by MOUNTAIN's work. It uses MOSES as main engine to process the input data and obtain output sentences. Since its intention is to build a system with low resources, it uses a small corpus to train MOSES. As a consequence, a small part of produced sentences has unacceptable quality, and can not be used. To identify sentences with lower quality, authors proposed a classifier module capable of providing information on the Intelligibility or Quality of the sentences. Sentences marked as unacceptable are replaced by template-based generated ones. This classifier module combines extraction of linguistic features with a classifier trained in a manually annotated corpus [8].

Later, in 2015, MEDICATION2PT classifiers and templates modules were integrated in a Hybrid System, proposing an integrated solution to produce sentences in Portuguese language, at a lower cost [8].

## 3 The Developed System

The developed system is based on the use of machine translation and draws on recent works, such as the MOUNTAIN and MEDICATION2PT, from the authors. The overall architecture of the developed system is presented in Figure 2.

The central part of this system is a module, named **Data2Sentences**, able to create sentences in response to vectors with data, provided as input. If several input vectors are sequentially fed to the Data2Sentence module, an aggregation module can be used to join the set of generated sentences to produce a small text.

The **Database** (DB) **module** is the component responsible for storing all data related with system where this module is used.

The **Translation module**, based in MOSES translation system, is responsible for the translation to Portuguese of the input vectors sent by the Data2Sentences module. To perform this task, MOSES must be trained with a corpus consisting of two languages perfectly aligned. Every sentence from first language, the *input language*, must match a phrase in the second language, the *output language*. In our system the input language consists of values related with the ones present on the DB module. The output language is constituted by expressions in Portuguese that represent the data on input language. MOSES uses this relation to generate new sentences, when an input vector is provided. Besides the definition

**Table 2** *Tokens* of input language and their correspondences.

| *Token* | Correspondence |
|---|---|
| Hotel name | name of hotel, under evaluation |
| Service name | name of hotel's service, that is going to be evaluated |
| Evaluation | evaluation to service |
| Number people | amount of people who did the evaluation |
| First adjective | main adjective that reflects evaluation |
| Second adjective | secondary adjective |

of languages and corpora, training Moses implies the creation of a language model. Because words in sentences do not combine in random order, the language model establishes the probability of one or more words appearing in a sentence, in a particular order. These probabilities are used in the process of translation. Commonly, language models consist of `n-grams` with probabilities, estimated from a training corpus of sentences on the language that Moses is expected to translate.

The **Data2Sentences module** is responsible for the coordination of all process. It receives requests from users and interacts with the data stored in the Database. It is also responsible for sending messages (written in the input language) to the Moses-based Translation module and receive its answer, in the output language. Finally, it is in charge of processing the answers and transmit them to the Application.

# 4 Corpora and training

## 4.1 Corpus structure

The first task is the definition of the input language and corresponding output language. The proposed work is integrated on an ongoing project, related with data extraction. The aim of that project is to collect and summarize data related with classifications to hotels' services, made by their customers. The provided data consists, mainly, in hotels names, service names, classifications given to services and the amount of people who classified the services. The input language is based on data stored on database module. A sentence on this language is a collection of tokens, in a specific order. Table 2 presents the tokens used in input language, and their order. The output language consists of Portuguese sentences, expressing, by words, the data of input language.

The first token corresponds to the hotel's name. The second token is related to the hotel's service name, that is subject to evaluation. The evaluation of the service is expressed on the Evaluation token, in three possible values: positive, negative, or neutral. The fourth token defines the amount of people that made the evaluation. It is configured with five possible values: very few people, some people, half of evaluators, many people, almost all people. Fifth and sixth tokens are related with adjectives used to express the evaluation to hotel's services. In all tokens, data is compressed and white spaces are replaced by hyphens. That is done to ease Moses work, because that way Moses treat the several words of tokens as a single word. To increase the accuracy of Moses, we need to use all tokens, in every input sentence, whether having, or not, data. When there is no data, the token related with that kind of data is identified with the word 'sem-valor' (no-data). A sample of our corpus is presented at Table 3.

**Table 3** Sample from aligned corpus ID1. For each example, first the real content (in Portuguese) is presented. After, in italic, English translations are presented.

| Input/Internal language | Corresponding Sentence |
|---|---|
| hotel-sem-valor servico-atendimento aval-neutra algumas-pessoas adj-diferente adj-aceitavel | algumas pessoas classificaram de forma neutra o atendimento, como diferente dos outros hotéis e aceitável |
| *hotel-no-data service-customer-service aval-neutral some-people adj-different adj-acceptable* | *some people classified as neutral the customer service, different from other hotels and acceptable* |
| hotel-faro-vintage-guest-house servico-restaurante aval-positiva poucas-pessoas adj-fantastico adj-sem-valor | Algumas pessoas acharam o restaurante do Faro-Vintage-Guest-House fantástico |
| *hotel-faro-vintage-guest-house service-restaurant eval-positive few-people adj-fantastic adj-no-data* | *Some people found the restaurant of Faro-Vintage-Guest-House fantastic* |

**Table 4** Corpus used in system's evaluation.

| Corpus | Num. Sentences | Description |
|---|---|---|
| *Corpora used to train the system* | | |
| ID1 (original) | 135 | sentences created by humans, in-domain |
| ID2 (extended) | 435 | An extended version of ID1 with input language tokens |
| ID3 (expanded) | 2,781 | ID2 expanded with new sentences obtained from seeds |
| *Corpora used to enrich the training of the Language Model* | | |
| IE | 129,130 | in-domain – sentences from customers evaluations |
| OD1 | +200,000 | out-of-domain – minutes from European Parliament |
| OD2 | +860,000 | out-of-domain – CETEMPúblico |

## 4.2 Corpus preparation

After the definition of tokens to be used on input language, is necessary to build the output language. To do that, about 20 people, Portuguese native speakers, with ages from 20 to 60 years old, produced sentences summarising the data presented by tokens. This task was made in two steps. First, we collected 135 sentences. Afterward, a second set of 230 sentences was collected.

With this sentences, several corpora were created to evaluate the system. They are presented in Table 4. First, we named corpus `ID1`, the one with seed sentences produced by humans (technically, we call to these sentences, *in-domain sentences*). This corpus is the *original corpus*, produced by humans. Second, corpus `ID2`, the *extended corpus*, was created, resulting from adding to ID1 all tokens' data used to describe the input language. Third, a corpus `ID3` was made by expansion of ID2. We used the same technique, as described by Langner [4, pag. 69]. This corpus is the *expanded corpus*.

To be used on the creation of *language models*, we defined three other corpora. As corpus `IE` (from, Information Extraction), we joined all the sentences produced by hotels' customers with their reviews. It has about 129 thousand sentences. These sentences belong to the same domain of the sentences produced by the system in evaluation. To test the impact of using *out-of-domain* (OD) sentences, on the language model, the corpus `OD1` was made with

Portuguese minutes of European Parliament[2]. This minutes are part of the Moses for Mere Mortals (MMM) system, a prototype which aims to help the production of a translation chain for real world documents. These minutes have 200 thousand sentences. The last corpus, `OD2`, corresponds to Portuguese sentences obtained from Público newspaper. These sentences belong to Linguateca project – CETEMPúblico [13]. `OD2` corpus has about 860 thousand sentences. Both corpora are, clearly, out of the domain of our system. The goal is to improve the richness of the produced sentences.

The process of collecting corpora by humans took approximately 2 weeks. After that, the corresponding expansion and preparation of corpora for building the language models took another week, leading to a 3 weeks work for all process.

# 5 Results

## 5.1 Examples of generated sentences

To give an idea of the generation capabilities of the developed system, Table 5 presents several representative examples of sentences produced. The table is divided in 4 parts.

The first part shows sentences produced by the system for two different input vectors with information from the ID3 corpus. The input vector is presented first, preceded by *in:*; *out:* identifies the generated sentence.

All sentences were produced with the system trained with ID3+IE+OD1+OD2 corpora (see Table 4). It is possible to see that the sentences have distinct quality. The first is considered 'good', since it is grammatically correct and transmits correctly all data expressed in the input vector. The second sentence can be considered 'acceptable' since it transmits the main idea of what is expressed by the input, but it is not grammatically correct nor uses all available data.

The second part of table (sentences 3 and 4) presents an alignment between sentences produced by humans and sentences produced by the system, for the same input. The sentences presented are short to be possible to show differences between human and system sentences. For each input data, when a sentence is produced by system, the generated sentence is usually different from those a human usually do. These differences can be the addition of new words, the suppression of words, or, even, by positioning words in different zones on the sentence. Here, we use the token "***" to highlight the addition or deletion of words. The use of upper cased words is used to represent words that are in different positions on the sentence. Their use on human produced sentences is only to make more visible the differences between the two sentences.

Examples 5 and 6, in part 3 of the table, present good quality sentences produced by the system, both in terms of intelligibility and naturalness. They are completely different from those that humans produced. Nevertheless, they express the same information as the corresponding human sentences. Last part of the table, part 4, presents two sets of sentences generated with different systems. Each set has the same *input* to make possible the comparison of generated sentences. For this set, systems were trained with corpus ID3, ID3+IE and ID3+IE+OD1+OD2. For the first set (example 7), all output sentences can be considered acceptable. They are very different, expressing the same information in quite different ways. The last sentence is the only one grammatically correct. In the last set, example 8, all systems generated the same output sentence.

[2] Available at `https://github.com/jladcr/Moses-for-Mere-Mortals`.

**Table 5** Examples of sentences produced by the developed system.

**Part 1 – Examples of input vectors and corresponding generated sentences**

| | |
|---|---|
| 1 | *in*: hotel-no-data service-dinner eval-positive all-people adj-good adj-no-data<br>*out*:o jantar foi bom para a esmagadora maioria das pessoas<br>*(the dinner was good for the majority of people)* |
| 2 | *in*: hotel-no-data service-customer-service eval-negative few-people adj-very-bad adj-no-data<br>*out*:atendimento péssimo serviço<br>*(customer service very bad service)* |

**Part 2 – Examples of produced sentences (S) aligned with the corresponding sentence made by humans (H).**

| | |
|---|---|
| 3 | *in*:hotel-no-data service-customer-service eval-positive no-data-people adj-pleasant adj-impeccable<br>H: NESTE HOTEL, o atendimento era agradável e BASTANTE impecável<br>S: *** *** o atendimento era agradável e *** impecável |
| 4 | *in*:hotel-seaside service-smell eval-neutral no-data-people adj-normal adj-no-data<br>H: NO SEASIDE, o cheiro é DESCRITO COMO normal<br>S: *** *** o cheiro DO SEASIDE é normal |

**Part 3 – Samples of generated sentences, very different from test sentence, but with quality. H for sentences produced by humans.**

| | |
|---|---|
| 5 | *in*:hotel-no-data service-noise eval-neutral all-people adj-acceptable adj-no-data<br>H: OUVIA-SE ALGUM BARULHO DE FORA MAS NÃO ERA MUITO INCOMODATIVO<br>S: A ESMAGADORA MAIORIA DAS PESSOAS DESCREVEM O BARULHO COMO ACEITÁVEL |
| 6 | *in*:hotel-no-data service-furniture eval-positive few-people adj-pretty adj-functional<br>H: POR ESTAR BONITO E SER FUNCIONAL, DIVERSOS CLIENTES AVALIARAM O MOBILIÁRIO POSITIVAMENTE<br>S: *** *** *** *** DIVERSOS HÓSPEDES GOSTARAM DO MOBILIÁRIO BONITO E FUNCIONAL |

**Part 4 – Sentences produced by different system variants for the same input.**

| | |
|---|---|
| 7 | *in*: hotel-no-data service-reception eval-positive half-people adj-charming adj-no-data<br>ID3: a receção cerca de metade dos clientes, como charmosa<br>ID3+IE: a receção , cerca de metade dos clientes charmosa<br>ID3+IE+OD1+OD2: a receção é charmosa |
| 8 | *in*: hotel-no-data service-parking eval-positive no-data-people adj-good adj-excellent<br>ID3: o estacionamento é bom e excelente<br>ID3+IE: o estacionamento é bom e excelente<br>ID3+IE+OD1+OD2: o estacionamento é bom e excelente |

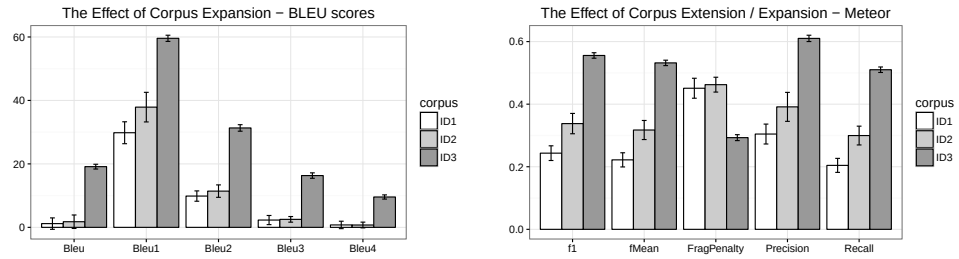



**Figure 3** The effect of corpus extension/expansion. BLEU (left) and Meteor (right) evaluation results with the 3 in-domain corpus (original, extended and expanded).

## 5.2 Evaluation Results

This section presents a formal evaluation made with automatic tools. Several versions of our system were created, using the corpora summarized in Table 4. Evaluation objective was to investigate the influence of the corpus expansion and different language models in the sentences produced, in order to arrive to the best combination.

The commonly used *10-fold cross-validation* technique was adopted, resulting in training 10 different versions (named from A to J) for each variant of the system. After training, all systems were tested with the corresponding test corpus. The BLEU [6] and Meteor [1] metrics were adopted to measure the accuracy of each system.

### 5.2.1 Effect of corpus expansion

It is well known that Moses works well with corpus of thousands/millions of sentences, but our initial corpus has only 135 sentences. The first questions that arise are: What performance can we have with such a small data set? Can results be improved by methods for extending/expanding? To find answers to these questions, the first evaluation addressed the effect of corpus expansion. Figure 3 presents BLEU and Meteor evaluation results for the 3 in-domain corpus: ID1 (original), ID2 (extended) and ID3 (expanded).

BLEU evaluates the strict correspondence between the evaluated corpus and the reference corpus. With this in mind, the result obtained by ID1 corpus, at BLEU global score is not a surprise. Also, as expected, the improvement to original corpus done with tokens' data (ID2 corpus) provided, globally, better results. The higher improvement was obtained at Bleu1. Effectively, at level of uni-gram the correctness of the provided sentences by ID2 corpus are better than sentences provided by ID1 corpus. This improvement starts to decrease at Bleu2, to a non statistically significant level, once the confidence intervals intersect each other.

The expansion of ID2 corpus with new sentences (ID3 corpus) allows better performance at all levels of BLEU evaluation. The improvement was quite high. BLEU scores corpus from 1 (total correspondence between evaluated corpus and reference corpus) to 0 (no correspondence). The scores obtained with ID3 corpus do not mean that sentences are, globally, not good. It only means that generated sentences are misaligned with reference corpus. And, to be misaligned is not synonymous of bad. As presented at middle section of Table 5, there are good sentences despite their difference from reference sentences.

For Meteor, it is evident that the best evaluation results from ID3 corpus, in all metrics. The F1 metric, which measures, at uni-gram level, the relation between *precision* and *recall*, for ID3 corpus has almost twice the value obtained by ID2 corpus. Almost the same result is obtained with $F_{mean}$ (which favours Recall) with this two corpora. The differences, with

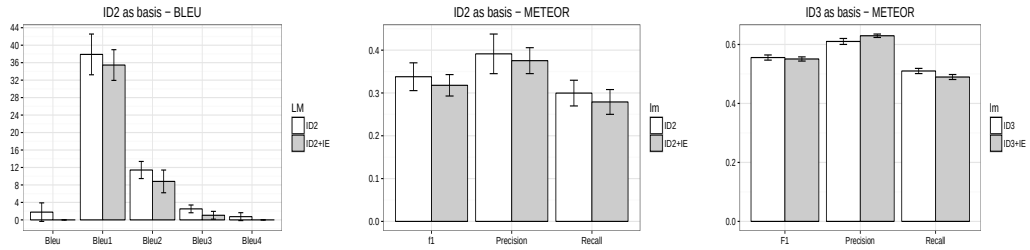



**Figure 4** Influence of different Language Models on BLEU and Meteor scores. The first two plots, at left and centre, refer to systems trained with ID2 corpus; plot at right refers for systems using ID3 corpus.

these two metrics, are also statistically significant with corpus ID2 and ID1, although not as large as the difference between ID3 and ID2 corpus.

Equal relation is observed at Precision and Recall metrics, for all corpora. The results for Fragmentation Penalty do not follow the previous tendency. As expected, ID3 corpus has a lower penalty because the corpus is richer. For ID1 and ID2 corpora, the difference is not statistically significant, although ID2 corpus has a slightly higher penalty. That is, probably, due the addition of tokens' data to ID2 corpus, which consist only in uni and bi-grams.

### 5.2.2 Effect of Language Model training

The second set of questions addressed in the evaluation was related to the effect of the language model, namely: How the corpus used to create the language model for the output language (Portuguese) affects the system performance?

The effect of language model was investigated for systems trained both with ID2 and ID3 corpus. For each, two variants of the language model were trained: one with only the ID corpus; other using ID and IE corpora. Figure 4 summarizes the results obtained. For a first model, only the output language part of ID2 corpus was used in the training process. For the second model, additional corpus IE was used.

Bleu and Meteor average scores obtained with language models trained with only ID2 or ID3 corpora are slightly better than when adding the IE corpus to the training set, but, in general, differences are not statistically significant.

The non-significant difference in scores, confirmed by manual inspection of generated sentences (see examples in Table 6), show as viable the use of both methods to create a language model.

### 5.2.3 Effect of out-of-domain corpora used in Language Model training

In the previous section, we presented the influence that a richer corpus could have on the production of language models. That enrichment was produced by the addition of a corpus with sentences in the same domain of the output corpus. As out-of-domain corpus are available, the effect of such additional training data for the creation of language models was also evaluated.

Two different corpora were used: (1) a set of Portuguese minutes of European Parliament – named as OD1 corpus; (2) a large set of Portuguese sentences from *Público* newspaper gathered by Linguateca project – named as OD2 corpus. As training corpus, we used ID3 corpus, as previous experiences revealed this corpus can achieve better results.

**Table 6** Output samples of systems based on ID2 corpus and using 2 different Language Models: one trained using only ID2 corpus; other trained with ID2 plus IE corpus.

| | |
|---|---|
| *input 1* | hotel-no-data service-prize eval-positive few-people adj-accessible adj-no-data |
| ID2 | os clientes do hotel a qualidade de serviço acessível |
| | *in English: The hotel clients service quality accessible* |
| ID2 + IE | a qualidade do hotel é acessível |
| | *in English: The hotel quality is accessible* |
| *input 2* | hotel-no-data service-wc eval-negative many-people adj-uncomfortable adj-no-data |
| ID2 | o wc a hotel é desconfortável |
| | *in English: the wc the hotel is uncomfortable* |
| ID2 + IE | o wc é muito desconfortável para o hotel |
| | *in English: the wc is very uncomfortable for the hotel* |

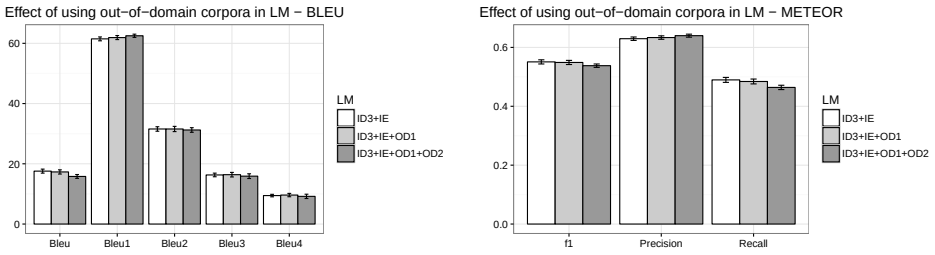



**Figure 5** Effect in BLEU and METEOR scores of using additional out-of-domain corpora in the train of the Language Model. As baseline was used the system with the LM trained with ID3 plus IE corpora.

Figure 5 presents the results obtained by systems with three different language models. The first language model was obtained by output corpus plus in-domain corpus (as explained in previous section). On the second language model, we added, to previous LM, the OD1 corpus. Lastly, on the third language model, we added the OD1 and OD2 corpus.

The differences between all metrics (Bleu, Bleu1 to Bleu4) are not statistically significant. Just, a slight improvement is observed at Bleu1 metric, for system ID3+IE+OD1+OD2. The Meteor evaluation reveals that there are no significant differences from what is expressed at BLEU evaluation. However, is possible to observe that the addition of a out-of-domain corpus to language model increases the precision, with consequent reduction of recall values, meaning that translated sentences, by ID3+IE+OD1+OD2 system, are closer to the reference sentences, than sentences translated with system ID3+IE, which has simpler language models. Nevertheless, the improvement achieved is quite small, if we consider the dimension of OD1+OD2 corpora. One possible reason is the level of specialisation of ID3 corpus. Other, is its dimension, where, possibly, there are not enough sentences to learn the language model to build a large diversity of sentences.

### 5.2.4 Comparison with similar systems

Previous sections presented the evaluation results obtained by different variations of our system. Results obtained, by automatic evaluations, are consistent. Better results were obtained by ID3 corpus with variations of language models. Figure 6 compares the results from our best systems, with Mountain's best systems [4, pag. 73–75] and, for the best of our knowledge, the most recent system developed for data-to-text in Portuguese – MEDICATION2PT [7].

In this comparison, we used the system obtained from ID3 corpus, with two variations of language model: language model made only with the output language, and language model

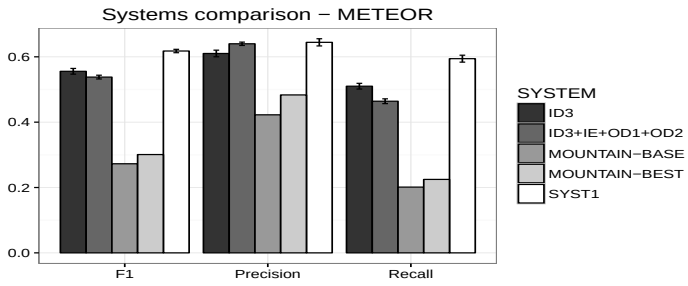



**Figure 6** Comparison of Meteor scores of the best systems presented in this paper versus MOUTAIN system and a recent system developed for Portuguese (MEDICATION2PT).

made with output language plus IE and OD1 and OD2 corpora. From Mountain, we choose the original system and the system with higher results. From MEDICATION2PT, we choose the phrase-based version. By simplicity, only Meteor evaluations are presented, since BLEU results are similar. In all metrics, our systems' results are better than MOUNTAINS systems. At $F_1$ and Recall, results are almost twice the results obtained by MOUNTAIN systems.

When comparing with MEDICATION2PT (SYST1 in the Figure 6), the new system presents similar Precision but lower Recall and, consequently, lower $F_1$. The lower Recall is not necessarily negative, as we also aim at having variability in the generated sentences.

## 6 Conclusions

Aiming at providing a natural form (written text in Portuguese) of transmission of the information extracted automatically from comments on Hotels to managers, this paper presents a Data-to-Text aimed at this specific domain. The system is based in phrase-based machine translation performed by MOSES trained with small corpora. It is capable of processing information vectors produced automatically by an IE system to create sentences.

Several variants of the system were created to investigate several factors: method of corpus expansion; corpora used in training language models for output language, using both in-domain and out-of-domain corpora.

Using the automatic metrics BLEU and Meteor, the system variants were evaluated and compared with previous results. Results show that developed system variants are capable of achieving good Precision and producing usable sentences. A factor that showed significant effect in performance increase was corpus expansion. Results were significantly better when the base corpus was *expanded*. Use of additional corpora in language model training did not reveal capable of similar improvements in evaluation scores. But, as automatic metrics do not give a final answer regarding quality of the generated sentences, an evaluation by humans is needed to investigate further this factor.

Future work should also include exploration of mechanisms to combine generated sentences to produce texts.

**Acknowledgements.** The authors thank the reviewers for their contribution.

## References

1 Michael Denkowski and Alon Lavie. Meteor 1.3: Automatic metric for reliable optimization and evaluation of machine translation systems. In *6th Workshop on Statistical Machine Translation – EMNLP*, pages 85–91, 2011.

2 Flávio Ferreira, Nuno Almeida, Ana Filipa Rosa, André Oliveira, José Casimiro Pereira, Samuel Silva, and António Teixeira. Elderly centered design for interaction – the case of the S4S medication assistant. *Procedia Computer Science*, 27(Dsai 2013):398–408, 2014.

3 Albert Gatt and Ehud Reiter. SimpleNLG: a realisation engine for practical applications. In *12th European Workshop on Natural Language Generation*, pages 90–93, 2009.

4 Brian Langner. *Data-driven Natural Language Generation: Making Machines Talk Like Humans Using Natural Corpora.* PhD thesis, Carnegie Mellon University, 2010.

5 Eder Miranda Novais, Rafael Lage Oliveira, Daniel Bastos Pereira, Thiago Dias Tadeu, and Ivandre Paraboni. A testbed for portuguese natural language generation. In *Brazilian Symposium in Information and Human Language Technology*, pages 154–157, 2009.

6 Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. BLEU: a method for automatic evaluation of machine translation. In *Annual Meeting of the Association for Computational Linguistics*, pages 311–318, 2002.

7 José Casimiro Pereira and António Teixeira. Geração de linguagem natural para conversão de dados em texto – aplicação a um assistente de medicação para o português. *Linguamática*, 7(1):3–21, 2015.

8 José Casimiro. Pereira, António Teixeira, and Joaquim Sousa Pinto. Towards a hybrid NLG system for Data2Text in Portuguese. In *Conferência Ibérica de Sistemas e Tecnologias de Informação (CISTI)*, pages 679–684, 2015.

9 François Portet, Ehud Reiter, Albert Gatt, Jim Hunter, Somayajulu Sripada, Yvonne Freer, and Cindy Sykes. Automatic generation of textual summaries from neonatal intensive care data. *Artifitial Intelligence*, 173(7-8):789–816, 2009.

10 Ehud Reiter. An architecture for data-to-text systems. In *Eleventh European Workshop on Natural Language Generation (ENLG)*, pages 97–104, 2007.

11 Ehud Reiter and Robert Dale. *Building Natural Language Generation Systems.* Cambridge University Press, 2000.

12 Mário Rodrigues and António Teixeira. *Advanced Information Extraction.* Springer, 2015.

13 Diana Santos and Paulo Rocha. Evaluating CETEMPúblico, a free resource for portuguese. In *Annual Meeting of the Association for Computational Linguistics*, pages 442–449, 2001.

14 Douglas Fernandes Pereira Silva Junior, Ivandré Paraboni, and Eder Miranda Novais. Um sistema de realização superficial para geração de textos em Português. *RITA – Revista de Informática Teórica e Aplicada*, 20(3):31–48, 2013.

15 Ross Turner, Somayajulu Sripada, Ehud Reiter, and Ian P. Davy. Generating spatio-temporal descriptions in pollen forecasts. In *Conference of the European Chapter of the Association for Computational Linguistics*, pages 163–166, 2006.

# Adapting Speech Recognition in Augmented Reality for Mobile Devices in Outdoor Environments*

**Rui Pascoal[1], Ricardo Ribeiro[2], Fernando Batista[3], and Ana de Almeida[4]**

1 **Instituto Universitário de Lisboa (ISCTE-IUL), Lisbon, Portugal**
2 **Instituto Universitário de Lisboa (ISCTE-IUL), Lisbon, Portugal; and INESC-ID Lisboa, Lisbon, Portugal**
3 **Instituto Universitário de Lisboa (ISCTE-IUL), Lisbon, Portugal; and INESC-ID Lisboa, Lisbon, Portugal**
4 **Instituto Universitário de Lisboa (ISCTE-IUL), Lisbon, Portugal; and CISUC Centro de Informática e Sistemas da Universidade de Coimbra, Coimbra, Portugal**

## Abstract

This paper describes the process of integrating automatic speech recognition (ASR) into a mobile application and explores the benefits and challenges of integrating speech with augmented reality (AR) in outdoor environments. The augmented reality allows end-users to interact with the information displayed and perform tasks, while increasing the user's perception about the real world by adding virtual information to it. Speech is the most natural way of communication: it allows hands-free interaction and may allow end-users to quickly and easily access a range of features available. Speech recognition technology is often available in most of the current mobile devices, but it often uses Internet to receive the corresponding transcript from remote servers, e.g., Google speech recognition. However, in some outdoor environments, Internet is not always available or may be offered at poor quality. We integrated an off-line automatic speech recognition module into an AR application for outdoor usage that does not require Internet. Currently, speech interaction is used within the application to access five different features, namely: to take a photo, shoot a film, communicate, messaging related tasks, and to request information, either geographic, biometric, or climatic. The application makes available solutions to manage and interact with the mobile device, offering good usability. We have compared the online and off-line speech recognition systems in order to assess their adequacy to the tasks. Both systems were tested under different conditions, commonly found in outdoor environments, such as: Internet access quality, presence of noise, and distractions.

**1998 ACM Subject Classification** I.2.7: Natural Language Processing

**Keywords and phrases** Speech Recognition, Natural Language Processing, Sphinx for Mobile Devices, Augmented Reality, Outdoor Environments

**Digital Object Identifier** 10.4230/OASIcs.SLATE.2017.21

* This work was supported by national funds through Fundação para a Ciência e Tecnologia (FCT) with reference UID/CEC/50021/2013.





## 1 Introduction

New technologies related with Machine Learning, Natural Language Processing (NLP) and Augmented Reality (AR) are currently arising. One of the world's leading information technology research and advisory company – Gartner, Inc. – predicts that, after a long period of technological development and refinement, the implementation of advanced Machine Learning technologies and conversational systems solutions for AR applications are achieving its peak.

Augmented Reality (AR) is an example of what Fred Brooks[1] calls "Amplification Intelligence": the use of the computer as a tool to perform human tasks in an easier way. Specifically, AR can be used to perform tasks more intuitively, and efficiently interact with the information displayed in a screen. Natural Language Processing (NLP) providing means to help with this interaction. As stated by Chowdhury [5], NLP is a research and application area that explores how computers can be used to understand and manipulate speech and natural language text to do useful things. Thus, wise interactions with AR technology are needed [10]. A richer end-user experience involves speech interaction rather than simply reading or writing at a screen. Because they include cameras, a smartphone or smart glass can run AR applications, and that led to accelerating the development of innovative AR [15] applications. Our goal is to address the end-users interaction needs for a quick augmented reality technology, especially for outdoor activities, such as walking, cycling, etc.

There are benefits of using automatic speech recognition (ASR) with AR technology in outdoor environments [15], but such environments involve a number of additional challenges. Speech recognition in an augmented reality interface contributes to the efficiency, intelligent information and communication use, good perception, and "common sense", as mentioned by Barry [3] and Ronald Azuma [2] in outdoor environments [1].

In a previous research we have developed an AR prototype and performed empirical tests with 12 end-users, revealing that speech provided a more quick interaction than gestual, where users were able to perform operations about 1 to 2 seconds faster in average [14]. Our current goal is to integrate an off-line speech recognition system in the AR mobile application. We have adopted the open source CMU Sphinx[2] system [13],[12]. The Sphinx-4 speech recognition system has been jointly developed by Carnegie Mellon University, Sun Microsystems Laboratories, and Mitsubishi Electric Research Laboratories (MERL) [12]. It does not need Internet access and it should also provide a socially acceptable interface, natural to interact with [1], overcoming the AR technology challenges [2].

The work performed in the scope of this paper provides access to five interface features for outdoor usage. It allows requesting geographical, biometric, and climatic information without communication overload [14]. Our solution is intended to provide a good usability while managing and interacting with the information on the AR application. We have followed available receipts on how to integrate Sphinx in the Android operating system. Such operating system is the starting point for our future plans on mobile augmented reality devices, also based on the android operating system, including Recon Jet™ glasses[3] or Epson Moverio BT-200™[4]. CMU sphinx speech recognition can also be equally integrated in other mobile Operating Systems, such as Windows or iOS. This module does not require Internet access, and provides means to address the noise observed in outdoor environments.

---

[1] Frederick Phillips Brooks, software engineer and computer, known by the project OS/360 operating system developed by IBM for the System/360 mainframe. He wrote a book of The Mythical Man-Month.
[2] `http://cmusphinx.sourceforge.net`
[3] `https://reconinstruments.com/products/jet/`
[4] `https://tech.moverio.epson.com/en/bt-200/`

The following questions arise when speech recognition is used in augmented reality technology in mobile operating systems and in outdoor environments.

- **Question 1: Is the automatic recognition system for mobile AR devices efficient enough in outdoor environments, containing phenomena, such as noise and distractions?** In fact, some problems may arise, such as the user remembering what are the possible requests, or the noisy conditions that may prevent the system from correctly recognizing the information provided.
- **Question 2: How is the performance achieved with Sphinx speech recognition instead of using a web-based speech recognition system?** This is a mandatory question, because in outdoor environments we may have Internet connection constraints and the default speech recognition system may not be effective to respond to the execution of hands-free features.

Other issues and difficulties arise when speech signal is corrupted by many sources, e.g., the wind is bad for performance of recognition system, because wind speed does interference with the sound input of the microphone. In addition the system has to cope with non-grammaticality of spoken communication and ambiguity of language [17], e.g., there are several ways of saying "i want to take a picture". The current challenge is an augmented reality system being able to interpret several ways to request operations by end-user speech.

This paper is structured as follows. Section 2 presents the Related Work. Section 3 describes our system architecture. Section 4 discusses the results and the associated difficulties. Finally, Section 5 presents the major conclusions and point out future research directions.

## 2 Related Work

AR is a new technology, but should not be categorized as mere technology. Instead, AR is an advanced computer interface, as mentioned by Alan Craig [6], which development started more than forty years ago. Still, there is a strong requirement to be adopted to people, being required usability of technology (as mentioned by Sawyer [16]), in various areas of society.

In previous work[5], the first author used a classification taxonomy of the different kind of environments that may exist abroad (as mentioned by Pascoal and Guerreiro [14]), and in these various environments performed tests with end-users, e.g., in the following four environments: (i) silent; (ii) silent with distraction; (iii) noisy with distraction; and, (iv) very noisy with distraction. That suggested environment taxonomy involves the factors noise and distractions. It helps to cluster results of end-user tests with a speech recognition prototype. The noise can be traffic, industry, animals, or wind speed, and so on. The distractions can be movement of people, animals, information overload, or forgetting system keywords.

By analyzing the various systems of speech recognition developed in recent years, anyone can identify that the software and hardware architecture adopted between them differs widely. However one difference is, e.g., Google speech recognition[6] needs Internet access, but CMU Sphinx does not need Internet access. That's why the developer will focus primarily on

[5] Chapter 12: Information Overload in Augmented Reality – The Outdoor Sports Environments, from book: Information and communication overload in digital age (2017). `www.igi-global.com`, as mentioned by Pascoal and Guerreiro [14] in Information and Communication Overload in the Digital Age (pp. 271–301).

[6] The Google speech recognition or Google Cloud Speech API, which enables developers to convert audio to text by applying powerful neural network models in an easy to use API, available at `https://cloud.google.com/speech/`.

Sphinx. However, the authors will show some differences between Sphinx and Google Speech. At the same time, they will perform tests with both tools (Sphinx vs Google speech), because the ultimate goal is to extract the best and the most suitable of these two systems.

Finally, the mission is to contribute to the implementation of a mobile AR system, which has the aim of being used outside, overcoming the constraints and limitations of current mobile AR applications[7]. Moreover, requirements faced by the application developers were identified, e.g., to overcome the technological and environmental limitations, because they are clearly interrelated. In addition to the human limitations in understanding due to information overload [4], restrictions also often relate to the limited capabilities of mobile devices, and the fact that AR equipment should be usable in a wide range of environmental conditions, as mentioned by Ronald Azuma [1, 2].

## 2.1 Potential Distractions in Outdoor Environments

What happens most often is that, in the case of an outdoor end-user like a cyclist when overloaded with distractions moves more slowly, and cannot keep in proper lane of the road correctly, but to compensate the risk of collision, experienced athletes psychologically maintain, or safeguard, for maintaining a greater distance of other cyclists and other obstacles there is ahead. It is a self-protection to reduce accidents. The conclusion is that attention is higher when using a voice interface, e.g., to send messages compared to text messages with hands. However, although attention is higher, the conduction is still impaired, therefore, there is a cognitive interference previewing messages, as mentioned by Sawyer [16].

If end-users are distracted with information overload on a smartphone or smart glasses or distracted with environment they cannot be remember of the keywords to interact with an AR application. This is what happened when Pascoal and Guerreiro [14] executed quantitative approach tests at twelve end-users using an AR prototype. They saw a task execution degradation (fifth task – “ok agent”) immediately after the another task (fourth task – “ok message”), i.e., six users have not complied with expectations (i.e., fifty per cent), but the previous task the execution was successful with everybody.

Now, on this work, the authors also evaluate with a quantitative approach, the performance of users executing only one task, but with two kind of automatic speech recognizers (Sphinx versus Google) and in distinct environments, some are noisy and with distractions and others are silent, but all running outside and with normal Internet access.

## 2.2 Human-Computer Interaction with Speech Recognition

The human-computer interaction must be a natural interface, meaning the interfaces of AR applications must be intuitive for users and easily controlled using the natural human movements. For hands-free, a good method could be with microphone interaction, by keywords, like “computer” or “photo”. The quantitative methods shown in the Discussion section are faster and usable for end-users.

The authors use a method with some kind of reverse word stemming, where all words with a common root are mapped from a single word (e.g., photo): all instances of photographing and beyond as “i would like a photo please” and so on, are mapped into “photo”, because “photo” is a single infinitive concept. The authors used this method because by experience on

[7] Other constrains are memory, storage capacity, battery autonomy and bandwidth on embedded devices are also very limited. For these reasons, has concentrated on simple tasks with restrictive grammars, as mentioned for David Huggins et al. [9].

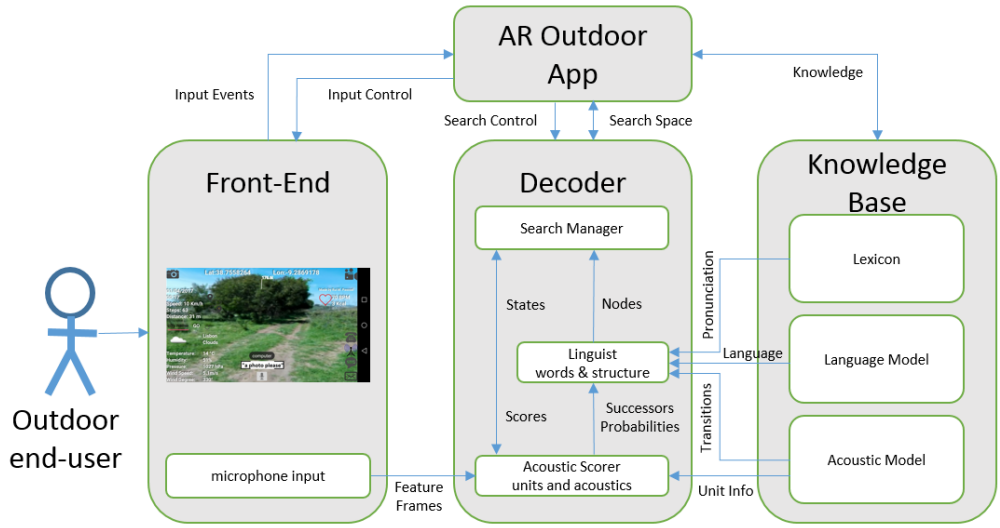



**Figure 1** Sphinx System Architecture [12, 13].

field, most end-users do not remember, or do not want to remember, rigid phrases or words: it is more cognitively easy and appropriate to have several ways of saying the same, and this is where probabilistic learning for natural language processing comes to help.

These kinds of human-computer interactions have been discussed in the International Symposium on Mixed and Augmented Reality (ISMAR), the leading international academic conference in the field of Augmented Reality and Mixed Reality. To create the best human-computer interaction, in other words, mobile human-device AR abroad with AR applications, that is, to give the user the ability to walk around large environments, outdoor is essential good guidance tracking abroad, as mentioned by Ronald Azuma [1] [2] and by Alan Craig [6]. Tangible AR interaction naturally leads to combining real object input with gesture and voice interaction, which often leads to multimodal interfaces, as shown at "A Review of Ten Years of ISMAR". They also discussed about AR technology interaction with speech commands, i.e., the survey work giving an overview of recent research in the field and conducts deploy of AR interactions with voice commands, as mentioned by Feng Zhou et al, from University of Canterbury, New Zealand [18].

## 3 System Architecture

Figure 1 shows the architecture of Sphinx and Figure 2 shows how are structured the possible interactions a end-user can have with the AR application using Sphinx as the automatic speech recognition module.

In what concerns the architecture of Sphinx (Figure 1), the speech signal is parameterized at the Front-End module, which communicates the derived features to the Decoder block. This block has three components: the search manager, the linguist, and the acoustic scorer. These work in tandem to perform the decoding. Inside of the Front-End there are several communicating blocks, each with an input and an output, linked to the output of its predecessor. When a block is ready for more data, it reads data from the predecessor and interprets it to find out if the incoming information is speech data or a control signal. A control signal might indicate the beginning or end of speech – important for the Decoder

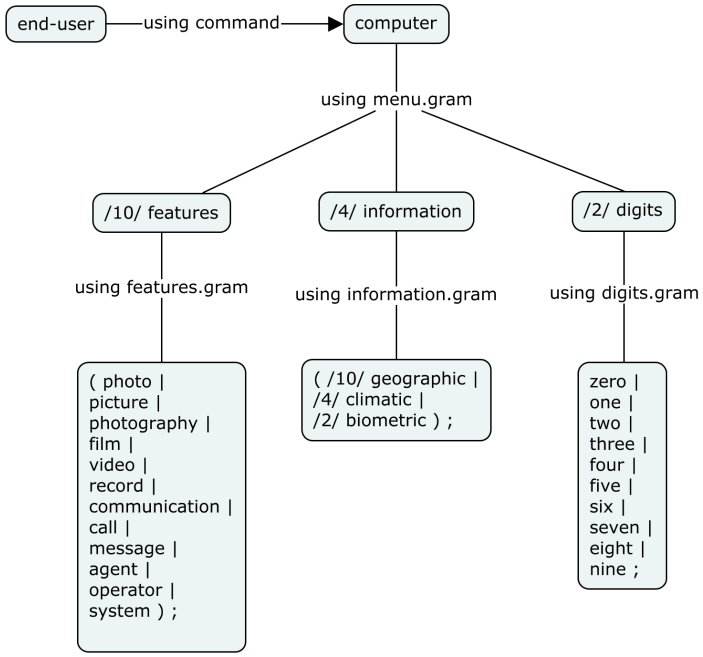



**Figure 2** Search tree of possibilities for the best hypothesis.

– or might indicate data dropped or some other problem. If the incoming data is speech, it is processed and the output is buffered, waiting for the successor block to request it. Additional blocks can also be introduced between any two blocks, to permit noise cancellation or compensation on the signal. Inside of the Decoder we have the Search Manager, the Linguist, and the Acoustic Scorer. The Search Manager has as primary function to construct and search a tree of possibilities for the best hypothesis. The construction of the search tree is done based on information obtained from the Linguist. In addition, the Search Manager communicates with the Acoustic Scorer to obtain acoustic scores for incoming data. The Linguist translates linguistic constraints provided to the system into an internal data structure called the grammar, which is usable by the Search Manager. The Acoustic Scorer has the task to compute the state output probability or density values for the various states, for any given input vector. It also provides these scores on demand to the search module. In order to compute these scores, the Scorer must communicate with the Front-End module to obtain the features for which the scores must be computed [12].

The authors observed and fit the AR application during interaction tests with five words and some equivalent sentences to the root word (e.g., “photo” = “a photo, please” or “photo” = “please, i would like a photo, thank you”), the difficulties to running the following solicited tasks: photo, film, communication, message, agent, biometric, climatic, and geographic.

Next, an abstraction through a frame conceptual map for speech recognition with these root words, i.e., keywords used by end-users when interact with the AR features.

The tree Figure 2, based on information obtained from the linguist, consists in all active paths in the search. The linguist translates linguistic constraints provided to the system into an internal data structure called the grammar. The numbers inserted in menu.gram mean weights of importance, e.g., the word “features” has a weight of 10, “digits” has a weight of 2 (worst execution priority), and so on. Figure 2 shows another detail, it’s the parenthesis in grammars of features and information. Only one word can be requested, and

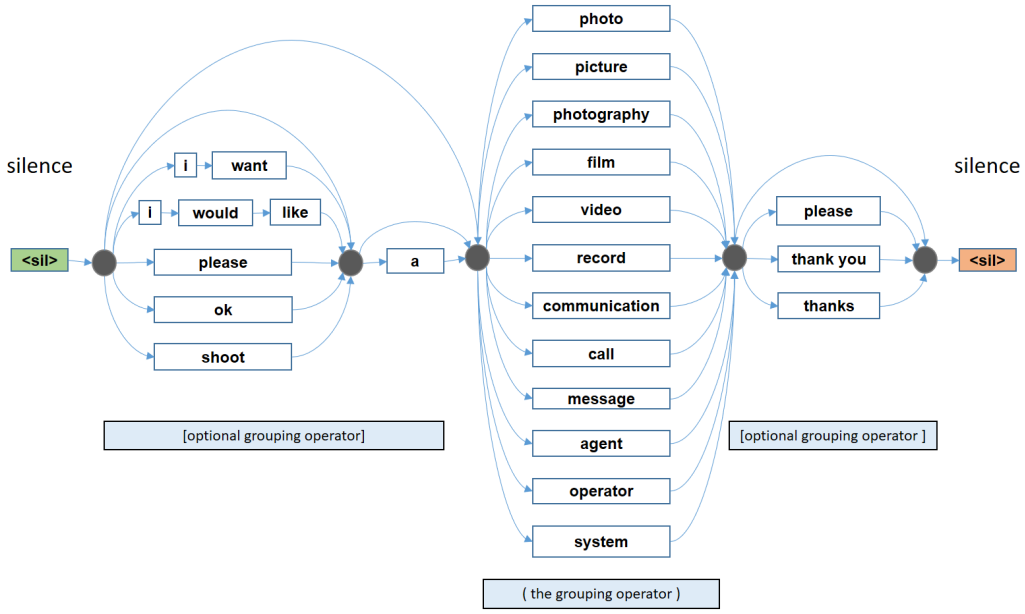



**Figure 3** Grammar graph to execute features (made by authors).

in information.gram has another detail it is the associated weight, i.e., "geographic" word is more likely to be chosen than "biometric" word because has only a weight of 2.

However, as previously mentioned, the authors will use a method with some kind of reverse word stemming, where all words with a common root are mapped from a single word (e.g., "photo"). And all instances of photographing as "a photo, please" and so on, are mapped to "photo", because "photo" is a single infinitive concept. Moreover, the end-user can also use other sentences if he/she likes or if he/she remembers. See Figure 3.

However, AR can be applied in every sense, not only visually. The usual researchers of AR fields, focused on mixing images and graphics real and virtual. However, *AR can be extended to include sound. Users can use headsets equipped with microphones*, as mentioned by Ronald Azuma [1, 2].

Next, we will see all instances of the word "photo" in a text. This is a process to check if string (the word spoken by the user) matches with defined grammar, as mentioned by L. Karttunen [11] and Walker et al. [17].

The grammar created by authors has simple words, like "photo", "film", or "biometrics", because the ability of human kind for long words is smaller than for short words. In general, the memory capacity for verbal contexts – digits, letters, words, and so on – strongly depends on the time it takes to speak aloud content and lexical function of the content, i.e., if the contents are known words or not. Several other factors also affect the measure of a person's memory and so it is difficult to establish the capacity of short-term memory by several chunks. That is why, in 2001, Nelson Cowan proposed that the activity of memory has a capacity of about four chunks in young adults (and lower in older children and adults).

Therefore, the authors specified grammars with key words and key sentences to execute associated methods (e.g., features grammar and information grammar). The recognition system must accept at least ten sentences or more, consisting of several words, which allow access to the five features (photo, film, communication, message, agent). To run this five features and programmatically speaking, authors suggest an if-else condition with a

**Listing 1** Defined JSGF grammar for features.

```
grammar features;

public <features> = <startCommand> <mainCommand> <endCommand> ;
<startCommand> = [i want|i would like|please|ok|shoot] [a];

<mainCommand> = (photo|picture|photography|film|video|record|
                 communication|call|message|
                 agent|operator|system) ;

<endCommand> = [please|thanks|thank you];
```

conditional "OR", e.g., to execute photo method. This is hard coded, but this could be avoided by simplify code, when using JSGF defined grammar[8]. See Listing 1 for the defined JSGF grammar for features. This grammar will simplify a wide range of equivalent sentences, which are, several possibilities of saying the same, in various ways, like "i would like a picture please", or "please a call thanks". See also Figure 3.

Another situation that the authors encountered when developing the augmented reality application and when unit tests were in execution was the running of undesired features without being ordered to run. This can be a serious problem during search, as mentioned by Paul Lamere et al. [13]. It is a pruning problem encountered by the search module in decoding parallel feature streams. The pruning is based on combined scores, paths with different contributions from the multiple feature streams get compared for pruning [13]. To break or reduce these pruning problems, the developed features on AR application are being combined in a weighted manner, with weights that can be more easily controlled, e.g., the "film" feature has low weight than the "photo" feature, and the "stop" word to exit of some features and return for "main activity" should have a little less. This is a specific algorithm for application learning. The result generated will provide by the search module is in the form of a tree, which can be queried for the best recognition hypotheses, or a set of hypotheses.

## 3.1 Information and Features Management Given to Outdoor End-User

Based on earlier subtopics and on the point of view of an outdoor end-user, the main question of this subsection, is how to get all relevant information with minimal effort and how to minimize dependency of communication with the Internet network without information overload [14]. Also, to get relevant features easily, in other words, is the Sphinx Android application efficient enough in outdoor environments, e.g., silent, silent with distractions, noise with distractions and very noise with distractions?

Next are described in detail the three groups "suggested" of informative data that could be submitted to the outdoor end-users:

1. **Climatic data:** involves temperature, atmospheric pressure, altitude, and relative humidity. This data serves, not only to inform, but also to intelligently calculate together with the user's health status data.
2. **Biometric data:** involves the heart rate and calorie expenditure. These are important data to calculate and provide vital advices. Without this sensor, it will not be possible to deliver alerts, which could make the difference.

[8] The rules of Java™ Speech API Grammar Format – can be found at `http://www.w3.org/TR/jsgf/`.

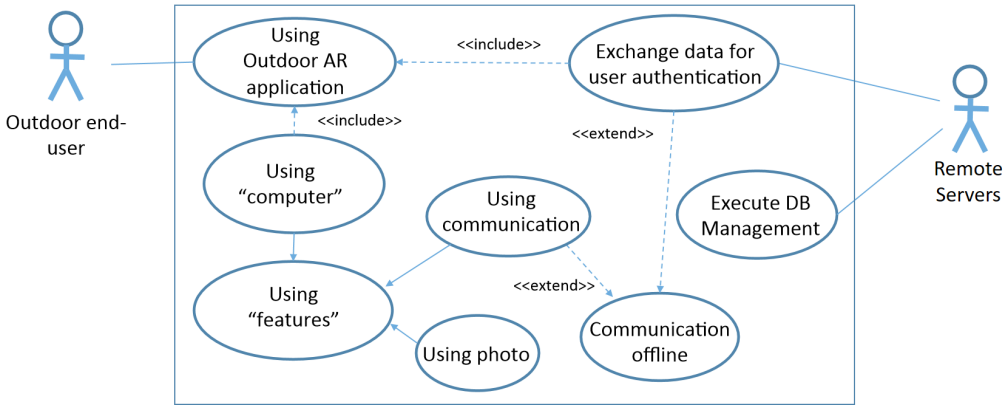



**Figure 4** UML use case diagram for outdoor end-user to take a photo with hands-free.

3. **Geographic data:** involves the global position system (GPS), compass, and stopwatch, also, involves speed, measured steps, and distances, initial and final positions. Geolocation serves not only to inform, but also to index the server database, events, and points of interest. Can be a good tracker of an user in motion and what their cadence or rhythm is.

To show end-users' interactions with an AR system, is presented the following particular Use Case Diagram (Figure 4). It is an abstraction of outdoor end-users when they take a photo with hands-free. Previously, users may be authenticated on the server[9].

## 4 Discussion

Spoken language processing is a diverse subject that relies on knowledge of many levels, including acoustics, phonology, phonetics, linguistics, semantics, pragmatics, and discourse. Difficulties arise when speech signal corrupted by many sources (environmental noises). There are observed issues and difficulties, arise when speech signal corrupted by many sources, e.g., the wind is bad for performance of recognition system, because wind speed does interference with the sound input of the microphone used by every users. Also another aspect, we must take into account a human cognitive limitation, e.g., when users are receiving information and several tasks to perform, can lead to information and communication overload [14] [4]. In addition the system has to cope with non-grammaticality of spoken communication and ambiguity of language. Also, this is a huge field, because of the diverse nature of spoken language processing requires knowledge in computer science, electrical engineering, mathematics, syntax, and psychology, as mentioned by Willie Walker et al. [17].

The main task that was requested to the end-users was the execution of the photo functionality, according to the grammatical probability of the request to run a photo method, e.g., "please i would like a photo thank you".

So, with a quantitative and qualitative treatment of research hypotheses are implemented the user's information, as next, an AR prototype that simulates an outdoor environment to collect data, through observation of interaction tests with fourteen end-users. Next will show reaction's results with noise and distractions for better usability. See next Figure 5.

[9] This authentication is a future development of the authors. This is a requirement to track a particular user, a help to him, but needs Internet connection.

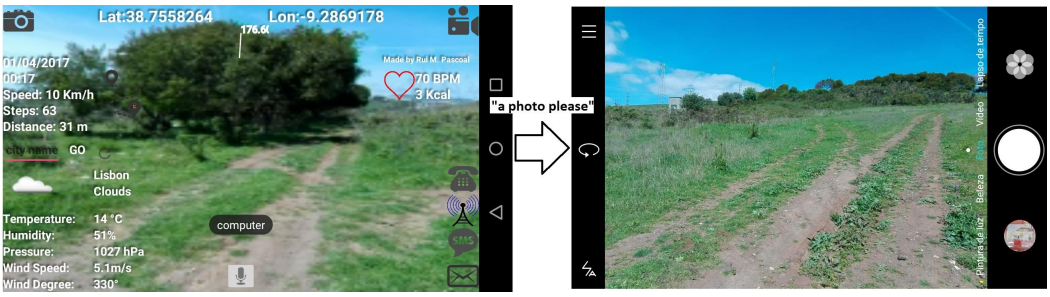



**Figure 5** Execution of photo feature in AR prototype with recognition system.

Tests up through a practical implementation output, the provision of the information given to end-users, wise information[10], in which case the use is preferably an Android smartphone, because it is portable and mobile [6].

This application had some difficulties when implementing Sphinx on android, e.g., there were difficulties in executing functionalities through speech recognition, to execute the associated methods (`startCameraFoto()`), it had to be through the following condition:

Previously the grammars had to be built, for the recognition of the words to be used in the interaction with the android application, as well as the static variables, with the key words of access to the three grammars:

- "features"
- "digits"
- "information"

The recommendations for researchers and future researchers with the influencing factor of outdoor environments are as follows. These recommendations are based on the difficulties experienced by the authors, as well as on the results of empirical field tests with some end-users.

The Sphinx Android prototype served to analyze real interactions in outdoor environments. This empirical research was conducted to obtain a quantitative approach. Afterwards, a questionnaire was applied to have a qualitative evaluation of end-users. Finally, reviews were collected by structured interviews.

The end-users were clustered in four groups, e.g., users 1, 2, 10 and 13 are in the silent environment group, as shown above on Table 1. Next, we will see time differences with interactions and differences with Sphinx speech recognition vs Google speech recognition. This is suitable, to perceive the correlation between variables, and take conclusions.

Google speech recognition system performs very well, but needs Internet connection and in some cases like with users 5, 7 and 8, had a delay. Also Sphinx had a difficult processing orders in environments with very noise and distractions, and there were some users who spoke too fast and too far from the microphone (user 3 and 7). Four users felt overloaded with informations and two of them are female, and did not remember what words to say. They repeated the test, later.

There are more things to take into account, e.g., the outdoor environments can be very wild and often have many restrictions, one of the restrictions are the difficulty of accessing the

[10] To adjust and filter the geographical, biometric and climatic information to provide users, is being developed research in parallel correlations between these three variables, the authors resorted to methods of statistical learning [8, 10].

**Table 1** Table with quantitative/qualitative results.

| User | Sex | Age | Environment | Quantitative Approach (in seconds) | | | | Qualitative Approach | |
|---|---|---|---|---|---|---|---|---|---|
| | | | | CMU Sphinx | | Google ASR | | | |
| | | | | word | sentence | word | sentence | Personal evaluation | Information overload |
| U1 | Male | 21 | silent | 0.5 | 0.5 | 0.5 | 0.5 | good | no |
| U2 | Female | 38 | | 0.5 | 0.5 | 0.5 | 0.5 | good | yes |
| U10 | Male | 39 | | 0.5 | 0.5 | 1.5 | 1.5 | good | no |
| U13 | Male | 22 | | 0.5 | 1.0 | 0.5 | 1.0 | good | no |
| U3 | Male | 14 | silent & distractions | 1.0 | 1.5 | 0.5 | 0.5 | good | yes |
| U4 | Male | 48 | | 0.5 | 1.0 | 0.5 | 1.0 | good | no |
| U9 | Male | 42 | | 0.5 | 1.0 | 1.0 | 1.0 | good | yes |
| U5 | Male | 22 | noise & distractions | 1.0 | 1.0 | 1.5 | 1.5 | good | no |
| U6 | Male | 36 | | 0.5 | 1.0 | 0.5 | 1.0 | no answer | no |
| U11 | Female | 38 | | 1.0 | 1.0 | 1.0 | 1.5 | good | no |
| U14 | Female | 37 | | 1.0 | 1.5 | 1.0 | 1.0 | bad | yes |
| U7 | Female | 37 | very noise & distractions | 1.0 | 1.5 | 1.0 | 1.0 | good | no |
| U8 | Female | 19 | | 1.0 | 1.5 | 1.0 | 1.0 | good | no |
| U12 | Male | 15 | | 1.0 | 1.0 | 1.0 | 1.0 | good | no |

Internet. In addition to this negative and critical factor, unfortunately the Google recognizer is also very ugly and covers most of the field of view of users (display on prototype). Figure 7 (left image) shows what happens when Google is running.

Figure 7 (right image) shows the end result of an AR prototype with an ideal recognition system. The authors focus on the field of augmented reality and the processing of natural language to create the best way for users to interact with these new technologies.

Also on the Google side and for this chapter discussion can be emphasize the interesting work of Javier Gonzalez-Dominguez et al., at [7], The researchers developed an end-to-end work with multi-language architecture, which was deployed at Google, that allows users to select arbitrary combinations of spoken languages (eight languages simultaneously). They leverage recent advances in language identification and a novel method of real-time language selection to achieve similar recognition accuracy and nearly-identical latency characteristics as a monolingual system.

In the case of the sphinx recognizer the grammar language is only developed in the English language, and in addition it is necessary to follow a sequence of keywords, e.g., “computer” + “features” to the final word or sentence used to associate with the respective method, e.g., for feature communication, “please, i would like to call you, thank you”.

## 5 Conclusions

This work identified issues and questions about the interaction with an AR application using two speech recognition modules (Sphinx and Google), experimenting different solutions in different conditions, such as with or without noise and distractions. Mobility with the aim of social acceptance, as well as usability, are the most important real benefits for real end-users.

The authors also presented some advantages of the Sphinx system over the Google system. The flexibility in the usage of various kinds of acoustic and language representations, as mentioned by Paul Lamere [12] and the independence of Internet access make Sphinx more advantageous and useful than Google’s speech recognizer. The Discussion section showed results of interactions with fourteen end-users and the advantages of the Sphinx system.

In the empirical tests, the end-users were clustered by environment conditions. Concerning the quantitative perspective, the time taken to respond to speech requests was affected by

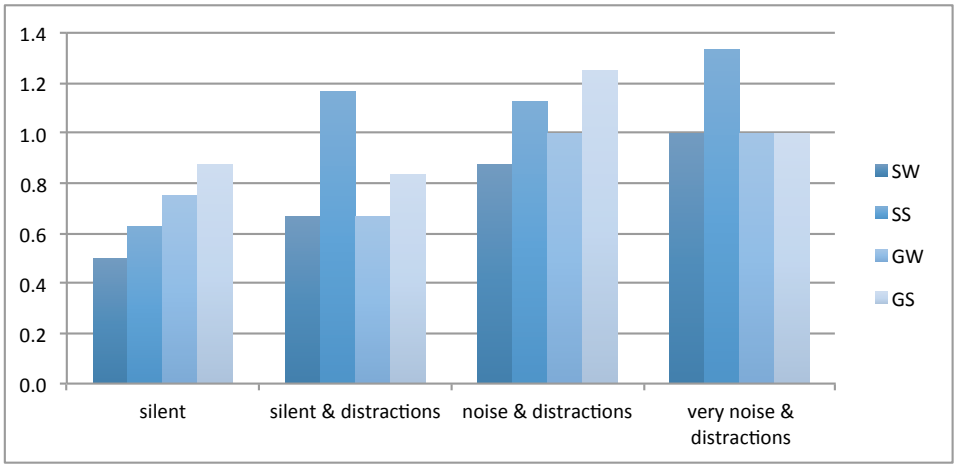



**Figure 6** Time taken (seconds) using the two ASR systems under different conditions: Sphinx using a single word (SW), Sphinx using a sentence (SS), Google using a single word (GW), Google using a sentence (GS).


**Figure 7** Google Speech recognition (left) and AR mockup with NLP – recognition system for outdoor environments (right).

recognition difficulties by both systems. Sphinx has shown difficulties in four cases, resulting in 1.5 seconds to generate the correct output for requests in the noisy with distractions context. Nevertheless, this system achieved good results in the quiet environment. Google speech recognition system performs well, but needs an Internet connection and in some cases was slower than Sphinx. Feedback from the users suggests that sometimes they spoke too fast or too far from the microphone.

Concerning the qualitative perspective of the evaluation, almost all users considered the interface good. The only exception was provided by a female user in the noisy environment with distractions context that considered that there was an information overload. Additionally, four other users in different contexts also felt that there was an information overload.

The authors propose the simultaneous adoption, deploy, and use of these two automatic speech recognition systems (Sphinx and Google) in the AR application. That is, when using features that do not require an Internet connection, the Sphinx recognizer can be used (e.g., to take a photo, film, and agent AR operational functions), and when access to the Internet is available, Google recognition may be used (e.g., making phone calls and sending messages, because these features require mandatory access to telecommunications infrastructures). In fact, as Google's recognizer was relatively faster in noisier environments than Sphinx recognizer, it can be considered more adequate for communications-based tasks, such as dialing or composing messages.

In societal terms, one important contribution is the adaptation of this kind of technologies, an AR environment with Speech-based interaction, for use in outdoor environments, which can be of great importance, for instance, for tourists (e.g., touristic itineraries with cultural information with landmarks, relevant cultural sites, and historical places).

## 6 Future Research Directions

The future of interactions with technology will be constantly progressing. New technologies such as NLP and AR in the everyday life of people will be more and more present. The authors and researchers found that the tendency imposed passes through the portability, mobility and simplicity. The use of information systems is transversal of every area of society, and will be increasingly present in the use of this advanced interface. The research done by the authors concerning the paradigms of interaction, noted that it is better to have a fast and practical interaction, to get the best possible benefit of these advanced technologies.

To other grammar possibilities as particular information needed to end-users like if an user requests specific information, e.g., "computer", "informations", "climatics", "tell me, can I play athletics?". Then computer will replay "no", if outlook = sunny and humidity = high. Or, if outlook = normal and windy = false, computer will replay "yes". These are particular cases of given attributes and arbitrary attributes (classification and association rules, respectively [8]). The authors are committed to, and middle the AR development of a capable system to intelligently answer these specific questions solicited by end-users. Also, the authors would like to develop a user login layout before entering in AR and NLP application. This is a requirement to track a particular end-user, but it needs Internet connection as well as to help the GPS tracking precision.

## References


1 Ronald T. Azuma. The challenge of making augmented reality work outdoors. In Yuichi Ohta and Hideyuki Tamura, editors, *Mixed Reality: Merging Real and Virtual Worlds*, pages 379–390. Springer-Verlag, 1999.

2 Ronald T. Azuma. The most important challenge facing augmented reality. *Presence*, 25(3):234–238, 2016.

3 Ann Marie Barry. *Visual intelligence: Perception, Image, and Manipulation in Visual Communication*. SUNY Press, 1997.

4 David Bawden and Lyn Robinson. The dark side of information: Overload, anxiety and other paradoxes and pathologies. *Journal of Information Science*, 35(2):180–191, 2009.

5 Dimitris Christodoulakis, editor. *Second International Conference on Natural Language Processing*. Springer, 2000.

6 Alan B. Craig. *Understanding Augmented Reality: Concepts and Applications*. Morgan Kaufmann, 2013.

7 Javier Gonzalez-Dominguez, David Eustis, Ignacio Lopez-Moreno, Andrew W. Senior, Françoise Beaufays, and Pedro J. Moreno. A real-time end-to-end multilingual speech recognition architecture. *Journal of Selected Topics in Signal Processing*, 9(4):749–759, 2015.

8 Trevor Hastie, Robert Tibshirani, and Jerome H. Friedman. *The elements of statistical Learning: Data Mining, Inference, and Prediction*. Springer, 2nd edition, 2009.

9 David Huggins-Daines, Mohit Kumar, Arthur Chan, Alan W. Black, Mosur Ravishankar, and Alexander I. Rudnicky. Pocketsphinx: A free, real-time continuous speech recognition system for hand-held devices. In *International Conference on Acoustics Speech and Signal Processing (ICASSP)*, pages 185–188, 2006.

**10** Edward C. Kaiser, Alex Olwal, David McGee, Hrvoje Benko, Andrea Corradini, Xiaoguang Li, Philip R. Cohen, and Steven Feiner. Mutual disambiguation of 3D multimodal interaction in augmented and virtual reality. In *5th International Conference on Multimodal Interfaces (ICMI)*, pages 12–19, 2003.

**11** L. Karttunen, Jean-Pierre Chanod, Gregory Grefenstette, and Anne Schille. Regular expressions for language engineering. *Natural Language Engineering*, 2(4):305–328, 1996.

**12** Paul Lamere, Philip Kwok, Evandro Gouvea, Bhiksha Raj, Rita Singh, William Walker, Manfred Warmuth, and Peter Wolf. The CMU Sphinx-4 speech recognition system. In *International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 2–5, 2003.

**13** Paul Lamere, Philip Kwok, William Walker, Evandro B. Gouvêa, Rita Singh, Bhiksha Raj, and Peter Wolf. Design of the CMU Sphinx-4 decoder. In *8th European Conference on Speech Communication and Technology (EUROSPEECH)*, 2003.

**14** Rui Miguel Pascoal and Sérgio Luís Guerreiro. Information overload in augmented reality: The outdoor sports environments. In *Information and Communication Overload in the Digital Age*, pages 271–301. IGI Global, 2017.

**15** Heather F. Ross and Tina Harrison. Augmented reality apparel: An appraisal of consumer knowledge, attitude and behavioral intentions. In *49th Hawaii International Conference on System Sciences (HICSS)*, pages 3919–3927, 2016.

**16** Ben D. Sawyer, Victor S. Finomore, Andrés A. Calvo, and Peter A. Hancock. Google Glass: A driver distraction cause or cure? *Human Factors*, 56(7):1307–1321, 2014.

**17** Willie Walker, Paul Lamere, Philip Kwok, Bhiksha Raj, Rita Singh, Evandro Gouvea, Peter Wolf, and Joe Woelfel. Sphinx-4: A flexible open source framework for speech recognition, 2004. Sun Microsystems, Inc.

**18** Feng Zhou, Henry Been-Lirn Duh, and Mark Billinghurst. Trends in augmented reality tracking, interaction and display: A review of ten years of ISMAR. In *7th International Symposium on Mixed and Augmented Reality (ISMAR)*, pages 193–202, 2008.

# Vocatives in Portuguese: Identification and Processing*

**Jorge Baptista[1] and Nuno Mamede[2]**

1 **University of Algarve/FCHS & INESC-ID Lisboa/L²F, Faro, Portugal**
jbaptis@ualg.pt
2 **University of Lisboa/IST & INESC-ID Lisboa/L²F, Lisbon, Portugal**
Nuno.Mamede@l2f.inesc-id.pt

**Abstract**

This paper describes the most salient linguistic aspects of vocative constructions in Portuguese, with special reference to its European variety. Next, the paper presents the strategy followed for implementing this linguistic knowledge in a computational grammar of Portuguese, developed for the natural language processing chain STRING and using the XIP rule-based parser. Very precise and detailed linguistic descriptions can be implemented in this way.

**1998 ACM Subject Classification** I.2.7 Natural Language Processing/Text analysis

**Keywords and phrases** Natural Language Processing, Text analysis, Portuguese, Vocative, Parsing

**Digital Object Identifier** 10.4230/OASIcs.SLATE.2017.22

## 1 Introduction

This paper deals with vocative constructions in Portuguese. This is the case of the initial phrases in the sentence (facultative elements in brackets): *(Ó) (meu caro) João/amigo, não faças isso!* '(Hey) (my dear) John/friend, don't do that!´ In these examples, those phrases are traditionally analysed as having the syntactic function of *vocative*, that is, phrases that are somewhat marginal to the main sentence, and that are used by the speaker to address his interlocutor. These phrases are *not* the subject of the sentence main verb, which is a dropped $2^{nd}$-person-singular pronoun *tu* 'you'. The different forms that this interpellation can take are related to different sociocultural values, which may reflect, for example, in verbal inflection, for example, in the opposition between the form of treatment by *tu* 'you_$2^{nd}$-sg.' and *você* 'you_$3^{rd}$-sg.'), as in: *(Ó) Dr. João, não faça isso!* '(Hey) Dr. John, dont do that!´ However, the vocative construction may also serve to express other pragmatic values such as the expression of *affection*: *Minha coisinha fofa, não faças isso!* 'My little fluffy thing, don't do that!´ or an *insult*: *Minha grandecíssima besta, não faças isso!* 'My most-great beast, do not do this!´

In some communicative situations and certain textual genres, the use of vocative is relatively frequent, such as in dialogues, in the (formulaic) opening of epistolary texts, or at the onset of formal addresses, speeches and lectures. In the later, the speaker or the lecturer usually addresses the official entities present at the venue, using a strict protocol in their ordering and in their designation. Such lists can be quite extensive and usually

* This work was partially supported by national funds through Fundação para a Ciência e a Tecnologia (FCT) with reference UID/CEC/50021/2013.





end with a generic vocative, using the formula *minhas senhoras e meus senhores* 'ladies and gentlemen'. Because of the cultural weight and the formulaic nature of such vocatives, these expressions are highly idiomatic and cannot be translated literally (v.g. **my ladies and my gentlemen*). In the previous example, the nouns in the formula are determined by a (facultative) possessive pronoun.

The communicational and institutional settings of the address or speech determine the use of specific vocative formulae or special combinations of vocatives. For example, in the Portuguese Parliament, the speaker will first address the president (chair) and then the members of the Parliament, v.g., *Senhor presidente, senhores deputados* 'Mister president, (ladies and gentlement)'. The host of a circus show will address the audience by *respeitável público* (lit: 'respectable public/audience') and will also include the children in the vocative *senhoras e senhores, meninos e meninas* 'ladies and gentlemen, boys and girls'. In a public address (PA) announcement on a supermarket, the speaker will address the clients by *estimados clientes* 'esteemed clients'. Other factors, such as language variety, also determine the choice of the adequate formulae. In Brazilian Portuguese, an epistolary vocative would naturally choose the adjetive *prezado* 'esteemed' rather than the European Portuguese *caro* 'idem' or 'dear'.

From the point of view of automatic syntactic analysis (or parsing), the precise identification and adequate linguistic analysis of this type of expressions is relevant, since: (i) they should not be analysed as fundamental constituents of the sentence in which they occur; (ii) they can serve as antecedents of anaphors in the subsequent text; (iii) they can help to determine the structure of a dialogue, namely the turn-taking or an exchange of roles between the dialogue participants, and (iv) they express different pragmatic values, as we have already seen, and their formulaic and often idiomatic nature can make them very hard to translate automatically.

This paper describe one of the modules of the Portuguese computational grammar developed for the STRING system [11] was implemented and evaluated in order to adequately process vocatives in unbound texts. The immediate motivation for this study was the need to process transcripts of public speeches and addresses delivered by various official entities in various formal contexts (the national parliament, municipal councils, etc.). This module aims, therefore, at the identification, delimitation and automatic syntactic analysis of constructions of vocative in Portuguese, with special reference to the European variant. The module is integrated into the rule-based computational grammar developed for the XIP (Xerox Incremental Parser) [1]).

This article is organized as follows: The following Section 2 presents the theoretical framework, trying to exemplify and discuss different situations, to determine a place for the vocative in the Portuguese grammar. Secondly, some parsing issues resulting from an inadequate analysis of the vocative constructs are identified and illustrated with examples taken from different syntactic parsers of Portuguese (Section 3). At the same time, a brief survey will be made into syntax dependency coding schemes developed for various languages, identifying the way the vocative is typically framed. The next section (4) presents the solution developed in the framework of a Portuguese grammar, using the declarative rules of the XIP formalism. A preliminary evaluation of this module performance is provided (Section 5). The paper concludes with some perspectives for the development of current work and future applications.

## 2 Theoretical framework

This section tries to determine the place of the vocative constructions in Portuguese grammar, by way of reviewing some reference work. Naturally, it is outside the scope of this paper to do a systematic survey of the phenomenon, as it seeks only to raise its most salient aspects and to discuss, if only briefly, the controversial topics that this grammatical category raises.

Traditionally, *vocative*[1] is one of the major cases of nominal declension (inflection) of several natural languages, along with other casual values (v.g. nominative, accusative, genitive, etc.) in which nouns (and other categories) can inflect. The *case* is thus seen as a morphological variation identified with the syntactic function that the affixed element performs in the sentence: *nominative* for the *subject* function, *accusative* for the function of *complement*, etc. Perhaps for this reason, the vocative has usually been integrated into the set of main syntactic functions, alongside the subject or the direct complement [7, p. 160-161]. Portuguese, however, has dropped the Latin case system (except for personal pronouns), and has no morphologic marker corresponding to the vocative case, which make identification of vocatives more difficult. For some authors [4], no particular syntactic function or dependency is defined corresponding to the vocative, though mention to such situations is unavoidable when dealing with (direct) imperatives (*idem*:p. 457-458), e.g. *Tu/Maria, empresta-me esse livro! Você/O senhor, arrume o carro* (examples taken from the authors). In this framework, a noun phrase designating the subject of imperatives is given no particular status, and it is only said "to be interpreted as a vocative and, consequently, occurs in a peripheral position in the sentence" (our translation), irrespective of its left, pre-verbal or right position in the sentence (no mention is made about the mobility of vocatives *within* the sentence).

On the same line, the authors of the more recent *Gramática do Português* [14, vol.1, p. 351 ff.] also do not mention any vocative syntactic function, but prefer to treat the matter within the description of proper nouns [14, vol.1, p. 1013 ff.]. The authors of this chapter consider the vocative as a "semantic-discursive function", though no definition of the concept seems to be provided, nor its articulation with the remainder of the grammar architecture. Still, from their (short) descriptive approach, the authors mention, among other aspects, the particular behaviour that proper nouns can present in various functions, namely the interdiction of definite article in vocatives (e.g. **(Ó) **o** João, não faças isso* '(Hey) *the* John, don't do that'). Finally, from their description, it is possible to infer that, in the authors' framework, (i) vocative should not be included in the set of syntactic relations, which hold between the constituents of the sentence/clause; and (ii) that the grammatical role/value of vocatives should be placed at the level of discourse analysis, in the broader context of language communicative functions.

On another perspective [9, p. 351 ff.][10, p. 135 ff.], though not explicitly addressed, vocatives find a natural framework within the description of *performatives* (like *say* and *ask*/*order*), which are consider as operators underlying direct discourse, namely declarative and interrogative/imperative sentences, respectively. Based on evidence of a zeroed subject in direct imperative sentences (e.g. [*You*] *wash yourself!*, where the reflex *-self* result from the repetition of the subject *you* as complement of the verb *wash*), an underlying performative is considered in these sentences (e.g. *I ask/order you that you wash yourself*), which, when zeroed, yields the interrogative sentence-type intonation .

A similar approach could be adopted for the vocative-appellative (in fact, it is even hinted at by [10, p. 139], about sentences like *You (there)!*). In this perspective, the vocative

[1] Sometimes also referred to by terms such as *appellative* or *conative*, though not necessarily in this perspective. For a general overview, refer to [2].

would constitute another case of performative operators (such as *chamar* '*address*' or '*call*'), having the speaker as its subject and the addressee as its complement, in much the same way as the other performatives. (The so-called interjection *ó* 'hey' could be treated as a specialised – but facultative – marker of the vocative in direct speech, in much the same way as an argument-marker preposition.) In this sense, utterances involving vocatives should be analysed as complex sentences, consisting of, at least, two performatives: the vocative proper, with its appellative discursive function (addressing the hearer/reader); and the second sentence (a statement, an order/request, etc.).

This analysis sorts out the problem of having an expression in a sentence that is not necessarily linked to the constituents of the remaining content of that sentence, though it can be the antecedent of anaphoric expressions (as noticed by [7, p. 161]). It also addresses the issue of vocatives showing a peripheral status, and their corresponding mobility within the sentence. Being the result of a performative (and its reduction), the vocative would not hinge (or depend) on any other sentence constituent, but rather it would be linked to the topmost sentence node in the parsing tree, in much the same way as sentence-modifying adverbs [12].

Vocatives, however, are also used to express other (pragmatic) meanings in addition to the barest appellative function, namely to produce an affective value (insult/politeness), in which the choice of the appellative (a proper name, a pronoun, a common noun invested with affective value, including profanity words) plays a major role, particularly, among other distinctions, the *tu*/*você* opposition. For lack of space, we do not pursue further in this paper the linguistic description of vocatives and the theoretical implications of the approach outlined above. These involve, for instance, the adequate analysis of so-called interjections such *ó* and *pá*, dialectal variation in the use of modifiers (*caro*/*prezado*/*estimado* 'dear'), profanity words and adjectival predication, mobility within sentence, use of politeness adverbs (e.g. *por favor*), etc.

For comparison, vocatives have been encoded in the annotated AnCora [16] *corpora* of Spanish and Catalan texts (approx. 500K words each), though its frequency is very low (13 and 7 instances, respectively). In the documentation of the *Floresta Sintá(c)tica* treebank[2], regarding the revised portion (*Bosque*), only 29 instances of vocative have been reported.

One could also refer to the proposals of de Nivre *et al.* [13][3] for a set of so-called "universal dependencies" having in view their implementation in the Stanford parser. Their proposal concerning vocatives points to a similar solution as far as the vocative dependency is concerned. However, though the definition of the vocative dependency in [13] seems to be clear enough:

> The vocative relation is used to mark dialogue participant addressed in text (common in emails and newsgroup postings). The relation links the addressee's name to its host sentence. A vocative commonly co-occurs with a null subject, [...]. If the nominal is clearly vocative in intent, the preference is to use the vocative relation.

It is not clear that the vocative element is linked to the host sentence as a whole, probably because this parser's formalism does not make use of a TOP/ROOT node, and the place of insertion of the vocative seems to vary depending of the element that fills the topmost slot in the parse tree (see Fig. 1).

[2] http://www.linguateca.pt/Floresta/BibliaFlorestal/anexo4.html
[3] http://universaldependencies.org/

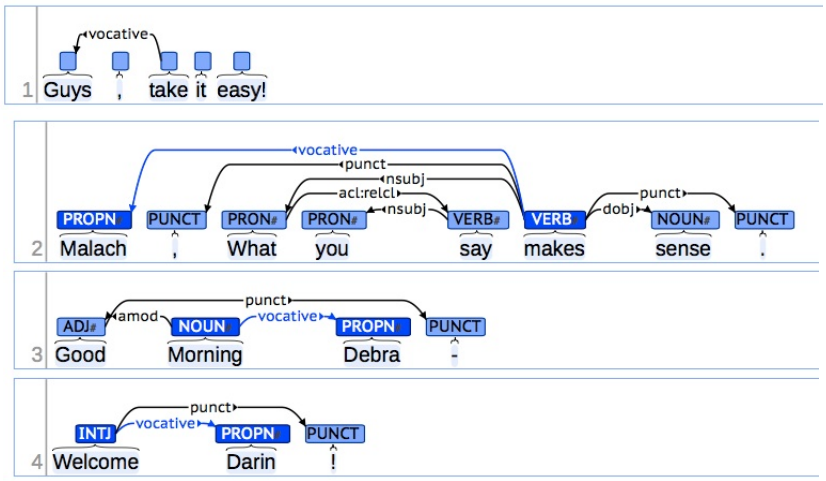



**Figure 1** Examples of the *vocative* dependency in the proposal of a set of universal dependencies for the Stanford parser.

Thus, in the first example (*Guys, take it easy!*) the vocative hinges on the main verb (in the imperative); in the second sentence (*Malach, What you say makes sense*), the vocative also depends on the main verb; however, in the third example (*Good Morning, Debra* ), as the parser analyzes the interjective *good morning* as an ordinary noun phrase, the vocative is made to depend on the noun *morning*; for the fourth sentence (*Welcome Darin!*), where *Welcome* is POS-tagged as an interjection, the proper name is made to depend on the only remaining element of the sentence. It should also be noted that, in spite of the definition of vocative dependency explicitly mentioning "the addressee's name", the first example involves a common noun. Thus, preference for vocative is determined for other types of noun phrases, as long as "clearly vocative in intent". Unfortunately, we were not able to find any evaluation of the Stanford parser concerning this vocative dependency, nor, to the best of our knowledge, how this proposal within a set of "universal dependencies" has been tackled for other languages.

In the next section, some parsing issues related to the processing of vocatives will be addressed.

## 3 Parsing issues

In this section, we look into the way that publicly available demo versions of Portuguese parsing systems treat a sample of clear-cut cases of the use of vocative. For sake of brevity, only a short commentary and not a detailed analysis can be made here. The performance of two systems was considered: (i) the LX-Suite [6], which uses the MaltParser[4]; and (ii) the Visl (Visual Interactive Syntax Learning) system[5], based on the parser *Palavras* [3]. A sample of sentences was used, exploring same variation factors, namely, the presence/absence of the interjective *ó* 'hey' in front of a proper noun, at the beginning of an imperative sentence and separated from it by comma, as well as the facultative use of affective modifiers *meu caro* 'my dear', v.g. (*Ó*) (*meu caro*) *Rui, não faças isso!* 'Hey my dear Rui, don't do that'.

[4] http://www.maltparser.org/parser
[5] http://visl.sdu.dk/visl

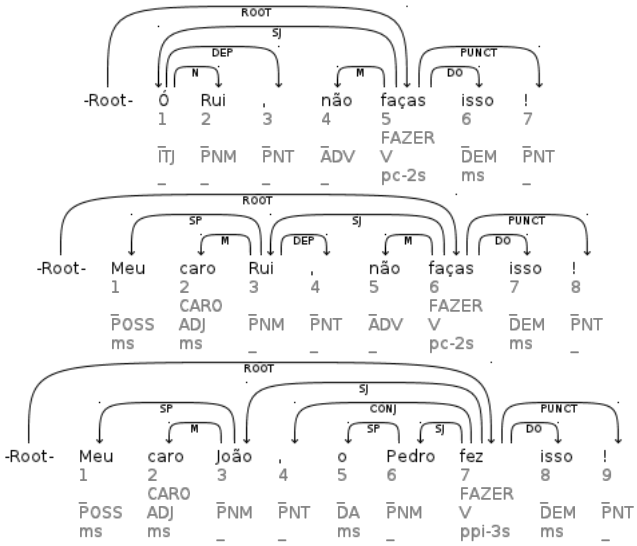



**Figure 2** Sentences parsed by the Lx-Suite using the MaltParser; top: imperative sentence with *ó*; center: imperative sentence without *ó*, but with a possessive and an adjective *meu caro* modifying the noun; bottom: a declarative/exclamative with the same type of vocative as above.

As far as we could ascertain from its documentation[6], the Lx-Suite parser does not seem to include a specific dependency relation in order to capture vocatives. Therefore, the remarks below aim only at highlighting parsing issues the lack of this dependency entails. As it can be seen, in Fig. 2 (top), the proper noun *Rui* is parsed as a sort of complement of the interjection (`N` dependence: nominal modifier?, apposition?), which, in turn, is analysed as the subject (`Sj`) of the imperative. This is arguably not an adequate parse. In the second sentence (center), *Rui* is parsed as the subject of the main verb. Since these are imperative sentences, and in the absence of a vocative dependency, this formal description may not be entirely inadequate, as the vocative and the (zeroed) subject of the verb in the imperative are necessarily co-referent. However, this co-reference is not obligatory in other sentence types, and, in the absence of a vocative dependency, systems may produce inadequate analyses in such cases. Thus, when parsing a similar, but declarative (or exclamative) sentence, with a vocative non co-referent to the main verb subject (below), the results are less then adequate. In this sentence, the vocative *João* is incorrectly parsed as subject of the main verb, in much the same way as the correct subject *Pedro*.

In its turn, the Visl system parser integrates the vocative dependency (`@VOK`), as shown in Fig. 3. Still, it seems that the vocative is triggered only in the presence of the interjection *ó* (left), which is linked directly to the topmost `root` note of the parse tree (by an `@ADVL` dependency) and apparently unrelated to the noun phrase *meu caro João* 'my dear João'. Without this lexical cue (right), no vocative is extracted, and the noun phrase is linked to the verb by a `@TOP` dependency. According to its definition[7], this `@TOP` dependency seems to have been construed for another type of relation, namely, instances of topicalization, such as those provided in the examples of the symbol set manual, v.g. *A Maria, não quero convidá-la* 'Maria, [I] don't want to invite her' (object topic); *Esse rapaz, ele sabe dançar* 'That boy, he

[6] `http://nlxserv.di.fc.ul.pt/depparser/intro_en.html`
[7] `http://visl.sdu.dk/visl/pt/info/symbolset-manual.html`

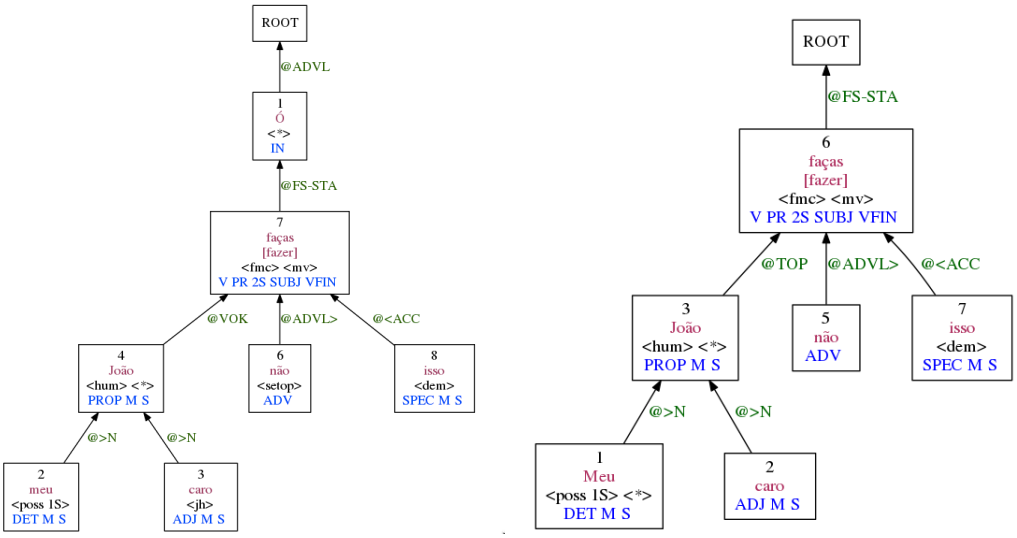



**Figure 3** Sentences parsed by the VISL system, based on the PALAVRAS parser [3]. Left: imperative with *ó*: *Ó meu caro João, não faças isso* 'Hey my dear João, don't do that'; Right: the same sentence without *ó*.

knows how to dance' (subject topic). Again, as with the LX-SUITE parser, in a declarative sentence, the VISL system, also extracts two subjects (Fig. 4, right). Apparently, there is no rule preventing two (heads of) constituents to be analysed as having the same syntactic function. Still, the position of the vocative detached by comma at the end of the sentence (Fig. 4, left) is sufficient to correctly extract the vocative dependency. In this last case, with the vocative detached by comma at the end of the sentence, the LX-SUITE parser still does not produce adequate results (Fig. 5), since it deals with the vocative as a specifier (`SP`), in the same way as the definite article in front of a proper name (v.g. *O Pedro*).

Since PALAVRAS [3] is a rule-base parser, its development is not hindered by the scarcity of occurrences of a given linguistic phenomena in texts, even in large *corpora*, allowing for a very precise description of the grammar of the language. In this sense, our approach to the phenomenon of vocatives, which will also use a rule-based parser, will be similar. However, we believe that a more comprehensive linguistic description of vocative constructions is necessary, in order to obtain a larger coverage of the phenomena. On the contrary, as the language model of the LX-SUITE parser is built upon previously annotated *corpus*, the sparseness of the phenomenon may have had an impact on its performance, regarding this specific grammatical aspect.

In the next section, we present the strategy adopted by the STRING (undisclosed ref.), using the rule-based parser XIP , to parse vocatives in Portuguese.

## 4 Grammar module

Having in mind the linguistic phenomena briefly sketched in Section 1 and the theoretical framework outlined in Section 2, we set out to produce a new module of the dependency

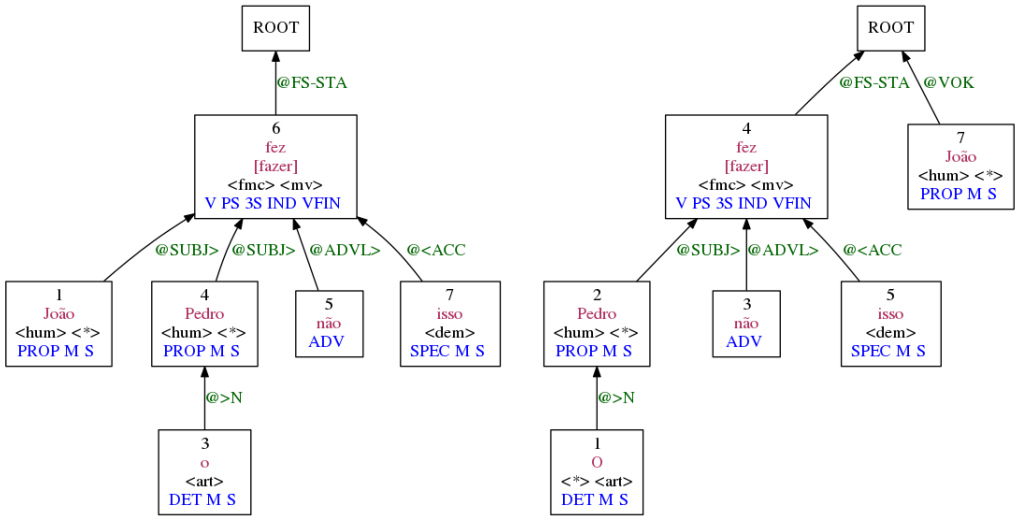



**Figure 4** Declarative/exclamative sentences parsed by the VSIL system, based on the PALAVRAS parser [3]. Left: the vocative at the begining of the sentence: *João, o Pedro fez isso.* 'João, Pedro has not done that'; Right: the same sentence but with the vocative at the end: *O Pedro fez isso, João.* 'Pedro has not done that, João.'.

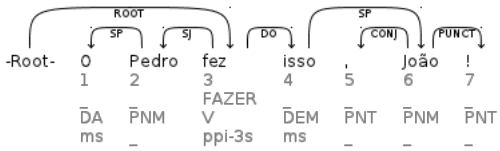



**Figure 5** Declarative/exclamative sentence parsed by the LX-SUITE using the MALTPARSER: vocative detached at the end.

grammar of XIP by building a batch of artificial examples[8], inspired in instances of vocatives found in corpora, and by systematically exploring the formal variation such examples permitted, namely: (i) the presence or absence of interjection *ó* 'hey' in the vocative phrase; (ii) the position of the vocative at the beginning or at the end of the sentence, or in the middle of the sentence and detached by commas; (iii) the use of proper names, both a single noun, as in *João*, and in longer strings of nouns (and prepositions), as in *João da Silva*; (iv) different titles (*engenheiro* 'engineer') and profession nouns (*professor*), used in isolation or in combination (*professor doutor* 'doctor'), eventually abbreviated (*Eng.º* 'engineer'), as well as when combined with proper names or preceded by the respectful addressing form (*senhor* 'sir'); (v) a closed set of adjectives conventionally used in vocatives (*caro*, *estimado*, *excelentíssimo*, *prezado*, *querido*, etc. (vi) the possible presence of an affective possessive, obligatorily in the $1^{st}$person-singular (v.g. *meu*), in all its gender-number values; (vii) a small number of expressions used mostly in vocatives, such as *cavalheiro*, *pá* 'man', *vossa excelência* 'your excelency', as well as certain conventional combinations with the form of two coordinated nouns: *senhoras e senhores* 'ladies and gentleman'.

The general strategy to extract the `VOCATIVE` dependency consists in identifying first the vocative phrases, based on their content and context, in order to mark them with a

[8] These sentences annotated with the `VOCATIVE` dependency can be retrieved from: `https://string.l2f.inesc-id.pt/w/images/1/16/Vocative.txt`.

`vocnp` (vocative noun phrase) feature. This feature is, then, used by a general parsing rule that extracts the `VOCATIVE` dependency. The first part of the process is handled by a *local grammar* (LG) of vocatives, specifically built for this purpose, which takes the form of declarative rules, describing a pattern that must be matched so that the `vocative` feature is added to the phrase node. For example, the following is a LG rule:

```
1> NP[vocnp=+] = ?[lemma:senhor,start], noun | ?[comma] | .
```

that adds the `vocnp` feature to a noun phrase at the begining of a sentence starting with the lema *senhor*, followed by a `noun`, and detached from the remainder of the sentence by a comma. We considered the presence of commas as a necessary context to trigger the local grammar of vocatives. This is not always the case in real texts, where the use of commas is subject to much individual variation. The `noun` is the result of a previous processing stage, the *chunking*, where some sequences of words, like strings of proper names (e.g. *João da Silva*) have been already grouped together as a single noun. The `VOCATIVE` dependency is extracted by the rule:

```
| #3[cat:0]{?*, NP#1[vocnp];PP#1[vocnp] } |
 if ( HEAD(#2,#1) & ~VOCATIVE(#3,#1) )
   VOCATIVE(#3,#2)
```

This rule reads as follows: When a noun phrase (`NP`) or a prepositional phrase (`PP`) (see below, the treatment given to `PP` phrases introduced by *ó*) marked as a vocative phrase (`vocnp`) is found, for which a `VOCATIVE` dependency as not been previously extracted, then that very dependency is extracted between the phrase head and the top node of the sentence (`[cat:0]`).

The vocative extracting rules operate at an earlier stage of the parsing process, before the other major dependencies (subject, direct complement, modifier, etc.) are extracted. Hence the rules that had been already built so far in order to extract these dependencies had to be modified by adding the condition that such constituent had *not* been previously parsed as a vocative. As one can see from the above, the bulk of the parsing is carried out by the *local-grammar* (LG) for vocatives. In the remainder of this section, we provide further details on the vocatives aimed at by the LG.

In most of its occurrences the interjective *ó* is used in combination with a noun phrase to form a vocative in an unambiguous way, This a rule-of-thumb, that explains much of the positive results from the VISL system, as shown in Section 3. In view of this regularity, we decided to chunk those combinations as a special type of prepositional phrase `PP` and mark it with the `vocnp` feature, irrespective of the content of the element appearing after the interjection, as shown in the chunking rule below:

```
> PP[vocnp=+] = interj[lemma:ó], NP.
```

These lead us then to determined other combinations and contexts of *ó* that do not constitute vocatives. For example, the (incorrect?) combination *ó quê?* 'or what?' has been treated as a compound (=multiword) interrogative interjection, whose canonical form is *ou quê?*, e.g. *Está tudo bêbado, ó quê?* 'Is everyone drunk, or what?'. In a similar way, when immediately followed by the interrogatives *quão* 'how [much]', the *ó* is being (incorrectly?) used instead of interjection *oh*. In this case, *ó* does not form a vocative, instead it contributes to the exclamative nature of the sentence, e.g. *Ó quão grande é o mundo!* 'O how large is the world!'. The same happens in the combinations *ó, sim!* and *ó, não!*, used instead of *oh, sim!* and *oh, não!*, and often without the intervening comma.

Due to its syntactic independence from the main clause, vocatives show a remarkable mobility within the sentence, though their most common position in the Portuguese *corpora* we consulted was clearly the beginning of the sentence and, less frequently, some position within the sentence. Cases of vocatives at the end of the sentence are relatively rare. Strings of vocatives found at the beginning of a speech/lecture (as in the Parliament discourses) were treated in the same way as a single vocative at the beginning of an ordinary sentence, e.g. *Sr. Presidente, senhoras e senhores deputados* 'Mister President, ladies and gentlemen (members of the Parliament)'. In this case, each vocative is linked independently to the `TOP` node of the sentence. The same applies to the initial salutation and forms of addressing the reader at the onset of a private correspondence, e.g. *Minha querida esposa* 'my dear wife', *Caríssimo amigo* 'most dear friend', even if in a separate line from the main text.

The complex `noun` chunks, formed with combinations of proper names, profession nouns and honorific titles, are identified by using, on one hand, a large lexicon of given names and surnames, nouns for professions and lists of titles, including abbreviations, and, on the other hand, a specific set of local grammars, that had been previously built for the processing of named entities [8].

A remarkably small set of adjectives is often used conventionally for vocatives. These include the relatively formal *caro* (and its Brazilian correspondent *prezado*) and the much more intimate *querido*; the superlative *caríssimo* is also used; other, less frequent adjectives, are: *belo*, *bom*, *doce*, *grande*, *ilustre*, *lindo*, *pobre*, *rico*, *sábio*, *santo*, *terno*, *triste*, *velho*, etc. To simplify the process of rule building, the most frequently occurring of these adjectives as found in the *corpus* were listed and they were given a new feature (`vocadj`). In the same way, a specific set of nouns are particularly apt to constitute vocatives: *amigo*, *amor*, *bem*, *camarada*, *colega*, *gente*, *leitor*, *menino*, *rapaz*, *senhor*, *súbdito*, *velho*, etc. The most frequent of these nouns were also listed and given a new feature: `vocnoun` (vocative noun). These lists are open, an can be extended at will. Besides this list of nouns, several subsets of nouns in the lexicon are being systematically added this `vocnoun` feature: titles, professions, family relations, etc. Notice that these adjectives can also modify proper names in a vocative: *caro João*, *querida Rita*. As mentioned above, a possessive modifier can also be combined, e.g. *meu caro amigo*, *minha querida Ana*. This possessive refers to the speaker, hence it can only be a $1^{st}$-person-singular, v.g. *meu*, *minha*, *meus*, and *minhas* 'my'. Notice that this person-number can also be used in insults, along with a $3^{rd}$-person-singular (*seu*, *sua*, *seus*, *suas*): '*meu/seu imbecil*'. This type of ambiguity was not addressed at this point. Below, one can see the form of two rules describing such combinations involving possessives, `vocadj` and `vocnoun`, at the beginning of the sentence:

```
1> NP[vocnp=+] = pron[poss,poss1s,start], (adj[vocadj]),
   noun[vocnoun];noun[human,proper] | ?[comma] | .
1> NP[vocnp=+] =  adj[vocadj,start],
   noun[vocnoun];noun[human,proper] | ?[comma] | .
```

Very specific and conventional combinations were left out of this subsets, and they were treated individually. For example, combinations such as *estimado cliente*, *respeitável público*, where the adjective is obligatory, and the possessive is not allowed, v.g. **meu estimado cliente*, **meu cliente*, **cliente*; **meu respeitável público*, **meu público*, **público*. Slightly less constraint is the noun *cavalheiro* 'gentleman', which can be facultatively modified by an adjective, *excelentíssimo cavalheiro* 'most excellent gentleman', but not by the possessive, **meu cavalheiro*.

We also mention the case of the formal way of addressing, *vossa excelência* 'your excellency', and the corresponding plural *vossas excelências* 'your excellencies', which is an excepcional

use of the possessive $2^{nd}$-person-plural. The possessive, however, is facultative. Though not alway used as a vocative, e.g. it can be reasonably identified given a detached context (i.e. separated by the sentence by commas).

The so-called interjection *pá* is always used as a vocative, sometimes preceded by the true interjections *ó*, *oh* and *he*, e.g. *Não , pá, não é isso.* 'No, man, it's not that.'; *Ó pá, acho que devíamos seguir o conselho ... mas, eh pá, isso vive muito da altura* 'Hey man, [I] think that [we] should follow the advice... but, hey man, that depends very much of timing' (real examples taken from *corpora*).

Another form of vocative, similar in use to *pá* (very much colloquial but typical of youngsters), consist in an isolated possessive pronoun, with the same inflection restrictions as described above, in a detached context, e.g. *Vi-te no cinema, meu.* 'I saw you in the movies, man.' Curiously, though there is no linguistic reason why it should not be so, the feminine forms of this possessive have not been found in written *corpora*.

## 5 Evaluation

As mentioned before, only 29 instances of vocative are reported in the documentation of the *Floresta Sintá(c)tica* treebank[9], regarding the revised portion (*Bosque*). To the best of our knowledge, there is no other available *corpus*, annotated with vocative constructions in Portuguese. Therefore, a procedure had to be devised to evaluate the precision of the STRING's parser XIP in the extraction of the `VOCATIVE` dependency. This section outlines the procedure adopted to build a reasonably sized sample of examples of vocatives and describes a preliminary evaluation of this module of the Portuguese grammar built in XIP [1].

To this end, real sentences were retrieved from the CetemPúblico *corpus* [15][10] using Linguateca's interface. The sentences were extracted using commas or full stops as boundaries of the targeted expressions, trying to capture vocatives at the beginning, the end and in the middle of the sentence, that is, detached from the main sentence by commas. The targeted expressions were represented in the queries by part-of-speech alone, except for the possessive $1^{st}$-person-singular (*meu* 'my') and its gender-number inflected forms. A distinction was made between proper and common nouns. Specific patterns were queried, namely, the isolated possessive, the interjections *pá* and *ó*. Once retrieved, duplicates were removed and the *corpus*' extracts were randomly sorted. Only the first 100 sentences from each pattern were kept for the evaluation.

These patterns, though constrained by the contextual delimiters with which they were defined, are broad in definition, so they can arguably be used to calculate the precision of the parser and – in an approximate way and only for those patterns, of course – its coverage/recall. At this stage of the grammar's development, it seems more pertinent to have a generic overview of the performance, as many of the shortcomings found can easily be corrected. Table 1 shows the breakdown of the retrieved patterns

As it can be seen, most frequently occurring patterns are the *Poss N* string and the medial context (between commas). The distribution of the patterns across the three contexts here defined is very uneven, in some cases only some very few instances were found.

The attentive reader will have noticed that some patterns were left out from the experiments: the isolated noun (both proper and common), and the the pattern *Adj PROP*. The first, though it can be used for vocatives (e.g. *João, não faças isso.* 'João, don't do that')

[9] `http://www.linguateca.pt/Floresta/BibliaFlorestal/anexo4.html`
[10] `http://www.linguateca.pt/acesso/corpus.php?corpus=CETEMPUBLICO`

**Table 1** Patterns retrieved from the *corpus*. *Poss*= possessive, *N*= noun (common), *PROP*= proper noun,*Adj*= adjective, `[vocnoun]`= vocative noun, `[vocadj]`= vocative adjective.

| Pattern | # _ , | , _ , | , _ # | Total |
|---|---|---|---|---|
| *Poss* | 1 | 21 | 8 | 30 |
| *Poss Adj* | 1 | 33 | 7 | 41 |
| *Poss N* | 43 | 285 | 97 | **425** |
| *Poss PROP* | 25 | 66 | 10 | 101 |
| *Poss Adj N* | 6 | 43 | 7 | 56 |
| *Poss Adj PROP* | 5 | 16 | 5 | 26 |
| **Subtotal** | 81 | **464** | 134 | 679 |
| *pá* | 0 | 41 | 31 | 72 |

produced a large number of matches (over 133,000), and a cursory analysis showed that most of them are spurious cases, mostly appositive `NP`. The second pattern occurred less frequently (736 matches). It can also provide instances of vocative (e.g. *Caro João, não faças isso.* 'Dear João, don't do that'), but in most cases, they correspond to: (i) the metalinguistic operador *chamado* 'called' (and its synonyms, like *denominado* 'idem'), used in appositions (e.g. *um produto semelhante , chamado Tiger*); (ii) the adjective *antiga*, also used in appositions (e.g. [*Visita a*] *Varanasi , antiga Benares, ...* '[visit to] Varanasi, formerly known as Benares') (iii) part-of-speech ambiguous adjectives (e.g. *viva Timor Leste* '[long] live *East Timor*'); (iv) compound proper nouns (e.g. *Nova Deli*) that, unlike STRING, the *corpus* does not treated as a single token. For both patterns, some strategy must be devised to rule out these cases. This is the precisely the point of the lexical focus of this paper's approach, by defining subsets of nouns and adjectives frequently occurring in vocative constructions.

The interjection *pá* does not constitute a real issue for the parser, since its use in isolation and the exceptions described in the grammar precluded false-positives. In the same way, the interjection *ó*, which occurred 475 times in the *corpus*, did not pose much of a difficulty to the parser. For example, since the STRING's lexicon includes the proper noun *Ó*, this word is always chunked together with the other elements of the name, forming a single noun chunk, e.g. *Luís do Ó*, *Nossa Senhora do Ó* 'Our lady of Ó [=birth]'. Combinations of *ó* with adjectives are also an important cause for missing vocatives, e.g. *Ó poderosos* 'O powerfull [people]'. Some of the false-negatives correspond to strings where the vocative is not strictly at beginning and the end of the sentence, as some separator (quotes and dashes) occur, e.g. *"–Ó filho, ...* 'O [my] son'; *J.G. – Ó pá , eu digo-te* 'J.G.[speaker] – O man, I'll tell you'. These all textual-orthographic issues that surely have to be addressed, probably at a pre-processing stage, but are somehow marginal to the scope of this paper.

Regarding the patterns with possessives, though the majority of the instances of vocatives have been correctly identified and the corresponding dependency extracted (approximately 75% precision), some remarks are in order. A major source for false-negatives are lexical *lacunae*: *Hastings* as proper name wass missing from the lexicon, thus, in some contexts some vocatives with this name were missed. Another reason is the incomplete (semantic) description of nouns forming homogenous subsets of the lexicon. This process of systematically extending (by way of rules) the feature `vocnoun` in still underway. This explains why several vocatives involving nouns designating family relations and professions were missed, e.g. *minha filha* 'my daughter', *meu general* 'my general', *meu caro psiquiatra* 'my dear psychiatrist'. Other cases of `vocnoun` not previously considered were also found: *amado*, *bem-amado* 'loved-one', *amor* 'love', *nené* 'baby', *jóia* 'jewel', *santo* 'saint', etc.

Other false-negatives relate to the ambiguous expression *meu deus* 'my god' (upper/lower-case variation involved), which can correspond to either an interjection (*Ai, meu Deus, eu quero ir para a beira dele* 'Oh, my God, I want to go near him') – which is the most frequent case – or a vocative proper (*Obrigado, meu Deus, ... clamava o pastor* 'Thank you, my God, ... the shepherd cried') – corresponding to a real address to the deity. Because of this ambiguity, this expression has been left out of the scope of the vocatives' grammar.

A special attention should be paid to names involving *insults* (usually with profanity words), which systematically enter vocative constructions, e.g. *meu grande paneleiro* 'my big fagot', *meu sacana* 'you bastard', *minha cabra* 'you bitch'. As mentioned before, vocative constructions can be used for insulting the addressee, though the specific lexicon involved has not been described in the STRING yet (see [5]). The lexicon-grammar of insults is the topic for another study.

## 6 Conclusion and future work

This paper aimed at providing a coherent theoretical framework to support the formal description of vocative constructions within a computational grammar of Portuguese. From a (necessarily brief) overview of different linguistic-grammatical perspectives on the status of vocatives, the paper has shown some of the issues vocatives raise for an adequate parsing. Theses issues regard, mostly, (i) the place on insertion (or attachment) of the vocative phrases within the sentence parse or to the sentence's topmost node (if the system adopts such formalism); and (ii) the inadequate parsing of other sentence constituents, foremost the subject. In our perspective, vocatives are to be treated as an independent syntactic function, distinct from sentence-internal constituents like the subject; and they hinge on (zeroed) metalinguistic operators involved in the communicative functions of language, hence, they should be linked to the sentence as a whole. A (very brief) comparison of two Portuguese parsing systems showed: (i) one that apparently does not consider a specific dependency for vocatives, with all the inadequacies in the resulting parses, whenever a vocative is involved; being a statistical parser, based on a language model, the sparsity of the phenomena may be the source for the vocatives not having been taken into account. (ii) another system that, while considering a vocative dependency, raises several issues on the *locus* the vocative phrase should be attached to, besides the coverage of different linguistic aspects of this phenomena.

We set out to produce a module for the processing of vocatives within the computational grammar of Portuguese built for XIP [1], by systematically exploring the formal variation found in an initial overview of examples taken from *corpus*. The strategy here adopted relies on preexisting modules for identifying and chunking proper names, including titles and profession nouns, and on a large-sized lexicon. The processing of vocatives involves two major steps, illustrated in detail by the paper: First, with a specifically built *local-grammar*, declarative rules identify vocative phrases and tag them with a `vocnp` feature, which is then used by the dependency rules to extract the `VOCATIVE` dependency. For a preliminary evaluation of the parser, a set of relatively flexible patterns were extracted from the CetemPúblico *corpus*, and results were commented in detail. Current results are promising but there is still room for improvement, especially in the fine-grained description of lexical features, the intersection of vocatives and the grammar of insults, and in the removal of spurious dependencies still left in the parse. This will constitute our next steps towards a comprehensive and efficient grammar for parsing vocatives in Portuguese.

## References


1. Salah Ait-Mokhtar, Jean Pierre Chanod, and Claude Roux. Robustness beyond shallowness: incremental dependency parsing. *Natural Language Engineering*, 8(3):121–144, 2002.
2. James M. Anderson. Case grammar. In Keith Brown and Jim Miller, editors, *Concise Encyclopedia of Syntactic Theories*, pages 58–65. Pergamon, 1996.
3. Eckard Bick. *The Parsing System "PALAVRAS". Automatic Grammatical Analysis of Portuguese in a Constraint Grammar Framework*. Arhus University Press, 2000.
4. Ana Maria Brito, Inês Duate, and Gabriela Matos. Estrutura das frases simples e tipos de frases. In Maria Helena Mira Mateus, Ana Maria Brito, Inês Duarte, and Isabel Hub Faria, editors, *Gramática da Língua Portuguesa*, pages 432–506. Caminho, 3rd edition, 2003.
5. Paula Carvalho. *Análise e representação de construções adjectivais para processamento automático de texto. Adjectivos intransitivos humanos*. PhD thesis, Faculdade de Letras, Universidade de Lisboa, 2007.
6. Authors: Francisco Costa and António Branco. LX-Gram: A deep linguistic processing grammar for portuguese. In *Computational Processing of the Portuguese Language (PROPOR)*, pages 86–89, 2010.
7. Celso Cunha and Luís Lindley-Cintra. *Nova Gramática do Português Contempoâneo*. João Sá da Costa, 1986.
8. 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: O Segundo HAREM*. Linguateca, 2009.
9. Zellig Sabettai Harris. *A Grammar of English on Mathematical Principles*. Jouhn Wiley & Sons, Wiley-Interscience Pub., 1982.
10. Zellig Sabettai Harris. *A Theory of Language and Information. A Mathematical Approach*. Clarendon Press, 1991.
11. Nuno Mamede, Jorge Baptista, Cláudio Diniz, and Vera Cabarrão. STRING - a hybrid statistical and rule-based natural language processing chain for Portuguese. In *Intl. Conf. Computational Processing of Portuguese (PROPOR)*, 2012.
12. Christian Molinier and Françoise Levrier. *Grammaire des adverbes: description des formes en -ment*. Droz, 2000.
13. Joakim Nivre, Marie-Catherine de Marneffe, Filip Ginter, Yoav Goldberg, Jan Hajic, Christopher D. Manning, Ryan McDonald, Slav Petrov, Sampo Pyysalo, Natalia Silveira, Reut Tsarfaty, and Daniel Zeman. Universal dependencies v1: A multilingual treebank collection. In *10th International Conference on Language Resources and Evaluation (LREC)*, pages 1659–1666, 2016.
14. Eduardo Buzaglo Paiva Raposo, Maria Fernanda Bacelar do Nascimento, Maria Antónia Coelho da Mota, Luísa Segura, Amália Mendes, Graça Vicente, and Rita Veloso. *Gramática do Português*. Fundação Calouste Gulbenkian, 2013.
15. Diana Santos and Paulo Rocha. Evaluating CETEMPúblico: A free resource for portuguese. In *39 Annual Meeting of the Association for Computational Linguistics*, pages 442–449, 2001.
16. Mariona Taulé, M. Antònia Martí, and Marta Recasens. AnCora: Multilevel annotated corpora for Catalan and Spanish. In *6th International Conference on Language Resources and Evaluation (LREC)*, pages 96–101, 2008.

# Linear Operators in Information Retrieval*

**Hawete Hattab[1] and Rabeb Mbarek[2]**

1 **Umm Al-qura University, Al-Jumum University College, Department of Mathematics, Makkah, KSA**
hshattab@uqu.edu.sa

2 **Sfax University, Multimedia Information Systems and Advanced Computing Laboratory, Sfax, Tunisia**
rabeb.hattab@gmail.com

## Abstract

In this paper, we propose a pseudo-relevance feedback approach based on linear operators: vector space basis change and cross product. The aim of pseudo-relevance feedback methods based on vector space basis change IBM (Ideal Basis Method) is to optimally separate relevant and irrelevant documents. Whereas the aim of pseudo-relevance feedback method based on cross product AI (Absorption of irrelevance) is to effectively exploit irrelevant documents. We show how to combine IBM methods with AI methods. The combination methods IBM+AI are evaluated experimentally on two TREC collections (TREC-7 ad hoc and TREC-8 ad hoc). The experiments show that these methods improve previous works.

**1998 ACM Subject Classification** H.3.3 Information Search and Retrieval

**Keywords and phrases** Pseudo-relevance feedback, vector space basis change, Cross product

**Digital Object Identifier** 10.4230/OASIcs.SLATE.2017.23

## 1 Introduction

The main goal of an Information Retrieval System (IRS) is to find the subset of documents potentially most relevant to a given query. Most IRS compute a numeric score which measures the relevance of an object with respect to a query, and rank the objects according to this value. Several Information Retrieval (IR) models, including Vector Space Model (VSM) [15], probabilistic models [12] and language models [11], have been proposed to model this scoring function.

In the VSM, documents and queries are represented by vectors. Each component in a vector represents the weight of a term in the document and so the set of index terms (original vector space basis) generates documents and queries.

The idea of Relevance Feedback (RF) is to take the results that are initially returned from a given query and to use information about whether or not these results are relevant to perform a new query. The most commonly used RF methods aim to rewrite the user query. In the VSM, RF is usually undertaken by re-weighting the query terms without any modification in the vector space basis. With respect to the initial vector space basis (index terms), relevant and irrelevant documents share some terms (at least the terms of the query which selected these documents). The Vector Space Basis Change (VSBC) is the algebraic operator responsible for change of basis and it is parameterized by a transition matrix. If

* This work was supported financially by the Deanship of Scientific Research at Umm Al-Qura University to Dr. Hawete Hattab (Grant Code: 15-COM-3-1-0018).





we change the vector space basis, then each vector component changes depending on this matrix. According to [9], the VSBC causes vector behavior changes. The best framework that could make the VSBC technique into application is RF: the user selects relevant and irrelevant documents in an initial ranking and instead of reformulating the query, we change the vector space basis in which it is written (as well as the documents). The strategy of VSBC has been shown to be effective in separating relevant document and irrelevant ones. Recently, using this strategy, some feedback algorithms have been developed [6, 8, 9]. These techniques are called IBM (Ideal Basis Model).

Pseudo-Relevance Feedback (PRF) is a well-studied query expansion technique which assumes that the top-ranked documents of the initial retrieval are relevant and expansion terms are then extracted from them [4]. If there are only a few or no relevant documents in the top-ranked documents, then we can add terms which have no relationships with the topic of relevance of the query and so PRF only improves the performance of queries which have good initial results. Thus, to improve the PRF technique it suffices to effectively select from top-ranked documents those terms that are most likely relevant to the query topic.

In [7], Mbarek et al. proposed to solve this problem by exploiting the role of irrelevant documents in selecting better expansion terms from the top-ranked documents. In particular they built an absorbing document which is the cross product of linearly independent irrelevant documents. This document is orthogonal to irrelevant ones. This method is called Absorption of Irrelevance (AI).

In [10], Mbarek et al. investigated the role of irrelevant documents in document re-ranking. This proposed re-ranking strategy is based on a negative RF approach which takes into account irrelevant documents in the initial document ranking. The key idea behind this approach is to use the absorbing document [7], which is orthogonal to irrelevant documents, to re-rank documents on the ground of their similarity with respect to the absorbing document.

Score vectors from two different scoring methods can be combined to yield a new score vector, and thereby a new scoring method. If the two scoring methods have complementary advantages, the combined scoring method may perform better than either scoring method alone. In this paper, we propose to combine IBM with AI methods.

The paper is structured as follows. Section 2 describes the two strategies IBM and AI and also the combination methods IBM+AI. Experiments performed for evaluating our combined approaches are presented in Section 3. The last section concludes.

## 2 Linear Operators in information retrieval

### 2.1 Ideal Basis Methods: IBM

In the IBM approches, the optimal matrix $M^*$ puts the relevant documents gathered to their centroïd $g_R$ and the irrelevant documents far from it. $g_R$ is done by:

$$g_R = \frac{1}{|R|} \sum_{d \in R} d$$

where $R$ is the set of relevant documents.

The optimal matrix $M^*$ should minimize the sum of squared distances between each relevant document and $g_R$ i.e.:

$$M^* = \underset{M \in M_n(\mathbb{R})}{\arg\min} \sum_{d \in R} (d - g_R)^T . M^T M . (d - g_R) \,. \tag{1}$$

By the same, matrix $M^*$ should maximize the sum of squared distances of each irrelevant document and $g_R$, which leads on the following:

$$M^* = \underset{M \in M_n(\mathbb{R})}{\arg\max} \sum_{d \in S} (d - g_R)^T . M^T M . (d - g_R) \tag{2}$$

where $S$ is the set of irrelevant documents.

According to [6] (IBM1 method), the optimal matrix $M^*$ should minimize the quotient and so equations 1 and 2 result on the following single equation:

$$M^* = \underset{M \in M_n(R)}{\arg\min} \frac{\sum_{d \in R} (d - g_R)^T . M^T M . (d - g_R) + \theta}{\sum_{d \in S} (d - g_R)^T . M^T M . (d - g_R) + \theta} \tag{3}$$

where $\theta$ is a real parameter.

According to [8] (IBM2 method), the optimal matrix $M^*$ should maximize the difference and so equations 1 and 2 result on the following single equation:

$$M^* = \underset{M \in M_n(\mathbb{R})}{\arg\max} \left( \sum_{d \in S} (d - g_R)^T . M^T M . (d - g_R) - \sum_{d \in R} (d - g_R)^T . M^T M . (d - g_R) \right) . \tag{4}$$

In [9], Mbarek et al. built the transition matrix $M^*$ using an algebraic method (IBM3 method). In the IBM3 method, if $d \in R$, the optimal matrix $M^*$, which satisfies equation 1, should contract the vector $d - g_R$ which implies that there exists a real parameter $0 < \gamma < 1$ such that:

$$M(d - g_R) = \gamma(d - g_R) .$$

Then $d - g_R$ is an eigenvector of the matrix $M$ associated to the eigenvalue $\gamma$.

If $d \in S$, the optimal matrix $M^*$, which satisfies equation 2, should dilate the vector $d - g_R$ which implies that:

$$M(d - g_R) = (1 + \gamma)(d - g_R) .$$

Then $d - g_R$ is an eigenvector of the matrix $M$ associated to the eigenvalue $1 + \gamma$. $M^*$ is a diagonalized matrix (similar to a diagonal matrix) having two distinct eigenvalues $\gamma$ and $1 + \gamma$. Therefore:

$$M = V.D.V^{-1} \tag{5}$$

where $D$ is the eigenvalues matrix of $M$, and $V$ is the eigenvectors Matrix of $M$.

## 2.2 Absorption of Irrelevance: AI

One of the main problem in information retrieval is how to exploit effectively the irrelevant documents? To give a satisfactory answer we need to build a solid structure which represents the irrelevant content. In the theory of vector space, the cross product is a very suitable candidate. Mbarek et al. used this approach in [10].

This section describes the PRF approach based on irrelevant documents. The main idea is to build an absorbing document noted $\widetilde{d}$, as the cross product of linearly independent

irrelevant documents and then terms of this document are used to re-weight the terms of the original query in the following way:

$$Q_{new} = \delta.Q_{int} + (1 - \delta).\widetilde{d} \tag{6}$$

where $\delta$ is a real parameter. Note that if $\delta = 0$ we obtain a re-ranking method studied in [10].

Let $D_{init}$ be the initial set of ranked documents and let $n$ be the number of indexing terms of $D_{init}$. Identifying relevant documents $D^+$ is quite straightforward, we assume that the top-ranked $k$ documents in $D_{init}$ as relevant. Let $p$ be the number of expansion terms of the top-ranked $k$ documents. Identifying irrelevant documents is not trivial, we propose to select the set of irrelevant documents $D^-$ from the bottom of $D_{init}$. Let $m$ be the number of linearly independent documents of $D^-$. Let $u_1$,...,$u_m$ denote these irrelevant documents. Each irrelevant document of $D^-$ is a linear combination of $u_1$,...,$u_m$.

To compute the absorbing document $\widetilde{d}$ which is the cross product of $u_1$,...,$u_m$, each vector must be written as a linear combination of $m + 1$ indexing terms [10]. For this reason $p = m + 1$.

The absorbing document is $\widetilde{d} = u_1 \wedge ... \wedge u_m$. By [10], $\widetilde{d}$ is orthogonal to each irrelevant document.

## 2.3 Combination methods IBM+AI

This section describes the PRF combination methods IBM+AI. We propose a novel PRF approach based on linear operators: vector space basis change and cross product, i.e. we propose to combine the IBM methods and AI method.

Score vectors from two different scoring methods (IBM and AI) can be combined to yield a new scoring method (IBM+AI). If the two scoring methods have complementary advantages, the combined scoring method may perform better than either scoring method alone. Note that VSBC-based methods IBM and cross product based method AI have complementary advantages. Indeed, the main goal of VSBC-based methods IBM is to optimally separate relevant and irrelevant documents and the main goal of cross product based method AI is to build an absorbing document, named $\widetilde{d}$, as the cross product of linearly independent irrelevant documents selected from the bottom. For these reasons, we propose to combine IBM and AI methods in the following way:

$$Q_{new} = \alpha.M^{*T}.M^*.Q_{int} + (1 - \alpha).\widetilde{d} \tag{7}$$

where $M^*$ is the transition matrix (Equations 3, 4 and 5), $\widetilde{d}$ is the absorbing document and $\alpha$ is a real parameter.

# 3 Experiments

The main goal of our experiments is to investigate whether the new combination methods IBM+AI described above perform better than BM25, BM25+Rocchio and linear operators based approaches (IBM and AI).

## 3.1 Evaluation Methodology

We set up a baseline system based on the BM25 formula proposed in [13]. BM25 parameters are $b$ and $k_1$. The TREC-7 ad hoc and TREC-8 ad hoc collections were used for test. They consist of the same set of documents (i.e., TREC disks 4 and 5, containing approximately 2

gigabytes of data) and different query sets (topics 351-400 and topics 401-450, respectively). The full topic statement was considered, including title, description, and narrative. Note that we choose the TREC-7 ad hoc and TREC-8 ad hoc test collections as they have the highest frequency of scores reported for ad hoc retrieval in recent years [1].

To generate a query $Q_{int}$, the title of a topic was used, thus falling into line with the common practice of TREC experiments; description and narrative title were not used. Using $Q_{int}$ the top 1000 documents are retrieved from the collections.

The set of relevant documents $D^+$ is the set of top-ranked $k$ documents, while the set of irrelevant documents $D^-$ is the set of retrieved documents $501 - 1000$, assumed to be irrelevant. This strategy is widely used in IR [13, 3].

The experiments consist of re-ranking the results of the Baseline Model. For our combined approach IBM+AI the reformulated query is done in Equation 7.

We compare our approach IBM+AI to the baseline model BM25 and to the traditional combination of BM25 and Rocchio's feedback model[1] (BM25+Rocchio).

The following improved version [16] of the original Rocchio's formula [14] is used:

$$Q_{new} = \lambda.Q_{int} + \beta.\frac{1}{|D^+|}\sum_{d \in D^+} d\,. \tag{8}$$

Here, $\lambda$ and $\beta$ are tuning constants controlling how much we rely on the original query and the feedback information. In practice, we can always fix $\lambda$ at 1, and only study $\beta$ in order to get better performance.

For the linear operators based approaches and the BM25+Rocchio model, the retrieved documents are re-ranked by the inner product done by:

$$< Q_{new}, d >= Q_{new}^T.d\,. \tag{9}$$

## 3.2 Parameter settings

The experiments and the evaluations are as follow. Comparison between IBM models, AI model, IBM+AI models, the BM25 model and the BM25+Rocchio model.

We vary the BM25 parameters $k_1$ from 1 to 3 in steps of 0.1 and $b$ from 0.05 to 1 in steps of 0.05, and vary the parameters $\theta$ in Equation 3, $\gamma$ in Equation 5, $\delta$ in Equation 6, $\alpha$ in Equation 7 and $\beta$ in Equation 8 from 0 to 1 in steps of 0.1.

The models IBM, AI, IBM+AI and Rocchio depend on the RF parameters. One parameter was the number $k$ of relevant documents. The other parameter was the number $p$ of expansion terms. We varied these two parameters in the following way: $k \in \{1, 2, 3, 4, 5\}$ et $p \in \{10, 20, 30, 50\}$. The number $m$ of linearly independent irrelevant documents must be equal to $p - 1$[2].

## 3.3 Results

To evaluate the performance of our approaches IBM+AI we use MAP, R-Precision, $P$@5, $P$@10 and $P$@20 as evaluation measures. These measures are the most commonly used measures of overall retrieval performance [2].

We present here the behavior of evaluation mesures on TREC-7 and TREC-8 for all models. The optimal results are illustrated in Tables 1 and 2.

[1] According to [17], BM25 [13] term weighting coupled with Rocchio feedback remains a strong baseline.
[2] In a vector space of dimension $n$, we compute the cross product of $n - 1$ vectors.

**Table 1** Comparison of the performance on TREC-7 collection.

| | BM25+Roc | AI | IBM1 | IBM2 | IBM3 | IBM1+AI | IBM2+AI | IBM3+AI |
|---|---|---|---|---|---|---|---|---|
| MAP | 0.253 | 0.283 | 0.263 | 0.269 | 0.276 | 0.296 | 0.311 | 0.323 |
| R-Prec | 0.295 | 0.355 | 0.317 | 0.321 | 0.327 | 0.365 | 0.374 | 0.387 |
| $P$@5 | 0.451 | 0.600 | 0.469 | 0.471 | 0.477 | 0.622 | 0.638 | 0.649 |
| $P$@10 | 0.440 | 0.560 | 0.460 | 0.480 | 0.510 | 0.580 | 0.590 | 0.598 |
| $P$@20 | 0.381 | 0.513 | 0.431 | 0.452 | 0.463 | 0.526 | 0.544 | 0.570 |
| Para | $\beta$=1<br>$k$=2<br>$p$=30 | $k$=2<br>$p$=50<br>$\delta = 0.3$<br>$m = 49$ | $k = 2$<br>$p = 50$<br>$\theta = 0.3$<br>$m = 49$ | $k = 2$<br>$p = 50$<br>$m = 49$ | $k = 3$<br>$p = 50$<br>$\gamma = 0.4$<br>$m = 49$ | $k = 2$<br>$p = 50$<br>$\alpha = 0.3$<br>$m = 49$ | $k = 2$<br>$p = 50$<br>$\alpha = 0.3$<br>$m = 49$ | $k = 3$<br>$p = 50$<br>$\alpha = 0.3$<br>$m = 49$ |

**Table 2** Comparison of the performance on TREC-8 collection.

| | BM25+Roc | AI | IBM1 | IBM2 | IBM3 | IBM1+AI | IBM2+AI | IBM3+AI |
|---|---|---|---|---|---|---|---|---|
| MAP | 0.264 | 0.302 | 0.281 | 0.284 | 0.289 | 0.311 | 0.324 | 0.334 |
| R-Prec | 0.313 | 0.383 | 0.356 | 0.363 | 0.371 | 0.388 | 0.390 | 0.398 |
| $P$@5 | 0.462 | 0.650 | 0.482 | 0.484 | 0.487 | 0.663 | 0.670 | 0.685 |
| $P$@10 | 0.425 | 0.590 | 0.454 | 0.465 | 0.471 | 0.610 | 0.646 | 0.654 |
| $P$@20 | 0.392 | 0.530 | 0.421 | 0.431 | 0.442 | 0.543 | 0.568 | 0.590 |
| Para | $\beta$=1<br>$k$=2<br>$p$=30 | $k$=2<br>$p$=50<br>$\delta = 0.3$<br>$m = 49$ | $k = 2$<br>$p = 50$<br>$\theta = 0.3$<br>$m = 49$ | $k = 2$<br>$p = 50$<br>$m = 49$ | $k = 3$<br>$p = 50$<br>$\gamma = 0.5$<br>$m = 49$ | $k = 2$<br>$p = 50$<br>$\alpha = 0.3$<br>$m = 49$ | $k = 2$<br>$p = 50$<br>$\alpha = 0.3$<br>$m = 49$ | $k = 3$<br>$p = 50$<br>$\alpha = 0.3$<br>$m = 49$ |

Tables 1 and 2 show the performance of BM25, BM25+Rocchio and various methods based on linear operators on TREC-7 ad hoc and TREC-8 ad hoc collections. For all methods, the best parameter values are shown.

The first two columns of Tables 1 and 2 report scores for BM25 and for the traditional combination BM25+Rocchio. It is evident that BM25+Rocchio outperforms BM25 which proves that the BM25 term weighting coupled with Rocchio feedback remains a strong baseline [17]. The traditional combination BM25+Rocchio is the standard Rocchio model using BM25 weights. These weights use the $b$ and $k_1$ parameters that are optimal for the BM25 method. When BM25 is used as a weighting scheme prior to Rocchio, the optimal parameters may be different, but fixed parameters are used for the sake of simplicity.

Performance results for the VSBC-based methods IBM1, IBM2 and IBM3 in Tables 1 and 2 are based on the computing of the transition matrices $M^*$ (Equations 4, 3 and 5). These matrices use control and feedback parameters. Table 1 and 2 show that these methods outperform the BM25 and BM25+Rocchio methods. In particular, method IBM3, which uses the algebraic properties of the transition matrix, outperforms IBM1 and IBM2.

Performance results for the cross product based method AI, in the third column of Tables 1 and 2, is based on the computing of the cross product of the irrelevant documents $\widetilde{d}$ [10]. Tables 1 and 2 show that this method outperforms the BM25, BM25+Rocchio methods and the VSBC-based methods IBM.

Tables 1 and 2 also show the performance of combination methods IBM1+AI, IBM2+AI and IBM3+AI. For each dataset and combination method, the best values of parameters are shown. The three combination methods, first, outperform BM25 and BM25+Rocchio models. Second, they outperform VSBC-based methods IBM1, IBM2 and IBM3 and also the cross product based method AI. The improvements are big and of practical importance for all evaluation measures.

The results of the proposed approaches IBM+AI are superior and consistent for five measures (MAP, R-Precision, $P$@5, $P$@10 and $P$@20) showed in Tables 1 and 2 in both

collections, TREC-7 and TREC-8. The experimental results prove that our combination methods outperform the BM25, BM25+Rocchio, IBM1, IBM2, IBM3 and AI methods significantly[3].

## 3.4 Discussion

Three parameters in our combination methods have been set. The first, $k$, is the number of top-ranked relevant documents. The second, $p$, is the number of expansion terms. The third one is the controlling parameter $\alpha$. For the irrelevant documents, in practice, we select $m = p - 1$ linearly independent documents.

In the following we will study the impact of varying the two RF parameters $k$ and $p$ and the parameter $\alpha$ .

The experiments show that the variation of $k$ involves the variation of the performance of each IBM+AI method. Indeed, if $k$ exceeds 3, then the performance decreases.

The experiments also show that the variation of the number of expansion terms $p$ involves the variation of the performance of each IBM+AI method. Indeed, if $p$ increases, then the performance increases also. Indeed, if $p$ increases, using the relation: $m = p - 1$, then $m$ increases, and so we can control more irrelevant documents.

The experiments also show that the variation of the parameter $\alpha$ involves the variation of the performance of each IBM+AI method.

# 4 Conclusion and future work

In this paper, the combination approaches has shown their effectiveness with respect to a baseline system based on BM25 and the traditional combination of BM25 and Rocchio model. Moreover, the evaluation has proved that the combination approaches may perform better than either scoring method alone. These results were duplicated on two test TREC collections (TREC-7 ad hoc and TREC-8 ad hoc).

The main outcome of this work is that how linear operator based methods improve search accuracy for difficult queries?

In this paper we apply linear operator to build a geometric PRF. In a future work we intend to apply super linear algebra [5] to solve the problem of semantic relations between terms (synonymy, polysemy. . . ).

**Acknowledgements.** The authors would like to thank the Deanship of Scientific Research at Umm Al-Qura University for the continuous support.

## References

1 Timothy G. Armstrong, Alistair Moffat, William Webber, and Justin Zobel. Improvements that don't add up: Ad-hoc retrieval results since 1998. In *18th ACM Conference on Information and Knowledge Management (CIKM)*, pages 601–610, 2009.

2 Javed A. Aslam, Emine Yilmaz, and Virgiliu Pavlu. A geometric interpretation of r-precision and its correlation with average precision. In *28th Annual International Conference on Research and Development in Information Retrieval (SIGIR)*, pages 573–574, 2005.

[3] Statistically significant improvement according to the Student t-test at the 0.05 level.

3 Pierpaolo Basile, Annalina Caputo, and Giovanni Semeraro. Negation for document re-ranking in ad-hoc retrieval. In *Third International Conference on Advances in Information Retrieval Theory (ICTIR)*, pages 285–296, 2011.

4 Efthimis N. Efthimiadis and Paul V. Biron. Ucla-okapi at TREC-2: query expansion experiments. In *Second Text REtrieval Conference (TREC)*, pages 278–290, 1993.

5 W. B. Vasantha Kandasamy and Florentin Smarandache. *Super Linear Algebra*. InfoLearnQuest, Ann Arbor, 2008. Available in arXiv:0807.3013.

6 Rabeb Mbarek and Mohamed Tmar. Relevance feedback method based on vector space basis change. In *19th International Symposium String Processing and Information Retrieval (SPIRE)*, pages 342–347, 2012.

7 Rabeb Mbarek, Mohamed Tmar, and Hawete Hattab Mohand Boughanem. On improving pseudo-relevance feedback using an absorbing document. *International Journal of Web Applications (IJWA)*, 8(1):8–15, 2016.

8 Rabeb Mbarek, Mohamed Tmar, and Hawete Hattab. A new relevance feedback algorithm based on vector space basis change. In *15th International Conference on Computational Linguistics and Intelligent Text Processing (CICLing)*, pages 355–366, 2014.

9 Rabeb Mbarek, Mohamed Tmar, and Hawete Hattab. Vector space basis change in information retrieval. *Computación y Sistemas*, 18(3), 2014.

10 Rabeb Mbarek, Mohamed Tmar, Hawete Hattab, and Mohand Boughanem. A re-ranking method based on irrelevant documents in ad-hoc retrieval. In *5th Symposium on Languages, Applications and Technologies (SLATE)*, pages 2:1–2:10, 2016.

11 Jay M. Ponte and W. Bruce Croft. A language modeling approach to information retrieval. In *21st Annual International Conference on Research and Development in Information Retrieval (SIGIR)*, pages 275–281, 1998.

12 Stephen E. Robertson and Steve Walker. Some simple effective approximations to the 2-poisson model for probabilistic weighted retrieval. In *Annual International Conference on Research and Development in Information Retrieval (SIGIR)*, pages 232–241, 1994.

13 Stephen E. Robertson, Steve Walker, Micheline Hancock-Beaulieu, Aarron Gull, and Marianna Lau. Okapi at trec. In *The First Text REtrieval Conference*, pages 21–30, 1992.

14 Joseph Rocchio. Relevance feedback in information retrieval. In Gerard Salton, editor, *Information Storage and Retrieval: Scientific Report No. ISR-9*, chapter 23. The National Science Foundation, 1965.

15 Gerard Salton. *Automatic Information Organization and Retrieval*. McGraw Hill Text, 1968.

16 Gerard Salton and Chris Buckley. Improving retrieval performance by relevance feedback. In Karen Sparck Jones and Peter Willett, editors, *Readings in Information Retrieval*, pages 355–364. Morgan Kaufmann Publishers Inc., 1997.

17 Chengxiang Zhai and John Lafferty. Model-based feedback in the language modeling approach to information retrieval. In *Tenth International Conference on Information and Knowledge Management*, pages 403–410, 2001.