8th Symposium on Languages, Applications and Technologies

SLATE 2019, June 27–28, 2019, Coimbra, Portugal

Edited by

Ricardo Rodrigues

Jan Janoušek

Luís Ferreira

Luísa Coheur

Fernando Batista

Hugo Gonçalo Oliveira

OASICS

Editors

Ricardo Rodrigues ⓘ
CISUC, University of Coimbra, Portugal
Polytechnic Institute of Coimbra, Portugal
rmanuel@dei.uc.pt

Jan Janoušek
Czech Technical University, Prague, Czech Republic
jan.janousek@fit.cvut.cz

Luís Ferreira ⓘ
Instituto Politécnico do Cávado e Ave,
Barcelos, Portugal
lufer@ipca.pt

Luísa Coheur ⓘ
Instituto Superior Técnico, Lisbon, Portugal
INESC–ID, Lisbon, Portugal
luisa.coheur@inesc-id.pt

Fernando Batista ⓘ
ISCTE–IUL, Lisbon, Portugal
INESC–ID, Lisbon, Portugal
fernando.batista@iscte-iul.pt

Hugo Gonçalo Oliveira ⓘ
CISUC, University of Coimbra, Portugal
hroliv@dei.uc.pt

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.

Contents

◼ Preface

SLATE, the Symposium on Languages, Applications and Technologies, is an international scientific event about languages, currently on its 8th edition. This volume compiles the proceedings of this edition, SLATE'19, held at the University of Coimbra, Portugal, on the 27th and 28th of June, 2019.

We often use languages — alas, languages are one of the most defining features of human beings since the dawn of times. For most of mankind's time on earth, languages were used only to communicate between ourselves. Later, however, they came also to be used to communicate with computers, instructing them, and then to receive data and information from them. More recently, with the advent of computer networks, languages for computers to communicate between themselves were also developed.

For each of the previous forms of communication, there are different kinds of languages, but they still share many similarities. In SLATE, we are interested in discussing these kinds of languages. Discussions in this symposium are organized in three main tracks:

- **HHL Track**: Processing Human-Human Languages, dedicated to the presentation and discussion of Natural Language Processing (NLP) tools and applications;
- **HCL Track**: Processing Human-Computer Languages, where researchers, developers, and educators exchange ideas and information on the latest academic or industrial work on design, processing, assessment and applications of programming languages;
- **CCL Track**: Processing Computer-Computer Languages, a broad space for discussing (mark-up) languages for communication between computers, including those used for visualization and presentation of information to the end user.

This volume is all about the use of languages in diverse contexts by humans and computers, and their processing. It compiles 23 papers — 10 long and 13 short — accepted for presentation on the three main tracks of SLATE'19.

In addition to the presentation of the papers, the symposium included two keynote talks: (*i*) *Structured Text and Image Generation — Structured Auto-Regressive Models with Latent Predictor Networks*, by Wang Ling, from DeepMind Technologies Limited; and (*ii*) *Ranking Programming Languages for their Energy Efficiency*, by João Paulo Fernandes, from the Department of Informatics Engineering of the University of Coimbra.

We feel confident that these works provide a valuable, even if brief, insight of what is currently being done in the areas of each track, in particular, and in language technologies, in general, being grateful to each of the authors of the selected papers, and to the keynote speakers for their participation.

As General Chairs of SLATE'19, we would like to thank the people without whom this event would have never been possible, namely: the four track chairs, *Luísa Coheur* and *Fernando Batista* (HHL), *Jan Janoušek* (HCL), and *Luís Ferreira* (CCL), for their valuable help when it came to making decisions of a scientific character; all members of the Scientific Program Committee of each track, for their effort in reviewing the submitted papers; both keynote speakers, *Wang Ling* and *João Fernandes*, for telling us about their current research, impactful in the scope of the symposium; jeKnowledge, a junior company of the University of Coimbra, for helping us in the local organization of the event, with a special mention going to *Miguel Marques*, *Rafaela Antunes*, and *Joana Santos*.

A word of gratitude should be also given to our institutional partners from the University of Coimbra, namely: our research center, the Center for Informatics and Systems of the

University of Coimbra (CISUC), for their logistic support at several levels; the Faculty of Science and Technologies (FCTUC), more precisely the direction board of the Department of Informatics Engineering (DEIUC), for its logistic support at various stages, the direction board of the Department of Mathematics (DMUC), for letting us use their space, and *Gabinete de Apoio à Divulgação* (GAD), for providing us with materials for the participants in the symposium. A special thank you note goes to *Márcia Espírito Santo* (from DEIUC) and to *Jorge Ávila* (from CISUC).

Last but not least, we would like to thank: our sponsors, namely Critical Software, SA, Talkdesk, Inc, and Nestlé Portugal, SA; Dagstuhl Publishing, for the publication of these peer-reviewed proceedings; and EasyChair, whose system was crucial to manage all the Program Committee work, from paper submissions to reviews and notifications.

<div align="right">

Hugo Gonçalo Oliveira
Ricardo Rodrigues
General Chairs

</div>

◼ List of Authors

Alberto Simões
Polytechnic Institute of Cávado and Ave
Barcelos, Portugal
`asimoes@ipca.pt`

Ana Alves
CISUC and Polytechnic Institute of Coimbra
Coimbra, Portugal
`ana@dei.uc.pt`

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

Baris Tekin Tezel
Dokuz Eylul University
İzmir, Turkey
`baris.tezel@deu.edu.tr`

Beatriz Calderón
University of Santiago de Compostela
Santiago de Compostela, Spain
`beatriz.calderon.cerrato@rai.usc.es`

Bostjan Slivnik
University of Ljubljana
Ljubljana, Slovenia
`bostjan.slivnik@fri.uni-lj.si`

Carla Lopes
University of Porto
Porto, Portugal
`ctl@fe.up.pt`

Cátia Dias
INESC–ID and Instituto Superior Técnico
Lisboa, Portugal
`catiadias089@gmail.com`

Christopher Shulby
University of São Paulo
São Paulo, Brazil
`chrisshulby@gmail.com`

Cristiana Araújo
University of Minho
Braga, Portugal
`decristianaaraujo@hotmail.com`

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

Falco Nogatz
University of Würzburg
Würzburg, Germany
`falco.nogatz@uni-wuerzburg.de`

Geylani Kardas
Ege University
İzmir, Turkey
`geylani.kardas@ege.edu.tr`

Gonçalo Carnaz
University of Évora
Évora, Portugal
`goncalojfcarnaz@gmail.com`

Hugo Gonçalo Oliveira
CISUC and University of Coimbra
Coimbra, Portugal
`hroliv@dei.uc.pt`

Jaroslav Porubän
Technical University of Košice
Košice, Slovakia
`jaroslav.poruban@tuke.sk`

João Ferreira
University of Coimbra
Coimbra, Portugal
`jdcoelho@student.dei.uc.pt`

João Lameiras
University of Minho
Braga, Portugal
`pg35398@alunos.uminho.pt`

José Devezas
University of Porto
Porto, Portugal
`jld@fe.up.pt`

José João Almeida
University of Minho
Braga, Portugal
`jj@di.uminho.pt`

José Paulo Leal
University of Porto
Porto, Portugal
`zp@dcc.fc.up.pt`

Julia Kübert
University of Würzburg
Würzburg, Germany
`julia.kuebert@stud-mail.uni-wuerzburg.de`

Luís Martins
University of Minho
Braga, Portugal
`lmfmrt@gmail.com`

Luísa Coheur
INESC–ID and Instituto Superior Técnico
Lisboa, Portugal
`luisa.coheur@l2f.inesc-id.pt`

Maksym Ketsmur
University of Aveiro
Aveiro, Portugal
`mvk@ua.pt`

Maria João Varanda Pereira
Polytechnic Institute of Bragança
Bragança, Portugal
`mjoao@ipb.pt`

Mariana Gaspar Fernandes
INESC–ID and Instituto Superior Técnico
Lisboa, Portugal
`mariana.gaspar.fernandes@tecnico.ulisboa.pt`

Mário Antunes
Polytechnic Institute of Leiria
Leiria, Portugal
`mario.antunes@ipleiria.pt`

Martinho Aragão
University of Minho
Braga, Portugal
`martinhoaragao@gmail.com`

Matej Madeja
Technical University of Košice
Košice, Slovakia
`matej.madeja@tuke.sk`

Mónica Guimarães
Fafe Municipal Archive
Fafe, Portugal
`monikaguimaraes@gmail.com`

Nuno Almeida
University of Aveiro
Aveiro Portugal
`nunoalmeida@ua.pt`

Pablo Gamallo
University of Santiago de Compostela
Santiago de Compostela, Spain
`pablo.gamallo@usc.es`

Patricia Martin-Rodilla
Spanish National Research Council
Santiago de Compostela, Spain
`patricia.martin-rodilla@incipit.csic.es`

Paulo Quaresma
University of Évora
Évora, Portugal
`pq@uevora.pt`

Pedro Cunha
University of Porto
Porto, Portugal
`up201405950@fc.up.pt`

Pedro Fialho
L2F/INESC–ID and University of Évora
Évora, Portugal
`peter.fialho@gmail.com`

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

Ricardo Filipe
Polytechnic Institute of Coimbra
Coimbra, Portugal
`ricardo.ferreira.filipe@gmail.com`

Ricardo Queirós
Polytechnic Inst. of Porto and INESC TEC
Porto, Portugal
`ricardo.queiros@gmail.com`

Ricardo Rodrigues
CISUC and Polytechnic Institute of Coimbra
Coimbra, Portugal
`rmanuel@dei.uc.pt`

Rok Novosel
University of Ljubljana
Ljubljana, Slovenia
`novosel.rok@gmail.com`

Rui Mendes
University of Minho
Braga, Portugal
`azuki@di.uminho.pt`

Salvador Abreu
University of Évora
Évora, Portugal
`spa@uevora.pt`

Samuel Silva
University of Aveiro
Aveiro, Portugal
`sss@ua.pt`

Sérgio Nunes
University of Porto
Porto, Portugal
`ssn@fe.up.pt`

Timur Galeev
Kazan Federal University
Kazan, Respublika Tatarstan, Russia
`tigaleev@kpfu.ru`

Vítor Nogueira
University of Évora
Évora, Portugal
`vbn@di.uevora.pt`

Vladimir Bochkarev
Kazan Federal University
Kazan, Respublika Tatarstan, Russia
`vbochkarev@mail.ru`

Xavier Gómez Guinovart
University of Vigo
Vigo, Spain
`xgg@uvigo.gal`

■ Committees

Conference Chairs

General Chairs

Hugo Gonçalo Oliveira
CISUC and University of Coimbra
Portugal

Ricardo Rodrigues
CISUC and Polytechnic Inst. of Coimbra
Portugal

Track Chairs

Human-Human Languages

Luísa Coheur
IST and INESC–ID
Portugal

Fernando Batista
ISCTE–IUL and INESC–ID
Portugal

Human-Computer Languages

Jan Janoušek
Czech Technical University
Czech Republic

Computer-Computer Languages

Luís Ferreira
Polytechnic Institute of Cávado and Ave
Portugal

Local Organization Committee

Hugo Gonçalo Oliveira
CISUC and University of Coimbra
Portugal

Ricardo Rodrigues
CISUC and Polytechnic Inst. of Coimbra
Portugal

Joana Santos
jeKnowledge and University of Coimbra
Portugal

Miguel Marques
jeKnowledge and University of Coimbra
Portugal

Rafaela Antunes
jeKnowledge and University of Coimbra
Portugal

Scientific Program Committee

Alberto Simões
Polytechnic Institute of Cávado and Ave
Portugal

Alda Lopes Gançarski
Telecom SudParis
France

Alexander Paar
TWT GmbH Science & Innovation
Germany

Alexandre Rademaker
IBM Research
Brasil

Ana Oliveira Alves
CISUC and Polytechnic Institute of Coimbra
Portugal

Antoni Oliver
Open University of Catalonia
Spain

António Cruz
Polytechnic Institute of Viana do Castelo
Portugal

António Menezes Leitão
Technical University of Lisbon
Portugal

António Teixeira
University of Aveiro
Portugal

Arkaitz Zubiaga
University of Warwick
United Kingdom

Partners and Sponsors

UNIVERSIDADE Đ
COIMBRA

jeKnowledge
Nurturing Excellence

CISUC

tdx | powered by
talkdesk

Critical
software

Graph-of-Entity: A Model for Combined Data Representation and Retrieval

José Devezas

INESC TEC, Porto, Portugal
Faculty of Engineering, University of Porto, Portugal
http://josedevezas.com
jld@fe.up.pt

Carla Lopes

INESC TEC, Porto, Portugal
Faculty of Engineering, University of Porto, Portugal
ctl@fe.up.pt

Sérgio Nunes

INESC TEC, Porto, Portugal
Faculty of Engineering, University of Porto, Portugal
ssn@fe.up.pt

──── **Abstract** ────

Managing large volumes of digital documents along with the information they contain, or are associated with, can be challenging. As systems become more intelligent, it increasingly makes sense to power retrieval through all available data, where every lead makes it easier to reach relevant documents or entities. Modern search is heavily powered by structured knowledge, but users still query using keywords or, at the very best, telegraphic natural language. As search becomes increasingly dependent on the integration of text and knowledge, novel approaches for a unified representation of combined data present the opportunity to unlock new ranking strategies. We tackle entity-oriented search using graph-based approaches for representation and retrieval. In particular, we propose the graph-of-entity, a novel approach for indexing combined data, where terms, entities and their relations are jointly represented. We compare the graph-of-entity with the graph-of-word, a text-only model, verifying that, overall, it does not yet achieve a better performance, despite obtaining a higher precision. Our assessment was based on a small subset of the INEX 2009 Wikipedia Collection, created from a sample of 10 topics and respectively judged documents. The offline evaluation we do here is complementary to its counterpart from TREC 2017 OpenSearch track, where, during our participation, we had assessed graph-of-entity in an online setting, through team-draft interleaving.

1   Introduction

As the production of digital documents continues to increase, the answers we are looking for become harder to reach, particularly when relying only on identifiers and linked data to directly reach relevant content. Moreover, using a structured query language is frequently inappropriate for a regular user, who prefers natural language to express their information needs [30]. Full-text search is often the answer, but it inherently discards structure, which is extremely valuable to increase precision. In this work, we attempt to integrate unstructured text and structured knowledge in order to improve retrieval effectiveness in entity-oriented search tasks.

Search has evolved from keyword-based matching. Over time, it has grown increasingly dependent on semantic matching, largely supported on natural language understanding techniques. The need to integrate unstructured text and structured knowledge has substantially increased. In fact, one of the biggest challenges in semantic search is dealing with heterogeneity [15], in particular on the web, where a potentially unlimited number of topics exist. We tackle the problem of heterogeneity in entity-oriented search by proposing a unified graph-based model for terms and entities, where relations are seen as leads to be followed in the investigation of a given information need.

The more accurately a user's information need is identified through query understanding, and the better the information within a document is understood, the more likely the query will be matched with relevant documents or entities mentioned in those documents. This frequently results in improved retrieval effectiveness and, therefore, increased user satisfaction. What about when there is ambiguity? Can we always use entity linking to segment and semantically tag a query, discarding all other segmentations, even those which are equally likely? What if we were unable to provide an adequate answer to the user, even though the information he/she sought was available in the indexed corpus?

In the graph-of-entity, we integrate query entity linking into the ranking process, that is, a given entity in the graph is more relevant if it was reached from a nearby seed node (usually another entity) whose probability of being a good representation of the query is high (i.e., it has a high confidence weight). This probability models the certainty degree of the query entity linking process.

Listing 1 SPARQL query for the shortest path between "Axel A. Weber" and "Solingen" in DBpedia.

```
PREFIX : <http://dbpedia.org/resource/>
SELECT DISTINCT ?s ?o1 ?t
WHERE {
  VALUES ?s { :Axel_A._Weber }
  VALUES ?t { :Solingen }
  ?s [] ?o1 .
  ?o1 [] ?t
}
```

Listing 2 SPARQL query for the shortest path between "Axel Weber (athlete)" and "Solingen" in DBpedia.

```
PREFIX : <http://dbpedia.org/resource/>
SELECT DISTINCT ?s ?o1 ?o2 ?o3 ?o4 ?o5 ?t
WHERE {
  VALUES ?s { :Axel_Weber_(athlete) }
  VALUES ?t { :Solingen }
  ?s [] ?o1 .
  ?o1 [] ?o2 .
  ?o2 [] ?o3 .
  ?o3 [] ?o4 .
  ?o4 [] ?o5 .
  ?o5 [] ?t
}
```

For example, let us assume the ambiguous mention to "Axel Weber", who, according to Wikipedia, can either be the athlete or the economist. Let us now assume that the query also mentions "Solingen", which is the birthplace of "Jens Weidmann", the successor of "Axel A.

Weber", the economist. Now, the probability of "Axel Weber" referring to the economist increases, but there might also be a longer path connecting "Axel Weber", the athlete, to "Solingen". We can easily check this using DBpedia's SPARQL endpoint, by manually testing increasingly longer paths between both "Axel Weber" individuals and "Solingen". Listing 1 shows the SPARQL query for the shortest path between "Axel A. Weber" and "Solingen", which are only linked by one other entity, "Jens Weidmann" – this is consistent with what we have already described. Listing 2 shows the SPARQL query for the shortest path between "Axel Weber (athlete)" and "Solingen", which are linked by five other entities, through two distinct paths – no shorter path would link the two entities. While the query `[ axel weber solingen ]` is more likely to refer to "Axel Weber", the economist, there might still be a niche where users could be searching for "Axel Weber", the athlete, investigating whether there is a relation between the person and the location.

This type of unified approach is more prepared to take advantage of available information, discarding no lead, in order to provide the freedom to search for all matching items. We might say that word or entity disambiguation would happen "organically" during the process of ranking. The hypothesis is that this might improve effectiveness for search queries in the long tail [4], in particular by increasing recall without decreasing precision.

The remainder of this article is organized as follows. In Section 2, we do a literature review about graph-based entity-oriented search, by separately covering entity-oriented search approaches and then graph-based search approaches, closing with a discussion on common ground. In Section 3, we introduce the concept of combined data and present the INEX 2009 Wikipedia Collection. In Section 4, we introduce the technological framework we used, as well as toy example, that we use to describe our implementation of graph-of-word, an existing graph-based representation and retrieval model, as well as our own novel model for combined data, the graph-of-entity. In Section 5, we describe the evaluation approach used to compare the graph-of-word and the graph-of-entity, based on a small subset of the INEX collection. Finally, in Section 6, we present our final remarks and conclusions.

2 Graph-based entity-oriented search

About 80% of queries contain at least one entity [3] and, on average, there are 1.6 entities per sentence (based on the CoNLL 2003 English training set [26]), which makes entity-oriented search a relevant problem within information retrieval. There have been multiple approaches to entity-oriented search where graphs have been used, in particular as a way of representing knowledge bases. A well-known example is Google, who, in 2012, created Knowledge Graph [28], partially powered by Freebase [8], to improve their search engine. In the last few years, there has been work in graph-based approaches for information retrieval [7, 25], and also a growing need for unified models [13, 33, 32]. While many solutions focus on the integration of signals obtained from text represented in an inverted index with signals obtained from external knowledge bases like Wikipedia [2], there have been fewer attempts at modeling text and knowledge in an unified manner, as a single data structure. In this section, we approach the literature about graph-based entity-oriented search by covering two main aspects: entity-oriented search and graph-based search.

Entity-oriented search. Entity-oriented search is a type of semantic search that takes keyword or [telegraphic] natural language queries as input and returns the results that best match the query. Results might include metadata about a particular entity (e.g., an infobox), a list of entities represented by a small subset of relevant metadata (e.g., photo and name), a

direct answer to the user's question, and the traditional documents (e.g., web pages), usually accompanied by a summary. Notable approaches to entity-oriented search include virtual documents [3, 24], learning to rank [20, 31, 10, 9] and the integration of an inverted index and a triplestore [5].

Bautin and Skiena [3] tackled the problem of entity search based on virtual documents called concordances. Each concordance, representing an entity in an inverted index, contained all the sentences mentioning that particular entity in the corpus. The approach does not take advantage of external knowledge and it does not extract information (structured data) from the indexed corpus, both of which could be used to improve performance. The authors claim to be the *first in literature* implementation of an entity search engine. Thus, it is also natural that they used their own evaluation approach, based on the juxtaposition score, instead of available test collections. This makes it difficult to position their work in regards to the state of the art.

Chen et al. [9] presented an empirical study of several learning to rank approaches over common benchmarks, such as the Entity Search Track from SemSearch. The RankSVM [19] model is shown to be the overall best solution and, in particular for SemSearch ES, it achieves the best MAP@100, P@10, P@20 and NDCG@20, when compared with four other models (SDM-CA, MLM-CA, FSDM and Coordinate Accent [34]). Another interesting conclusion presented by the authors is that the correct identification of query types (e.g., keyword queries, long natural language question queries) is important to increase the effectiveness of learned models, in order to boost particular entity fields. Nevertheless, the goal of the graph-of-entity, which we present here, is to make such steps inherently part of the ranking process. The intuition is that the query type is implicitly represented by the structural position of the nodes that best represent the query in a graph. Similarly, instead of query entity linking, we might use the graph to take into account the probability that a given entity is linked to a part of the query, in order to better rank results.

Graph-based search.   In graph-based search, particularly over text corpora, discourse properties can be captured using a graph, either per document or for the whole collection. Such graphs can represent relations between words [6, 7, 25], passages [12], or documents [23, 18], modeling similarities [12], dependencies [25], or even temporal [17] or spatial [22] dimensions.

Blanco and Lioma [6] proposed a method for term weighting based on random walks over a graph of terms, where each term was linked to other terms co-occurring in a window of size n. Similarly to PageRank [23], the weight of a term modeled the probability of jumping from that vertex to another random vertex in the undirected unweighted graph. They were able to obtain a performance comparable to TF-IDF and even outperform it for large window sizes, capturing relations between terms within a distance of 6 to 30.

Rousseau and Vazirgiannis [25] expanded on the work by Blanco and Lioma, presenting an alternative but similar approach for graph-based term weighting. They proposed a directed unweighted graph, the graph-of-word, that similarly captured relations between terms within a window of size n, but this time the target terms were required to follow the source terms instead of simply co-occurring within the window. The authors also used the indegree of the vertex instead of random walks to assign a weight to each term. Their approach significantly outperformed BM25 and, in some cases, even BM25+ [21], a lower-bounded version of BM25. Besides achieving a better performance, another reason to use their graph-based approach is that it does not require any parameter tuning or lower-bounding normalization. The graph-of-word only requires a parameter n for the window size, during indexing, which can be semantically set, since it simply captures a larger context as it grows, at the expense of efficiency.
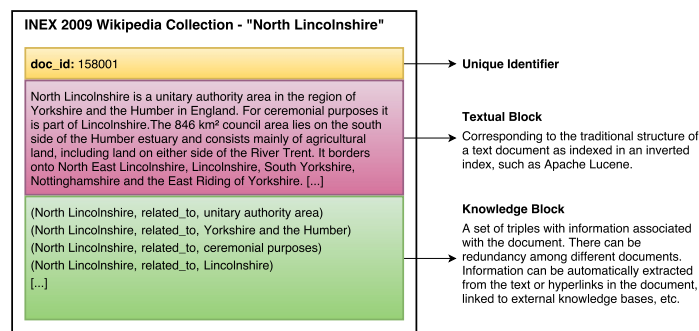
Figure 1 Extended document definition for combined data. Example from INEX 2009 Wikipedia Collection, for the XML representing the Wikipedia article about "North Lincolnshire".

Two aspects combined.   From the surveyed literature, we can make two assumptions about entity-oriented search. First, structured data from knowledge bases, which are inherently representable as graphs, is a fundamental part of the semantic search process. Therefore, knowledge bases must somehow be integrated into the existing frameworks, which are mostly supported by inverted files. Many approaches exist to integrate signals from text and knowledge, but fewer common representation models have been proposed so far. Secondly, graphs have consistently been used to improve text retrieval, even outperforming weighting schemes such as BM25. Graphs can thus be used to represent text and are also frequently used to represent knowledge. It is definitely of value to study how to combine these types of graphs, in order to take advantage of the information locked within unstructured data through the integration of structured data – the knowledge base augments the text, through entities and their relations, and the text augments the knowledge base, providing leads to new information, seamlessly and through a common model (all are nodes in a graph).

What we propose is that the representation model for text and knowledge should be shared, using a graph data structure to capture discourse properties from text, relations between entities from knowledge bases, and term–entity associations based, at the very least, on potentially obvious relations between terms and entities (e.g., through substring matching). The ideal graph-based representation should: (i) capture information complexity, while avoiding redundancy; (ii) facilitate the cross-reference of information from distinct individual sources; (iii) propose a clear representation for combined data (text + knowledge) [2, Definition 2.3] that is easily extensible to other types of media. The open research question is whether or not such a combined data model will, through the unlocking of innovative weighting schemes, improve retrieval effectiveness. In this paper, we propose and evaluate a baseline model, the graph-of-entity, which defines a graph-based representation for combined data, as well as a graph-based weighting scheme that can be used for entity ranking. We compare the graph-of-entity with an implementation of the graph-of-word, in order to position our baseline model within the state of the art.

3   INEX 2009 Wikipedia Collection

In order to assess effectiveness, we take advantage of the INEX 2009 Wikipedia Collection [27], which includes semi-structured data from Wikipedia. The INEX 2009 Wikipedia Collection is an XML collection of Wikipedia articles, which have been annotated with over 5,800 entity classes from the YAGO [29] ontology. It contains over 2.6 million articles and requires 50.7 GB of disk space for storage, when uncompressed. The INEX Ad Hoc Track [16, 1] also

provides 115 topics from the 2009 occurrence, with 50,725 individual relevance judgments, and 107 topics from the 2010 occurrence, with 39,031 individual relevance judgments. Each individual relevance judgment contains the query identifier, the document identifier, the number of relevant characters, the offset of the best entry point (usually the first relevant passage) and offset–length pairs for the relevant passages.

As we built the graph-of-entity for all the collection, each document was represented by an entity node (i.e., the *type* attribute was set to `entity`), containing three main attributes: *doc_id*, *name* and *url*. XPath was used to extract relevant attributes. The *doc_id* was given by `//header/id/text()`, the *name* attribute was given by `//header/title/text()`, and the *url* attribute was built from the entity's Wikipedia page, based on `http://en.wiki-pedia.org/?curid=`<*doc_id*>. Textual content was extracted from `//bdy/descendant-or-self::*[not(ancestor-or-self::template) and not(self::caption)]/text()`. It was then tokenized, storing, for each token, a term node (i.e., the *type* attribute was set to `term`), and creating edges, with a *doc_id* attribute, between pairs of adjacent terms. For each document, links were extracted from the value given by `//link/@xlink:href` and then an entity node was created for each link, with an edge labeled `related_to`, linking the entities from the source and target documents. Further details of the representation model, including the creation of edges between term and entity nodes, will be given in Section 4.2.

Figure 1 illustrates the extracted elements from each XML document, forming what we designate as an extended document for combined data. A regular document usually contains multiple text fields (e.g., *title*, *content*, etc.), which corresponds to the textual block in the extended document. However, we also include a knowledge block, in the form of triples, that are usually available as structured data in the original document. For the INEX collection, the knowledge block can be directly extracted from the XML (we used links to other documents, in order to implicitly build the triples), but in other collections this could be obtained as the result of an information extraction pipeline. There is no restriction about the source of the knowledge block, except that it should represent a set of triples related to the document. For example, the triples might represent co-occurring entities in a sentence or paragraph, or statements obtained from a dependency parser, or they could represent external knowledge about identified entities, from an external knowledge base.

In order to evaluate retrieval over the graph-of-entity, we use the title of each topic as a search query. This is given by `//topic/title/text()` of the `2010-topics.xml` file. We then assess effectiveness based on whether or not retrieved documents contain relevant passages, according to the provided relevance judgments (`inex2010.qrels`). The evaluation process will be further detailed in Section 5.

Smaller subset: INEX 2009 10T-NL. Due to the inability of efficiently indexing the complete INEX 2009 Wikipedia Collection with our graph-based implementations, which were supported by a graph database, we were forced to lower the scale to a smaller subset of the INEX 2009 collection. Accordingly, we prepared a sampling method, based on the topics used for relevance assessment in the INEX 2010 Ad Hoc Track. In order to create the subset, we first selected 10 topics, uniformly at random, from a total of 52 topics with available relevance judgements (out of the 107 topics for 2010). Then, we filtered the relevance judgments, keeping only those regarding the selected topics. Finally, we filtered out documents that were not mentioned in the relevance judgments from each of the four archives (`pages25.tar.bz2`, `pages26.tar.bz2`, `pages27.tar.bz2` and `pages28.tar.bz2`). Our sampling method also provided an option to include all documents linked from the selected documents directly mentioned in the relevance judgments. However, this would result in a much larger dataset than the version without linked documents, defeating the purpose of lowering the scale.

(a) Graph-of-word (document-based graph; text-only). Nodes represent terms. Query term nodes are identified by a thicker border.

(b) Graph-of-entity (collection-based graph; text+knowledge). Pink nodes represent terms and green nodes represent entities. A seed node for the given query is displayed in white. Query term nodes are identified by a thicker border.
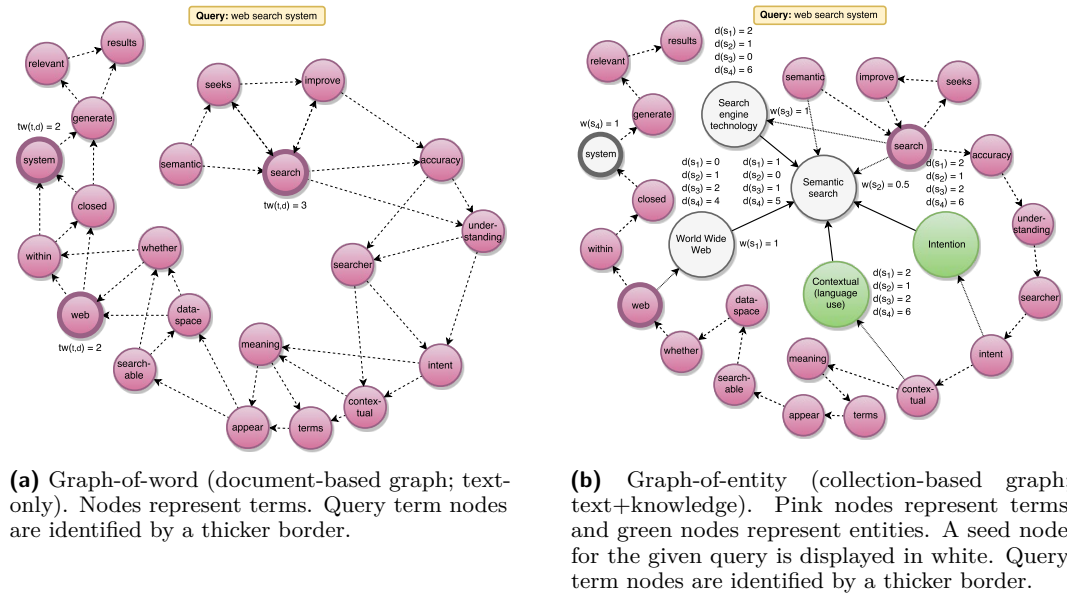
Figure 2 Graph-based representations for the first sentence of Wikipedia's "Semantic Search".

From this moment on, any mention to INEX 2009 data refers to the subset that we created and identify as 10T-NL (10 Topics; No Links). This is the dataset that we use in the evaluation process and, while only 10 topics were selected, the subset contains 7,487 documents and 7,504 individual relevance judgments.

4   Representation and retrieval

In our experimental workbench, we implemented the graph-based models using a graph database per index (Neo4j[1]) and the ranking functions using the Gremlin DSL[2]. The goal of this work was to propose a graph-based representation for combined data (text and knowledge), while using the graph-of-word as a text-only baseline. Figure 2 illustrates the graph-of-word and graph-of-entity models, described in the following sections, based on the first sentence of the Wikipedia article for "Semantic Search" (i.e., our example collection consists of only one document with a single sentence):

> *"Semantic search seeks to improve search accuracy by understanding the searcher's intent and the contextual meaning of terms as they appear in the searchable dataspace, whether on the Web or within a closed system, to generate more relevant results."*

4.1   Graph-of-word

The graph-of-word [25] is a document-based graph [7], where each node represents a term and each edge links to the following terms within a window of size n. The graph is unweighted, but directed, defying the term independence assumption of the bag-of-words approach. Figure 2a

---

[1] `https://neo4j.com/`
[2] Apache Gremlin is a domain-specific language for graph querying. More information at `https://tinkerpop.apache.org/gremlin.html`.

shows a graph-of-word instance for the first sentence of the Wikipedia article on "Semantic Search", using a window size of $n = 3$. The graph-of-word is thus able to capture the context of each term within a particular document.

In the original graph-of-word implementation, the term weight (TW) metric was pre-computed based on the indegree of each term node and stored in the inverted index to be used in place of the term frequency (TF). In our implementation, however, this was done in real time by filtering over the union of all document-based graphs and selecting a given subgraph based on a *doc_id* attribute stored in the edge. This is a less efficient solution, but it simplified the process of exploring and developing the novel graph-of-entity model, based on the graph-of-word, by defining a common representation framework. Additionally, the focus of our experiment was retrieval effectiveness; we were not particularly concerned with index efficiency.

Equation 1 shows the ranking function used for retrieval over the graph-of-word [25, Equation 7].

$$TW\text{-}IDF(t,d) = \frac{tw(t,d)}{1 - b + b \times \frac{|d|}{avdl}} \times log\frac{N + 1}{df(t)} \tag{1}$$

The formula was derived from the TF-IDF approach as defined by Lv and Zhai [21], replacing the $tf(t,d)$ function by the $tw(t,d)$ given by the node indegree of term t, for document d, on the graph-of-word. For example, in Figure 2a, we assume the query `[ web search system ]` and find that the largest term weight, $tw(t,d) = 3$, was assigned to "search", while "web" and "system" were tied in second place with $tw(t,d) = 2$. The parameter b was fixed at 0.003, since, according to the authors [25], it consistently produced good results across various collections, with $|d|$ representing the length of document d, *avdl* the average length of all documents in the corpus, N the number of documents in the corpus, and $df(t)$ the document frequency of term t in the corpus. In our implementation, both $|d|$ and *avdl* were approximated by the number of edges within the respective document-based graph, since we did all computations directly based on the graph.

4.2   Graph-of-entity

The graph-of-entity is a collection-based graph [7], where nodes can represent either terms or entities and edges can be of three types: term – `[before]`→term, entity – `[related_to]`→entity and term – `[contained_in]`→entity. While the graph-of-entity was inspired by the graph-of-word, it only captures term sequence instead of term context, through term→term relations, that is, the window size is always one. Additionally, we also encode entity→entity relations in the graph as a way of representing knowledge associated with the document (e.g., obtained from an information extraction pipeline applied to the text, or simply consisting of Wikipedia concepts linked in some manner). Finally, term→entity relations are established based on a substring matching approach, where a link between a term and an entity is created whenever the term is contained within the entity's name as a whole word (i.e., partial word matches are not considered). The goal for the first version of this model was to keep it simple (e.g., refraining from using similarity edges), but strongly connected (namely capturing all obvious relations). The main goal was to capture the properties of text, while modeling knowledge and establishing relations between text and knowledge.

We rank entities in the graph-of-entity based on the entity weight (EW) for an entity e and a query q. A set of seed nodes S_q are derived from query q, based on the links between query term nodes and entity nodes; when there are no entity nodes linked to a query term node, then the term node becomes its own seed node. This step provides a representation of

the query in the graph, that will be used as the main input for the ranking function. Next, we present a formal definition for $EW(e, q)$, based on three main score components: coverage $c(e, S_q)$, confidence weight $w(s)$ for a seed node s, and the average weighted inverse length of the path between a seed node s and an entity node e to rank.

Let us assume a graph-of-entity represented by an attributed labeled multigraph G_e, similar to the one depicted in Figure 2b, and a set of operations over G_e to obtain a ranking of entity nodes with a *doc_id* attribute. Let q be a query represented by a sequence of term nodes q_n and let e be an entity node that we want to rank (i.e., it has a *doc_id* attribute). Let Sq be the set of seed nodes derived from query q. For each node q_n that represents a term in query q, we obtain the set of seed entity nodes S_{q_n} that are adjacent to term node q_n. Whenever q_n has no entity node neighbors, $S_{q_n} = \{q_n\}$. The set S_q of all seed nodes derived from query q is then given by $S_q = \bigcup_{q_n} S_{q_n}$. This means that S_q will contain all entity nodes adjacent to query term nodes, as well as query term nodes that are not adjacent to any entity node (i.e., they represent themselves). For example, in Figure 2b, assuming query $q = q_1, q_2, q_3$, the seed nodes are given by $S_q = \{e_1, e_2, e_3, q_3\}$. Let p_{es} be a path between an entity node e and a seed node s, as defined by a sequence of vertices $e, v_1, \cdots, v_{(\epsilon-1)}, s$ in the undirected version of G_e. Let P_{es} be the set of all simple paths p_{es} between e and s. Assume the function $\epsilon(p_{uv})$ as the length of a given path p_{uv} between vertices u and v, representing the number of traversed edges[3].

Equation 2 can be read as the ratio between the number of paths linking entity node e and seed nodes s and the total number of seed nodes S_q. That is, the coverage represents the fraction of reachable seed nodes from a given entity.

$$c(e, S_q) = \frac{|\{s \in S_q | \exists p_{es} \in P_{es}\}|}{|S_q|} \tag{2}$$

Let e_{ts} be the edge incident to both a term node t and a seed node s. Equation 3 can be read as the confidence weight of seed node s. It represents the confidence that a seed node is a good representation of the query term it was derived from.

$$w(s) = \begin{cases} \dfrac{|\{e_{ts} \in E(G_e) | \forall t \exists q(t = q_n)|\}}{|\{e_{ts} \in E(G_e)\}|} & \text{if } s \text{ is an entity node} \\ 1 & \text{otherwise} \end{cases} \tag{3}$$

Finally, Equation 4 shows the ranking function for a given entity e and query q.

$$EW(e, q) = c(e, S_q) \times \frac{1}{|S_q|} \sum_{s \in S_q} \left(\frac{1}{|P_{es}|} \sum_{p_{es} \in P_{es}} w(s) \frac{1}{\epsilon(p_{es})} \right) \tag{4}$$

The query is only used to obtain the seed nodes S_q that best represent q in the graph. This is analogous to a step in a query entity linking process. The remaining steps are quite straightforward. We obtain the average weighted inverse length of the path between each seed node s and the entity e that we want to rank. Assuming that the seed nodes are good representations of the query in the graph, the closer an entity is from all seed nodes, the more relevant it is – closeness is measured by the inverse length of the path. Given there is a degree of uncertainty associated with the selection of seed nodes, we scale this value based

---

[3] In practice, we also defined a maximum distance threshold to compute the length of a path between two nodes. That is, no paths above the given threshold were considered. For this particular experiment, we used a maximum distance of one, which is an extremely conservative value.

▨ **Table 1** Evaluation metrics for the graph-of-word (GoW) and graph-of-entity (GoE) based on INEX 2009 10T-NL (precisions and recall were [macro] averaged over all topics).

Model	P@10	MAP	NDCG@10	Prec.	Recall
GoW	**0.3000**	**0.2333**	**0.3265**	0.1085	**0.9816**
GoE	0.1500	0.0399	0.1480	**0.1771**	0.2233

on the confidence weight of the seed node – an entity close to a high confidence seed node is more relevant than an entity close to a low confidence seed node, but an entity further apart from a high confidence seed node might be on par, or even more relevant.

5 Evaluation

During the evaluation stage, we aimed at assessing the retrieval effectiveness of the graph-of-entity in comparison with a slightly altered implementation of the graph-of-word. Particularly, the document length $|d|$ and the average document length $avdl$, used for pivoted document length normalization [14], were calculated based on the number of term nodes per document, which appear only once per document – this means that we were only able to account for unique terms to obtain the document length in the graph-of-word. However, this change is not particularly critical, given the low sensitivity of the graph-of-word to document length [25, Section 5.3] (using $b = 0.003$ is close to using no pivoted document length normalization at all). That is to say, our implementation of the graph-of-word is only slightly different from the original and still provides a solid baseline.

We prepared two indexes based on the 7,487 documents from the INEX 2009 10T-NL collection, one for the graph-of-word and another one for the graph-of-entity. For our experiment, each index was stored as a graph database. We then retrieved the results for each topic, labeling each entry using a binary relevance attribute based on whether there were any identified passages in the judgments file.

Table 1 shows the result of the assessment for this small subset of INEX 2009 Wikipedia Collection. In particular, we present the precision for the first 10 results (P@10), the mean average precision for a maximum of 1,000 retrieved results (MAP), the normalized discounted cumulative gain for the first 10 results, using binary relevance grades (NDCG@10), and the overall precision and recall. As we can see, the graph-of-word (GoW) obtained the best overall scores, except for precision. Recall for the graph-of-word was nearly optimal (0.9816) and significantly above the recall for the graph-of-entity (0.2233). Such a high recall also translated into a lower precision for the graph-of-word (0.1085), which was the only metric that was beat by the graph-of-entity (0.1771). This means that we were unable to improve graph-of-entity (GoE) over the baseline, as expected. Nevertheless, we obtained a better precision, which is encouraging, given our simplistic first attempt at designing a graph-based representation for combined data.

Given the small dimension of the dataset and in order to better understand the obtained MAP scores, in Table 2 we present the average precision for each topic. We also present the issued query and highlight the highest and lowest scores per model. As we can see, `[ dinosaur ]` achieved the highest average precision in graph-of-word, retrieving 703 results (425 relevant), but only 3 results (all relevant) for the graph-of-entity. The lowest average precision for the graph-of-word was achieved for `[ composer museum ]`, retrieving 1,674 results, out of which only 64 were relevant; this was beat by the graph-of-entity, retrieving 179 results, out of which 30 were relevant. The lowest average precision for the graph-of-entity was achieved for `[ Monuments of India ]`, retrieving only 2 results, none of which were relevant.

Table 2 Average precision per topic for the graph-of-word (GoW) and graph-of-entity (GoE) based on INEX 2009 10T-NL. Highest and lowest average precision per model is shown in bold; results are ordered by decreasing average precision for GoW.

Topic ID	Topic Title (Query)	Average Precision	
		GoW	GoE
2010038	[dinosaur]	**0.6189**	0.0069
2010057	[Einstein Relativity theory]	0.2899	**0.1364**
2010003	[Monuments of India]	0.2888	**0.0000**
2010079	[famous chess endgames]	0.2541	0.0448
2010023	[retirement age]	0.2513	0.0027
2010040	[President of the United States]	0.2408	0.0051
2010096	[predictive analysis +logistic +regression model program application]	0.2185	0.0410
2010049	[European fruit trees]	0.0756	0.0119
2010014	[composer museum]	**0.0624**	0.1185
2010032	[japanese ballerina]	0.0331	0.0315
	MAP	0.2333	0.0399

While the graph-of-entity clearly captures additional information, differing mainly on the lack of explicit representation of word context, overall it did not present an improvement over the graph-of-word. Our approach focused on assessing the effectiveness of the model, in order to iteratively improve it and eventually surpass existing state-of-the-art graph-based approaches through the integration of text and knowledge and using a collection-based approach. Despite the disregard for efficiency, at this stage, the complexity of the model and its inefficient implementation supported on a graph database were critical challenges in setting up an evaluation workbench with acceptable run times. While we did not index the full INEX 2009 Wikipedia Collection, with over 2.6 million documents, we were able to index a smaller test collection, based on a sample of 10 topics and corresponding judged documents (INEX 2009 10T-NL), in order to obtain some insight. Additionally, during the participation in the TREC 2017 OpenSearch track [11] we had been able to index the complete SSOAR[4] collection and evaluate the models in a real-world scenario, which acts as complementary information to the performance results we present here.

6   Conclusions

We tackled the problem of entity-oriented search through the proposal of a novel graph-based model for the representation and retrieval of combined data (text and knowledge). We proposed a collection-based representation of terms, entities and their relations (term-term, entity-entity and term-entity), as a way to unify unstructured text and structured knowledge as a graph. We then proposed a very basic ranking function, supported on the graph-of-entity, where we mapped the terms of the query into nodes in the graph, preferentially expanding into neighboring entities, in order to obtain a query representation in the graph (seed nodes). We treated this as an open step in an entity linking process, that was only closed during ranking. Ranking was done based on the seed nodes, by treating them as leads. These leads were followed by trying to exhaust all available paths within a maximum distance, which resulted in the scoring of entity nodes. For evaluation purposes, not all entity nodes were

---

[4] https://www.gesis.org/ssoar/home/

ranked, limiting this operation to nodes that directly represented a document in the corpus (e.g., for Wikipedia, the entity mapped to the corresponding article, while, for SSOAR, a special entity had been created to represent the document). This enabled us to map the problem of entity ranking into the domain of documents, thus providing a way to evaluate using the traditional test collections and strategies that were available to us at the time.

The main goal of this work was to provide a simple baseline model that was graph-based and represented combined data in a unified manner. We performed evaluation based on a sample of the INEX 2009 Wikipedia Collection, which complemented the assessments from TREC 2017 OpenSearch track. In particular, we compared the graph-of-entity (our model) with the graph-of-word (a baseline text-only model). Overall, our model could not outperform the baseline, except regarding precision. However, we were able to establish a graph-based strategy to jointly represent combined data, taking into account terms, entities and their relations in order to perform ranking. At the same time, we explored the unification of entity linking and entity ranking as a single task over the graph-of-entity.

References

1 Paavo Arvola, Shlomo Geva, Jaap Kamps, Ralf Schenkel, Andrew Trotman, and Johanna Vainio. Overview of the INEX 2010 Ad Hoc Track. In *Comparative Evaluation of Focused Retrieval - 9th International Workshop of the Inititative for the Evaluation of XML Retrieval, INEX 2010, Vugh, The Netherlands, December 13-15, 2010, Revised Selected Papers*, pages 1–32, 2010. `doi:10.1007/978-3-642-23577-1_1`.

2 Hannah Bast, Björn Buchhold, Elmar Haussmann, et al. Semantic Search on Text and Knowledge Bases. *Foundations and Trends® in Information Retrieval*, 10(2-3):119–271, 2016.

3 Mikhail Bautin and Steven Skiena. Concordance-Based Entity-Oriented Search. In *The 2007 IEEE / WIC / ACM Conference on Web Intelligence (WI '07)*, pages 2–5, 2007.

4 Michael S. Bernstein, Jaime Teevan, Susan T. Dumais, Daniel J. Liebling, and Eric Horvitz. Direct answers for search queries in the long tail. In *CHI Conference on Human Factors in Computing Systems, CHI '12, Austin, TX, USA - May 05 - 10, 2012*, pages 237–246, 2012. `doi:10.1145/2207676.2207710`.

5 Ravish Bhagdev, Sam Chapman, Fabio Ciravegna, Vitaveska Lanfranchi, and Daniela Petrelli. Hybrid search: Effectively combining keywords and semantic searches. In *European Semantic Web Conference*, pages 554–568. Springer, 2008.

6 Roi Blanco and Christina Lioma. Random walk term weighting for information retrieval. In *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 829–830. ACM, 2007.

7 Roi Blanco and Christina Lioma. Graph-based term weighting for information retrieval. *Information Retrieval*, 15(1):54–92, 2012. `doi:10.1007/s10791-011-9172-x`.

8 Kurt D. Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. Freebase: a collaboratively created graph database for structuring human knowledge. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2008, Vancouver, BC, Canada, June 10-12, 2008*, pages 1247–1250, 2008. `doi:10.1145/1376616.1376746`.

9 Jing Chen, Chenyan Xiong, and Jamie Callan. An Empirical Study of Learning to Rank for Entity Search. In *Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval, SIGIR 2016, Pisa, Italy, July 17-21, 2016*, pages 737–740, 2016. `doi:10.1145/2911451.2914725`.

10 Ruey-cheng Chen, Damiano Spina, W Bruce Croft, Mark Sanderson, and Falk Scholer. Harnessing Semantics for Answer Sentence Retrieval. In *Proceedings of the Eighth Workshop on Exploiting Semantic Annotations in Information Retrieval (ESAIR 2015)*, pages 21–27, 2015.

11   José Devezas, Carla Teixeira Lopes, and Sérgio Nunes. FEUP at TREC 2017 opensearch track: Graph-based models for entity-oriented. In *The Twenty-Sixth Text REtrieval Conference Proceedings (TREC 2017)*, Gaithersburg, MD, USA, 2017.

12   Taoufiq Dkaki, Josiane Mothe, and Quoc Dinh Truong. Passage Retrieval Using Graph Vertices Comparison. In *Third International IEEE Conference on Signal-Image Technologies and Internet-Based System, SITIS 2007, Shanghai, China, December 16-18, 2007*, pages 71–76, 2007. `doi:10.1109/SITIS.2007.82`.

13   Pedro Domingos. *The Master Algorithm: How the Quest for the Ultimate Learning Machine Will Remake Our World*. Basic Books, 2015. URL: `https://books.google.pt/books?id=glUtrgEACAAJ`.

14   Hui Fang, Tao Tao, and ChengXiang Zhai. A formal study of information retrieval heuristics. In *SIGIR 2004: Proceedings of the 27th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, Sheffield, UK, July 25-29, 2004*, pages 49–56, 2004. `doi:10.1145/1008992.1009004`.

15   Miriam Fernández, Iván Cantador, Vanesa López, David Vallet, Pablo Castells, and Enrico Motta. Semantically enhanced information retrieval: An ontology-based approach. *Web semantics: Science, services and agents on the world wide web*, 9(4):434–452, 2011.

16   Shlomo Geva, Jaap Kamps, Miro Lehtonen, Ralf Schenkel, James A. Thom, and Andrew Trotman. Overview of the INEX 2009 Ad Hoc Track. In *Focused Retrieval and Evaluation, 8th International Workshop of the Initiative for the Evaluation of XML Retrieval, INEX 2009, Brisbane, Australia, December 7-9, 2009, Revised and Selected Papers*, pages 4–25, 2009. `doi:10.1007/978-3-642-14556-8_4`.

17   Udayan Khurana and Amol Deshpande. Efficient snapshot retrieval over historical graph data. In *29th IEEE International Conference on Data Engineering, ICDE 2013, Brisbane, Australia, April 8-12, 2013*, pages 997–1008, 2013. `doi:10.1109/ICDE.2013.6544892`.

18   Jon M. Kleinberg. Authoritative Sources in a Hyperlinked Environment. *J. ACM*, 46(5):604–632, 1999. `doi:10.1145/324133.324140`.

19   Ching-Pei Lee and Chih-Jen Lin. Large-Scale Linear RankSVM. *Neural Computation*, 26(4):781–817, 2014. `doi:10.1162/NECO_a_00571`.

20   Bo Lin, Kevin Dela Rosa, Rushin Shah, and Nitin Agarwal. LADS : Rapid Development of a Learning-To-Rank Based Related Entity Finding System using Open Advancement. In *The First International Workshop on Entity-Oriented Search (EOS 2011)*, 2011.

21   Yuanhua Lv and ChengXiang Zhai. Lower-bounding term frequency normalization. In *Proceedings of the 20th ACM Conference on Information and Knowledge Management, CIKM 2011, Glasgow, United Kingdom, October 24-28, 2011*, pages 7–16, 2011. `doi:10.1145/2063576.2063584`.

22   Bruno Martins and Mário J. Silva. A Graph-Ranking Algorithm for Geo-Referencing Documents. In *Proceedings of the 5th IEEE International Conference on Data Mining (ICDM 2005), 27-30 November 2005, Houston, Texas, USA*, pages 741–744, 2005. `doi:10.1109/ICDM.2005.6`.

23   Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The PageRank citation ranking: Bringing order to the web. Technical report, Stanford InfoLab, 1999.

24   Hadas Raviv, David Carmel, and Oren Kurland. A ranking framework for entity oriented search using Markov random fields. In *Proceedings of the 1st Joint International Workshop on Entity-Oriented and Semantic Search (JIWES 2012)*, pages 1–6, 2012. `doi:10.1145/2379307.2379308`.

25   François Rousseau and Michalis Vazirgiannis. Graph-of-word and TW-IDF: new approach to ad hoc IR. In *Proceedings of the 22nd ACM International Conference on Information & Knowledge Management*, pages 59–68. ACM, 2013.

26   Erik F. Tjong Kim Sang and Fien De Meulder. Introduction to the CoNLL-2003 Shared Task: Language-Independent Named Entity Recognition. In *Proceedings of the Seventh Conference on Natural Language Learning, CoNLL 2003, Held in cooperation with HLT-*

NAACL 2003, Edmonton, Canada, May 31 - June 1, 2003, pages 142–147, 2003. URL: `http://aclweb.org/anthology/W/W03/W03-0419.pdf`.

27 Ralf Schenkel, Fabian M. Suchanek, and Gjergji Kasneci. YAWN: A semantically annotated wikipedia XML corpus. In *Datenbanksysteme in Business, Technologie und Web (BTW 2007), 12. Fachtagung des GI-Fachbereichs "Datenbanken und Informationssysteme" (DBIS), Proceedings, 7.-9. März 2007, Aachen, Germany*, pages 277–291, 2007. URL: `http://subs.emis.de/LNI/Proceedings/Proceedings103/article1404.html`.

28 Amit Singhal. Introducing the Knowledge Graph: things, not strings. `https://googleblog.blogspot.pt/2012/05/introducing-knowledge-graph-things-not.html`, May 2012.

29 Fabian M. Suchanek, Gjergji Kasneci, and Gerhard Weikum. Yago: a core of semantic knowledge. In *Proceedings of the 16th International Conference on World Wide Web, WWW 2007, Banff, Alberta, Canada, May 8-12, 2007*, pages 697–706, 2007. `doi:10.1145/1242572.1242667`.

30 Valentin Tablan, Danica Damljanovic, and Kalina Bontcheva. A Natural Language Query Interface to Structured Information. In *The Semantic Web: Research and Applications, 5th European Semantic Web Conference, ESWC 2008, Tenerife, Canary Islands, Spain, June 1-5, 2008, Proceedings*, pages 361–375, 2008. `doi:10.1007/978-3-540-68234-9_28`.

31 Alberto Tonon, Michele Catasta, Gianluca Demartini, Philippe Cudr, and Karl Aberer. TRank: Ranking Entity Types Using the Web of Data. In *International Symposium on Wearable Computers 2013 (ISWC 2013)*, 2013. URL: `http://infoscience.epfl.ch/record/196256/files/TRank.pdf`.

32 Chenyan Xiong, Jamie Callan, and Tie-Yan Liu. Word-Entity Duet Representations for Document Ranking. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval, Shinjuku, Tokyo, Japan, August 7-11, 2017*, pages 763–772, 2017. `doi:10.1145/3077136.3080768`.

33 Chenyan Xiong, Zhengzhong Liu, Jamie Callan, and Ed Hovy. JointSem: Combining query entity linking and entity based document ranking. In *Proceedings of the 26th ACM International Conference on Information and Knowledge Management (CIKM 2017)*, 2017.

34 Nikita Zhiltsov, Alexander Kotov, and Fedor Nikolaev. Fielded sequential dependence model for ad-hoc entity retrieval in the web of data. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 253–262. ACM, 2015.

Using Lucene for Developing a
Question-Answering Agent in Portuguese

Hugo Gonçalo Oliveira 🆔
CISUC, Department of Informatics Engineering, University of Coimbra, Portugal
`https://eden.dei.uc.pt/~hroliv/`
hroliv@dei.uc.pt

Ricardo Filipe
ISEC, Polytechnic Institute of Coimbra, Portugal
ricardo.ferreira.filipe@gmail.com

Ricardo Rodrigues 🆔
CISUC, University of Coimbra, Portugal
ESEC, Polytechnic Institute of Coimbra, Portugal
rmanuel@dei.uc.pt

Ana Alves 🆔
CISUC, University of Coimbra, Portugal
ISEC, Polytechnic Institute of Coimbra, Portugal
ana@dei.uc.pt

--- **Abstract** ---

Given the limitations of available platforms for creating conversational agents, and that a question-answering agent suffices in many scenarios, we take advantage of the Information Retrieval library Lucene for developing such an agent for Portuguese. The solution described answers natural language questions based on an indexed list of FAQs. Its adaptation to different domains is a matter of changing the underlying list. Different configurations of this solution, mostly on the language analysis level, resulted in different search strategies, which were tested for answering questions about the economic activity in Portugal. In addition to comparing the different search strategies, we concluded that, towards better answers, it is fruitful to combine the results of different strategies with a voting method.

1 Introduction

A natural way of interacting with computational systems is by communicating with them in the same way we communicate with other humans: using natural language. This is indeed one of the long-term goals of Natural Language Processing (NLP), currently materialised in Intelligent Personal Assistants, like Apple's Siri or Amazon's Alexa, which accept commands in natural language, and are running on our smartphones, smartwatches or smart homes. In fact, since ELIZA [22], chatbots and dialog systems have become more human-like, able to engage in informal conversations and to learn from user input.

This shift is so present that most organisations are adopting one or more assistants of this kind as alternative channels to interact with their customers. This high demand lead

to the development of several platforms for easing the creation of conversational agents [3], through a high-level interface, which reduces much of the time that would, otherwise, be spent on the process. Yet, their pipeline is often not so much customisable on the lower-level pre-processing, especially for non-English languages. Furthermore, some of those platforms are proprietary and have usage restrictions. This means that, from the beginning, solutions based on them are tied to the underlying pre-processing and the so-called Natural Language Understanding (NLU) mechanisms. Among other drawbacks, this prevents their experimentation with different techniques or language resources.

Still, in many scenarios, dialog capabilities are not necessary, and an improved search mechanism, possibly deployed as a question-answering (QA) agent, is enough. Such an agent would have the main goal of: (1) processing user input; (2) matching it with questions from its knowledge-base; (3) providing adequate answers. A knowledge-base for such an agent should cover questions about the target organisation and the services it provides, with a focus on those frequently asked by customers. This suggests that such a knowledge base could be made of frequently asked questions (FAQs).

This paper describes how a QA agent can be developed on top of Lucene, an open-source solution for Information Retrieval (IR), which provides high-performance full text indexing and searching, while offering a high level of customisation, namely concerning the applied pre-processing, indexed information and search metrics. Since our main domain of application is on Portuguese text, the previous reasons lead us to tune the agent for Portuguese, with the integration of specific pre-processing for this language. The developed system indexes a list of FAQs and tries to match user input, written in natural language, with the available questions. Once a question is matched, its answer is retrieved. Furthermore, as different metrics can be applied for matching, the developed agent enables the integration of different strategies for this purpose, considering different search fields, analysis or levels of tolerance. Ultimately, all the available metrics can be used in parallel, towards a better decision on the best candidate answer.

In the remainder of this paper, we provide a brief overview on the background scientific areas that support this work, covering search technologies, chatbots and dialog systems, and also their intersection. After that, we describe the architecture of our QA agent, including details on tuning Lucene for our purposes, tools and resources exploited, and also on the implemented search strategies. We admittedly tried to balance the previous description between the scientific contributions and implementation details, having in mind future applications of this agent, possibly by other researchers, but also those only interested in tuning Lucene for Portuguese. Before concluding, we report on the utilisation of the QA agent in a specific scenario. For demonstration purposes, it was tested with a list of FAQs obtained from the Portuguese Entrepreneur's Desk (in Portuguese, "*Balcão do Empreendedor*") and different search strategies were used for answering variations of the questions in that list. Besides comparing the implemented search strategies, we concluded that answer accuracy is higher when results of more than one strategy are combined.

2 Background and Related Work

Since ELIZA, chatbots and dialog systems have become more human-like, capable of engaging in informal conversations (see, e.g., Mitsuku[1] or Rose[2]), and of learning from human input (see,

---

[1] `https://www.pandorabots.com/mitsuku/`
[2] `http://brilligunderstanding.com/rosedemo.html`

e.g., Cleverbot[3]). Common approaches for developing such a system exploit large collections of text, often including conversations. Generative systems try to model conversations with a neural network that learns to decode a sequence of text and translate it to another sequence, used as a response [20]. They are generally scalable, versatile, and always generate a response, but have limitations when it comes to performing specific tasks. They make few assumptions about the domain and generally have no access to external sources of knowledge, which means that they can rarely handle factual content. They also tend to be too repetitive and provide inconsistent, trivial or meaningless answers.

Task-oriented agents tend to follow other strategies and integrate Information Retrieval (IR) and Question Answering (QA) techniques, in order to find the most relevant response to requests in natural language. The long-established task of IR has the goal of finding information automatically in a collection of documents, typically a large one. The input of a traditional IR system is a query that represents an information need, typically in the form of keywords, to be answered with a list of documents. There are countless models for retrieving relevant documents for a query and for ranking them.

Yet, a traditional IR system does not interpret the meaning of the query. Relevant documents are generally selected because they mention the keywords, possibly their stems, or are about the topics they convey. Diversely, automatic QA [9, 21] has the main goal of finding answers to questions formulated in natural language. Answers can be retrieved from a knowledge base [16] or from a collection of documents [12]. This has similarities to IR, especially the latter [12], but queries have to be further interpreted, possibly reasoned – this is where NLU capabilities may be necessary – , while answers are expected to go beyond just a list of documents.

Given a user input, IR-based conversational agents search for the most similar request on the corpus and output its response (see, e.g., [10]). They rely on an IR system for indexing the documents of the corpus and, in order to identify similar texts and computing their relevance, they apply IR ranking techniques, often based on the vector space model and on the cosine between the request vector and vectors of indexed requests or possible responses. But those measures can also be combined with an alternative ranking function learned specifically for that purpose. This can be achieved, for instance, with a regression model that considers several lexical or semantic features to measure semantic textual similarity [5].

As opposed to the generative approach, IR-based conversational agents do not handle very well requests for which there is no similar text in the corpus. Nevertheless, an alternative IR-based strategy can still be followed, in this case, for finding similar texts in a more general corpus, such as movie subtitles [14]. In order to minimise the limitations of generative and IR-based approaches for chatbots, some authors tried to combine them. For instance, IR techniques have been used for reordering a set of generated answers [19].

The high demand for chatbots and virtual assistants lead to the development of several platforms [3] – for instance, DialogFlow[4], Wit.ai[5], Luis[6], Watson Assistant[7] – for easing the creation of such systems, through high-level interfaces. This turns out to be an easy solution or creating conversational agents out-of-the-box and reduces much of the time that would, otherwise, be spent on this process. However, not all of such platforms are completely free – some are proprietary and their utilisation may be dependent on, or restricted according to, a paid license – and most are not as flexible as a NLP researcher would wish, especially when

---

[3] `https://www.cleverbot.com/`

[4] `https://dialogflow.com/`

[5] `https://wit.ai/`

[6] `https://luis.ai/home`

[7] `https://www.ibm.com/cloud/watson-assistant/`

it comes to non-English languages.

Those platforms are usually based on the concepts of intent – i.e., purpose of the users input – and entity – i.e., terms that are relevant for the intent [3]. For instance, in the question *What is the weather like in Coimbra?*, the intent would be to know weather information, while Coimbra is the target entity. We analysed some platforms and made some experiments with DialogFlow. Yet, version 1 of its SDK did no enable intent management. This had to be done through the web interface in the platform website. Version 2 beta has had some improvements, but, still, does not enable to control the NLU techniques applied, for example, how user input is matched with intents. This means that it does not enable, for instance, the experimentation of different models for this purpose – e.g., using language-specific tools – nor the integration of a lexical resource – e.g., for handling synonyms or related words.

In some scenarios, dialog capabilities are not that crucial, and a QA agent, capable of matching natural language requests with known questions is enough. In this case, when available, lists of frequently asked questions, typically abbreviated to FAQs, are valuable resources for exploitation, due to their nature and structure. In fact, QA systems have exploited FAQs on the web for open-domain QA [11]; in the last decade, there has been interest on SMS-based interfaces for FAQs [13]; and there has also been a shared task on QA from FAQs in Italian [4]. The previous QA system relied on Lucene for indexing and as a baseline for retrieving documents [4]. Furthermore, FAQ-based QA agents often pre-process text in questions, answers and user requests, applying tokenisation and stopword removal operations. When matching user requests with FAQs, they exploit word overlap or the presence of similar words. In some cases, synonyms [13, 15], acronyms [7] or distributional semantic features [7] are also considered.

3   System Architecture

As stated earlier, this paper is about the development of a flexible QA agent for Portuguese, based on a list of FAQs that might be changed, depending on the agent's purpose. As other IR-based QA [11, 13] or conversational agents [14, 5], it was built on top of Apache Lucene[8], an open-source Java library for IR, which provides high performance full-text indexing and search capabilities. Specifically, we used version 7.7.0 of this library.

Lucene enables the representation of textual documents in a set of fields, some containing single values and other longer texts. Not all fields have to be indexed, specifically those that are not considered in searches. In its basic use case, Lucene would represent each document by a text field with the textual contents of the document. This enables full-text search on the collection of documents. Additional fields may be useful for adding metadata, such as the document location, and they can also be used for storing and indexing the result of the text after some pre-processing operations.

The diagram in figure 1 shows the question-answering flow in our agent. Textual input by the user is interpreted as a question and then analysed, to be prepared for the retrieval phase. The analysis made here should be equivalent to the one made in the creation of the index (see section 3.1). It may cover pre-processing tasks such as tokenisation or synonym expansion. After the analysis, the input is matched with a suitable question on the knowledge base. In fact, Lucene will retrieve a list of candidate documents (questions), ranked according to their relevance. The agent will retrieve not only the top-n questions, but also their answers, which are then shown to the user. Hopefully, one of them will be the right answer.

---

[8] `https://lucene.apache.org/`
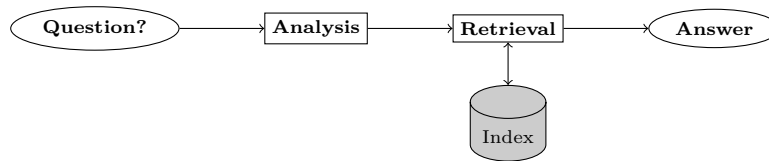
■ **Figure 1** Question-answering flow.

The remainder of this section details how Lucene is adjusted in the implementation of the aforementioned flow. It starts with the creation of the index, moves on to a brief description of external tools and resources exploited in the analysis, and finally enumerates some of the strategies applied for searching on the index.

3.1 Index creation

The starting point for creating the underlying index for the QA agent is a list of questions and their answers, both written in natural language. This can be a list of FAQs, such as those commonly found in institutional websites, which include answers to questions frequently asked about the institution, in order to help (potential) customers finding out more about it, as fast as possible, and, at the same time, avoiding a congestion of contacts through other channels, such as e-mail or telephone.

The QA agent may work with any list of FAQs, as long as, before indexing, this list is organised in a text file with questions and answers interleaved, in lines respectively starting with `P:` and `R:`. This means that, changing the FAQs and creating a new index is all it takes for adapting the agent to a different domain. Due to our goal and performed adaptations, besides the input format, the only restriction is that FAQs are written in Portuguese.

When creating an index, the text in the FAQs file is analysed by Lucene. This is performed by an instance of the Analyzer class of Lucene's API, which has the main purpose of tokenising the text. Lucene offers several analysers out-of-the-box, including StandardAnalyzer and PortugueseAnalyzer. The former applies standard tokenisation rules, converts text to lower case and ignores English stopwords. PortugueseAnalyzer, available under the package `org.apache.lucene.analysis.pt`, extends StandardAnalyzer but, as its name suggests, considers some specificities of Portuguese: it uses a list of Portuguese stopwords instead, borrowed from Snowball[9], and applies rules for stemming Portuguese words and stripping off accents, with PortugueseLightStemmer.

We implemented search strategies using each of the previous analysers, but, for higher control, we also reimplemented the Analyzer class and created CustomPortugueseAnalyzer. This analyzer has the option of stemming or not, and of considering a map for words and their possible synonyms. The latter is easily integrated in the Analyzer with an instance of the SynonymMap class, under the package `org.apache.lucene.analysis.synonym`. Search recall should benefit from this map, as it helps dealing with language variation. Producing lemmatised versions of each question-answer pair and indexing them in specific fields should also have a positive impact on handling language variation. Lemmatisation resorts to an external tool, further introduced in section 3.2.

Figure 2 illustrates the indexing procedure, to be run for each list of FAQs. Once the index is ready, it will only have to be created again if there are changes on the data (e.g., a new question is added) or a different analysis is required. In the index, each question is

---

[9] `http://snowball.tartarus.org/algorithms/portuguese/stop.txt`

represented as illustrated in table 1, namely with four fields: original question (P), lemmatised question (PL), answer (R), and lemmatised answer (RL). All of these fields are searchable, though no implemented search strategy considers the four of them.
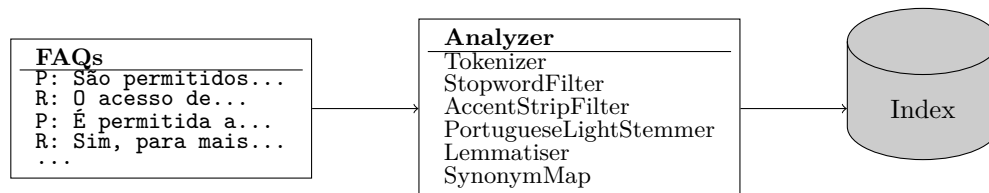
■ Figure 2 Indexing the list of FAQs.

■ Table 1 Document fields in Lucene and their contents for one question.

Field	Content
P	São permitidos animais em estabelecimentos de restauração ou bebidas?
	(Are pets allowed in restaurants or bars?)
PL	Ser permitido animal em estabelecimento de restauração ou bebida?
	(Be pet allow in restaurant or bar?)
R	O acesso de animais é permitido apenas às esplanadas, exceto cães de assistência que podem aceder a toda a área frequentada pelos clientes...
	(Animal access is allowed only to the terraces, except for service dogs that can access the entire area used by customers.)
RL	O acesso de animal ser permitido apenas à esplanada, exceto cão de assistência que pode aceder a toda a área frequentada pelo cliente...
	(Animal access be allow only to the terrace, except for service dog that can access the entire area use by customer.)

3.2 External Tools & Resources

As mentioned in the previous section, PortugueseAnalyzer has several useful filters for removing Portuguese stopwords, stripping off accents and stemming. Yet, we created a specific instance of the Analyzer class, which includes some of the previous filters but is further augmented, to be used by some search strategies. First, questions and answers were lemmatised as an alternative to stemming. Although both stemming and lemmatisation are normalisation alternatives, they are applied at different stages. Stemming is applied by Lucene's Analyzer as part of the indexing and retrieval process, while the lemmatised versions of the question and of the answer are stored and indexed in specific fields and applied to every query, when these fields are considered. In opposition to stemming, lemmatisation always results in a valid word (a lexeme), generally the dictionary entry for the original word. For this purpose, lemmatisation depends on a previous morphology analysis and thus requires more language-specific knowledge. Contrarily, stemming simply cuts off the end of the words, and sometimes also the prefixes, which is not always enough to match words in different forms. Consider, for instance, the words *question*, *questions*, *mouse*, *mice*, *is*, and *were*. Possible stems are *quest*, *quest*, *mous*, *mic*, *is*, and *wer*, while their lemmas are *question*, *question*, *mouse*, *mouse*, *be*, and *be*.

Lemmatisation was performed with LEMPORT, part of the NLPPORT [18] toolkit. To enable lemmatisation, both question and answer had to be first tokenised and part-of-speech tagged with tools that are also part of the previous toolkit, namely TOKPORT and TAGPORT.

The second adaptation took advantage of Lucene's SynonymMap class, which, although easy to integrate in an Analyzer, was not available for Portuguese. As its name suggests, an instance of a SynonymMap maps words with their possible synonyms. For our agent, the source of this map was a lexical knowledge base extracted from ten different Portuguese lexical resources [8], freely available online[10]. This knowledge base is represented in a text file of which we show some lines in figure 3. Each line contains relations of different types, between two words, each followed by a number that represents the number of resources where the relation was found. Since this number is related to reliability and acceptability of the semantic relation, we decided to use only the 3,483 synonym pairs with a number of resources higher than five – i.e., the pair occurs in at least six resources. Relations with a lower number or of other types were not used.

Figure 4 shows the map resulting from the relations in figure 3. Depending on the relations in the list, a word can be mapped to one or more synonyms.

```
multa SINONIMO_N_DE coima 6
procedimento SINONIMO_N_DE comportamento 6
permitir SINONIMO_V_DE admitir 6
permitir SINONIMO_V_DE deixar 6
autorizar SINONIMO_V_DE permitir 6
consentir SINONIMO_V_DE permitir 7
permitir SINONIMO_V_DE conceder 7
```

Figure 3 Selected lines of the lexical knowledge base file.

```
        multa  →  coima
        coima  →  multa
      conduta  →  procedimento
 procedimento  →  conduta, comportamento
comportamento  →  procedimento
     permitir  →  admitir, deixar, autorizar, consentir, conceder
      admitir  →  permitir
       deixar  →  permitir
    autorizar  →  permitir
    consentir  →  permitir
     conceder  →  permitir
```

Figure 4 Synonym Map resulting from the synonym pairs in Figure 3.

3.3 Search strategies

Even with the performed analysis and the resulting index, some issues arise about the best way to exploit all the strategies towards the best results for a user query. For instance, in some cases, matching the input text with the original questions might be more fruitful than lemmatising both; answers often contain relevant information for this process; not to mention that the input text might contain typos or spelling mistakes, so tolerance is sometimes necessary. In order to cover all the aforementioned issues, different search strategies can be implemented, considering the contents of different fields, applying fuzzy instead of exact search, or using different similarity metrics. We ended up not exploring the latter. All the implemented search strategies rely on the BM25 similarity [17], a probabilistic model for IR, currently the default ranking method of Lucene, and an alternative to the classic TF-IDF.

Alternatively, we implemented strategies that match the input with different indexed fields (Question, Question and Answer, or their lemmatised versions), and based on different analysers (Standard, Portuguese, Custom), considering synonyms or not. Table 2 lists all

---

[10] http://ontopt.dei.uc.pt/index.php?sec=download_outros

the implemented search strategies, together with their configuration. It should be noted that some of the analysis made to the same fields is incompatible (e.g., StandardAnalyzer vs PortugueseAnalyzer, considering synonyms vs not). To cope with this situation, different indexes are created and not all search strategies share the same index.

Table 2 Search strategies and their configuration.

Strategy	Fields	Analyzer	Normalisation	Synonyms
VanillaQuestion	P	StandardAnalyzer	None	No
VanillaQuestionAnswer	P, R	StandardAnalyzer	None	No
StemQuestion	P	PortugueseAnalyzer	Stemming	No
StemQuestionAnswer	P, R	PortugueseAnalyzer	Stemming	No
StemSynsQuestion	P	CustomPTAnalyzer	Stemming	Yes
StemSynsQuestionAnswer	P, R	CustomPTAnalyzer	Stemming	Yes
LemmaQuestion	PL	CustomPTAnalyzer	Lemmatisation	No
LemmaQuestionAnswer	PL, RL	CustomPTAnalyzer	Lemmatisation	No
LemmaSynsQuestion	PL	CustomPTAnalyzer	Lemmatisation	Yes
LemmaSynsQuestionAnswer	PL, RL	CustomPTAnalyzer	Lemmatisation	Yes

We also created a fuzzy version of each strategy, taking advantage of Lucene's fuzzy search feature, which allows searching for words with an edit distance of at most two characters. This also increases tolerance to misspelled words or other typos. Implementing the fuzzy versions was a matter of adding the ~ (tilde) search operator to the end of each term in the query.

4   Case study

In order to test the QA agent in a real scenario, it was used for indexing a list of FAQs in Portuguese, thus allowing user queries that would be matched against the available questions, for which the answers would then be presented. This instantiation of the QA agent confirmed that adapting the system to a domain is just a matter of providing a list of FAQs in the desired format, while a list of acronyms can optionally be exploited. This section is about experiments and results in the previous domain. It starts by describing the list of FAQs, the list of acronyms, and a manually created evaluation dataset. After that, results on the previous dataset are presented and discussed for different search strategies.

4.1   Domain description

The FAQs used for this instantiation of the QA agent were collected from the "*Balcão do Empreendedor* (BDE)" portal, the Portuguese Entrepreneur's Desk[11], which is a single point of access to digital services related to the exercise of economic activity in Portugal. BDE is directed to entrepreneurs who wish to perform services and obtain information inherent to the economic activities that they practice. Specifically, the list used for this purpose included 120 FAQs from the Guide for the Application of the RJACSR ("*Guia de Aplicação do Regime Jurídico de Acesso e Exercício de Atividades de Comércio, Serviços e Restauração*") and 56 from the Legislation of the Local Accomodation ("*Legislação do Alojamento Local*")[12], thus totalling 176 FAQs.

---

[11] `https://bde.portaldocidadao.pt/evo/balcaodoempreendedor.aspx` (retrieved on June 2018)

[12] Both of these documents were downloaded from BDE on June 2018.

In order to increase recall, while indexing the aforementioned FAQs, Lucene's SynonymMap, used by four of the search strategies, was enriched with 38 acronyms and their full meaning. The aforementioned list was provided by the Portuguese Administrative Modernisation Agency (AMA) and included acronyms commonly used in the BDE, such as RJACSR, AL or MB, respectively for "*Balcão do Empreendedor*", "*Regime Jurídico das Atividades de Comércio, Serviços e Restauração*" (Legal Regime for Trade, Services and Catering Activities), "*Alojamento Local*" (Local Accommodation), or "*Multibanco*" (ATM).

These are the only adaptations required for using the developed QA agent on a specific domain. In fact, the list of acronyms is optional and not used by the majority of the search strategies implemented.

4.2 Evaluation dataset

In order to test how well the agent was doing its job, a more systematic evaluation was designed. Bearing in mind that, in most cases, users will not search for the exact question, variations of the original questions were also created and then used for assessing the system. Those variations included paraphrases or closely-related questions, including some in which words were on a different case or accents were missing. They were produced manually by three native Portuguese-speaking volunteers, who also assigned them to a correct answer in the list of FAQs. The resulting list had a total of 447 different questions, which means that, on average, it had ≈ 2.5 ways of asking each original question. It should be noted that not all of the original questions had the same number of variations produced for. Table 3 shows examples of the variations produced, in lines starting with "*P". Lines starting with a "P" and an "R" mark the original question and its answer, respectively.

4.3 Performance of search strategies

Each implemented search strategy was used to answer each question in the evaluation dataset. In this case, we considered that the answer would be the highest ranked search result, corresponding to one of the FAQs and its answer. Once an answer was selected, we checked whether it was the same of the original question (correct) or not (incorrect). In the end, we computed the accuracy of the QA agent, given by the ratio of correct answers to all question variations on the dataset (excluding the original questions).

Table 4 shows the results obtained for each search strategy. Its figures confirm that, due to language variation, matching questions in natural language is a challenging task. The best strategy, with 70% of the questions answered correctly, is the fuzzy version of VanillaQuestionAnswer. This is surprising, because this strategy relies on Lucene's StandardAnalyzer, which makes minimal pre-processing and has no knowledge specific of the Portuguese language. On the other hand, this is a fuzzy search, where variations of two characters per word in the query are accepted. This suggests that adding tolerance with fuzzy search might have a similar or even more positive effect than ignoring stopwords and normalising text with stemming or lemmatisation operations. Yet, fuzzy search only improves accuracy when it is applied to the vanilla search strategies. For strategies that include text normalisation, fuzzy searches always lead to accuracies lower than the normal search.

Accuracy is always better when the agent searches on both question and answer, which suggests that both should be exploited in this process. Finally, considering synonyms (and acronyms) increases the accuracy, but only when lemmas are used. This makes sense because the synonym pairs are established between lemmas, not inflected words nor stems.

Table 3 Original questions, answers, and manually created variations, for testing purposes.

P	*São permitidos animais em estabelecimentos de restauração ou bebidas?*
*P	*podem entrar animais num estabelecimento abrangido pelo rjacsr*
*P	*Que animais podem entrar num bar ou restaurante?*
R	*O acesso de animais é permitido apenas às esplanadas, exceto cães de assistência que podem aceder a toda a área frequentada pelos clientes...*
P	*Qual a coima aplicável às contraordenações graves?*
*P	*coima para contraordenação grave*
*P	*sanção das infrações graves*
*P	*Qual o valor da multa para contraordenações graves?*
R	*As contraordenações graves são sancionáveis com coima: a) Tratando-se de pessoa singular, de € 1 200,00 a € 3 000,00;b) Tratando-se de microempresa, ...*
P	*Se a exploração do meu apartamento passar para outra pessoa, tenho de fazer um novo registo?*
*P	*o meu imovel de alojamento local passou para outra pessoa, o que preciso de fazer*
*P	*É necessário renovar o registo de um apartamento de que deixei de explorar?*
R	*Não, mantendo-se o mesmo estabelecimento de alojamento local, apenas é necessário efetuar uma alteração ao respetivo registo, através do balcão único eletrónico, ...*
P	*No alojamento local é obrigatória a certificação energética? Em que termos deve ser efetuada?*
*P	*estabelecimento de alojamento local deve ter certificação energética*
*P	*certificação energética do alojamento local*
*P	*Como fazer a certificação energética do meu alojamento local?*
*P	*Qual o procedimento para certificar energeticamente o meu alojamento local?*
R	*De acordo com esclarecimento da DGEG (Direção-Geral de Energia e Geologia) se os estabelecimentos de alojamento local se reportarem a edifícios ou frações autónomas ...*

Table 4 Accuracy of different search strategies.

Strategy	Correct answers			
	Normal		**Fuzzy**	
VanillaQuestion	275	(61%)	292	(65%)
VanillaQuestionAnswer	296	(66%)	314	(70%)
StemQuestion	297	(66%)	258	(57%)
StemQuestionAnswer	309	(69%)	263	(58%)
StemSynsQuestion	290	(64%)	244	(54%)
StemSynsQuestionAnswer	308	(68%)	237	(53%)
LemmaQuestion	279	(62%)	264	(59%)
LemmaQuestionAnswer	296	(66%)	292	(65%)
LemmaSynsQuestion	270	(60%)	252	(56%)
LemmaSynsQuestionAnswer	302	(67%)	280	(62%)

4.4   Combining Search Strategies

Implemented search strategies are substantially different and some seem to have a complementary nature, which leads to the selection of different answers. This made us wonder whether their search results could be combined, all contributing to the selection of better answers.

To test our hypothesis, we adopted a consensus-based voting method, BordaCount [6], which considers that searches often retrieve more than one result, in a ranked list. BordaCount scores candidates according to their rank on several lists. The higher the rank, the higher the

score. More precisely, the score of the first candidate on a rank will be equal to the number of considered positions. For instance, if we consider the top-5 candidates, the first candidate gets 5 points and the fifth gets 1 point. The selected answer will be the one of the FAQ that has the highest final score, obtained by summing all of its partial scores, in all the considered lists, in this case, retrieved by each search strategy.

Tables 5 and 6 illustrate how this method works with three search strategies (VanillaQuestion, StemQuestionAnswer, LemmaSynsQuestionAnswer), for the query "*o que acontece se perto de uma sexshop for construido um espaço para crianças*" (what happens if, next to a sex shop, a space for children is built). Table 5 has the ranked candidates, identified as AX, for each strategy, and the resulting BordaCount ranking, including the score of each candidate. Table 6 has the answers of all the candidates in the previous table. This example also shows that not all strategies retrieve the same answers – e.g., some are introduced only when stemming or lemmatisation is performed (A6, A7), and others only when considering synonyms (A10, A11) – and combining different strategies broadens the search space.

Table 5 Top-5 candidate FAQs for the query "*o que acontece se perto de uma sexshop for construído um espaço para crianças*", with three search strategies plus the resulting ranking with BordaCount.

Strategy	Top candidate FAQs				
	1 (5 points)	**2 (4)**	**3 (3)**	**4 (2)**	**5 (1)**
VanillaQuestion	A1	A2	A3	A4	A5
StemQuestionAnswer	A6	A1	A7	A8	A9
LemmaSynsQuestionAnswer	A1	A10	A6	A7	A11
BordaCount	A1 (14 points)	A6 (8)	A7 (5)	A2 (4)	A10 (4)

Table 6 Content of answers used in the example of the previous table.

ID	Content
A1	*A sua instalação não impede o funcionamento da sex shop, ainda que sejam sujeitos a obras ou se verifique a alteração do respetivo titular.*
A2	*O Turismo de Portugal I.P. fixa um prazo não inferior a 30 dias para que o estabelecimento inicie o processo de autorização de utilização para fins turísticos.*
A3	*A instalação de estabelecimentos de alojamento local em edifícios construídos de raiz para o efeito não está impedida por lei...*
A4	*Estes estabelecimentos podem continuar a utilizar essa denominação mas dispõem de um prazo de cinco anos ...*
A5	*Não existe um uso de "alojamento local"...*
A6	*Tratando-se de um imóvel construído antes da entrada em vigor do Decreto-Lei n.º 38382 de 7 de agosto de 1951 ...*
A7	*Os estabelecimentos sex shop devem cumprir os seguintes requisitos: ...*
A8	*Estes estabelecimentos podem continuar a utilizar essa denominação mas dispõem de um prazo de cinco anos ...*
A9	*A instalação de estabelecimentos de alojamento local em edifícios construídos de raiz para o efeito não está impedida por lei ...*
A10	*Só pode registar o apartamento como estabelecimento de alojamento local se ...*
A11	*Não, mantendo-se o mesmo estabelecimento de alojamento local, apenas é necessário efetuar uma alteração ao respetivo registo, ...*

Table 7 has the results obtained by combining all the search strategies, excluding (Normal), including (Normal+Fuzzy) or exclusively with fuzzy versions (Fuzzy), plus a selection of three somehow complementary strategies (Three): VanillaQuestion, StemQuestionAnswer and LemmaSynsQuestionAnswer. BordaCount considered always the top-5 candidates.

The obtained results confirm that it might be wise to combine results of different strategies, as they often complement each other. In this case, when combining all the results, 15 ($> 3\%$) more questions were answered correctly, when compared to the best search strategy.

Yet, some search strategies are very similar, so it might not always be necessary to consider them all. Among other combinations of three strategies tested, the one presented got the highest accuracy. Considering both fuzzy and normal searches, it answered one more question correctly than combining all the search strategies. This suggests that combining too many strategies, some of which similar, may result in bias and have a negative impact on the final results.

Table 7 Results of the BordaCount using different search strategies.

Combination	Correct answers	
All (Normal)	307	(68%)
All (Fuzzy)	313	(67%)
All (Normal+Fuzzy)	328	(73%)
Three (Normal)	317	(70%)
Three (Fuzzy)	313	(70%)
Three (Normal+Fuzzy)	329	(73%)

5   Conclusion and Future Work

We described how the IR library Lucene can be used in the development of a QA agent that finds answers to questions in Portuguese. This is often enough and avoids the development of an agent with dialog capabilities, possibly resorting to closed platforms that only provide limited control to the developer.

As others did for similar purposes [11, 13, 14, 5], we took advantage of Lucene's indexing and search capabilities, but also exploited other utilities offered by this library and tuned them to our specific context. This included the use of a synonym map, considering fuzzy searches, or adding search fields with the result of additional pre-processing, namely lemmatisation. Configurations based on the previous lead to the development different search strategies.

Adapting the agent to a domain is a matter of changing the underlying list of FAQs it should be able to answer. In order to test the agent, we used it to answer FAQs related to economic activity in Portugal. In this scenario, we also created an evaluation dataset with variations of the original questions, which enabled the comparison of different search strategies.

Results suggest that fuzzy searches may be an alternative to language-specific normalisation; that searching both in the question and in the answer is more fruitful; and that synonyms contribute to better results when the text is lemmatised. Yet, perhaps the most relevant finding is that the best results are obtained when different search strategies are combined with a voting method. This happens because some strategies end up being complementary.

The code of the developed QA agent, to be used with any list of FAQs or underlying agents, is available from `https://github.com/hgoliv/qa_agent`.

Despite the previous insights, results obtained also confirm that answering natural language questions is a challenging task and there is much room for improvement. Towards better results, several developments and experiments were left to perform. Some of them will be tackled in the future. For instance, we aim to analyse the impact of considering other

related words, such as hypernyms, or using more synonyms, though possibly less reliable; the impact of choosing a similarity measure other than Lucene's default BM25; or even to compute accuracy considering not only the first search result, but also how close the correct answer is to the first (i.e., whether it is on the top-3 or top-5).

We also aim to test Semantic Textual Similarity measures for matching user input with FAQs, possibly integrating the results of our previous work [1]. This may be considered as an alternative to Lucene search mechanisms, or, to avoid performance issues, it can be applied to an initial selection of candidates by Lucene, as others did [5].

On the technical level, we aim to investigate how to integrate lemmatisation as a filter in a Lucene's Analyzer class, which would avoid the creation of additional fields for lemmatised versions of the text. As the lemma depends on the part-of-speech and the latter on the sequence of words, this would have to be done before stopword removal.

In order to test future configurations, a larger dataset will soon be created, with more FAQs and also more variations, created and/or validated by a larger crowd. Automatising the creation of such variations may resort both to search logs or paraphrase generation techniques [2].

A longer-term goal would be to consider both the session context and the user feedback in the selection of answers. The former should include the previous questions made and the latter may be used, for instance, for adjusting the weights of a voting method according to its reliability.

References

1   Ana Alves, Hugo Gonçalo Oliveira, Ricardo Rodrigues, and Rui Encarnação. ASAPP 2.0: Advancing the State-of-the-Art of Semantic Textual Similarity for Portuguese. In *Proceedings of 7th Symposium on Languages, Applications and Technologies (SLATE 2018)*, volume 62 of *OASIcs*, pages 12:1–12:17, Dagstuhl, Germany, June 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

2   Anabela Barreiro and Luís Miguel Cabral. ReEscreve: a translator-friendly multi-purpose paraphrasing software tool. In *Proceedings of the Workshop Beyond Translation Memories: New Tools for Translators*, 2009.

3   Daniel Braun, Adrian Hernandez-Mendez, Florian Matthes, and Manfred Langen. Evaluating natural language understanding services for conversational question answering systems. In *Proceedings of the 18th Annual SIGdial Meeting on Discourse and Dialogue*, pages 174–185, 2017.

4   Annalina Caputo, Marco Degemmis, Pasquale Lops, Francesco Lovecchio, and Vito Manzari. Overview of the EVALITA 2016 Question Answering for Frequently Asked Questions (QA4FAQ) Task. In *Proceedings of Third Italian Conference on Computational Linguistics (CLiC-it 2016) & Fifth Evaluation Campaign of Natural Language Processing and Speech Tools for Italian. Final Workshop (EVALITA 2016)*, volume 1749 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2016.

5   Lei Cui, Shaohan Huang, Furu Wei, Chuanqi Tan, Chaoqun Duan, and Ming Zhou. SuperAgent: A Customer Service Chatbot for E-commerce Websites. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, ACL 2017, Vancouver, Canada, July 30 - August 4, System Demonstrations*, pages 97–102. ACL Press, 2017.

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

7   Erick R. Fonseca, Simone Magnolini, Anna Feltracco, Mohammed R. H. Qwaider, and Bernardo Magnini. Tweaking Word Embeddings for FAQ Ranking. In *Fifth Evaluation Campaign of Natural Language Processing and Speech Tools for Italian*, volume 1749. CEUR-WS.org, 2016.

8   Hugo Gonçalo Oliveira. A Survey on Portuguese Lexical Knowledge Bases: Contents, Comparison and Combination. *Information*, 9(2), 2018.

9   Lynette Hirschman and Robert Gaizauskas. Natural Language Question Answering: the View from Here. *Natural Language Engineering*, 7(4):275–300, 2001.

10   Zongcheng Ji, Zhengdong Lu, and Hang Li. An Information Retrieval Approach to Short Text Conversation. *CoRR*, abs/1408.6988, 2014.

11   Valentin Jijkoun and Maarten de Rijke. Retrieving Answers from Frequently Asked Questions Pages on the Web. In *Proceedings of the 14th ACM International Conference on Information and Knowledge Management*, CIKM '05, pages 76–83, New York, NY, USA, 2005. ACM.

12   Oleksandr Kolomiyets and Marie-Francine Moens. A Survey on Question Answering Technology from an Information Retrieval Perspective. *Information Sciences*, 181(24):5412–5434, December 2011.

13   Govind Kothari, Sumit Negi, Tanveer A. Faruquie, Venkatesan T. Chakaravarthy, and L. Venkata Subramaniam. SMS Based Interface for FAQ Retrieval. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 2 - Volume 2*, ACL '09, pages 852–860, Stroudsburg, PA, USA, 2009. ACL Press.

14   Daniel Magarreiro, Luísa Coheur, and Francisco S. Melour. Using subtitles to deal with Out-of-Domain interactions. In *Proceedings of 18th Workshop on the Semantics and Pragmatics of Dialogue (SemDial)*, pages 98–106, 2014.

15   Arianna Pipitone, Giuseppe Tirone, and Roberto Pirrone. ChiLab4It system in the QA4FAQ competition. In *Fifth Evaluation Campaign of Natural Language Processing and Speech Tools for Italian*, volume 1749. CEUR-WS.org, 2016.

16   Fabio Rinaldi, James Dowdall, Michael Hess, Diego Mollá, Rolf Schwitter, and Kaarel Kaljurand. Knowledge-Based Question Answering. In *Proceedings of the 7th International Conference on Knowledge-Based Intelligent Information and Engineering Systems (KES 2003)*, pages 785–792, Oxford, UK, September 2003. Springer-Verlag.

17   Stephen Robertson and Hugo Zaragoza. The Probabilistic Relevance Framework: BM25 and Beyond. *Found. Trends Inf. Retr.*, 3(4):333–389, April 2009.

18   Ricardo Rodrigues, Hugo Gonçalo Oliveira, and Paulo Gomes. NLPPort: A Pipeline for Portuguese NLP (Short Paper). In *7th Symposium on Languages, Applications and Technologies (SLATE 2018)*, volume 62 of *OASIcs*, pages 18:1–18:9, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

19   Yiping Song, Cheng-Te Li, Jian-Yun Nie, Ming Zhang, Dongyan Zhao, and Rui Yan. An Ensemble of Retrieval-Based and Generation-Based Human-Computer Conversation Systems. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*, pages 4382–4388. International Joint Conferences on Artificial Intelligence Organization, July 2018.

20   Oriol Vinyals and Quoc V. Le. A Neural Conversational Model. In *Proceedings of ICML 2015 Deep Learning Workshop*, Lille, France, 2015.

21   Ellen M. Voorhees. The TREC Question Answering Track. *Nat. Lang. Eng.*, 7(4):361–378, December 2001.

22   Joseph Weizenbaum. ELIZA: a computer program for the study of natural language communication between man and machine. *Commun. ACM*, 9(1):36–45, January 1966.

Tracing Naming Semantics in Unit Tests of Popular Github Android Projects

Matej Madeja

Department of Computers and Informatics, Faculty of Electrical Engineering and Informatics,
Technical University of Košice, Slovakia
`https://madeja.github.io/`
matej.madeja@tuke.sk

Jaroslav Porubän

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

— Abstract —

The tests are so closely linked to the source code that we consider them up-to-date documentation. Developers are aware of recommended naming conventions and other best practices that should be used to write tests. In this paper we focus on how the developers test in practice and what conventions they use. For the analysis 5 very popular Android projects from Github were selected. The results show that 49 % of tests contain full and 76 % of tests contain a partial unit under test (UUT) method name in their name. Further, there was observed that UUT was only rarely tested by multiple test classes and thus in cases when the tester wanted to distinguish the way he or she worked with the tested object. The analysis of this paper shows that the word "test" in the test title is not a reliable metric for identifying the test. Apart from assertions, the developers use statements like `verify`, `try-catch` and `throw exception` to verify the correctness of UUT functionality. At the same time it was found out that the test titles contained keywords which could lead to the identification of UUT, use case of test or data used for test. It was also found out that the words in the test title were very often found in its body and in a smaller amount in UUT body which indicated the use of similar vocabulary in tests and UUT.

1 Introduction

Antoniol et al. in [1] argue that a programmer trying to understand an existing system or its code usually goes beyond documentation that is written in a natural language. There are several resources within the documentation, such as document of requirements, user manuals, maintenance book, system manual, etc. These forms of documentation, mostly expressed in a natural language, can facilitate interaction between the system (its source code) and the programmer. The problem with these documents is that each change of source code mostly requires a documentation change as well.

Demeyer et al. [4] indicate that the best reflection of the source code are tests which must remain consistent with the source code during the whole product maintenance. Thanks to that developers can use tests as always up-to-date documentation. Requirements, depending on the development approach, affect the source code and tests in parallel. Test driven

development (TDD) [2] drives the development by building tests according to the requirements and the master (production) code is produced according to tests developed in the first step. In a behavioral driven development (BDD) is the production source code programmed as first and the tests are implemented in addition. Therefore, the requirements are expressed in a particular production or test code.

From the cooperation with companies in our geographical area, during building a testing environment for Android courses [8], we found out that tests are often used to understand the source code by developers engaged in a project. When tests are considered as a source code documentation, the programmer expresses his/her mental model in the source code of the test, and later, another programmer can comprehend the expected functionality from that code. This opens up the possibility to improve the program comprehension through tests.

If a programmer uses tests to understand the application functionality, this process can be very tedious as it is necessary to manually create relations between the test and the production code in programmer's head. If these relations can be formed in the programmer's head, we can assume that creation of these relations could be automated with possibility to enrich the source code, e.g. using comments. Today there are not information about real testing practices used by developers in practice and without this knowledge no one is able to expect how can the test code looks like. There are recommendations, e.g. naming conventions or test smells, whose try to unify the testing process in different projects but the developer can ignore following these conventions. Therefore, the aim of this paper is to found out semantics of vocabulary used in unit tests, which should support program comprehension in the future, especially with focus on open-source Android projects from the GitLab platform. Authors answer the following research questions in this paper:

RQ1: Can a test method or a test class be clearly distinguished by using the word "test" in its name?

RQ2: Does the test title (test method name) contain name of the unit under test (UUT)? Is one UUT tested by multiple test classes?

RQ3: Do words used in the test title clearly describe the status of the application before, during and after the test?

RQ4: Is it possible to evaluate the comprehension of the test body analyzing its statements?

RQ5: Does exist a relation between the test title and test body or UUT body?

In section 2 recommended naming conventions and bad smells with focus on tests are described. Section 3 describes the process of projects selection for analysis, data collection, case study results and possible threads to validity. In section 4 related work is presented, section 5 describes the future research directions and in section 6 results conclusions are pointed out.

2   Best practices of writing tests

To make tests more reliable and effective many authors tried to define practices that help maintain the test code. In the following sections we briefly describe them with focus on our case study.

2.1   Naming conventions

In the listing 1 is an example of a simple test for the `MyUnit` class by recommendations of Beck and Gamma [3]. Authors write about the basic test writing conventions, e.g. for Java programs is expected that one test class tests only one UUT and the test title should consist of the name of the UUT and the word `Test` at the end (e.g. `MyUnitTest`). This way

we can clearly assume which production class will be tested. An integrated development environment (IDE), e.g. *Android Studio*[1], usually automatically offers the test title in the form of described convention, so we can assume that such naming as common.

▮ **Listing 1** Example of test class writing practices in JUnit.

```
public class MyUnitTest {
    @Test
    public void testConcatenate () {
        MyUnit myUnit = new MyUnit ();
        String result = myUnit.concatenate ("one", "two");
        assertEquals ("onetwo", result );
    }
}
```

Beck and Gamma also recommend to include the word `test` as first in the test title to distinguish a test method from a tested method. At the same time, the test title should include the name of the UUT from the production code, e.g. test with title `testConcatenate` will test the `concatenate` method. Fixtures may also be used in the tests class and this way we can clearly distinguish fixtures with test method. When generating tests using the *Android Studio* IDE, the names of the generated test methods are identical to the test class methods and without word "test" in it, but it is still possible to clearly identify the method (e.g. by `@Test` annotation).

However, with the described method of test naming an issue can occur if a developer would like to test one UUT by several tests. Meszaros in his book [9] also mentions testing single UUT by multiple test, where he also describes how to name them, e.g. in the form of use case or test data. An example of such tests are shown in the listing 2.

▮ **Listing 2** Use case or data information the in test method name.

```
public void testAddWithoutExtraParam (){
    int res = (new MyUnit ()).add (2, 8);
    assertThat (res ).isEqualTo (10);
}

public void testAddWithDouble (){
    int res = (new MyUnit ()).add (2.88, 7.12, true );
    assertThat (res ).isEqualTo (10);
}
```

2.2 Test smells

The concept of smells in the source code was introduced by Fowler et al. [5] and it refers to the bad design and implementation of the source code which subsequently exacerbates its sustainability. Van Deursen et al. [13] on the basis of the findings of Fowler et al., define the smells for tests. For example, tests should not be dependent on external services, external data sources, should be unambiguous, etc. In this article authors focus on one of the smells called *Assert Roulette*, i.e. one test should contain only one `assert` method call. Otherwise, if an unsuccessful assertion in the test occurs it is difficult identify the error and it also aggravates the comprehension. In the listing 3 can be seen an example of mentioned test smell where `assert` statement is used multiple times and if the test fails by an assert, it will be difficult to identify the cause.

---

[1] `https://developer.android.com/studio`

■ **Listing 3** Assert Roulette test smell.

```
public void testFlightMileage() {
    Flight newFlight = new Flight(validFlightNumber);

    // verify constructed object
    assertEquals(validFlightNumber, newFlight.number);
    assertEquals("", newFlight.airlineCode);
    assertNull(newFlight.airline);

    // setup mileage and assert
    newFlight.setMileage(1122);
    assertEquals(expectedKilometres, newFlight.getMileageAsKm());
}
```

3 Case study

Based on the recommendations in the section 2 we decided to find out how developers write tests in practice, focusing exclusively on unit tests in popular open-source Android projects. In the following sections realized case study is described.

3.1 Method

The decision to use open-source Android projects for the case study was done especially for ability to compare results with user interface (UI) tests in the future. Authors think that UI tests can suitably complement the source code with the real use cases that are normally done by end users. However, in this case study analysis of UI tests is not important because we would like to focus on the context of the test title with the name of UUT and the bodies of both (production and test methods). UI tests are mostly associated with the source code only by UI elements identifiers, so analysis of an UI test meaning will need to be evaluated in relation with the entire production class or multiple classes which are called during the test execution. According to *test pyramid* [10] concept, we can expect unit, integration and UI tests in Android projects.

3.1.1 Projects selection

According to Pham et al. [11] we can expect presence of tests in popular projects. That's the reason for selection the most popular Android projects on Github. Open-source projects take advantage of developing software in a transparent environment which motivates developers to produce quality source code because their code is often reviewed by project owner or other developers. This is the reason why developers in a transparent environment use coding conventions more often.

We selected projects from 3 different blogs (DZone[2], Aritra's Musings[3], MyBridge[4]) devoted to the Android open-source popular projects. By merging all 3 sources we received a total of 52 projects. Projects were ranked by popularity using Github API[5], i.e. by the number of project stars. Because the relations between the test and the UUT have been done manually the projects sample was limited to the 5 most popular projects.

---

[2] `https://dzone.com/articles/amazing-open-source-android-apps-written-in-java`
[3] `https://blog. aritraroy.in/20-awesome-open-source-android-apps-to-boost-your-development-skills-b62832cffafa4`
[4] `https://medium.mybridge.co/38-amazing-android-open-source-apps-java-1a62b7034c40`
[5] `https://developer.github.com/v3/`

3.1.2 Data collection

Firstly a manual semantic analysis has been carried out. For each project we tried to comprehend the test, identify the UUT and create a relation between them. We analyzed 617 tests in total. Detailed statistics can be seen in the table 1. Full dataset is available at Github[6].

Table 1 General stats of manually analyzed data for particular project.

Position	Project	Production classes	Production methods	Test classes	Test methods
1	Shadowsocks client	6	7	6	8
2	iosched	16	40	16	77
3	Android-CleanArchitecture	17	22	17	29
4	Plaid	37	71	39	180
5	ExoPlayer	49	98	53	323
	SUM	**125**	**238**	**131**	**617**

The relations between the tests and a particular UUT were created using a simple web application in a relational database to have related units (UUT and test) prepared for keyword comparison between the test title and other source code (method body of the test and the UUT). Analysis of a test included the following steps:

1. Comprehension of the test and production code functionality to exactly identify the UUT.
2. Saving the test method body, its name and the class name in which it is located.
3. Saving the UUT method body, its name and the class name in which it is located.
4. Creation of the found relation between the UUT and the test.

The entire data collection lasted 120 hours. Initially it was planned to use the Rompaey's and Demeyer's approach [12] which claims that UUT identification is possible by catching the last method call before the `assert`. However, this solution has proved to be ambiguous because multiple assert calls in a single test suggested several target classes. Another problem was the fact that the last called method was not always the right UUT. In listing 4 there is a specific example from our data collection where the `toString` method is not the target of test but the `fromReader` is the right one. That is the reason why relations between collected data have been done manually.

Listing 4 The last called method is not always the target method for a test.

```
expected = Acl().fromReader(INPUT1.reader()).toString();
assertEquals(expected, "testing string");
```

Some of the tests tested several UUT at once as a result of assert roulette test smell. In this case multiple relations with the production code have been created. If the test contained multiple calls of the `assert` method, the number of calls have been recorded, too, to evaluate the average occurrence of assertions in the test. Also the assertions placed within a help method have been included. If the assert was in the loop the number of its execution has been calculated depending on the test data.

---

[6] `https://github.com/madeja/android-tests-semantics`

This case study would like to find out if the words included in the test title occur in the test or UUT body. It is considered only the first level of the code (means only main bodies of UUT and test have been saved). E.g. if the UUT named `myMethod` calls in its body an external method `externalMethod`, the body of the external method `externalMethod` is not included for test title words analysis. An exception was a case if the UUT body contained a call to a single foreign or parent method (such as in the listing 5, the `constuctor` method with three parameters would be used). Following rules have been used for saving the method body for the analysis:

- from the production code the method body with the highest semantic value for the tested functionality,
- from the test code body of the particular test method.

The highest semantic value was determined by the the observer comparison of UUT and test body, i.e. which UUT accurately implements tested functionality.

■ **Listing 5** For keyword analysis is used method with real method statements.

```
public function constuctor(int param1, int param2){
    this(param1, param2, true);
}

public function constuctor(int param1, int param2, boolean param3){
    // real method statements
}
```

3.2   Results

RQ1 is focused on the ability to clearly identify the test or test class. All test classes from analysis included "test" word in the class name, so we could always clearly identify a test class by its name. The word "test" in the test method title was found in 384 of 617 tests so in most cases is possible to use the method name to determine test. The name of the test method is also influenced by JUnit framework which annotates tests using `@Test` annotation so it is possible to identify a unit test using the annotation. However, the use of the word "test" may be an appropriate habit for other languages in which the code can not be annotated.

At the same time, there was observed the relation between the word "test" and the full name of the UUT in the test title. If word "test" is placed at the beginning of the test title then it expresses meaning: "test *of something*". E.g. in the test called `testReadMethod` a man would be able to predict that it is a *test of read method*. Even 124 tests included the UUT name immediately after word "test" so this metric can partially help with UUT identification.

▷ Claim 1.   It is possible to exactly identify test class in Android projects by searching for "test" word in the test class name. Unit tests written in JUnit are annotated by `@Test` annotation, nevertheless the general naming convention of using word "test" in test title was found in 62% of tests.

To answer the **RQ2** it was tried to look for the occurrence of words of UUT method name in the test title. As can be seen in the table 2, in the 303 tests (49%) the full name of the UUT was found in the test title, so it was possible to precisely identify the UUT. If the name of the test is composed of multiple UUT names using the last called method before `assert` [12] is possible to identify the UUT more precisely. This way is possible to help the developer to comprehend the meaning and the purpose of the test.

■ **Table 2** The occurrence of the UUT method name in the test method name.

Name of prod method name included in test method name									
fully included		**partially included***							
yes	**no**	**0 w**	**1 w**	**2 w**	**3 w**	**4 w**	**5 w**	**6 w**	**7 w**
303	314	168	191	170	87	10	11	1	2
SUM		472							

*) w = word/words;

Looking for the partial UUT name in the test title we can get at least a partial link between them. For example, for the UUT with name `onRemoveClicked` was the test title `testRemoveReservation`. As can be seen, the use case of the test is to remove the reservation. If the method is in the `Reservation` class than we can exactly create the relation between the test and the UUT. The name of each UUT was divided by the Java *camel case* conventions (e.g. `camelCaseMethod` produces 3 words: camel, case, method). Since the use of *snake case* convention has been observed in some tests (such as `snake_case_method`) this convention was also accepted. In the table 2 we can see that in 472 tests the partial UUT name was found in the test title, which makes 76% of the analyzed tests.

Developers test a class by 2 or more test cases very rarely. Only 3% of all UUT were tested by 2 different test classes. This occurred only in the *ExoPlayer* project where the tested object was in the first class tested directly and in the second class with impact of the various callbacks. For example, if there was a test class `TsExtractorTest` and `TsExtractorSeekTest`, their meaning was as follows:

- `TsExtractorTest` - tests of the basic functionality of the object.
- `TsExtractorSeekTest` - test of the object during expected changes using callbacks.

▷ Claim 2.   Unambiguous UUT identification based on the test title can be performed on approximately half of the tests in Android projects and in about 76% of tests is possible to predict the UUT at least partially. An UUT is typically tested by one test class, rarely it is tested by multiple test classes for easier understanding of tested object manipulation.

To answer the **RQ3** manual grouping of the frequently occurring words in test names has been carried out. Words used in test title can affect the reader's comprehension. The table 3 contains a complete list of all the keywords found that can help with automation of test code meaning. By the nature the developer tries to name the test in the way which helps him or her quickly identify the cause of test failure in the future. When a keyword is found in the test name it is possible to assume the content of the data contained in the test title and use them to produce complete sentences in natural language based on predefined templates.

During the analysis it has been noticed that observer comprehended the test easier when the "_" character was used in the title. This character has been used as a delimiter of the semantics, e.g. between the data used in the test and the expected test result. If the delimiter was not part of the test then the readability of the method name was worse and many times there was no clear intention in the test title. For example, the test named `testPeekFullyAtEndThenReadEndOfInput` may have two meanings:

- indicates the entire use case that is a prerequisite to perform the test
- or the test verifies whether during `PeekFullyAtEnd` call the end of input read occurs.

From the test body it is possible to determine the expected behavior but using the delimiter in the test title makes the meaning straightforward. The meaning of some keywords affects the use of other keywords in front of it. As can be seen in the table 3 the most

common occurrence was for `when` and `with` keywords. It means that it is possible to read the test prerequisites and data used for testing from the test title quite often.

▷ Claim 3.   Developers in test titles often describe the use case of the test, the testing data or the expected result. Occurrence of keywords `test`, `testShould`, `when`, `after`, `if`, `then`, `with`, `is`, `has` and `return/returns` in the test can enable to detect the UUT state before, during and after the test execution, as well as the data used for the test.

Answering the **RQ4** should evaluate the quality of the tests and find out whether the control flow statements are used in the body of test. In the table 4 we can see that 4 different test evaluation approaches were used in the tests. The quality of a test case can be defined as the ability to detect a source code defect. Developer should be able to comprehend the cause of the failure from assertion message. Most often the `assert` statement was used but in 200 tests this command was used multiple times. In case of test failure the error identification is unclear (assert roulette test smell). We counted assertions with and without loops for every test separately. If there was used a loop in the test we calculated the number of assertions according to loop iterations counted from the amount of testing data. We had one big data test that executed `assert` statement 3,794,020 times. In general the following statements are used for test evaluation:

1. `assert` – classical approach,
2. `verify` – to verify calling the correct method on the object after an action,
3. `try-catch` – to verify whether right exception has been thrown,
4. `exception throw` – if an exception is thrown, test failed.

In the case of a multiple exception throw occurrence in a test it would not be considered as test smell because exceptions could be uniquely identified by an exception message.

According to the best practices for testing [9] developers should avoid loops and consequently other control flow statements of particular programming language because these statements create an unstable and obscure test full of smells. In the table 5 we can see the usage of these statements in tests which negatively affect the simplicity of test comprehension and the test failure identification. As we can see, the average assertion count with loops is 6,155, most likely due to the above mentioned big data test, but 30 tests contained more than 10 assertions in the body in total.

▷ Claim 4.   Developers often (in our study 46%) create tests that contain multiple verification of UUT which worsen exact and quick identification of UUT. This state directly affects the use of control flow statements in the test body where occurrence of the assert roulette test smell is very high.

In order to find out how much information we can find from the test method name in the UUT or test body (**RQ5**) we created a 20% distribution of words included in test title covered by UUT or test body (see figure 1). Even 94% of the tests contain more than 20% of the test title words in the test body and 57% covers more than 60% of words. The coverage of words from the test title in the UUT body was a little bit worse but not negligible. In 64% of all tests titles was the coverage of corresponding UUT bodies greater than 20% indicating that the test title meaning use similar vocabulary as test which can help to establish the relationship between the source code and the test.

▷ Claim 5.   The words included in test title are used mainly in the test body and partly in the corresponding UUT body. Based on the similar vocabulary it is possible to combine the semantic of method calls in the tests and the UUT.

Table 3 Observed keywords which can be used to uniquely identify the purpose of the test from its name.

Keyword	Test method example*	Semantic	Occurrence
test	**test***Read*	what is tested or particular action on object (use case)	370
testShould	**testShould***FailWhenExecute*WithNullObserver	expected test result or behaviour	2
when	**when***NoIntervalsTriggered*_thenMapFn_isOnlyCalledOnce	description of test prerequisite (data or use case)	82
after	testSeekBack**After***ReadingAhead*	description of test prerequisite (data or use case)	15
if	testDownloadLicenseFails**If***NullInitData*	description of test prerequisite (data or use case)	9
then	whenNoIntervalsTriggered_**then***MapFn*_isOnlyCalledOnce	expected result after condition met (if occurred immediately after "when")	3
	testStartsIn0Minutes_**then***HasNullTimeUntilStart*	expected result after condition met (if occurred immediately after "test" and "_" occurred before)**	3
	testPeekFullyAtEnd**Then***ReadEndOfInput*	consequence of method calls in test (if occurred immidiately after "test")	2
with	comment_**with***NoComments*	description of used data for test	85
is	whenNoIntervalsTriggered_thenMapFn_**is***OnlyCalledOnce*	expected result or object state	21
has	testSkipInAlternatingTestSignal_**has***CorrectOutputAndSkippedFrameCounts*	expected result or object state (at the begining of name or exactly after "_")	3
return/returns	testReadDuration_**returns***CorrectDuration*	expected result or object state	40

* Bold text shows the keyword, italic text object to which the keyword refers.

** "_" ⟹ semantic delimiter between UUT, input data, use case etc., used in 322 tests of analyzed data.

Table 4 Statements used for test result evaluation.

	Statement/evaluation of test failure				
	assert (with loops)	assert (without loops)	verify	try-catch	exception throw
Tests using particular statement	542	542	77	42	13
Tests using multiple same statements	249	249	26	4	0
Average per test	6155.79	2.16	0.194	0.076	0.021

Table 5 Java language control flow statements used in test methods' bodies.

	Statement				
	for	foreach (enhanced for)	while	if	switch
Tests using particular statement	16	7	9	8	0

3.3 Threads to validity

In the performed case study 5 very popular Android projects have been analyzed whose do not include all projects in the world. Therefore, it is possible that the results and claims may not be accurate and adaptable for other projects. With focus on the Android platform the results may be affected by the naming conventions of the selected platform which considerably influence the naming of the methods in the production code or the final implementation. At the same time, the case study was focused on open-source projects that may have a different nature and quality compared to proprietary projects that we did not have access to. The source of the projects was the Github platform. Other collaborative tools can have a different impact on the motivation to write quality code which could also affect the results.

The number of tests between projects was uneven as it is not possible to guarantee the same number of tests in each project. In the future, it would be advisable to limit the maximum number of tests per project so the coding style of a particular team code did not affect the results in the undesirable way.

Because creation of relations between tests and the production code have been done manually some faults could occurred while comprehending the source code and during creation of relations between UUT and test. For more accurate relations between the analyzed data it would be necessary to increase the number of observers in the study.

Analysis of the production code and tests was performed only at the first level of the method call, i.e. only main bodies of the tests' methods. Content and semantics of helper methods and test fixtures have not been included in the analysis, so in case of RQ5 can be meaning of the code hidden in other levels of method calls, too.

4   Related work

Kochhar et al. [7] conducted a case study on open-source Android projects in which authors looked at understanding of testing culture in selected projects. They used F-droid[7] as source of open-source Android projects and they analyzed 600 projects in total. Authors conclude

---

[7] `https://f-droid.org/`
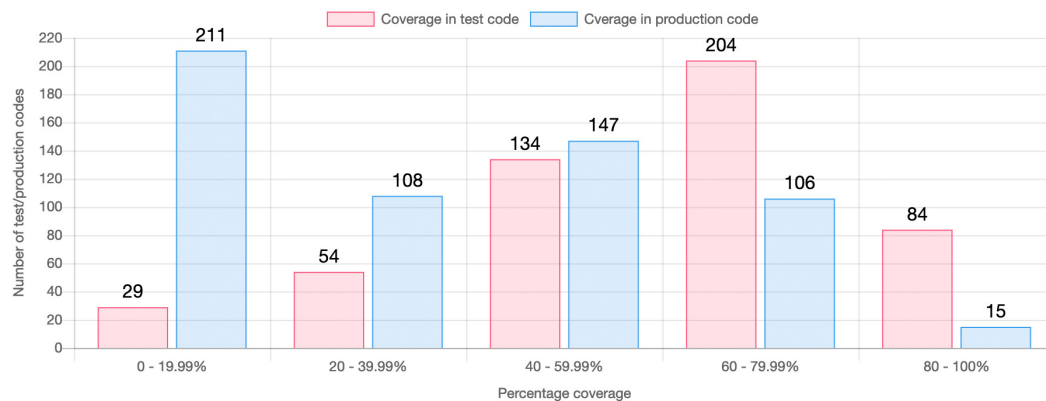
Figure 1 Coverage of words from test method name in body of test or production code.

that many Android projects are poorly tested. They found out that only about 14% of the apps contained test cases and only about 9% of the apps had executable test cases with coverage above 40%. At the same time, they describe what testing tools developer use in practice, however, developers often prefer to test their apps manually. In this paper we analyze the general practices of writing unit tests in open-source Android projects, regardless of the testing framework.

Gellenbeck and Cook [6] devoted their work to the influence of naming variables and procedures on program comprehension. They claims that beacons, sets of key program features, help developers to comprehend an unfamiliar program. The impact of procedure naming on comprehension has been explored on the production code of binary search and sorting procedures. Authors found out that both meaningful procedure and variable names are severest as beacons to high-level comprehension. In this paper we look at naming methods as well, but with respect to the UUT and the test body.

In [12] Rompuy and Demeyer describe the existing strategies for searching the UUT from the test code. However, we have found out that these strategies could not be used in our case study. Described strategies in [12] relatively inaccurately determine the UUT because they depend on the particular structure of the test. By combining these strategies it could be possible to identify the UUT more precisely which is a challenge for future research in this area. The results from our case study complement their strategy based on naming conventions with the real practices of writing tests which can improve the UUT identification.

Pham et al. [11] observed the behavior of developers and their motivation of making tests in a transparent GitHub environment. They contacted 16,000 project developers from whom they received 569 filled out questionnaires of Github members. They found several strategies that software developers and managers can use to positively influence test behavior in their projects. Authors report on the challenges and risks caused by this and suggests guidelines for the promotion of a sustainable test culture in software development projects.

5 Future Work

In the future we would like to compare the results from unit tests with the semantic of UI tests. Since UI tests have less connection to the production source code and they execute functionality from a user perspective it is possible to assume that they will contain the particular user stories that are expected during the real application use in production.

However, for analyzing UI tests we will have to track the calls during the program runtime and in the real context of the device, so creating connections to the source code could be more complicated.

We plan to create a more general overview of real testing practices in different programming languages to maximize the generalization of results. We would like to identify the differences in testing for different languages to find propose features for new IDE tools supporting program comprehension which can have a great impact on the efficiency of creating and maintaining the source code. Our results show that the tests contain many smells that have a negative impact on the comprehension of the program. Based on these results, it is possible to focus on warn the programmer about a test smell occurrence and navigating him/her in better naming of test methods whose name can greatly improve the comprehension.

On the other hand, there exist other techniques that can be used to find semantic relations between UUT and test. In the future it can also be analyzed acronyms, partial words or synonyms to reach more accurate semantics. It is also appropriate to consider approaches that improve program comprehension using code refactoring, e.g. renaming methods names and other identifiers.

With exact determination the UUT from the test it will be possible to enrich the source code with additional information obtained from tests, e.g. how many times the method is tested and in which use cases it is used. If a developer changes the production code then it is possible to notify the programmer that it is necessary to update a specific test that could have been affected.

6   Conclusion

This paper presents a case study on real practices of writing tests in open-source Android projects. For the case study 5 very popular projects from the Github platform have been selected and by manual data collection the relations between the test and the UUT were created. We analyzed 617 tests in 131 test classes and 238 production methods in 125 classes.

There was examined the consistency of the words used in the title of the test and the target UUT. The word was identified based on the Java camel case naming conventions. It was found out that 76% of the tests contain at least the partial UUT name in the test title. At the same time, we tried to identify whether the UUT is tested by only one test class and if not why the developer creates multiple test classes for the same UUT. From the study we claim that developers test an UUT with different test classes rarely and mostly use multiple test classes to distinguish work with the object in the test.

According to the best practices the word "test" should be found in the test name for its clear identification. It was found out that only 61% of the tests included word "test" in its name. JUnit is mostly used for unit testing in Java projects which use `@Test` annotation to denote test. This annotation practice influenced results of performed case study in way of the "test" word occurrence in the test title. Developers in addition to the `assert` statement also use `verify`, `try-catch` and `throw exception` to evaluate the success or failure of the test. In the tests a high incidence of assert roulette test smell has been found which negatively influences the clear identification of UUT. Control flow statements in the test bodies make test difficult to comprehend and complicates creation of relations to the production code.

When creating tests developers use keywords and patterns which can help us to identify the UUT or use case of the test. This work created a summary of all observed keywords used by developers. Importance of individual keywords may depend on their position against other keywords. There was also observed that despite the Java camel case writing convention testers use the "_" character to separate semantically related data groups in the test title.

In the test title words coverage analysis in the UUT or test body we found out that 57% of the tests had a coverage of more than 60% which means that a short description of the test functionality can be often found in the test title, i.e. particular use case. When comparing the words from the test title to the UUT, the coverage is smaller, but 64% of UUT bodies covers more than 20% words of test title. From the above is possible to claim that the body of the test and the body of the UUT method uses the similar vocabulary.

Despite the fact that this case study focused solely on open-source Android Github projects, the results obtained in this paper can help to develop new and more reliable methods of identifying UUT from the test. Achieving more accurate UUT identification can help to enrich the production code with information from the tests in the future and it has potential to improve the program comprehension.

References

1 G. Antoniol, G. Canfora, G. Casazza, A. De Lucia, and E. Merlo. Recovering traceability links between code and documentation. *IEEE Transactions on Software Engineering*, 28(10):970–983, October 2002. `doi:10.1109/TSE.2002.1041053`.

2 Kent Beck. *Test-driven development: by example.* Addison-Wesley Professional, 2003.

3 Kent Beck and Erich Gamma. Test infected: Programmers love writing tests. *Java Report*, 3(7):37–50, 1998.

4 Serge Demeyer, Stéphane Ducasse, and Oscar Nierstrasz. *Object-oriented reengineering patterns.* Elsevier, 2002.

5 M. Fowler, K. Beck, J.C. Shanklin, E. Gamma, J. Brant, W. Opdyke, and D. Roberts. *Refactoring: Improving the Design of Existing Code.* Addison-Wesley object technology series. Addison-Wesley, 1999.

6 Edward M. Gellenbeck and Curtis R. Cook. An Investigation of Procedure and Variable Names As Beacons During Program Comprehension. Technical report, Oregon State University, Corvallis, OR, USA, 1991.

7 P. S. Kochhar, F. Thung, N. Nagappan, T. Zimmermann, and D. Lo. Understanding the Test Automation Culture of App Developers. In *2015 IEEE 8th International Conference on Software Testing, Verification and Validation (ICST)*, pages 1–10, April 2015. `doi:10.1109/ICST.2015.7102609`.

8 Matej Madeja and Jaroslav Porubän. Automated testing environment and assessment of assignments for Android MOOC. *Open Computer Science*, 8(1):80–92, 2018.

9 Gerard Meszaros. *xUnit test patterns: Refactoring test code.* Pearson Education, 2007.

10 Godfrey Nolan. *Agile Android.* Apress, 2015.

11 R. Pham, L. Singer, O. Liskin, F. F. Filho, and K. Schneider. Creating a shared understanding of testing culture on a social coding site. In *2013 35th International Conference on Software Engineering (ICSE)*, pages 112–121, May 2013. `doi:10.1109/ICSE.2013.6606557`.

12 B. V. Rompaey and S. Demeyer. Establishing Traceability Links between Unit Test Cases and Units under Test. In *2009 13th European Conference on Software Maintenance and Reengineering*, pages 209–218, March 2009. `doi:10.1109/CSMR.2009.39`.

13 Arie Van Deursen, Leon Moonen, Alex Van Den Bergh, and Gerard Kok. Refactoring test code. In *Proceedings of the 2nd international conference on extreme programming and flexible processes in software engineering (XP2001)*, pages 92–95, 2001.

Robust Phoneme Recognition with Little Data

Christopher Dane Shulby 🆔
Institute of Mathematical and Computer Sciences – University of Sao Paulo, Brazil
Samsung SIDI Institute, São Paulo, Brazil
`www.nilc.icmc.usp.br`
c.shulby@icmc.usp.br

Martha Dais Ferreira 🆔
Institute of Mathematical and Computer Sciences – University of Sao Paulo, Brazil
daismf@icmc.usp.br

Rodrigo F. de Mello 🆔
Institute of Mathematical and Computer Sciences – University of Sao Paulo, Brazil
mello@icmc.usp.br

Sandra Maria Aluisio 🆔
Institute of Mathematical and Computer Sciences – University of Sao Paulo, Brazil
sandra@icmc.usp.br

—— Abstract ——
A common belief in the community is that deep learning requires large datasets to be effective. We show that with careful parameter selection, deep feature extraction can be applied even to small datasets.We also explore exactly how much data is necessary to guarantee learning by convergence analysis and calculating the shattering coefficient for the algorithms used. Another problem is that state-of-the-art results are rarely reproducible because they use proprietary datasets, pretrained networks and/or weight initializations from other larger networks. We present a two-fold novelty for this situation where a carefully designed CNN architecture, together with a knowledge-driven classifier achieves nearly state-of-the-art phoneme recognition results with absolutely no pretraining or external weight initialization. We also beat the best replication study of the state of the art with a 28% FER. More importantly, we are able to achieve transparent, reproducible frame-level accuracy and, additionally, perform a convergence analysis to show the generalization capacity of the model providing statistical evidence that our results are not obtained by chance. Furthermore, we show how algorithms with strong learning guarantees can not only benefit from raw data extraction but contribute with more robust results.

1 Introduction

Acoustic modeling, or the statistical representation of speech signals, is essential for parametric ASR (Automatic Speech Recognition). Generally PER (Phone Error Rate) or FER (Frame Error Rate) are used to measure model performance. FER is a more exact metric, since it shows exactly what the acoustic model is capable of and PER is more interesting for applications because in the end, this is the goal of the acoustic model. Still, some kind of PER smoothing technique is inevitably used. For the field of ASR, large companies have been positive on one hand, as they have matured the technology into production ready algorithms

and increased the access to machine learning libraries like never before, but negative when it comes to publishing reproducible work. Often proprietary databases are used to train for benchmark results as in [6, 12] and PER or FER is rarely given. One of our principal objectives has been to be as transparent as possible and establish metrics for fair comparison of acoustic models. We also show that deep learning techniques, combined with careful engineering can be useful even for non-big-data scenarios, especially in the case of raw feature extraction (completely statistical-based). In this article, we experiment with context windows to better understand the performance of the model with respect to phonemic context and then, we demonstrate the robustness of the model with the learning guarantees provided by the Vapnik Chervonenkis (VC) theory [23]. It is important to accurately model acoustic properties so that errors do not propagate to other models. Reliable phoneme-level error detection is still a great need for automatic pronunciation training. This is not trivial, since acoustic models are still quite far from perfect and most ASR pipelines rely on pronunciation models (and sometimes language models) to overcome these deficiencies. In general, there is not an equal share of the wealth of data resources and the majority of the world's languages have not benefited from deep speech technology for a number of reasons: i) they require thousands of hours of transcribed/annotated data [17]; ii) they require a respectable infrastructure with large scale GPU-computing and often take weeks or months to train [5]; and iii) they do not offer solid learning guarantees [26]. More specifically, the amount of training data is a limiting factor and for more robust applications the amount of training data could grow exponentially with augmentation techniques like MCT (Multi-conditional Training) [13]. Secondly, infrastructures which can adequately support this type of training exists in very few university laboratories and is mostly limited to the private sector. Lastly, these vastly trained models which have been long suspected and shown by [26] to be great data memorizers, but poor generalizers and difficult to scale in real world applications, even in the best conditions. In this paper, we focus on reproducible results, showing that raw feature extraction can still be used for SotA (State of the Art) ASR in low-resource training environments, where careful parameter selection and good architecture choices can compensate for a deficit in data. We chose to use a CNN (Convolutional Neural Network) as the feature extractor for this work, because of raw feature extraction via convolution and dimension reduction by max pooling. Since human experts are capable of recognizing phonemes in a spectrogram given the concentration of visual formants, it seems reasonable to ask this of a computer vision algorithm which was inspired by a mammalian visual cortex. This is why we chose to treat this as an image processing problem, instead of a more classical signal processing approach which would use the Discrete Fourier Transforms directly. In Section 3, we describe the TIMIT database used in our experiments, the preprocessing steps taken for images used in the CNN feature extractor and the architecture proposed for the experiments. The experiments, results and discussions are presented in Section 4 followed by the convergence analysis in Section 5. Finally we will make our concluding remarks in argument for reproducible and robust acoustic models in Section 6.

2   Related Work

The CNN has proven to be useful for phoneme recognition. [1] proposed a hybrid CNN-HMM model using local filtering and max-pooling in the frequency domain where a strong benchmark of 21.6% PER is established. Their network is also pretrained on 18 hours of Google voice search data. [18] explore the optimal CNN architecture presenting the best results with large corpora (300-400 hours). While the results are promising for SotA

applications using a CNN, both of these strategies require a great deal of resources to train and can be difficult for comparison by other researchers without access to these datasets or pre-trained weights. One can assume that some pre-processing on the images and data smoothing was used to generate PER. It is also likely a pronunciation and possibly language model was used after the acoustic model's posterior probabilities were generated, but these details are not clarified in those works. FER could be useful and is not given in most SotA papers like [16, 2, 18, 1, 11, 21]

Table 1 shows the SotA results we were able to compile, which use both FER and PER metrics. In the table, each study is listed with the method used, whether the paper provides sufficient information to reproduce it exactly (REP? - short for reproducible) and the PER and FER. Studies without both PER and FER were not included in the table.

■ **Table 1** SotA Results using PER and FER as metrics.

Ref	Method	REP?	PER	FER
[10]	DBLSTM-RNN	N	**17.7**	27.9
[20]	CNN + CTC	N	29.4	**22.1**
[22]	DLSTM-RNN	Y	25.4	29.4

In 2013, [10] benchmarked the TIMIT corpus at 17.7% PER and 27.88% FER. This DBLSTM-RNN (Bidirectional Long Short-Term Memory-Recurrent Neural Network) included 3 Hidden layers with 250 units in each and pre-trained (from a larger dataset) CTC (Connectionist Temporal Classification) finite state transducers. The 50 speaker development set was used for fine-tuning and early stopping as well as a biphone language model for predictions. Interestingly, [22] did make a best-effort attempt to reproduce the network by [10]. The main difference was that a DLSTM instead of a DBLSTM is used. The author explains that this was because of the lack of availability of a Bi-directional LSTM within the TensorFlow library (as a project scope setting). Still, he does use a three layer LSTM network with 250 hidden units. The author used the openly available default initialization from TensorFlow based on [25], since the data from [10] was not available. His network used a mini-batch size of 6 sub-sequences of 20 frames and applied dropout regularization. The final results of this reproducible network were 29.43% FER and 25.36% PER. The FER results are quite comparable to [10] but the PER is much higher. This is probably due to the bi-phone language model. It also seems like the pretraining was rather similar (using the TensorFlow default) to the database that was used in the previous studies by [9] and [10]. As far as FER is concerned, the best work is [20], beating the mark set by [10]. This technical report is interesting since it uses a CNN as well with CTC and achieves a FER of 22.1% but does not break the mark for PER. While the authors do not offer any evidence for the generalization capacity, they do explain their training philosophy as stated in the article: "we train until the model begin[s] to overfit on the training set and the dev accuracy begins to fall. Much of the training is done on SAIL's Deep clusters, which uses nVidia GTX780 GPUs". The network with the best performance was the 25 frame windowed 128-256-384-384 CNN followed by 1024-512 dense layers. This means that the network is very large and likely makes use of a lot of data. They also do not give the hyper-parameters used in their CNN and explain that they used their own pretrained language model for predictions and a pretrained RNN-CTC from some dataset (also not described). Since none of these resources are made available, this study is unfortunately not reproducible either.

3   Dataset, Features and Architecture

3.1   Dataset and Feature Extraction

We used the TIMIT Corpus. This corpus was used because it has been an industry benchmark for decades and is a small (ca. 5 hours) dataset which is phonetically balanced for English. As suggested in [14], we collapsed the phone set into 39 monophones. We did not; however, train the 61 phone set before collapsing so as not to dilute the training samples. To be consistent with other works, we also removed silence frames from the final predictions for FER and PER. We extracted the features from spectrogram images which we preprocessed with sox, from Hann windows of 25ms with a stride of 10ms. The spectrogram of each 25ms window has 5x128 color pixels according to the sox color palete. This means that each pixel in time direction corresponds to 5ms (200 pixels per second) and 128 frequency points are taken in the frequency direction (DFT size of 254). Afterwards, we searched for the best kernels to adequately represent our dataset, as explained in subsection 3.2.
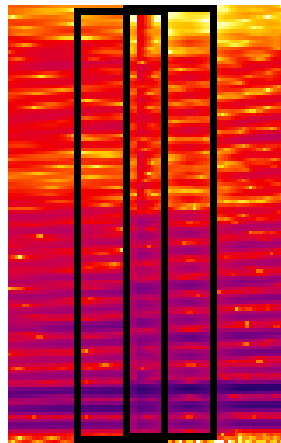
Figure 1 Spectrogram illustrating 25ms Hann windows with a stride of 10ms.

3.2   Architecture

3.2.1   CNN Feature Extractor

Our parameters network architecture, was estimated using an approach based on False Nearest Neighbors (FNN) proposed by [8]. This technique estimates the kernel sizes to best represent data patterns and found the most aggressive pooling layers to lower the dimensions of our feature maps since these would then be passed on to a SVM (Support Vector Machine). This method was used to generate the best five configurations and from those options, we chose the simplest configurations which was decided by using the one which presented the largest masks with the fewest neurons. The final parameters used were 3 Convolutional layers: 1. 36 units, kernel=15×2; 2. 31 units, kernel=15×1 + max-pooling=3×3, stride=2×2; 3. 15 units, kernel=8×1 + max-pooling=3×3, stride=2×2. ReLU activation was applied due to its widespread adoption in literature. We used a batch size of 128, a learning rate of 0.01 and trained over 31 epochs. The convolutional network required 8 hours using a single Titan-X GPU and 32GB of RAM. In the SVM, we augmented the context while skipping some frames. The defining principle in our frame skipping is that we left the two adjacent frames (1 right, 1 left) always intact, believing that these frames are the most important

ones. After the three middle frames, we skipped every other frame until we reached the extremity of the window. For example if we have 11 overlaping frames (25ms/step of 10 ms) of context, we would only train the first, third, fifth, sixth, seventh, ninth and eleventh frames, considering the sixth as the central frame. This would be considered 1 instance for training and the next instances would follow the 10ms step size for the next frame until the end of the audio (the first and last 10 full seconds are padded with zeros). The full pipeline can be seen in Figure 2. We found that frame skipping greatly reduces the number of features with very little impact on results as later explained in section 4.
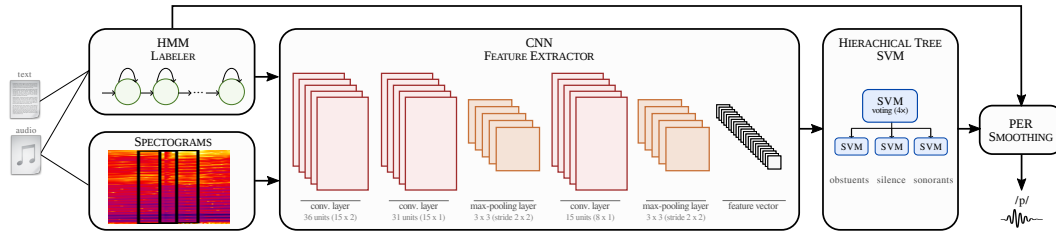
Figure 2 CNN-HTSVM architecture defined for the experiments.

3.2.2 HTSVM Classifier

The SVM parameters were found empirically after several experiments. The selected kernel for final experiments was a 4th order polynomial kernel with $coef0 = 1$ (as a non-homogeneous kernel) and a cost $C = 10,000$. The interested reader can find more details about the exact architecture of the hierarchical tree structure in [19]. One of the biggest issues of the TIMIT dataset is that it is not balanced with respect to its classes. For example: in the training set, the phoneme /k/ appears in $60,433$ frames, whereas /g/ is found in only $17,727$. In order to build a robust system, it is important to learn this phonemic distinction and minimize the influence of probability in the training set. We were able to deal with this by using the SMOTE [4] data augmentation technique. In order to obtain PER, the classified frames were converted to phonemes by taking the mode of all of the SVM classifications for each HMM boundary generated in the labeler.

4   Experiments and Results

We ran two types of experiments in this paper. The first experiments were designed to demonstrate the cost/benefit of window skipping and can be seen in Table 2. An asterisk symbol in the number of frames column denotes that window-skipping was used and in the classifier column indicates that stochastic gradient descent (SGD) was used, otherwise the solver was Adam. Once we completed the MLP (Multi-Layer Perceptron) experiments, we ran selected experiments with the SVM.

■ **Table 2** Experiments on CNN features with window-widening.

Frames	Classifier	FER	PER	F1
1	MLP	0.49	0.54	0.36
3	MLP	0.45	0.50	0.41
5	MLP	0.43	0.48	0.45
7	MLP	0.41	0.45	0.48
9	MLP	0.40	0.44	0.49
11	MLP	0.39	0.43	0.51
11*	MLP	0.39	0.43	0.50
11*	MLP w/SGD	0.38	0.42	0.51
13	MLP	0.39	0.42	0.51
15*	MLP w/SGD	0.37	0.40	0.52
19	MLP w/SGD	0.37	0.39	0.52
1	SVM	0.42	0.47	0.44
3	SVM	0.41	0.45	0.48
11*	SVM	0.30	0.33	0.61
19*	SVM	0.28	0.32	0.63

The second set of experiments were designed to compare our algorithm with the SotA. Here we compare our results to the replication study done by [22], our attempted replication of the famous CNN from [1] (without any pre-training or language model) and the traditional GMM-HMM as a baseline classifier which is generally used for classification on small datasets. As a traditional baseline, we used one of the most popular ASR toolkits for a database the size of TIMIT, the HTK toolkit. A triphone HTK model with 31 Guassians was trained on the same TIMIT training set used in our method and recognition was performed on the test set with a zero-gram language model and only the individual monophones as the pronunciation model in order to obtain only the posterior values from the acoustic model. We used 31 Gaussian components because this number is one which we have found useful in the past. It is higher than what is recommended by the voxforge tutorial [15], which uses 15 and is similar to the models used by Keith Vertanen [24], where he uses a maximum of 32 for the Wall Street Journal and TIMIT datasets together. The model was trained using MFCC 0DANZ acoustic features, where 0 uses the zeroth cepstral coefficient, D=delta coefficients, A=delta delta acceleration coefficients, N=absolute energy suppression and Z=zero mean normalization. The predictions were then segmented in the same fashion as the proposed method with 25ms sliding windows and a step of 10ms. In the case of the CNN replication, we used the closest parameters possible to those given in [1] as follows: conv layer 1: $1,000$ units, kernel size $= 8 \times 8$; max-pooling 1: 5×5 / 2×2; conv layer 2: $1,000$ units, kernel size$=8 \times 8$; max-pooling 2: 5×5 / 2×2, flattened into a 512 unit dense layer and softmax. This created a feature map with 4,224,000 dimensions. The first 15 epochs were trained with a learning rate of 0.08 and the last 10 with 0.002. A batch size of 128 and 11 frames for context and ReLu activation were used. One adaptation should be pointed out: the 6×6 max-pooling layer created negative divisions (due to an unknown pre-processing step) so as a conservative measure, we used the maximum possible which was 5×5. The network overfit the data quickly to 87% accuracy after the first 15 iterations. We used the same HMM labeler for all experiments and all other default parameters in the CNN were kept the same. Table 3 presents the FER, PER and F1 scores of the experiments.

■ **Table 3** F1 Scores in Frames, Frame Error Rates and Phone Error Rates for each model.

Study	Classifier	FER	PER	F1
This work Baseline	GMM-HMM	0.76	0.75	0.16
This work Rep.[1]	HMM-CNN	0.58	0.52	0.31
[22] Rep.[10]	DLSTM RNN	0.29	**0.25**	n/a
This work	CNN-HTSVM	**0.28**	0.32	**0.63**

It should be pointed out that the poor performance of the HMM-CNN was expected on such a small dataset. The network was built to be trained using the weights already aquired from the larger private dataset used in that study. Also, the poor performance of the vanilla GMM-HMM model can be explained since the tied-state probabilities are highly dependent on the pronunciation and language models used in the traditional decoder which we did not use here. Our purpose was simply to show that this method does not perform well in frame-wise classification, even though it is widely used for speech-recognition tasks with small datasets. On the other hand the replication studies can be compared and show that when the proper tools are used, excellent results can be achieved, even when large amounts of data are not available.

Independent of the models accuracy, it is also important to understand what sort of errors the models are actually committing. Table 4 lists the 15 most frequent errors committed by each system, including the true values, predicted values and the confusion percentage. The GMM-HMM systems are known to produce rather jumbled posterior values since they typically rely on providing a ranking to the pronunciation model where these issues are normally solved. Still, this is not an adequate approach for phoneme recognition.

■ **Table 4** Most Frequent FER Confusion percentages.

GMM-HMM			CNN-MLP			CNN-HTSVM		
True	Pred	Conf (%)	True	Pred	Conf (%)	True	Pred	Conf (%)
s	z	33.14	s	f	21.48	s	z	15.16
ih	uw	16.00	ih	ae	16.23	ay	ae	39.64
t	ch	17.58	iy	ae	14.99	ao	aa	26.58
er	r	32.46	z	f	28.01	r	er	18.84
ao	l	28.00	ay	ae	24.59	sh	s	26.01
iy	y	14.23	eh	ae	25.46	aa	ae	16.07
s	sh	10.09	aa	ae	20.96	ah	ae	14.79
ae	t	14.32	er	ae	14.32	t	s	7.61
ih	z	10.07	k	t	13.22	iy	ih	6.48
w	ao	45.52	ey	ae	25.29	er	r	7.64

Here one can see that the findings in studies like [3] can be confirmed that the SVM does get confused when phonemes are very similar. Still, this may be caused by the transcription of the 8 dialects where some sounds, especially vowel sounds can have some overlap. One

observation between the SVM and MLP classifiers is that the MLP makes more erratic errors. Since the SVM is governed by a decision boundary function and the MLP is a probability of an argmax function, this makes sense. Even though the same CNN features were extracted, it seems that the MLP was more likely to develop a "trash" category where when in doubt it goes with a "more probable" class. In the example of /ae/, it was correctly classified 60% of the time so it became a go-to class when in doubt. In the case of /f/, the phoneme was classified correctly in 85% of the instances, so other fricatives were more likely to be classified as /f/ as well. The GMM-HMM is notoriously unsuccessful on the phoneme level which is why HMM-based engines use tied states to calculate the cost of substitutions to find the correct transcription in the pronunciation model. What is most evident is that often when it errs, it fails badly. The MLP is also not the most graceful failure, where the SVM is more robust in that its errors are more reasonable as pointed out in [3].

5   Convergence analysis

We believe that it is important to present statistical evidence that results are not found by chance. To save space, all formulas referred to in this section can be found below and are numbered for easy reference:

$$P\left(\sup_{f \in \mathcal{F}} |R(f) - R_{emp}(f)| \geq \epsilon\right) \leq \delta \tag{1}$$

$$\delta = 2\mathcal{N}(\mathcal{F}, 2n)e^{-n\epsilon^2/4} \tag{2}$$

$$R(f) \leq R_{emp}(f) + \sqrt{4/n(\log(2\mathcal{N}(\mathcal{F}, n)) - \log(\delta))} \tag{3}$$

$$R(f) \leq R_{emp}(f) + \sqrt{c/n(R^2/\rho^2 - \log(1/\delta))} \tag{4}$$

$$s(n) = (198.4n^2 - 2574.8n + 6744.3)^{36}$$
$$+ (150.6n^2 - 2000.9n + 5386.5)^{31}$$
$$+ (89.3n^2 - 1200.5n + 3304.4)^{15}$$
$$\lim_{n \to \infty} \frac{\log\{s(n)\}}{n} \approx 0, \tag{5}$$

$$R(f) \leq 0.28 +$$
$$\sqrt{4/n(3,332,567^2/173,869,050^2 - \log(1/\delta))} \tag{6}$$

SLT (Statistical Learning Theory) provides theoretical support for such convergence proofs in terms of how supervised learning algorithms generalize examples. The empirical risk minimization [23] defines the main principle of SLT. As seen in equation 1, that formulation intends to bound the divergence ϵ between the empirical risk R_{emp}, i.e., the error measured in a sample, and the expected risk $R(f)$, i.e., the expected error while assessing the joint probability distribution of examples and their respective classes, as the sample size n tends towards infinity. In the same equation, f refers to some classifier, and $\mathcal{F}$ is the space of admissible functions provided by some supervised algorithm, a.k.a. the algorithm bias as described in [23] and [7]. Additionally, Vapnik proved a bound for supervised learning algorithms considering the shattering coefficient $\mathcal{N}(\mathcal{F}, 2n)$. Such a coefficient is a measure function to compute the complexity of the algorithm bias, i.e., the cardinality of functions contained in the space $\mathcal{F}$ that produce different classification outputs, provided a sample size

n. Throughout our formulation, we employ the generalization bound defined in equation 3 to ensure that the expected risk is bounded by the empirical risk plus an additional term associated with the shattering coefficient and some probability δ. In the case of the SVM, the same bound is formulated as shown in equation 4, in which c is some constant, R corresponds to the dataset radius, and ρ represents the maximal margin. In this context, we assessed our CNN and SVM to understand the sample size they require to ensure learning in the context of speech recognition, allowing to estimate their expected risk value over unseen examples. Now, we proceed by computing the generalization bound for the CNN (equation 3). Considering $\delta = 0.05$, which represents a probability of 0.95 (i.e., 95%) to ensure that the empirical risk $R_{\mathrm{emp}}(f)$ is a good estimator for the expected risk $R(f)$, meaning the error results measured for our classifier indeed work on unseen examples. The convergence curve (CC) obtained from the same equation is shown in Figure 3. Observe that our CNN requires at least $985,128$ examples to converge, while we had in practice $1,447,869$ examples in training set.
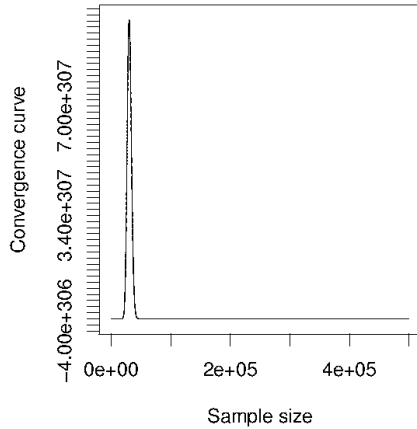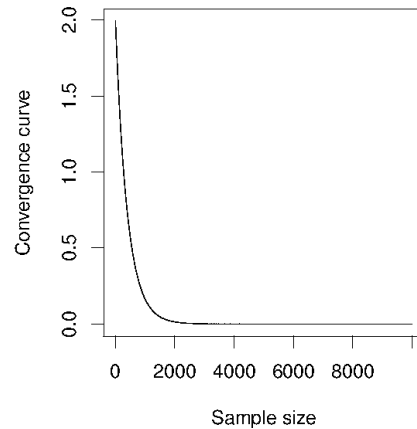
Figure 3 CC, from eq. 5.   **Figure 4** CC, from eq. 6.

We can employ another result from Vapnik to prove that our CNN converges. Equation 5 simplifies the previous equation and considers only the most relevant term to prove learning convergence. Notice that as the term $\frac{\log CNN(n)}{n}$ approaches zero, term $\sqrt{4/n(\log(2CNN(n)) - \log(0.05))}$ goes to zero, remaining the empirical risk as an assessment measure of the learning performance. Next, the SVM is also analyzed considering equation 4. In this case, we have an accuracy of 0.72 leading to $v(f) = 1 - \mathrm{accuracy} = 1 - 0.72 = 0.28$, $R = 3,332,567$ (the radius we estimated for the whole dataset), and $\rho = 173,869,050$ as the maximal margin found. Consequently, the generalization bound for our SVM is defined in equation 6. Also, considering $\delta = 0.05$ as before, and $c = 4$ as taken in the default formulation (Equation 3), Figure 4 illustrates the convergence curve obtained for the SVM. Notice our SVM requires at least $11,981$ examples to converge, while we had in practice $1,447,869$ examples in the training set. Based on [7], we conclude the convergence analysis for our CNN and SVM, ensuring learning in the context of speech recognition.

6   Conclusion

In this paper, we have focused on a feature extraction approach from two biases: 1.) a domain bias where we took great care in preprocessing the data in a way which would minimally preserve the most important acoustic information; and 2.) a statistical learning

bias where we checked in every step that we had the most simple approach which would guarantee learning and would lead to the least complex hypothesis space possible. This work shows that even without pretraining or fine-tuning, we achieve better FER results than the SotA replication by [22] and we are not so far from the reported SotA which cannot be faithfully reproduced [10, 20] and better than our reproduction tests of the CNN by [1]. Large networks like these cannot generalize well for small datasets without pretraining or heavy regularization as we can see a large difference in the training and test scores as well as the F-measure. As for our CNN, we would like to point out that although the best results were obtained with 19 frames, we believe that 11 frames seems to be the most cost-effective number, since larger configurations demand a much larger number of feature maps and offer only little improvement. The multiple max-pooling layers were great facilitators for the window widening technique since without this dimension reduction the SVM would have been prohibitive to train and even the MLP would have been very difficult on our laboratory work-station. Therefore, while more layers in the CNN architecture do not seem necessary for better feature extraction, they can help reduce dimensionality which aids in classification. We show that careful parameter selection and attention to supervised learning guarantees are essential for building robust models with little data. We also believe that the results from the convergence analysis further evidences the robustness of this strategy and show that with enough examples we are able to guarantee learning. Without this guarantee, it is possible that the results could have been obtained by chance. We hope that this work will inspire others to seek statistical evidence to support their models and that they will describe them in a way which can be reproduced.

7   Future Work

For future work, it would be interesting to test our method on other datasets, especially those with accented speech, noisy speech and different recording conditions, since we believe that the CNN features could be robust enough to deal with them.

It would also be interesting to see what effect more data has on the classifier to see how well this method performs when data is not so limited. Of course, this would also mean devising strategies to train more data with a SVM, which would need to employ sample reduction techniques.

References

1 Ossama Abdel-Hamid, Abdel-Rahman Mohamed, Hui Jiang, Li Deng, Gerald Penn, and Dong Yu. Convolutional neural networks for speech recognition. *IEEE/ACM Transactions on audio, speech, and language processing*, 22(10):1533–1545, 2014.

2 Ossama Abdel-Hamid, Abdel-rahman Mohamed, Hui Jiang, and Gerald Penn. Applying convolutional neural networks concepts to hybrid NN-HMM model for speech recognition. In *2012 IEEE international conference on Acoustics, speech and signal processing (ICASSP)*, pages 4277–4280. IEEE, 2012.

3 Rimah Amami and Noureddine Ellouze. Study of phonemes confusions in hierarchical automatic phoneme recognition system. *arXiv preprint*, 2015. `arXiv:1508.01718`.

4 Nitesh V. Chawla, Kevin W. Bowyer, Lawrence O. Hall, and W. Philip Kegelmeyer. SMOTE: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, 16:321–357, 2002.

5 Xie Chen, Adam Eversole, Gang Li, Dong Yu, and Frank Seide. Pipelined Back-Propagation for Context-Dependent Deep Neural Networks. In *Interspeech*, pages 26–29, 2012.

6   Chung-Cheng Chiu, Tara N Sainath, Yonghui Wu, Rohit Prabhavalkar, Patrick Nguyen, Zhifeng Chen, Anjuli Kannan, Ron J Weiss, Kanishka Rao, Katya Gonina, et al. State-of-the-art speech recognition with sequence-to-sequence models. *arXiv preprint*, 2017. `arXiv:1712.01769`.

7   R. Fernandes de Mello, M. Dais Ferreira, and M. Antonelli Ponti. Providing theoretical learning guarantees to Deep Learning Networks. *ArXiv e-prints*, 2017.

8   Martha Dais Ferreira, Deborah Cristina Correa, Luis Gustavo Nonato, and Rodrigo F de Mello. Designing Architectures of Convolutional Neural Networks to Solve Practical Problems. *2017 Elesvier pre-print*, 2017.

9   Alex Graves and Navdeep Jaitly. Towards End-To-End Speech Recognition with Recurrent Neural Networks. In *ICML*, volume 14, pages 1764–1772, 2014.

10   Alex Graves, Navdeep Jaitly, and Abdel-rahman Mohamed. Hybrid speech recognition with deep bidirectional LSTM. In *Automatic Speech Recognition and Understanding (ASRU), 2013 IEEE Workshop on*, pages 273–278. IEEE, 2013.

11   Awni Y Hannun, Andrew L Maas, Daniel Jurafsky, and Andrew Y Ng. First-pass large vocabulary continuous speech recognition using bi-directional recurrent DNNs. *arXiv preprint*, 2014. `arXiv:1408.2873`.

12   Zhiheng Huang, Geoffrey Zweig, and Benoit Dumoulin. Cache based recurrent neural network language model inference for first pass speech recognition. In *Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on*, pages 6354–6358. IEEE, 2014.

13   Tae Yoon Kim, Chang Woo Han, Sangha Kim, Donghoon Ahn, Seokyeong Jeong, and Jae Won Lee. Korean LVCSR system development for personal assistant service. In *Consumer Electronics (ICCE), 2016 IEEE International Conference on*, pages 93–96. IEEE, 2016.

14   K-F Lee and H-W Hon. Speaker-independent phone recognition using hidden Markov models. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 37(11):1641–1648, 1989.

15   Ken MacLean. Tutorial: Create Acoustic Model - Manually, 2018. Accessed: 2018-03-01. URL: `http://www.voxforge.org/home/dev/acousticmodels/linux/create/htkjulius/tutorial/triphones/step-10`.

16   Abdel-rahman Mohamed, George E Dahl, and Geoffrey Hinton. Acoustic modeling using deep belief networks. *IEEE Transactions on Audio, Speech, and Language Processing*, 20(1):14–22, 2012.

17   Vassil Panayotov, Guoguo Chen, Daniel Povey, and Sanjeev Khudanpur. Librispeech: an ASR corpus based on public domain audio books. In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5206–5210. IEEE, 2015.

18   Tara N Sainath, Abdel-rahman Mohamed, Brian Kingsbury, and Bhuvana Ramabhadran. Deep convolutional neural networks for LVCSR. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 8614–8618. IEEE, 2013.

19   Christopher Dane Shulby, Martha Dais Ferreira, Rodrigo F de Mello, and Sandra Maria Aluisio. Acoustic Modeling Using a Shallow CNN-HTSVM Architecture. *2017 Brazilian Conference on Intelligent Systems*, pages 85–90, 2017.

20   William Song and Jim Cai. End-to-end deep neural network for automatic speech recognition. *Technical Report*, 2015.

21   László Tóth. Phone recognition with hierarchical convolutional deep maxout networks. *EURASIP Journal on Audio, Speech, and Music Processing*, 2015(1):25, 2015.

22   Timo van Niedek, Tom Heskes, and David van Leeuwen. Phonetic Classification in TensorFlow. *Bachelor's Thesis, Radboud University*, 2016.

23   Vladimir Vapnik. *Statistical learning theory*, volume 1. Wiley New York, 1998.

24   Keith Vertanen. HTK Acoustic Models, 2018. Accessed: 2018-03-01. URL: `https://www.keithv.com/software/htk/us/`.

25   Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. Recurrent neural network regularization. *arXiv preprint*, 2014. `arXiv:1409.2329`.

26   Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. *arXiv preprint*, 2016. `arXiv:1611.03530`.

Towards European Portuguese Conversational Assistants for Smart Homes

Maksym Ketsmur
DETI/IEETA, Universidade de Aveiro, Aveiro, Portugal
mvk@ua.pt

António Teixeira
DETI/IEETA, Universidade de Aveiro, Aveiro, Portugal
ajst@ua.pt

Nuno Almeida
DETI/IEETA, Universidade de Aveiro, Aveiro, Portugal
nunoalmeida@ua.pt

Samuel Silva
DETI/IEETA, Universidade de Aveiro, Aveiro, Portugal
sss@ua.pt

Abstract

Nowadays, smart environments, such as Smart Homes, are becoming a reality, due to the access to a wide variety of smart devices at a low cost. These devices are connected to the home network and inhabitants can interact with them using smartphones, tablets and smart assistants, a feature with rising popularity. The diversity of devices, the user's expectations regarding Smart Homes, and assistants' requirements pose several challenges. In this context, a Smart Home Assistant capable of conversation and device integration can be a valuable help to the inhabitants, not only for smart device control, but also to obtain valuable information and have a broader picture of how the house and its devices behave. This paper presents the current stage of development of one such assistant, targeting European Portuguese, not only supporting the control of home devices, but also providing a potentially more natural way to access a variety of information regarding the home and its devices. The development has been made in the scope of Smart Green Homes (SGH) project.

1 Introduction

Home Assistants, such as Amazon Alexa [14], have gained popularity and there is also a consistent upward trend of integration of smart devices into our homes, from small sensors used to monitor home environment to large smart appliances.

Smart Home assistants aim to provide a more natural way to interact with our home and the smart devices that are becoming an intrinsic part of its environment. The most popular Smart Home assistants are Google Assistant, Amazon Alexa and a recent intelligent assistant

contributed by Yandex, known under the name "Alice" [13]. These assistants are voice based and, in most cases, are integrated with small devices such as Google Home (Google Assistant) and Echo Dot (Amazon Alexa), but are mostly limited to supporting simple features, such as controlling the lights or checking the weather. Also, they do not support the European Portuguese language.

Despite all recent advancements, and the popularity of home voice assistants, the control and access to the information regarding Smart Homes needs improvement. Considering the diversity of smart devices (e.g., smart lights, smart plugs, smart appliances) and sensors (e.g., air quality, temperature, occupancy) available today, it is possible to obtain a wide range of potentially valuable information pertaining our home and its devices, which can be harnessed to provide the occupants with a broader picture of how home and its devices behave. This can, for instance, allow controlling appliances remotely, schedule their activities, and obtain detailed information regarding their consumption, also bringing forward a more effective use of device smartness towards economy and comfort.

These new capabilities need to be aligned with the users' needs. A recent inquiry, by the authors [10] considering 20 participants aged between 10 and 63 years old with different scientific backgrounds revealed a list of capabilities deemed important by users, such as: remote control, home state report, voice communication, appliances and light state querying and control, appliances activities scheduling, temperature control of appliances (e.g., water heater) and house divisions; and being informed about resource consumption.

Considering the limitations of already existing Smart Home assistants, the need for a consistent integration of new devices into our homes, and users expectations, the development of adequate Smart Home assistants poses several challenges to: (1) store all the relevant information (consumptions, interactions, inhabitant behavior) in organized way, creating a knowledge base; (2) provide the means for the assistant to use this information; (3) support users native language, in this case European Portuguese.

The remainder of this article is organized as follows: next section presents related work regarding Conversational Assistants and ontologies for smart environments. Afterwards, Section 3 describes the scenario selected as context to support the development and Section 4 presents the conversational assistant architecture and the knowledge base to support it. Then, Section 5 presents the main capabilities of the developed Assistant and the results obtained during its first evaluation. Finally, conclusions and future work complete the paper, in Section 6.

2   Background and Related work

This section presents background information in Assistants and ontologies, as well as some relevant related work, considered important to contextualize the presented work.

2.1   Conversational Assistants

Conversational assistants perform similar interactions to chatbots and allow speech as input and output [12].

Google Assistant, Apple's Siri, and Microsoft Cortana are examples of popular conversational interfaces. They are, in general, single-turn assistants only suitable for information seeking and simple execution of control commands. They do not adequately support multi-turn voice interactions (dialogue) and use of context.

To enable the interaction between the human and the machine, in a dialog format, a typical conversation system, as illustrated in Fig. 1, must integrate at least five modules [2, 22]:
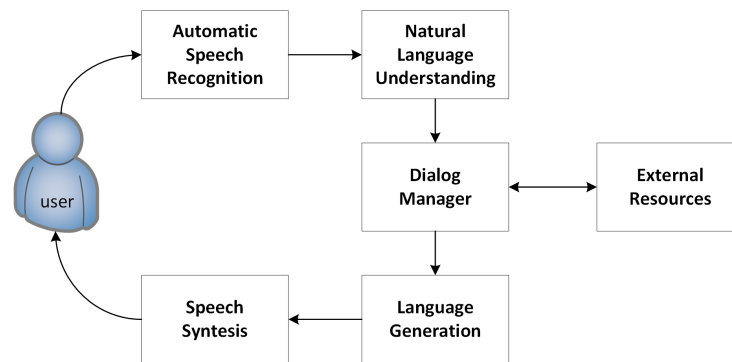
Figure 1 Typical system structure of a conversational assistant.

(1) Automatic Speech Recognition (ASR) converts recorder audio signal (speech) into text; (2) Natural Language Understanding (NLU) tries to understand the input sequences of words to identify important information such as intentions and entities; (3) Dialog Manager (DM) manages dialog and context information, considering intention, entities and previous conversations; (4) Natural Language Generation (NLG) generates sentences; (5) Speech Synthesis, uses Text-to-Speech (TTS) to produce synthetic speech.

2.2 Tools for Assistant development and NLU processing

Even though there is no option of a complete (and configurable) assistant to use as basis for the development, there are some tools that can be used to ensure the features for the different modules identified, above. Particularly relevant for the development of such assistant is the NLU module. Considering the specificity of the domain and the language requirements, there are some tools and resources available to support the developing of conversational assistants [11], as summarized in Table 1.

Table 1 Tools and APIs useful for Chatbots and Conversational Assistants development.

Name	Main capabilities	Access method	Portuguese
DialogFlow	NLU + DM	HTTP	Brazilian
Microsoft LUIS	NLU	HTTP/SDK	Brazilian
IBM Watson Assistant	NLU + DM	HTTP/SDK	Brazilian
Amazon LEX	NLU + DM	HTTP/SDK	Yes
WIT.AI	NLU	HTTP	Yes
Microsoft Speech	ASR + TTS	SDK	Yes

2.3 Ontologies for the Smart Home

Ontologies are commonly used to address data, knowledge, and applications heterogeneity, enabling the support to a service-oriented framework in Smart environments [19, 20, 5], such as Smart Homes that, unlike the traditional home, can be autonomous, consider the different habits of the occupants and adjust the setting accordingly in order to facilitate the household daily life. The use of an ontology for the Smart Home scenario allows to model and describe the different aspects of the smart "things" and residents by defining their semantic properties, the information they can supply or, even, the actions or controls they can perform [17, 4].

DomoML [18] was one of the first approaches that provided a full, modular ontology of household environments, divided into three main ontologies: (1) DomoML-env, defining all fixed elements inside the house; (2) DomoML-fun, describing the functionalities of each house device, in a technology independent manner; and (3) DomoML-core, providing support for the correlation of elements of DomoML-fun and DomoML-env, including the definition of physical quantities.

DogOnt [3] is another approach supporting device/network independent description of homes, classified as "controllable" and "uncontrollable" things, providing one of the few domain models to fit real world domotic systems capabilities and support interoperation between current and future solutions. Also, it uses inheritance mechanisms to automatically associate states and functionalities to the modeled elements.

Human activities, needs, and preferences must also be covered by the ontology and were addressed by Ye et al. [21], defining a generic approach to derive knowledge about context predicates and activities in a structured manner.

Beyond the home environment, and occupant activities and preferences, ontologies related to smart appliances and sensors [17], home energy management [16], and devices and appliances resource consumption [4] have been proposed, in the literature, which cover operational data of appliances and smart devices, the classification of home electrical appliances provided by various vendors and manufacturers, and handling of their consumption data.

3 Application scenario

To support the implementation of our ideas and to have a concrete scenario eliciting requirements, we designed a virtual Smart Home that contains virtual partitions mapped to rooms, in our lab, and many devices, in part real, in part virtual. The real part of the virtual home has the infrastructure to communicate and interact with devices; the virtual devices (appliances) are simulated as computer programs, which generate data and allow the interaction between the assistant and the virtual device.

Our virtual Smart Home has more than 10 virtual appliances generating, using probabilistic algorithms, resource consumption data that posteriorly is stored in the Smart Home semantic knowledge base. Also, new virtual appliances can be easily attached to the system using a simple XML file containing the appliance's definition and characteristics.

4 The Conversational Assistant

This section presents our conversational assistant architecture and its modules, along with a brief description of the knowledge base and ontology supporting the whole system.

4.1 Architecture

Figure 2 presents the architecture adopted for the assistant, composed by three core modules (NLU, DM and NLG) and two attachable modules (ASR and TTS).

The three main modules can support interaction based in written text, making possible a chat version of the assistant. By coupling to the pipeline two additional modules (ASR and TTS), we get a full conversational assistant capable of accepting speech as input and producing speech as its output.
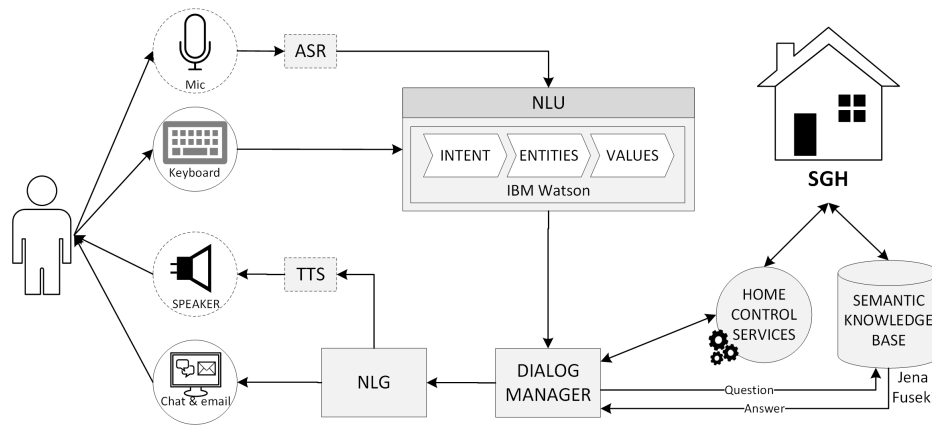
Figure 2 Overall architecture for the proposed Assistant.

4.2 Natural Language Understanding (NLU)

The natural understanding module aims to process the input information in text format and identify its intention and entities.

The NLU processing, in our system, is carried out by an external NLU service, IBM Watson Assistant, that identifies intents, entities and their values, essential to the correct understanding of the user's input, dialog management and output. IBM Watson was selected mainly due to simplicity of use and support for NLU and dialog management capabilities. Initial experiments proved that being available only for the Brazilian variant of Portuguese was not a major limitation.

Watson intents represent the purposes or goals expressed in an input text such as appliances control or consumption querying. While intents represent the purpose or goal, entities represent the context for that purpose. They are important nouns and named entities in the input text. For example, if the user's intention is control then the device and location entities are required.

In Watson, entities can be of two types: dictionary-based entities and contextual entities. The dictionary-based entities are those for which specific terms, synonyms, or patterns can be defined. At run time, the service finds entity mentions only when a term in the user input exactly matches (or closely matches if fuzzy matching is enabled) the value or one of its synonyms. Context-based entities are those for which occurrences of the entity, in sample sentences, are annotated to teach the service about the context in which the entity is typically used.

Each of the entities has a set of values that represent it, for example, the home entity can have room, garage, garden values. Each of the values may have synonyms or patterns that help to better identify them in the input text (e.g., the room value may have synonyms, such as kitchen, living room, and bathroom).

Taking into account the domain of the Assistant, intents and entities were defined to enable the identification of the user's purpose, what actions to take, on which appliances, and in which home partition. The main intents and entities added to Watson are presented in Table 2.

Adding these intents and entities to Watson is done through a training process, consisting essentially in providing sentence examples. Table 3 presents some of the sentences (examples) used to train each of the intents and the related entities. The first rows presents some of the examples used to train the identification of the intent `request` where some sentences are about

Table 2 Watson intents, entities and values.

Name	Type	Description
greet		Identifies when user starts the conversation with the assistant.
command	intent	Identifies when user tries to control any of the appliances or lights.
request		Identifies when user asks the assistant about appliances state or consumption.
action		Represents the action to take, for example consumption querying, appliances state control or error identification.
building_thing	entity	Identifies the appliances or lights that must be controlled in case of command intent or queried about its state or consumption in case of request intent.
error		Identifies when users tries to know about appliance issues, if it has any errors or not.
building_environment		Identifies the room (e.g. kitchen or living room) or home part (e.g. garden or garage) if present in the input sentence.
resource		Identifies resource (water, gas or energy) present in the input sentence.
sys_date		Identifies dates in the input sentences (this month, this week, today)

consumption and others about device state querying. Also, in general, all of the examples for the `request` intent contain already defined entities, such as: `action`, `resource`, `sys_date`, `building_environment`, and `building_thing`, which helps to get a better confidence level in intent recognition.

Table 3 Illustrative examples for training Watson intents.

Intent	Examples	English translation
request	Qual foi o [consumo]?	What was the [consumption]?
	Quanto [consumi]?	How much I've [consumed]
	Qual foi o [consumo] de [energia] da [TV]?	What was the [energy] [consumption] of [TV]?)
	Qual foi o [consumo] de [água] [esta semana]?	What was the [water] [consumption] [this week]?)
	Quanto [consumi] [esta semana] na [cozinha]?	How much I've [consumed] [this week] in the [kitchen]?)
	Como [estão] as [luzes] na [cozinha]?	How [are] the [lights] in the [kitchen]?)
	A [TV] na [sala] [está] [ligado]?	[TV] in the [living room] [is] [on]?)
command	Liga a televisão!	Turn [on] the [TV]!
	Desliga o GEOS !	Turn [off] the [GEOS]!
	Ligue as luzes na cozinha!	Turn the [lights] [on] in the [kitchen]!

To register a new appliance in the system, the information about its synonyms, location and capabilities, made available in a XML file, is used to produce examples and train Watson. As an example, to register a new TV and querying about its state, the implemented training process will automatically generate example sentences, like the following: What is the TV state? The state of television is on? What is the state of TV in the living room?

As an example of what Watson delivers, after being trained, Figure 3 represents Watson's response, in JSON format, for the input "What was the water consumption this week in the kitchen?". The JSON output contains valuable information for the identified intents and entities, location and the corresponding confidence. The location specifies the identified term position, in the input text, and may be helpful in the construction of complex responses. The confidence information is important to make dialogue decisions considering only intents or entities with confidence above a certain threshold (we use 80%).

4.3   Dialog Manager

The Dialog Manager module processes all the information from the NLU module in order to obtain the requested information or execute desired commands. This module consists in a set of rules that detect information changes and conduct the dialog accordingly.

```
{
"intents": [ { "intent": "request", "confidence": 0.9816005229949951 } ],
"entities":
[
    {   "entity": "action", "location": [ 11, 18 ],
        "value": "consumption_query", "confidence": 1 },
    {   "entity": "resource", "location": [ 22, 26 ],
        "value": "water", "confidence": 1 },
    {   "entity": "sys-date", "location": [ 27, 38 ],
        "value": "2019-03-03", "confidence": 1 },
    {   "entity": "sys-date", "location": [ 27, 38 ],
        "value": "2019-03-09", "confidence": 1 },
    {   "entity": "building_environment", "location": [ 42, 49 ],
        "value": "kitchen", "confidence": 1 }
],
"input": {
"text": "What was the water consumption this week in the kitchen?" }
}
```

Figure 3 Example in JSON of the structure and contents of the Watson response.

For each intent, the Dialog Manager has a predefined frame (set of slots) that need to be filled. For instance, when the intent is a `command` the values for an `action, device and home division` need to be provided. Figure 4 shows how the system processes the Watson output and identifies missing information in each step. If the verification of the information fails, a feedback message (question) is generated to the user. The user is queried in order to obtain the missing information and complete the request. The information regarding intent, entities and values are kept until user change the intent.
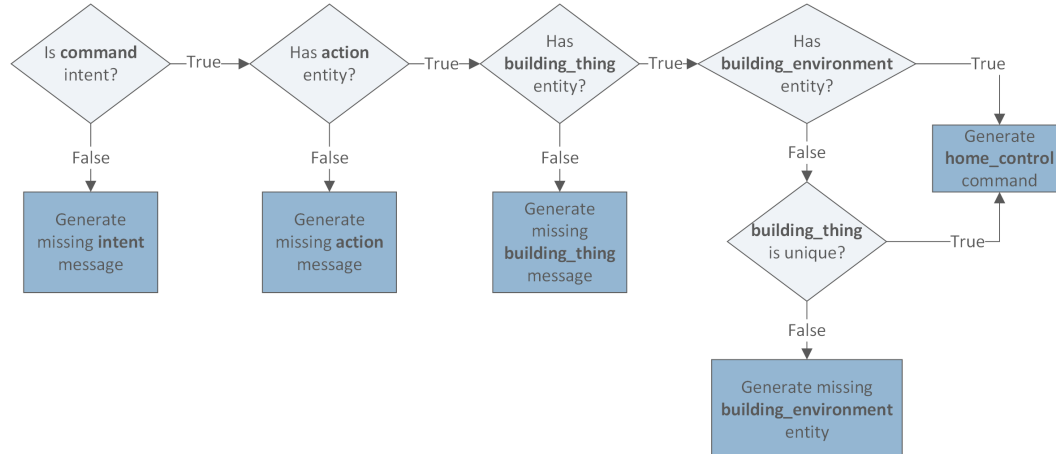
Figure 4 Command intent decision flow.

If the user does not state the `building_environment`, the Dialog Manager tries to infer that information by querying the knowledge base. If inference results in a single value the request is complete. Otherwise, the system generates a message, notifying the user that more than one device of that type is available and requests the specification of its location to the user.

In the case of `command` intents, using previously identified information, the Dialogue Manager builds the command to send to the Home Control Service, which tries to execute it, the result of command execution is sent to the DM, as reply, and feedback is provided to the user.

Besides the `command` intent, users can request information about the house. In this case, a query to the knowledge base is built using SPARQL [15]. For instance, the SPARQL query for the user request "What was the water consumption of GEOS in the kitchen, this month?", to obtain consumption information, is presented in Figure 5. The system builds this query in three steps: (1) select the `building_thing` by name and by `building_environment`; (2) select the consumption events generated by the `building_thing` that have water as consumed resource; (3) filter the information by date interval (this month).

```
SELECT (SUM (?amount) as ?totalAmount)
WHERE {
    ?building_thing :hasName "geos"^^xsd:string ;
                    :isLocatedIn :kitchen .
    ?event :hasType :ConsumptionEvent ;
           :isGeneratedBy ?building_thing ;
           :hasResource :water ;
           :hasDate ?date ;
           :hasAmount ?amount .
FILTER(?date >= "2019-03-01T00:00:00"^^xsd:dateTime &&
       ?date <= "2019-03-31T23:59:59"^^xsd:dateTime)
```

■ **Figure 5** Example of a consumption request SPARQL query.

4.4 Natural Language Generation (NLG)

The NLG module provides mechanisms to create a response to the user. It uses predefined templates (Figure 6), which are completed with information obtained from the knowledge base. A template is chosen according to intent and entities. For example, if the user asks about water consumption for the water heater located in the kitchen, the system: (1) selects a synonym of the involved entities; (2) identifies the measurement unit (cubic meter) of the requested resource (water); (3) compiles the final response.
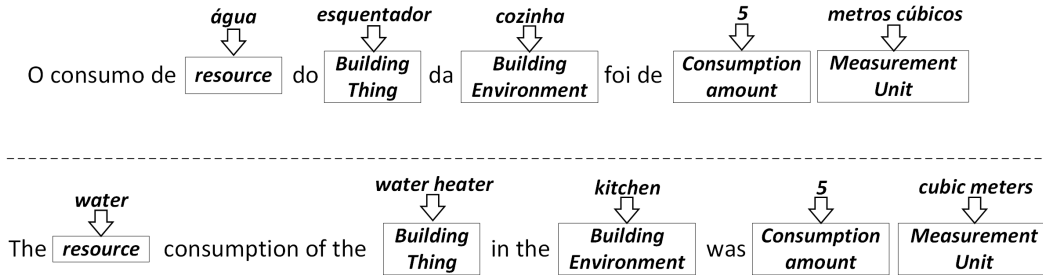
■ **Figure 6** Example of the response template. English translation of actual Portuguese sentence is presented in the bottom of the figure.

4.5 Knowledge base

All the relevant information of the Smart Home and Users-Home interactions are stored using a semantic knowledge base, implemented using the Apache Jena Fuseki server. Information is structured according to a domain ontology developed by the authors integrating and extending existing ontologies.

The knowledge base ontology (Fig. 7) is based on 6 already defined ontologies (dogOnt [3], [1], FOAF [6], TIME [7], DUL [9] and OEMA [8]), and some classes specific to our scenario. The ontology covers all the existing Assistant functions as well as provision for future features.
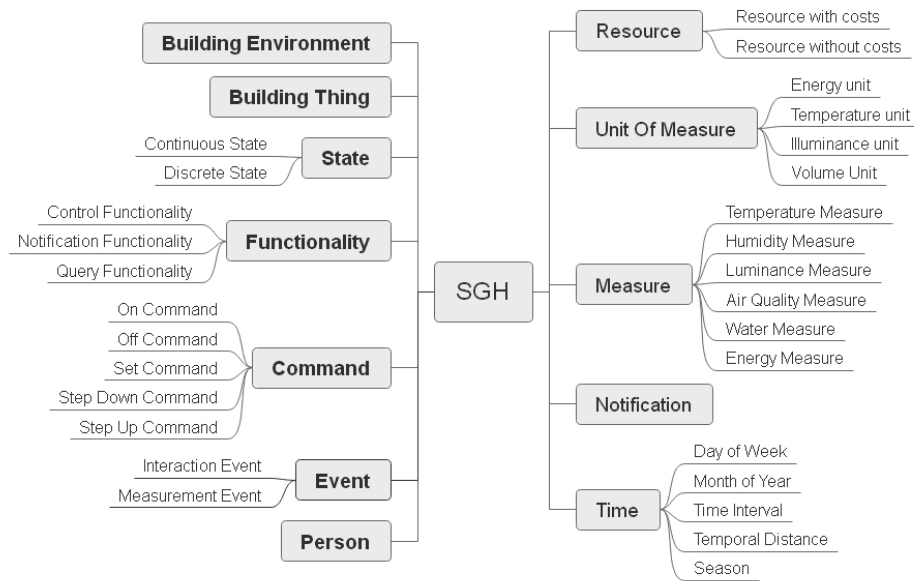
Figure 7 Ontology high level classes overview.

The `Building Environment` and `Building Thing` are the classes that define the Smart Home. The first describes the physical environment in which people live (e.g. living room, bathroom). The second class defines: (1) the equipment that is capable to produce, consume, store and measure data (`Energy Equipment`); (2) systems for detecting abnormal situations (`security system resource`).

The `Person` class represents the home occupants who have some common characteristics (name, surname, nickname, family member and age) and can interact with the home in two ways, physically or using the assistant. In the case of using the assistant, the ontology provides `Command` and `Functionality` classes. They are used by the assistant to identify if the target appliance supports the desired action.

The `Functionality` class is used to specify the appliances' capability. Functionalities are divided into three subclasses, `Control Functionality` describes the control capability of the `Building Things`; `Notification Functionality` specify which notification the `Building Things` can reply (e.g., door sensor notifies "open" or "close"); `Query Functionality` defines the capability to reply to an interrogation about its state (e.g., lights state, oven temperature).

The `Event` class defines the measurements and interaction events occurred in the home. The `Measurement Event` correspond to an appliance's consumption data or sensors data and the `Interaction Events` are physical interactions between an inhabitant and the home (e.g., turning on the oven, open the door, enter the living room).

The `State` class describes all the possible states of `Building Things`. State has two subclasses, a `Continuous State` defining an analog quantity (e.g., temperature, pressure, distance) and `Discrete State` defining a digital quantity from a set of possible values (e.g., on, off, closed, open).

The `Resource`, `Unit of Measure`, `Measure`, `Notification` and `Time` are the auxiliary classes to support store and query of consumption data and interaction notifications.

4.6   Dynamic handling of devices and appliances

One of the main characteristics of smart environments, including Smart Homes, is the dynamic nature of the devices and appliances they integrate. Therefore, a method to expand the capabilities of the assistant to integrate new devices is needed, which can recognize and handle them. Also, the process must be simple and, as much as possible, based in information that can be provided by device manufacturers.

To facilitate production of the needed information regarding the devices while making it readable by humans and machines, the XML format was adopted. Providing an XML file containing the device information (metadata) allows an easy and generic way to register the device. The file describes the available functionalities and how the assistant can interact with the new device. As an example, figure 8 illustrates the XML file for the registration of a new household equipment (water heater). The XML contains the definition of several elements to enable the recognition of devices and future interactions, such as: (1) `type`, specifying the new equipment's semantic type; (2) `room`, defining the new equipment's location in the home; (3) `synonyms`, enabling the assistant to recognize different names for the device, also helping the assistant to respond more naturally. For example, the user asks "Qual foi o consumo de água do esquentador esta semana?"(*What is the water consumption of water heater, this week?*) and the assistant responds "O consumo do esquentador esta semana, foi ..." (*The consumption of the water heater, this week, was ...*); (4) `functionalities`, enumerating the new device's capabilities. Considering the water heater, for instance, it is capable of on/off control, measure water and gas consumption and query state and errors.

The XML file structure contain an `errors` element that is optional, since this feature depends on each device and its capability to generate errors. When devices generate errors, it is necessary to provide a list of possible errors, their `identification`, `description`, possible `solution` and `severity`, usually described in user manuals. This kind of information is important since the user can ask the assistant about the error, what it means, receive suggestions of how to solve it and understand how critical it is, without consulting user manual.

```xml
<device>
  <name>geos</name>
  <type>waterHeater</type>
  <room>Bathroom</room>
  <synonyms>
    <synonym>geos</synonym>
    <synonym>esquentador</synonym>
  </synonyms>
  <errors>
    <error>
      <id>A0</id>
      <description>...</description>
      <solution>...</solution>
      <severity>4</severity>
    </error>
    ...
  </errors>
  <functionalities>
    <functionality>WaterConsumptionMeasurement</functionality>
    <functionality>GasConsumptionMeasurement</functionality>
    <functionality>StateQuerying</functionality>
    <functionality>ErrorQuerying</functionality>
    <functionality>OnOff</functionality>
  </functionalities>
</device>
```

Figure 8 XML structure to new appliance registering.

5 Results

The result section starts by presenting a proof-of-concept Conversational Assistant, describing two important examples of use of the assistant: chat messages and email. It is followed by a presentation of the results for the first evaluation with users.

5.1 Examples of use

An early user study conducted by the authors [10] revealed that, in most cases, users would like to control the home remotely. Our system enabled users to have a conversation with the Assistant from anywhere using Skype or email. Figure 9 shows a screen of a Skype conversation where the users tries to obtain the water consumption for the kitchen. Skype is intended to support more interactive dialogue while using the email is for simple command and control, such as "Desligue a TV na sala!" (*Turn off the TV in the living room!*) or "Desligue as luzes na cozinha!" (*Turn off the lights in the kitchen!*). This way, the user can select the best suited platform to remotely interact with the Conversational Assistant.
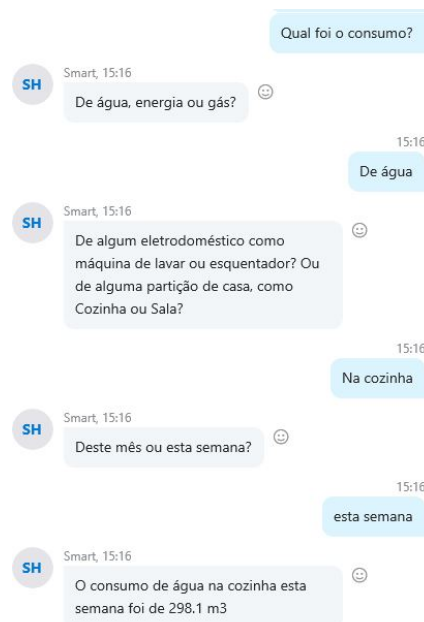
Figure 9 Example of a Skype chat with the conversational assistant: the user is querying the system about water consumption and the assistant is using context.

5.2 First Evaluation

5.2.1 Method

The main goal of the first evaluation with real users was to identify the strongest and weakest points of the first proof-of-concept version of the proposed Smart Home Conversational Assistant. The definition of the evaluation tasks, shown in Table 4, was one of the most challenging steps, since they should lead the participants to explore all the features of the assistant and to discover its dialog capabilities.

Table 4 List of tasks considered for the first evaluation of the proposed conversational assistant.

Nr.	Task
1	Turn on the lights in two home partitions of your choice
2	Find out in which home partition were spent more energy, this month
3	Find out in which home partition were spent more gas, this month
4	Ensure that all lights, in the home, are off.
5	Which TV consumed more energy, this month?
6	Try to find out if exists any issue with the water heater (partition of your choice)
7	If any issue exists, try to understand what is the origin of it and how to resolve it.
8	Compare the water consumption of the current and previous month for the washing machine located in the kitchen
9	Consult the state of any of the appliances.

The first evaluation was conducted with 6 users with diversity of academic backgrounds with ages between 10 and 30 years. In the first step of the evaluation, each participant attended to a brief presentation introducing the Conversational Assistant. The presentation explained its capabilities (obtaining resource consumption, appliance and lights control, and error management) without specifying how to perform these tasks. Additionally, the presentation also included a brief introduction to the virtual home shown in Figure 10, its partitions and virtual devices. Soon after the presentation, each participant performed the required tasks.
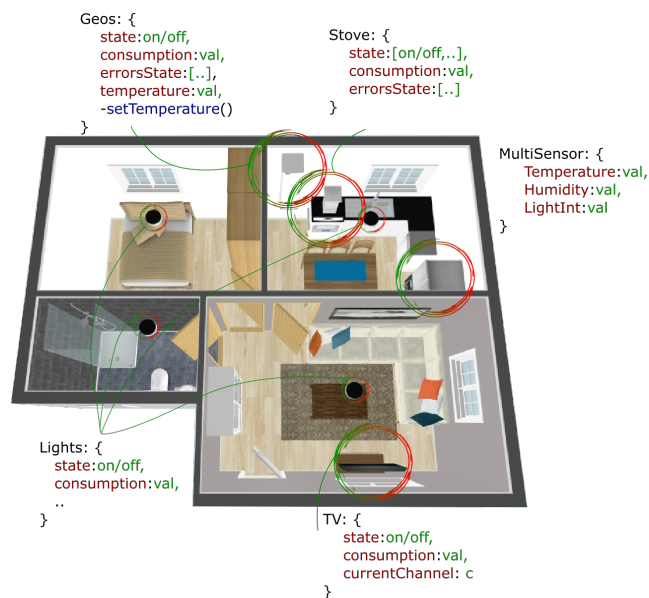
Figure 10 Virtual Smart Home overview.

5.2.2   Evaluation Results

The first evaluation was performed by 6 participants, who did not receive any external help regarding the tasks at hand. While executing each task, the Assistant onboarding mechanisms gave some insights of what the user could ask to obtain certain results.

The results allowed the identification of a few weak points in some modules of the system, namely in the onboarding logic and in the dialogue manager.

Explaining the task to the user without giving him the solution was one of the biggest issues of the evaluation. In some cases, in the first attempt, participants just tried to copy the task description to ask the assistant. The evaluation also showed that most participants gave up easily when they did not get the expected result, while a smaller group tried more than one way to successfully complete the task.

The onboarding system proved to be a valuable help, despite that sentence generation is still rather static. Participants did not explore the full conversational capabilities of the assistant. They often repeated the complete question rather than just completing the missing information. This was more notable in the second and third task, which required more information to make the queries.

6   Conclusion and Future work

This paper presents a first proof-of-concept of a Conversational Assistant for Smart Homes and its first evaluation with users to obtain insights regarding the current strengths and weaknesses and how users explore and use the system capabilities. Despite its stage of development, the Assistant presents unique capabilities and, to best of authors' knowledge, is the first assistant for Smart Homes in (European) Portuguese including dialog capabilities.

Users can interact with the conversational assistant using written or spoken conversation in European Portuguese. The assistant processes the dialog to interpret the commands and requests made by the user. When the user issues a command, the assistant invokes the control service and when the user makes a request it look up in the knowledge base. The proposed ontology provides a structure for the information and enables easy access to it. The ontology is a key component of the system, providing simple methods for the assistant to query the information requested by the user. Using this structure, the information supporting the current implementation of the conversational assistant is stored in a knowledge base, and the ontology was designed to support future additions, such as inhabitants preferences and interactions with the home.

Resulting from the first evaluation, we observed that participants were able to use the system to interact with the house. During the evaluation, some difficulties were perceptible in taking full advantage of the existing features.

Future improvements and new features to the assistant can help users overcome those difficulties, for instance, improving the onboarding system, implementing initiative of the assistant to start a conversation, and adding notifications on critical situations.

─── **References** ───

1   Marjan Alirezaie, Jennifer Renoux, Uwe Köckemann, Annica Kristoffersson, Lars Karlsson, Eva Blomqvist, Nicolas Tsiftes, Thiemo Voigt, and Amy Loutfi. An ontology-based context-aware system for smart homes: E-care@ home. *Sensors*, 17(7):1586, 2017.

2   Suket Arora, Kamaljeet Batra, and Sarabjit Singh. Dialogue System: A Brief Review. *arXiv preprint arXiv:1306.4134*, 2013.

3   Dario Bonino and Fulvio Corno. Dogont-ontology modeling for intelligent domotic environments. In *International Semantic Web Conference*, pages 790–803. Springer, 2008.

4   Dario Bonino, Fulvio Corno, and Luigi De Russis. Poweront: An ontology-based approach for power consumption estimation in smart homes. In *International Internet of Things Summit*, pages 3–8. Springer, 2014.

5   Dario Bonino and Giuseppe Procaccianti. Exploiting semantic technologies in smart environments and grids: Emerging roles and case studies. *Science of Computer Programming*, 95:112–134, 2014.

6   Dan Brickley and Libby Miller. FOAF vocabulary, 2014. URL: `http://xmlns.com/foaf/spec/`.

7   Simon Cox and Chris Little. Time Ontology in OWL, 2017. URL: `https://www.w3.org/TR/owl-time/`.

8   Javier Cuenca. Ontology for Energy Management Applications, 2018. URL: `https://innoweb.mondragon.edu/ontologies/oema/ontologynetwork/1.1/index-en.html`.

9   Aldo   Gangemi.   DUL,   DOLCE+DnS   Ultralite   ontology.   `http://www.ontologydesignpatterns.org/ont/dul/DUL.owl`. Accessed: 2019-03-05.

10   Maksym Ketsmur, Nuno Almeida, António Teixeira, and Samuel Silva. Contribute for an Ontology for Smart Homes and their Conversational Assistants. *Submitted and under review*, 2019.

11   Jangho Lee, Gyuwan Kim, Jaeyoon Yoo, Changwoo Jung, Minseok Kim, and Sungroh Yoon. Training IBM Watson using Automatically Generated Question-Answer Pairs. *arXiv preprint arXiv:1611.03932*, 2016.

12   Cathy Pearl. *Designing Voice User Interfaces: Principles of Conversational Experiences*. O'Reilly, 2016.

13   E. V. Polyakov, M. S. Mazhanov, A. Y. Rolich, L. S. Voskov, M. V. Kachalova, and S. V. Polyakov. Investigation and development of the intelligent voice assistant for the Internet of Things using machine learning. In *2018 Moscow Workshop on Electronic and Networking Technologies (MWENT)*, pages 1–5. IEEE, 2018.

14   Martin Porcheron, Joel E Fischer, Stuart Reeves, and Sarah Sharples. Voice interfaces in everyday life. In *proceedings of the 2018 CHI conference on human factors in computing systems*, page 640. ACM, 2018.

15   Eric Prud'hommeaux and Andy Seaborne. SPARQL Query Language for RDF, 2008. URL: `https://www.w3.org/TR/rdf-sparql-query/`.

16   Nazaraf Shah, Kuo-Ming Chao, Tomasz Zlamaniec, and Adriana Matei. Ontology for Home Energy Management Domain. In *Digital Information and Communication Technology and Its Applications: International Conference, DICTAP*, volume 167, pages 337–347, Dijon, France, June 2011. `doi:10.1007/978-3-642-22027-2_28`.

17   Alexander Smirnov, Alexey Kashevnik, and Andrew Ponomarev. Multi-level self-organization in cyber-physical-social systems: smart home cleaning scenario. *Procedia Cirp*, 30:329–334, 2015.

18   Lorenzo Sommaruga, Antonio Perri, and Francesco Furfari. DomoML-env: an ontology for Human Home Interaction. In *Swap*, volume 166. Citeseer, 2005.

19   Ming Tao, Kaoru Ota, and Mianxiong Dong. Ontology-based data semantic management and application in IoT-and cloud-enabled smart homes. *Future generation computer systems*, 76:528–539, 2017.

20   Ming Tao, Jinglong Zuo, Zhusong Liu, Aniello Castiglione, and Francesco Palmieri. Multi-layer cloud architectural model and ontology-based security service framework for IoT-based smart homes. *Future Generation Computer Systems*, 78:1040–1051, 2018.

21   Juan Ye, Graeme Stevenson, and Simon Dobson. A top-level ontology for smart environments. *Pervasive and mobile computing*, 7(3):359–378, 2011.

22   Steve Young, Milica Gašić, Blaise Thomson, and Jason D Williams. POMDP-based statistical spoken dialog systems: A review. *Proceedings of the IEEE*, 101(5):1160–1179, 2013.

Acquiring Domain-Specific Knowledge for WordNet from a Terminological Database

Alberto Simões   (ORCID)

2Ai–Polytechnic Institute of Cávado and Ave, 4750–810 Barcelos, Portugal
asimoes@ipca.pt

Xavier Gómez Guinovart   (ORCID)

SLI-TALG, Universidade de Vigo, Vigo, Galiza
xgg@uvigo.gal

—— Abstract ——

In this research we explore a terminological database (Termoteca) in order to expand the Portuguese and Galician wordnets (PULO and Galnet) with the addition of new synset variants (word forms for a concept), usage examples for the variants, and synset glosses or definitions.

The methodology applied in this experiment is based on the alignment between concepts of WordNet (synsets) and concepts described in Termoteca (terminological records), taking into account the lexical forms in both resources, their morphological category and their knowledge domains, using the information provided by the WordNet Domains Hierarchy and the Termoteca field domains to reduce the incidence of polysemy and homography in the results of the experiment.

The results obtained confirm our hypothesis that the combined use of the semantic domain information included in both resources makes it possible to minimise the problem of lexical ambiguity and to obtain a very acceptable index of precision in terminological information extraction tasks, attaining a precision above 89% when there are two or more different languages sharing at least one lexical form between the synset in Galnet and the Termoteca record.

1   Introduction

Princeton WordNet (PWN) [12, 26] is undoubtedly one of the most successful resources ever built. Even though it was not developed specifically for natural language processing (NLP), its usage in this field is indispensable. The relevance of this resource in NLP led scientists from all around the world to work on the creation of similar resources for their languages. The main problem on creating those kind of resources is the amount of specialised labour required for this purpose. While the PWN for English was created manually from scratch, most wordnets for other languages are created using automatic methods, followed by a more superficial or in-depth manual analysis.

Galnet [19][1] and PULO [37][2], two wordnets built semi-automatically for Galician and Portuguese, have been enlarged during the last years with a set of experiments of lexical acquisition that explore different resources and methods, extracting data for inclusion in the knowledge databases after expert review and validation [3, 15, 38, 40, 41]

While there is only one wordnet project for the Galician language – Galnet – , for the Portuguese language there are other projects aiming at the creation of WordNet-based linked lexical resources, like Open Multilingual Wordnet (OMW), OpenWordNet-PT, Ufes WordNet or Onto.PT [11].

This document presents a further experiment in order to enrich Galnet and PULO not just with new synset variants, but also with variant usage examples and synset glosses. This experiment explores Termoteca [14], a terminological database that includes records for different areas, ranging from medicine to tourism, and including data for Galician, English, Portuguese, Spanish and French.

The experiment is based on the alignment between concepts included in WordNet (synsets) and concepts described in Termoteca (terminological records). Each entry in Termoteca contains terms that refer to a specific concept, and these concepts are very much like synsets. This alignment between WordNet and Termoteca concepts was performed taking into account the lexical forms already existing in both resources, their morphological category and their knowledge domains.

Our hypothesis is that using specific domain terms, aligned by form, field and category, will reduce the incidence of lexical polysemy (and homography) and will therefore contribute to raising the quality (and precision) level of the extracted information.

The document is organised as follows: Section 2 presents related work, reviewing a number of similar approaches based on lexical resources used to enlarge different languages wordnets. Section 3 describes the specific lexical resources used in the experiment and Section 4 discusses the algorithm designed for their exploitation. In Section 5 the results are evaluated, analysed and commented. The article concludes with Section 6 where some final remarks and future work is presented.

2   Related Work

The Galnet and PULO wordnets have been created from PWN 3.0, following the expand model [42], where the variants associated with the PWN synsets are obtained through different strategies. This model has also been used in the development of the wordnets for Italian [30], Indonesian [33], Hungarian [25], Croatian [34], French – WOLF [36] and WoNeF [32] wordnets – and Kurdish [2]. The same approach has been taken in the MCR framework [20] for the creation of the wordnets of Spanish [6], Catalan [8] and Basque [31].

In the expand model, the main methodology used to extend a wordnet coverage from the variants associated with the PWN synsets is the acquisition of their translations from existing lexical resources. We have applied that methodology in previous phases of the Galnet and PULO developments.

On the one hand, we have used the WN-Toolkit [28] – a set of Python programs for the creation or enlargement of wordnets – to expand the Galnet first distributions from different existing bilingual English–Galician resources: Wikipedia (whose Galician version is known as

---

[1] `http://sli.uvigo.gal/galnet/`
[2] `http://wordnet.pt`

Galipedia)[3], the English–Galician CLUVI Dictionary [4][4], the Apertium English-Galician dictionary[5], the Galizionario (the Galician Wiktionary)[6], Babelnet 2.0[7], the multilingual dictionary OmegaWiki[8], the database of toponyms GeoNames[9], and the catalogue of species Wikispecies[10] [17]. Due to the difficulty of this task, the use of automatic extraction techniques was complemented with an arduous process of human revision where the variant candidates identified by the extraction tool were either approved or rejected one by one by a team of reviewers, but no comparable evaluation measures for precision were provided.

On the other hand, we have designed several lexical extraction experiments aimed to enlarge the coverage of Galnet and PULO from the lexical information contained in classical monolingual dictionaries for the Galician and Portuguese language, using the *Dicionario de Sinónimos do Galego* [13][11], the *Dicionário da Língua Portuguesa Contemporânea* (DLPC) [10] and the *Dicionário Aberto*[12].

In the first case, the methodology used for the extraction from the *Dicionario de Sinónimos do Galego* was based on the matching of lexical forms among the variants of Galnet synsets and the variants of dictionary synsets – i.e. the lexical forms included in each dictionary entry [15, 18, 41]. In this way, and with many nuances and human validation, the variants of a dictionary synset can become variants of a Galnet synset if there is a formal matching between any of the variants included in these two synsets. The highest precision obtained with this method is about 65%, selecting the new candidates among the variants appearing only once in the dictionary of synonyms and in Galnet.

In the second experiment, we present an exploratory approach to enrich the PULO lexical ontology with the synonyms present in the DLPC [40]. The dictionary was converted from PDF into XML and senses were automatically identified and annotated. This allowed us to extract them, independently of definitions, and to create sets of synonyms which are then aligned with the WordNet synsets. We also project the Portuguese terms into English, Spanish and Galician. This process allowed both the addition of new term variants to existing synsets, as the creation of new synsets for Portuguese. An evaluation of synonym extraction based on the selection of 100 random synsets gave a precision of 62% for intersections with two variants in both synsets and 76% for intersections with three variants.

Following a similar methodology, the third experiment [39] explores the entries of the *Dicionário Aberto* in order to extract synsets from its entries. This dictionary includes an interesting way to present definitions based on synonyms. This was exploited to extract synsets from the dictionary (bags of words presented as synonyms). These synsets were intersected with existing synsets in PULO. The experiment got an accuracy of 58% for intersections of two variants from the evaluation of 200 random synsets.

Other related experiments which use bilingual lexical resources to enhance existing wordnets from existing bilingual English dictionaries by intersecting lemmas are conducted for Sanskrit [9], Bengali [35], Croatian [22] or Moroccan Darija [27]. In all the cases, the automatic extraction results are subjected to human revision and validation. In the case of Croatian, the only of these experiments with a true evaluation, the reported precision of the automatic process is about 30%.

---

[3] `http://gl.wikipedia.org`
[4] `http://sli.uvigo.gal/dicionario/`
[5] `http://sourceforge.net/projects/apertium/`
[6] `http://gl.wiktionary.org`
[7] `http://babelnet.org`
[8] `http://www.omegawiki.org`
[9] `http://www.geonames.org`
[10] `http://species.wikimedia.org`
[11] `http://sli.uvigo.gal/dicionario/`
[12] `http://dicionario-aberto.net`

3   Resources

This section describes the three main resources used for this experiment: the terminological database Termoteca, the English, Portuguese and Galician wordnets, as well as the WordNet Domains Hierarchy.

3.1   Termoteca

The Termoteca is a corpus-based Galician-centred multilingual terminological database based on the monolingual and parallel specialty texts collected in the CLUVI Parallel Corpus [16][13] and in the CTG Galician Technical Corpus [1][14].

This terminological database is freely accessible[15] and freely downloadable[16] on the web under a CC-BY 4.0 license. It contains 8,085 records with information about 16,387 terms in Galician (8,172 terms), Spanish (3,257), English (3,031), Portuguese (1,112) and French (815) documented in the CLUVI and CTG corpora, and belonging to the areas of law (1,681 records), sociology (1,145), economy (1,268), ecology (1,673), computer science (564), medicine (1,155) and tourism (1,176)[17].

The information extracted from the corpora and collected in the Termoteca includes the terms, their contexts, and their intra- and inter-linguistic formal variants together with their frequencies of use. Additionally, it includes their definition and their semantic relations (antonyms, holonyms, hyperonyms, etc.) with other terms, when they are explicitly coded in the textual corpora. Finally, all the terminological records are catalogued according to their thematic field, with reference to a conceptual ontology hierarchy.

3.2   PWN, MCR, Galnet and PULO

PWN is a lexical database of the English language, organised as a semantic network where the nodes are concepts represented as sets of synonyms and the links between nodes are semantic relations between lexical concepts. These nodes contain nouns, verbs, adjectives and adverbs grouped by synonymy. In WordNet terminology, a set of synonyms is called a *synset*. The term *variant* applied to WordNet refers to each synonym in a synset, which is considered a lexical variant of the same concept. Thus, each synset represents a distinct lexicalised concept and includes all the synonymous variants of this concept. Additionally, each synset must contain (at least in PWN) a brief definition or gloss, which is common to every variant in the synset, and, in some cases, one or more examples of the use of the variants in context.

Both Galnet for Galician and PULO for Portuguese wordnets are part of the Multilingual Central Repository (MCR) [20][18], a database that currently integrates wordnets from six different languages (English, Spanish, Catalan, Galician, Basque and Portuguese) with PWN 3.0 as Interlingual Index (ILI). Table 1 provides the number of synsets and variants for the different languages gathered in this repository, and their percentage of development with respect to the PWN.

---

[13] `http://sli.uvigo.gal/CLUVI/`

[14] `http://sli.uvigo.gal/CTG/`

[15] `http://sli.uvigo.gal/termoteca/`

[16] `http://sli.uvigo.gal/download/SLI_Termoteca/`

[17] In the Termoteca, each record could be assigned to more than one thematic domain. This is why the total number of assigned domains (8,665) is slightly superior to the number of records (8,085).

[18] `http://adimen.si.ehu.es/web/MCR/`

■ **Table 1** Current coverage relative to English of wordnets in MCR.

| | English (PWN 3.0) | | Galician (Galnet 3.0.28) | |
	variants	synsets	variants	synsets
Total	206,941	117,659	70,030	43,043
%	100%	100%	33.8%	36.6%
	Spanish (MCR 2016)		Portuguese (MCR 2016)	
Total	146,501	78,995	32,604	17,942
%	70.8%	67.1%	15.8%	15.2%
	Catalan (MCR 2016)		Basque (MCR 2016)	
Total	100,793	60,956	50,037	30,263
%	48.7%	51.8%	24.2%	25.7%

It is also worth noting that the concepts contained in the MCR are categorised into domain hierarchies and ontologies, such as the WordNet Domains [7], the Suggested Upper Merged Ontology (SUMO) [29] and the Top Concept Ontology [5], which allows the various applications benefiting from these semantic categorisations to make better use of the resource.

3.3 WordNet Domains

The WordNet Domains hierarchy (WDH) is a freely available[19] lexical resource created in a semi-automatic way by augmenting WordNet synsets with one or more domain labels selected from an original set of 165 hierarchically organised semantic fields. These domains are mainly based on the subject field codes used in lexicography, and on the subject codes from the Dewey Decimal Classification (DDC), a general taxonomy used worldwide for library organisation. For the purposes of this experiment, we use the version of WDH distributed with the MCR resource.

The exploitation of WDH permits reducing word polysemy and grouping the synsets by domain. For instance, the noun *tongue* has eight senses in PWN 3.0, but the first sense *tongue#1* (included in the PWN 3.0 synset with the inter-linguistic offset 05301072-n) is labelled with the domain label ANATOMY, the second sense *tongue#2* (06904171-n) with the label LINGUISTICS, *tongue#3* (13918387-n) with the label FACTOTUM, *tongue#4* (07082198-n) with ART, *tongue#5* (09442595-n) with *Geography*, *tongue#6* (07652995-n) with GASTRONOMY and *tongue#7* (04450994-n) with FASHION.

WDH has been used in several NLP tasks, as word-sense disambiguation [21, 23] or text categorisation [24]. In this experiment, we use WDH to intersect the Termoteca with WordNet by means of the semantic field domains coded in both lexical resources, in order to acquire terminological information from the Termoteca to be incorporated in WordNet.

4 Methodology

As pointed out earlier, the experiment is based on the alignment between the synsets in WordNet and the terminological records in Termoteca, taking into account the lexical forms, their morphological category and their knowledge domains. Compared with other techniques,

---

[19] http://wndomains.fbk.eu

■ **Table 2** Mapping from Termoteca domains to WordNet Domains.

Termoteca	WordNet Domains Hierarchy
ACHEGA (economy)	Administration, Commerce, Industry
AUGA (environment)	Environment, Biology, Agriculture, Earth
GALEX (law)	Law, Administration, Politics, Economy
MEDIGAL (medicine)	Medicine, Psychology, Health
TURIGAL (tourism)	Food, Tourism, Transport
UNESCO (sociology)	Art, Social science, Telecommunication, Politics, Sexuality, Psychological features
XIGA (computing)	Computer science, Telecommunication, Engineering

the methodology applied in this research is characterised by the use of domain information, which is rarely found in the lexical resources employed to extend wordnets.

In order to trace the alignments between the concepts in Termoteca and WordNet, three different experiments were performed (using the same algorithm, but different data):

Experiment 1: Treat each domain independently, both in WordNet and in Termoteca. For this purpose, a rough mapping between the Termoteca domains and the WordNet Domains was created, as presented in Table 2). Note that we just present the top classes of both ontologies, although all sub-domains were used. As an example, the Medicine WDH domain includes all their child domains, like Dentistry or Surgery. In this experiment, for each mapping, we take into account all the WordNet synsets and all the Termoteca records pertaining to any domain in the considered mapping.

Experiment 2: Use the synsets from WordNet for the domains included in Table 2 as a whole, as well as all the Termoteca records. This experiment may produce some relevant inter-domain alignments, but the resulting precision will diminish.

Experiment 3: Use the synsets from WordNet independently of their domain, as well as the whole Termoteca.

Some of the WordNet synsets were discarded from all these experiments:

- All synsets composed exclusively by terms starting with an upper case letter were not considered. The main reason for this is the amount of proper names available in WordNet, ranging from toponyms to anthroponym. As Termoteca does not include proper names, their analysis is not relevant.
- All adverbs were also discarded, as our experiment is focused on terminological information and adverbs are not usually considered in terminology work.

Table 3 shows some statistics about the data and results of the three experiments just mentioned.

The first line includes the number of considered synsets. The next eight lines show the synsets coverage per language.

For instance, looking up the ACHEGA column, the number of considered synsets is 5 172. From these, only 575 synsets include a variant in the four languages, while 2 725 synsets just have variants in the English language.

For each of these experiments, each selected WordNet synset was compared with each selected Termoteca record. An alignment was defined between a synset and a record if at least one variant (for any of the four languages considered) is shared and if their morphological category is the same (verb, noun or adjective).

Table 3 Statistics per experiment: (i) number of considered synsets, (ii) synset coverage per languages, (iii) aligned synsets and alignment score (maximum ↑ and average $\bar{x}$), and (iv) variants, examples and gloss extraction metrics.

| | Experiment 3 | Experiment 2 | Experiment 1 | | | | | | |
	Whole WordNet	Relevant Domains	ACHEGA	AUGA	GALEX	MEDIGAL	TURIGAL	UNESCO	XIGA
Synsets	101 162	43 936	5 172	20 920	3 160	4 710	5 607	6 110	2 024
EN ES GL PT	10 913 10,8%	3 275 7,5%	575 11,1%	922 4,4%	462 14,6%	357 7,6%	404 7,2%	872 14,3%	180 8,9%
EN ES GL	6 667 6,6%	4 310 9,8%	185 3,6%	3 110 14,9%	148 4,7%	314 6,7%	242 4,3%	306 5,0%	228 11,3%
EN GL PT	638 0,6%	209 0,5%	61 1,2%	61 0,3%	53 1,7%	31 0,7%	19 0,3%	48 0,8%	9 0,4%
EN ES PT	5 212 5,2%	1 554 3,5%	271 5,2%	295 1,4%	196 6,2%	248 5,3%	227 4,1%	443 7,3%	80 4,0%
EN GL	17 883 17,7%	9 789 22,3%	686 13,3%	6 448 30,8%	385 12,2%	1 343 28,5%	502 9,0%	734 12,0%	327 16,2%
EN PT	394 0,4%	124 0,3%	5 0,1%	19 0,1%	9 0,3%	61 1,3%	6 0,11%	34 0,6%	2 0,1%
EN ES	10 832 10,7%	4 745 10,8%	664 12,8%	1 588 7,6%	475 15,0%	469 10,0%	923 16,5%	845 13,8%	281 13,9%
EN	48 623 48,1%	19 930 45,4%	2 725 52,7%	8 477 40,5%	1 433 45,4%	1887 40,1%	3284 58,6%	2 828 46,3%	917 45,3%
Aligned Synsets	5 750 5,7%	2 358 5,4%	294 5,7%	195 0,9%	233 7,4%	63 1,3%	172 3,1%	282 4,6%	76 3,8%
Total Alignments	7 654	3 212	343	202	279	67	250	315	81
Align Scores	↑=4.5 $\bar{x}$=1.38	↑=4.5 $\bar{x}$=1.37	↑=3.0 $\bar{x}$=1.17	↑=3.0 $\bar{x}$=1.01	↑=3.0 $\bar{x}$=1.44	↑=2.0 $\bar{x}$=1.07	↑=4.0 $\bar{x}$=1.34	↑=4.0 $\bar{x}$=1.73	↑=4.5 $\bar{x}$=1.83
New GL variants	3 984	1 405	77	26	138	40	61	186	126
New GL examples	8 292	3 402	357	229	317	116	109	370	177
New PT variants	1 198	479	—	—	—	—	196	—	—
New PT examples	1 619	667	—	—	—	—	264	—	—
New GL glosses	652	381	—	27	—	61	—	6	9

Each alignment was then scored according to its alignment strength. A score of 1.0 was assigned for each different language sharing at least one variant. A score of 0.5 was added for each extra variant shared per language. Thus, if two variants for a language are shared, that language alignment will score 1.5. Table 3 presents three lines regarding alignment metrics. The first shows the number of synsets aligned with at least one Termoteca record. The second line shows the total number of alignments. For example, if a synset can be aligned with two different Termoteca records, there are two possible alignments. Finally, the third line presents the maximum score achieved in this experiment, as well as their average.

Every possible alignment is then processed in order to understand what information from Termoteca can be used to enrich PULO or Galnet. Termoteca includes three different kinds of relevant data:

- Portuguese and Galician variants that are not present in PULO or Galnet;
- Galician definitions for synsets missing a Galician gloss[20];
- Galician or Portuguese usage examples, when none is available[21].

Table 3 also presents the amount of items extracted for each one of these categories. The lack of Portuguese suggestions for all experiments – except for those that include TURIGAL – is justified by the fact that Termoteca only includes Portuguese terms and examples for the tourism domain.

5 Results and Evaluation

Considering the number of alignments in the different experiments, and taking into account the amount of extracted information, for the evaluation we only considered the experiment in the tourism domain. The main reason is that it is the only experiment with contributions for both the Portuguese and Galician languages. Although no glosses are suggested by the Termoteca in this knowledge area, our evaluation is based on the concept alignment quality and not in each specific information item extracted. That is, if a form w from terminology entry $\mathcal{T}$ is suggested to be added to the synset $\mathcal{S}$, and even if that form might seem adequate to be added to $\mathcal{S}$, it will only be considered correct if synset $\mathcal{S}$ and entry $\mathcal{T}$ refer to the same concept $\mathcal{C}$. Thus, our evaluation is not based exactly on which information items are contributed to Galnet or PULO, but rather on the alignment of the concepts. When a specific alignment is correct all this record information should be correct given that Termoteca was manually produced.

All the 250 alignments obtained for this domain were manually evaluated. Whereas the number of results is low, it should be noted that there are other alignments from different domains that can be used for Galician and that, if the methodology proves effective, the same methodology can be applied to other terminological resources. Each concept alignment was classified as being (i) correct, (ii) incorrect or (iii) incorrect due to mistakes found in the lexical resources used in the experiment:

(i) Correct Alignments

From the 250 alignments, 160 were classified as correct alignments. Table 4 splits this evaluation by score, showing that when there are two or more common languages between the synset in Galnet and the Termoteca record, the alignment has a precision above 89%.

---

[20] PULO has machine translation glosses for every synset and therefore no definitions were considered for the Portuguese language. Also, Termoteca includes just about 10 records with a Portuguese definition.

[21] As PULO does not include any example, all examples in the Portuguese language were considered.

■ **Table 4** Evaluation metric results for the tourism domain.

Score	Alignments	Correct	Incorrect	Error in resource	Precision
4.0	6	5 (83%)	1 (17%)	0	0.83
3.5	2	2 (100%)	0	0	1.00
3.0	6	5 (83%)	1 (17%)	0	0.83
2.5	4	4 (100%)	0	0	1.00
2.0	39	33 (85%)	4 (10%)	2 (5%)	0.89
1.5	2	2 (100%)	0	0	1.00
1.0	191	109 (57%)	74 (39%)	4 (2%)	0.58

Although both a bilingual link (two variants from different languages) and three monolingual links (three variants from a single language) yield the same score, there are only eight alignments sharing two variants in the same language – and no alignment shares more than two variants in the same language. Furthermore, there are six alignments sharing at least one variant in four different languages, and ten alignments sharing at least one variant in three different languages.

As the number of links decreases to just one language, the precision goes down to 58%, but still comparable with the experiments referred in Section 2, as the evaluations given for those experiments (with precision values of 65%, 62% and 58%) are based on alignments with at least two shared variants. Table 5 shows an example of a correct alignment (with score 1.0).

(ii) Incorrect Alignments
Incorrect alignments are due to the polysemy of lexical forms that are present in two or more different concepts, although sharing the same terminological domain. This is specially true in the tourism knowledge domain, as it intersects with different areas, like history, architecture, leisure, etc. Table 6 shows one such problem, where the form "*nave*" is used to refer both to a type of boat and to a section of a church or cathedral.

(iii) Errors in the Resources
There are three main types of errors found in the used resources: (i) synsets that are classified in the wrong domain (as WDH was expanded automatically to cover all WordNet, some mistakes exist) (ii) wrong variants in MCR wordnets and (iii) Termoteca records with incomplete information. One example of an entry from a different domain is presented in Table 7, where "*colina*" is used both as a biological term and a geographic term. On the other hand, Table 8 shows an example of an incomplete record in Termoteca. In this case, although the alignment is correct, the context of use for the term gathered in Termoteca is not valid.

6 Final Remarks

We presented a methodology to align a terminological database with WordNet at the concept level, with the objective of acquiring domain specific terms, usage examples and definitions to enrich the Portuguese and Galician wordnets.

To validate the proposed methodology we used the MCR (Multilingual Central Repository) wordnets – that include the English, Spanish, Galician and Portuguese wordnets linked at the concept level – , the WordNet Domains hierarchy (WDH) and the Termoteca, a multilingual terminological database focusing on different fields of knowledge.

■ **Table 5** Correct alignment with score 1.0.

ILI: 03594945-n	**Termoteca ID:** 2220741
WordNet gloss:	a car suitable for travelling over rough terrain
New PT variants:	jipe, viatura todo-o-terreno, TT, 4x4, veículo todo-o-terreno, jeep
New PT Examples:	
veículo todo-o-terreno	Só é aconselhável a veículos todo-o-terreno porque a calçada, apesar de curta, é de meados do século XIX e apresenta alguns troços em mau estado.
4x4	Circuito Megalítico de Barbacena (em 4x4).
jeep	Este, pode ser o início de um dos muitos circuitos que, a pé ou de jeep, levam à descoberta da Serra da Lousã.
TT	Acessibilidade: Boa para TT e BTT.
viatura todo-o-terreno	Partindo do Centro de Recepção de Muxagata e de Castelo Melhor, os visitantes do Parque Arqueológico serão transportados em viaturas todo-o-terreno para apreciar pormenorizadamente todo o ciclo artístico.
jipe	O trajecto só se aconselha a quem viajar de jipe ou não se importar de meter o automóvel por maus caminhos.

■ **Table 6** Alignment error.

ILI: 04194289-n	**Termoteca ID:** 2220366
WordNet gloss:	a vessel that carries passengers or freight
New PT Example:	
nave	No interior, a catedral é composta por três naves principais e três capelas.
New GL Example:	
nave	Como manda o canon, son rexos edificios de cantaría, cada un dividido en tres naves, a central abovedada, e cuxa estrutura ternaria queda tamén reflectida nas fachadas das frontes, que teñen tres portas e tres ventás apoiadas nos oportunos arcos de medio punto peraltados.

■ **Table 7** Incorrect alignment due to an error in the WDH.

ILI: 14810561-n	**Termoteca ID:** 2220018
WordNet gloss:	a B-complex vitamin that is a constituent of lecithin; essential in the metabolism of fat
New PT Example:	
colina	O Maciço desenvolve-se em duas grandes unidades morfológicas: a primeira, situada no lado oriental e dominada por um conjunto de colinas dolomíticas formadas a partir dos 300 metros de altitude e onde se distingue a depressão do Rabaçal.

■ **Table 8** Incorrect alignment due to an incomplete record in the Termoteca.

ILI: 03093427-n	**Termoteca ID:** 2220920
WordNet gloss:	diplomatic building that serves as the residence or workplace of a consul
New PT Example:	
consulado	Consulado

The alignment was mainly based on the semantic domain information included in WDH and the Termoteca, thus minimising the problems derived from the lexical ambiguity of the terms. The evaluation of the results in the tourism domain shown that the methodology has a very valuable precision, even when using no more than one term to link the concepts. Currently, the obtained resources are being manually validated and added to their respective wordnets.

Taking advantage of the high precision obtained in this approach, we are planning the alignment of WordNet with other relevant terminological databases, like bUSCatermos[22] for Galician and IATE[23] for Portuguese.

References

1 Rodrigo Agerri, Xavier Gómez Guinovart, German Rigau, and Miguel Anxo Solla Portela. Developing New Linguistic Resources and Tools for the Galician Language. In *Eleventh International Conference on Language Resources and Evaluation (LREC)*, pages 2322–2325, 2018.

2 Purya Aliabadi, Mohamed Sina Ahmadi, Shahin Salavati, and Kyumars Sheykh Esmaili. Towards building KurdNet, the Kurdish WordNet. In *7th Global Wordnet Conference (GWC)*, pages 1–6, 2014.

3 María Álvarez de la Granja, Xosé María Gómez Clemente, and Xavier Gómez Guinovart. Introducing Idioms in the Galician WordNet: Methods, Problems and Results. *Open Linguistics*, 2(1):253–286, 2016. `doi:10.1515/opli-2016-0012`.

4 Alberto Álvarez Lugrís and Xavier Gómez Guinovart. Lexicografía bilingüe práctica basada en corpus: planificación y elaboración del Dicionario Moderno Inglés-Galego. In *Lexicografía de las lenguas románicas: Aproximaciones a la lexicografía moderna y contrastiva*, pages 31–48, 2014. `doi:10.1515/9783110310337.31`.

5 Javier Álvez, Jordi Atserias, Jordi Carrera, Salvador Climent, Antoni Oliver, and German Rigau. Consistent Annotation of EuroWordNet with the Top Concept Ontology. In *4th Global WordNet Conference (GWC)*, 2008.

6 Jordi Atserias, Salvador Climent, Xavier Farreres, German Rigau, and Horacio Rodriguez. Combining multiple methods for the automatic construction of multilingual WordNets. In *Recent Advances in Natural Language Processing II. Selected papers (RANLP)*, volume 97, pages 327–338, 1997.

7 Luisa Bentivogli, Pamela Forner, Bernardo Magnini, and Emanuele Pianta. Revising Word-Net Domains Hierarchy: Semantics, Coverage, and Balancing. In *COLING Workshop on Multilingual Linguistic Resources*, pages 101–108, 2004.

8 Laura Benítez, Sergi Cervell, Gerard Escudero, Mònica López, German Rigau, and Mariona Taulé. Methods and tools for building the Catalan WordNet. In *ELRA Workshop on Language Resources for European Minority Languages*, 1998.

9 Sudha Bhingardive, Tanuja Ajotikar, Irawati Kulkarni, Malhar Kulkarni, and Pushpak Bhattacharyya. Semi-Automatic Extension of Sanskrit Wordnet using Bilingual Dictionary. In *7th Global Wordnet Conference (GWC)*, pages 324–329, 2014.

10 João Malaca Casteleiro, editor. *Dicionário da Língua Portuguesa Contemporânea*. Academia das Ciências de Lisboa, Verbo, 2006.

11 Valeria de Paiva, Livy Real, Hugo Gonçalo Oliveira, Alexandre Rademaker, Cláudia Freitas, and Alberto Simões. An overview of Portuguese WordNets. In Verginica Barbu Mititelu, Corina Forăscu, Christiane Fellbaum, and Piek Vossen, editors, *8th Global WordNet Conference (GWC2016)*, pages 74–81, 2016.

---

[22] `https://aplicacions.usc.es/buscatermos/publica/index.htm`
[23] `https://iate.europa.eu`

12   Christiane Fellbaum, editor. *WordNet: An electronic lexical database*. MIT Press, Cambridge, 1998.

13   Xosé María Gómez Clemente, Xavier Gómez Guinovart, and Alberto Simões. *Dicionario de sinónimos do galego*. Xerais, Vigo, 2015.

14   Xavier Gómez Guinovart. A hybrid corpus-based approach to bilingual terminology extraction. In *Encoding the past, decoding the future: corpora in the 21st Century*, pages 147–175, 2012.

15   Xavier Gómez Guinovart. Do dicionario de sinónimos á rede semántica: fontes lexicográficas na construción do WordNet do Galego. In *XV Colóquio de Outono: As humanidades e as ciências: disjunções e confluências*, pages 331–358, 2014.

16   Xavier Gómez Guinovart. Enriching parallel corpora with multimedia and lexical semantics: From the CLUVI Corpus to WordNet and SemCor. In *Parallel Corpora for Contrastive and Translation Studies: New resources and applications*, pages 141–158. John Benjamins, Amsterdam, 2019. `doi:10.1075/scl.90.09gom`.

17   Xavier Gómez Guinovart and Antoni Oliver. Methodology and evaluation of the Galician WordNet expansion with the WN-Toolkit. *Procesamiento del Lenguaje Natural*, 53:43–50, 2014.

18   Xavier Gómez Guinovart and Miguel Anxo Solla Portela. O dicionario de sinónimos como recurso para a expansión de WordNet. *Linguamática*, 6(2):69–74, 2014.

19   Xavier Gómez Guinovart and Miguel Anxo Solla Portela. Building the Galician wordnet: methods and applications. *Language Resources and Evaluation*, 52(1):317–339, 2018. `doi: 10.1007/s10579-017-9408-5`.

20   Aitor Gonzalez-Agirre, Egoitz Laparra, and German Rigau. Multilingual Central Repository version 3.0. In *8th International Conference on Language Resources and Evaluation (LREC)*, pages 2525–2529, 2012.

21   Sopan Govind Kolte and Sunil G. Bhirud. Word Sense Disambiguation Using WordNet Domains. In *1st International Conference on Emerging Trends in Engineering and Technology*, pages 1187–1191, July 2008. `doi:10.1109/ICETET.2008.231`.

22   Matea Filko Krešimir Šojat and Antoni Oliver. Further expansion of the Croatian WordNet. In *9th Global WordNet Conference (GWC)*, 2018.

23   Wei Jan Lee and Edwin Mit. Word Sense Disambiguation by using domain knowledge. In *International Conference on Semantic Technology and Information Retrieval*, pages 237–242, June 2011. `doi:10.1109/STAIR.2011.5995795`.

24   Angela Locoro, Daniele Grignani, and Viviana Mascardi. When You Doubt, Abstain: From Misclassification to Epoché in Automatic Text Categorisation. In *IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology*, volume 3, pages 209–212, August 2011. `doi:10.1109/WI-IAT.2011.65`.

25   Márton Miháltz, Csaba Hatvani, Judit Kuti, György Szarvas, János Csirik, Gábor Prószéky, and Tamás Váradi. Methods and results of the Hungarian wordnet project. In *4th Global WordNet Conference*, pages 387–405, 2008.

26   George A. Miller, Richard Beckwith, Christiane Fellbaum, Derek Gross, and Katherine Miller. WordNet: An on-line lexical database. *International Journal of Lexicography*, 3:235–244, 1990.

27   Khalil Mrini and Francis Bond. Building the Moroccan Darija WordNet (MDW) using Bilingual Resources. In *International Conference on Natural Language, Signal and Speech Processing (ICNLSSP)*, 2017.

28   Antoni Oliver. WN-Toolkit: Automatic generation of wordnets following the expand model. In *7th Global Wordnet Conference (GWC)*, pages 7–15, 2014.

29   Adam Pease, Ian Niles, and John Li. The Suggested Upper Merged Ontology: A Large Ontology for the Semantic Web and its Applications. In *Working Notes of the AAAI-2002 Workshop on Ontologies and the Semantic Web*, 2002.

30   Emanuele Pianta, Luisa Bentivogli, and Christian Girardi. MultiWordNet. developing an aligned multilingual database. In *1st International WordNet Conference*, pages 293–302, 2002.

31 Elisabete Pociello, Eneko Agirre, and Izaskun Aldezaba. Methodology and Construction of the Basque WordNet. *Language Resources and Evaluation*, 45(2):121–142, 2011. `doi:10.1007/s10579-010-9131-y`.

32 Quentin Pradet, Gaël de Chalendar, and Jaume Baguenier Desormeaux. WoNeF, an improved, expanded and evaluated automatic French translation of WordNet. In *7th Global WordNet Conference (GWC)*, pages 32–39, 2014.

33 Desmond Darma Putra, Abdul Arfan, and Ruli Manurung. Building an Indonesian WordNet. In *2nd International MALINDO Workshop*, 2008.

34 Ida Raffaeli, Bekavac Božo, Željko Agić, and Marko Tadić. Building Croatian WordNet. In *4th Global WordNet Conference (GWC)*, pages 349–359, 2008.

35 K.M. Tahsin Rahit, Tabin Hasan, Md.Al Amin, and Zahiduddin Ahmed. BanglaNet: Towards a WordNet for Bengali language. In *9th Global WordNet Conference (GWC)*, 2018.

36 Benoît Sagot and Darja Fišer. Building a free French wordnet from multilingual resources. In *OntoLex*, pages 14–19, 2008.

37 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 *Lecture Notes in Computer Science*, pages 239–248, 2014.

38 Alberto Simões and Xavier Gómez Guinovart. Extending the Galician wordnet using a multilingual Bible through lexical alignment and semantic annotation. In *7th Symposium on Languages, Applications and Technologies (SLATE 2018)*, volume 62 of *OpenAccess Series in Informatics (OASIcs)*, pages 14:1–14:13, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. `doi:10.4230/OASIcs.SLATE.2018.14`.

39 Alberto Simões and José João Almeida. Experiments on Enlarging a Lexical Ontology. In *Languages, Applications and Technologies*, volume 563 of *Communications in Computer and Information Science*, pages 49–56. Springer International Publishing, 2015. `doi:10.1007/978-3-319-27653-3_5`.

40 Alberto Simões, Xavier Gómez Guinovart, and José João Almeida. Enriching a Portuguese WordNet using Synonyms from a Monolingual Dictionary. In *9th International Conference on Language Resources and Evaluation (LREC)*, May 2016.

41 Miguel Anxo Solla Portela and Xavier Gómez Guinovart. Ampliación de WordNet mediante extracción léxica a partir de un diccionario de sinónimos. In *Actas de las V Jornadas de la Red en Tratamiento de la Información Multilingüe y Multimodal*, volume 1199, pages 29–32. CEUR Workshop Proceedings (CEUR-WS.org), 2014.

42 Piek Vossen, editor. *EuroWordNet: A multilingual database with lexical semantic networks*. Kluwer Academic Publishers, Norwell, 1998.

Definite Clause Grammars with Parse Trees: Extension for Prolog

Falco Nogatz

University of Würzburg, Department of Computer Science,
Am Hubland, 97074 Würzburg, Germany
falco.nogatz@uni-wuerzburg.de

Dietmar Seipel

University of Würzburg, Department of Computer Science,
Am Hubland, 97074 Würzburg, Germany
dietmar.seipel@uni-wuerzburg.de

Salvador Abreu

LISP, Department of Computer Science, University of Évora, Portugal
spa@uevora.pt

Abstract

Definite Clause Grammars (DCGs) are a convenient way to specify possibly non-context-free grammars for natural and formal languages. They can be used to progressively build a parse tree as grammar rules are applied by providing an extra argument in the DCG rule's head. In the simplest way, this is a structure that contains the name of the used nonterminal. This extension of a DCG has been proposed for natural language processing in the past and can be done automatically in Prolog using term expansion.

We extend this approach by a meta-nonterminal to specify optional and sequences of nonterminals, as these structures are common in grammars for formal, domain-specific languages. We specify a term expansion that represents these sequences as lists while preserving the grammar's ability to be used both for parsing and serialising, i.e. to create a parse tree by a given source code and vice-versa. We show that this mechanism can be used to lift grammars specified in extended Backus–Naur form (EBNF) to generate parse trees. As a case study, we present a parser for the Prolog programming language itself based only on the grammars given in the ISO Prolog standard which produces corresponding parse trees.

1 Introduction

The Logic Programming language Prolog has a long history in natural language parsing, and so have *Definite Clause Grammars* (DCGs). Since its introduction by Alain Colmerauer in the early 1970s [6], Prolog has been developed with a focus on *natural language processing* (NLP). This led to *Metamorphosis Grammars* [5] in 1978, a first framework based on first-order logic to parse French. Its rewriting rule mechanism led to the development of DCGs in 1980 [11]. Unlike the established extended Backus–Naur form (EBNF), DCGs come with logical variables and brought all of Prolog's built-in capabilities to drive the parsing process, they could therefore also be used to describe non-context-free grammars.

Besides their use for natural languages, DCGs are the means of choice for parsing and serialising formal languages and data exchange formats in Prolog. For instance, Swi-Prolog's *library(http/html_write)*, which is used to parse and generate HTML fragments, relies on DCGs. A new application to describe formal languages using DCGs arose with the introduction of quasi-quotations [16] in Swi-Prolog. This feature allows the embedding of any external domain-specific language (DSL) directly within Prolog code. For instance, Nogatz et al. used this approach with quasi-quotations to add support for *GraphQL* to Swi-Prolog [9]. The DSL is parsed by a DCG at compile time and replaced by the generated parse tree. It is likely that the embedding of external DSLs in Prolog will become even more popular with the integration of quasi-quotations, increasing the need for a mechanism that creates the corresponding Prolog term for the embedded string based on the DSL's grammar.

The remainder of the paper is organised as follows. In Section 2, we introduce the working with grammars in Prolog. As most formal languages are specified in EBNF, we compare this notation with DCGs, Prolog's de-facto standard for grammars. Our proposed modified term expansion is introduced in Section 3. As an example application, we present in Section 4 a parser for Prolog source code which uses a generative grammar that is based on the definitions in the ISO Prolog standard. In Section 5, we present existing extensions to the DCG formalism and argue about the relative merits of this approach when compared to others. Finally, we conclude with a summary and outlook of future applications in Section 6.

2 Grammars in Prolog

In this section, we first shortly introduce the extended Backus–Naur form, which is the de-facto standard to describe formal languages, including ISO Prolog. We then present the syntax and semantics of DCGs in Prolog as well as their usage and translation in Swi-Prolog.

2.1 Extended Backus–Naur Form

EBNF is a notation to formally describe grammars with production rules. The list of EBNF rules consists of nonterminals and symbols (terminals). Symbols are typically alphanumeric characters, punctuation marks, etc., specified in quotation marks.

Each rule has three parts: a *left-hand side* (LHS) of just a single nonterminal, a *right-hand side* (RHS) consisting of nonterminals and symbols, and the = symbol, which separates the two sides and reads as "is defined as". The elements of the RHS either describe an ordered sequence (denoted by commas ,) or alternative choices (denoted by vertical bars |, with a smaller precedence than the ordered sequence). Repetitions are enclosed by curly brackets { ... }, optional nonterminals by square brackets [...], and comments by brackets of the form (∗ ... ∗).

As a motivational example in this paper, we consider in Figure 1 an extract of the ISO Prolog standard that specifies the syntax of a variable token [1, Sec. 6]. A Prolog variable is either the anonymous variable given by the underscore character, or a named variable which has to start with an underscore character or capital letter. For instance, "_" is the anonymous variable, and "_a" and "A" are named variables. The comments refer to the sections of the ISO Prolog standard where the nonterminals are defined.

2.2 Syntax of DCGs

DCGs are not yet part of the ISO Prolog standard [1], but are under consideration for inclusion in the future [2]. Nevertheless, as of today DCGs are supported by all major Prolog implementations, including Swi-Prolog [14].

$$
\begin{array}{rcl}
\textit{variable token} & = & \textit{anonymous variable } (*\ 6.4.3\ *) \\
& | & \textit{named variable } (*\ 6.4.3\ *)\ ; \\
\textit{anonymous variable} & = & \textit{variable indicator char } (*\ 6.4.3\ *)\ ; \\
\textit{named variable} & = & \textit{variable indicator char } (*\ 6.4.3\ *), \\
& & \textit{alphanumeric char } (*\ 6.5.2\ *), \\
& & \{\ \textit{alphanumeric char } (*\ 6.5.2\ *)\ \} \\
& | & \textit{capital letter char } (*\ 6.5.2\ *), \\
& & \{\ \textit{alphanumeric char } (*\ 6.5.2\ *)\ \}\ ; \\
\textit{variable indicator char} & = & \textit{underscore char } (*\ 6.5.2\ *)\ ; \\
\textit{underscore char} & = & "\_"\ ; \\
\textit{capital letter char} & = & "A"\ |\ "B"\ |\ "C"\ |\ \dots\ ; \\
\textit{alphanumeric char} & = & \dots
\end{array}
$$

■ **Figure 1** EBNF rules for a variable token in Prolog.

In its simplest form to just specify context-free grammars, DCGs are very similar to EBNF. Again, a DCG is a list of *grammar rules*, consisting of a LHS and a RHS. Instead of =, the Prolog operator `-->/2` is used in between.[1] For compatibility with EBNF, the vertical bar `|/2` can be used to denote alternatives, but Prolog's traditional disjunction `;/2` is also supported.

Compared to EBNF, DCGs provide three major extensions, resulting in the ability to describe possibly non-context-free grammars:

- *Arguments in the LHS.* In contrast to EBNF, the nonterminal on the LHS of a DCG is allowed to have any number of arguments of any type. Since it is common in Prolog to use the same variables for input and output, these additional arguments can also be used to describe the corresponding parse tree, that either is used as input while serialising or gets generated while parsing.

- *Complex control structures in the RHS.* Besides the *conjunction* `,/2` and the *disjunction* `|/2` and `;/2`, the Prolog control structures for *If-Then/-Else* `->/2` and *not* `\+/1` can be used to express relationships between items in the RHS. *Parentheses* `(...)` and Prolog's *cut* operator `!/0` can be used as usual. In addition, any Prolog code can be embedded by using curly brackets `{...}`.

- *Pushback arguments.* DCGs allow the definition of rules in the form "`H, P --> B1, ..., Bn`", with `P` being a list of terminals that are prepended to the parsed list after successfully evaluating the grammar's body [2]. As we do not use it in our application, it is mentioned here only for completeness.

The arguments in the LHS are also often used together with embedded Prolog code in the RHS to condition the application of a rule or alternative by some provided options. For instance, as of version 7.3.27 Swi-Prolog provides a flag `var_prefix` [15, Sec. 2.16.1] that

---

[1] In the rest of this paper, we will use the notation `A/N` to denote a Prolog operator with the name `A` and an arity of `N`. In contrast, a nonterminal `A` with `N` arguments is denoted by `A//N`.

restricts the syntax of variables. Given `var_prefix(true)`, only variables starting with the underscore character are allowed, i.e. identifiers starting with a capital letter denote atoms.[2]

We can reproduce this behaviour by extending the DCG's `named_variable` LHS by the argument `Flags` and put the condition to check for `var_prefix(false)` in front of the nonterminals on the RHS:

```
1  named_variable(Flags) -->
2    { option(var_prefix(false), Flags) },
3    capital_letter_char, ...
```

As usually in Prolog, comments are written as `/* ... */`. Unlike EBNF, DCG provides no pre-defined syntax neither for optional nonterminals nor repetitions.

2.3 EBNF as an Internal DSL in Prolog

EBNF and DCG are already very similar in their syntax. In fact, the example of Figure 1 can be embedded directly into Prolog with only minor modifications:

– Nonterminals in DCGs must be valid Prolog atoms, so included whitespaces have to be replaced, e.g., by underscore characters.
– Comments are written as `/* ... */` instead of `(* ... *)`.[3]
– The very last rule have to end with a dot `"."`.

Because Swi-Prolog and YAP allow the definition of the *block operators* `[]/1` and `{}/1` [15, Sec. 5.3.3], this slightly modified EBNF is already valid Prolog syntax. However, to not confuse them with Prolog's list notation and DCG's embedded Prolog code, we write optional elements as `?c` instead of `[ c ]`, and sequences as `*c` instead of `{ c }`, with `?/1` and `*/1` defined as prefix operators.

As of Swi-Prolog version 7, text enclosed in double quotes is read as objects of type *string*. Using the Prolog directive `:- set_prolog_flag(double_quotes, chars).` this can be changed, so that double-quoted text is read as a list of characters. As a result, terminals in DCGs can be written as strings enclosed in double quotes as in EBNF.

Prolog provides a mechanism to rewrite Prolog code at compilation time, similar to macros in other programming languages. This is called *term expansion*. When loading code into Swi-Prolog, its compiler calls the predicate `expand_term/2` on each term read from the input. As part of it, `term_expansion/2` is executed to apply user-defined term expansions first. With term expansion, EBNF can be translated into normal DCG notation at compile time. For instance, the single EBNF rule *underscore_char = "_"* gets replaced by the following Prolog fact:

```
1  user:term_expansion(A = B, [A --> B]).
```

With similar expansions for "|" (alternatives), ";" (rule endings), "?" (optional elements), and "$*$" (sequences), EBNF becomes an internal DSL in Prolog. As a result, grammars of formal languages provided as EBNF can be directly used in Prolog, resulting in executable parsers.

---

[2] This has been introduced for compatibility with Prolog by BIM. It has proven to be useful for defining internal domain-specific languages in Prolog that require identifiers to start with an uppercase letters, e.g., for RDF.

[3] Note that ehe term `(* ... *)` is valid Prolog syntax when `*/1` is defined as both a prefix and postfix operator, since every Prolog term is allowed to be bracketed. However, this would require a comma in front of the comment, since a bracketed term is only allowed as an argument.

2.4 Procedural Semantics of DCGs and its Implementation

Essentially, the DCG notation is only syntactic sugar for writing Prolog predicates that operate on difference lists. In most Prolog implementations, DCGs are translated into normal Prolog code at compilation time using term expansion and the pre-defined predicate `dcg_translate_rule/2`. It adds the two arguments which are normally hidden by the DCG notation.

For instance, consider the definition of the second alternative for named variables of Figure 1 that describes a named variable beginning with a capital letter. It is only allowed for the `var_prefix` flag set to `false` (cf. Section 2.2). Using appropriate term expansions as presented in Section 2.3, the equivalent DCG is as follows:

```
1  named_variable(Flags) -->
2    { option(var_prefix(false), Flags) },
3    capital_letter_char, /* 6.5.2 */
4    *alphanumeric_char.  /* 6.5.2 */
```

For every nonterminal A//N, two additional arguments `S` and `R` are added, resulting in a Prolog predicate A/(N+2) with `S = [C|R]`, and `C` the list of symbols consumed by the rule's body. Every body item operates on the result of the previous one. If there is embedded Prolog code given in curly brackets, it is inserted at the specified position. I.e., the aforementioned DCG for `named_variable//1` gets expanded to the Prolog predicate `named_variable/3`:

```
1  named_variable(Flags, A, C) :-
2    option(var_prefix(false), Flags),
3    capital_letter_char(A, B),
4    *(alphanumeric_char, B, C).
```

The generated Prolog predicates can be directly used. For instance, the following call consumes all possible prefixes of the string "*Abc.D*" that are valid named variables and returns the rest:

```
1  ?- Flags=[var_prefix(false)], L="Abc.D", named_variable(Flags,L,R).
2  R = "bc.D" ; % first solution
3  R = "c.D" ;  % second solution
4  R = ".D" .   % last solution because "." is no alphanumeric character
```

Prolog backtracks over the three possibilities for the sequence of `alphanumeric_char//0`, beginning with the empty string. Following Prolog's *SLD resolution* mechanism, the rules are tried in their order of appearance. With Prolog's *backtracking* mechanism, multiple rules with compatible LHSs will be tried. For the rest of the paper, we assume a basic knowledge of these two fundamentals of Prolog.

3 Modified Term Expansion for Parse Tree Generation

The practical benefits of the DCG presented in Section 2.4 are very limited – the grammar can only be used to check if a given string can be parsed by the grammar. Even generating all allowed variable names is narrow, because Prolog's SLD resolution first backtracks over the sequence of `alphanumeric_char//0`, i.e. it generates the possible variable names in the order of "*Aa*", "*Aaa*", "*Aaaa*" instead of "*Aa*", "*Ab*", "*Ac*".

For practical use, the application of the grammar shall generate the corresponding parse tree on-the-fly. In the field of natural language processing, it has been proposed to extend the DCG's LHS by an additional argument that holds the parse tree. Following the ideas proposed

by Abramson and Dahl [3], the additional parse tree argument for a DCG rule `H --> B` is a term of the form `H(T)`, with `H` being the name of the rule's head without arguments, and `T` a term whose structure depends on the rule's body `B`. This way, `named_variable//1` becomes `named_variable//2`:

```
1  named_variable(Flags, named_variable(T1,T2)) -->
2    { option(var_prefix(false), Flags) },
3    capital_letter_char(T1), *(alphanumeric_char, T2).
```

This construction method is generic because the additional parse tree argument is constructed based only on the LHS's nonterminal symbol and the structure of the grammar rule's RHS. The extension of an existing DCG can therefore be done automatically at compile time using term expansion.

As part of our contribution, we provide a Swi-Prolog package *library(dcg4pt)* ("DCG for parse trees") which defines a predicate `dcg4pt_rule_to_dcg_rule/2` that takes a DCG as its first argument and returns an equivalent DCG where the nonterminals have been extended by an additional parse tree argument. The library is listed in Swi-Prolog's package list at `http://www.swi-prolog.org/pack/list?p=dcg4pt`. Its source code is published under MIT License at `https://github.com/fnogatz/dcg4pt`. It can be used to get the extended version of every DCG rule at first, and translate the result afterwards using Prolog's built-in predicate `dcg_translate_rule/2` as introduced in Section 2.4:

```
1  :- use_module(library(dcg4pt)).
2  user:term_expansion(H --> B, Rule) :-
3    dcg4pt_rule_to_dcg_rule(H --> B, DCG),
4    dcg_translate_rule(DCG, Rule).
```

3.1   Handling of Optionals and Sequences of Nonterminals

Grammars for formal languages often make great use of optional and sequences of nonterminals. For instance, the nonterminal *named variable* of Figure 1 allows any number of alphanumeric characters, including zero. Another typical use case is the whitespace for indentation in programming languages; newline and whitespace characters can be set arbitrarily and even include comments. This is different from the previous applications in the field of natural language processing as in [3]. There, the number of nonterminals in a grammar rule's RHS is known in advance. As a result, the parse tree argument can have a fixed number of children. E.g., the parse tree is represented as `np(Name)` for a noun phrase that is simply a name; or `np(Det,Noun,Rel)` for a noun phrase that consists of a determiner, noun, and relative clause.

However, when working with optionals and sequences, the number of children is not limited. It is therefore desirable to use a list if there is a conjunction, optional, or sequence on a grammar rule's RHS. The DCG for `named_variable//2` should therefore produce a parse tree of the form `named_variable([T1|T2])`, with `T1` being the parse tree generated by `capital_letter_char//1`, and `T2` the (possibly empty) list of parse trees each generated by `alphanumeric_char//1` in the sequence.

This follows the ideas of the library of [12], which also introduces additional control structures for parsing sequences, such as `sequence(Mode,...)`, that further make the code more compact, readable and declarative, where `Mode` can be one of `'*'`, `'**'`, `'+'`, and `'?'`. The previous library is the basis of our current implementation of *library(dcg4pt)*. It is available in the *Declare* package at `http://www1.pub.informatik.uni-wuerzburg.de/databases/ddbase/` and has been used in an application of DCGs to language processing

for electronic dictionaries in linguistics by Seipel et al. [12], and represents the parse trees in an XML term for Prolog. Instead of using `sequence/2`, *library(dcgt4pt)* defines the prefix operators `?/1` (optional element), `+/1` (non-empty sequence), and `*/1` (possibly empty sequence).

3.2 Term Expansion Scheme

The general formation principles of *library(dcg4pt)* depending on the body of a DCG rule are presented in Table 1. The parse tree is always added as the very last argument, i.e. for a rule's LHS of `h(some)`, the head of the generated DCG rule becomes `h(some, h(...))`.

■ **Table 1** Formation principles to construct the parse tree for a DCG rule "`h --> Body`".

DCG Body	Example DCG	Extended by Parse Tree Argument
Terminal	`h --> "_".`	`h(h('_')) --> "_".`
Nonterminal	`h --> a.`	`h(h(A)) --> a(A).`

Conjunction	`h --> a , b.`	`h(h(R0)) -->`
		`  { R0 = [A|R1] }, a(A),`
		`  { R1 = [B] }, b(B).`

Disjunction	`h --> a	b.`	`h(h(R0)) -->`
			`  { R0 = A }, a(A) ;`
			`  { R0 = B }, b(B).`

Embedded Prolog	`h --> a, { p }.` *Embedded Prolog is ignored for the parse tree generation.*	`h(h(A)) --> a(A), { p }.`
Sequence or Optional	`h --> *a.` *And similar for the prefix operators +/1 and ?/1.* *R is a list.* *In sequence//3, we distinguish whether the DCG is called with bound or unbound arguments.*	`h(h(R)) --> sequence('*',a,R).`

Note that the embedded Prolog code for variable unifications presented in Table 1 is needed to support complex RHSs with combinations of all these structures. For instance, a rule "`h --> a, *b, c`" should not be translated into:

```
1  h(h([A,Bs,C])) --> a(A), sequence('*',b,Bs), c(C).
```

Because `sequence//3` describes a list `Bs`, the generated parse tree for `h//1` would otherwise contain a list of lists. Instead, the following extended DCG is generated by *library(dcg4pt)*:

```
1  h(h(R0)) -->
2    { R0=[A|R1] }, a(A),
3    sequence('*', b, Bs), { append(Bs, R2, R1) },
4    { R2=[C] }, c(C).
```

With no `b//1` being present in the sequence, the resulting parse tree for `h//1` is just `h([A,C])`.

Using this compilation scheme, the extended DCG is capable to generate all possible combinations of strings and corresponding parse trees. For instance, in Listing 1, `variable_token/3` is used to parse an input of `"_a"`. Using backtracking, it returns two solutions. First, only the first symbol `"_"` is consumed, because it represents the anonymous

variable; the remainder `"a"` of the string is bound to the third argument. As a second solution, the string is parsed as a named variable.

■ **Listing 1** Usage example for the generated `variable_token/3` with the input string `"_a"`.

```
1  ?- variable_token(T, "_a", R).
2  % first solution, consuming only "_":
3  R = "a", T = variable_token(anonymous_variable(
4               variable_indicator_char(underscore_char('_')))) ;
5  % second solution, consuming the whole string "_a":
6  R = "",  T = variable_token(named_variable([
7               variable_indicator_char(underscore_char('_')),
8               alphanumeric_char(alpha_char(letter_char(
9               small_letter_char(a) ))) ])) .
```

3.3   Support for Parsing and Serialising

The aim of *library(dcg4pt)*'s term expansion scheme is to create a modified DCG that expresses a relation between the string and parse tree. In particular, the generated Prolog program can also be used "in reverse" to a normal parser, i.e. to serialise a string by a given parse tree. For this purpose it has to be ensured that the term expansion scheme presented in Section 3.2 uses only Prolog predicates that are *pure*, i.e. they can be used no matter which of the arguments are bound. For instance, the aforementioned rule "`h --> a, *b, c`" could also be expanded to use Prolog's built-in predicate `flatten/2`. It calculates a flattened list from a list of lists and therefore also avoids the use of nested lists. However, `flatten(+ListOfLists,-FlattenedList)` can not be used the other way around. As a result, the generated extendend DCG can only be used to parse a given string and return the corresponding parse tree; serialising a given parse tree back to the corresponding string is not possible.

Note that this is an improvement on the previous implementations of [7] and [12]. In addition, because of possibly left-recursive rules, or rules that consume resp. produce no symbols, the expanded rules have to behave differently depending on whether they are called with bound or unbound arguments. For instance, consider the rules that describe a *variable* as presented in Listing 2: it is a *variable_token* optionally prepended by layout characters. *layout_text_sequence* succeeds for any non-empty sequence of whitespace, tab, or newline characters.

■ **Listing 2** Grammar rules for the nonterminal *variable* in Prolog.

```
1  variable = ?layout_text_sequence, variable_token ;
2  layout_text_sequence = layout_text, *layout_text ;
3  layout_text = layout_char | comment ;
4  layout_char = space_char | horizontal_tab_char | new_line_char .
```

For a given string `"  _a"` (the string `"_a"` preceded by two whitespaces), the RHS of *layout_text_sequence* should try to consume as many whitespace characters as possible to avoid unnecessary backtracking. On the other hand, this greedy approach is undesirable when both arguments are unbound, i.e. when generating all allowed strings with their corresponding parse tree. In that case, the smallest possible string should be created at first. The four combinations of an (un)bound string or parse tree are automatically handled by the DCGs generated by our tool. For instance, in the definition of `*//1` (resp. `*/4` after expansion with the arguments for the parse tree, and the lists `S` and `R`) in Listing 3, we test whether

the argument S is a variable. If so, we make use of `sequence('*',_)` that starts with the smallest string, i.e. the empty list, whereas `sequence('**',_)` tries to consume as many symbols as possible from the given argument S. Similar checks have been implemented for the sequences `'?'` and `'+'`, resulting in Prolog programs that can be used both for parsing and serialising, based on a single grammar.

■ **Listing 3** Implementation of the meta-predicate `*/4` to support sequences of nonterminals.

```
1  *(DCGBody, Tree, S, R) :-
2    \+ var(S), !,
3    sequence('**', DCGBody, Tree, S, R).
4  *(DCGBody, Tree, S, R) :-
5    var(S), !,
6    sequence('*', DCGBody, Tree, S, R).
```

4 Case Study: A Parser for Prolog

Quite often parsing is only a single step when working with formal languages. A common use case is program transformations. These require to first parse the program based on a grammar, then generate an abstract syntax tree, modify it, and serialise it again. With our tool, the same language specification – i.e. the same code – can be used for the parsing and serialising steps. They share a single data structure – the parse tree which was automatically added by our tool's modified term expansion –, and the resulting Prolog program can be used in both directions without any modification. Compared to the common approach of using a parser generator like ANTLR [10] instead, our library relieves the programmer from the burden of keeping two tools, for parsing and serialising, in sync.

As a case study, we present the implementation of a grammar for Prolog programs. With the help of our tool *library(dcg4pt)*, the DCG is extended by a hidden argument to store the parse tree. We will show how this generic parse tree can be modified and used by the same grammar to produce a valid Prolog program again. It can be used in Swi-Prolog as the package *library(plammar)*.

The programming language Prolog is specified in the ISO Prolog standard [1]. While most of the syntax is described using EBNF, the ISO standard also contains informally specified requirements which cannot be expressed by context-free-grammars. For instance, it provides grammar rules for the language's tokens, but also states informally:

> *A token shall not be followed by characters such that concatenating the characters of the token with these characters forms a valid token [...].*

This requirement cannot be expressed by a context-free grammar. As the ISO standard contains several similar requirements, parsing Prolog is a prime example for a realistic parser based on DCGs.

Analysing the syntax of a programming language usually requires two phases: (i) the lexical analysis, that converts a sequence of characters into a sequence of tokens, and (ii) the parsing of the tokens in order to generate a structural representation. With DCGs it is possible to write *scannerless parsers* that combine these two steps into a single grammar. However, the ISO standard defines Prolog similarly: it first declares that a Prolog program consists of Prolog terms that are a sequence of tokens, and later defines the grammars for tokens and terms separately. Therefore, our implementation is also split into the two phases. Both make use of grammars but work on lists of different types: the *lexer* handles the program source code as a string and is presented in Section 4.1; the *parser* works with a

list of tokens and is described in Section 4.2. An example on how to use this grammar for a source-to-source transformation is presented in Section 4.3. In Section 4.4 we present the integration of *library(plammar)* into a graphical interface to interactively explore a parse tree.

4.1 Lexical Analysis

The ISO standard specifies the syntax of Prolog in more than 200 EBNF grammar rules. After having defined EBNF as an internal DSL as presented in Section 2.3, the grammar rules can be directly used as a Prolog program.

The beginning of the DCGs as defined in the ISO standard is given in Listing 4. It states that a *term* is a sequence of *token*. A *token* is one of *name*, *variable*, *integer*, *float_number*, etc.

■ **Listing 4** Definition of tokens according to Sec. 6.4 of the ISO Prolog standard [2].

```
1  term = *token ;  % sequence of token
2  token = name | variable | integer | float_number | ht_sep | open |
3          close | open_ct | double_quoted_list | comma | open_list |
4          close_list | open_curly | close_curly .
```

As presented in Section 3, these grammar rules are expanded so that they match the corresponding parse tree. For *token* this is simply a structure `token(I)` with `I` one of `name(...)`, `variable(...)`, and so on, because every element of the choice is a single nonterminal. The EBNF in the ISO standard is deeply structured and thus creates very verbose parse trees. For instance, consider the Prolog term that consists of just the named variable "`_a`" as given as the second solution in Listing 1.

The extended DCG for the nonterminal *term* describes a parse tree of the form `term(I)`. According to our modified term expansion, `I` is always a list of `token(...)`. This is exactly the result one expects from a lexer.

As an example, we consider the implementation of the `member/2` predicate in Prolog:

```
1  member(X,[X|_]).
2  member(X,[_|Xs]) :-
3    member(X,Xs).
```

`member(?Elem,?List)` is true, iff `Elem` is the `List`'s first element (fact in l. 1), or one of the following (recursive rule in ll. 2). The tokenisation of the first line of this implementation is given in Listing 5. Our tool *library(plammar)* provides the Prolog predicate `prolog_tokens(?Source,?Tokens)` that takes Prolog source code and generates the list of tokens, and the other way around. For the `member/1` fact it returns 11 tokens.

■ **Listing 5** Tokenisation of "`member(X,[X|_]).`" as generated by *library(plammar)*.

```
1  ?- use_module(library(plammar)). % load package
2  ?- prolog_tokens(string("member(X,[X|_])."), Tokens).
3  Tokens = [
4    name([name_token(letter_digit_token([small_letter_char(m),...]))]),
5    open_ct(open_token(open_char('('))),
6    variable([variable_token(named_variable([capital_letter_char('X')]))]),
7    comma([comma_token(comma_char(','))]),
8    open_list([open_list_token(open_list_char('['))]),
9    variable(/* as before for X */),
10   ht_sep([head_tail_separator_token(/* shortened */)]),
11   variable([variable_token(anonymous_variable(/* as in Listing 1 */))]),
12   close_list([close_list_token(close_list_char(']'))]),
13   close([close_token(close_char(')'))]),
14   name([name_token(graphic_token([graphic_token_char(graphic_char(.))]))
15  ]) ] .
```

4.2 Parsing

DCGs are not only useful when working with strings. In general, they describe difference lists over any type, and strings are just a special case, since in Prolog they are represented by a list of characters. Thus, the grammar to process the list of tokens generated in the lexical analysis can also be defined by an extended DCG. The ISO standard even specifies valid sequences of tokens as EBNF, as well. For instance, the compound term `member(X,[X|_])` is a valid Prolog term: it is the sequence of an *atom*, the opening parenthesis, a list of arguments (denoted by the nonterminal *arg_list*), and the closing parenthesis. The corresponding grammar rule is presented in Listing 6. Here, elements given in brackets represent terminals, i.e. the parse trees `open_ct(...)` and `close(...)` are elements in the list of tokens generated by the lexer. Using our modified term expansion, this rule describes a parse tree of the form `term([atom(...), open_ct(...), arg_list(...), close(...)])`.

▪ **Listing 6** Definition of compound terms in functional notation according to Sec. 6.3.3 of the ISO Prolog standard [2].

```
1  term(0) = atom, [ open_ct(_) ], arg_list, [ close(_) ] ;
2  atom = [ name(_) ] ;
3  arg_list = arg ;
4  arg_list = arg, [ comma(_) ], arg_list ;
5  arg = term(P), { P < 1000 } .
```

Note that parsing Prolog source code requires one to annotate all terms by their precedence (called *priority* in the ISO standard). A compound term has the highest precedence *zero*, which is specified in the argument of *term*. The nonterminal *term* therefore has a higher arity than the *term* of Listing 4, making the two predicates distinguishable: our term expansion creates a Prolog predicate `term/4` for the former, and `term/3` for the latter.

Given the definitions of Listing 6, the tokens returned by `prolog_tokens/2` (cf. Listing 5) are correctly recognised as a Prolog term in functional notation. The term consists of a function symbol that is an atom – represented by the list element `name(_)` –, followed by the opening parenthesis `open_ct(_)`, the list of arguments *arg_list*, and the closing parenthesis `close(_)`. *arg_list* is a comma-separated list of terms with a precedence lower than 1000. The token `variable(_)` is used as the first argument. The second argument given by the tokens `[open_list(_), variable(_), ht_sep(_), variable(_), close_list(_)]` denotes a list and is parsed by another grammar rule for `term(0)`, which we do not present in detail here.

4.3 Source-to-Source Transformation

The Prolog grammar provided in *library(plammar)* can be used for parsing and serialising. An application would be code reformatting. For instance, let's assume we want to automatically add whitespace characters after every comma in the *arg_list* of a compound term. Given a Prolog predicate `transform_parse_tree/2`, we can apply the source-to-source transformation by just combining the created `term/3` and `term/4` predicates, as they can be used in both directions:

```
1  ?- term(term(ListOfTokens_1), Old, []),        % lexical analysis
2     term(Prec, AST_1, ListOfTokens_1, []),      % parsing
3     transform_parse_tree(AST_1, AST_2),         % transformation
4     term(Prec, AST_2, ListOfTokens_2, []),      % serialise tokens
5     term(term(ListOfTokens_2), New, []).        % serialise string
```

In *library(plammar)*, we provide Prolog predicates to ease the work with tokens, parse trees, and abstract syntax trees: `prolog_parsetree/2`, `parsetree_ast/2`, and `prolog_ast/2`. They have been developed with a focus on their usage as relations and can handle the input of only the Prolog source code, only the tokens, parse tree, or abstract syntax tree, or both arguments being unbound.

4.4 Integration into Graphical AST Explorer

As part of our contribution we integrated the Prolog parser *library(plammar)* into `https://astexplorer.net/`, an open source web application that provides parsers for several programming languages, including PHP, JavaScript, and SQL. Figure 2 presents the graphical representation of the generated parse tree for the Prolog program that defines the `member/2` predicate.
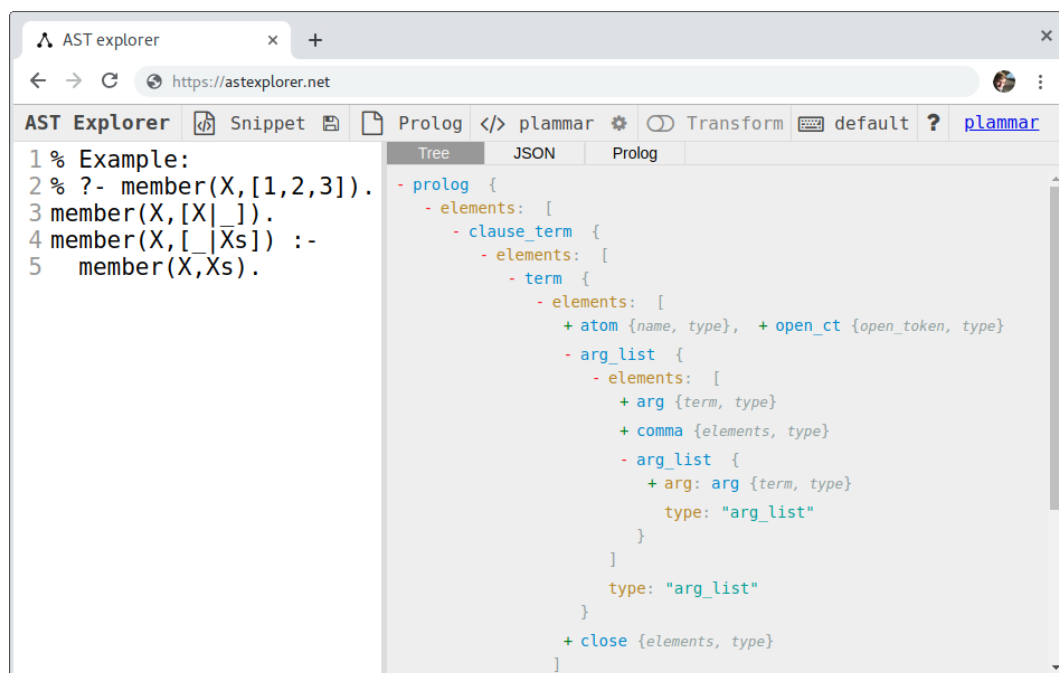
Figure 2 Integration of the Prolog parser into `https://astexplorer.net/`.

5 Related Work

Since its introduction by A. Colmerauer, the logic programming language Prolog was developed with a focus on natural language processing. This resulted in a first representation of grammars as clauses of first-order logic in 1975 by Colmerauer [4, 5]. Definite Clause Grammars were introduced by Pereira and Warren in 1980 [11]. As a usage example of extra arguments in nonterminals, they manually extend rules that parse sentences by their corresponding *building structures* – a term holding information about the applied rule and its RHS elements.

This idea is adopted by Dahl and McCord in 1983 [7]. Their *modifier structure grammars* extend a grammar with two additional arguments to obtain a meaning representation (called *semantic structure*), and its corresponding *syntactic structure* in form of a parse tree. Simultaneously and independently, *restriction grammars* were developed by Hirschman and

Puder [8], which also automatically create parse trees. An overview of these approaches is given in [3, Chapters 7–8], where the idea of hiding the parse tree argument from the user is discussed.

The aforementioned approaches are focused on context-free grammars. In particular, they do not make use of embedded Prolog on a rule's RHS, and higher-order structures like sequences. Although they expand grammar rules by an additional argument to store a parse tree, its actual construction is not specified. Hence, we have observed that they do not address the challenges that arise when grammar rules that consume resp. produce no symbols are called with unbound arguments. This is a requirement for grammars that are to be used for both parsing and serialising.

In [13], DCGs are again augmented with hidden additional arguments. Instead of generating parse trees, they allow to define multiple accumulators, e.g., to calculate and store the size of the consumed symbols. Accumulators are defined using Prolog predicates. It might be possible to use this technique to define a hidden accumulator that creates the corresponding parse tree, though to the best of our knowledge this has not yet been done.

6 Conclusion

The development of Definite Clause Grammars was long driven by a focus on natural language processing. As of today, other techniques have gained popularity in this area. In this paper, we emphasise the usefulness of DCGs for the work with formal languages, as DCGs provide a unified mean to specify both a parser and serialiser. We took up again the idea of extending DCGs by a hidden argument to store the corresponding parse tree, and presented a generalised formation principle that supports optional and sequences of nonterminals and provides optimisations depending on which arguments are bound. Our tool can be used with any existing DCG, resulting in a generative grammar. It is published as the SWI-Prolog package *library(dcg4pt)*.[4]

As an example application, we implemented a generative grammar for Prolog source code, bundled as *library(plammar)*.[5] Because Prolog's syntax is defined by more than 200 grammar rules in EBNF, we implemented EBNF as an internal domain-specific language for Prolog. This enabled an easy adoption of the ISO Prolog standard with only minor modifications.

The tool for automatic parse tree generation will become useful for implementing external domain-specific languages in Prolog, in particular using quasi-quotations. On the other hand, there are several future applications for the Prolog parser: besides source-to-source transformation for refactoring, it can be used for static source code analysis, e.g., to find all predicates that are available only in some Prolog systems.

References

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

2 ISO/IEC 13211-3:2006: Definite clause grammar rules, 2015.

3 Harvey Abramson and Veronica Dahl. *Logic Grammars. Symbolic Computation.* Springer, 1989.

4 Alain Colmerauer. Les grammaires de métamorphose. Technical report, Groupe d'Intelligence Artificielle, Université de Marseille-Luminy, 1975.

[4] `https://github.com/fnogatz/dcg4pt` (MIT License)
[5] `https://github.com/fnogatz/plammar` (MIT License)

5   Alain Colmerauer. Metamorphosis grammars. In *Natural language communication with computers*, pages 133–188. Springer, 1978.

6   Alain Colmerauer. An Introduction to Prolog III. *Communications of the ACM*, 33(7):69–90, 1990. `doi:10.1145/79204.79210`.

7   Veronica Dahl and Michael C. McCord. Treating coordination in logic grammars. *American Journal of Computational Linguistics*, 9(2):69–80, 1983.

8   Lynette Hirschman and Karl Puder. Restriction grammar: A Prolog implementation. *Logic Programming and its Applications*, pages 244–261, 1985.

9   Falco Nogatz and Dietmar Seipel. Implementing GraphQL as a Query Language for Deductive Databases in SWI–Prolog Using DCGs, Quasi Quotations, and Dicts. In *Proc. 30th Workshop on Logic Programming (WLP)*, 2016.

10   Terence J. Parr and Russell W. Quong. ANTLR: A predicated-LL (k) parser generator. *Software: Practice and Experience*, 25(7):789–810, 1995.

11   Fernando Pereira and David Warren. Definite clause grammars for language analysis – a survey of the formalism and a comparison with augmented transition networks. *Artificial intelligence*, 13(3):231–278, 1980.

12   Christian Schneiker, Dietmar Seipel, Werner Wegstein, and Klaus Prätor. Declarative Parsing and Annotation of Electronic Dictionaries. In *6th International Workshop on Natural Language Processing and Cognitive Science (NLPCS)*, pages 122–132, 2009.

13   Peter Van Roy. Extended DCG notation: A tool for Applicative Programming in Prolog. Technical Report UCB/CSD 90/583, Computer Science Division, UC Berkeley, 1990.

14   Jan Wielemaker. An Overview of the SWI-Prolog Programming Environment. In *Proc. 13th International Workshop on Logic Programming Environments (WLPE)*, pages 1–16, 2003.

15   Jan Wielemaker. SWI-Prolog Reference Manual 7.6, 2017.

16   Jan Wielemaker and Michael Hendricks. Why It's Nice to be Quoted: Quasiquoting for Prolog. In *Proc. 23rd Workshop on Logic-based Methods in Programming Environments (WLPE)*, 2013.

A Conceptual Generic Framework to Debugging in the Domain-Specific Modeling Languages for Multi-Agent Systems

Baris Tekin Tezel 🆔

Department of Computer Science, Dokuz Eylul University, Izmir, Turkey
International Computer Institute, Ege University, Izmir, Turkey
baris.tezel@deu.edu.tr

Geylani Kardas

International Computer Institute, Ege University, Izmir, Turkey
geylani.kardas@ege.edu.tr

── Abstract ──────────────────────────────

Despite the existence of many agent programming environments and platforms, the developers may still encounter difficulties on implementing Multi-agent Systems (MASs) due to the complexity of agent features and agent interactions inside the MAS organizations. Working in a higher abstraction layer and modeling agent components within a model-driven engineering (MDE) process before going into depths of MAS implementation may facilitate MAS development. Perhaps the most popular way of applying MDE for MAS is based on creating Domain-specific Modeling Languages (DSMLs) with including appropriate integrated development environments (IDEs) in which both modeling and code generation for system-to-be-developed can be performed properly. Although IDEs of these MAS DSMLs provide some sort of checks on modeled systems according to the related DSML's syntax and semantics descriptions, currently they do not have a built-in support for debugging these MAS models. That deficiency causes the agent developers not to be sure on the correctness of the prepared MAS model at the design phase. To help filling this gap, we introduce a conceptual generic debugging framework supporting the design of agent components inside the modeling environments of MAS DSMLs. The debugging framework is composed of 4 different metamodels and a simulator. Use of the proposed framework starts with modeling a MAS using a design language and transforming design model instances to a run-time model. According to the framework, the run-time model is simulated on a built-in simulator for debugging. The framework also provides a control mechanism for the simulation in the form of a simulation environment model.

1 Introduction

Agents are mostly defined as the computer systems which are capable of autonomous actions inside an environment in order to achieve its design objectives [44]. They may behave as simple as just responding environmental changes or their behavior model can be too complex and the agents may need to proactively act to anticipate future goals. These agents interact with other agents and hence constitute Multi-agent Systems (MASs). Despite the existence of many agent programming environments and platforms such as CLAIM [35], GOAL [19],

JADE [1], JACK [21], MOISE+ [22], the developers may still encounter difficulties on implementing MAS due to above mentioned features of agents and agent interactions inside the MAS organizations [8].

Working in a higher abstraction layer and modeling agent components within a model-driven engineering (MDE) process before going into depths of MAS implementation may help building MAS [23]. Various agent metamodels (e.g. [3, 16, 40]) are defined by the Agent-oriented Software Engineering (AOSE) [36] community for modeling agents, their plans, beliefs, goals and agent interactions inside MAS organizations. Perhaps the most popular way of applying MDE for MASs is based on creating Domain-specific Modeling Languages (DSMLs) with including appropriate integrated development environments (IDEs) in which both modeling and code generation for system-to-be-developed can be performed properly [25]. Proposed MAS DSMLs (e.g. [15, 7, 14, 2, 12, 26] have abstract syntaxes based on the above referred agent metamodels and they usually support modeling both the static and the dynamic aspects of agent software from different MAS viewpoints including agent internal behaviour model, interaction with other agents, use of other environment entities, etc. To give some flavor of current modeling environments of these DSMLs, screenshots taken from the IDEs of two relatively new MAS DSMLs, Sam [12] and DSML4BDI [26], are given in Figure 1.

As can be seen from the screenshots, these IDEs provide a modeling area with a palette (at the right side) with including the graphical representations of MAS components. Developers may drag and drop these components and create the agent models pertaining to the specific MAS viewpoints. Upon completion of modeling, a series of model-to-model and/or model-to-text transformations are applied on the models to generate the executables (e.g. agent codes) required for the exact implementation of the MAS. Although IDEs of these MAS DSMLs provide some sort of checks on modeled systems according to the related DSML's syntax and (mostly static) semantics descriptions, currently they do not have a built-in support for debugging these MAS models [41]. That deficiency causes the agent developers not to be sure on the correctness of the prepared MAS model at the design phase. To help filling this gap, we introduce how a conceptual generic debugging framework can be derived for MAS DSMLs in this paper. The framework is conceptual since its components are defined but not fully implemented yet.

Understanding of software execution behavior has always been very hard task. Debuggers help developers to understand execution behavior of software by accessing directly executed programs [17, 46, 13]. Besides, debugging activities have a broader meaning in the domain-specific modeling (DSM) since a model developer usually needs to debug models at the model level, not at the code level [29]. In our previous study [41], we investigated the ways of creating debugging mechanisms for MAS DSMLs and introduced two possible approaches. The first approach is based on the construction of a mapping between MAS model entities and the generated codes while the second one considers the metamodel-based description of the operational semantics of executing agents. A brief evaluation of these approaches showed that the application of the first approach can be easier since it benefits from using already existing general-programming language (GPL) debuggers. However, use of MAS DSMLs do not only produce executable codes; other artifacts (e.g. agent configuration files, service descriptions) also need debugging. Moreover, generated codes mostly do not contain complete behavioral logic required for the exact implementation of agents. These can make the application of the first approach inefficient. The second approach, utilizing the metamodel-based description of agent operational semantics, seems promising since it is free from underlying GPL structures. However, it is more difficult to apply because
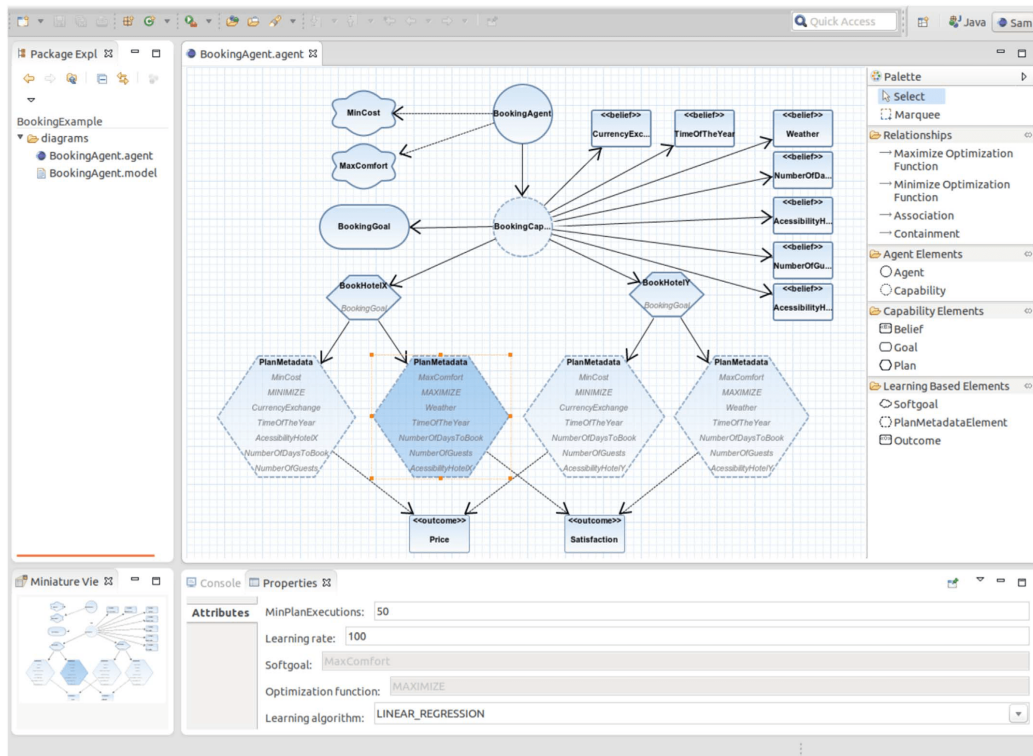
(a) A screenshot from the IDE of Sam [12].
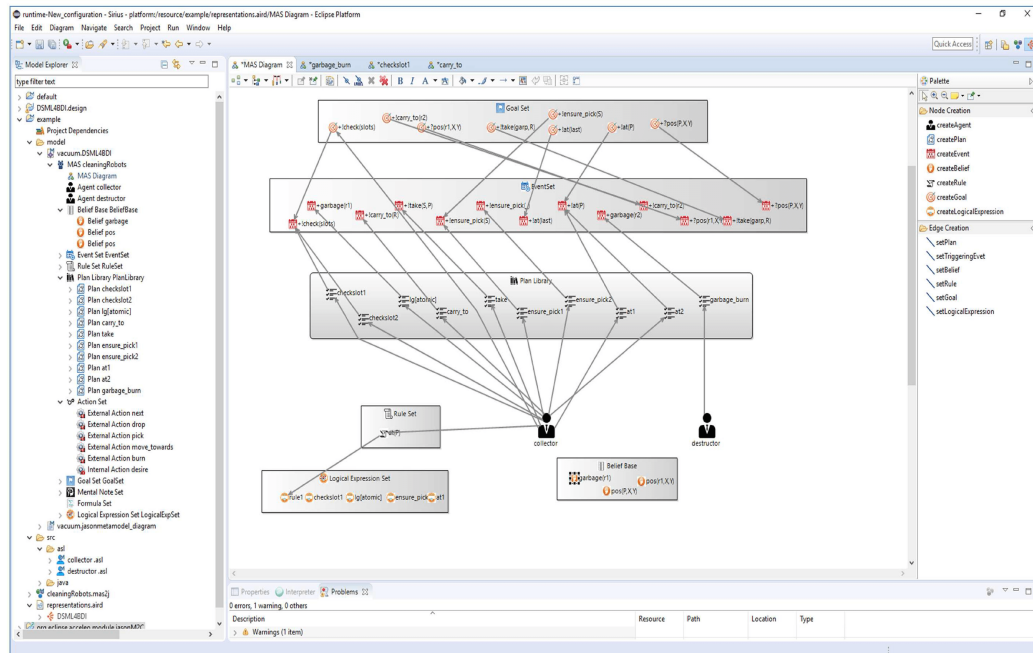
(b) A screenshot from the IDE of DSML4BDI [26].

Figure 1 Screenshots from the IDEs of two different MAS DSMLs.

it needs addition of parts describing the run-time state of MAS model into the language metamodel and writing the corresponding model to model (M2M) transformation rules. Based on the findings of this previous work, in this paper, we present how the current design structures of the existing MAS DSMLs can be enriched with additional run-time, simulation and visualization languages to construct a conceptual debugging framework. That framework is generic to provide the design of both agent internals and communications and it may pave the way for implementing MAS DSMLs with built-in debuggers. Thus, it is possible to complete the debugging phase at the modeling level before the code generation which leads to creating a MAS model conforming to the specifications at the beginning.

The rest of the paper is organized as follows: In section 2, we introduce the overview of the proposed conceptual generic debugging framework. Debugging operations which can be managed inside the framework is given in section 3. Supporting model simulator is described in section 4. A brief review of the related work within the context of debugging on DSMLs, software agents and MAS is given in section 5. Finally, section 6 concludes the paper.

2   The conceptual generic debugging framework for MAS DSMLs

In this study, our goal is to provide a generic framework for debugging during MAS modeling inside IDEs of MAS DSMLs.The framework may enable providing debugging abilities to existing MAS DSMLs as well as guiding DSML developers for the insertion of model debugging features during design and implementation of new MAS DSMLs. The debugging framework is generic enough to be reusable and easily adaptable for various aspects and viewpoints of MAS DSMLs. For instance, it would be possible to model and validate the execution of agents plans, consistency of beliefsets or construction of the agent communications according to the well-defined agent protocols, such as Contract Net [39]. Overview of the proposed framework is shown in Fig 2. The framework is the composition of 4 different metamodels and a simulator enabling the MAS operational semantics. This formalization of the framework is adopted from [43] which provides a structured approach for turning modelling and simulation environments into interactive debugging environments. Furthermore, the overall structure of the proposed framework is inspired from the sub-languages descriptions discussed in the ProMoBox [30] system and the dynamic modeling language composition defined in [18].

Inside the framework, a MAS design language metamodel, MM_{MASDL}, defines the structure of design language which is used for modelling the static structure of a MAS. Existing MAS DSMLs already have this kind of viewpoint(s) usually merged with their behaviour representations. So, if the debugging ability is desired for any existing MAS DSML, probably the metamodel of the language is needed to be refined to achieve the related static structure. For example, MAS DSMLs such as DSML4BDI [26], DSML4MAS [15] and SEA_ML [7] can be considered as the design languages since they does not have any behavioral diagram. However, if a MAS DSML has both behavior and structural viewpoints, while structural viewpoints considered as the design language, behaviour viewpoints or part of these viewpoints can be used in the run time language which is explained in the next. An instance of this metamodel is called the MAS design model (M_{MASDL}). The general structure of the system could be modeled with these instance models. They are designed and created by the users. The concepts such as agents, roles, capabilities, plans or events and the relationships between these concepts would be modelled in this model.

Next, a MAS run-time language metamodel MM_{MASRL} supports modeling in the framework to represent the run-time states of the above discussed design models. For example; meta-entities such as currentPlan, nextPlan, currentAction, nextAction, and currentBelief
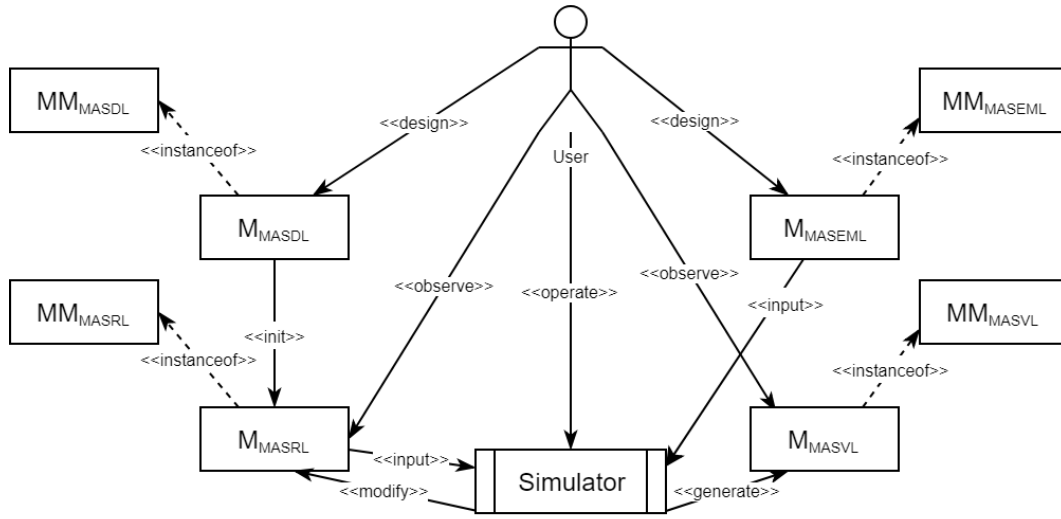
Figure 2 Overview of the conceptual generic debugging framework for MAS DSMLs.

are possibly included in this metamodel. Run-time model (M_{MASRL}) of the system is the instance of MM_{MASRL}. It represents the system run-time states and it is originated from the design model. In other words, these models express the snapshots of the MAS states at run-time.

Third, a metamodel for MAS simulation environment language (MM_{MASEML}) lets modeling of the simulation environment in which the MAS model is simulated. It can also be said that MAS simulation scenarios are modeled as being the instance models (M_{MASEML}) of MM_{MASEML}. Actually, this model represents the behavior of the simulation environment. Situations such as the results of the actions of an agent in the simulation environment, the events that will arise in the environment, communication conditions between agents, resource accesses and resource utilization cases should all be modelled with this simulation environment model. It is worth indicating that it expresses the conditions under which the MAS will be tested.

Finally, a MAS visualization language metamodel, MM_{MASVL}, allows to create customized models which graphically shows the related parts of a MAS in a way that increasing the comprehension of the MAS models by the developers. These models (M_{MASVL}) are requested by the developer. For example, if a developer is just interested in the communication between the specified agents, then the simulator generates the visualization of a complete simulation trace that is used to present the communication step-by-step. Moreover, the simulation trace is generated for not only MAS views but also inner views of an agent such as plan execution.

This conceptual framework supports debugging on a MAS, which is designed with the help of MAS DSML models created by the agent developers before the implementation phase. Thus, it is aimed to minimize the bugs on the MAS to be constructed. According to proposed framework, run-time model is initialized from the design model. Initialized run-time model represents the zero-moment of the system. At the same time, the simulated environment in which the run-time model is located, should be modeled by the simulation environment modeling language by the developer. The run-time model and the simulation environment model are given as inputs to the simulator, while the simulator builds the next state of the MAS by modifying the run-time model. The user is responsible for initiating and using the simulator with the debugging operations. In the next section of this paper, debug operations are introduced.

If a MAS DSML developer wants to re-tailor an existing MAS DSML with the proposed framework, s/he has to derive the abstract syntaxes of the above mentioned sub-languages. The metamodels of these languages could be generated from the existing DSML metamodel. For this purpose, the DSML metamodel should be divided into parts which are static / dynamic, agent internal / MAS organization, etc. and the developer determines which of them can be input into the simulation environment. By this way, initial version of the metamodels of the sub-languages become evident. This process can be completed automatically (or at least) semi-automatically in case the metamodel of the DSML is already annotated to provide additional information. After the abstract syntax of the sub-languages are generated, the developer has to create transformation rules between the sub-languages. As a final step, a simulator has to be implemented.

Once a concrete implementation of the framework is already available for a specific MAS DSMLs, it is also possible to inherit the same implementation for another MAS DSML by just re-engineering the model transformations via constructing transformations between the metamodel of the new DSML and current run-time language inside the framework. It is worth indicating that providing a horizontal transformation between the metamodels of MAS DSMLs can also enable debugging for the target DSML when the framework is already applied for the source DSML in the transformation. A discussion on how the mechanism for bridging MAS DSMLs with horizontal model transformations can be found in [24].

3   Debugging Operations

In this section, we discuss some debugging operations that may take place in the simulation environment to be built within the proposed framework.

3.1   Steps

In code debugging, stepping through code is often used by the users to understand how system states change during execution. This brings an opportunity for exposing a detailed representation of the behavior of the relevant system. If we want to use this kind of approach for debugging on models, it can be considered as changing current model state to the next state according to operational semantics provided by the simulator. Since the coding structure in the current agent programming languages / environments (such as JADE [1]) is generally similar to the coding structure of the object-oriented paradigm, the run-time states of the program is a kind of call hierarchy. In its simplest form, we can observe that the software agents can call their plans, which can be triggered by the events in the environment and, if necessary, call for more sub-plans to achieve their goals. As in object-oriented paradigm, stepping over code can usually be achieved in three possible operators: Step into, Step over, and Step out. The definition of these operators within our debugging framework are presented as follows:

Step into: As it is well known, an object is an encapsulation unit in object-oriented paradigm. It encapsulates data and methods and also hides its current state from the outside. Similar to object-oriented paradigm, an agent is also an encapsulation unit covering agent beliefs, goals and plans. These encapsulated elements could be considered as the composite elements of an agent entity in a model. Step into refers to stepping through the model element including any composite elements defined within this model element. For example, when the user tries to step into an agent's internal state, stepping contains all possible plans with all possible actions or sub-plans of the selected plans.

Step over: Step over and step into concepts state a duality. Step over refers to the stepping through the model at the composite level. It may be considered as some kind of filters while debugging. However, it does not mean that underlying elements of the model at the composite level are executed. It only hides them. For example, when the user steps over a plan of an agent, the user just observes elements which compose this agent plan. Actions or sub-plans can still trigger something at lower levels, but that is hidden.

Step out: Step out refers to stepping through a model element from inner composition level of it. For example while the user steps into a plan of an agent, the user has the option to step out from inside this plan to the composition level of the plan. At that state, the simulation continues at the composite level and the details inside the plan are hidden.

3.2 Breakpoints

Basic assertions in programming are breakpoints. When the assertion fails, execution of the program is interrupted. Generally, in breakpoints concept, this situation can be triggered by reaching specific code line. For debugging on models, interruption is defined as pausing the simulation in our framework. Here, the user will place specific breakpoints on the model, allowing the simulation to pause when certain conditions are met. Three possible stopping points are introduced:

State based: This is the case when the simulation is stopped if the model reaches a certain predefined state or a specific pattern within the state. For example, the emergence of a particular event may trigger a plan of an agent, reaching a pattern of beliefs for a specific agent or capturing a predefined communication pattern between agents.

Condition based: When some agent features are defined in the model, provide certain logical conditions which will be checked on these features. For example, if the agent's beliefs are not valid anymore due to the changes in the environment facts, update the agent belief set.

Time based: A predefined time has been reached in the simulation environment. However, this is directly related to the time definition of the simulation environment. (Real-time, scaled real-time, as fast as possible, timeless (discrete event-based), etc.)

3.3 Execution Modes

A simulator allows the transition from one execution mode to another. By this way, the control of the simulation is left to the user. The user can stop or pause and then resume the simulation of the run-time model at any point in time. We define 4 different execution modes and transitions between them. The different execution modes and transitions are illustrated in Figure. 3. These execution modes are briefly described as follows:

Ready: It is the default state of the run-time model. Also, it refers to the first state of the simulation which is also the initial state of a MAS. That state may consist e.g. the initialization of agent beliefs, goals and plans.

Running: When the user starts the simulation, the simulation environment switches to Running mode. In Running mode, the simulation can switch to different simulation modes. However, this is just possible if something such as breakpoint interrupts interrupts the run of simulation. For instance, in a state of the running mode, each agent of a MAS executes an atomic event that is simply a action in a plan of an agent.

Pause: While the simulation continues, the simulation environment can be passed to Pause mode. In Pause mode, the simulation is freezed in the current state. The user can switch back to Running mode at any time. In pause mode, a user can see a screenshot of current

state of a MAS including execution traces of the plans of each agent, communication
traces between agents, etc.

Stopped: If the simulation is terminated, it switches to Stopped mode. In this case, it can
only be passed to Ready mode. When simulator stops, current MAS model is transformed
to the initial state of itself. From this moment on, it cannot be possible to turn back to
continue the simulation.
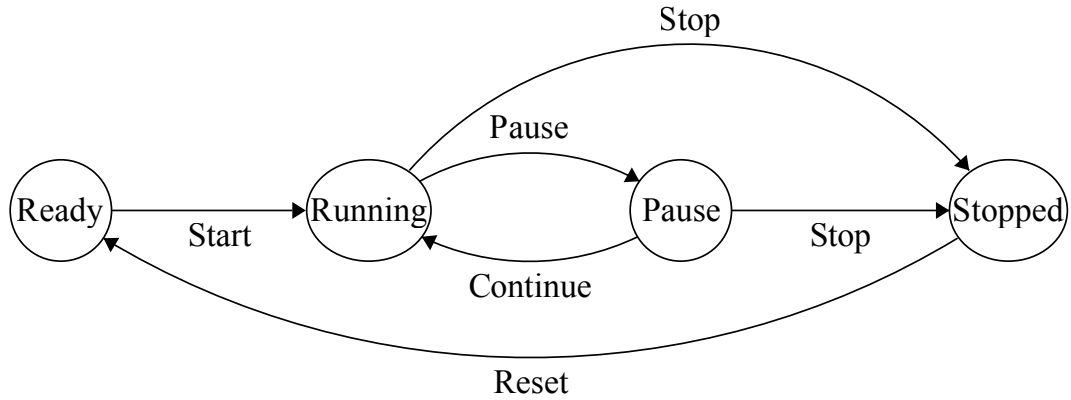
Figure 3 Execution modes transition graph of the simulator.

4   Model Simulator

In the proposed framework, operational semantics of the modelled MAS is provided by a
simulator. A number of options are available to define the simulator. For example, directly
modifying the run-time model by model transformation rules to obtain next state of the
system is one of these options. However, general structure of the simulation algorithms are
very similar at a higher level abstraction. The algorithm 1 is an abstract pseudocode of the
simulator considered in the framework. As can be seen from the algorithm 1, the simulation
is divided into the following functions:

initialize: This function is responsible for generating the initial state of the simulation,
which is represented by the MAS run-time model.

terminationCondition: This function takes the run-time model, simulation environment
model and the time as inputs and returns "true" if the desired termination condition is
provided.

updateBeliefs: This function updates the current beliefs of an agent according to the
run-time model and the simulation environment model.

updateGoals: After updating the belief of an agent, the simulator has to decide which goals
to be achieved by an agent. In order to achieve updating goal, simulator calls this function
with the run-time model, current state of an agent and the simulation environment model.

updateActivePlan: This function uses reasoning mechanism to find a plan to achieve goals
of an agent. If there is a plan already in progress, it expects the plan to end in order not
to disturb the integrity. If the active plan has ended, it determines a new active plan.

nextAction: This function executes next action of an agent according to the active plan.

increaseTime: This function increases the simulation time by adhering to the time semantics
determined according to the simulation setup.

checkForBreakpoints: This function supports to check breakpoints to pause the simulation

pauseSimulation: The simulation environment allows users to set breakpoints and when the condition holds, this function pauses the simulation.

Algorithm 1: Generic Simulation Algorithm.

Input: to be simulated Model (M), Simulation Environment Model (E)
Output: Run-time Model (RM), time (t)
(time) t, (run-time model) $RM \leftarrow$ initialize(M,E) ;
while *not terminationCondition(RM,E,t)* **do**
    foreach *agent $a_i \in RM$* **do**
        $a_i \leftarrow$ updateBeliefs(RM,a_i,E);
        $a_i \leftarrow$ updateGoals(RM,a_i,E);
        $a_i \leftarrow$ updateActivePlan(RM,a_i,E);
        $a_i \leftarrow$ nextAction(RM,a_i,E);
    end
    $t \leftarrow$ increaseTime(RM,t);
    if *checkForBreakpoints(RM)* **then**
        pauseSimulation();
    end
end

Every turn of the while loop in the Algorithm 1 runs one step of the simulation . By applying endogenous transformation to run-time model with the help of the defined functions, it creates a snapshot of the next state of the system.

5 Related Work

The most common approaches to debugging of MASs have been to adopt visualization techniques [42, 32, 31, 34].van Liedekerke and Avouris [42] introduce a technique using abstractions to reduce the amount of information being presented to the user during agent development. Nwana et al. [32] provide debugging tools based on multiple views of the computation to limit the information flow by combining results from different views. Poutakidis et al. [34] suggest applying a method for the translation from Agent UML (AUML) to Petri nets that should enable developers to construct, or convert their own protocols into an equivalent Petri net. This generated Petri net is used for debugging MASs by monitoring the exchange of messages between agents. Also, there exist some basic tools visualizing agent states in the development environments [10, 11, 19, 33]. Such tools give information about the current state of an agent while running MAS. Some of them allow users to modify the state of an agent too. In addition, Hindriks[20] presents a more advanced approach enabling users to ask questions about MAS behaviors.

All above approaches are primarily concerned with only providing a visual representation of message exchange between agents in a MAS and do not consider the embracing model of agent internals and agent interactions altogether as in existing MAS DSMLs. We believe that the study introduced in this paper contributes those efforts by composing the static and dynamic aspects of MAS modeling as well as covering the debugging needs of full-fledged MAS DSMLs which own much more complicated modeling environments comparing the visual environments considered in the above studies.

On the other hand, the studies on DSML debugging are rare and recently emerging. Mannadiar and Vangheluwe [29] propose the matching of the concepts of debugging between the programming languages and the DSMLs. This study can be accepted as the starting point for the development of the DSML debuggers. As an application of the conceptual mapping described in [29], Kosar et al. [27] present a DSML with the debugging tool called Sequencer, which is developed for the measurement systems. A similar approach appears in [9] to present a debugger development framework for domain-specific languages (DSLs).

One of the most advanced domain-specific debuggers to date is presented in [45]. Wu et al. provide the reuse of existing, tried and familiar debugging facilities in DSLs. The approach allows the DSL designer to use the internal debugging possibilities of a general programming language by the detailed mapping between model elements and the synthesized code. Lindeman et al. [28] enrich a language definition to support debugging similar to Wu et al.

Blunk et al. [4] propose a method to model debuggers for a DSML. This approach requires a metamodel-based description of the abstract syntax of the language. Debugging is defined by the process semantics at the meta-modeling level where possible run-time states are modeled as part of a DSL metamodel and the transitions are defined as a transformation from one model to another. The omniscient debugger in executable DSMLs (xDSML) is explored in [5]. In that study, the domain-specific metamodels are produced, as well as a domain tracking manager is generated to enable developers to use a general all-inclusive debugger with xDSMLs.

Although these noteworthy DSML debugging studies have guided us on evaluating design issues and helped the derivation of the framework proposed in this paper, none of them considers the needs of debugging in MAS DSMLS and in fact, the agent domain is already out of these studies' scope. Being inspired from these efforts, possible ways of constructing debugger mechanisms for MAS DSMLs are investigated in our previous work [41] which, to the best of our knowledge, is the only work so far concerned with providing debugging on MAS DSMLs.

6   Conclusion

A conceptual generic debugging framework supporting the design of both agent internals (plans, goals, etc.) and interaction between agents inside the modeling environments of MAS DSMLs has been introduced in this paper. The debugging framework is composed of 4 different metamodels and a simulator. Use of the proposed framework starts with modeling a MAS using a design language and transforming design model instances to a run-time model. According to the framework, the run-time model is simulated on a built-in simulator for debugging. The framework also provides a control mechanism for the simulation in the form of a simulation environment model. With this model, simulation scenarios can be sent to the simulator as an input. Hence, it allows a user to model agent behaviors and environment responses while the simulation is running. Finally, the framework supports the visualization of models while they are executed on the simulation environment.

In our future work, a concrete implementation of the generic framework will be completed for one of the newest MAS DSMLs, called DSML4BDI [26] in order to support the debugging of software agents according to Belief-Desire-Intention model. In fact, we already divided the metamodel of DSML4BDI into the parts to generate the metamodels of the sub-languages. So, we will provide transformation rules among the metamodels of the design language and the visual language. Finally, we will implement a simulator, which takes the instances

of the run-time language and the simulation environment language as inputs in order to modify both the run-time model step-by-step and generate visual models for each step in the simulation. Beside that, we will investigate the ways of adopting different debugging approaches such as omniscient debugging[6], model slicing[38] and algorithmic debugging[37] in the framework.

References

1 Fabio Luigi Bellifemine, Giovanni Caire, and Dominic Greenwood. *Developing multi-agent systems with JADE.* John Wiley & Sons, 3 edition, 2007.

2 Federico Bergenti, Eleonora Iotti, Stefania Monica, and Agostino Poggi. Agent-oriented model-driven development for JADE with the JADEL programming language. *Computer Languages, Systems & Structures*, 50:142–158, 2017.

3 Ghassan Beydoun, Graham Low, Brian Henderson-Sellers, Haralambos Mouratidis, Jorge J Gomez-Sanz, Juan Pavon, and Cesar Gonzalez-Perez. FAML: A Generic Metamodel for MAS Development. *IEEE Transactions on Software Engineering*, 35(6):841–863, 2009.

4 Andreas Blunk, Joachim Fischer, and Daniel A. Sadilek. Modelling a Debugger for an Imperative Voice Control Language. In *SDL 2009: Design for Motes and Mobiles. SDL 2009. Lecture Notes in Computer Science*, volume 5719, pages 149–164. Springer, 2009.

5 Erwan Bousse, Jonathan Corley, Benoit Combemale, Jeff Gray, and Benoit Baudry. Supporting efficient and advanced omniscient debugging for xDSMLs. In *Proceedings of the 2015 ACM SIGPLAN Int. Conf. Software Language Engineering (SLE 2015)*, pages 137–148, 2015.

6 Erwan Bousse, Dorian Leroy, Benoit Combemale, Manuel Wimmer, and Benoit Baudry. Omniscient debugging for executable DSLs. *Journal of Systems and Software*, 137:261–288, 2018.

7 Moharram Challenger, Sebla Demirkol, Sinem Getir, Marjan Mernik, Geylani Kardas, and Tomaz Kosar. On the use of a domain-specific modeling language in the development of multiagent systems. *Engineering Applications of Artificial Intelligence*, 28:111–141, 2014.

8 Moharram Challenger, Marjan Mernik, Geylani Kardas, and Tomaž Kosar. Declarative specifications for the development of multi-agent systems. *Computer Standards & Interfaces*, 43:91–115, 2016.

9 Andrei Chiş, Marcus Denker, Tudor Gîrba, and Oscar Nierstrasz. Practical domain-specific debuggers using the Moldable Debugger framework. *Computer Languages, Systems & Structures*, 44(Part A):89–113, 2016.

10 Rem Collier. Debugging agents in agent factory. In *International Workshop on Programming Multi-Agent Systems*, pages 229–248. Springer, 2006.

11 Mehdi Dastani, Jaap Brandsema, Amco Dubel, and John-Jules Ch Meyer. Debugging BDI-based multi-agent programs. In *International workshop on programming multi-agent systems*, pages 151–169. Springer, 2009.

12 Joao Faccin and Ingrid Nunes. A tool-supported development method for improved BDI plan selection. *Engineering Applications of Artificial Intelligence*, 62:195–213, 2017.

13 Josep Silva Galiana. The New Generation of Algorithmic Debuggers. In *1st Symposium on Languages, Applications and Technologies (SLATE 2012)*, page 3, 2012.

14 Enyo José Tavares Gonçalves, Mariela I Cortés, Gustavo Augusto Lima Campos, Yrleyjander S Lopes, Emmanuel SS Freire, Viviane Torres da Silva, Kleinner Silva Farias de Oliveira, and Marcos Antonio de Oliveira. MAS-ML 2.0: Supporting the modelling of multi-agent systems with different agent architectures. *Journal of Systems and Software*, 108:77–109, 2015.

15 Christian Hahn. A Domain Specific Modeling Language for Multiagent Systems. In *Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems - Volume 1*, pages 233–240, Estoril, Portugal, 2008. International Foundation for Autonomous Agents and Multiagent Systems. AAMAS '08. URL: `http://dl.acm.org/citation.cfm?id=1402383.1402420`.

16   Christian Hahn, Cristian Madrigal-Mora, and Klaus Fischer. A platform-independent metamodel for multiagent systems. *Autonomous Agents and Multi-Agent Systems*, 18(2):239–266, 2009.

17   Brent Hailpern and Padmanabhan Santhanam. Software debugging, testing, and verification. *IBM Systems Journal*, 41(1):4–12, 2002.

18   Ábel Hegedüs, István Ráth, and Dániel Varró. Replaying execution trace models for dynamic modeling languages. *Periodica Polytechnica Electrical Engineering and Computer Science*, 56(3):71–82, 2012.

19   Koen V Hindriks. Programming rational agents in GOAL. In *Multi-agent programming: languages and tools and applications*, pages 119–157. Springer, New York, 2009.

20   Koen V Hindriks. Debugging is explaining. In *International Conference on Principles and Practice of Multi-Agent Systems*, pages 31–45. Springer, 2012.

21   Nick Howden, Ralph Rönnquist, Andrew Hodgson, and Andrew Lucas. JACK intelligent agents-summary of an agent infrastructure. In *5th International conference on autonomous agents*, 2001.

22   Jomi F Hübner, Olivier Boissier, Rosine Kitio, and Alessandro Ricci. Instrumenting multi-agent organisations with organisational artifacts and agents. *Autonomous agents and multi-agent systems*, 20(3):369–400, 2010.

23   Geylani Kardas. Model-driven development of multiagent systems: a survey and evaluation. *The Knowledge Engineering Review*, 28(04):479–503, 2013.

24   Geylani Kardas, Emine Bircan, and Moharram Challenger. Supporting the platform extensibility for the model-driven development of agent systems by the interoperability between domain-specific modeling languages of multi-agent systems. *Comput Sci Inf Syst*, 14(3):875–912, 2017.

25   Geylani Kardas and Jorge J. Gomez-Sanz. Special issue on model-driven engineering of multi-agent systems in theory and practice. *Computer Languages, Systems & Structures*, 50:140–141, 2017.

26   Geylani Kardas, Baris Tekin Tezel, and Moharram Challenger. Domain-specific modelling language for belief–desire–intention software agents. *IET Software*, 12(4):356–364, 2018.

27   Tomaž Kosar, Marjan Mernik, Jeff Gray, and Tomaž Kos. Debugging measurement systems using a domain-specific modeling language. *Computers in Industry*, 65(4):622–635, May 2014. `doi:10.1016/j.compind.2014.01.013`.

28   Ricky T Lindeman, Lennart C L Kats, and Eelco Visser. Declaratively Defining Domain-Specific Language Debuggers. In *International Conference on Generative Programming and Component Engineering (GPCE)*, pages 127–136, 2012. `doi:10.1145/2189751.2047885`.

29   Raphael Mannadiar and Hans Vangheluwe. Debugging in Domain-Specific Modelling. In *Lecture Notes in Computer Science*, volume 6563, pages 276–285. Springer, 2011.

30   Bart Meyers, Romuald Deshayes, Levi Lucio, Eugene Syriani, Hans Vangheluwe, and Manuel Wimmer. ProMoBox: a framework for generating domain-specific property languages. In *International Conference on Software Language Engineering*, pages 1–20. Springer, 2014.

31   Divine T Ndumu, Hyacinth S Nwana, Lyndon C Lee, and Jaron C Collis. Visualising and debugging distributed multi-agent systems. In *Proceedings of the third annual conference on Autonomous Agents - AGENTS '99*, pages 326–333, New York, New York, USA, 1999. ACM Press. `doi:10.1145/301136.301220`.

32   Hyacinth S. Nwana, Divine T. Ndumu, Lyndon C. Lee, and Jaron C. Collis. Zeus: A toolkit for building distributed multiagent systems. *Applied Artificial Intelligence*, 13(1-2):129–185, January 1999. `doi:10.1080/088395199117513`.

33   Alexander Pokahr, Lars Braubach, Andrzej Walczak, and Winfried Lamersdorf. Jadex-engineering goal-oriented agents. *Developing multi-agent systems with JADE*, pages 254–258, 2007.

34   David Poutakidis, Lin Padgham, and Michael Winikoff. Debugging multi-agent systems using design artifacts: The case of interaction protocols. In *Proceedings of the first international*

joint conference on Autonomous agents and multiagent systems: part 2, pages 960–967. ACM, 2002.

35 Amal El Fallah Seghrouchni and Alexandru Suna. Claim and sympa: A programming environment for intelligent and mobile agents. In *Multi-Agent Programming*, pages 95–122. Springer, 2005.

36 Onn Shehory and Arnon Sturm. *Agent-Oriented Software Engineering: Reflections on Architectures, Methodologies, Languages, and Frameworks*. Springer-Verlag Berlin Heidelberg, 2014.

37 Josep Silva. A survey on algorithmic debugging strategies. *Advances in engineering software*, 42(11):976–991, 2011.

38 Josep Silva. A vocabulary of program slicing-based techniques. *ACM computing surveys (CSUR)*, 44(3):12, 2012.

39 Reid G. Smith. The contract net protocol: High-level communication and control in a distributed problem solver. *EEE Transactions on computers*, 12:1104–1113, 1980.

40 Baris T. Tezel, Moharram Challenger, and Geylani Kardas. A metamodel for Jason BDI agents. In *5th Symposium on Languages, Applications and Technologies (SLATE'16)*, volume 51, pages 8:1—-8:9, 2016.

41 Baris Tekin Tezel and Geylani Kardas. Towards Providing Debugging in the Domain-Specific Modeling Languages for Software Agents. In *Proceedings of the Second International Workshop on Debugging in Model-Driven Engineering (MDEbug 2018) co-located with ACM/IEEE 21st International Conference on Model Driven Engineering Languages and Systems (MODELS 2018)*, 2018.

42 Marc H Van Liedekerke and Nicholas M Avouris. Debugging multi-agent systems. *Information and Software Technology*, 37(2):103–112, 1995.

43 Simon Van Mierlo. *A multi-paradigm modelling approach for engineering model debugging environments*. PhD thesis, University of Antwerp, 2018.

44 Michael Wooldridge. *An introduction to multiagent systems*. John Wiley & Sons, 2009.

45 Hui Wu, Jeff Gray, and Marjan Mernik. Grammar-driven generation of domain-specific language debuggers. *Software: Practice and Experience*, 38(10):1073–1103, 2008.

46 Andreas Zeller. *Why Programs Fail: A Guide to Systematic Debugging*. Morgan Kaufmann, 2009.

From Lexical to Semantic Features in Paraphrase Identification

Pedro Fialho 🟢

INESC-ID, Lisboa, Portugal
Universidade de Évora, Portugal
pedro.fialho@l2f.inesc-id.pt

Luísa Coheur 🟢

INESC-ID, Lisboa, Portugal
Instituto Superior Técnico, Universidade de Lisboa, Portugal
luisa.coheur@l2f.inesc-id.pt

Paulo Quaresma 🟢

INESC-ID, Lisboa, Portugal
Universidade de Évora, Portugal
pq@uevora.pt

── Abstract ──

The task of paraphrase identification has been applied to diverse scenarios in Natural Language Processing, such as Machine Translation, summarization, or plagiarism detection. In this paper we present a comparative study on the performance of lexical, syntactic and semantic features in the task of paraphrase identification in the Microsoft Research Paraphrase Corpus. In our experiments, semantic features do not represent a gain in results, and syntactic features lead to the best results, but only if combined with lexical features.

1 Introduction

The task of paraphrase identification consists in deciding if two sentences have the same meaning. It is a popular task in Natural Language Processing, as it can be used in several scenarios. For instance, it can be used for evaluation purposes in Machine Translation: a translation result can be missing a reference, and, still, be a good translation; thus, we should be able to see if it is a paraphrase of some sentence in the reference [24]. In addition, paraphrase identification can also be used by a chatbot that has in its knowledge base a set of pre-defined question/answer pairs. Here, a question submitted by the user needs to be compared with existing questions. If the user question is a paraphrase of an existing question, the system only needs to return the appropriate answer [19]; other applications in which paraphrase identification can help include summarization [21], or plagiarism detection [18].

In many cases, just by comparing the shared lexical elements of two sentences (seen as bags of words) we are able to identify paraphrases. However, in many other cases we need to move to a semantic level to be able to say that two sentences are equivalent. For instance, *Symptoms of influenza include fever and nasal congestion.* and *Fever and nasal congestion are symptoms of influenza.* can be identified as paraphrases by taking advantage of features at a lexical level (for instance, by counting the number of common words). However, the previous sentences and the sentence *A stuffy nose and elevated temperature are signs you may have the flu.*[1] will only be identified as paraphrases if we have access to some semantic information, for instance, if we know that *fever* is similar or equal to *elevated temperature* and the same between *nasal congestion* and *stuffy nose.* Thus, a system with the goal of identifying paraphrases should be able to reason at a semantic level. Unfortunately, some semantic features, such as explicit meaning representations, only exist for some languages. The same happens with syntactic features, although at a less dramatic scale, as syntactic analyzers exist for many languages.

In this paper we present a comparative study on the performance of lexical, syntactic and semantic features for paraphrase identification. To the best of our knowledge, the whole set of features that we use in this work was never employed altogether for paraphrase identification, particularly the ensemble of structural modelling for syntax and explicit whole sentence meaning representations for semantics. Results show that syntactic features lead to the best results, but only if combined with lexical features; semantic features in comparison with lexical features, bring a small improvement to recall, f-measure and accuracy when applied in addition to the lexical features.

This paper is organized as follows: in Section 2 we present Related Work, in Section 3 we describe the features from the different linguistic levels, and, in Section 4 we present the experimental setup. Finally, in sections 5 and 6 we present the obtained results and main conclusions, respectively; in the latter section we also point to future work.

2   Related work

As previously mentioned, this work is focused on paraphrase identification. Two sentences are paraphrases of each other when they express equivalent meanings. The difficulty of detecting if two sentences have equivalent meaning varies with the linguistic mechanisms employed in paraphrasing, since a target sentence may employ various lexical and/or syntactic transformations on its source.

Popular features employed in paraphrase identification were primarily designed for machine translation evaluation, such as BLEU [26]. However, many other features have already been applied to paraphrase identification, and there are even toolkits that allow to extract features from different linguistic levels. For instance, HARRY [28] provides lexical features from string similarity metrics applied to various word granularities, and SEMILAR [29] provides sentence to sentence similarity metrics based on techniques such as BLEU. It also provides word to word similarity metrics based on semantic information, as it employs Wordnet [7] and co-occurrence models such as Latent Semantic Analysis [16]. In this work we will take advantage of both these toolkits (along with INESC-ID@ASSIN [8]).

Still in the semantic features domain, explicit meaning representations of sentences can also be compared for paraphrase identification purposes. For instance, in [34] features based on the overlap among semantic representations are used. Examples of meaning representations

---

[1] `https://examples.yourdictionary.com/examples-of-paraphrasing.html`

are *Abstract Meaning Representation* (from now on AMR) [1] and *Discourse Representation Structures* [14]. In this work we will use AMR representations of sentences to calculate semantic features, as suggested in [12].

Considering syntactic features, some works (e.g., [33, 23]) take advantage of these structures on paraphrase identification. In these scenarios, the features extracted from structural comparison of parse trees, from constituent or dependency analysis, identify which sub trees are the same (structure wise), and may employ lexical semantics on leaf nodes (which carry the words of the sentence) to weight the importance of a common sub tree.

Typically, approaches for paraphrase identification employ a supervised learning setting, where a model is derived from a training corpus, composed by pairs of sentences labeled with 1 or 0 (for instance) considering that they are or they are not paraphrases, respectively. The Microsoft Research Paraphrase Corpus [6], from now on the MSRP corpus, is a popular choice to train and benchmark such models, since there is a constantly updated ranking of the various systems using it[2]. Features from machine translation evaluation achieve competitive results in MSRP, as shown in [18]. Although other publicly available corpora exist, as the paraphrases from Twitter messages [15], or, more recently, the open domain questions from Quora[3], in this paper we will target the MSRP corpus.

3 Features from different linguistic levels

We gathered features at the different linguistic levels. In the following we describe these sets.

3.1 Lexical Features

We call *lexical features* to the ones based on different distance metrics calculated between the lexical elements of a sentence, and assuming that these distances can be computed both at the character or word level. We also assume that words can be transformed in their lexical variants, by applying, for instance, stemming or encoding text into the way it sounds. An example of a lexical feature is the *longest common subsequence* metric applied to lowercased versions of the sentences in analysis.

Table 1 illustrates some of the lexical features used in this work, where each feature corresponds to the application of the metric on the leftmost column to two sequences, built according to the lexical variants identified in the remaining columns (a detailed explanation of each metric can be found in [8]). Such variants comprise lowercased (L) and stemmed (S) versions of the original (O) text. The cluster (C) and Double Metaphone (DM) variants produce a sequence composed by non verbal codes, which:

- for cluster are binary strings that identify the cluster of each word, according to the Brown clustering algorithm [3] on the Yelp dataset of online reviews[4],

- for DM are the codes of the Double Metaphone algorithm for each word.

The trigrams (T) variant produces a sequence with a different length from the number of words in the original sentence, since it is composed by strings of 3 characters, one for each character in the original text.

■ **Table 1** Combination of features with representations, where O, L, S, C, DM and T correspond to Original, Lowercased, Stemmed, Cluster, Double Metaphone and Trigrams, respectively.

Feature	O	L	S	C	DM	T
LCS	X	X	X	X	X	
Edit Distance	X	X	X	X	X	
Cosine Similarity	X	X	X	X	X	X
Abs Length	X	X	X	X	X	
Max Length	X	X	X	X	X	
Min Length	X	X	X	X	X	
Jaccard	X	X	X	X	X	X
Soft TF-IDF	X	X	X			
NE Overlap	X	X	X	X	X	X
NEG Overlap	X	X	X	X	X	X
Modal Overlap	X	X	X	X	X	X
METEOR	X	X	X	X	X	
ROUGE N	X	X	X	X	X	
ROUGE L	X	X	X	X	X	
ROUGE S	X	X	X	X	X	
TER	X	X	X	X	X	
NCD	X	X	X	X	X	
Numeric	X	X	X			

3.2 Syntactic Features

In what concerns *syntactic features* we consider that these features are also based in distances, but between syntactic constituents of the sentence. Thus, similarity scores are computed for pairs of trees, based on the number of common substructures [22]. Here, a tree kernel is applied to a pair of parse trees, to automatically produce the similarity scores. For instance, an adjective attached to a noun corresponds to a sub-tree in the full tree of constituents for a source sentence, and if the tree of the target sentence contains a sub-tree with exactly the same leafs (adjective and noun) and root (the syntactic relation), then a tree kernel would consider 3 fragments in common, meaning that both sentences apply the same adjective to the same noun. Further details on such calculation are found in [22].

3.3 Semantic Features

We follow a broad definition of *semantic features* as all the features that take advantage of some sort of semantic information, either at the lexical level (for instance, by comparing synonyms of two words) or at the sentence level (for instance, by taking advantage of semantic spaces or explicit meaning representations). Considering the latter, we draw on the previously mentioned AMR [1]. An example AMR for the sentence *My drawing was not a picture of a hat.*, from the AMR corpus for the novel "The Little Prince", can be seen in Figure 1, as produced by trained annotators [1].

In Figure 1 is shown an AMR rooted at concept *picture-01*, with *01* indicating an entry in OntoNotes [11] where this concept is defined as the act of displaying something in a picture, such that its *ARG1* represents what is displayed, as detailed in the corresponding

```
(p / picture-01 :polarity -
     :ARG0 (p2 / picture
            :ARG1-of (d / draw-01
                  :ARG0 (i / i)))
     :ARG1 (h / hat))
```

Figure 1 AMR example.

PropBank [25] frame [5] in which OntoNotes is based (since the latter is not available for free). Hence, this AMR includes expression *a pictured hat*, negated by setting attribute *polarity* of the root concept to a minus sign.

4 Experimental setup

In the following we present the resources involved in our experiments, and the method for their preparation and usage.

4.1 Corpora

As previously mentioned, we will use the Microsoft Research Paraphrase Corpus [6]. Each example in MSRP is composed of 2 sentences and a positive or negative value (0 or 1) representing whether the sentences are a paraphrase or not. We take as train/test set the usual suggested partitions.

4.2 Gathering Lexical Features

Considering the lexical features, we collect them from the two aforementioned toolkits: INESC-ID@ASSIN, a framework used in the ASSIN competition, and HARRY, a toolkit providing string similarity metrics.

In the INESC-ID@ASSIN framework, language independent metrics are applied to different representations of the original text, such as Double Metaphone codes or character trigrams. The 91 features identified in Table 1 were gathered from the INESC-ID@ASSIN framework.

We also use lexical features extracted from HARRY, which also provides a way of extracting lexical features based on 3 different representations of a text: bytes, bits or words. It contributes with 21 different metrics to apply to each representation, although not all metrics are compatible with all representations. For instance, the Normalized compression distance is only applicable to bits. From HARRY, we obtain 62 features, which include string distances such as the Hamming distance and similarity coefficients such as Jaccard. The complete set of features is described in [28].

4.3 Gathering Syntactic Features

Regarding syntactic features, constituency parse trees are obtained with the Shift-Reduce version of the Stanford parser[6]. Then, tree kernels are applied to such trees. An efficient approach for structural kernels, and particularly tree kernels, was proposed by [30] in *uSVM-TK*, an SVM modelling platform based on the SVM-LIGHT engine [13]. This is the

---

[5] `http://verbs.colorado.edu/propbank/framesets-english-aliases/picture.html`
[6] `http://nlp.stanford.edu/software/srparser.shtml`

chosen learning platform for all our experiments (using tree kernels or not). All the tree kernels available in *uSVM-TK* were employed, namely "Subtree", "Subset tree", "Subset tree considering leaf labels" and "Partial tree kernel" [22].

4.4   Gathering Semantic Features

Taking into consideration semantic features, we used the ones from the already mentioned SEMILAR. From this framework, we gather 9 different features on lexical semantics, such that most correspond to a score on sentence similarity calculated from word to word similarities based on Wordnet, Latent Semantic Analysis or Latent Dirichlet Allocation [2]. The latter two word similarities are based on models provided with SEMILAR, pre-trained on Wikipedia and the TASA corpus as described in [32].

In what concerns explicit meaning representations, we obtain the AMR for the sentences with the JAMR parser [10]. Then, and in order to extract semantic features for the AMR, we use SMATCH [4], a metric that computes the distance between two AMR, with its default configuration (hill-climbing with smart initialization and 4 random restarts), established as best setting in the original SMATCH research.

4.5   Evaluation Metrics

Performance is measured with Precision, Recall, F-measure and Accuracy, except for the comparison with other systems from previously mentioned MSRP rank, where only F-measure and Accuracy are reported.

4.6   Machine Learning kits

We use both *uSVM-TK* [30] and LIBSVM [5] (from its *scikit-learn* [27] interface) in our experiments. The former allow us to test syntactic features in a plug and play way. The latter was used just for sanity checking, considering the non-syntactic features, as it does not allows a "plug and play" evaluation of syntactic features.

5   Experiments and results

5.1   The impact of the different features

The best results of applying our feature sets to MSRP are shown in Table 2. By SEMANTICS we understand a feature set containing the SEMILAR and SMATCH features, as opposed to using only one of these semantic feature sets.

As expected, lexical features achieve the best results when the majority of words are common or very similar. Also, as expected, lexical features are almost useless when a paraphrase has low lexical overlap, such as when most words in a target sentence are synonyms of the words in the source sentence. In fact, some lexical features are 0 for all training examples of MSRP, as identified with the Facets tool[7]. Listing 1 shows an example corresponding to paraphrases from the MSRP test partition that were only correctly identified using semantic features, due to low lexical overlap.

---

[7] https://pair-code.github.io/facets/

Table 2 Evaluation results on MSRP (best of all configurations attempted).

Features	Prec	Rec	F	Acc
lexical	78.79%	85.18%	81.86%	74.90%
lexical + SEMILAR	78.22%	86.40%	82.10%	74.96%
lexical + SMATCH	77.98%	85.53%	81.58%	74.32%
lexical + SEMANTICS	77.44%	85.88%	81.44%	73.97%
syntax	69.87%	**95.46**%	80.69%	69.62%
lexical + syntax	**79.90**%	86.66%	**83.14**%	**76.63**%
lexical + syntax + SEMILAR	79.44%	86.57%	82.85%	76.17%
lexical + syntax + SMATCH	79.31%	86.92%	82.94%	76.23%
lexical + syntax + SEMANTICS	79.61%	86.83%	83.06%	76.46%

Listing 1 Example that was not successful classified in *lexical + syntax*, but it was successful classified in *lexical + syntax + SEMANTICS*.

Consumers would still have to get a descrambling security card from their cable operator to
plug into the set .

To watch pay television, consumers would insert into the set a security card provided by their
cable service .

When syntax is not involved (the first 4 results in Table 2), semantics do not improve the performance of lexical features isolated. Overall, syntactic features in combination with lexical features lead to the best results.

5.2 How do we compare with other systems

In order to compare our results with state-of-the-art systems, Table 3 shows the performance of other systems on the MSRP corpus.

Of particular interest is the result from system [31], which employs neural networks, and performs similarly to our best ensemble of features. Although no feature engineering is needed, we are able to explain our results.

System [9] is the most similar to ours, in that it also employs lexical, syntactic and semantic features in the *uSVM-TK* platform. Although with fewer features, it achieves better results, as it involves more experiments, additional kernels and an exhaustive configuration of SVM parameters.

5.3 The influence of the Machine Learning toolkit

Finally, experiments were also performed in LIBSVM [5], which implements the SVM decision process in a different manner from SVM-LIGHT. Using LIBSVM for the *lexical + SEMANTICS* experiment results in F measure of 82.62% and accuracy of 76%. Hence, results improved (previous results were of 81.44% and 73.97%, respectively), which suggest an influence of the SVM implementation.

■ **Table 3** Other systems employing MSRP on similar feature types.

	F	**Acc**
lexical similarity [20]	81.3%	70.3%
distributional semantics [17]	82.8%	75.7%
neural networks [31]	83.6%	76.8%
MT metrics [18]	84.1%	77.4%
tree and graph kernels [9]	85.2%	79.1%
our best: lexical + syntax	83.1%	76.6%

6   Conclusion and Future Work

We have presented a study on the contribution of lexical, syntactic and semantic features in paraphrase identification on the MSRP corpus.

Semantic features contribute to a performance enhancement over lexical features isolated (if Precision is not considered), but slightly decreases performance when combined with lexical and syntactic features, although by less than 1%. Best results were achieved by syntactic features in combination with lexical ones. Future work includes balancing the amount of features in vector sets, further exploration of SVM parameters, enrich the set of semantic features, study the behaviour of these features in other corpora, and apply the same approach to the tasks of Semantic Textual Similarity and Recognizing Textual Entailment.

References

1   Laura Banarescu, Claire Bonial, Shu Cai, Madalina Georgescu, Kira Griffitt, Ulf Hermjakob, Kevin Knight, Philipp Koehn, Martha Palmer, and Nathan Schneider. Abstract Meaning Representation for Sembanking. In *Proceedings of the 7th Linguistic Annotation Workshop and Interoperability with Discourse*, pages 178–186, Sofia, Bulgaria, August 2013. Association for Computational Linguistics. URL: http://www.aclweb.org/anthology/W13-2322.

2   David M. Blei, Andrew Y. Ng, and Michael I. Jordan. Latent Dirichlet Allocation. *J. Mach. Learn. Res.*, 3:993–1022, March 2003. URL: http://dl.acm.org/citation.cfm?id=944919.944937.

3   Peter F. Brown, Peter V. deSouza, Robert L. Mercer, T. J. Watson, Vincent J. Della Pietra, and Jenifer C. Lai. Class-Based n-gram Models of Natural Language. *Computational Linguistics*, 18(4), 1992. URL: http://aclweb.org/anthology/J92-4003.

4   Shu Cai and Kevin Knight. Smatch: an Evaluation Metric for Semantic Feature Structures. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics, ACL 2013, 4-9 August 2013, Sofia, Bulgaria, Volume 2: Short Papers*, pages 748–752. The Association for Computer Linguistics, 2013. URL: http://aclweb.org/anthology/P/P13-2131.pdf.

5   Chih-Chung Chang and Chih-Jen Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011. Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm.

6 Bill Dolan and Chris Brockett. Automatically Constructing a Corpus of Sentential Paraphrases. In *Third International Workshop on Paraphrasing (IWP2005)*. Asia Federation of Natural Language Processing, January 2005. URL: `https://www.microsoft.com/en-us/research/publication/automatically-constructing-a-corpus-of-sentential-paraphrases/`.

7 Christiane Fellbaum. *WordNet: An Electronic Lexical Database*. Bradford Books, 1998.

8 Pedro Fialho, Ricardo Marques, Bruno Martins, Luísa Coheur, and Paulo Quaresma. INESC-ID@ASSIN: Medição de Similaridade Semântica e Reconhecimento de Inferência Textual. *Linguamática*, 8(2):33–42, December 2016. URL: `https://www.linguamatica.com/index.php/linguamatica/article/view/v8n2-4`.

9 Simone Filice, Giovanni Da San Martino, and Alessandro Moschitti. Structural Representations for Learning Relations between Pairs of Texts. In The Association for Computer Linguistics, editor, *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing, ACL 2015, July 26-31, 2015, Beijing, China, Volume 1: Long Papers*, pages 1003–1013. The Association for Computer Linguistics, 2015. URL: `http://aclweb.org/anthology/P/P15/P15-1097.pdf`.

10 Jeffrey Flanigan, Sam Thomson, Jaime Carbonell, Chris Dyer, and Noah A. Smith. A Discriminative Graph-Based Parser for the Abstract Meaning Representation. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics*, volume 1, pages 1426–1436. Association for Computational Linguistics, 2014.

11 Eduard Hovy, Mitchell Marcus, Martha Palmer, Lance Ramshaw, and Ralph Weischedel. OntoNotes: The 90% solution. In *Proceedings of the Human Language Technology Conference of the NAACL, Companion Volume: Short Papers*, pages 57–60, New York City, USA, June 2006. Association for Computational Linguistics. URL: `https://www.aclweb.org/anthology/N06-2015`.

12 Fuad Issa, Marco Damonte, Shay B. Cohen, Xiaohui Yan, and Yi Chang. Abstract Meaning Representation for Paraphrase Detection. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 442–452. Association for Computational Linguistics, 2018. `doi:10.18653/v1/N18-1041`.

13 Thorsten Joachims. Making large-Scale SVM Learning Practical. In B. Schölkopf, C. Burges, and A. Smola, editors, *Advances in Kernel Methods - Support Vector Learning*, chapter 11, pages 169–184. MIT Press, Cambridge, MA, 1999.

14 Hans Kamp and Uwe Reyle. *From Discourse to Logic - Introduction to Modeltheoretic Semantics of Natural Language, Formal Logic and Discourse Representation Theory*, volume 42 of *Studies in linguistics and philosophy*. Springer, 1993. `doi:10.1007/978-94-017-1616-1`.

15 Wuwei Lan, Siyu Qiu, Hua He, and Wei Xu. A Continuously Growing Dataset of Sentential Paraphrases. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1224–1234. Association for Computational Linguistics, 2017. URL: `http://aclweb.org/anthology/D17-1126`.

16 Thomas K. Landauer and Susan T. Dumais. A Solution to Plato's Problem: The Latent Semantic Analysis Theory of Acquisition, Induction, and Representation of Knowledge. *Psychological Review*, 104(2):211–240, 1997.

17 Mihai C. Lintean and Vasile Rus. Measuring Semantic Similarity in Short Texts through Greedy Pairing and Word Semantics. In G. Michael Youngblood and Philip M. McCarthy, editors, *Proceedings of the Twenty-Fifth International Florida Artificial Intelligence Research Society Conference, Marco Island, Florida. May 23-25, 2012*. AAAI Press, 2012. URL: `http://www.aaai.org/ocs/index.php/FLAIRS/FLAIRS12/paper/view/4421`.

18 Nitin Madnani, Joel Tetreault, and Martin Chodorow. Re-examining Machine Translation Metrics for Paraphrase Identification. In *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Tech-*

nologies, NAACL HLT '12, pages 182–190, Stroudsburg, PA, USA, 2012. Association for Computational Linguistics. URL: `http://dl.acm.org/citation.cfm?id=2382029.2382055`.

19 Jerome L. McClendon, Naja A. Mack, and Larry F. Hodges. The Use of Paraphrase Identification in the Retrieval of Appropriate Responses for Script Based Conversational Agents. In William Eberle and Chutima Boonthum-Denecke, editors, *Proceedings of the Twenty-Seventh International Florida Artificial Intelligence Research Society Conference, FLAIRS 2014, Pensacola Beach, Florida, May 21-23, 2014.* AAAI Press, 2014. URL: `http://www.aaai.org/ocs/index.php/FLAIRS/FLAIRS14/paper/view/7793`.

20 Rada Mihalcea, Courtney Corley, and Carlo Strapparava. Corpus-based and Knowledge-based Measures of Text Semantic Similarity. In *Proceedings of the 21st National Conference on Artificial Intelligence - Volume 1*, AAAI'06, pages 775–780. AAAI Press, 2006. URL: `http://dl.acm.org/citation.cfm?id=1597538.1597662`.

21 Amita Misra, Brian Ecker, and Marilyn Walker. Measuring the Similarity of Sentential Arguments in Dialogue. In *Proceedings of the 17th Annual Meeting of the Special Interest Group on Discourse and Dialogue*, pages 276–287. Association for Computational Linguistics, 2016. `doi:10.18653/v1/W16-3636`.

22 Alessandro Moschitti. Efficient Convolution Kernels for Dependency and Constituent Syntactic Trees. In *Proceedings of the 17th European Conference on Machine Learning*, ECML'06, pages 318–329, Berlin, Heidelberg, 2006. Springer-Verlag.

23 Alessandro Moschitti. Making Tree Kernels Practical for Natural Language Learning. In *11th Conference of the European Chapter of the Association for Computational Linguistics*, pages 113–120, 2006. URL: `http://www.aclweb.org/anthology/E06-1015`.

24 Sebastian Pado, Michel Galley, Dan Jurafsky, and Christopher D. Manning. Robust Machine Translation Evaluation with Entailment Features. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, pages 297–305. Association for Computational Linguistics, 2009. URL: `http://www.aclweb.org/anthology/P09-1034`.

25 Martha Palmer, Daniel Gildea, and Paul Kingsbury. The Proposition Bank: An Annotated Corpus of Semantic Roles. *Computational Linguistics*, 31(1):71–106, 2005. `doi:10.1162/0891201053630264`.

26 Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a Method for Automatic Evaluation of Machine Translation. In *Proceedings of 40th Annual Meeting of the Association for Computational Linguistics*, pages 311–318, Philadelphia, Pennsylvania, USA, July 2002. Association for Computational Linguistics. `doi:10.3115/1073083.1073135`.

27 F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

28 Konrad Rieck and Christian Wressnegger. Harry: A Tool for Measuring String Similarity. *J. Mach. Learn. Res.*, 17(1):258–262, January 2016. URL: `http://dl.acm.org/citation.cfm?id=2946645.2946654`.

29 Vasile Rus, Mihai Lintean, Rajendra Banjade, Nobal Niraula, and Dan Stefanescu. SEMILAR: The Semantic Similarity Toolkit. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 163–168, Sofia, Bulgaria, August 2013. Association for Computational Linguistics. URL: `http://www.aclweb.org/anthology/P13-4028`.

30 Aliaksei Severyn and Alessandro Moschitti. Large-scale Support Vector Learning with Structural Kernels. In *Proceedings of the 2010 European Conference on Machine Learning and Knowledge Discovery in Databases: Part III*, ECML PKDD'10, pages 229–244, Berlin, Heidelberg, 2010. Springer-Verlag. URL: `http://dl.acm.org/citation.cfm?id=1888339.1888355`.

31   Richard Socher, Eric H. Huang, Jeffrey Pennin, Christopher D Manning, and Andrew Y.
        Ng. Dynamic Pooling and Unfolding Recursive Autoencoders for Paraphrase Detection. In
        J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. Pereira, and K. Q. Weinberger, editors,
        Advances in Neural Information Processing Systems 24, pages 801–809. Curran Associates,
        Inc., 2011. URL: `http://papers.nips.cc/paper/4204-dynamic-pooling-and-unfolding-`
        `recursive-autoencoders-for-paraphrase-detection.pdf`.

32   Dan Stefanescu, Rajendra Banjade, and Vasile Rus. Latent Semantic Analysis Models on
        Wikipedia and TASA. In *Proceedings of the Ninth International Conference on Language
        Resources and Evaluation (LREC-2014)*. European Language Resources Association (ELRA),
        2014. URL: `http://www.lrec-conf.org/proceedings/lrec2014/pdf/403_Paper.pdf`.

33   Kai Sheng Tai, Richard Socher, and Christopher D. Manning. Improved Semantic Repres-
        entations From Tree-Structured Long Short-Term Memory Networks. In The Association for
        Computer Linguistics, editor, *Proceedings of the 53rd Annual Meeting of the Association for
        Computational Linguistics and the 7th International Joint Conference on Natural Language
        Processing of the Asian Federation of Natural Language Processing, ACL 2015, July 26-31,
        2015, Beijing, China, Volume 1: Long Papers*, pages 1556–1566. The Association for Computer
        Linguistics, 2015. URL: `http://aclweb.org/anthology/P/P15/P15-1150.pdf`.

34   Rob van der Goot and Gertjan van Noord. ROB: using semantic meaning to recognize
        paraphrases. In Daniel M. Cer, David Jurgens, Preslav Nakov, and Torsten Zesch, editors,
        *Proceedings of the 9th International Workshop on Semantic Evaluation, SemEval@NAACL-HLT
        2015, Denver, Colorado, USA, June 4-5, 2015*, pages 40–44. The Association for Computer
        Linguistics, 2015. URL: `http://aclweb.org/anthology/S/S15/S15-2007.pdf`.

Learning JavaScript in a Local Playground

Ricardo Queirós 🔗

Department of Informatics – School of Media Arts and Design, Polytechnic of Porto, Portugal
CRACS INESC TEC, Porto, Portugal
http://www.ricardoqueiros.com
ricardoqueiros@esmad.ipp.pt

— Abstract —

JavaScript is currently one of the most popular languages worldwide. Its meteoric rise is mainly due to the fact that the language is no longer bound to the limits of the browser and can now be used on several platforms. This growth has led to its increasing use by companies and, consequently, to become part of the curriculum in schools. Meanwhile, in the teaching-learning process of computer programming, teachers continue to use automatic code evaluation systems to relieve their time-consuming and error prone evaluation work. However, these systems reveal a number of issues: they are very generic (one size fits all), they have scarce features to foster exercises authoring, they do not adhere to interoperability standards (e.g. LMS communication), they rely solely on remote evaluators being exposed to single point of failure problems and reducing application performance and user experience, which is a feature well appreciated by the mobile users. In this context, LearnJS is presented as a Web playground for practicing the JavaScript language. The system uses a local evaluator (the user's own browser) making response times small and thus benefiting the user experience. LearnJS also uses a sophisticated authoring system that allows the teacher to quickly create new exercises and aggregate them into gamified activities. Finally, LearnJS includes universal LMS connectors based on international specifications. In order to validate its use, an evaluation was made by a group of students of Porto Polytechnic aiming to validate the usability of its graphical user interface.

1 Introduction

Nowadays, we are seeing a great evolution of the JavaScript language. The language commonly used on the client-side of the Web, has freed itself in recent years from browser boundaries and is nowadays also used on the server side (Node.js) and also on other platforms such as desktop (e.g. Electron) and mobile devices (NativeScript, React Native). With this evolution, companies began to adopt more and more the language for the multi-platform development. This growth in the companies made the schools begin to integrate the teaching of language in the courses curricula.

In the teaching-learning of computer programming, one of the most used teaching methodologies is the massive resolution of programming exercises. The methodology enhances the practice, but on the other hand creates an excessive workload on the teacher who has to evaluate all the exercises of all the students. To mitigate this task, automatic evaluation systems have been used in recent years, whose mission is to assess and give feedback on student performance. Despite their apparent success, these systems present several problems, namely issues related to performance, interoperability, feedback or even personalized gamification.

This article introduces LearnJS as a playground for JavaScript language training. The system provides a set of thematic activities composed of various types of educational resources (exercises, videos). After the resolution of the exercise, the system presents feedback on the resolution allowing the user to progress in the activity in a sustained way.

LearnJS distinguishes itself from other systems in various facets, namely in the exercises authoring, performance and interoperability.

In the following sections we will focus the attention in the computer learning environments, namely, the code playgrounds. Then, we present LearnJS as a tool to practice JavaScript. In this section we present its architecture, main components, data and integration models and the playground GUI. In the next section, we conduct an usability survey for the LearnJS user interface. Finally, we share the main contribution of this work and point out some future work.

2 Computer Programming Learning

The process of learning computer programming can be very complex and demotivating [1, 3, 6]. In order to help with those issues, several approaches appeared trying to simplify the process and make it accessible to everyone, even those with absolutely no coding experience or knowledge [7, 8]. These approaches come in several formats ranging from non-interactive approaches (e.g. YouTube channels, blogs, books) to integrated and interactive solutions (e.g. intelligent tutors, on-line coding providers, code playgrounds, pastebins, MOOCs). All these approaches can be grouped in four categories as shown in Figure 1.
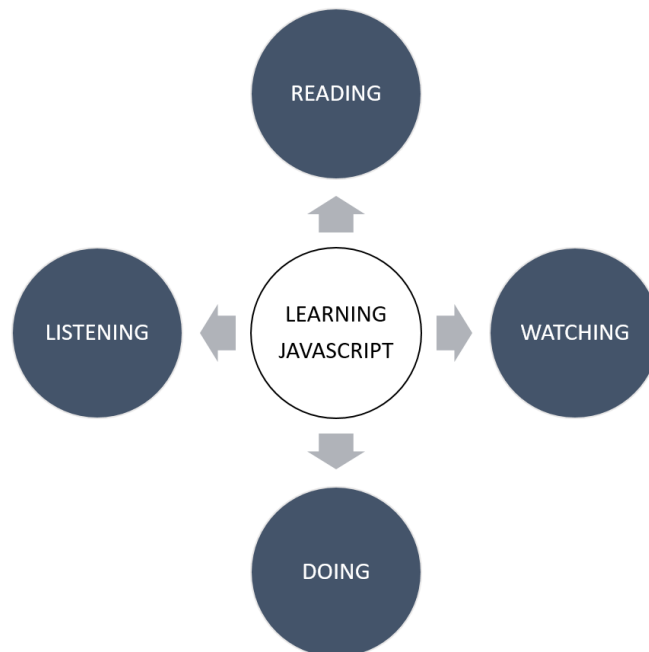
Figure 1 Learning approaches.

One of the best ways to learn is by reading books. Books are always a solid knowledge base because they synthesize best practices and guarantee a quality certified by a publisher. The disadvantage of books is that in the area of Information Technology, always in constant update, they become quickly obsolete. Other reading options are modern development blogs (e.g. Dev.to, Medium) and newsletters.

We can also observe or listen others to code. The former, can be useful if we want to dig deeper into certain aspects of coding. The later, is also a great way to learn to code without needing to be actively tied to a screen. In this realm, the podcasts community is evolving at a good pace[1].

The last, and arguably most important, part of learning JavaScript is the actual doing part: writing code, getting it to work, and repeating. This is the intrinsic concept of practice. However the practice will only be useful if there is immediate and expressive feedback on what we practice. The absence of feedback or the inclusion of false positives is detrimental to practice and will only make the learner more demotivated. Several online learning environments appeared to address theses needs. The goal of this section is not to detail all of them in depth, but rather to focus on those that have the following characteristics:

- Web-based
- practice oriented
- with automatic feedback
- based on gamified challenges
- JavaScript language support

Based on these features, **online code playgrounds** are increasingly dominating and a selected set of them will be detailed and compared in the next section.

2.1 Front-End Code Playgrounds

A variety of front-end code playgrounds have appeared over the years. These type of tools are extremely flexible and help students to practice a programming language. These tools have several common features. Among the main features of these tools are: collection of exercises organized by modules or categories, sophisticated JavaScript code editors with preview windows, developer consoles and code validation tools, code sharing facilities, gamification elements, such as level unblocking, badges and rankings.

There are a lot of playgrounds with these main features. A selected set of Code Playgrounds will be described:

- Sololearn (SL)
- CoderByte (CB)
- CodeWars (CW)
- CodinGame (CN)
- HackerRank (HR)

In the following section, we compare these code playgrounds based on set of predefined criteria such as: edition, evaluation, gamification and interoperability.

2.1.1 Edition

This type of Web-based tools use typically a sophisticated JavaScript editor (e.g. Ace) with some of the inherited functionality of traditional IDEs editors such as syntax highlighting, intelligent code completion, themes, snippets and syntax checker.

At the same time, there are several types of exercises that may be available in the editor: exercises where the student will have to code a solution for a particular problem, puzzles, fill in the blanks, questionnaires, etc.

Table 1 compares these features on the select editor.

---

[1] URL: https://player.fm/podcasts/Js

Table 1 Edition features.

Features	SL	CB	CW	CN	HR
Syntax Highlighting	X	X	X	X	X
Code Completion	X	X	X	X	X
Themes	X	X	-	X	-
Snippets	-	-	-	X	X
Syntax checkers	X	-	X	X	X
Quizzes	X	X	-	-	-
Puzzles	-	-	-	-	X
Fill the blanks	X	-	X	-	X

2.1.2   Evaluation

A programming exercise can be evaluated dynamically and/or statically.

The dynamic evaluation consists in the definition of a set of test cases composed by input data and corresponding output. In the evaluation process the input data is injected into the student's solution and the result is obtained. This same result is compared to the output data provided by the teacher. If they are identical it means that the test was successful and the procedure for the remaining tests is repeated. Typically several predefined tests are provided in the learning environment. The student can perform only one or all of the tests at any time. Some systems provide the functionality of adding new test cases on the part of the student in order to encourage the creation of tests. Even in the tests it is possible to have hidden test cases in order to prevent the student from solving the tests-oriented challenges and not the problem to solve in a generic way. Each test, in addition to the input and expected output, may have associated feedback that will be shown to the user if that test fails.

In the previous evaluation type, the feedback that is given to the student is whether the test has passed or not. In short, the student will have a test to say that his solution is correct or, alternatively, that it is not correct because it failed some tests, undercutting them. To enrich the feedback it is necessary to apply a static analysis that instead of executing the student code and inject input data, does an introspect to the code, without executing it, and checks a predefined set of metrics. In this context, the presence of a certain keyword or block of code, the style of code (convention of variables names), the presence of comments or even the application of certain algorithm can be verified. For this type of analysis linters or other static analysis tools are usually used.

Table 2 compares these features on the select editors.

Table 2 Evaluation features.

Features	SL	CB	CW	CG	HR
Dynamic Analysis	X	X	X	X	X
Hidden Tests	X	X	X	X	X
Feedback Tests	X	-	-	X	X
New tests	X	X	-	-	-
Static Analysis	-	-	X	-	X

2.1.3 Gamification

Regarding gamification and social features, most platforms adhere to the same components, such as forums (F), learning dashboards (D), user profiles (UP), comments (COM), recommendation (REC), levels and badges. For instance, CodePlayer offers a different approach to learning code by playing code like a video, helping people to learn front-end technologies quickly and interactively. The platform also includes a commenting tool and links to related walkthroughs. CodeAcademy includes a user progress dashboard informing of the current state of the learner regarding its progress in the courses. This platform enhances the participation in the courses by also including achievements (ACH) that are rewarded with badges and users are also able to share completed projects with the rest of the site community and potentially showcase their skills to employers. Except for Code.org, all the platforms have a strong presence in the mobile world, with app versions for Android and iOS.

Table 3 compares these features on the select editors.

Table 3 Gamification features.

Features	SL	CB	CW	CG	HR
Forums	X	X	X	X	X
Dashboards	X	X	-	X	X
User Profiles	-	X	X	X	X
Comments	X	-	X	X	X
Recommendation	X	-	-	X	X
Levels	-	X	-	X	X
Badges	X	X	X	X	X
Achievements	X	X	X	X	X
Likes and followers	-	X	-	X	X
Awards	X	X	X	X	-
Hints and code skeletons	-	X	-	X	X

2.1.4 Interoperability

The category of interoperability is perhaps one of the most important, but possibly one of the most overlooked by learning environments. Most systems live on their own server without any integration with existing systems and have proprietary forms of exercises and activities. Communication with other systems, such as evaluators or plagiarism systems, is done in a ad-hoc way using internal APIs.

Also the integration of these playgrounds with Learning Management Systems (LMS) would be an expected and desirable functionality. The student authenticates in the LMS and solves a programming activity in the playground. At the end of the activity, a report on student activity is automatically sent and its grade book updated in the LMS.

At the same time, the authorship of programming exercises remains complex. Not only by the absence of systems that facilitate his construction, but also by the absence of standard formats that allow his formal definition, dissemination and discovery by other systems. The lack of repositories with these characteristics makes reusing and adapting exercises difficult.

Table 4 compares these features on the select editors.

Table 4 Interoperability features.

Features	SL	CB	CW	CG	HR
Prog. exercise formats	X	-	X	-	X
Activities formats	X	-	-	-	X
LMS integration	X	-	-	X	-
Repository integration	X	X	-	X	X
Gamification standards	X	-	-	X	X

3   LearnJS

LearnJS is a Web playground which enables anyone to practice the JavaScript language [4]. The architecture of LearnJS is depicted in Figure 2.
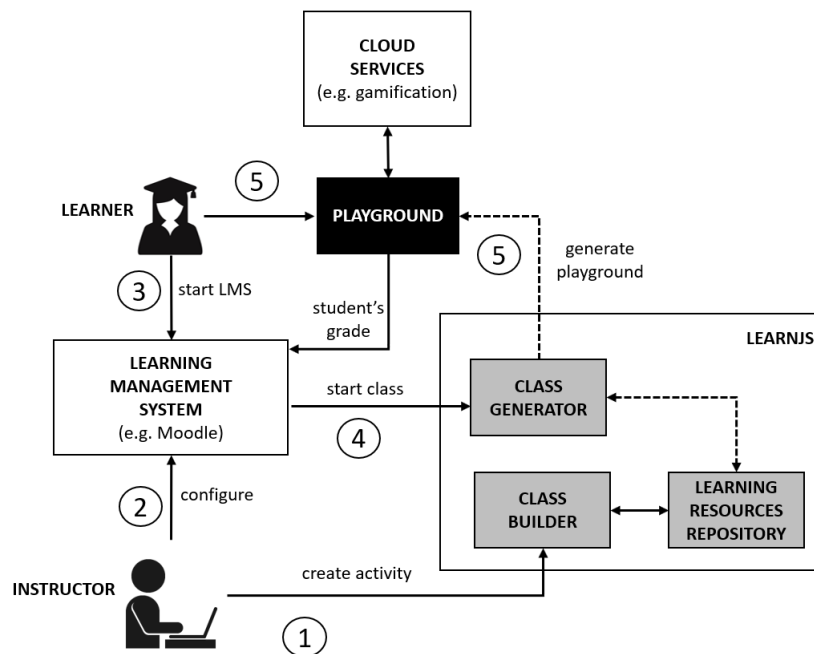
Figure 2 LearnJS architecture.

At its core, LearnJS is composed by two components used by the two system user profiles:

- Instructors: use the LearnJS Class Builder to create and select resources from the Learning Resources Repository in order to compose a learning activity. Next, they configure the activity in a Learning Management System.
- Learners: they run the LMS in a browser and launch the activity in the LMS. The launch process is controlled by the class generator that receives a LearnJS manifest and generates the corresponding activity in a playground which is returned to the user. Beyond the internal gamification features, the playground can benefit from other Gamification Services to foster student's competitiveness and engagement.

3.1   Data Model

The LearnJS data model is composed by two entities: the activity and the resource.

A resource is an educational object, either expository or evaluative. An expository resource is typically a video or a PDF file showing how to master a specific topic. An evaluative resource is a JavaScript challenge to be solved by coding. Instructors can use the LearnJS class builder to contribute with new resources. The supported resources in LearnJS follow Sweller and Cooper [6] paradigm based on a learner centred approach to define a constructivist learning model. This model foster the learning by viewing and learning by doing approaches where educational resources, either expository or evaluative, play a pivotal role. A resource is defined as a JSON file which should comply with the LearnJS official resource schema formalized by a public JSON Schema [2]. It contains basic properties for identification and metadata purposes.

Instructors can also perform other operations in the class builder, such as the creation of activities. An activity combines a set of resources of several types (evaluative, expositive) with gamification attributes. An activity JSON file is composed by several properties. We highlight two:

- levels: can be considered as sub-activities composed by a set of resources identified in the resources sub-property. Students should see and solved the respective resources of the level. The completion of the level and the respective unlock of the next level is granted after the student solved a specific percentage of evaluative resources defined in the perc property of the level.
- gamify: a set of attributes that can be assigned to resources. After a success completion of an evaluative resource, students can be awarded in multiple forms. Hence, the award property can have one of the following values:
  - HintExtra: gives an extra point to the learner. The learner can spend the hint points on any exercise by unhiding the hint associated;
  - ShowNextSkeleton|ShowAllSkeleton: gives the learner the ability to unhide the code skeleton associated to the next (or all) gamified resources;
  - UnlockLevel|UnlockAllLevels: gives the learner the ability to unlock the next (or all) level.

The example on Listing 1 shows an activity JSON instance for learning JavaScript arrays:

■ **Listing 1** Learning activity JSON instance.

```
{
 "id": "http://learnJS/activities/129387",
 "title": "Learn the basics of Arrays",
 "metadata": {
   "author":"Ricardo Queiros",
   "date":"19-04-1975",
   "level":"basic",
   "tags":["arrays"]
 },
 "levels": [
  {"id":"1", "name":"Basic operations", "perc":"75",
         "resources": ["...resources/125412", "..."]},
  {"id":"2", "name":"Sort", "perc":"50", "resources":["..."]}
 ],
 "gamify": [
```

---

[2] Public GitHub link: `https://github.com/.../learnJS`

```
    {"resource": ".../resources/125412", "award":"HintExtra"},
    {"resource": ".../resources/225232", "award":"ShowNextSkeleton"}
 ]
}
```

3.2   Integration Model

The purpose of LearnJS is also to integrate an e-learning ecosystem based on an LMS (e.g. Moodle, Sakai, BlackBoard). For this, it benefits from the interoperability mechanisms to provide authentication directly from the LMS and to submit exercises grades back to the LMS, using the Learning Tools Interoperability (LTI) specification.

3.3   Playground

The LearnJS Playground is a Web-based component which will be used by learners to browse learning activities and interact with the compound resources. Here students can see videos of specific topics and solve exercises related with those topics with automatic feedback on their resolutions (Figure 3).
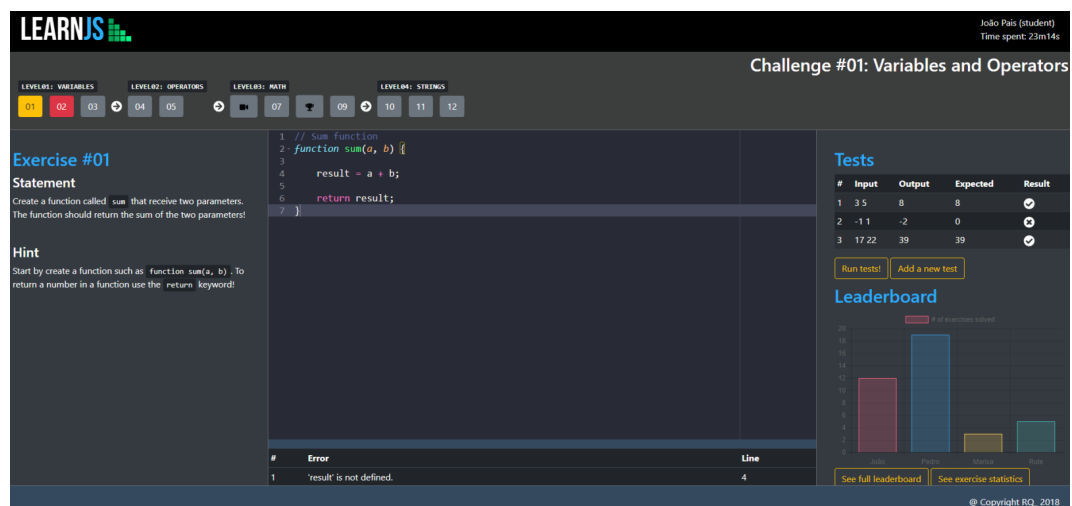
Figure 3 LearnJS playground GUI.

The playground is composed by three main components:
1. Editor: allows students to code their solutions in an interactive environment;
2. Evaluator: assess the student's solution based on static and dynamic analyzers;
3. Gamification Engine: gamifies the learning activity with the management of levels and several awards.

For the Editor component, the playground uses Ace (maintained as the primary editor for Cloud9 IDE) which can be easily embedded in any web page and JavaScript application. The editor is properly configured for the JavaScript language and supports the Emmet toolkit for the inclusion of dynamic JavaScript snippets. Ace editor can display errors on the editor itself but does not handle language dependencies. A parser needs to be used to detect errors and determine their positions on the source file. There are several tools that can improve code quality. One of such cases is code linters. Linters (e.g JSLint, JSHint) can detect potential

bugs, as well as code that is difficult to maintain. These static code analysis tools come into play and help developers spot several issues such as a syntax error, an unused variable, a bug due to an implicit type conversion, or even (if properly configured) coding style issues. LearnJS uses JSHint to accomplish this behavior. While static code analysis tools can spot many different kinds of mistakes, they can not detect if your program is correct, fast or has memory leaks. For that particularly reason, LearnJS combines JSHint with functional tests (based on test cases). For this kind of tests, and since the code is written in JS and the context is the browser, we use a simple approach by iterating all the case tests and applying the eval function for tests injection.

Both analyzers (linter and Test Case runner) are subcomponents of the LearnJS evaluator component that runs exclusively on the client side. This approach avoids successive round-trips to the server which affects negatively the user experience. Lastly, the Gamification Engine component is responsible for loading/parsing the LearnJS manifest and fetching resources from the learning resources store. If levels are defined, the engine sequences and organizes the resources properly. Upon completion of evaluative resources from students, the engine deals with all the logic associated with the respective awards by unhiding/unlocking features of next challenges. Finally, the component send the results back to the server. At this moment, we have a simple running prototype. The source code is available at a GitHub repository. Figure 3 shows the front-end GUI of the playground.

4 Evaluation

This section evaluates the usability of the graphical user interface of LearnJS based on the Nielsen's model [2].

4.1 Nielsen heuristics

According to Nielsen the practical acceptability of a system includes factors such as usefulness, cost, reliability and interoperability with existing systems. The usefulness factor relates the utility and usability offered by the system. Utility is the capacity of the system to achieve a desired goal. As the system perform more tasks, more utility he has. Usability is defined by Nielsen as a qualitative attribute that estimates how easy is to use an user interface. He mentions five characteristics involved in the concept of usability:

- ease of learning - the system should be easy to learn so that the user can start doing some work with the system;
- efficiency - the system should be efficient to use, so after the user learns the system, a high level of productivity is possible;
- memorability - the system should be easy to remember so that the casual user is able to return to the system after a period without using it, without requiring to learn it all over again;
- errors - the system should prevent the user from committing errors as should deal with them gracefully and minimizing the chance of occurring catastrophic errors;
- satisfaction - the system should be pleasant to use so that users become subjectively satisfied when using.

4.2 Usability evaluation

LearnJS was evaluated according to the Nielsen's model using a heuristic evaluation methodology. A heuristic evaluation is an inspection method for computer software that helps to identify usability problems in the user interface (UI) design. Jakob Nielsen's heuristics

are arguably the most used usability heuristics for user interface design. Based on these heuristics a questionnaire with 41 questions was designed in Google Forms. The aim of the questionnaire is to identify usability problems in the UI design of LearnJS.

The experiment took place at the Media Arts and Design School (ESMAD) - a school of the Polytechnic Institute of Porto - during the month of March of 2019. The participants were students from the first-year of the course Object-Oriented Programming. This course is offered to the Web Technologies degree and aims to introduce students to programming concepts using the JavaScript language.

Students used LearnJS during one week. In the end, they were invited to fill a survey using Google Forms. The questionnaire includes questions on LearnJS usability. It had an average of 45 responses (the equivalent to 91% of the total of students).

Figure 4 shows the results obtained grouped by the Nielsen's heuristics. The data collected are shown in a chart graphs where the heuristics are sorted in ascending order of user satisfaction.
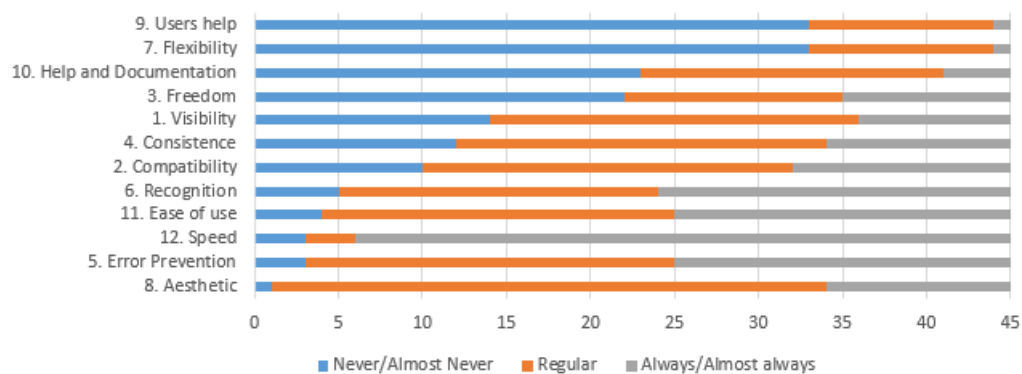
Figure 4 LearnJS usability evaluation.

The results highlight deficiencies in three areas: users help, flexibility, help and documentation and freedom. In regard to the flexibility of the system, respondents considered that the system do not allow the personalization of the interface, more precisely, the activation/deactivation of certain functions and the configuration of the screens. The possibility of use of accelerator keys to speed up the interaction with the editor and evaluator is also a handicap of LearnJS at this moment.

The Help and documentation is another heuristic with negative values. The respondents state that is difficult to find help and documentation in LearnJS. This a fact since we did not yet integrate any kind of tutorial mode in the system.

The freedom heuristic is the fourth worst facet. Most of the complaints focused on the inability to cancel or roll back mistakes made to a previous and safe state.

The respondents also reveal that the error messages are sometimes unclear and inadequate in LearnJS. The respondents also state that the documentation is scarce and is hard to find it.

In an overall comment, one can conclude that the majority of students classified LearnJS as a good tool according to the parameters evaluated.

5   Conclusions

In this paper we present LearnJS as a flexible playground for JavaScript learning. The paper stresses the design of the platform divided in two main components: the management tool and the playground. In the former, instructors can contribute with new exercises and

bundle related exercises in learning activities. All these entities were formalized using JSON schemata. The later, allows students through a sophisticated and interactive UI, to see and solve educational resources (mostly, videos and exercises). In order to engage students, the platform can be configured to gamify resources through the subgrouping of activities in levels, the assignment of awards and the exhibition of a global leaderboard. The main contributions of this work is the design of a platform with interoperability concerns in mind and the respective schemata for the simple concepts of educational resources and activities.

LearnJS is already being used in a Polytechnic School with promising results. In this paper, we present an usability survey that shows several issues which are already being tackled.

As future work, in mid 2019 we will start to work on several components, namely:

- **Class Builder** - to foster activities creation based on educational resources. We will also create a GUI for assisting instructors in the creation of exercises.
- **Importer** - to import exercises from other repositories. The importer will use an existing converter service for exercises with different formats called BabeLO [5].
- **Gamified sequencer** - to dynamically sequence exercises based on student's pace and progress. The sequencer should be enhanced by the gamified elements included in the activity.

References

1 Kirsti M Ala-Mutka. A Survey of Automated Assessment Approaches for Programming Assignments. *Computer Science Education*, 15(2):83–102, 2005. `doi:10.1080/08993400500150747`.

2 J. Nielsen. *Usability engineering*. Academic Press, 1993. URL: `https://books.google.pt/books?id=fnvJ9PnbzJEC`.

3 Jackie O'Kelly and J. Paul Gibson. RoboCode & Problem-based Learning: A Non-prescriptive Approach to Teaching Programming. *SIGCSE Bull.*, 38(3):217–221, June 2006. `doi:10.1145/1140123.1140182`.

4 Ricardo Queirós. LearnJS - A JavaScript Learning Playground (Short Paper). In *7th Symposium on Languages, Applications and Technologies, SLATE 2018, June 21-22, 2018, Guimaraes, Portugal*, pages 2:1–2:9, 2018. `doi:10.4230/OASIcs.SLATE.2018.2`.

5 Ricardo Queirós and José Paulo Leal. BabeLO - An Extensible Converter of Programming Exercises Formats. *TLT*, 6(1):38–45, 2013. `doi:10.1109/TLT.2012.21`.

6 Anthony Robins, Janet Rountree, and Nathan Rountree. Learning and Teaching Programming: A Review and Discussion. *Computer Science Education*, 13(2):137–172, 2003. `doi:10.1076/csed.13.2.137.14200`.

7 Elena Verdú, Luisa M. Regueras, María J. Verdú, José P. Leal, Juan P. de Castro, and Ricardo Queirós. A Distributed System for Learning Programming On-line. *Comput. Educ.*, 58(1):1–10, January 2012. `doi:10.1016/j.compedu.2011.08.015`.

8 J. Xavier and A. Coelho. Computer-based assessment system for e-learning applied to programming education. In *ICERI2011 Proceedings*, 4th International Conference of Education, Research and Innovation, pages 3738–3747. IATED, 2011.

Scaling up a Programmers' Profile Tool

Martinho Aragão[1]
Algoritmi R.C., University of Minho, Portugal
Dep. of Informatics, University of Minho, Portugal
martinhoaragao@gmail.com

Maria João Varanda Pereira 🄳
Algoritmi R.C., University of Minho, Portugal
CeDRI, DIC, Polytechnic Institute of Bragança, Portugal
mjoao@ipb.pt

Pedro Rangel Henriques 🄳
Algoritmi R.C., University of Minho, Portugal
Dep. of Informatics, University of Minho, Portugal
pedrorangelhenriques@gmail.com

―――― **Abstract** ――――

The style of programming, the proficiency on the programming language, the conciseness of the solution, the use of comments and so on, allow comparison of programmers through static analysis of their code. The Programmer Profiler Tool, which has been commonly named PP Tool, is an open source profiling tool for Java language where the programmer's ability can be classified in one out of five possible profiles and the distinction among them falls upon the levels of both skill and readability. Taking a set of correct solutions the comparison between solutions for the same problems is fundamental to evaluate proficiency on the analysed criteria. As such, there was a need to tune the tool in order to handle, simultaneously, with a bigger amount of programs and with a wider scope of solutions. By scaling up PP Tool it will be possible to apply it in a far wider scope of situations as it will be able to cope with programmers from different geographies, with or without formal education, between 1 and 20 years of experience amongst other factors. For that, a set of features were implemented and tested and are described in this paper.

1 Introduction

The PP Tool [6] is based on program analysis and can be applied in educational and professional contexts to compare the proficiency of a set of solutions. The main idea is to profile different programmers by using their solutions to the same problem in terms of bad-practices, ability to master a programming language and code readability (indentation, use of comments, descriptive identifiers). In this work only correct programs producing the desired output were used and the efficiency of the solution is not analysed. A programmer's ability can be classified as one of a set possible profiles and the distinction among them falls upon the levels of both skill and readability that are evaluated based on code metrics. By aiming at proficiency on these criteria one can achieve a more experienced profile.

---

[1] Corresponding author

The basic idea is to statically analyse Java source code and extract a selection of metrics. Some metrics can be directly extracted from source-code and provide a lot of information to understand the programmer proficiency like number of files, classes, methods and statements; number of lines code and comments, and their ratios; usage of control flow statements (if, while, for, etc); variable declarations and datatypes used; usage of advanced Java operators (bitshift, bitwise, etc); usage of repetitive patterns; usage of indentation and identifiers of good quality.

Moreover, it is possible to detect automatically bad-practices using the PMD tool [2]. PMD is a free source code analyser that finds common programming flaws like unused variables or code, empty catch-blocks, unnecessary object creation, poor identifier names, non-optimised code, inappropriate code size and so on.

Based on the metrics described above and the number of violations detected by PMD Tool, values are given to the parameters skill and readability. Skill is defined as the language knowledge and the ability to apply that knowledge in an efficient manner and to measure that the most important metrics are: number of statements; use of control flow statements (if, while, for, etc) and advanced Java operators; number and datatypes used. Readability is defined as the aesthetics and general concerns related with code legibility, so other metrics are taken into account: number of methods, classes and files; total number and ratio of code, comments and empty lines.

The paper is organised as follows. In Section 2 the work done by others on profiling is reviewed and compared to ours. In Section 3 a brief introduction to the main components and techniques of the original PP Tool is presented. Also in this section the original profile classes are characterised, and a refinement of that initial classification is discussed; at last, the metrics used to measure programmers' level of skill and readability, necessary to determine the profile class, are listed. After describing the problems encountered when PP Tool was applied to a big collection of programs gathered from a new source, in Section 4 we enumerate the various and important decisions taken to scale up the tool and cope properly with this kind of program sets. Then Section 5 will contain a detailed discussion on the results attained with the new version of PP Tool to enhance the gains. Section 6 concludes the paper with a summary of the work reported and a mention to the generation of detailed feedback on programmers improvement as a future research direction.

2   Related Work

Before deciding on pursuing improvements to a tool which uses a source code analysis, other alternatives of profiling were explored.

Perhaps the most used way is actually through their experience. Often one of the first steps for companies when recruiting is in the form of a *curriculum vitae*. However, this has been known to be flawed, hence requiring other methods.

One technique which has been growing in popularity employs the use of *gamification*. Particularly one can use the example of code challenge websites where programmers are ranked based on the number and difficulty of the challenges that they have solved. Scoring systems feed leaderboards and these approaches are also evidenced on [2]. However, this feeds on very particular knowledge as it completely disregards efficiency, how long it took to solve the exercise and code legibility as the only information it provides is how many challenges have been solved. It also only capable of profiling users after several exercises, while difficult exercises can take hours to be solved.

---

[2] http://pmd.github.io/

In recent years, the surge of software communities has accumulated countless data of their users. *GitHub* tracks number of commits and their information as well as pull requests and even project popularity. *StackOverflow* also tracks number of answers divided by topics and with a voting system on both the answers and the questions. In [4] the CPDScorer is introduced which aggregates the information of the platforms mentioned previously to claim very high precision. However, it once again requires a lot of information and is dependant on popularity.

Pietrikova [7] also explores techniques aiming the evaluation of Java programmers' abilities through the static analysis of their source code.They classify knowledge profiles in two types: subject and object profile. The subject profile represents the capacity that a programmer has to solve some programming task, and it's related with his general knowledge on a given language. The object profile is the model to follow and refers to the actual knowledge necessary to handle those tasks. This work is also based on metrics whose values are compared with an optimal solution. In PP Tool [5] there is no need to define an optimal solution because it is based on the relative position between a set of solutions.

There are other tools more concern with learning programming. The tool presented In [8], provides two types of analysis: software engineering metrics analysis to look for poor programming practices and logic errors in student programs and structural similarity analysis for comparing students' solutions to a model solution. Flowers et al. present a tool, Gauntlet [1], that allows beginner students understanding their Java syntax errors. It is based on a set of the most common errors for these kind of students and it uses a very friendly and helpful way of displaying those errors. Also concerned with error handling, Expresso tool [3] is a reference on Java syntax, semantic and logic error identification.

3 PP Tool at a glance

PP Tool, whose architecture implementation and tests were described in detail in [6], uses language processing techniques for static analysis and automatically extracts metrics from programs aiming to profile their writers. As was said, this process will be complemented with the use of PMD Tool, to get information on the use of good Java programming practices.

The PP Tool has two key moments for analysis, one for scoring and finally one for profiling. First, on the PP Analysis, metrics are extracted from the source code and stored on specially created class. On the second one, the PMD Analyser is used to identify common programming flaws which are also called violations. During scoring, both of the previously obtained information is transformed to impact in either skill or readability. Finally, all the solutions are provided profiles based on the comparison between their scores.

3.1 Code analysis

For each set of metrics a class with the purpose of extracting those metrics was created.

These metrics can be customised on an auxiliary file such as whether they have a positive or negative effect to skill or readability, or even the weight of the impact.

The PMD Analyser has a set of rules which can also be customised. Currently the quickstart set is used which provides a general list of rules which are valid for most situations. However if the PP Tool is to be applied on a controlled environment then it is recommended to set its own list of rules.

Each rule has a priority associated with the penalty to be inflicted. When running the analyser, rule violations are registered with information regarding the line where they occurred. Violations are then summed up based on number of occurrences and the priority to inflict a penalty.
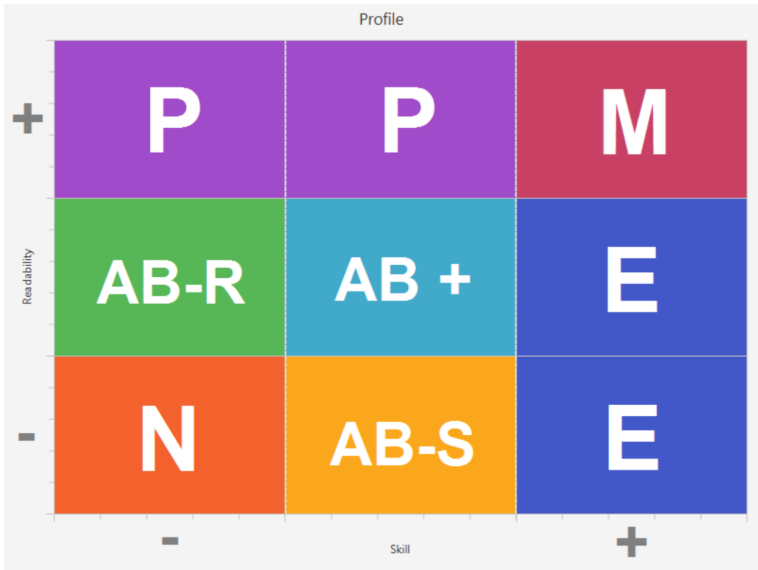
Figure 1 Profiling Distribution.

3.2   Profiling

There are 4 main profiles. The novice profile (N) identifies a programmer that is not yet familiar with all the language constructs and usually does not show language readability or good programming practices concerns. The advanced beginner (AB) programmer shows variety in the use of language constructs and data-structures, starts showing some readability concerns but still writes programs in a safely manner. The proficient programmer is familiar with a great variety of language constructs, usually follows good programming practices, has readability and code-quality concerns. The expert programmer masters a great variety of language constructs and is focuses on producing efficient code usually without readability concerns.

As time progressed, the profiles shifted a bit from the original idea. The Experts should be the ones with maximum focus on Skill, the Proficients on Readability, the Advanced Beginners were divided in three subsets and a new profile called Master was created to be associated to a high level of skill and readability.

So the profiles used in this work are the following: Novice (N): Low Skill and Low Readability; Advanced Beginner (AB): Low-to-Average (LtA) Skill and Readability; Proficient (P): LtA Skill and High Readability; Expert (E): High Skill and LtA Readability; Master (M): High Skill and High Readability.

Profiling is the last step of the tool. A grid is created with the lowest and highest values of skill and readability in mind, and all results are distributed in the grid. The grid is divided in 9 blocks of equal size as can be seen at Figure 1.

4   Scaling Up

When testing the scalability of the tool by using a big amount of programs, it lead to a great variety of results that are semantically different from the ones got from the analysis of a small amount of programs. One of the problems was the lack of distinction between solutions. Although each metric has different impact it was common to find very different

solutions that had practically the same readability and skill results. It was concluded that several metrics should be better calculated taking into account, for instance, the priority and the number of occurrences.

Two important decisions were made:

- Refine some criteria, rules and values (to cope with a bigger variety of solutions):
  - use only the percentage of blank lines and comment lines instead of also their absolute values;
  - introduction of the notion of criterion weight;
  - increase variety of violations;
  - the profile is always based on solution comparison but "isolated" solutions (very very good or very very bad) must have lower impact on the results;
  - assign weight and number of occurrences to each violation in order to tune the influence of it in skill and readability;
  - change violations impact to be proportional to readability and skill score to remove negative values due to "isolated" solutions;
  - adjust the number and the impact of each metric in order to balance both skill and readability results.
- Improve PMD performance (to cope with a bigger amount of solutions):
  - introduce a new caching option that speed up the tool;
  - turn easier the system maintenance associating the impact attribute to each group of violation rules and not to each violation individually;
  - the violations belonging to the same type are grouped and it is much more easier to associate each group to the factors skill and readability;

All of these changes lead to a more robust system that could handle the new multitude of scenarios. The scoring system changed considerably, as metrics became the only source of positive score, and violations the only source of negative ones. PMD violations now can provoke up to 50% penalty in a given score (if the solution is the one with the most severe penalties) and metrics no longer reduce score.

5 Testing the tool

In order to ensure the Programmer Profiler Tool was ready to be used in a more generic environment, we needed to test it with a far more diverse input of exercises. As such, instead of requesting more exercises from a classroom we looked into platforms which provided hundreds of challenges and solutions. In that search, online programming exercise platforms came up as an ideal solution. These type of platforms have several years worth of exercise solutions from all experience levels and with users across the globe. Other services are often either tailored for specific use cases such *Stack Overflow* with just code bits or there is great difficulty in comparing solutions for profiling which is the context of whole Open Source projects like found in *Github*.

By request CodeChef, a not-for-profit educational initiative, supplied the solutions.

In order to test the results of the changes, an exercise of medium difficulty has been chosen. Specifically we will be looking at the following solutions: solution A, solution B, solution C.

The Figure 2 represents the distribution at that stage of all 300 solutions. It's clearly visible that almost all solutions are profiled as "Experts". With the average skill being higher than the average readability, which seems consistent with the programming challenges
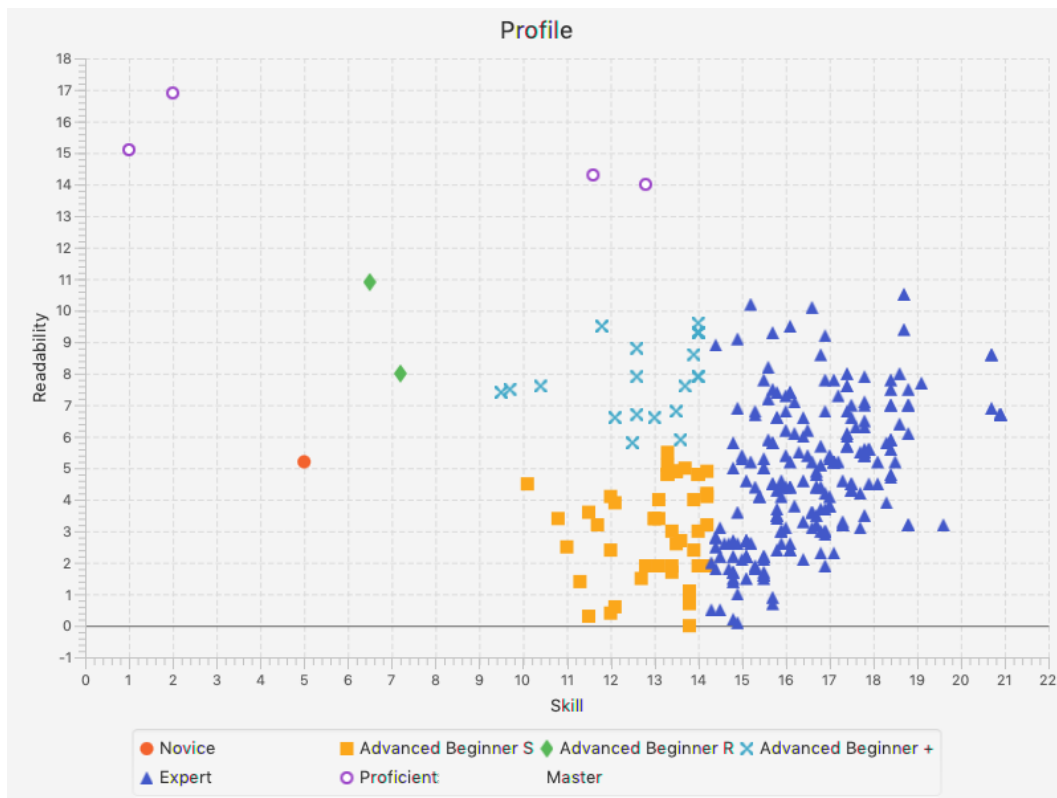
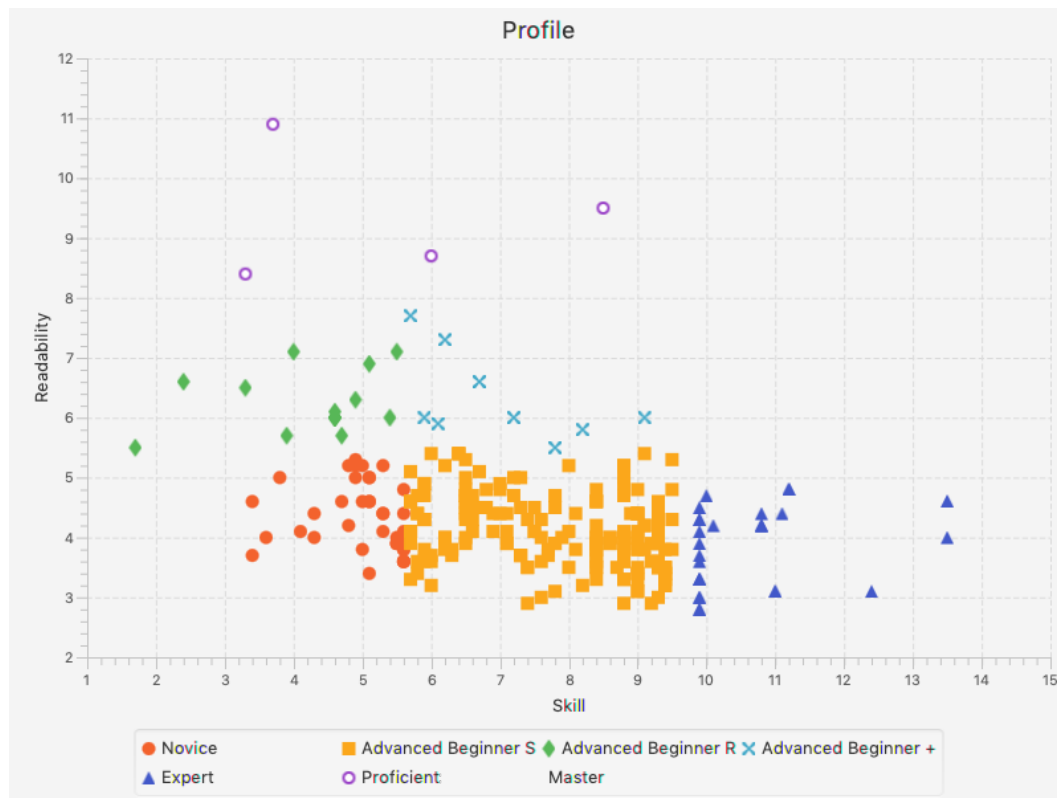■ **Figure 2** Distribution of solutions without scaling changes.

environment expectations. However, the distribution is also very tight with several points practically on top of one another. Solution A was profiled as Proficient while B and C were as Experts.

On the Figure 3 the graph shows the final distribution after all the changes explained on the previous subsections. Now most of the profiles are considered Advanced Beginner S. There is still a larger influence on the skill score, but the distribution is slightly more spread about. Solution A was profiled as Expert, solution B as Advanced Beginner + while C as Proficient.

To summarise the results of some of these changes Table 1 can be viewed. Only some of the key metrics have been listed. Solution A had been profiled as "Proficient", this is a profile leaning towards more readability than skill, however it has: The least number of skill penalties; The smallest number of statements; Far less total lines, almost a 1 to 10 factor compared to solution B; Just 2 methods and 1 class; Quite a few readability penalties and no comment lines.

By looking at these factors it's obvious the solution A leans towards skill instead of readability. In fact, we can make a direct contrast to solution C, in fact they swapped profiles. Solution C leans towards readability while keeping a good skill score, some of the factors for comparison with solution A: One skill penalty; Three more classes; Four time more the number of lines of code and of statements; 2.7 percent of lines of comment; Just 2 methods and 1 class; Quite a few readability penalties.

Finally, solution B clearly is too long compared to the others, with the most penalties and no good points in its favour. However, it doesn't necessarily lean more towards either skill or readability, hence the profile given is "Advanced Beginner +".

Figure 3 Final distribution of solutions.

Finally, and just from a programmer's direct point of view, there are some things that are easily noticeable and also serve as a validation of the adjustments made.

Solution C is clearly the most readable, it has good descriptions, spacing, more classes and methods. Solution A, was able to solve the exercise in simply 25 lines of code, and one of the smallest number of statements. On the other hand Solution B is very long, it is more complex than necessary compared to other alternatives, it seems more the work of a beginner.

To conclude, the comparison between the images shows that with this new version of PP Tool the results are more distributed across the chart.

6   Conclusion

We can anticipate several situations where it is necessary to carry out programmer's profiling: programming contests, contracting of new programmers, evaluation of programming students, analysis of source code quality for some purpose and so on. As we presented in this paper, it's possible to extract important information from the static analysis of source code in order to obtain values for parameters like skill and readability and following that approach, PP Tool infers the programmer's profile. This profile varies from novice to master passing through advanced beginner, proficient and expert. PP Tool was tested in a different more demanding environment and it did not scale up conveniently. So we extended it with some new features to obtain a finer and more efficient metrics evaluation method (also weights were tuned) in order to cope with a bigger diversity of solutions for more complex problems. Some tests

■ **Table 1** Comparison 3 solutions before and after the PP Tool scaling adjustments.

	Solution A	Solution B	Solution C
Skill PMD Penalty	0	1	1
Readability PMD Penalty	7	14	8
# Classes	1	2	3
# Methods	2	18	6
# Statements	4	60	17
Lines of Code	13	99	52
Percentage of Comment	0	2.3%	2.7%
Total Lines	26	214	73
# Declarations	4	16	10
Profile - Before	Proficient	Expert	Expert
Profile - After	Expert	Advanced Beginner +	Proficient

were made, as discussed in this paper, showing that the accuracy of the new version of our programmer's profiling tool was actually improved. The direction for future research will include the generation of detailed feedback on programmers performance based on the bad practices detected. The idea is to open the possibility to use PP Tool not only for profiling but as a recommendation tool that will contribute to improve the quality of programmer's code specially for students that are learning their first programming language.

References

1   T Flowers, Curtis Carver, and J Jackson. Empowering students and building confidence in novice programmers through Gauntlet. *34th Annual Frontiers in Education*, pages T3H/10–T3H/13 Vol. 1, November 2004.

2   Markus Fuchs and Christian Wolff. Improving programming education through gameful, formative feedback. *2016 IEEE Global Engineering Education Conference (EDUCON)*, pages 860–867, 2016.

3   Maria Hristova, Ananya Misra, Megan Rutter, and Rebecca Mercuri. Identifying and correcting Java programming errors for introductory computer science students. *ACM SIGCSE Bulletin*, 35(1):153–156, 2003.

4   Weizhi Huang, Wenkai Mo, Beijun Shen, Yu Yang, and Ning Li. Automatically Modeling Developer Programming Ability and Interest Across Software Communities. *International Journal of Software Engineering and Knowledge Engineering*, 26(09n10):1493–1510, 2016. `doi:10.1142/S0218194016400143`.

5   Daniel Novais, Maria Joao Varanda Pereira, and Pedro Rangel Henriques. Program analysis for Clustering Programmers' Profile. In M. Ganzha, L. Maciaszek, and M. Paprzycki, editors, *Proceedings of the 2017 Federated Conference on Computer Science and Information Systems (FedCSIS)*, pages 701–705. PTI; IEEE, 2017. FedCSIS, Prague, Czech Republic, Sep 03-06, 2017. `doi:10.15439/2017F147`.

6   Daniel José Ferreira Novais. Programmer profiling through code analysis. Master's thesis, University of Minho, December 2016.

7   Emília Pietriková and Sergej Chodarev. Profile-driven Source Code Exploration. *Computer Science and Information Systems (FedCSIS), pp. 929-934, IEEE.*, 2015.

8   Nghi Truong, Paul Roe, and Peter Bancroft. Static analysis of students' Java programs. In *Proceedings of the Sixth Australasian Conference on Computing Education-Volume 30*, pages 317–325. Australian Computer Society, Inc., 2004.

Beyond Classical Parallel Programming Frameworks: Chapel vs Julia

Rok Novosel

Faculty of Computer and Information Science, University of Ljubljana
Večna pot 113, 1000 Ljubljana, Slovenia
novosel.rok@gmail.com

Boštjan Slivnik[1]

Faculty of Computer and Information Science, University of Ljubljana
Večna pot 113, 1000 Ljubljana, Slovenia
bostjan.slivnik@fri.uni-lj.si

─── **Abstract** ───

Although parallel programming languages have existed for decades, (scientific) parallel programming is still dominated by Fortran and C/C++ augmented with parallel programming frameworks, e.g., MPI, OpenMP, OpenCL and CUDA. This paper contains a comparative study of Chapel and Julia, two languages quite different from one another as well as from Fortran and C, in regard to parallel programming on distributed and shared memory computers. The study is carried out using test cases that expose the need for different approaches to parallel programming. Test cases are implemented in Chapel and Julia, and in C augmented with MPI and OpenMP. It is shown that both languages, Chapel and Julia, represent a viable alternative to Fortran and C/C++ augmented with parallel programming frameworks: the programmer's efficiency is considerably improved while the speed of programs is not significantly affected.

1 Introduction

To implement a parallel algorithm or to write a parallel application, most programmers would use Fortran or C/C++ and a parallel programming framework that best suits the target parallel computer's architecture. Hence, MPI and OpenMP would be used for programs designed to run on distributed and shared memory computers, respectively, or OpenCL/CUDA if the computation must run on a GPU. In 2019 this remains de facto approach to parallel programming even though parallel programming languages have been around for decades.

Among parallel programming languages, Fortran stands out as in its 2018 version it includes a wide range of constructs supporting data parallelism and concurrency. Otherwise, many languages died away, e.g., SISAL, ZPL and Fortress, or faded into obscurity, e.g., X10. Nevertheless, it has always been claimed that languages supporting parallel programming will one day boost parallel application development [8], and were therefore studied and analyzed [6, 16].

---

[1] The corresponding author.

In this paper writing parallel programs in two programming languages, namely Chapel and Julia, is considered. Chapel has been developed at Cray, a traditional supercomputer manufacturer, and has been intended for increasing supercomputer productivity and parallel programming from the start. With its explicit support for declaring a topology of processor unit the program is to be run on it is a border case of a domain-specific language. Julia is rather different. Designed for programming high-performance computational science, it is definitely a general programming language. It is a dynamic language with garbage collection and uses just-in-time compilation. Hence, a list of characteristics usually not associated with the fastest possible execution.

Chapel and Julia are compared one against the other and against the combination of C and a selected parallel programming framework. The comparison is performed by implementing the algorithms for using three selected problems (Section 2) and evaluating the programming process and the final result (Section 3). Although this approach is far from new [12, 4], it has been called for more studies like this as no better methodology for comparing programming languages is available today [26].

2   Test Cases

2.1   *k*-Means Clustering

The k-means algorithm [21], also known as Lloyd's algorithm, is a method of finding clusters in a dataset. Its aim is to organize n points in d-dimensional space into k clusters: in the end, each point should belong to the cluster with the nearest mean (centroid) according to the Euclidean distance. The algorithm starts by creating initial clusters and then iteratively updating them. There are multiple ways of initializing the clusters: despite sophisticated methods like kmeans++ [2], we choose random points from the dataset to serve as initial clusters. Each update performs two steps:

1. Compute the centroid of a cluster.
2. Reassign points to be in the cluster with the nearest centroid.

The iterative procedure ends when no points exchange clusters, or it reaches a maximum iteration limit. Fig. 1 illustrates the iterations of the k-means algorithms where $n = 1000$ and $k = 3$.
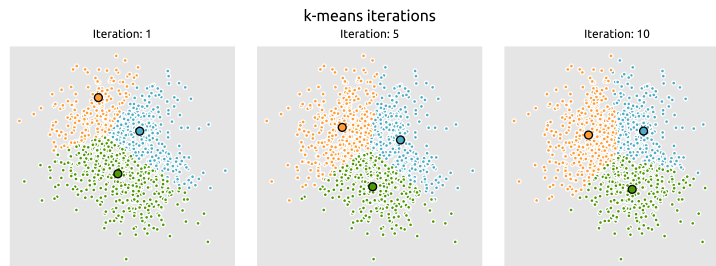
Figure 1 Iterations of the k-means clustering where $n = 1000$ and $k = 3$.

The algorithm is parallelized by distributing the points among the parallel workers and then independently calculating the nearest cluster. We compute global centroids by combining the local centroids of each worker. We decided to parallelize the k-means algorithm since it serves as an example of the map-reduce pattern [11]. It can be generalized to other machine learning algorithms that can be parallelized using the same pattern [10].
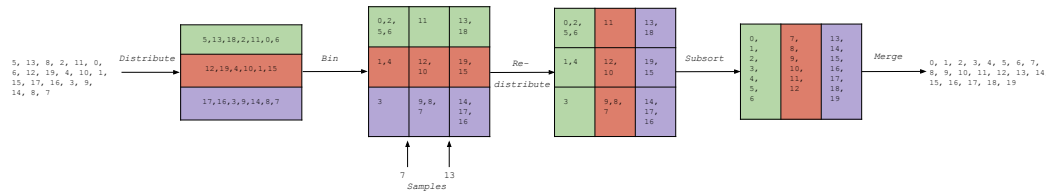
2.2 Samplesort

Sorting is a ubiquitous operation in computer science that has to handle large quantities of data in a short amount of time. That is why efficient parallel implementations are necessary. Samplesort [14] provides a good benchmark for testing whether new parallel languages are able to handle the mentioned constraints.

Samplesort is a divide-and-conquer sorting algorithm. It addresses the limitations of the Quicksort algorithm by allowing more than one partitioning element. Partitioning elements are determined by sampling the elements of the input array. Samplesort partitions the elements into m bins by constructing a $m \times m$ matrix where m is the number of parallel workers. Each column in the matrix corresponds to a bin. The entire algorithm consists of the following steps:

1. Sample $m - 1$ elements from the input array.
2. Distribute elements among workers.
3. Each worker partitions the local array into m bins according to the selected samples.
4. The bins matrix is re-distributed, and each worker takes ownership of a column.
5. Each worker sorts the elements in its column.
6. Finally, the columns are merged to get the sorted array.

The choice of the actual sorting algorithm in the fifth step is arbitrary. We used the Quicksort algorithm. Fig. 2 demonstrates sorting 20 input elements with three parallel workers. Each of the differently colored rows or columns indicate that a worker can execute them in parallel.

Figure 2 Samplesort with 20 elements and 3 parallel workers.

2.3 *n*-Body Simulation

The n-body problem requires that positions and velocities of n interacting particles is computed, usually in discrete time intervals. It is regularly used when evaluating parallel languages and frameworks [23, 25] as it was classified as a parallel "dwarf" [3]. The solution to the n-body problem is found by simulating the behavior of the particles.

The simplest way of simulating particle behavior is by calculating the effect of all other particles on a specific particle. This results in a $O(n^2)$ algorithm, which is computationally expensive for a large number of particles. The key to lowering the computational cost is to group nearby bodies and treat them as a single body. If it is far enough, we can approximate the gravitational effect of the group by using the center of its mass. The Barnes-Hut algorithm [5] achieves this by using the octree data structure [22] to split the particle space into cubic cells. Once the octree is constructed, the algorithm traverses the tree structure for each body and calculates the gravitational effect of each node. If the bodies contained in the node sub-tree are sufficiently far away, the effect is approximated by their center of mass. Otherwise, the algorithm reaches the leaf nodes which contain the original bodies. In
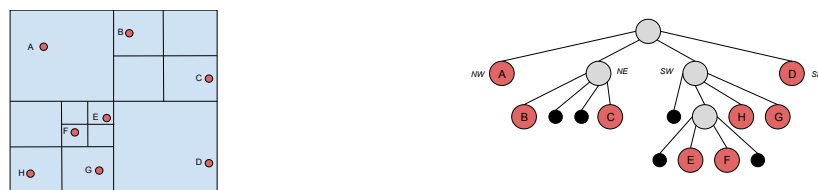
Figure 3 A 2-dimensional representation of the particle space and the corresponding Barnes-Hut quadtree: particle space divided into quadrants (left) and Barnes-Hut quadtree (right).

Fig. 3(a), we used 2-dimensional particle space to simplify the explanation. A 2-dimensional particle space is divided into quadrants and the Barnes-Hut algorithm constructs a quadtree as illustrated in Fig. 3(b).

We used different parallelization strategies for shared and distributed memory implementations. For shared memory, we statically divided the 3rd level of the octree (containing 64 nodes) among the threads. Each thread computes the necessary subtrees and the main thread combines the subtrees into the Barnes-Hut octree. For the distributed implementation we followed the algorithm outlined in [24]. Its main advantage is that it can balance the amount of work for each worker using orthogonal recursive bisection [13]. The algorithm works as follows:

1. Decompose the particle space using orthogonal recursive bisection so that each cell contains equal amount of work.
2. Each worker owns one cell and the particles within it.
3. Each worker builds a local octree from the particles (locally essential tree).
4. Next, it sends its local octree nodes to any other worker that might need them.
5. It receives the necessary octree nodes from other workers and inserts them into his octree.
6. Each worker can now compute the positions and velocities independently.

3 Evaluation

We evaluated the parallel capabilities of Chapel and Julia by implementing the test cases from the previous section. For each test case we implemented an algorithm targeted for a shared and a distributed memory computer. We used the programming language C with OpenMP for shared memory and MPI for distributed memory as the baseline for comparison. Since Julia is using JIT compilation, we also discarded the first two execution times to properly "warm-up" the JIT compiler.

We used Chapel version 1.18 and Julia version 1.1. For shared memory, we used Ubuntu 18.04, AMD Ryzen 7 2700X (8 cores, 16 threads, and 3.7GHz) processor and 16GB of RAM. For distributed memory, we used 32 HP DL160 G6 servers, each running Ubuntu Server 11.04 with Intel Xeon 5520 processor and 6GB of RAM.

3.1 *k*-Means Clustering

We generated 4 datasets for evaluation purposes. Each dataset contains 256 clusters with 128-dimensional points. The results in Fig. 4 indicate that the performance bottleneck in the k-means clustering is the computation of the Euclidean distance. Julia has a highly optimized linear algebra module based on the LAPACK library [1]. This is the main reason why we were able to outperform the C shared-memory implementation. The Julia distributed
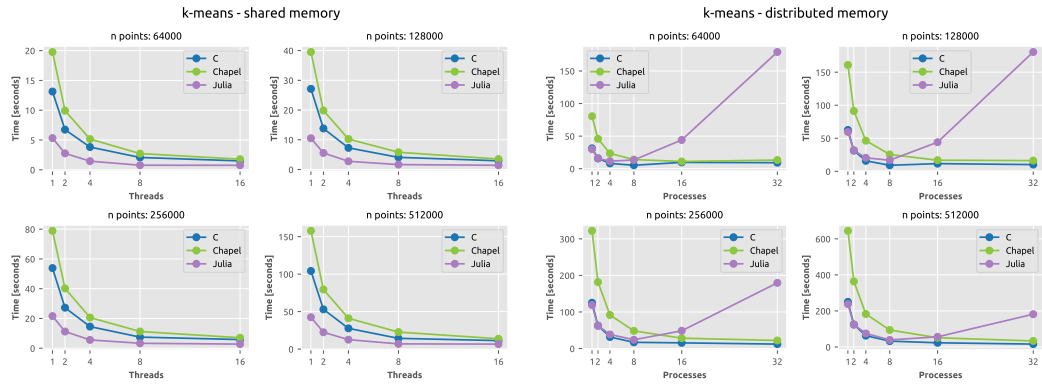
(a) k-means clustering benchmark for the shared memory implementation.

(b) k-means clustering benchmark for the distributed memory implementation.

■ **Figure 4** k-means clustering benchmarks.
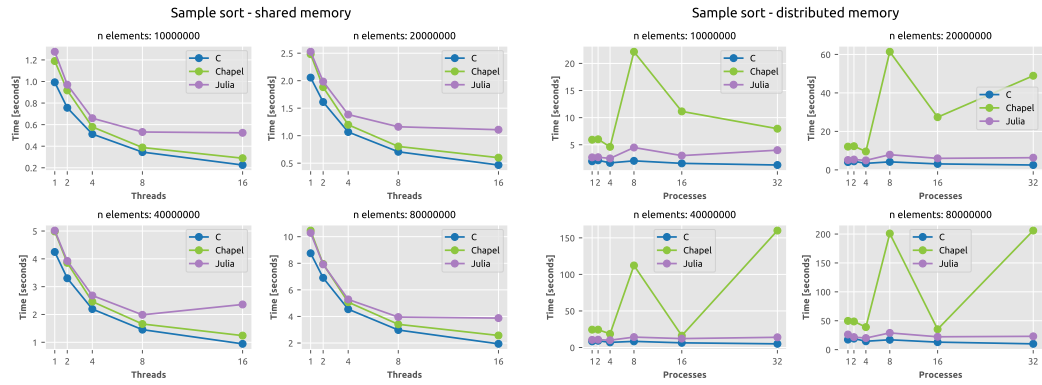
(a) Samplesort benchmark for the shared memory implementation.

(b) Samplesort benchmark for the distributed memory implementation.

■ **Figure 5** Samplesort parallel benchmarks.

memory implementation suffered from a significant overhead when spawning remote processes and communicating through remote channels. Based on Hoare's CSP [15], a channel is a FIFO data structure enabling remote processes to send and receive data.

Increasing the number of processes only exacerbated the problem. To eliminate the overhead we would have to benchmark a much larger dataset.

3.2 Samplesort

Sorting efficiency depends highly on the input elements themselves. Therefore, we could not pre-generate the input arrays. For each input size, we generated 100 random arrays and measured the execution time for each array. The final result was the average of all the execution times. The results are in Fig. 5.

In Julia, we add multi-threading by prefixing a for loop with the `@threads` macro. The macro wraps the body of the for loop in a closure and splits the iterations between the available threads. Using variables captured in a macro closure is a performance concern and is still an open issue in Julia [19]. The recommended solution, for now, is to extract the body
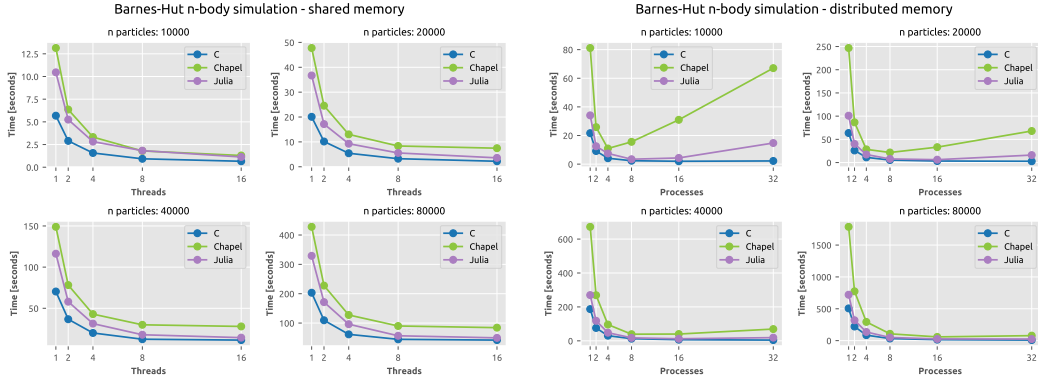
(a) Barnes-Hut n-body simulation benchmark for the shared memory implementation.

(b) Barnes-Hut n-body simulation benchmark for the distributed memory implementation.

Figure 6 Barnes-Hut n-body simulation benchmarks.

of the for loop in a separate function and thus bypass the macro closure. It is not a perfect solution and that is why we still see a deviation from the C and Chapel results in shared memory implementations.

The Chapel distributed memory implementation caused the most issues. The standard way of diagnosing distributed implementations is by using the `CommDiagnostics` module, which outputs all the implicit Chapel communication. It did not output any unnecessary or unintended communication. Our reasoning is that the Chapel compiler and the underlying GASNET [7] communication library are not able to optimize the transfer of large arrays during sorting. In spite of the mentioned issues, we found that using Chapel domains we were able to elegantly solve the samplesort problem.

3.3 *n*-Body Simulation

For the n-body simulation, we generated equally distributed particles in a pre-defined cube. The results are in Fig. 6. In the shared memory implementations we observed consistent results for all test cases. The execution times of C with OpenMP and Julia both running with 16 threads are almost equal. It would be interesting to see if Julia could outperform C and OpenMP if both are given more threads.

The algorithm we chose for the distributed memory implementations was designed for the SPMD (single program, multiple data) parallel model. As such, MPI was the ideal target when the algorithm was designed. With Julia and Chapel, we were forced to replicate the SPMD behavior. In Julia, it was relatively easy to replicate the SPMD model using channels. On the contrary, Chapel does not have a way for processes to directly communicate with each other. All communication has to be done through distributed global arrays. We used sync variables to prevent multiple processes from altering the same element in the array. Sync variables are roughly analogous to mutex locks in other languages with the exception that a sync variable is itself a lock. Any basic type (integer, float, etc.) can become a sync type by prefixing it with the `sync` keyword. Chapel does not support sync variables on complex types like arrays or objects. For complex types, we had to create two arrays one containing values and one containing sync variables. Considering the orchestration behind the Chapel implementation it is not surprising that for a small amount of data and a large number of processes it does not perform well. Increasing the amount of data hides the overhead from sync variable contention.

It is also important to note that both Julia and Chapel do not support fixed-sized arrays out-of-the-box. Using regular (dynamic) arrays was a huge performance bottleneck in the n-body simulation and the k-means clustering implementations. We used the `StaticArrays` package in Julia but resorted to using tuples in Chapel. Further research is necessary to design n-body simulation algorithms that would better fit Julia and Chapel.

4 Conclusion

In general, we had little to no issues with our C implementations. Once we eliminated all segmentation faults C implementations needed only minor improvements to run efficiently. In contrast, Chapel and Julia implementations were easier to write initially but required a lot of efficiency improvements and fine-tuning afterwards.

Julia provides a small but powerful set of parallel features. The multi-threading module is still marked as experimental, but we were still able to get good results with it. The Julia development team is planning to replace the current multi-threading module with an implementation based on parallel depth-first scheduling [9]. The main two issues with Julia were type instability and unexpected memory allocations. If Julia is not able to infer the type of variable it resorts to runtime type checking which adds significant overhead. The solution was to always annotate all variables with types to avoid runtime type checking. The Julia standard library contains multiple macros for code inspection and benchmarking. Examples include `@code_warntype` which outputs variables with ambiguous types and `@time` which outputs execution time and total memory allocated. Julia provides a web page [20] containing performance tips that have to be followed rigorously if performance is an issue.

Chapel provides a broad spectrum of parallel functionality. It can serve as an excellent introductory language for parallel and distributed programming courses. It includes enough the high-level features from other languages so that it immediately feels familiar to novices. Parallel concepts can be easily demonstrated using built-in constructs like domains, sync variables, parallel iterators, etc. Chapel itself is still under heavy development. Developers are currently trying to implement as many parallel features as possible. Consequently, the performance of Chapel programs is not as important right now. That causes it to lag behind C and Julia in our benchmark tests. During the development of this paper, we also found multiple bugs and performance issues in the compiler [17, 18]. We also experienced that the Chapel compiler is fairly slow when compiling large applications. As an example, our distributed n-body simulation takes roughly 1 minute to compile. During the implementation process, this prevented us from quickly iterating and examining new solutions.

The authors realize that the definitive comparison of languages for parallel computing would require a randomized controlled trial as advocated in [26]. Nevertheless, the results presented here can be understood as a justification for carrying out such experiment.

References

1 E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. LAPACK *Users' Guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA, 3rd edition, 1999.

2 David Arthur and Sergei Vassilvitskii. k-means++: The advantages of careful seeding. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 1027–1035. Society for Industrial and Applied Mathematics, 2007.

3 Krste Asanovic, Ras Bodik, Bryan Christopher Catanzaro, Joseph James Gebis, Parry Husbands, Kurt Keutzer, David A Patterson, William Lester Plishker, John Shalf, Samuel Webb Williams, et al. The landscape of parallel computing research: A view from berkeley. Technical report, EECS Department, University of California, 2006.

4   Henri E. Bal. A comparative study of five parallel programming languages. *Future Generation Computer Systems*, 8(1–3):121–135, 1992.

5   Josh Barnes and Piet Hut. A hierarchical O($N \log N$) force-calculation algorithm. *Nature*, 324(6096):446, 1986.

6   A. P. W. Böhm and R. R. Oldehoeft. Two Issues in Parallel Language Design. *ACM Transactions on Programming Languages and Systems*, 16(6):1675–1683, 1994.

7   Dan Bonachea and P Hargrove. GASNet Specification, v1.8.1. `https://gasnet.lbl.gov/dist/docs/gasnet.html`, 2017.

8   Bradford L. Chamberlain, Sung-Eun Choi, Steven J. Deitz, and Lawrence Snyder. The high-level parallel language ZPL improves productivity and performance. In *Proceedings of the IEEE International Workshop on Productivity and Performance in High-End Computing*, pages 66–75, 2004.

9   Shimin Chen, Phillip B Gibbons, Michael Kozuch, Vasileios Liaskovitis, Anastassia Ailamaki, Guy E Blelloch, Babak Falsafi, Limor Fix, Nikos Hardavellas, Todd C Mowry, et al. Scheduling threads for constructive cache sharing on CMPs. In *Proceedings of the nineteenth annual ACM symposium on Parallel algorithms and architectures*, pages 105–115. ACM, 2007.

10 Cheng-Tao Chu, Sang K Kim, Yi-An Lin, YuanYuan Yu, Gary Bradski, Kunle Olukotun, and Andrew Y Ng. Map-reduce for machine learning on multicore. In *Advances in neural information processing systems*, pages 281–288, 2007.

11 Jeffrey Dean and Sanjay Ghemawat. MapReduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

12 John T. Feo, editor. *A Comparative Study of Parallel Programming Languages: The Salishan Problems.* North Holland, New York, NY, USA, 1992.

13 Geoffrey C Fox. A graphical approach to load balancing and sparse matrix vector multiplication on the hypercube. In *Numerical Algorithms for Modern Parallel Computer Architectures*, pages 37–61. Springer, 1988.

14 W Donald Frazer and AC McKellar. Samplesort: A sampling approach to minimal storage tree sorting. *Journal of the ACM (JACM)*, 17(3):496–507, 1970.

15 Charles Antony Richard Hoare. *Communicating sequential processes.* Prentice-Hall, 1985.

16 Ken Kennedy, Charles Koelbel, and Hans Zima. The Rise and Fall of High Performance Fortran: An Historical Object Lesson. In *Proceedings of the Third ACM SIGPLAN Conference on History of Programming Languages*, pages 7–1–7–22, 2007.

17 Chapel Programming Language. Chapel Issue: Performance issues when running n-body simulation. `https://github.com/chapel-lang/chapel/issues/11333`. Accessed: 2019-04-24.

18 Chapel Programming Language. Chapel Issue: Using with clause in coforall loop for distributed k-means. `https://github.com/chapel-lang/chapel/issues/12006`. Accessed: 2019-04-24.

19 Julia Programming Language. Julia Issue: performance of captured variables in closures. `https://github.com/JuliaLang/julia/issues/15276`. Accessed: 2019-04-24.

20 Julia Programming Language. Julia Performance Tips. `https://docs.julialang.org/en/v1/manual/performance-tips/index.html`. Accessed: 2019-04-22.

21 Stuart Lloyd. Least squares quantization in PCM. *IEEE transactions on information theory*, 28(2):129–137, 1982.

22 Donald Meagher. Geometric modeling using octree encoding. *Computer graphics and image processing*, 19(2):129–147, 1982.

23 Lars Nylons, Mark Harris, and Jan Prins. Fast n-body simulation with CUDA. `https://developer.download.nvidia.com/compute/cuda/1.1-Beta/x86_website/projects/nbody/doc/nbody_gems3_ch31.pdf`.

24 John K Salmon. *Parallel hierarchical N-body methods.* PhD thesis, California Institute of Technology, 1991.

25 Jaswinder Pal Singh, Wolf-Dietrich Weber, and Anoop Gupta. SPLASH: Stanford parallel applications for shared-memory. *ACM SIGARCH Computer Architecture News*, 20(1):5–44, 1992.

26 Andreas Stefik and Stefan Hanenberg. Methodological Irregularities in Programming-Language Research. *Computer*, 50(8):60–63, 2017. `doi:10.1109/MC.2017.3001257`.

Knowledge Representation of Crime-Related Events: a Preliminary Approach

Gonçalo Carnaz
Department of Informatics, University of Évora, Portugal
d34707@alunos.uevora.pt

Vitor Beires Nogueira
Department of Informatics, University of Évora, Portugal
vbn@uevora.pt

Mário Antunes
School of Technology and Management, Polytechnic Institute of Leiria, Portugal
INESC-TEC, CRACS, University of Porto, Porto, Portugal
mario.antunes@ipleiria.pt

— Abstract —
The crime is spread in every daily newspaper, and particularly on criminal investigation reports produced by several Police departments, creating an amount of data to be processed by Humans. Other research studies related to relation extraction (a branch of information retrieval) in Portuguese arisen along the years, but with few extracted relations and several computer methods approaches, that could be improved by recent features, to achieve better performance results.

This paper aims to present the ongoing work related to SEM (Simple Event Model) ontology population with instances retrieved from crime-related documents, supported by an SVO (Subject, Verb, Object) algorithm using hand-crafted rules to extract events, achieving a performance measure of 0.86 (F-Measure).

1 Introduction

We are living in an era of data overloading, produced by machines and humans, and spread over the World Wide Web (WWW). This data has different formats, such as text documents, that is retrieved from heterogeneous sources. Therefore, different approaches have been released during the last decades that extract relevant information. These computer methods extract information in the form of named-entities (Named-Entity Recognition systems), relation extraction (Open Information Extraction or Traditional Information Extraction methods), or semantic roles (Semantic Role Labeling methods).

The extracted named-entities and relations/events could be represented by a knowledge base, such as ontologyíes that are conceptual models that aim to represent a particular domain, building concepts, notions or properties that represent the knowledge that exists in such domain.

In this paper, we present the ongoing work related to an approach to the SVO algorithm using hand-crafted rules to extract events and a posterior analysis regarding the SEM ontology population with the extracted events and named-entities.

The remainder of this paper is organized as follow: section 2 describes the related work regarding information retrieval and ontologies; in section 3 describes the background knowledge regarding the SEM ontology; in section 4 presents the Natural Language Processing (NLP) pipeline setup to support the relations/events extraction; in section 5 describes the relations/events extraction using the SVO algorithm; in section 6 discusses the results obtained and finally the section 7 explains the future work and conclusions.

2 Related Work

In this section we analyze the previous works related to Information Retrieval (IR) using the Portuguese language. In 2008, Mota et al. [14] proposed the SEI-Geo System to extract *part-of* relationships between geographic entities, using hand-crafted patterns to detect geographic entities. SeRELeP [2] proposed to recognize three different types of relationships (*occurred*, *part-of*, and *identity*) supported by heuristic rules applied to linguistic and syntactic features. The REMBRANDT [3] system aims to identify 24 different relations types using hand-crafted rules-based and supported by two knowledge bases: DBpedia and Wikipedia.

Garcia et al. [9] proposed in 2011, extracting *occupation* relationship instances over Portuguese texts. Training sentences using a Support Vector Machines classifier where each word evaluated (the lemma and the POS-tag) by computing the syntactic dependencies between words, using a syntactic parser.

In 2014, Souza et al. [17] proposed a supervised OIE (Open Information Extraction) approach for extracting relational triples from Portuguese texts (using Corpus CETENFolha).

Collovini et al. [5] proposed in 2016, an evaluation of the Conditional Random Fields (CRF) classifier to extract relations between named-entities, such as Organizations, Locations, or Persons from Portuguese texts.

In 2017, Ricardo Rodrigues [16] proposed the RAPPORT system, a Portuguese Question-Answering System that uses a NLP pipeline with a fact extraction task (based on syntax and semantic patterns).

Along the years, several works were proposed related to ontologies applied to the criminal domain, and how to represent concepts/terms retrieved from crime-related documents. Despres et al. [18] proposed in 2004, the alignment of terms from a legal domain and a core ontology, generating a legal ontology from a European community legislation text (reuse of LRI-Core [1] and DOLCE[1]).

Casanovas et al. [4] developed in 2007 an Ontology of Professional Judicial Knowledge, called *OPJK*. Based on a manual selection of relevant terms from legal questions and modeled according to the the *DILIGENT* methodology. Using the *PROTON* [6] ontology as an upper-level ontology. Additionally, a methodology was presented to build a multilingual semantic lexicon for the law. Additionally, the Eurovoc thesaurus is integrated for project lexicon enrichment purposes [19]. Francesconi et al. [8] aimed to ensure that legal drafters and decision makers lead to control over a legal language, specified by *DALOS* Knowledge System. The *DALOS* project is divided into ontological and lexical layers. A domain ontology supports the Ontological Layer; and the *LOIS* database supports lexical Layer.

In 2009, Hoekstra et al. [20] proposed a legal core ontology that was part of the Legal Knowledge Interchange Format, known as *LKIF* Core Ontology, as a core in a legal knowledge system. Saskia et al. [20] described a system called *OWL Judge* using OWL 2 reasoning

---

[1] See `http://www.loa.istc.cnr.it/old/DOLCE.html` [Accessed: April 2019].

for legal assessments, where norms are represented in LKIF Core Ontology, associated with design patterns for norms and user cases definition. The use case used was the University Library Regulations.

Mehmet et al. [12] proposed in 2010, a class diagram to define an ontology, applied to money laundering schemes. Using class diagram objects to represent terms and relations used in money laundering schemes, e.g., people, organization, portfolios, messages, communication medium, invoice and identification documents.

Rajpu et al. [15] tried in 2014 to find suspicious financial transactions through an expert system, based on an ontology and a set of rules. The authors created a set of classes, objects, and properties that represent the transactions to be processed by the expert system — additionally, a set of rules, using SWRL (Semantic Web Rules Language), in order to infer new knowledge through existing knowledge. Also in 2014, and using Akoma Ntoso XML schema, LKIF-Core, Legal Case Ontology, JudO and Carneades Argument Format, [10] proposed a basic and semantic annotation approach for complaints, using a Serbian Judiciary use case for validation.

In 2017, Oliveira Rodrigues et al. [7] proposed a reuse of *UFO-B* and LKIF ontology, as a concept model for property crimes representation applied to the Brazilian Criminal Code, called *OntoPropertyCrime*. Additionally, a theory of crime is formalized, called *OntoCrimeAlpha*. Mezghanni el al. [13] proposed *CrimAr* ontology is defined by a handcrafted approach, for the Arabic legal domain, supported by LRI-Core as top-level ontology. McDaniel et al. [11] proposed a framework, based on an ontology for physical evidence from a crime scene. They are adding an identity judgment (in an id-situation) aligned to legal cases. The ontology includes a situation ontology, focus on physical evidence.

3 Background Knowledge

Developing ontologies from scratch can be costly, lengthy and with several points of view for the same concept. Therefore, the reuse of existing ontologies, with slight adaptations to the study domain, could reduce time and cost regarding ontology construction. At this stage of the work, the purpose is to represent existing events in crime-related documents, such as persons, locations, organizations or time/date. Ontology could support the knowledge representation that could answer the following questions: *Who did what?, Where?, When?, How? Why?*.

Figure 1 shows the SEM ontology, that was created to model events that are present in various application domains, without making assumptions about the domain-specific vocabularies and without no connection to any domain, for example historical, cultural heritage or geographical domains. If we look carefully at newspaper news, such as crime-related news, we can easily denote that events are also central elements, because news is based on events that occurred in a fixed or extended time, with entities (such as persons, locations or objects).

The SEM ontology is based on four main classes: Events, Actor, Place and Time. The events on the SEM ontology are represented by the class *sem:Event*, this being the central class where the ontology is based. Having as properties: *eventProperty*, *eventType*, *hasSubEvent* and *SubEventOf*. The *sem:Actor* class was proposed to describe *"who or what participated, who is doing something"*. This class (a powerful entity of the domain that can activate or perform events) holds instances (retrieved from corpus) that are part of a given event, actively or passively. We can not see the actors only as persons, but also as objects, which are animate or inanimate and physical or not physical. The *sem:Place* is the class meant
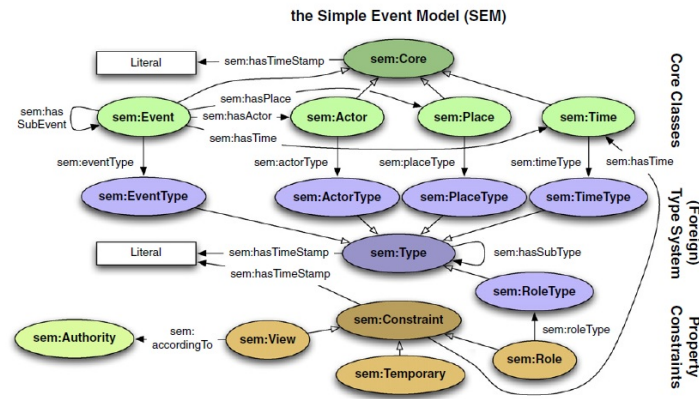
■ **Figure 1** SEM Ontology. [Retrieved from `https://semanticweb.cs.vu.nl/2009/11/sem/`]

to describe "where' is something happening. Places are locations where an Event happens. Neither do they need to have any significance apart from them being the location of an Event. Finally, the *sem:Time* is the class meant to describe "when" is something happening.

4 Natural Language Processing Pipeline Setup

Figure 2 describes a typical NLP (Natural Language Processing) pipeline. First, as data input, we have the crime-related documents retrieved from a Portuguese newspaper; in second, the first phase of the proposed pipeline (Stop-Words Removal, Sentence Detection, Tokenization, Named-Entity Recognition, POS Tagging, Lemmatization and a Dependency Parser); followed by Relation Extraction Module supported by the SVO algorithm and finally, the Ontology Module that have as input (SEM Ontology) and as output, the SEM ontology populated with instances retrieved from crime-related documents.
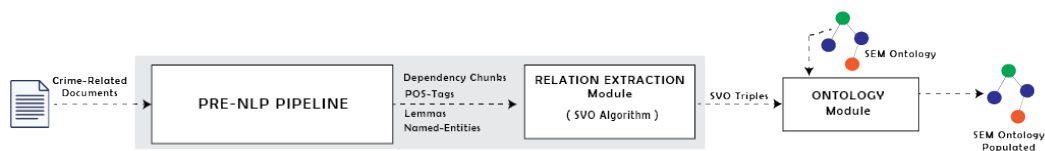
■ **Figure 2** NLP pipeline (high-level design) proposal.

For pipeline prototyping, we used the RAPPORT [16] system, with some tweaks in the NLP pipeline, such as:

- The Stop-Words Removal task was introduced to remove the words that are not relevant for the NLP processing, using an external file with Stop-Words[2];
- The Named-Entity Recognition task was training with the following corpus[3]: Amazonia + CETEMPúblico;
- Dependency parser model for Portuguese (MaltParser), using Bosque[4] (contains both European (CETEMPúblico) and Brazilian (CETENFolha) variants) ConLL treebank.

---

[2] See `https://github.com/stopwords-iso/stopwords-pt` [Accessed: April 2019]
[3] See `https://www.linguateca.pt/Floresta/corpus.html` [Accessed: April 2019]
[4] See `https://github.com/UniversalDependencies/UD_Portuguese-Bosque` [Accessed: April 2019]

5 SVO Algorithm Proposal for Event Extraction

Our approach is based on SVO (Subject, Verb, Object) sentence analysis to construct triples data by parsing crime-related sentences, using the Maltparser[5] (dependency parser) tool, and then extracting SVO triples from parser sentences. Because the Portuguese language is supported by a word order language, such as SVO, VSO, VOS, OVS or OSV, the algorithm must be adapted to identify all variations.

The algorithm 1 describes the instructions to retrieve the subjects, verbs, and objects. The entities types are identified and collected to populate the ontology, such as a person, locations, places or time/date. It is using hand-crafted rules based on syntactic and semantic features to extract relations (verbs) between entities.

Algorithm 1: SVO Algorithm.

---

1 $Subject, Object, Verb \leftarrow$ NULL;
 // Extracted named-entities with the Named-Entity Recognition module
 by its tokens
2 $Entities \leftarrow$ NamedEntityRecognitionModule(tokens);
 // Extract the tokens in CoNNL format with Dependency Parser
 (Maltparser tool)
3 $CoNLLToken \leftarrow$ DependencyParser ();
 // Identify the number of verbs by its verbal tense
4 $NumVerbs \leftarrow$ RetrieveNumberOfVerbsBySentence ();
 // For every Named-Entity Detected
5 **for** $entities \leftarrow 0$ **to** n **do**
6 **for** $CoNNLToken \leftarrow 0$ **to** n **do**
7   **if** $CoNNLToken \leftarrow$ **matches the** $(VerbTense)$ and $NumVerbs{>}0$ **then**
8 $Verb \leftarrow EventDetected$;
9   decrement NumVerbs;
10 **end**
11 **if** $CoNNLToken \leftarrow$ **contains a** $(NamedEntity)$ **then**
12   **if** $CoNNLToken \leftarrow$ **matches the** $Subject$ **then**
13 $Subject \leftarrow SubjectDetected$;
14 $SubEntType \leftarrow$ NamedEntityType;
15   **end**
16   **if** $CoNNLToken \leftarrow$ **contains the** $(RelationDependency)$ then **then**
17 $Object \leftarrow ObjectDetected$;
18 $ObjEntType \leftarrow$ NamedEntityType;
19   **end**
20 **end**
21 SEMOntologyInstances(Subject,Verb,Object,SubjEntType,ObjEntType);
22 **end**
23 **end**

Aforementioned, the algorithm is based on the extraction of the subject, verb and object, each identified with the help of the dependency parser in ConLL format (token). Also, the NER task detected the named-entities (subject or object), used to delimit the verbs,

---

[5] See http://maltparser.org/ [Accessed: April 2019]

that could be combined to create a relation extraction tagged sentence with the following format: *<**Entity** as Subject><**Verb** as event><**Entity** as Object>*.

We aim to detect the highest number of relations, by the verbs in their different forms (tense), detecting the total number of verbs in a sentence, and cyclically detecting the parallel entities. Finally, the subjects, verbs, objects, and named-entities types are used to populated the SEM ontology, using the method (SEMOntologyInstances).

6 Preliminary Results

As preliminary results, we evaluated a set of sentences related to crime extracted from a Portuguese newspaper. As an example, the following sentence was evaluated with SVO algorithm: *"Arminda Marta foi detida a 21/08/1976."* (in Portuguese) or *"Arminda Marta was arrested in 21/08/1976."* (in English), obtaining the following results:

- In Portuguese: "<Subject>**Arminda Marta**</Subject> <Verb>**deter**</Verb> <Object>**21/08/1976**</Object>";
- In English: "<Subject>**Arminda Marta**</Subject> <Verb>**arrested**</Verb> <Object>**21/08/1976**</Object>";

Also we have obtained the candidate instances to populate the SEM ontology:

- Events: **"deter, ser"** (in Portuguese), **"arrest, to be"** (in English);
- Actor: **"Arminda Marta"**;
- Time: **"21/08/1976"**.

The crime event is the result of criminal behavior and consists of the offense (an actor) to an interested protected by Law. Some of the sentences analyzed enumerated crime types, such as *"Pedro é suspeito de violar, sequestrar e agredir uma jovem em Braga."* (in Portuguese), or *"Pedro are suspect of raping, kidnapping and assaulting a young woman in Braga."* (in English). Therefore, the criminal domain has the main event - the crime (a sequence of events that lead to crime type), in its different crime types, such as violation, abduction, or aggression. Moreover, these different types are not reproduced by verbs (in some cases, because "kill" or "matar" in Portuguese, is a verb), such as homicide (in Portuguese,"homicidio") that is a male noun.

The table 1 shows the results obtained by evaluating the measures, such as the Correct Events (CE), Identified Events (EI) and Total Events (TE). The SVO algorithm obtained an F-measure result of 0.86, that is a trade-off between precision and recall measures.

Table 1 Results of the SVO algorithm evaluation.

Correct Events	Identified Events	Total Events	P	R	F1
209	221	267	0,95	0,78	0,86

Figure 3 represents the extracted entities manually populated, using the Protege[6] tool. As we can see, the SEM Ontology could represent the event extracted and the named-entities: hasActor, an object property (Arminda Marta) and hasTimeStamp, a data property (21/08/1976).

---

[6] See `https://protege.stanford.edu/` [Accessed: April 2019]
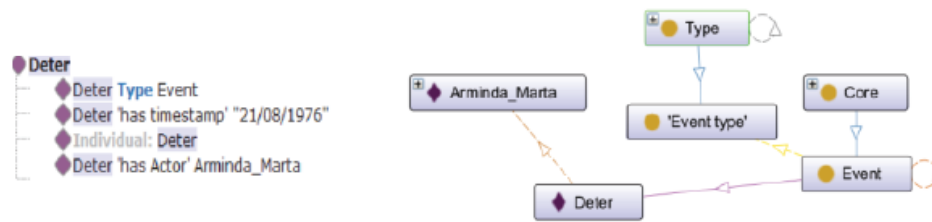
Figure 3 SEM Ontology populated example, using the sentence above enumerated.

7 Conclusion and Future Work

Concluding, the knowledge representation of already developed ontologies allows us to remove the lack of time and resources, like adapting the SEM ontology to our domain. Therefore, the representation of the events, by themselves, and the named-entities extracted from the crime-related documents, allow the representation of the event (crime and others) in SEM ontology.

The SVO approach allows us to extract, even to a limited extent, the verbs as events and the named-entities. There is a way to improve the extraction of events and named-entities and the relationship between them, where our work must continue to be improved.

For future work, we enumerated the following items to improve our work:

- improve the SVO algorithm to detect the variations, such as VSO, VOS, OVS or OSV;
- use a large dataset related to crime, created or reused, to test our approach and evaluate the performance measures (Precision, Recall, and F-Measure);
- extract crime related concepts that denotes events, such as homicide, abduction or others;
- extract relations that are important to detect geo-localization (like "District-of", "County-of", "Street", "Country") of named-entities, such as persons, objects or organizations;
- adapt the SEM ontology regarding the crime related concepts and properties;

References

1 JAPJ Breukers and RJ Hoekstra. Epistemology and ontology in core ontologies: FOLaw and LRI-Core, two. *Citeseer*, 2004.

2 José Guilherme Mírian Bruckschen, Re-nata Vieira Souza, and Sandro Rigo. Desafios na avaliação conjunta do reconhecimento de entidades mencionadas: O Segundo HAREM. *Desafios na avaliação conjunta do reconhecimento de entidades mencionadas: O Segundo HAREM*, page 436, 2008.

3 Nuno Cardoso. Rembrandt-reconhecimento de entidades mencionadas baseado em relaçoes e análise detalhada do texto. *quot; Encontro do Segundo HAREM (Universidade de Aveiro Portugal 7 de Setembro de 2008)*, 2008.

4 Pompeu Casanovas, Núria Casellas, Christoph Tempich, Denny Vrandečić, and Richard Benjamins. OPJK and DILIGENT: ontology modeling in a distributed environment. *Artificial Intelligence and Law*, 15(2):171–186, 2007.

5 Sandra Collovini, Gabriel Machado, and Renata Vieira. A Sequence Model Approach to Relation Extraction in Portuguese. In *LREC*, 2016.

6 John Davies. Lightweight ontologies. In *Theory and Applications of Ontology: Computer Applications*, pages 197–229. Springer, 2010.

7 Cleyton Mário de Oliveira Rodrigues, Frederico Luiz Goncalves De Freitas, and Ryan Ribeiro De Azevedo. An ontology for property crime based on events from ufo-b foundational ontology.

In *2016 5th Brazilian Conference on Intelligent Systems (BRACIS)*, pages 331–336. IEEE, 2016.

8   Enrico Francesconi, Pier-Luigi Spinosa, and Daniela Tiscornia. A Linguistic-ontological Support for Multilingual Legislative Drafting: the DALOS Project. In *LOAIT*, pages 103–111, 2007.

9   Marcos Garcia and Pablo Gamallo. Evaluating various linguistic features on semantic relation extraction. In *Proceedings of the International Conference Recent Advances in Natural Language Processing 2011*, pages 721–726, 2011.

10   Marko Marković, Stevan Gostojić, and Zora Konjović. Structural and semantic markup of complaints: Case study of Serbian Judiciary. In *2014 IEEE 12th International Symposium on Intelligent Systems and Informatics (SISY)*, pages 15–20. IEEE, 2014.

11   Marguerite McDaniel, Emma Sloan, William Nick, James Mayes, and Albert Esterline. Ontologies for situation-based crime scene identities. In *SoutheastCon 2017*, pages 1–8. IEEE, 2017.

12   Murad Mehmet and Duminda Wijesekera. Ontological Constructs to Create Money Laundering Schemes. In *STIDS*, pages 21–29. Citeseer, 2010.

13   Imen Bouaziz Mezghanni and Faiez Gargouri. CrimAr: A Criminal Arabic Ontology for a Benchmark Based Evaluation. *Procedia Computer Science*, 112:653–662, 2017.

14   Cristina Mota and Diana Santos. Desafios na avaliação conjunta do reconhecimento de entidades mencionadas: O Segundo HAREM. In *Desafios na avaliação conjunta do reconhecimento de entidades mencionadas: O Segundo HAREM*, chapter : Geo-ontologias e padrões para reconhecimento de locais e de suas relações em textos: o SEI-Geo no Segundo HAREM, page 436. Desafios na avaliação conjunta do reconhecimento de entidades mencionadas: O Segundo HAREM, 2008. `doi:10.1103/PhysRevB.82.193405`.

15   Quratulain Rajput, Nida Sadaf Khan, Asma Larik, and Sajjad Haider. Ontology based expert-system for suspicious transactions detection. *Computer and Information Science*, 7(1):103, 2014.

16   Ricardo Manuel da Conceição Rodrigues. *RAPPORT: A Fact-Based Question Answering System for Portuguese*. PhD thesis, Universidade de Coimbra, 2017.

17   Erick Nilsen Pereira Souza and Daniela Barreiro Claro. Extração de relações utilizando features diferenciadas para português. *Linguamática*, 6(2):57–65, 2014.

18   Sylvie Szulman Sylvie Despres. Construction of a legal ontology from a european community legislative text. In *Legal Knowledge and Information Systems: JURIX 2004, the Seventeenth Annual Conference*, volume 120, page 79. IOS Press, 2004.

19   Daniela Tiscornia. The LOIS project: Lexical ontologies for legal information sharing. In *Proceedings of the V Legislative XML Workshop*, pages 189–204. Citeseer, 2006.

20   Saskia Van De Ven, Rinke Hoekstra, Joost Breuker, Lars Wortel, Abdallah El Ali, et al. Judging Amy: Automated Legal Assessment using OWL 2. In *OWLED*, volume 432, 2008.

Distinguishing Different Classes of Utterances – the UC-PT Corpus

Mariana Gaspar Fernandes
INESC-ID, Lisboa, Portugal
Instituto Superior Técnico, Universidade de Lisboa, Portugal
mariana.gaspar.fernandes@tecnico.ulisboa.pt

Cátia Dias
INESC-ID, Lisboa, Portugal
Instituto Superior Técnico, Universidade de Lisboa, Portugal
catiadias089@gmail.com

Luísa Coheur
INESC-ID, Lisboa, Portugal
Instituto Superior Técnico, Universidade de Lisboa, Portugal
luisa.coheur@tecnico.ulisboa.pt

─── **Abstract** ───

Conversational bots are being used in many scenarios and we can find them playing museum guides or providing customer support, for instance. These bots base their answers in specific information related with their domain of expertise, but there is general information, presented in each user request that, when properly identified, could also be useful for the agent to decide what to answer. As an example, if the user is asking a question or uttering a statement, the bot's action in its search for a response will probably differ. In this paper we present three corpora for the Portuguese language – the UC-PT corpus – that can be used to help conversational bots to distinguish: a) questions from non questions, b) yes-no-questions from other types of questions; and c) personal from non-personal questions. With this information, the agent can decide, for instance, not to answer, redirect the question to a persona chatbot or decide to answer it with a simple "yes", "no" or "maybe". In addition, we benchmark the classification process in these corpora. This corpora will be made publicly available.

1 Introduction

Conversational bots (often called chatbots) have been accompanying the recent advances in Artificial Intelligence (AI). We can find them in many different scenarios [11], such as in museums or providing customer support: Edgar Smith [6], the Monserrate's Palace butler illustrates the former; IKEA's Anna, the latter. Although some recent conversational agents are data-driven, and take advantage of the latest advances in Deep Learning (e.g. [12]), they also need large quantities of data to be trained. Thus, in most scenarios this approach is not

possible. Several platforms, as for instance RASA[1], contribute to a faster development of chatbots, but, still, much manual work is needed, as corpora needs to be provided, so that the bots can learn, for instance, to identify the user intentions or the entities mentioned in his/her utterances. If the data needed to train these systems is usually dependent of the application domain, there is general and useful information that can be extracted from the user requests, which can help the chatbot to further decide how to answer, namely to identify if the user request is a question or not, if the user is posing a personal question to the agent or not, or if the question can be answered with a yes/no/maybe or not. Considering that such resource could be useful for the Portuguese community developing conversational bots, we propose, in this work, the UC-PT corpus, which is constituted of the following corpora:

- the Question vs. Non-question corpus: a corpus with 5034 utterances labeled as "question" (e.g., "O que são minhocas de pesca?" – "What are fishing worms?"), and "non-question" (e.g., "Precisaremos de manter a lei e a ordem, se isto acontecer." – "We will need to maintain law and order, if this happens.");
- the Personal vs. Impersonal corpus: a corpus with 3698 utterances labeled as "personal" (e.g., "Quem é a pessoa mais calma que conheces?" – "Who is the calmest person that you know?"), and "impersonal" (e.g., "Que diferentes tipos de plástico existem?" – "How many different kinds of plastic are there?");
- the Yes/No questions corpus: a corpus with 360 utterances labeled as "yes/no" questions (e.g., "Gostas de cangurus, querida?" – "Do you like kangaroos, sweetheart?"), and "other" (e.g., "Quantos cardeais elegem o Papa?" – "How many cardinals elect the pope?").

After building and annotating these three corpora, we calculate the inter-annotator agreement with Cohen's kappa score, obtaining two near-perfect and one perfect agreement. In addition, we benchmark the classification process. By using simple features as n-grams, we get accuracies ranging from 97% to 100%.

This paper is structured as follows: in Section 2 the three aforementioned corpora are described, and, in Section 3, we explain what we did to classify the different corpora, and which solutions provided the best results. Finally, in section 4 we present our conclusions and point to some future work.

2 Building Corpora

The sentences in all the three corpora came from different sources, namely:
- from the translation into Portuguese [5] of the widely used Li & Roth corpus [7];
- from a manual Portuguese translation of parts of the corpora of a chatbot called JustChat [10];
- from the B-Subtle corpus, a corpus built from movies Subtitles, as described in [3].

In addition, some sentences were gathered from the web, created by the authors of this work or suggested by Técnico Students at Taguspark, in a Natural Language course.

In the following we describe each one of the three corpora.

2.1 Question vs. Non-Question corpus

In this section we give a brief description of the Question vs. Non-question corpus, namely, the different formulations of questions that were gathered, as well as some examples of the non-questions.

---

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

2.1.1 Questions

Several types of questions were taken into consideration. Besides the usual *direct questions*, we also gave some room to *imperative sentences*[2] that constitute a request for information.

Regarding *direct questions*, the corpus contains several examples of the so called "Wh-questions", that is, questions that contain the keywords "quem" ("who"), "onde" ("where"), "porquê, porque" ("why"), "o quê, qual" ("what"), "o quê, qual" ("which"), "quando" ("when") and "como" ("how"). Examples of such questions are:

- "Quem é Alan Turing?" – "Who is Alan Turing?";
- "Qual é o nome abreviado do Mississippi?" – "What is the nickname for the state of Mississippi?";
- "O que é a viscosidade?" – "What is viscosity?";
- "Quando foi travada a batalha de Somme?" – "When was the battle of Somme fought?";
- "Por que motivo foi inventado o fecho de correr?" – "Why was the zipper invented?".

We also included in the corpus wh-questions that present a possibility, like an imagined scenario, and then inquire something with that scenario in mind (e.g., "Se o mundo inteiro estivesse ouvindo, que dirias?" – "If the whole world was listening, what would you say?"). In addition, we also addressed questions whose answer can be a simple "Sim" ("Yes") or "Não" ("No") (e.g., "Gostas de ler?" – "Do you like to read?" or "Andas na escola?" – "Do you go to school?"), including questions that are only one word (e.g., "Jantar?" – "Dinner?"). Moreover, questions that contain two possibilities of answer separated by the connector "or" (choice questions), were also added to the corpus (e.g., "Do que gostas mais: factos ou ficção?" – "What do you like more: facts or fiction?").

In what concerns *imperative sentences* that constitute a request for information, or ask for a description or definition of something, several cases were included in the corpus. Some examples are:

- "Mencione um cetáceo." – "Mention a cetacean.";
- "Diga o nome da organização que é presidida por um Conselho de Segurança." – "Say the name of the organization that is presided by a security counsel.";
- "Descreva a aparência do músico Finlandês Salonen." – "Describe the Finnish music personality Salonen's appearance.";
- "Defina cosmologia." – "Define cosmology.".

Finally, we opted to add some cases where several questions are formulated in the same entry. The reason for this is that sometimes people ask several questions related to each other in a row (e.g., "Se tivesses de escolher, qual animal de uma quinta gostarias ser? Porquê? Podes fazer o som?" – "If you had to pick, which farm animal would you like to be? Why? Can you do its sound?").

In summary, the utterances labeled as questions encompass: choice questions, wh-questions, yes/no questions and imperative sentences.

2.1.2 Non-questions

The non-questions part of the corpus is constituted of sentences such as:

- "A ideia é os dez formarem um círculo de protecção em torno do possuído." – "The idea is that the ten form a circle around the possessed.";

---

[2] Sentences that are an order, an instruction or a request to do something [1].

- "O David precisa de ir ao lançamento de um filme." – "David needs to go to a movie launch.";
- "Deixou a faculdade de direito, não tem emprego." – "Left law school, has no job.";
- "Não é motivo para renegar a família." – "It is no motive to renegate the family.".

2.1.3 Some Statistics

We randomly split the corpus in two, one part for training and one part for testing. The training corpus contains 4526 entries, from which 2280 are labeled as "question" and the remaining 2246 as "non-question". The testing corpus contains 508 entries from which 264 are "non-questions" and 244 are "questions". Extra details can be found on Table 1[3].

Table 1 Statistics about the Question vs. Non-question corpus.

	Training Set	Testing Set	Whole Corpus
Number of Tokens	42614	4552	47166
Number of Unique words	7741	1509	8253
Average Word Length	4.26	4.31	4.26
Number of Characters	162610	17464	180074
Number of StopWords	13651	1442	15093
Number of Words	36812	3901	40713

2.2 Personal vs. Impersonal Questions

In this section we explain what can be found in the Personal vs. Impersonal corpus.

2.2.1 Personal Questions

In European Portuguese, the way personal questions are formulated depend on who we are talking to. Hierarchy and age difference, among others, will lead to more formal/informal conversations. When two people engage in an informal conversation, the second person of the singular is usually used; otherwise the third person of the singular is employed. For instance, if we ask a friend if he likes to read, we would ask "Gostas de ler?" or "Tu gostas de ler?", but if we asked a person we do not know or has one of the aforementioned differences, we would ask "Gosta de ler?" or "Você gosta de ler?" (being the latter in a more Brazilian Portuguese style). In the English language all these questions translate to "Do you like to read?". In the corpus for personal and impersonal questions we took these cases into account. Examples are (the first one is an example of formal speech, and the second of informal speech):

- "Diga algo que fez em criança que os seus pais não sabem." – "Say something that you did as a child that your parents do not know of.";
- "Diz 1 coisa que desejavas mudar em ti." – "Say 1 thing that you wish to change in yourself.".

Other examples of personal questions that can be found in the corpora are related with: a) situations in which the user presents a scenario and then asks what the other person would do considering it (e.g., "Se tivesses que comer um guaxinim ... como irias cozinhá-lo?" – "If

---

[3] In this and in the remaining corpora, the number of tokens and the number of characters take into consideration punctuation.

you had to eat a raccoon... how would you cook it?"); b) personal preferences (this can be regarding to movies, food, among other personal tastes) (e.g., "Qual é o teu filme favorito?" – "What is your favourite movie?"); c) family, friends, romantic relationships, among others (e.g., "O que me podes dizer sobre um dos teus avós?" – "What can you tell me about one of your grandparents?"); d) feelings, opinions, beliefs and visions in life: (e.g., "Achas que é correto namoriscar se tens namorado/namorada?" – "Do you think it is ok to flirt if you have a boyfriend/girlfriend"); e) past and/or a person's experience (e.g., "Indica 1 coisa que te faz falta das férias quando eras criança." – "State 1 thing that you miss of the vacations you had when you were a child."); f) what a person wears and his/her appearance, habits, skills, personal info/data, personal options, facts about personal life, etc. (e.g., "És bom a escrever na tua língua materna?" – "Are you any good at writing in your mother tongue?"). In conclusion, personal questions are questions about the interlocutor's personal matters, such as his opinions, feelings, memories, home city, friends, among others. If the questions are about the personal life of a person that is not an acquaintance of the interlocutor and if that question is not asking for an opinion, then it is not personal.

2.2.2 Impersonal Questions

As for the impersonal questions, they are mostly factoid questions extracted from the aforementioned translation of Li & Roth corpus for Portuguese. Some examples include:

- "O que faz com que um tornado gire?" – "What makes a tornado turn?";
- "Quais são os dois países cuja costa faz fronteira com a Baía de Biscaia?" – "What two countries' coastlines border the Bay of Biscay?";
- "Que actor casou com a irmã de John F. Kennedy?" – "What actor married John F. Kennedy's sister?".

2.2.3 Some Statistics

The personal/impersonal training corpus has 3329 queries, from which 1746 are labelled as "impersonal" and the other 1583 are labelled as "personal". The testing corpus has 369 entries from which 205 are tagged as "impersonal" and the other 164 are tagged as "personal". More detailed statistics about this corpus can be found on Table 2.

■ **Table 2** Statistics about the personal and impersonal corpus.

	Training Set	**Testing Set**	**Training + Testing Set**
Number of Tokens	33407	3733	37140
Number of Unique words	5714	1173	6099
Average Word Length	4.38	4.29	4.37
Number of Characters	132257	14413	146670
Number of StopWords	10197	1117	11314
Number of Words	29272	3248	32520

2.3 Yes/No Questions vs. Other

In this section we explain what are Yes/No questions and we provide some examples of the questions of this kind that can be found in this corpus. We also present some examples of the questions that cannot be answered with a simple "yes", "no" or "maybe".

2.3.1   Yes/No Questions

Examples of Yes/No questions are:

- "Lês muito?" – "Do you read a lot?";
- "Gostas de dançar?" – "Do you like to dance?";
- "Tens dinheiro?" – "Do you have money?";
- "Ontem choveu?" – "Did it rain yesterday?".

Notice that, in the set of Yes/No questions, one can find questions constituted of one single word (e.g., "Pizza?").

2.3.2   Other

As to the questions labeled as other, they are like the ones presented in Section 2.1, excluding the Yes/No ones. Under the label "other" we can find questions such as "Wh-questions", imperative sentences, among others. Here are some examples (extracted from the corpus):

- "Indique um pesticida." – "State a pesticide.";
- "Em que cidade se encontra a Basílica de São Pedro?" - "In what city is Saint Peter's basilica located?";
- "És de que clube?" – "Of what club are you?";
- "Quanto custou o Túnel da Mancha?" – "How much did the channel tunnel cost?".

2.3.3   Some Statistics

The training corpus has 320 entries, from which 157 are labeled as "yes/no-question" and the other 163 as "other". As for the testing corpus it contains 40 entries from which 19 are labelled as "other" and the other 21 are labelled as "yes/no-question". More detailed information can be found in Table 3.

Table 3 Statistics about the Yes/No Question and Other corpus.

	Training Set	Testing Set	Training + Testing Set
Number of Tokens	2058	261	2319
Number of Unique words	723	147	787
Average Word Length	4.59	4.69	4.60
Number of Characters	8199	1057	9256
Number of StopWords	530	65	595
Number of Words	1711	216	1927

2.4   Inter-annotator Agreement

A random sample of 100 queries was selected from each of the above corpus, rendering for each corpus 50 queries for each label. This sample was given to three different annotators (one for each corpus) which in turn gave their annotation for each query. Upon doing this, the results were compared with the original labelling, made by a single annotator, using the Cohen's kappa coefficient metric (using the implementation provided by scikit-learn [9]). The results obtained can be found in Table 4, and show that, for the Question vs. Non-question corpus, there is a perfect agreement between the two annotators. As for the other two corpus there is a near-perfect agreement.

Corpus	Cohen Kappa Score
Question and Non-Question	1.00
Personal and Impersonal	0.88
Yes/No and Other	0.98

Some examples of sentences in which the annotators did not agree in the Personal vs. Impersonal corpus are:

- **impersonal**: "Porque estamos na Terra?" – "Why are we on Earth?";
- **personal**: "Quando saem os objectos de Halloween nas lojas no teu país?" – "When do the Halloween objects come out in your country's stores?".

As these questions could be answered with both opinions and facts, it is understandable that that ambiguity causes a non-agreement between the two annotators.

The only sentence in which the annotators did not agree in the Yes/No question vs. Other corpus was:

- **yes/no-question**: "Do Stephen King? Um filme de terror?" – "From Stephen King? An horror movie?".

This question could be both answered with a simple yes or no, and with a movie. Which explains why the annotators did not agree on the label.

3   The Classification Process

We conducted the classification process by creating models with Naive Bayes (NB) and Support Vector Machines (SVM). We used NB due to its simplicity and SVM for its proven effectiveness in the task of text classification, as discussed in [2]. We have experimented with all the three Vectorizers (CountVectorizer, HashingVectorizer and TfidfVectorizer), with all implementations of NB (Complement, Bernoulli, Multinomial and Gaussian), and with SVM where we have tried 4 distinct kernels (linear, rbf, sigmoid and poly). As features we used Unigrams, Bigrams, Trigrams and combinations of them. We have also tested with a custom tokenizer (the TweetTokenizer available in the Natural Language Toolkit [4]). The evaluation metric that we used was accuracy. The best results ranged from 98.10% to 100% (accuracy) in the three corpus for the aforementioned train/test partitions. TweetTokenizer led to the best results in all the corpora. 100% accuracy was obtained for both the Question/Non-Question and the Yes/No question corpus, with SVM + linear kernel, CountVectorizer and Unigrams as features. As for the Personal vs. Impersonal corpus, the obtained accuracy was of 98.10%, with SVM + linear kernel, TfidfVectorizer, and Unigrams + Bigrams or Unigrams + Bigrams + Trigrams as features. For the previous experiment, NB led to similar results (Complement).

Additionally, we performed a 10-fold cross-validation on each corpus (training and testing corpus together) using the classification pipeline that presented the best results for each corpus in the train/test partition classification. With these pipelines, we obtained a range of accuracies between 97% and 100%.

Results show that it is not complicated to discriminate between the proposed different types of utterances.

Although we do not want to impose an order in the usage of these corpora, an obvious scenario is: first, the model trained in the Question vs. Non-Question corpus is used to check whether a query is or is not a question. If it is a question it can be further classified as a yes/no question vs. other and, in addition, as personal or impersonal.

4   Conclusions and Future Work

We presented the UC-PT corpus (which will be made available upon request), which is built on three different corpora: one that is constituted of questions and non-questions, one that has personal and impersonal questions, and the last one that comprises questions that can and cannot be answered with a yes/no/maybe. We had very high inter-annotator agreements with the three corpora. We also tested several classifiers and obtained accuracies higher than 97%. As future work we would like to use the created models to improve a conversational bot as well as create new corpora or rules to get a classification of a sub-type of questions or non-questions (wh-questions, declarative sentences, or-questions, and so on and so forth). Additionally, the Question vs. Non-Question corpus can be enriched with indirect questions such as "Pergunto-me qual é a capital da Finlândia." – "I wonder what is the capital of Finland." (could also be added to the Personal vs. Impersonal corpus) which indirectly ask for an answer about something. Finally, we will focus on answers' classification, and, in particular, we will explore how to take advantage of relations between answers [8].

───── **References** ─────

1   Bas Aarts. *Oxford Modern English Grammar*. Oxford University Press, 2011.
2   Charu C. Aggarwal and ChengXiang Zhai. *A Survey of Text Classification Algorithms*, pages 163–222. Springer US, Boston, MA, 2012.
3   David Ameixa. Say Something Smart - ensinando um chatbot a responder com base em legendas de filmes. Master's thesis, Instituto Superior Técnico, Lisboa, Portugal, 2015.
4   Edward Loper Bird, Steven and Ewan Klein. *Natural Language Processing with Python*. O'Reilly Media Inc., 2009.
5   Ângela Costa, Tiago Luís, Joana Ribeiro, Ana Cristina Mendes, and Luísa Coheur. An English-Portuguese parallel corpus of questions: translation guidelines and application in SMT. In *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC-2012)*. European Language Resources Association (ELRA), 2012.
6   Pedro Fialho, Luísa Coheur, Sérgio dos Santos Lopes Curto, Pedro Miguel Abrunhosa Cládio, Ângela Costa, Alberto Abad, Hugo Meinedo, and Isabel Trancoso. MEET EDGAR, A TUTORING AGENT AT MONSERRATE. In *ACL*, Proceedings of the 51st Annual Meeting of the Association f, August 2013.
7   Xin Li and Dan Roth. Learning Question Classifiers. In *Proceedings of the 19th International Conference on Computational Linguistics - Volume 1*, COLING '02, pages 1–7, Stroudsburg, PA, USA, 2002. Association for Computational Linguistics. `doi:10.3115/1072228.1072378`.
8   Ana Cristina Mendes and Luísa Coheur. An Approach to Answer Selection in Question-Answering Based on Semantic Relations. In *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, July 16-22, 2011*, pages 1852–1857, 2011. `doi:10.5591/978-1-57735-516-8/IJCAI11-310`.
9   F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.
10 Maria Pereira. Just.chat - dos sistemas de pergunta/resposta para os chatbots. Master's thesis, Instituto Superior Técnico, Lisboa, Portugal, 2015.
11 Maria João Pereira, Luísa Coheur, Pedro Fialho, and Ricardo Ribeiro. Chatbots' Greetings to Human-Computer Communication. *CoRR*, abs/1609.06479, 2016. `arXiv:1609.06479`.
12 Yiping Song, Rui Yan, Yansong Feng, Yaoyuan Zhang, Dongyan Zhao, and Ming Zhang. Towards a Neural Conversation Model With Diversity Net Using Determinantal Point Processes. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, New Orleans, Louisiana, USA, February 2-7, 2018*, 2018.

Digital Collection Creator, Visualizer and Explorer

Luís F. Martins[1]
Algoritmi-DI, University of Minho, Portugal
lmfmrt@gmail.com

Cristiana Araújo
Algoritmi-DI, University of Minho, Portugal
decristianaaraujo@hotmail.com

Pedro Rangel Henriques
Algoritmi-DI, University of Minho, Portugal
pedrorangelhenriques@gmail.com

Abstract

In this paper we introduce and discuss a recent project, called CortaColaEspia, aimed at extending with some extra relevant features the 'Ontology-based Collection Processor' developed previously in the context of a Compilers course. The basic processor, based on the OntoDL tool, was able to read the ontological description of a small collection of objects (cards, pencils, toys, etc.) and produce automatically a web-based exhibition space to display the objects, providing a conceptual navigation through them. The extension under discussion is intended to create a new DSL to describe the details of the exhibition room organization (what concepts and relations to show; where and how to show them; etc.). A second objective consists of a new module to merge two collections, or to enrich a collection with extra information about the collected objects. The last requirement is the incorporation of a natural language processor to analyze the objects' captions or short inscriptions in order to extract information that can create knowledge about a specific domain, a society or an epoch.

1 Introduction

Collecting objects is a very old human habit and the digitalization of collections to manage them on the computer and exhibit the objects on the Web is a very common practice nowadays.

Describing the objects of a museum collection and defining all the exhibition details of those objects, allowing them to be linked conceptually in order to tell stories and transmit knowledge to the space visitors, is a demanding task requiring time and effort but chiefly the know-how of a museum's expert like a curator.

We believe that the exhibition of a small collection gathered by an individual, not a professional, is valuable to the community. Handling these small collections can be useful in different contexts: exhibition of familiar collections in personal web pages; creation of small learning spaces to be used in educational and recreative activities; construction of specific catalogues to be used in e-commerce activities; and so on . . .

---

[1] Corresponding author

In this context, the paper presents a light alternative for the web publishing and navigation of small collections resorting to Domain-Specific Languages (DSL) as a twofold artifact: a specification language for the actual collection, and the definition of the representation itself [1]. To describe our approach, we first present a DSL, OntoDL, to define collections (car miniatures, toys, pencils, sugar bags, coins, post stamps, etc.) based on ontologies. The ontology specifies the collection's knowledge domain creating a set of concepts (to classify domain objects), and defining relationships (hierarchical or not) among the main concepts (or classes); then individuals (or concrete objects) are linked to concepts using an 'instance-of' special relation. This DSL allows to describe the conceptual level of the collection and the collection objects that will populate the ontology in a very easy way using a light notation [2]. The proposed approach uses a second DSL, CollExhibDL, to describe the exhibition space, resorting again to the ontology concepts and relations, and never concerned with the instances. After discussing the role of the ontology and the related DSLs to define the collection and how to show it, we propose a system that automatize both phases, the collection upload (or the information ingestion) and its exhibition on a web site (information dissemination) [9].

This system will be built based on: the attribute grammars that specify the referred DSLs; and on the compiler generator ANTLR [8] that analyzes the grammars and generates automatically the Java code that implements the desired processors.

The present section (Section 1) serves as an introduction to the paper. Here we present the work context and project motivation, provide a brief description of the proposed system, and then we introduce the case studies used in order to test CortaColaEspia and prove its feasibility and versatility. This paper covers two of the modules of the system we intend to build. A short presentation of the already developed OntoDL language and processor is the content of Section 2. The specific language CollExhibDL and the Web exhibition rooms generated from the ontology and the room description are discussed in Section 4. A detailed overview of the system's architecture is presented in Section 3; this section explains the system modules and the connections among them. Before the conclusion, in Section 6, another section (Section 5) discusses related work.

1.1   Case Studies: Introduction

In order to prove the usefulness of our system, three case studies were carefully chosen:

Religious Greeting Cards. The first case study was the major case for the development of the OntoDL tool already built. The collection consists on greeting cards given to attendants of religious events, such as holy communions and weddings, that usually contain information about the event and where it took place, the person and saint involved, and a handwritten message.

Sugar Packets. Some brands of coffee make limited edition collections of sugar packets with curiosities about a specific event or about a region and its uses or history. Usually each element of these collections has a small description of an individual of the subject matter and a small graphic/picture representing the topic described.

Tickets. The last case study is a collection of tickets for sport events or music concerts that contain useful information to prove this project concept, like locations, company advertising and clubs or artists.

2 An Ontology for Digital Collections

In this section we try to explain the first developed module of this system, the OntoDL processor and our perspective in answering this module objectives. In this section we first briefly introduce the definition of Ontology and then we present the formal language, OntoDL, to describe, in a natural and syntectic way, ontologies at a abstract level (just concepts and relations) or at a concrete level adding indicuals to instantiate classes. In a third subsection, we will use OntoDL language to specify the ontologies for the three collections chosen as case-studies.

2.1 Ontologies

An ontology may be described as a body of formally represented knowledge, based on *conceptualization*, making possible to represent an abstract simplified view of the world according to the wished purpose [2].

The reasons to develop an ontology may be summarized in five items:

- To share common understanding of the structure of information among people or software agents;
- To enable the reutilization of domain knowledge;
- To make domain assumptions explicit;
- To separate domain knowledge from the operational knowledge;
- To analyze domain knowledge.

2.2 Case Study: OntoDL Description

OntoDL is a DSL, developed by our group, that allows for a fast and easy specification of the structure of any given ontology and also describe its population.

Listing 1 Example of an Ontology in OntoDL.

```
1   Ontology Packets
2
3   concepts {
4       Packet[title:string,imagePath:string,
5               packetNumber:int,description:string],
6       Brand[name:string],
7       Collection[name:string, series_size:int, year:int],
8       Event[name:string, category:string]
9       }
10  individuals {
11      c1, p11, chris
12      }
13  relations {
14      belongs, madeBy, reference
15      }
16  triples {
17      Packet = belongs => Collection;
18      Packet = madeBy => Brand;
19      Collection = reference => Event;
20      Collection = belongs => Brand;
21      chris =iof=> Brand[name='Christina'];
22      c1 =iof=> Collection[name='Expressoes Tipicas do Norte',
            series_size='9'];
```

```
23        c1 =belongs=> chris;
24        p11 =iof=> Packet[title='Olha estes dois a picar o ponto',
25            imagePath = 'expressoes_christina_namorar.png',
26            packetNumber='3', description='Significado: Olha estes
                  dois a namorar'];
27        p11 =madeBy=> chris;
28        p11 =belongs=> c1  }
```

As shown in Listing 1, an ontology is described starting in the first line with its name. In the following lines, 3 to 15, the concepts, the individuals and the relations that characterize the ontology are declared. Please notice that in the first block 'concepts' attributes (defined by a pair name and type) can be assigned to concepts under definition. In the last block, lines 16 to 28, the triples (linking concepts and individuals using the relation) are finally specified. The OntoDL file, shown in Listing 1, is then processed by an OntoDL compiler. That compiler, produced automatically by ANTLR from the OntoDL Attribute Grammar, validates all the input specification and generates both a OWL file and a DOT file (to draw the ontology graph).

When processed by our OntoDL tool, we successfully generated a graph representation for each case study. For the sake of space we just included, in Figure 1, the graphic visualization for the second case study. The graphs obtained for all the case studies can be found at `www.di.uminho.pt/~gepl/CORTACOLAESPIA`.
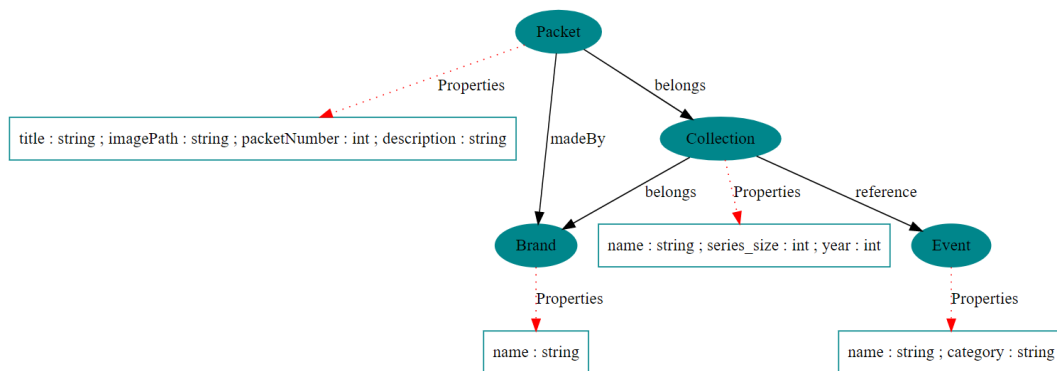
**Figure 1** OntoDL Generated Sugar Packets Graph.

3   CortaColaEspia: the Architecture

The system we proposed to develop can be seen has a combination of three main modules, one for the knowledge specification, one for the viewing specification and another for the merging of knowledge.

The architecture of the system is represented by the block diagram shown in Figure 2.

The CPM (in blue) is composed by the OntoDL tool explained above. It generates an OWL file providing an OntoDL collection specification is given. This module consists on a DSL that specificies the structure of an ontology and its population.

The CVNM (in green) allows to generate a XSL Transformation file to be applied in the previously generated OWL, given a CollExhibDL collection visualization file. This module allows the user to specify which fields to show and which to give more emphasis to. Also the user can specify the whole organization of the information to display, such as whether and where it show an index or where the images are shown.
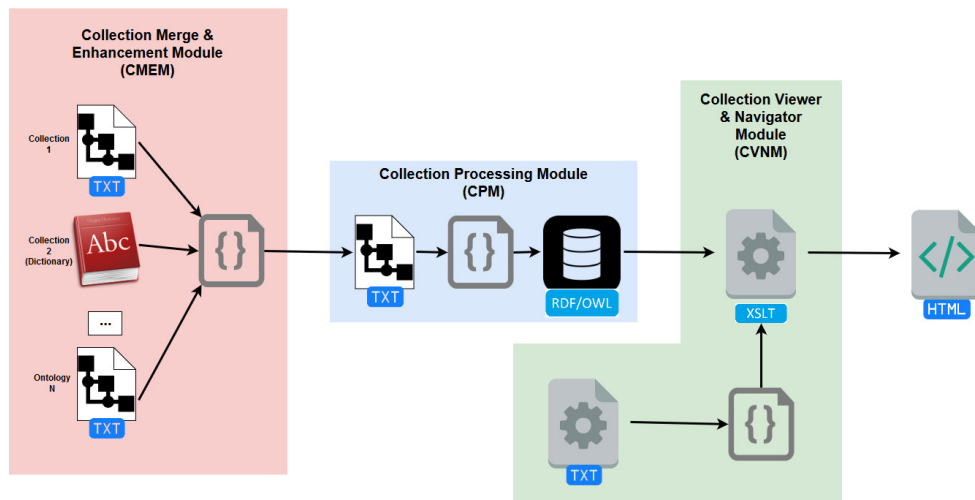
Figure 2 System Architecture.

The CMEM (in red) allows to merge multiple similar collections and successfully generate a single OntoDL file to be processed as a whole. In the merging module the main objective is to merge data from diverse ontologies, mapping them, in order to complement or expand the existing knowledge.

4 A Website to Explore Collections

This section is splitted into two subsections. The first one is used to introduced a new formal language to describe the layout desired to exhibit a specific collection given its ontological definition, and the second part contains an example of the use of that language to state how the collection (second case-study, specified in the previous section) shall be shown.

4.1 CollExhibDL

This subsection is divided into the DSL Approach that was taken and the Grammar developed to process said DSL.

The components required to achieve a basic template that works for every collection are:

- The concept that we are looking to display;
- Every attribute of that concept that we want to see;
- Every related concept of interest.

Focusing on this three aspects, a second and new DSL was designed aimed at controling the layout and the final details of the Webpage to be created to exhibit the collection.

This DSL is composed of three key elements: a **card** tag, that works as a container or a menu of a respective class and which can be contained in another instance of card; a **show** tag, descendant of card allowing to show attributes of that class; and a **relation** tag, also descendant of card allowing to relate its specific instance with its parent card.

An instance of this DSL is represented in the Listing 2.

Listing 2 Snippet of a basic specification written in CollExhibDL

```
1   view 'Sugar Packets' green
2   card '#Brand'
3     show '#name'
4     card '#Collection' relation '#belongs'
5       show '#name'
6       card '#Packet' relation '#belongs'
7         show '#title' description 'Packet'
8         showI '#imagePath'
9         showC '#packetNumber' description 'Packet Number'
10        showC '#description' description 'Description' . . .
```

Each card is delimited by a punctuation point, allowing to have both sibling and descendant cards.

In order to process any given specification written in CollExhibDL, a grammar was developed, defining syntactic rules and semantic validations. This grammar is described in Listing 3.

Listing 3 CollExhibDL Grammar

```
1   view : 'view' label=PAL card* ;
2
3   card : 'card' concept=PAL (relation)? (show|showC|showI)* (
        subcard)* '.' ;
4   subcard : card ;
5
6   show : 'show' attr=PAL ('description' desc=PAL)? ; %% Show
        content in coll. page and index
7   showC : 'showC' attr=PAL ('description' desc=PAL)? ; %% Show
        content in coll. page only
8   showI : 'showI' attr=PAL ('description' desc=PAL)? ; %% Show
        image
9
10  relation : 'relation' rel=PAL ;
11
12  PAL: ([a-zA-Z][-_a-zA-Z0-9?]*)|('\'' ~['\'']* '\'')|('\"'
        ~['\'']* '\"') ;
13  NUM: '-'?[0-9]+ ;
14  Sep: ('\r'? '\n' | ' ' | '\t')+ -> skip ;
15  Comment: '%%'~('\n')+ -> skip ;
```

4.2   Case Studies: the Exhibition

When the previously presented ontology representations of the collections are processed by this tool, it successfully generates the HTML websites as expected. In Figure 3 we show one of those websites to illustrate the processor output. Other examples, obtained for the three case studies can be seen at `www.di.uminho.pt/~gepl/CORTACOLAESPIA`.

Figure 3 CollExhibDL Generated Sugar Packets HTML.

5   Related Work

Although anyone can find over the Web many sites exhibiting small collections – for instance, Colnect[2] that contains more than 35 instances, Portal do Colecionismo[3], Pacotada[4], or even commercial sites like Casa do Colecionador[5] – as fas as we know, there is not any automatic generation tool that is able to accept a formal description of the collection to build the site. However the project was mainly inspired in CaVa platform, developed by Ricardo Martini [6, 5], to create virtual museums from ontology-based descriptions of the desired museum.

Also, concerning the use of DSL [4, 7] to generate other family of information systems, we can cite [9, 3].

6   Conclusion

All people have hobbies, some like to watch movies, others like to play sports, others like to collect objects... Collectors usually do not only have one collection, but multiple collections of different objects. In this context, we found it important to create a system (CortaColaEspia) that can store and exhibit in Web pages collections of objects and even merge different collections.

So in a first phase we identified 3 different case studies (collections) to store in our system, and we described these 3 collections by means of ontologies using the formal language OntoDL. Following a DSL Approach, we then implemented a grammar to define a new formal language (CollExhibDL) to describe the layout desired to display a specific collection in a Web page. Given the collection ontological definition, that language can be used to declare how each collection shall be shown and navigated.

---

[2] `https://colnect.com/`, last access in April 2019
[3] `http://www.portaldocolecionismo.pt`, last access in April 2019
[4] `https://www.pacotada.com/`, last access in April 2019
[5] `http://casadocolecionador.net/`, last access in April 2019

As future work it is necessary to develop the module that allows the merge of different collections to get a richer exhibition, and then test the overall system with other collections. We also want to encourage collectors to manage their collection with our system so that we can assess CortaColaEspia with real size cases.

References

1 Ines Ceh, Matej Crepinsek, Tomaz Kosar, and Marjan Mernik. Ontology driven development of domain-specific languages. *Comput. Sci. Inf. Syst.*, 8(2):317–342, 2011.

2 Thomas R. Gruber. Toward principles for the design of ontologies used for knowledge sharing. In *International Journal of Human-Computer Studies*, pages 907–928. Kluwer Academic Publishers, 1993.

3 Milan Kosanović, Igor Dejanovic, and Gordana Milosavljevic. Applang – A domain-specific language for rapid development of data-oriented mobile applications. In *Int. Conf. of Numerical Analysis and Applied Mathematics, ICNAAM*, volume 1738, page 240003, June 2016. `doi:10.1063/1.4952022`.

4 Tomaž Kosar, Sudev Bohra, and Marjan Mernik. Domain-Specific Languages: A Systematic Mapping Study. *Information and Software Technology*, pages –, 2015. Accepted manuscript (article in Press). `doi:10.1016/j.infsof.2015.11.001`.

5 Ricardo G. Martini, Cristiana Araújo, Pedro Rangel Henriques, and M.João Varanda Pereira. CaVa: an example of the automatic Generation of Virtual Learning Spaces. In Álvaro Rocha, Hojjat Adeli, Luís Paulo Reis, and Sandra Costanzo, editors, *Trends and Advances in Information Systems and Technologies, WorldCist2018*, volume 745 of *Advances in Intelligent Systems and Computing*, pages 633–643. Springer International Publishing, Cham, March 2018. `doi:10.1007/978-3-319-77703-0_63`.

6 Ricardo Giuliani Martini. *Formal Description and Automatic Generation of Learning Spaces based on Ontologies*. PhD thesis, University of Minho, September 2018.

7 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. `doi:10.1145/1118890.1118892`.

8 Terence Parr. ANTLR. `http://www.antlr.org/`, 2016. Accessed: 2016-09-14.

9 Agnis Stibe and Janis Bicevskis. Web Site Modeling and Prototyping Based on a Domain-Specific Language. In *Computer Science and Information Technologies Vol. 751*, pages 7–21, Riga, Latvia, 2009. Scientific Paper, University of Latvia.

Urban Evolution of Fafe in the Last Two Centuries

João Filipe C. Lameiras[1]
Algoritmi-DI, University of Minho, Portugal
Jonas_Filipe_9@hotmail.com

Mónica Guimarães
Fafe Municipal Archive, Portugal
monikaguimaraes@gmail.com

Pedro Rangel Henriques
Algoritmi-DI, University of Minho, Portugal
pedrorangelhenriques@gmail.com

Abstract

Human Beings love to collect, store and preserve documents for later exploration leading to the creation of Archives. Actually, to consult municipal archives' asset, seeking information in order to explore the knowledge implicit in their documents, is the main reason for the existence of those memory institutions. On the other hand, it is known that the movement of people from dispersed living to concentration in urban environments has a strong impact both in human civilization and in the environment. This statement motivates Social Science researchers to study of urban evolution of cities. In this context and having noticed that Fafe's Archive holds an important collection of municipal records (since XIX Century) concerning the application for authorization to construct or reconstruct private or public buildings, it came up to our minds to create a digital repository with those documents enabling their analysis. An information system shall be developed around it for information retrieval and knowledge exploration; it is also desirable that this application provides features to visualize the information extracted in convenient ways, like positioning buildings over a map. This paper discusses the development of the referred Web-based system to study the Urban Evolution of Fafe in the XIX and XX Centuries, focussing on the ontology created to understand the domain to be explored. The definition of a markup language (as a XML dialect), to annotate the Archive documents in order to enable the automatic data extraction and the semantic search, is also one of the paper topics. It will be discussed that this annotation was not defined from the scratch; instead, its design followed the ontology. It is actually an ontology-driven system. At last, the state of the Web interface (the system front-end) so far developed will be presented.

1 Introduction

The study of Urban Evolution of Fafe in the XIX and XX Centuries is an interesting theme not only because of the lack of works in this area but also because of the possibility of understanding the organization of the current city. In other words, we are going to study the morphology of the city. Given the non existence of an exhaustive investigation of an urban history we will have to seek to interpret from the present formation the successive processes

---

[1] Corresponding author

of urbanization and respective extensions, juxtapositions and overlaps. More important is the diverse set of sources that allow to characterize the urbanism of Fafe. In that context, with the project here discussed, we want to prove that, with the extraction of information from documents belonging to the Municipal Archive and crossing that information using an ontology, it is possible to reconstruct the urban evolution of Fafe. With that said, the main problem that we face is that as the years go by, the mapping and buildings of cities change. However, the information of these changes is recorded in documents of different kinds, pictures, photos, maps, etc. This fact makes the study of urban evolution difficult because the information is widespread and hard to gather. So, in order to study Fafe urban evolution we need to recover and merge information of the changes and new buildings in the city during the XIX and XX centuries. It is important to create an integrated repository in digital format to enable its analysis and visual exploration. For that purpose, it is necessary to develop web-supported tools for the acquisition of the state and the location of the buildings in order to analyze the changes as the years go by. More specific our objectives are:

- Design an ontology about urbanism to define a vocabulary that covers all the project;
- Create a repository to store the information about the different buildings;
- Create a web interface to insert the buildings' data;
- Populate the repository with information;
- Allow users to search for information;
- Make available the visualization of the buildings on the map (in different years)

As a final result it is expected to have a tool that can be easily used. This tool will be very useful for historians to analyze the evolution of the city along the years.

This article is organized in 7 sections. The first section describes the motivation behind this work, as well as its goals. Its main purpose is to contextualize the project. The second section (2) introduces the main concepts of Urban Evolution, providing the basis for the work. Also the ontologies that were built, to help understand the knowledge domain underlying the project, were described. The third section (3) discusses some projects similar to the one we are developing, to give some knowledge of the works done in this area. The fourth section (4) presents a XML dialect created specifically for this project according to the concepts and relations defined in the above referred ontology. Then the annotation (with that markup language) of the archival documents, related to municipal building licenses, is discussed; that annotation is crucial to enable the data extraction from those documents. A short reference to the XML querying technology used is also made. The fifth section (5) describes the architecture of a Web system that we propose to develop in order to store in a digital repository the annotated documents enabling the knowledge exploration via a visual search. The sixth section (6) presents and describes a web page of the project, and also a search mechanism contained therein. Finally, the seventh (7) section presents the conclusions that can be drawn from the work yet done, and defines directions for future work.

2   Urbanization

Urbanization[2] is the process of changing from natural habitats to dense grey space made up primarily of buildings, roads, and accessory infrastructure accompanied by dense human populations. While many cities are well established, humans continue to build new cities or expand cities outward in a network of suburban environments. It is important to notice that urbanization is not simply about a transition from green to grey space; other abiotic

---

[2] See `https://ourworldindata.org/urbanization`.

changes – such as changes in light regimens due to artificial lighting, increased pollution, and increased impervious surfaces leading to runoff – are found in urban areas [8]. A study in 2009 [3] showed that half of the world's human population lived in cities, and that it was expected this number to grow to 66% by 2050.
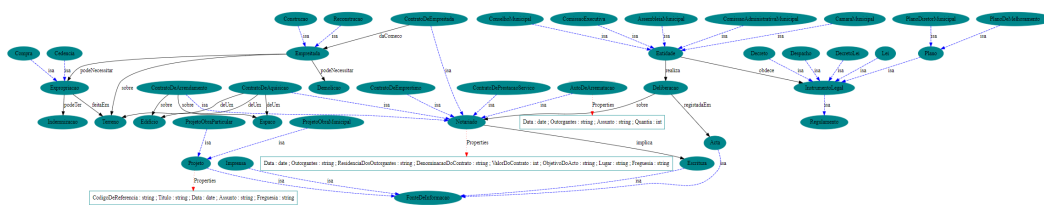
2.1 Urban Ontologies

In an initial phase, aiming at the understanding/interpretation of the archive's documents, it was necessary to analyze these sources of information collecting all the concepts found and organizing them in a ontology.

Soon, as the development of the first ontology started, it was realized that some terms should be separated to another ontology to get a better organization. Notice that the concepts were separated into two ontologies, but both ontologies share terms that make possible to connect, or interrelate, them.
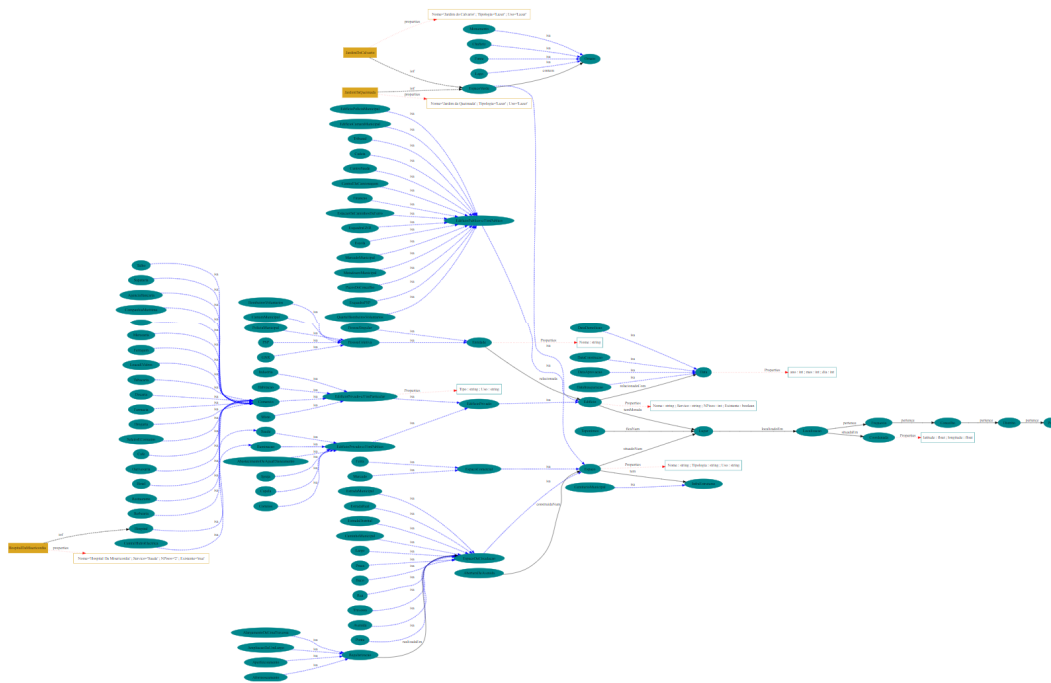
Figure 1 Ontology describing Buildings and Public Spaces.

Figure 2 Ontology describing Legal base and muncipal Deliberations.

---

[3] See `https://www.un.org/development/desa/en/news/population/2018-revision-of-world-urbanization-prospects.html` to read more about this study.

The terms about familiar homes, public builds or infrastructures, and public spaces were related to make a coherent discourse domain in a first ontology (Figure 1). The terms about legal base and municipal deliberations were related on a second ontology (Figure 2).

In more detail, the ontology in Figure 1 describes all types of buildings classifying them in public buildings with private or public services and in private buildings with private or public services; it also classifies spaces as green spaces, circulation spaces and commercial spaces. Moreover, that ontology associates attributes to each concept to better characterize them. The ontology in Figure 2 describes all the process of deliberations, legal support, and notary describing every type of contract that the entities could do in result of the deliberations.

Experts from the Urbanism domain joined the informatics technical team to help ont he decisions underlying the ontologies creation. After the identification of all the relevant concepts, relations, triples (linking an subject concept to an object concept by means of a predicate (a relation), the ontologies were written in a specific description language OntoDL (developed last year by our research); the individuals were identified ant the abstract concepts were instantiated using again OntoDL, to keep all the domain specification under a formal context. The OntoDL definitions were submitted to our OntoDL-Processor to be validated and transformed into OWL [9, 1] (the well known language commonly used for ontologies description). For each ontology a DOT ^{4} file was also generated by the referred processor to allow for a graphical visualization of the underlying graph. From the DOT files, using the Graphviz Web tool available at `http://www.webgraphviz.com/`, the graphics shown in Figures 1 and 2 were produced.

3   Related Work

In our investigation we just found a few projects similar to our proposal. In this section we discuss the two more relevant, Memo and ImagineRio.

Project MEMO

Aware of the growing restrictions on the availability of natural resources and the implications of urban growth over the territory and the environment, the MEMO Project seeks to contribute to a better understanding of the relationship between Urban Morphology and the metabolic behavior of the territory in order to support the development of guidelines for land-use planning that aim to optimize the use of natural resources. MEMO's main objective is developing a comparative analysis of Urban Metabolism of the Metropolitan Area of Lisbon (LMA), in three historical periods (1900-1950-2000), while assessing the relationship between the LMA Urban Morphology with the water and soil resources management, for each period of time under analysis. More information about the project MEMO can be found in `https://memoproject.wordpress.com/`.

ImagineRio

ImagineRio is a searchable digital atlas that illustrates the social and urban evolution of Rio de Janeiro, as it existed and as it was imagined. Views, historical maps, and ground floor plans –from iconographic, cartographic, and architectural archives– are located in both time and space while their visual and spatial data are integrated across a number of databases and servers, including a public repository of images, a geographic information system, an

---

^{4} See `https://en.wikipedia.org/wiki/DOT_(graph_description_language)`

open-source relational database, and a content delivery web system. The relationship between the various project elements produces a web environment where vector, spatial, and raster data are simultaneously probed, toggled, viewed, and/or queried in a system that supports multiple expressions of diverse data sources. It is an environment where, for example, historians can visualize specific sites both temporally and spatially, where architects and urbanists can see proposed design projects in situ, where literary scholars can map out novels while visualizing their contexts, and where archaeologists can reconstruct complex stratigraphy's. Scaled down into a mobile version, the site allows tourists and residents to walk about town while visualizing the city as it once was as well as it was once projected. More information about ImagineRio can be found in `https://imaginerio.org/#en`.

4 Annotation of Documents

With the analyses of the sources soon was realized that those sources are valuable, meaning that every single piece of text contains extractable information that is relevant for the desired study. So the solution to extract and process the information was XML. To use XML, it is necessary to create the appropriate tags. So, it was considered to use the ontologies to define the tags, since they establish all the concepts related to the subject. In Figure 3 is showed an example of a municipal document (namely a 'Contracto para Remodelação e Ampliação dos Paços do Concelho') annotated with the XML tags create specifically for that purpose.
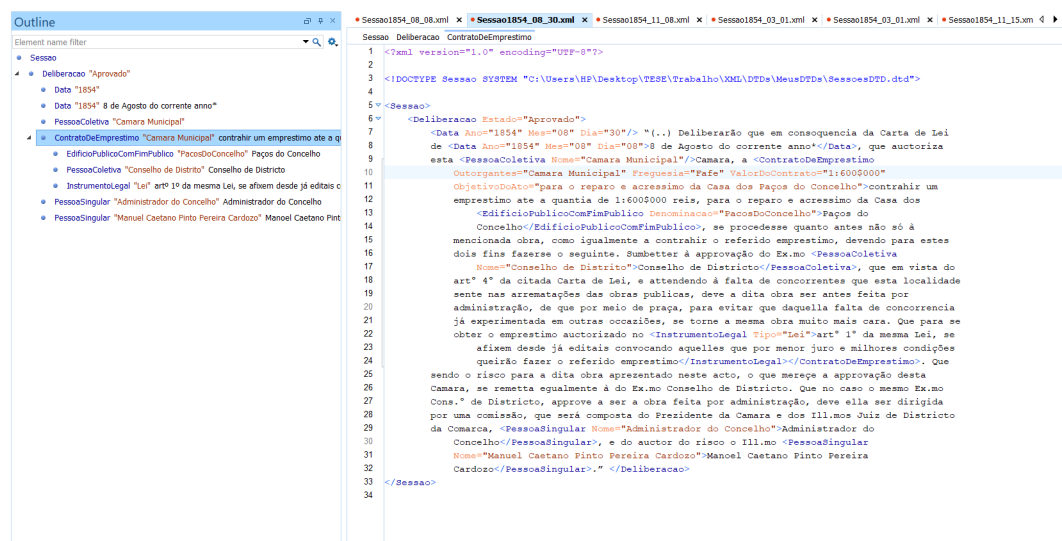

Figure 3 A municipal document fully markup with XML tags.

4.1 A XML dialect (DTD)

To create formally an XML dialect we decided to write a DTD (Document Type Definition[5]). A DTD defines the markup elements, or tags, their structure and attributes. Given an XML document and the respective DTD it is possible to verify if it is valid, that is, to check if it is *well-formed* and if it uses correct tags and has a valid structure.

[5] See for instance `https://www.xmlfiles.com/dtd/dtd-intro/`.

So the DTDs were elaborated taking in account all the possible structures that can appear in the documents like declarations, deliberations, etc. As the documents really have not a fixed structure, it was necessary to design the DTDs in such a way.

4.2   Query System

To answer the end user needs for Urban evolution information, queries are built using XPath[6] specification. XPath is a major element in the XSLT standard (for details on XML & Companion see [13]). XPath can be used to navigate through elements and attributes in an XML document. In XPath, there are seven kinds of nodes: element, attribute, text, namespace, processing-instruction, comment, and document nodes. XML documents are treated as trees of nodes. The topmost element of the tree is called the root element. XPath uses path expressions to select nodes or node-sets in an XML document. The node is selected by following a path or steps. An XPath expression returns either a node-set, a string, a Boolean, or a number.Example of one XPath-expression used in our context is `"/*//EspacoCirculacao[contains(text(),'Feira Velha')]"`. This path expression will search for a tag "EspacoCirculacao" that contains "Feira Velha" regardless of the path that "EspacoCirculacao" is in.

5   System Architecture

To store the annotated documents (recovered in the Municipal Archives) in a digital repository, retrieve them according to the end users queries, and visualize the results, we proposed to build a computer-based Web system following a back-end/front-end architectural approach, as depicted in Figure 4.
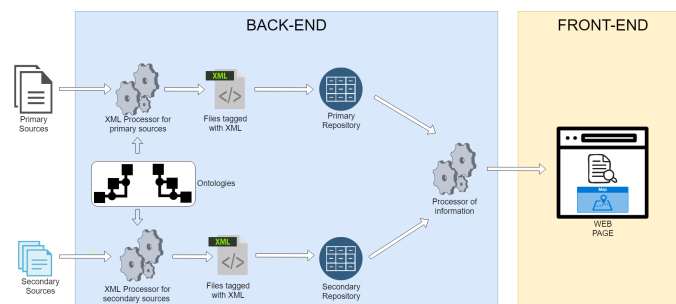
Figure 4 Architecture of the tool.

In the back-end component, digital documents (both classes, primary and secondary sources) are read by the system and markup according to the specially created DTDs (following the Urbanism or Legal Rules Ontologies). After being annotated and checked for compliance, those documents are stored in the respective digital repositories in such a way that can be retrieve later.

In the front-end (see next section for details and output illustrations) the user is able to search for information concerned with specific private or public buildings. Different search parameters will be available. The user needs is transformed into a query format that is sent to the back-end. The query results returned by the query engine are then displayed in the Web page.

---

[6] `https://en.wikipedia.org/wiki/XPath`

This is a simple and traditional architecture that can be efficiently implemented with conventional XML/HTML technologies.

6 A Web site to explore Urban Knowledge

To illustrate the work done until now and to provide more details on the project in geral, and on the ontologies in particular, the reader can have a look on the project Web page at `http://www4.di.uminho.pt/~gepl/UEF`.

In that site, and the most important reason for its construction, the ontology-driven search mechanism can be tested. The visitor can choose any term to be searched for in the documents' text body, or he can select any tag to be searched for in the documents' annotation system. The search page is shown in Figure 5.

Figure 5 An example of the results obtained from a search, in this case "Paço".

As a first output, the search engine returns in textual format all documents that satisfy the query.
A second and more interesting output, is the location on a Google map of the building referred to in the document selected for visualization. Figure 6 illustrate this feature.

Figure 6 An example of the results obtained in a map from a search, in this case "feira velha".

Notice that the system also allows the user to move the graphic mark on the map. The proposed position is read and saved in the repository as the user's feedback on the real building location.

7 Conclusion

As discussed along the paper, the general area of urbanism, including the cities evolution, lacks research and publications. As time passes, the city maps, and even the private or public buildings, suffer multiple and various changes. The information about these changes is recorded in various (textual or graphical) documents. The fact that this information is scattered in different sources makes the study of urban evolution difficult – this challenge gave the motivation for the project here reported, which aims at gathering the dispersed data and providing a framework to understand the permanent city reshape.

The Archive of Fafe, as many other municipal archives, keeps an important fund of application forms, submitted to the town-hall by citizens or the city governors, to construct or reconstruct residential/office/municipal buildings. If linked together and made available in a digital system, those documents can be of very useful to study and understand the urbanism. This objective requires that the archive documents are converted to a digital format that enables their storage, properly classified, in a computer; however, to allow the linkage, the documents must be markup.

The approach described in the paper lays upon the use of an urban ontology (two components were presented) that provide the vocabulary used as tags for the annotation system. The tool proposed makes that process easier and faster because the information will be stored in one place and will be made available through a friendly search engine. The front-end interface will show the query results in a textual or graphical format, allowing to locate buildings in the present city plan.

The tool development is the next step. Then it will be tested carefully to assess its performance. Notice that although the development will consider the city of Fafe, the approach and engine can be applied to other cities in Portugal or other countries. So, the impact can be big and innovative.

References

1 Dean Allemang and James Hendler. *Semantic Web for the Working Ontologist: Effective Modeling in RDFS and OWL*. Elsevier Science, 2011.

2 Leonardo Benevelo. *Origenes del Urbanismo Moderno*. Celeste Ediciones, 1963.

3 Artur Ferreira Coimbra. *A terra e a memória*. Câmara Municipal de Fafe, 1997.

4 Artur Ferreira Coimbra. *Apontamentos de história local*. Câmara Municipal de Fafe, 2003.

5 M. R. G. Conzen. *The use of town plans in study of urban history*. Edward Arnold, London, 1968.

6 Michel Goossens, Sebastian Rahtz, and Frank Mittelbach. *The LaTeX Graphics Companion*. Addison-Wesley, 1997.

7 B.W. Kernighan and D.M. Ritchie. *The C Programming Language (ANSI C)*. Prentice Hall Software series, 2nd edition, 1988.

8 T. G. McGEE. *The urbanization process in the third world*. London: G. Bell and Sons, Ltd, 1971. ISBN 0713516232.

9 D. L. McGuinness and F. Van. Harmelen. OWL Web Ontology Language Overview, 2004.

10 Miguel Monteiro. *Fafe dos 'brasileiros', Perspectiva histórica e patrimonial*. ed. Autor, 2nd edition, 2004.

11 Juli Esteban Nogueira. *Elementos de ordenación urbana*. Arquitextr, 2001.

12 Nuno Portas. *Os tempos das formas*. Departamento Autónomo de Arquitetura da Universidade do Minho, vol. i: a cidade feita e refeita edition, 2005.

13 José Carlos Ramalho and Pedro Rangel Henriques. *XML & XSL: da teoria à prática*. Série Tecnologias de Informação ISBN-972-722-347-8. Editora FCA, 1st ed. edition, October 2002.

14 Maria do Carmo Ribeiro. *Braga entre a época romana e a Idade Moderna. A metodologia de análise para a leitura da evolução do espaço urbano.* PhD thesis, Universidade do Minho, 2008. Tese de Doutoramento em Arqueologia; Área do conhecimento em Arqueologia da Paisagem e do Povoamento.

15 Orlando Ribeiro. *Proemio metodológico ao estudo das pequenas cidades portuguesas.* Opúsculos Geográficos. Fundação Calouste Gulbenkian, vol. v: temas urbanos, lisboa edition, 1994.

Alexa, How Can I Reason with Prolog?

Falco Nogatz
University of Würzburg, Department of Computer Science, Germany
falco.nogatz@uni-wuerzburg.de

Julia Kübert
University of Würzburg, Department of Computer Science, Germany
julia.kuebert@stud-mail.uni-wuerzburg.de

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

Salvador Abreu
LISP and Department of Computer Science, University of Évora, Portugal
spa@uevora.pt

─── **Abstract** ───

As with Amazon's Echo and its conversational agent Alexa, smart voice-controlled devices become ever more present in daily life, and many different applications can be integrated into this platform. In this paper, we present a framework that eases the development of *skills* in Prolog. As Prolog has a long history in natural language processing, we may integrate well-established techniques, such as reasoning about knowledge with Attempto Controlled English, instead of depending on example phrases and pre-defined slots.

1 Introduction

With Amazon's Echo, voice-controlled devices became available to a large number of customers and developers. As of today, Amazon's voice-controlled intelligent personal assistant service *Alexa* is one of the most used speech-based natural user interfaces [9]. Its popularity is also due to the system's openness to developers: with *Amazon Developer*[1], it is possible to create speech-based applications even without any prior knowledge of natural language processing (NLP); new so-called *skills* can be defined just by providing example phrases. The user's speech-based request is then sent as serialised text to the developer's server. This way, the developer only needs to handle and produce text messages instead of speech, i.e. the text that is returned by the developer's server is synthesised into speech by Amazon's services and spoken back to the user.

Although this division in responsibilities – Amazon converts the speech to text and vice-versa, the developer's web-service receives and returns only text-based messages – eases the development process and lowers the bar for developers who are not familiar with NLP,

---

the resulting skills are limited to the pre-defined example phrases and capable of handling only small placeholders, the so-called *slots*. As a result, there are in fact more than 50,000 skills available[2], but their common types of interactions are only very simple tasks [8], for instance: *check the weather*, *find facts*, *set timers*, or *tell a joke.*

With a controlled natural language (CNL), it is possible to use a more expressive language, which is nevertheless defined to remain easy to parse. The CNL can be integrated with Amazon's Alexa skills by defining an intent that contains only a single slot, whose content is then entirely handled by the server. In this way, the resulting skill is capable of handling much more complicated requests.

Building on the vast experience in applying Prolog to natural language processing, we present its integration with Amazon's Alexa architecture. Prolog has good support for the CNL *Attempto Controlled English* (ACE) [2]. Its reasoner [1] allows one to extend Alexa with a skill to store knowledge, retrieve it, and even infer new knowledge. It turns out this particular skill exposes some of the current limitations when using Amazon's voice service with long, free speech in a single slot.

The remainder of the paper is organised as follows. In Section 2, we give a short introduction into the definition of a skill for Amazon's Alexa service. The basic concepts of ACE are presented in Section 3. In Section 4, we present our proposed architecture for a Prolog web-server that processes ACE via Alexa and finally, the paper ends with concluding remarks in Section 5.

2   Intents for Alexa Skills

Amazon's speech-based applications can be used by various smarthome devices. Besides the smart speakers of the *Echo* family, there are integrations for all modern mobile platforms as well as smart TVs. Their common basis are *skills* – applications that can be created and released by developers. Its usage requires an internet connection, since the recorded user's speech is sent to Amazon's servers. With the help of the *Alexa Voice Service*[3], it is converted into text, and then sent to the developer's server.

$$\underbrace{Alexa,}_{1}\ \underbrace{ask}_{2}\ \underbrace{Prolog}_{3}\ \underbrace{to\ remember\ that}_{4}\ \underbrace{Sam\ is\ a\ human}_{5}$$

Figure 1 Structure of an example intent.

A single skill provides multiple actions, the *intents*. Besides the ones defined by the developer (e.g., "tell a joke"), Amazon provides built-in intents, e.g., to ask for help, or cancel the current action. The general structure of an intent formulated by a user is given in the example of Figure 1. It consists of the following five parts:

1. *Wake word.* By speaking this word, the smarthome device is activated. It can be changed by the user to "Alexa", "Echo", or "Computer".
2. *Phrase to start a skill.* To start or use a skill, the user can use one of the phrases as defined by Amazon, e.g., "ask", "start", or "open".

---

[2] IFA 2018: Alexa Devices Continue Expansion Into New Categories and Use Cases, Amazon's Developer Blog, Sep 1, 2018 – `https://developer.amazon.com/blogs/alexa/post/85354e2f-2007-41c6-b946-5a73784bc5f3/`

[3] Alexa Voice Service – `https://developer.amazon.com/alexa-voice-service`

3. The *invocation name* is a name defined by the developer that identifies the used skill. In our example application, we use "Prolog", which could be easily changed.
4. *Intent name.* Since a skill can provide multiple actions, the user can specify the requested intent. In the example of Figure 1, we use "remember" to store knowledge in the knowledge base.
5. An intent can optionally process any number of *slot values.* These are placeholders with pre-defined data types, e.g. `AMAZON.Number`, that can be extended by self-defined slot types.

For our integration of Alexa with ACE, we use the invocation name "Prolog", and define three intents, each with a single slot: (i) "remember" takes a sentence, tests for its consistency with the prior knowledge, and stores it as part of the knowledge base; (ii) "prove" checks if the given sentence can already be inferred from the current knowledge base; and (iii) "question" can be used to ask for an answer. Based on these intents, additional actions could be defined, for instance to forget parts of the stored knowledge.

3 ACE: Syntax, Interpretation, and Reasoning

In most existing Alexa skills, the slots are expected to contain just a single word, often as part of a list of pre-defined values (e.g., an animal),[4] or free text of only a few words (e.g., "tell a joke about [topic]", where the topic is generally described in up to three words). In our application, the slots are expected to contain several words, like "Sam is a human", as seen in Figure 1 for the intent "remember". In order to process the sentence or question given in a slot, we make use of Attempto Controlled English.

3.1 Attempto Parsing Engine

The language definition of ACE consists of three parts. The *vocabulary* contains pre-defined function words (e.g., determiners, conjunctions, and query words), a basic lexicon with content words (e.g., proper names, verbs, and adjectives), and typical fixed phrases to state knowledge or ask questions about (e.g., "there is", and "it is false that"). Their combination into valid ACE sentences is defined by the *construction rules*. To avoid ambiguities, ACE defines several *interpretation rules* that specify the conceivable meaning of, e.g., relative clauses, and prepositional phrases.

In ACE, sentences always relate to the previous. To guarantee a distinct internal representation, ACE makes use of the following strategies:

- *Restricted language.* ACE consists of a pre-defined set of known words. Unknown words can be prefixed by their word-class, for instance "p:Alexa" for the proper name "Alexa". Because this is only an option for written but not spoken text, we instead make use of ACE's best-effort guess for the correct word-class.

  ACE also requires that every noun is used together with a noun marker or quantifier. That means the indefinite article "a" in "Sam likes an animal" stands for the existential quantification and is strictly needed; it is equivalent to "There is an animal that Sam likes".

  Besides all restrictions, ACE's language is not limited to just simple, declarative sentences – determined by a subject, verb, and object. Composite sentences that combine declarative sentences by constructors like "or", "not", and "if" are also allowed.

---

[4] Amazon provides several slot types for commonly used categories, e.g. `AMAZON.Animal`. These slot types increase the chance to correctly identify the spoken word.
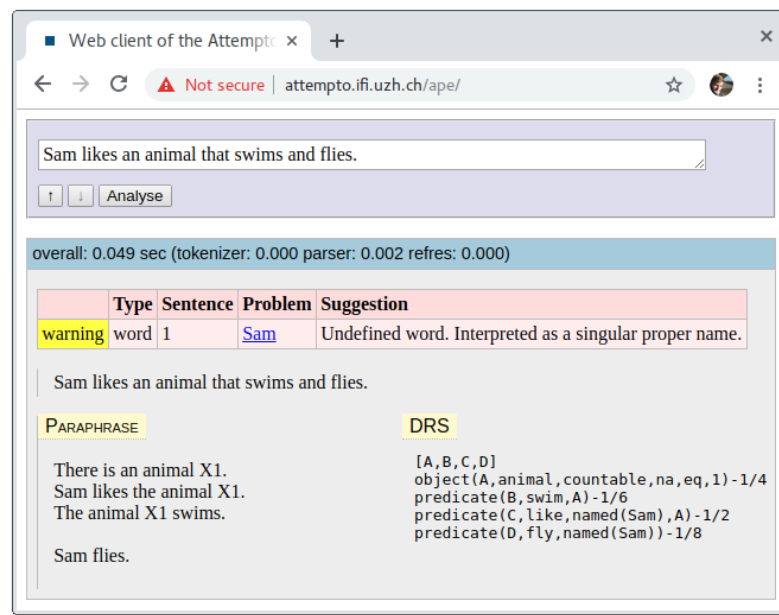
Figure 2 Screenshot of the APE web client.

- *Solving of ambiguities with constraints and interpretation rules.* The easiest way to avoid ambiguities is to constrain the language to not allow ambiguous constructs, when unambiguous alternatives are available. For instance, the sentence "Sam likes an animal that swims and flies" will have the unequivocal meaning that Sam flies. To express that the animal is able to swim and fly, ACE requires the repetition of the relative pronoun "that", so the sentence should be "Sam likes an animal that swims and that flies" instead.

  However, not all ambiguities can be safely removed from ACE without making the sentences stilted and artificial. Therefore, ACE specifies interpretation rules that define the scope of sub-sentences, quantifiers, etc.

- *Paraphrasing.* The last step in recognising an erroneous interpretation of the given sentence is to return a paraphrase to the user. For instance, the (unintended) paraphrase of the sentence "Sam likes an animal that swims and flies" is given in Figure 2.

The three parts – vocabulary, construction rules, and interpretation rules – are combined in the *Attempto Parsing Engine* (APE) [3]. It translates a given ACE text into *discourse representation structure* (DRS) [7], which uses a variant of the language of first-order logic. Since APE is implemented in Prolog, the DRS representation can be also modelled as a Prolog term, in which content words are represented by atoms and relationships by Prolog predicates with shared variables.

For a more detailed introduction into ACE, we refer to [4, 6]. A short overview about the language features supported by ACE is given in the online documentation at `http://attempto.ifi.uzh.ch/site/docs/ace_nutshell.html`. There is a public APE web service available at `http://attempto.ifi.uzh.ch/ape/`. In Figure 2, we present a screenshot of its web client for the example sentence "Sam likes an animal that swims and flies", with its corresponding paraphrase and DRS, showing the interpretation assumed for ACE.

3.2 The ACE Reasoner

Because DRS is a variant of first-order logic, it can be used to formulate theorems. The knowledge expressed as ACE sentences can be combined to infer new knowledge, to prove theses or to answer questions. In [1], Fuchs presents first-order reasoning for ACE texts. Its implementation is called *ACE Reasoner* (or *RACE* for short), and is based on Prolog. Like for ACE, there is a public web service available at `http://attempto.ifi.uzh.ch/race/`. Its web client supports three operations to (i) check the consistency of a given ACE text, (ii) to prove a theorem based on known axioms, and (iii) to answer a query based on known axioms.

In addition to the web client, RACE can also be accessed using a SOAP interface.[5] Because the source code of RACE is not freely available, this XML-based request-reply protocol is the only way to use RACE in connection with our Alexa skill.

4 Service Architecture

In Figure 3, we describe the service architecture for our integration of ACE into an Alexa skill using Prolog. In this example, the user wants to prove that Sam is a human, based only on previously expressed knowledge.
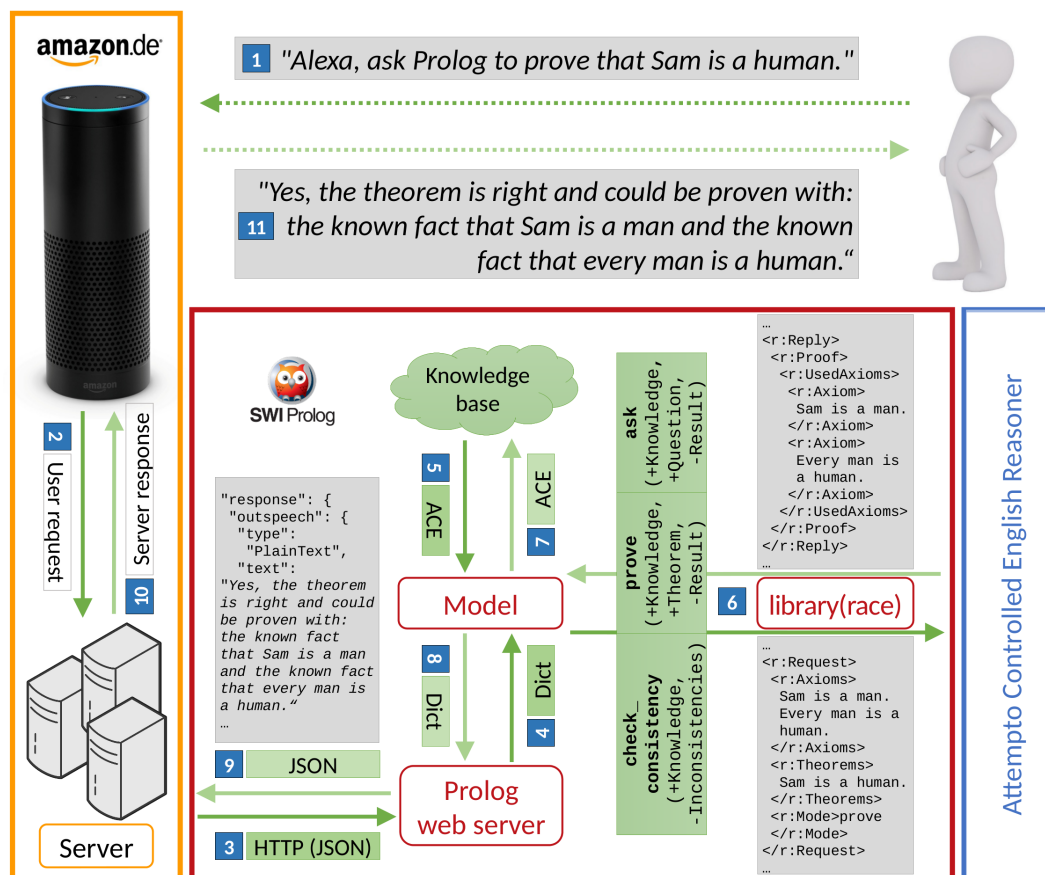

Figure 3 Service Architecture for the integration of RACE in an Alexa skill.

4.1   Speech-to-text by Alexa Voice Service

The smarthome device sends the spoken user request to Amazon's *Alexa Voice Service* (AVS), where it is translated into a textual representation. As part of this process, the requested skill is recognised, as well as the needed intent, and possibly the values for the slots of this particular intent.

The development of a new skill requires an Amazon Developer account. The skill creation consists of several steps, but first, the name of the application and supported languages have to be chosen. Amazon provides several pre-defined *models*, e.g., for a typical request to a media player. By defining a *custom skill*, the model can be adapted to fit our three intents "remember", "prove" and "question". In particular, they are created as example phrases in the so-called *interaction model*, each with a single slot. They are of the data type `AMAZON.SearchQuery`, which does not restrict the slot value, but consequently often comes with inaccuracies.

■ **Listing 1** Definition of the Prolog interaction model as JSON.

```
1  { "interactionModel": {
2      "languageModel": {
3        "invocationName": "prolog",
4        "intents": [
5          { "name": "remember", ... },
6          { "name": "question", ... },
7          { "name": "prove",
8            "slots": [
9              { "name": "proveslot",
10               "type": "AMAZON.SearchQuery" } ],
11               "samples": [ "prove {proveslot}" ], ... }] }}}
```

Amazon provides a JSON data format to exchange defined interaction models, an excerpt is presented in Listing 1. It defines the invocation, its intents, as well as their slots and sample sentences. The complete JSON file can be downloaded as part of the project's source codes at GitHub: `https://github.com/fnogatz/alexa.pl`.

4.2   Prolog Web Server

As part of the skill creation it is required to state the address of the web service that handles the text-based user request. For security reasons, Amazon requires a valid SSL certificate for this endpoint.[6] Amazon uses HTTPS over port 443 to communicate with this web service. For every user request, an HTTP request with JSON body is sent to the Prolog web server. It contains information about the requested intent; the slot values are passed as a string. For instance, for the example sentence of Figure 3, the value of the `proveslot` as defined in the interaction model of Listing 1 is simply passed as the string `"Sam is a human"`.

As part of our contribution, we provide a basic Prolog web server based on Swi-Prolog's *library(http)*. It makes use of the Swi-Prolog version 7 extensions [10], as the recently added `dict` Prolog data type closely resembles JSON. Listing 2 shows the basic definition of the Prolog web server. It is a generic framework to parse HTTP requests and create JSON replies according to the data format as provided and expected by AVS.

---

[6] The libraries to support HTTPS in Swi-Prolog are still under development. We therefore use *nginx* (`https://www.nginx.com/`) as a reverse proxy. As a result, the described Prolog web server runs under port 8080.

All HTTP requests are dispatched to the Prolog predicate `alexa/1`, which first creates a dict from the request's JSON body. With the help of `get_intent/2`, this nested JSON document is transformed into a small Prolog term `IntentData` that holds the information about the intent and slot values.

As a result, for a new Alexa skill one only needs to define the appropriate `process_alexa_request/2`, that gives an answer as text depending on the passed `IntentData`. The JSON reply is then again built using dicts, and finally sent to Amazon's servers for user output.

■ **Listing 2** Definition of the Prolog web server in alexa.pl. Only the implementation of `process_alexa_request/2` needs to be provided to support a new skill.

```
1  :- use_module(library(http/http_dispatch)).
2  :- use_module(library(http/thread_httpd)).
3
4  :- http_server(http_dispatch, [ port(8080)]).
5  :- http_handler(/, alexa, [methods([get,head,options,post]),prefix]).
6
7  alexa(Request) :-
8    http_read_json_dict(Request, DictIn),
9    get_intent(DictIn, IntentData),
10   process_alexa_request(IntentData, Answer),
11   answer_dict(Answer, DictOut),
12   reply_json(DictOut).
```

4.3 Connecting Alexa with RACE

For our example of using Alexa with RACE, we define `process_alexa_request/2` to forward the sentences to the RACE web service. This is done in the *model* in `alexa_mod.pl`. In particular, the current request (e.g., "prove that Sam is a human") has to be combined with the previously stated axioms. Once a new axiom is given by the user in a "remember" intent, its text-based paraphrase is stored in the service's *knowledge base*. For instance, given the previously stored axioms "Sam is a man" and "Every man is a human", it can be proven that Sam is a human.

4.4 library(race)

As the source code of RACE is not available, it can currently only be integrated into the Alexa skill via its web service (cf. Section 3.2). In order to access interfaces described in the *web service description language* (WSDL) standard, Swi-Prolog provides the add-on *library(wsdl)*.[7] On top of this, we built *library(race)*, a Prolog client for the SOAP interface of RACE. Its source code is available at GitHub: `https://github.com/fnogatz/race`.

library(race) defines the three Prolog predicates `check_consistency`, `prove`, and `ask`. In Listing 3, an example query is presented. RACE returns two proofs. In the first answer, "who" is substituted by "Sam". The second solution is deduced because there is a human if there is a at least one man, which could be proven by knowing that Sam is a man. In our service's model we select the best answer to give to the skill's user. The parts of the proof are combined into a single sentence, as shown in the example of Figure 3. The sentence has a generic form and integrates all `substitution/2` terms as returned by *library(race)*.

---

[7] Swi-Prolog package *wsdl* – `http://www.swi-prolog.org/pack/list?p=wsdl`

■ **Listing 3** Usage example for `ask/4` of *library(race)*.

```
1  %% ask(+Knowledge, +Question, -Result, +Options)
2  ?- ask("Every man is a human. Sam is a man.",
3         "Who is a human?", Result, []).
4  Result = results([
5    proof([ fact("Every man is a human."),
6            fact("Sam is a man."),
7            substitution("who", "Sam") ]),
8    proof([ fact("Every man is a human."),
9            fact("Sam is a man."),
10           substitution("who", "(at least 1) man") ]) ]).
```

Using the SOAP interface of RACE requires an additional network round-trip time for every interaction with the skill. In addition to just send the speech to AVS, our Prolog web server has to forward the text-based message to the RACE web service. As a result, the response time – which is critical in speech-based natural user interfaces – gets longer. This problem becomes even more serious because the underlying SOAP protocol is stateless, i.e., the whole knowledge base has to be sent to the RACE web service as part of each request, resulting in even bigger messages and longer round-trip times.

5   Conclusions and Future Work

Voice-controlled systems become more and more available – both for consumers and developers. With its *Developer Platform*, Amazon opens the new field of speech-based natural user interfaces to a broad community of developers which are not familiar with NLP. But although the specification of new commands using example phrases and slots is easy to use, it is also very limited. We herein present an integration of Amazon's Alexa with the controlled natural language ACE, which allows us to support more flexible user queries. As a proof of concept, our skill can be used to formulate and store knowledge in natural language, propose questions, and prove theorems. It provides a Prolog web server that can be adopted for multiple different Alexa skills as well, and *library(race)*, an add-on for Swɪ-Prolog to use the SOAP web interface of the ACE Reasoner.[8]

Field testing showed that the speech-to-text translation of AVS becomes fuzzy for longer input sentences. With simple slots, their values are very restricted, not least because Amazon provides several pre-defined slot data types. This is not the case for long inputs such as `AMAZON.SearchQuery`. As the Attempto Parsing Engine also translates text, a better recognition rate might have been achieved for AVS returning the best n sentences instead of just one.

We now plan to further experiment on platforms other than Alexa, resorting for instance to Google Assistant (a.k.a. Google Home) or Apple Siri, as far as these provide APIs which allow for high-level extensions such as that which we explored in the present work. One development we are looking forward to work on is to build a prototype based on the speech recognition software Mycroft [5] running on a PC with Linux, thereby achieving full control of the entire software stack. Some components of the current integration can be easily re-used, e.g., the Prolog web server, and *library(race)*.

---

[8] The source codes are available at `https://github.com/fnogatz/alexa.pl` and `https://github.com/fnogatz/race` (MIT License).

── **References** ──

1 Norbert E. Fuchs. First-Order Reasoning for Attempto Controlled English. In *International Workshop on Controlled Natural Language*, pages 73–94. Springer, 2010.

2 Norbert E. Fuchs, Kaarel Kaljurand, and Tobias Kuhn. Attempto Controlled English for Knowledge Representation. In *Reasoning Web*, pages 104–124. Springer, 2008.

3 Norbert E. Fuchs, Kaarel Kaljurand, and Gerold Schneider. Attempto Controlled English Meets the Challenges of Knowledge Representation, Reasoning, Interoperability and User Interfaces. In *FLAIRS Conference*, volume 12, pages 664–669, 2006.

4 Norbert E. Fuchs, Uta Schwertel, and Rolf Schwitter. Attempto Controlled English (ACE) Language Manual Version 3.0. `http://attempto.ifi.uzh.ch/site/pubs/papers/ace3manual.pdf`, 1999.

5 W. Wayt Gibbs. Build your own Amazon Echo – Turn a PI into a voice controlled gadget. *IEEE Spectrum*, 54(5):20–21, 2017.

6 Stefan Hoefler. The syntax of Attempto Controlled English: An abstract grammar for ACE 4.0. Technical Report ifi-2004.03, Department of Informatics, University of Zurich, 2004.

7 Hans Kamp and Uwe Reyle. *From discourse to logic: Introduction to modeltheoretic semantics of natural language, formal logic and discourse representation theory*, volume 42. Springer, 2013.

8 Irene Lopatovska, Katrina Rink, Ian Knight, Kieran Raines, Kevin Cosenza, Harriet Williams, Perachya Sorsche, David Hirsch, Qi Li, and Adrianna Martinez. Talk to me: Exploring user interactions with the Amazon Alexa. *Journal of Librarianship and Information Science*, page 0961000618759414, 2018.

9 Gustavo López, Luis Quesada, and Luis A Guerrero. Alexa vs. Siri vs. Cortana vs. Google Assistant: a comparison of speech-based natural user interfaces. In *International Conference on Applied Human Factors and Ergonomics*, pages 241–250. Springer, 2017.

10 Jan Wielemaker. SWI-Prolog version 7 extensions. In *Workshop on Implementation of Constraint and Logic Programming Systems and Logic-based Methods in Programming Environments*, pages 109–123, 2014.

Improving NLTK for Processing Portuguese

João Ferreira
Department of Informatics Engineering of the University of Coimbra, Portugal
jdcoelho@student.dei.uc.pt

Hugo Gonçalo Oliveira[1] ORCID
Centre for Informatics and Systems of the University of Coimbra, Portugal
Department of Informatics Engineering of the University of Coimbra, Portugal
hroliv@dei.uc.pt

Ricardo Rodrigues ORCID
Centre for Informatics and Systems of the University of Coimbra, Portugal
College of Education of the Polytechnic Institute of Coimbra, Portugal
rmanuel@dei.uc.pt

—— Abstract ——
Python has a growing community of users, especially in the AI and ML fields. Yet, Computational Processing of Portuguese in this programming language is limited, in both available tools and results. This paper describes NLPyPort, a NLP pipeline in Python, primarily based on NLTK, and focused on Portuguese. It is mostly assembled from pre-existent resources or their adaptations, but improves over the performance of existing alternatives in Python, namely in the tasks of tokenization, PoS tagging, lemmatization and NER.

1 Introduction

Nowadays, the amount of information that can be accessed is virtually infinite, and we may have every answer that is known by mankind at the palm of our hand. Yet, we often seem to be unable to find, understand or use it for other purposes. This happens because most information is presented with a fair amount of what can be considered noise – for instance, utterances or contextualization text that are not always relevant – and in text format, which forces us to read large blocks of text to get simple answers.

Natural Language Processing (NLP) addresses such issues. It comprises tasks that range from tokenization to Named Entity Recognition (NER), and applications such as Information Retrieval (IR), Information Extraction (IE) and Automatic Summarization.

Computational systems that deal with human language are complex, and have improved over time, but, in many respects, are still somewhat sub-optimal. Furthermore, results are generally worse when we consider languages other than English, such as Portuguese, because there are many more researchers working on and, thus, more tools developed for English. This worsens with language specificity, which may stop an otherwise global system for performing every NLP task in the same way for English and Portuguese.

---

[1] Corresponding author

In order to get better outcomes in NLP, we looked at the foundations of such systems – the set of NLP tools that process information. A typical NLP toolkit is formed by several modules that operate in a pipeline – that is, one module usually feeds the next and more information is added in each stage. Some of the baseline modules include tokenization, Part-of-Speech (PoS) Tagging, Lemmatization, Syntactic Parsing, and Named Entity Recognition (NER)

Several toolkits provide the previous tools. However, for Portuguese, there are only a few, and, to the best of our knowledge, none with a full pipeline (e.g., including lemmatization) in Python. Due to its growing use in the Artificial Intelligence and Data Processing fields, and its ease of use, many are using Python – as such, we chose it as the programming language of the pipeline to assemble. Therefore, the first steps were to understand which NLP pipelines already exist and how they could be improved.

Our main reference for the process was the NLPPort toolkit [14], also focused on improving the processing of Portuguese in OpenNLP [9], a toolkit in Java. Given that we were using Python, some experiments were performed, leading us to select NLTK [1] as the base pipeline, on top of which improvements were to be done or new modules plugged. Despite covering the Portuguese language, results with NLTK are limited and far from perfect. Following the approach of NLPPort, a post-processing step was added to NLTK's tokenizer to handle clitics and contractions. Further, in opposition to the default model for Portuguese, the NLTK PoS tagger was trained in manually annotated corpora. Based on the outputs of the tokenizer and the PoS tagger, a lemmatizer was developed, once again inspired by NLPPort. In NLTK, such a module was unavailable for Portuguese. Lastly, a Conditional Random Fields (CRF) NER tool was trained and added, towards easier training and better results in identifying named entities. These changes allowed for a better processing of Portuguese text. As we show, using NLPyPort, rather than the default NLTK pipeline, leads to more reliable results in the baseline tasks, which should have a positive impact on higher level tasks and applications. Although one option would be to make contributions directly to NLTK, for the time being, we chose to adapt it to our needs by wrapping NLTK and adding features: testing different models for already existing modules and changing or adding other modules.

The remaining of the paper is organized as follows. Section 2 briefly analyses and compares some of the existing toolkits. Section 3 describes how the pipeline was assembled, explaining what was changed from existent elements and what was created from scratch. Section 4 presents the results of testing the new pipeline and the comparison of these to previously existing results. Finally, Section 5 presents the conclusion extracted from our results and explain how the pipeline can be further improved in the future.

2   Related Work

There is already a handful of NLP tools and toolkits, some of which freely available for use, and supporting Portuguese. Without getting into much detail, we believe the following toolkits should be considered peers to the pipeline we are developing, and therefore with results that we should aim to obtain.

In Java, one of the most known, is the OpenNLP toolkit, and from which a more use-ready version for Portuguese was assembled – the NLPPORT toolkit. Another well-known Java-based toolkit is the Stanford CoreNLP [7], which does not support Portuguese out-of-the-box, but has enough resources for training models of any language.

In Python, there are two main alternatives: spaCy[2] whose utilization was considered (see Section 3); and the aforementioned NLTK [1], upon which the current pipeline is developed.

Among other toolkits that support Portuguese, in other programming languages, it is worth mentioning Freeling [11], developed in C++, and LinguaKit [5], developed in Perl.

3 The NLPyPort Pipeline

When choosing the base pipeline, several aspects were considered. First, due to the afore-mentioned reasons, our goal was to develop a pipeline in the Python programming language. This limited the choice of tools to NLTK and spaCy. In order to choose between them, some shallow testing was made to determine which tool was better-suited for the purpose, i.e., which was easier to use, change and adapt. We came to the conclusion that, even though spaCy is more friendly from a user perspective, as it allows to process text through the whole pipeline with a single function, NLTK allowed for a better manipulation of each stage of the pipeline, individually, making the changes more obvious.

For this reason, NLPyPort is based on the NLTK toolkit, with some layers and modules added. NLTK already provides a series of resources that can be used for Portuguese. It has a generic tokenizer, a PoS tagger trained on the "Floresta Sintá(c)tica" [3] corpus, and a general Named Entity Recognizer. With these, the base results were obtained and then changes were made, towards better results. Changes are described in detail in the following subsections. Briefly, a layer was added to the tokenizer, the PoS tagger was re-trained, and a lemmatizer was implemented based on LemPort. Lastly, a new NER, based on a CRF, was trained and integrated in the pipeline. The assembled pipeline will provide the user all the results specified, but can be changed to output only the desired data from a given task.

To illustrate the results of NLPyPort, Fig. 1 depicts the output of the initial pipeline, respectively for the simple NLTK and the NLPyPort pipeline, for the sentence: "*O António Costa deu um passeio no Porto.*"

```
Token        PoS            Token      Lemma      PoS      Entities
O,           art            O,         o,         art,     O
António,     prop           António,   antónio,   prop,    B-PESSOA
Costa,       nprop          Costa,     costa,     nprop,   I-PESSOA
deu,         v-fin          deu,       dar,       v-fin,   O
um,          art            um,        um,        art,     O
passeio,     n              passeio,   passeio,   n,       O
no,          adp            em,        em,        prp,     O
Porto,       nprop          o,         o,         art,     O
.            punc           Porto,     porto,     n,       B-LOCAL
                            .          .,         punc,    O
```

Figure 1 Output of simple NLTK (left) and of the NLPyPort pipeline (right).

3.1 Improving the Tokenizer

One of the critical elements of any pipeline is the tokenizer. It is responsible for splitting text into the smallest units that represent information – tokens. Since this is, for most cases, the starting point of NLP, its errors will propagate throughout the whole pipeline and thus impact performance. For this reason, it is essential to get the best possible results out of it.

We first need to understand where tokenization may fail for the specific language, or where it could be better handled, towards more useful outputs. Upon exploring NLTK, we concluded that many of the errors obtained were due to language specific elements not handled properly; this was expected because NLTK's tokenizer is operating in a language-independent manner. The main problem found was not splitting tokens of two distinct types, clitics and contractions, which, depending on the application, results in loss of information.

A **clitic** is a word form generated by joining and concatenating, by means of an hyphen, a verb and a personal pronoun into a single word – for instance, "*[eu] comprei-o*" ↦ "*[eu] comprei ele*" (I have bought it). A correct separation will allow for the PoS tagger to better identify the verb and the personal pronoun as such. A **contraction** is generated when a preposition and a pronoun are joined and concatenated into a single word – for instance, "*na*" ↦ "*em a*" (in the). Similarly to the clitics, proper separation into two words should improve PoS tagging.

The implementation of these modules was rather simple, mainly because the most work-intensive part had already been done – the creation of lists of clitics and contractions. These were part of NLPPort, and available in a standard XML format, which allowed their easy integration in Python. In order to apply these, the default NLTK's tokenizer is used and both lists are checked to find word matches with any of the known contractions or clitics. Such cases are changed to the expanded from, replacing the original word by simpler words.

3.2   Improving the PoS tagger

As mentioned earlier, improving tokenization would improve PoS tagging. This happens because most PoS tagging approaches are based on models for automatic sequence labelling. So, when both clitics and contractions are not handled, both new tokens (e.g., *lo*, *no*) and PoS tags for these (e.g., *ADP* instead of *PREP+DET*) are introduced, with a negative impact on the generalization power of the tagger. Not to mention that some of those tokens may increase ambiguity (e.g., "*nos*" can either be the clitic for "*a nós*" or the contraction of the preposition "*em*" with the determinant "*os*").

NLTK provides a few different models for PoS tagging, one of which is based on Maximum Entropy. In theory, it can learn from any PoS tagged text and it is language independent. Yet, the quality of the training text used is a determinant factor in the results. Its base implementation uses not only the current word, but also some of the previous and of the following words. In fact, it is not limited to words and uses other features, such as PoS tags, if available. This means that, even if there was no change in the current word, the fact that the previous word had been split into two other words with two other tags will influence the tag of the current word.

The default NLTK PoS tagging model for Portuguese is learned from the "Floresta Sintá(c)tica" corpus [3]. Even though it contains a large number of annotated documents, not all were manually reviewed. So, in order to improve the PoS tagging model, we decided to use only "Bosque", a smaller, but manually annotated subset of "Floresta Sintá(c)tica". The reasoning behind this is that a manually annotated resource is less likely to have errors, and thus a system that learns from it should do its job better. In addition to "Bosque", the PoS tagger was also trained with the "Mac-Morpho" [2] corpus, this too manually annotated, yielding a combined number of 35 PoS tags.

3.3 Integrating a Lemmatizer

In order to normalize text, NLTK includes the RSLP stemmer for Portuguese [10]. This tool is based on the removal of suffixes from Portuguese words and, while it may be worth for some text mining tasks, the result is often not an existing word. An alternative would be a lemmatizer that transforms words in their dictionary form, thus easing their linking with lexicons or other resources. However, NLTK does not include a Portuguese lemmatizer.

Due to its wide range of applications, we chose to implement a Portuguese lemmatizer and added it to the pipeline. The code and design for this module was largely inspired in LemPort, a part of NLPPort. This allowed for using the same resources as LemPort, and apply them exactly in the same situations.

The lemmatizer is defined by two modules: the first relies on a lexicon; the second is a set of predefined hierarchical rules that will perform transformations on the token, in case there is no match with lexicon words.

For the first module, the process is straightforward: each word with a valid tag is searched on LABEL morphology lexicon [13]. If the lexicon contains the word, its lemma is retrieved and the lemmatization process ends. If, on the other hand, the word is not in lexicon, rules corresponding to the tag are applied and repeated until there is a match against the lexicon. This process must be repeated after each rule is applied, as there are words that must go through multiple normalizations – for instance, gender and number. If no rules match the word and it is not found in the lexicon, it is assumed to be already in its lemmatized form.

It is also important to note that each word may have a ranking, used when there is more than one lemmatization possibility. This ranking was determined by the word's frequency in the frequency lists of the AC/DC project [15].

The second module is based on a hierarchical set of rules, that are applied if no match was found in the lexicon. In order to make the correct lemmatization, a set of handcrafted rules perform it in a progressive order. This way, some normalizations are always made before others, which allows for a reduction of the rules on the following normalizations. To better understand this, lets take the example of a noun of feminine gender and plural form. If we applied gender normalization before number normalization, the result would not match an existing lemma. By executing the number normalization first, we only need the masculine form of the noun (if applicable). Considering this simplification, the following order of rules was defined: (1) Manner (adverb) normalization; (2) Number normalization; (3) Superlative normalization; (4) Augmentative normalization; (5) Diminutive normalization; (6) Gender normalization; (7) Verb normalization (for regular and irregular verbs).

To illustrate this, lets consider the word "*certinhas*". It will first be processed by the number normalizer ("*certinhas*" ↦ "*certinha*"), then the diminutive normalizer ("*certinha*" ↦ "*certa*"), and lastly the gender normalizer ("*certa*" ↦ "*certo*"). The final lemmatized form would be the much simple word "*certo*".

3.4 Adding a new Named Entity Recognizer

The most recent addition to the pipeline was a Named Entity Recognizer (NER) module. The default NLTK's NER tool offers many challenges when training on a new model or a different set of categories. There is no way to train it out-of-the-box. Even if the user can understand how to do it, they need to previously know for which entity categories it is being trained, and cannot get them from the training data. For this reason, we considered an alternative and ready-to-use NER module, with easier training, while maintaining or even improving the baseline NLTK's results. The NER tags utilize were those of the Second

HAREM collection[4]. The integrated NER module is based on CRFsuite, an implementation of Conditional Random Fields (CRF) [6], which, in a not so distant past, have been used for NER with relative success. Although originally available by Naoaki Okazaki [8], we used a wrapper for the scikit-learn toolkit[3], easier to train and test than NLTK NER module.

Entities were identified with the BIO notation. This is a way to indicate if a token is the beginning (B), inside (I), or outside (O) of an entity. This letter is then followed by the type of entity. An example of this can be seen in Figure 1.

4   Measuring Improvements

We recall that our goal was to achieve better results on the processing of the Portuguese language, using Python. So, in order to quantify the improvement over NLTK, we compared our results to the base results of using NLTK out-of-the-box.

4.1   Improvements in Tokenization

To measure how much the tokenizer has improved, we used it to tokenize the documents in the manually-reviewed Bosque corpus. Achieved results were compared to those obtained with the original NLTK's tokenizer. For both tokenizers, Table 1 reports on the proportion of correct tokens over all the tokens in the corpus.

Table 1 Percentage of correct tokens.

Toolkits	Tokenization Accuracy(%)
NLTK	83
NLPyPort	90

Numbers confirm that the changes made in the tokenization module lead to an improvement of 7% over NLTK's baseline. As mentioned earlier, this is due to the separation of clitics and contraction. An example of this can be seen in Fig. 1, where the word "*no*" is split into two simpler words: "*no*" $\mapsto$ "*em o*" .

4.2   Improvements in PoS tagging

To assess the second change in the pipeline, the PoS tagger was fed with the Bosque tokenization, one per line. The previous tokens were then tagged and the result compared to the tags in Bosque. Per token accuracy for both the original PoS tagging model of NLTK and the Bosque trained NLPyPort PoS tagger are reported in Table 2.

Table 2 Percentage of correct tags.

Tookits	PoS tagging(%)
NLTK	60
NLPyPort	86

Figures show significant improvements on PoS tagging after the changes made, namely of 26% over the base pipeline. We recall that this is due to training in different corpora.

---

[3] `https://github.com/TeamHG-Memex/sklearn-crfsuite`

An example of PoS tagging can also be seen in Fig. 1. The word "no", previously classified as *adp*, is now split and its parts differently classified ($no_{\#adp} \mapsto em_{\#prp} + o_{\#art}$).

4.3 Improvements in Tokenization followed by PoS tagging

Improvements over the original NLTK's pipeline were confirmed for the tokenizer and the PoS tagger, individually. Since each module feeds the next one, we can assume that these improvements are also reflected on the pipeline. To test whether this is true, we compared the results of tokenization followed by PoS tagging, using the original NLTK and using NLPyPort. Table 3 shows the per-token accuracy of PoS tagging, after the improvements in the tokenizer and in the PoS Tagger. Again, Bosque was used as our gold reference.

Table 3 Percentage of correct tags after Tokenization.

Tool	PoS tagging after Tokenization(%)
NLTK	58
NLPyPort	82

Results show that there is a substantial improvement, of 24 points, when the new tokenizer and PoS tagger are used together, one after the other, thus confirming the improvements achieved with NLPyPort. Differences of these tasks against the original NLTK are also seen in Figure 1.

With NLPyPort, complex tokens are now split into simpler tokens, and more of the tokens are correctly classified. For example, we can see that there has been a division of the word "*no*", as defined earlier, into the to simpler forms "*em*" and "*o*", which then leads to a better classification of the tags. NLTK also fails to identify the punctuation in the sentence.

4.4 Improvements in NER

A study of Portuguese NER with NLTK has already been made [12], and the results obtained were used as a base for our comparison. To ensure that the tests are fair, the same train and test splits used to assess the default NLTK NER module were used for CRF NER, as well as the testing approach – ten-fold cross-validation with 4 repeats on a BIO conversion[4] of the Second HAREM [4] golden collection, where named entities were manually annotated.

The metrics used for the comparison were Precision, Recall and F1-score, which were compared to those presented by Pires [12]. The obtained results are reported in Table 4.

Table 4 Comparison of Precision, Recall and F-Measure of NLTK NER and CRF NER.

NER	Precision(%)	Recall(%)	F-Measure(%)
Baseline	30.58	31.38	30.97
CRF	57.05±2.45	46.56±1.29	51.28±1.49

Results are far from perfect and some points below the 0.59 best F1 in Second HAREM [4]. Yet, this comparison confirmed that the CRF NER outperforms the default NLTK's NER for Portuguese, and has therefore been a worthwhile addition to the assembled pipeline.

---

[4] For the NER datasets used, check `https://github.com/arop/ner-re-pt/`.

5   Conclusions and Future Work

We have presented NLPyPort, a new NLP pipeline implemented in Python and based on NLTK, specifically focused on Portuguese processing, for which improvements were achieved. The current version of NLPyPort is freely available from: `https://github.com/jdportugal/NLPyPort`.

In the near future, we will keep with the continuous improvement of the different modules of the pipeline and, similarly to NLPPORT, we will add a feedback loop for sending the identified entities to the PoS tagging module, hopefully leading to better results. Another possible and needed addition to this pipeline is a more user friendly interface that allows the user to change the pipeline as desired. Given their potential utility for more advanced applications, we will also consider the addition of other modules, namely a phrase chunker and a relation extractor.

References

1   Steven Bird and Edward Loper. NLTK: The Natural Language Toolkit. In *Proceedings of ACL 2004 on Interactive poster and demonstration sessions*, page 31. ACL, 2004.

2   Erick Rocha Fonseca and João Luís G Rosa. Mac-Morpho Revisited: Towards Robust Part-of-Speech Tagging. In *Proceedings of 7th Brazilian Symposium in Information and Human Language Technology*, 2013.

3   Cláudia Freitas, Paulo Rocha, and Eckhard Bick. Um mundo novo na Floresta Sintá(c)tica — o treebank do Português. *Calidoscópio*, 6(3):142–148, 2008.

4   Cláudia Freitas, Paula Carvalho, Hugo Gonçalo Oliveira, Cristina Mota, and Diana Santos. Second HAREM: advancing the state of the art of named entity recognition in Portuguese. In *Proceedings of 7th International Conference on Language Resources and Evaluation*, LREC 2010, La Valleta, Malta, May 2010. ELRA.

5   Pablo Gamallo and Marcos Garcia. LinguaKit: uma ferramenta multilingue para a análise linguística e a extração de informação. *Linguamática*, 9(1):19–28, 2017.

6   John D. Lafferty, Andrew McCallum, and Fernando C. N. Pereira. Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data. In *Proc. 18th International Conference on Machine Learning*, ICML '01, pages 282–289. Morgan Kaufmann, 2001.

7   Christopher D. Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven J. Bethard, and David McClosky. The Stanford CoreNLP Natural Language Processing Toolkit. In *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 55–60. ACL Press, 2014.

8   Naoaki Okazaki. CRFsuite: a Fast Implementation of Conditional Random Fields (CRFs), 2007. URL: `http://www.chokkan.org/software/crfsuite/`.

9   Apache OpenNLP. Apache software foundation. *URL http://opennlp. apache. org*, 2011.

10   Viviane Moreira Orengo and Christian Huyck. A Stemming Algorithm for the Portuguese Language. In *Proceedings of 8th International Symposium on String Processing and Information Retrieval (SPIRE)*, pages 183—-193, Laguna de San Raphael, Chile, 2001.

11   Lluís Padró and Evgeny Stanilovsky. FreeLing 3.0: Towards wider multilinguality. In *Proceedings of 8th International Conference on Language Resources and Evaluation (LREC-2012)*, pages 2473–2479, Istanbul, Turkey, May 2012. ELRA.

12   André Ricardo Oliveira Pires. Named Entity Extraction from Portuguese Web Text. Master's thesis, Faculdade de Engenharia da Universidade do Porto, 2017.

13   Elisabete Ranchhod, Cristina Mota, and Jorge Baptista. A Computational Lexicon of Portuguese for Automatic Text Parsing. In *Proceedings of SIGLEX99 Workshop: Standardizing Lexical Resources*. ACL Press, 1999.

14   Ricardo Rodrigues, Hugo Gonçalo Oliveira, and Paulo Gomes. NLPPort: A Pipeline for
         Portuguese NLP (Short Paper). In *7th Symposium on Languages, Applications and Technologies
         (SLATE 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.
15   Diana Santos and Eckhard Bick. Providing Internet access to Portuguese corpora: the AC/DC
         project. In *Proceedings 2nd International Conference on Language Resources and Evaluation*,
         LREC 2000, pages 205–210, 2000.

Quarmic: A Data-Driven Web Development Framework

Pedro Miguel Pereira Cunha

CRACS & INESC Tec LA / Faculty of Sciences, University of Porto, Portugal
up201405950@fc.up.pt

José Paulo Leal

CRACS & INESC Tec LA / Faculty of Sciences, University of Porto, Portugal
http://www.dcc.fc.up.pt/~zp
zp@dcc.fc.up.pt

Abstract

Quarmic is a web framework for rapid prototyping of web applications. Its main goal is to facilitate the development of web applications by providing a high level of abstraction that hides Web communication complexities. This framework allows developers to build scalable applications capable of handling data communication in different models, data persistence and authentication, requiring them just to use simple annotations. Quarmic's approach consists of the replication of the shared object among clients and server in order to communicate through its methods execution. Where the annotations, namely decorators, are used to indicate the concern (model or view) that each method addresses and to implement the framework's inversion of control. By indicating the method concern, it enables the separation of its execution across the clients (responsible for the view) and the server (responsible for the model) which facilitates the state management and code maintenance.

1 Introduction

The emergence of Web 2.0 and the popularization of mobile apps with internet connectivity broaden the scope of web applications thus increasing the demand of interactive applications that support a high number of users and deliver a real-time experience. As a result, web development has become more difficult, because the range of problems that developers have to deal have also increased, such as concurrency and scalability, or security and session management issues.

This paper presents Quarmic, a web development framework for rapid prototyping of web applications built on top of Node.js. Specifically, it consists of a high-level tool that facilitates deployment of web apps with a sophisticated communication system, capable of handling data communication in different models (broadcast, multicast and unicast), data persistence and authentication in a transparent and scalable way. The main goal of Quarmic is to simplify the development of web applications by raising the level of abstraction. A high level of abstraction enables developers, with little expertise on web communication, to build complex web apps, since it completely frees the developers from implementing any component regarding communication. Consequently, it also allows them to focus on other parts of the application, such as the user interface and the logic itself. Web applications built with

Quarmic follow a class-based object-oriented paradigm in which each class and its methods can be annotated using the annotation syntax provided by the framework. This annotation syntax is the mechanism used by the developer to assign a particular role to each method (a model or view concern) since the communication is fully based on methods execution. Also, the objects of the annotated classes are shared among the server and clients, allowing to keep the server's object as the single-source-of-truth. Which implies the separation of the method execution by its concern between the server and the clients. Hence, once the objects are instantiated, Quarmic inverts control thus implementing and controlling the execution of its methods (including persistence and authentication).

As a result, developers can create web apps as shared objects. For instance, Quarmic can be used to prototype a real-time multiroom chat. Where the chat itself is a shared object that contains a list of rooms, which are also shared objects. Every single client will share the chat object, but only the room participants will share the object that represents its room. Another example can be a collaborative spreadsheet app, where each cell is a shared object, allowing to maintain the state of each cell or a group of cells separately.

This novel approach is an advantage relative to the existing frameworks since it only requires a few annotations to implement in contrast with the coding required by any other approach. Therefore, because of its level of abstraction, a developer acquainted with the oriented object paradigm will be able to quickly gain web development proficiency. In addition, Quarmic does not restrict the use of other framework or libraries to program other tiers of the application, which is advantageous to the flexibility of programming.

The remainder of this paper is organized as follows. Section 2 reviews the architecture of Node.js and some features of a few web application frameworks that influenced the creation of this framework. Section 3 describes Quarmic's architecture as well as its main features, presenting some implementation details. Section 4 concludes the purpose of this paper and summarizes the future work.

2   State-of-the-Art

Node.js & Javascript

In the past years, Node.js has becoming increasingly popular among developers [4], as a result of its architecture and its modules that allow to quickly and effectively build highly scalable Web applications. Node's architecture introduces event-driven programming to the server-side scripting, which is a much more efficient approach in terms of memory when dealing with concurrency. Despite not being the first platform to do this, it is by far the most successful [5]. Its asynchronous event-driven JavaScript (single-threaded) runtime provides a preferable application environment to develop highly scalable systems [2].

Another contributing factor to the Node's adoption as the preferred platform to application development is the single language feature. Node enables client-side and server-side scripting in Javascript, elevating the language, which formerly was just used as a client-side language, to a new height of server-side web development [1]. Therefore, it simplifies the development environment, since developers can build the entire tier of their applications in the same language. An approach that Google Web Toolkit framework introduced, where developers write all components in Java. Then, the browser components would be compiled to Javascript and HTML.

Web Application Frameworks

Web application frameworks are software frameworks that make it easier to build web applications. They provide tools and libraries that simplify common web development tasks, such as HTML generation, state and session management and database interaction.

Frameworks such as Ruby on Rails[1] or Laravel[2], provide an Object-Relational Mapping (ORM) layer that simplifies the use of relational databases in Web applications by mapping each database table with a class in the underlying language thus relieving developers of the hassles of dealing with the underlying database [8]. Another common feature is its architecture. Both follow the Model-View-Controller (MVC) pattern that divides the applications into three interconnected parts. The Model, that manages the business logic; the View, that defines the presentation of the Model to the user interface; and the Controller, that serves as an intermediary between the View and Model, responding to the user input and interactions [7].

Other frameworks, such as Vue.js[3], offer state management features. Vues' reactivity system makes state management simple and intuitive. When a JavaScript object is passed to a Vue instance, all properties are converted to getters / setters using `Object.defineProperty`. These getters/setters are hidden by an abstraction layer that makes them invisible to users, but internally they allow you to run dependency-tracking and change-notification whenever the properties in question are accessed or changed. Each instance of a Vue component has a corresponding watcher instance. During component rendering, this watcher instance registers, the properties converted to getter as dependencies. As such, if a setter of a dependency is executed, it will notify the watcher, which will cause the re-rendering of its component [9].

3 Quarmic

Quarmic is a Node.js framework for *rapid prototyping* of web applications. Its main feature is the support for data communication between multiple clients. As a web framework, its goal is to facilitate the web application development process. This framework is aimed at web applications that deal with near real-time behaviour, in particular, applications that share objects among multiple clients. It accomplishes this goal by taking control over the execution flow of the application, which provides the ability to sequence and coordinate the application activity in order to receive and process data in a consistent way, and return results quickly enough to ensure a near real-time behaviour. Thus, it frees the programmer from the implementation of these tasks, allowing them to focus on the application custom logic and appearance.

The main differentiation feature of Quarmic is its architecture 1. It follows the object-oriented paradigm to design the application, allowing the developers to create *shared objects*, objects that are shared among different clients. Unlike the majority of the web application framework, this framework doesn't rely on the Model-View-Controller (MVC) pattern to assign different web app components to each of those roles. Instead, it enables the association of those roles to methods of the corresponding classes by annotating them according to the concern they address, which can be either a business logic operation (model) or a presentation logic operation (view).

The approach to share an object relies on the replication of the server's instance. Once the server instantiates a shared object, this object becomes shareable among all clients. This means that clients that instantiate it will share it with the server and the other clients that have instantiated it as well. And since the client's instance is created as a replica of the

---

[1] `https://rubyonrails.org/`
[2] `https://laravel.com/`
[3] `https://vuejs.org/`
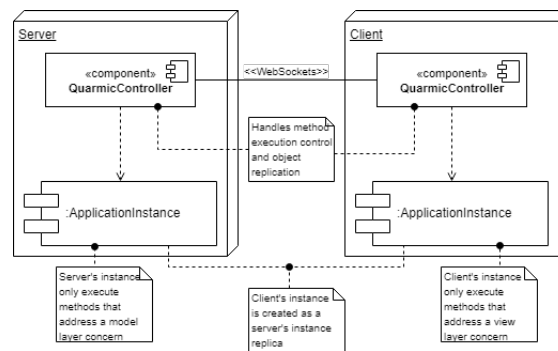
█ **Figure 1** Quarmic's architecture.

server's object, the framework does not allow clients to instantiate a shared object without it had been previously instantiated in server. By performing a distributed execution, the framework is able to control the method execution of these objects, ensuring that methods that have been annotated as addressing a model concern are only executed by the server's instance and methods that have been annotated as addressing a view concern are only executed by the clients' instance (server's instance replica). Moreover, the framework updates the clients' replica before executing any method.

Therefore, by annotating methods with the concerns they address, classes declarations can be shared among server and clients, facilitating data management. Since the server is the only component allowed to change the object state, an object that is shared among multiple clients maintains a consistent state. The fact that this framework is built on top of Node.js provides a JavaScript runtime environment that allows the application to be coded in a single language, JavaScript. This approach is also compatible both with popular client-side cross-browser JavaScript libraries and toolkits, such as Bootstrap or JQuery and with Node.js modules.

In the following subsections, the framework is described in more detail. Firstly, the annotation syntax is presented. Secondly, the inversion of control of the framework is addressed by presenting its core processes and some implementation details.

3.1   Annotation Syntax

The following code snippet presents a class with Quarmic annotations. In this example, the class `Counter` encodes a counter with a button to increment it. The counter's initial state is set by the class annotation and its two methods are annotated according to their role. The `increase` method increments the counter value (object state modification) and calls the `update` method, thus addressing a business logic concern. Conjointly, the `update` method addresses a view concern, since it updates the counter value in the user interface. Incidentally, in this example, the controller role is provided by the constructor since it binds user actions to the model, but this is not relevant to Quarmic.

```
@sharedProperties({value: 0})
class Counter{
constructor(){
this.counterElem = document.getElementById("counter-value");
this.incElem = document.getElementById("increase-button");
this.incElem.onclick = this.increase.bind(this);
this.update.call();
}
```

```
@cause
increase(){
this.value++;
this.update();
}

@effect
update(){
this.counterElem.value = this.value;
}
}
```

Quarmic's annotations use the ES7 decorator proposal, which enables the modification of JavaScript classes and properties at compile time without explicitly modifying them [3]. Decorators allow programmers to apply the desired behaviour to the application built with Quarmic. As mentioned before, they inform the framework of the concern addressed by the methods. But more importantly, they implement Quarmic's inversion of control, which gives the ability to take control over the application lifecycle in order to perform the object replication among clients and server and to control both method execution.

The code is influenced by annotations according to the side where it is executed. Thus, a decorated class on the server-side will behave differently from the same decorated class on the client-side. The main Quarmic annotations are the following.

@sharedProperties(properties)

This annotation is applied to a class declaration to specify the properties (fields) of shared objects and their initial values. Essentially, it modifies the constructor in order to be used as a dependency injection container. On the server-side, the constructor assigns the shared object state properties passed by the 'properties' variable (list of properties) of the argument and sets up the database facility. On the client-side, the constructor performs the object replication (process described in the following section) before executing the original code.

@cause(auth)

Indicates to the framework that the method annotated with this decorator addresses a model concern, delegating its execution if the method is invoked on the client. Also, if the "auth" variable is passed to the decorator as "required", the cause will require authorization in order to be executed. Otherwise, if it is passed as "none" or it is omitted, the cause method will no longer require authorization to be executed. This argument is only used when the class is protected by authentication (process described in the following section).

@effect(scope)

Indicates to the framework that the method annotated with this decorator addresses a view concern, delegating its execution if the method is invoked on the server. Also, if the "scope" variable is passed to the decorator as "private", the effect will only be executed in the client that has invoked the cause method of that effect. Otherwise, if it is passed as "public" or it is omitted, the effect method will be executed by all clients.

3.2   Distributed Execution

Quarmic's architecture is based on a distributed execution. This process relies on WebSockets to keep a continuous exchange of data between the clients and the server in order to perform the object replication and to control the execution of methods. It is implemented using the Socket.io library, which enables near real-time, bidirectional and event-based communication between server and clients.

The following subsections address how the methods execution control and the object replication are performed.

3.2.1   Object Replication

Object replication consists in pushing the state of the server's object to their client's replicas. This process occurs when the object is instantiated and establishes a socket to the server, which is used to request the up-to-date state. As a result, the client's object will inherit the server's object properties, transforming the client's object into a replica of the server's object. The following figure 2 illustrates this process.
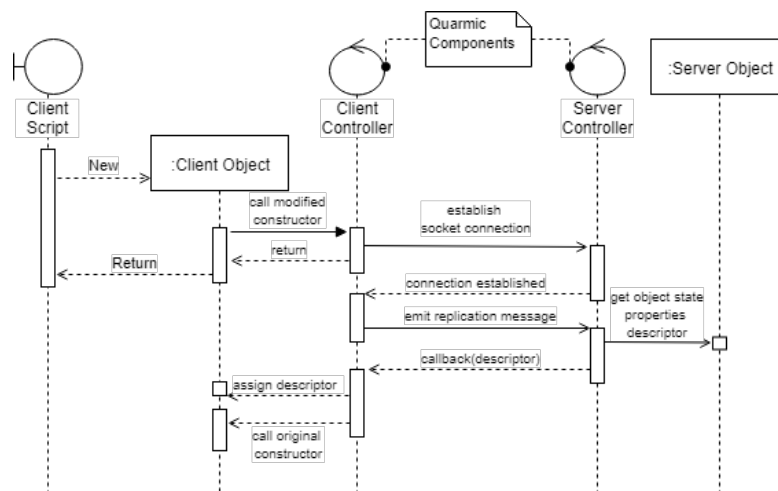
■ **Figure 2** Sequence Diagram - Object Replication.

3.2.2   Method Execution Control

As previously referred, the method execution control is intended to ensure that the model layer methods are only executed in the server and the view layer methods are only executed in the client. It is inspired in the *causality principle* that states that for every *effect* there is a definite *cause*. Hence, a view method execution (effect) is always preceded by a model method execution (cause). Therefore, the framework will not allow clients to directly invoke an effect method. Both server and clients can directly invoke cause methods. For instance, we can have an application with a web service running in the server that calls cause methods, and components in the user interface bound to cause methods, but only the server can execute them. Moreover, it's important to note that this relationship can be chained. In particular, a cause method can be called within another cause method or within an effect method.

This approach allows to maintain a consistent state among multiple clients. As the server executes a cause method, invoked by itself or by one of the clients, it propagates the up-to-date state and the effects methods execution instructions to each one of the clients.

Eventually, each one of them will update its object's state and execute the effects, updating the object visual representation in an accurate and consistent way. . The following 3 figure illustrates this process.
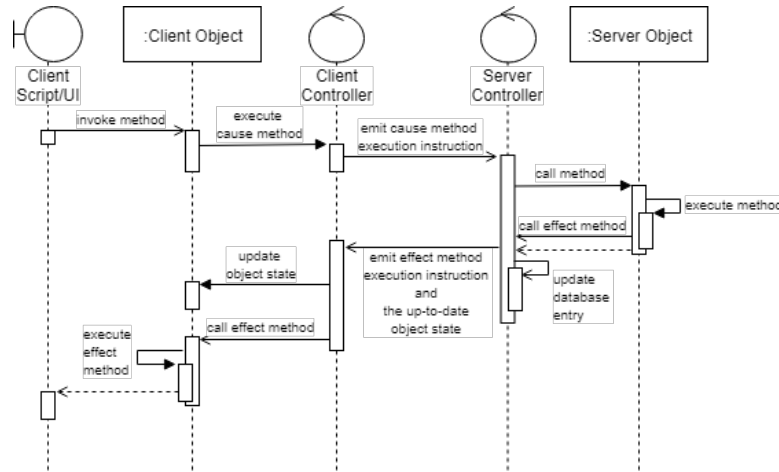
Figure 3 Sequence Diagram - Method Execution Control.

Since JavaScript has a concurrency model based on an event loop, it ensures that whenever a method is executed, it cannot be pre-empted[6]. This enables the server to handle multiple concurrent calls of cause methods by multiple clients because the server is able to coordinate the execution of those methods even if they were called almost simultaneously. And the same goes to the propagation of the object state and the resulting effects.

3.2.2.1 Data persistence

Quarmic handles data persistence by itself. When the server instantiates a shared object, the framework creates an entry in the database to store the object state or to get the object data if already exists. This entry is updated whenever the object state is modified, ensuring an up-to-date backup of the shared object. Since this process is done asynchronous, it won't overload the server. Moreover, to improve performance, up to a certain number of objects are kept in memory, using a least recently used (LRU) cache.

3.2.2.2 Authentication

Quarmic also provides an authentication feature. It consists of a token-based authentication system and is also controlled through the invocation and execution of methods. These methods are a special type of cause (annotated with the `@authenthicator` decorator) and they are responsible for assigning a token that initiates the client's authorized session in the shared object domain.

When a class has a method of this kind (authenticator method), it will be treated as a protected class, which means that all others methods will require authorization to be invoked. The authorization is provided by the authenticator method and since it addresses a model concern, it is executed in the server. If its execution determines that was successfully authenticated, the framework assigns a token to the clients' object, establishing an authorized session. This is useful in methods that implement login, for instance.

4   Conclusions and future work

This paper presents Quarmic, a web application framework that aims to facilitate the development of applications, in particular, near real-time web applications. The framework includes a high-level mechanism that supports real-time applications as a single shared object (e.g. a webchat) or a set of shared objects (e.g. a collaborative spreadsheet app).

Quarmic is a work in progress, currently in the final development stage. At the time of writing, it lacks some improvements in the deployment of the applications and validation regarding system testing and user acceptance testing. The main challenge it has been the implementation of the facility to deployment because at this moment the decorators' proposal is at stage 2 (Draft), which is a hindrance to work with them in the browser. The current workaround is the transpilation of the decorators to ES5. The remaining steps go through to test its performance in the real world by performing some stress tests and user acceptance tests.

References

1 Dave Anderson. How Node.js can accelarate development, 2014. Modulus.

2 Nimesh Chhetri. A Comparative Analysis of Node.js (Server-Side JavaScript), 2016. Culminating Projects in Computer Science and Information Technology. Paper 5.

3 JavaScript Decorators. [Online; accessed April 2019]. URL: `https://github.com/tc39/proposal-decorators`.

4 Node.js Foundation. Node.js User Survey, 2018. [Online; accessed April 2019]. URL: `https://nodejs.org/en/user-survey-report`.

5 Tom Hughes-Croucher and Mike Wilson. *Node: Up and Running*. O'Reilly Media, Inc., 2012.

6 Neelakantan R. Krishnaswami Jennifer Paykin and Steve Zdancewic. The Essence of Event-Driven Programming, 2016. Unpublished Draft. URL: `https://www.cl.cam.ac.uk/~nk480/essence-of-events.pdf`.

7 Abdul Majeed and Ibtisam Rauf. MVC Architecture: A Detailed Insight to the Modern Web Applications Development, 2018.

8 David B. Copeland Sam Ruby and Dave Thomas. *Agile Web Development with Rails 5.1*. Pragmatic Bookshelf, 2017.

9 Reactivity in Depth. [Online; accessed April 2019]. URL: `https://vuejs.org/v2/guide/reactivity.html`.

Identifying Causal Relations in Legal Documents with Dependency Syntactic Analysis

Pablo Gamallo
Centro de Investigación en Tecnoloxías Intelixentes (CiTIUS)
University of Santiago de Compostela, Galiza
pablo.gamallo@usc.es

Patricia Martín-Rodilla
Centro de Investigación en Tecnoloxías Intelixentes (CiTIUS)
University of Santiago de Compostela, Galiza
patrica.martin.rodilla@usc.es

Beatriz Calderón
University of Santiago de Compostela, Galiza
beatriz.calderon.cerrato@rai.usc.es

Abstract

This article describes a method for enriching a dependency-based parser with causal connectors. Our specific objective is to identify causal relationships between elementary discourse units in Spanish legal texts. For this purpose, the approach we follow is to search for specific discourse connectives which are taken as causal dependencies relating an effect event (head) with a verbal or nominal cause (dependent). As a result, we turn a specific syntactic parser into a discourse parser aimed at recognizing causal structures.

1 Introduction

Compared to work on syntactic analysis, approaches focused on higher linguistic levels, such as discourse analysis, are scarcer for all languages, including English [10]. In the case of Spanish language, there is still very little work on discourse analysis, which is mainly focused on RST-like models [1]. Among the different approaches on discourse analysis, special attention deserves automatic identification of causal relationships. Causation plays a central role in scientific and social domains, including the legal domain where cause-effect relationships stand for the scaffolding necessary to provide an argumentative diagnosis. By identifying cause-effect relationships in legal documents, it is also possible to identify the agents that play a role in legal acts [7]. Besides, in the legal domain, we are confronted with specific text types and, therefore, with the need to adequate the formalism for representing discourse patterns typical of this domain. [11].

Current Machine Learning (ML) based approaches have shown good results at lexical and syntactic levels, and there is some attempt to work on extraction of discursive components with ML algorithms, with good results [8]. However, existing systems require a large volume of annotated corpus for the training phase, which is time-consuming. Besides, pre-training resources do not focus on the legal domain and on its particular structure and semantics,

so we would have to annotate *ad hoc*. In addition, existing works focus on textual sources in English and, as far as we know, the analysis of this type of discursive structures for Spanish from the ML perspective is not widespread. Because of this, a rule-based work on discursive structure identification conforms, in addition to a breakthrough in the state of the art of automatic discourse treatment in Spanish, a structural basis for further computational treatment from ML perspectives.

The objective of this article is to provide a dependency-based syntactic analyzer with specific rules to identify causal relations between linguistic units in legal domain documents. We convert a specific syntactic analyzer into a discourse parser by adding specific discursive rules adapted to the legal domain. This will be done by identifying causal markers appearing in legal texts, as well as by defining the corresponding grammar rules for each marker and the linguistic/discursive units they put in relation. Our work, therefore, resembles those few approaches that propose to use dependency structure to directly represent the relations between elementary discourse units [9, 13]. Unlike other approaches focused on causality verbs to identify elementary discourse units [12], our main focus are syntactic connectors codified as causal conjunctions and/or locutions.

The enriched syntactic parser was applied to Spanish legal texts and was evaluated using a test document manually annotated. Performance of the system reached 0.65 F-score with high precision and rather low recall. This short article is organized as follows. Section 2 describes the proposed method, which is evaluated in Section 3. Conclusions and future work are addressed in Section 4.

2   The Method

Our method consists of two main tasks: first, we study a set of legal documents so as to identify the main causal connectors and their syntactic behavior, namely what type of units they relate. And second, we improve an existing rule-based syntactic parser by adding specific grammar rules to identify causal relationships between sentences and/or nominal phrases.

2.1   Causal Markers

In the current work, we focus on cause-effect relationships established through causal markers, where the cause can be either a verb clause or a nominal phrase, and the effect is a verb clause. Implicit causal relations are not considered since they are not introduced by any connector. We only take into account causal relations identified by means of explicit markers such as in the work by [14], where the authors introduced a set of *concessive* discursive markers (e.g., *though*, *but*, etc) aimed at selecting opposition relationships between discursive units for both English and Spanish languages.

In our framework, the cause is syntactically codified as the subordinate or dependent unit, while the effect is the head. The connector represents an element specifying the dependency relation. Causal connectors (including those having a consecutive meaning) are conjunctions or locutions classified into two main subcategories:

Verbal Cause: connectors putting in relation a verbal effect with a verbal cause, for instance, "porque", "pues", "puesto que", "por lo que", etc (all of them might be translated into English as *because*),

Nominal Cause: prepositional locutions connecting a verbal effect with a nominal cause, for instance, "en virtud de" (*by virtue of*) or "en razón de" (*because of*).

■ **Table 1** List of Spanish causal conjunctions and locutions manually identified in legal texts.

Verbal Cause	porque, pues, puesto que, dado que, visto que, considerando que, teniendo en cuenta que, ya que, en consecuencia, por tanto, por consiguiente, por eso, por lo cual, por lo tanto, por lo que
Nominal Cause	a causa de, a fuerza de, a propuesta de, a petición de, a efecto de, a los efectos de, con motivo de, en razón de, en virtud de, en vista de, por causa de, por culpa de, por razón de, de acuerdo con, gracias a

It is worth noting that *verbal cause* connectors allow us to identify inter-sentence relations while *nominal cause* locutions identify intra-sentence relations. In both cases, the related elements represent elementary discourse units. By using Spanish legal documents, we have identified and defined 30 causal and consecutive locutions/conjunctions, 15 of them being verbal cause connectors, and 15 nominal cause ones. We make use of a broad sense of causal and consecutive relations, including *finality* values in some cases.

2.2 Dependency Rules

To implement causal rules, we use the grammar formalism, DepPattern [5], which is suited to define syntactic depedencies between any syntactic unit. These formal grammars are compiled into finite-state transducers working as dependency parsers [4, 3]. To carry out our objective, we used as starting point the freely available Spanish grammar of the DepPattern project.[1] Lemmatization and PoS tagging has been performed with the multilingual toolkit, Linguakit [2].

A DepPattern grammar is constituted by a set of dependent rules. Every rule aims at identifying a specific dependent-head relation by means of a pattern of part-of-speech (PoS) tags. Any dependency rule is constituted by two elements:

- a pattern of PoS tags,
- the name of a dependent-head relation found within the pattern.

Let us see an example:

DobjR: VERB NOUN

The two elements of a rule are separated by a semicolon. The first element is "DobjR", which stands for the name of a specific dependency relation, namely "a direct object (Dobj) appearing to the right (R) of the head". The second element is a sequence of PoS tags: VERB and NOUN, respectively represent the "head" and the "dependent" units. This rule is applied after having identifying the dependent units of the noun within the nominal phrase and the dependent units of the verb within the verbal phrase.

Each rule is implemented as a transducer that recognizes head-dependent token relations and removes the *dependent* tokens from the input sequence. It is applied from left to right until it reaches the end of the input sentence. The successive application of these rules simplifies and reduces the search space of the next rule to be applied [4]. For instance, to process the sequence "to have a nice day", the parser applies first the rules that identify the dependency between the adjective "nice" (dependent) and "day" (head), and that between the determiner "the" (dependent) and "day" (head), before applying the direct object rule linking the verbal head "have" to the dependent noun, "day". At each rule application, the dependent unit is removed from the search space as each dependent word only must have one head.

---

[1] `https://github.com/citiususc/DepPattern`

For the current work, the Spanish grammar was provided with a set of rules aimed at identifying causal relations. All causal connectors we have identified in the previous task were declared and classified in the enhanced version of the Spanish grammar so as to be used in specific rules. Table 2 shows two specific cause/effect rules. The first one is the dependency relation "CausalR" which detects the relationship between two verbs linked by a causal connector (CONJ<loc:verbal_cause>), and where the dependent/subordinated verb (the verbal cause) appears to the right of the head verb (the effect). This rule is able to identify the causal relation between the two verbs appearing in the language sequence of the second column. Symbol "[Fc]?" represents an optional comma. CONJ<loc:verbal_cause> means that the rule is using connectors belonging to the *verbal cause* subcategory. It is worth noting that this rule is only applied after having identified all the dependent elements (complements, auxiliars, and modifiers) of the two verbs heading the related sentences. So, once this rule is applied, we are able to build the entire syntactic tree starting from the root verb, which is the head of the cause/effect dependency.

The second rule in Table 2 is used to identify causal relations starting with the connector and the dependent noun (nominal cause), followed by a verbal head (effect). The notation CONJ<loc:nominal_cause> means that the rule is using connectors belonging to the *nominal cause* subcategory.

■ **Table 2** The first column shows DepPattern rules to identify causal dependencies using conjunctions or locutions as markers. In the second column, we show a corpus-based sequence the rule was applied to. The identified units are in bold: cause, effect and connector.

Cause/effect rules	Language sequences
CausalR: VERB [Fc]? CONJ<loc:verbal_cause> VERB	"Galicia, compendio de universalidad, **quiere participar** con plena dignidad y protagonismo en el concierto de las culturas, **por lo que** en este texto se **asumen** mandatos, criterios y principios recogidos en las diversas cartas" (*Galicia, a compendium of universality, **wants to participate** with full dignity and protagonism in the concert of cultures, **so** this text **assumes** mandates, criteria and principles contained in the various letters*)
CausalL: CONJ<loc:nominal_cause> NOUN [Fc]? VERB	"**A los efectos de** esta **ley**, **integran** el patrimonio artístico de Galicia las manifestaciones pictóricas, escultóricas, cinematográficas, fotográficas, musicales y de las restantes artes plásticas, de especial relevancia, de interés para Galicia" (***For the purposes of*** *this **law**, the artistic heritage of Galicia **includes** pictorial, sculptural, cinematographic, photographic, musical and other plastic arts of special relevance, of interest to Galicia.*

■ **Table 3** Performance of the system in the process of recognizing causal relations.

total	tp	fp	fn	precision	recall	F-score
91	45	2	46	0,96	0,50	0,65

The output of the system can be either in CoNLL-X format [6] or in dependency triples (similar to Standford dependencies).[2] The grammar updated with causal rules is freely available from the DepPattern project.[3]

3 Evaluation

In order to evaluate our system, a legal document, namely the law 5/2016 on Galician cultural inheritance, containing 13k words, was manually annotated. More precisely, a linguist identified all causal connectors and their related discursive units. The document was syntactically analyzed and the performance of the results were measured. Precision is the number of correct decisions (true positives) divided by all decisions taken by the system (true positives + false positives). Recall is the number of correct decisions (true positives) divided by the total number of causal relationships found in the document (true positives + false negatives). Finally, F-score is the harmonic average of the precision and recall.

Table 3 shows the results of the evaluation. 91 causal relations were manually found in the test document. In order to compute precision, recall, and F-score, we count the number of true positives (tp), false positives (fp) and false negatives (fn). The system achieves 0.95 precision, while only 0,50 recall. F-score thus reaches 0,65.

According to the results depicted in Table 3, the system has a very high degree of accuracy, but coverage fails. Among the main problems of coverage, we must highlight the cases of very ambiguous connectors that were not introduced into the grammar. For instance, the ambiguous conjunction "como" (*as*) is missing as a causal marker in the grammar, even if it is used with this meaning in the test document. Preposition "por" (*by*) is also used as a causal connector but it has not adding to the grammar as such due to its very ambiguous behavior.

4 Conclusions

This paper describes a rule-based method to introduce causal dependencies into a dependency parsing by updating DepPattern grammars. The performance of the system reaches 0.65 F-score with high precision but still low recall due to the fact that ambiguous connectors are missing in the grammar.

In future work, the grammar will be provided with more specific rules and enriched with new connectors. Moreover, the Spanish grammar will be adapted to other languages, namely Portuguese, Galician and English, which are the languages that DepPattern supports in the current version of the formalism. In addition, we will create RST-like structures from the dependency output, by converting the dependencies between causal units into full constituents. For this purpose, we will establish functional equivalence between *head* and *nucleus* as well as *dependent* and *satellite*. Finally, we will also include a visualization module so as to allow users to look up linguistic patterning and discourse structure in a more friendly way.

---

[2] `https://nlp.stanford.edu/software/stanford-dependencies.shtml`
[3] `https://github.com/citiususc/DepPattern/tree/master/grammars/grammar-devel-es`

References

1 Iria da Cunha. A Symbolic Corpus-based Approach to Detect and Solve the Ambiguity of Discourse Markers. *Research in Computing Science*, 70:95–106, 2013.

2 P. Gamallo, M. Garcia, C. Piñeiro, R. Martinez-Castaño, and J. C. Pichel. LinguaKit: A Big Data-Based Multilingual Tool for Linguistic Analysis and Information Extraction. In *2018 Fifth International Conference on Social Networks Analysis, Management and Security (SNAMS)*, pages 239–244, 2018. `doi:10.1109/SNAMS.2018.8554689`.

3 Pablo Gamallo. Dependency Parsing with Compression Rules. In *Proceedings of the 14th International Workshop on Parsing Technology (IWPT 2015)*, pages 107–117, Bilbao, Spain, 2015. Association for Computational Linguistics.

4 Pablo Gamallo and Marcos Garcia. Dependency parsing with finite state transducers and compression rules. *Information Processing & Management*, Available online 5 June 2018, 2018.

5 Pablo Gamallo and Isaac González. A Grammatical Formalism Based on Patterns of Part-of-Speech Tags. *International Journal of Corpus Linguistics*, 16(1):45–71, 2011.

6 Johan Hall and Jens Nilsson. CoNLL-X shared task on multilingual dependency parsing. In *The tenth CoNLL*, 2006.

7 Rinke Hoekstra and Joost Breuker. Commonsense Causal Explanation in a Legal Domain. *Artificial Intelligence Law*, 15(3):281–299, 2007. `doi:10.1007/s10506-007-9033-5`.

8 Shafiq Joty, Giuseppe Carenini, and Raymond T. Ng. CODRA: A Novel Discriminative Framework for Rhetorical Analysis. *Computational Linguistics*, 41(3):385–435, 2015. `doi:10.1162/COLI_a_00226`.

9 Sujian Li, Liang Wang, Ziqiang Cao, and Wenjie Li. Text-level Discourse Dependency Parsing. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 25–35, Baltimore, Maryland, June 2014. Association for Computational Linguistics. `doi:10.3115/v1/P14-1003`.

10 Amália Mendes and Iria del Río. Using a Discourse Bank and a Lexicon for the Automatic Identification of Discourse Connectives. In Aline Villavicencio, Viviane Moreira, Alberto Abad, Helena Caseli, Pablo Gamallo, Carlos Ramisch, Hugo Gonçalo Oliveira, and Gustavo Henrique Paetzold, editors, *Computational Processing of the Portuguese Language*, pages 211–221, Cham, 2018. Springer International Publishing.

11 Marie-Francine Moens, Caroline Uyttendaele, and Jos Dumortier. Intelligent Information Extraction from Legal Texts. *Information & Communications Technology Law*, 9(1):17–26, 2000. `doi:10.1080/136008300111583`.

12 Chaveevan Pechsiri and Asanee Kawtrakul. Mining Causality for Explanation Knowledge from Text. *J. Comput. Sci. Technol.*, 22(6):877–889, 2007. `doi:10.1007/s11390-007-9093-8`.

13 Kenji Sagae. Analysis of Discourse Structure with Syntactic Dependencies and Data-Driven Shift-Reduce Parsing. In *Proceedings of the 11th International Conference on Parsing Technologies (IWPT'09)*, pages 81–84, Paris, France, October 2009. Association for Computational Linguistics. URL: `https://www.aclweb.org/anthology/W09-3813`.

14 Maite Taboada and María de los Ángeles Gómez-González. Discourse markers and coherence relations: Comparison across markers, languages and modalities, 2012.

Quantitative Analysis of Suffix Variability of Comparative Adjectives in Russian

Timur I. Galeev
Kazan Federal University, Kazan, Russia
Justus-Liebig-Universität Giessen, Germany
https://kpfu.ru/timur.galeev
tigaleev@kpfu.ru

Vladimir V. Bochkarev 🄳
Kazan Federal University, Kazan, Russia
https://kpfu.ru/vladimir.bochkarev
vbochkarev@mail.ru

—— Abstract ——

There are two variants of the productive suffix of comparative adjectives used in modern Russian. They are a full two-syllable form and a reduced one-syllable suffix. Both variants are normative. However, they are slightly different in terms of stylistics. The suffix -ee makes the word sound neutral and the word with the suffix -ei sounds more colloquial. The article presents a quantitative study of variability of the suffixes of comparative adjectives and analyzes linguistic and extralinguistic factors that influence the frequency of the variants. The authors concluded that there is no previously anticipated influence of phonetic and morphological factors on the choice of the suffix of an adjective in a bookish speech.

1 Introduction and Literature Review

The presence of extra-large diachronic corpora led to significant changes in linguistics. It became possible to conduct a microanalysis of linguistic phenomena, such as morphosyntactic variability. Such studies have changed significantly in both theoretical and methodological terms over the past ten years. Many theoretical works on syntactic variability were inscribed in the paradigm of generative linguistics and reduced themselves to identifying semantic differences between variants and determining the primacy of one of the variants. In terms of methodology, a lot of works reduced to mono-factorial studies based on relatively simple text fragments (resolving collocations).

Nowadays variability studies have become much more functional and methodologically more complex: language facts are now interpreted and motivated by psycholinguistic factors, and the study material is analysed in terms of the cognitive or usage-based approach [3, 6, 7, 10]. Regression analysis of linguistic phenomena has gained popularity and been used in many recent works [12, 13, 14].

The common feature of the above-mentioned works is that they use statistical analysis of corpus or experimental (survey) data and try to identify new indicators that influence the choice of a competing form. Our study, in contrast to the studies of foreign colleagues, is

carried out on the material of the standardized literary Russian language.

Using regression analysis that considers many factors and can be widely used for comparative studies, European and American scientists illustrate that, on the one hand, grammatical variability is found within a particular regional and cultural tradition (indigenization, or "privatization"). But on the other hand, different regional dialects can show the same cases of variability as in the standard language (literary language) [14]. Within the same ethnic group that has mastered a particular language and, according to A.A. Shakhmatov [16], "treated the language as his own property", such variability arises after some time despite the initial absence of negative language experience.

In this case, some factors (such as different cultures of speakers of different languages) prove the universal nature of grammatical variability and its potential as a marker of certain cognitive processes typical of human brain. Within the framework of a psycholinguistic-based grammar model, we also interpret these results from different explanatory points of view, including the phenomena of language contact, assimilation of the second language (albeit to a lesser extent), semantic variations and changes.

2   Methods

We searched for examples of variability in the Google Books Ngram, unprecedently extra-large corpus [11]. For example, the pattern of morphological variability is "N-ee \N-ei", where N is a combination of letters same for the first and second variants (umn-ee\umn-ei (smarter)). Henceforth, we used the ALA-LC transliteration system.

These patterns are tested after the discussion. Data are collected from the Google Books Ngram corpus. To be more objective, the search is performed (1) for the whole Russian corpus (the period 1607-2009) and (2) for the period 1920-2009 (after the spelling reform). The "purity" of the obtained results indicates the quality of the pattern. If the obtained data are incorrect the pattern is refined. Part-of-speech tagging of the obtained data array was checked using the OpenCorpora dictionary [2]. Not only Russian comparative adjectives but also some forms of nouns and pronouns can end in -ei. Some of them can be homonymous to the studied part of speech. For example, the forms umnei and starei can both correspond to verbs (singular imperative form) and comparative adjectives.

Such isomorphism of dynamic verb forms and forms of comparative adjectives in slavic languages [1], as well as statistical probability of homonymy in inflectional languages in general [15] were studied by J.D Bobaljik and his coauthors. Homonyms belonging to different parts of speech were excluded. For example, verbs in the imperative form like umnei (be clever!) (V) were excluded from the list, which allowed us to analyse only the occurrences of the adjectives umnei (smarter) (Adj), which are variants of the word umnei (smarter) (Adj).

At the second stage of the research, we manually excluded the homonymous forms. The examples that were not variants were excluded (for example, vechernee// vechernei (zarei) (evening/evening (dawn))). The selected list of words contains quantitative data: the number of each variant occurrence in the corpus (for example, bolnee - 49060, bolnei - 22691 (more sick)) and the ratio of the variants (bolnee - 68%, bolnei - 32%). The ratio of the frequencies was summed, and the arithmetic average of the whole group was calculated. For example, the percentage of adjectives with the suffixes -ee// -ei is 71.54% // 28.46% within the entire period, and 71.41% // 28.59% over the last 100 years.

Graphs of changes in the frequency use of both variants were constructed for each of the pairs and for the sum of the variants of the whole group in order to demonstrate the language dynamics and compare the trends and total quantitative data. The obtained data

on the number and dynamics were interpreted and described to assess the degree of relevance and expediency of certain standard prescriptions.

To improve the accuracy of the results, we performed preprocessing of the corpus raw data. After that, we selected only the vocabulary 1-grams from the obtained ones. By 1-grams we understand words composed only of the letters of the Russian alphabet. Word forms that differ only by the presence or absence of the letter "er" at the end of them were regarded as the same words. To normalize and calculate the relative frequencies, the number of vocabulary 1-grams was calculated for each year (unlike Google Books Ngram Viewer, where normalization is performed for the total number of 1-grams). Another difference is that we calculate frequencies without taking into account differnces in capitalisation.

3 Results and Discussion

Graphs of changes in the frequency use of both variants were constructed for each of the pairs and for the sum of the variants of the whole group in order to demonstrate the language dynamics and compare the trends and total quantitative data.

In total, 1571 pairs of comparative adjective were analyzed. The suffix -ee prevails in 1395 cases, the suffix -ei prevails in 174 cases. The same number of word usage with the suffixes -ee/-ei was found in 2 cases. The average ratio of the variants for all pairs of comparative adjectives is as follows: 71.54% (-ee) // 28.46% (-ei).

The obtained quantitative and diachronic data were interpreted and described to assess the degree of relevance and expediency of certain standard prescriptions taking into account linguistic factors (phonetics, morphology and style).
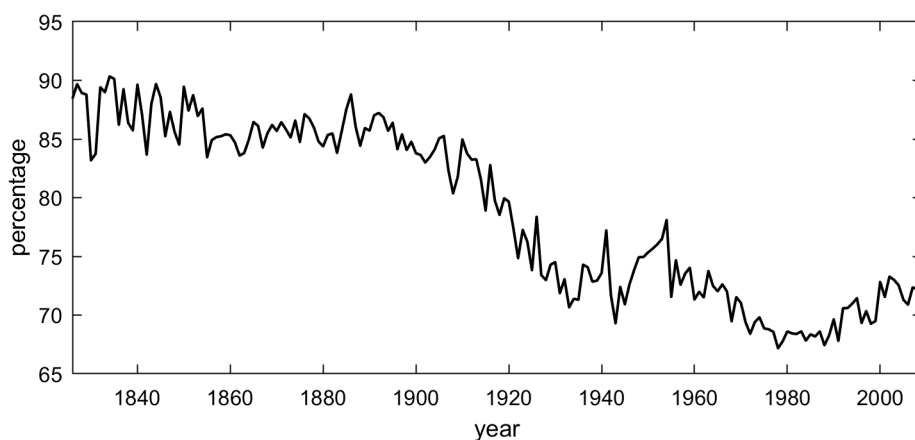
**Figure 1** The mean value of the word share (for the total number of words with -ee/-ei).

The following tendency was detected. Low-frequency words are characterized by approximately the same number of occurrences of the variants (for example, nepriglyadnee (60) // nepriglyadnei (ugly) (41) (59.41% vs. 40.59%); dramatichnee (51) // dramatichnei (dramatic) (56) (47.66% vs 52.34%). High-frequency words show the opposite tendency, the number of their use significantly differs (sern'esnee (1900960) // ser'eznei (more serious) (197850) (90.57% vs 9.43%). High-frequency words with the suffix -ee prevail over the words with the suffix -ei (vazhnee, trogatelnee, effektnee (more important, more touching, more effective)). The only exception is the word masshtabnei. Low-frequency words with the suffix -ei slightly prevail over the words with the suffix -ee (dramatichnei, conservativnei).

Analysis of changes in the frequency dynamics of the whole group of comparative adjectives showed the following overall tendency: the number of words with the suffix -ei gradually increased within the period from the end of the 20th century to the beginning of the 20th century and rapidly increased in the 20th century. Figure 1 shows the ratio of the variants in different years. The variants ratio with the suffixes -ee/-ei was approximately 90%/10% in the middle of the 20th century. The share of the adjectives had gradually decreased over the century and a half and reached a minimum by 1980 (below 70%).
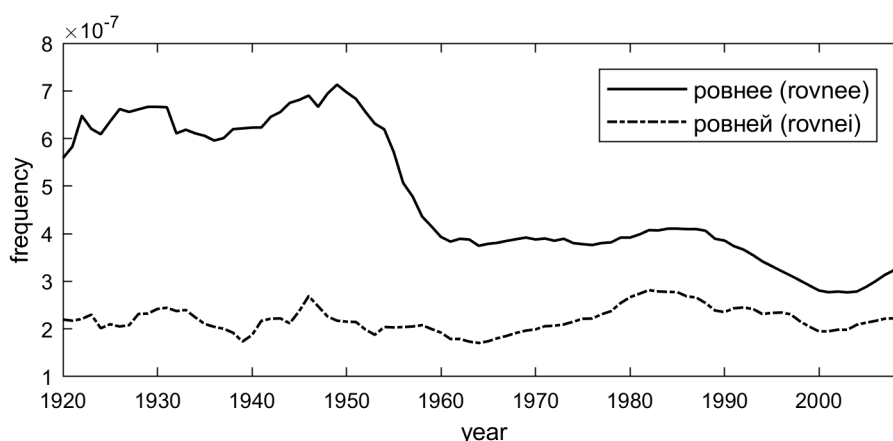
Figure 2 Frequency graph of the adjectives rovnee/rovnei (more even).

This tendency can be traced while analyzing the pair of words rovnee/rovnei (more even). The frequency of these variants will probably be equal in some time. An example of this case is shown in figure 2. Like in the Google Books Ngram Viewer service, a moving average with a window +- 3 years (the window length is 7 years) is used.
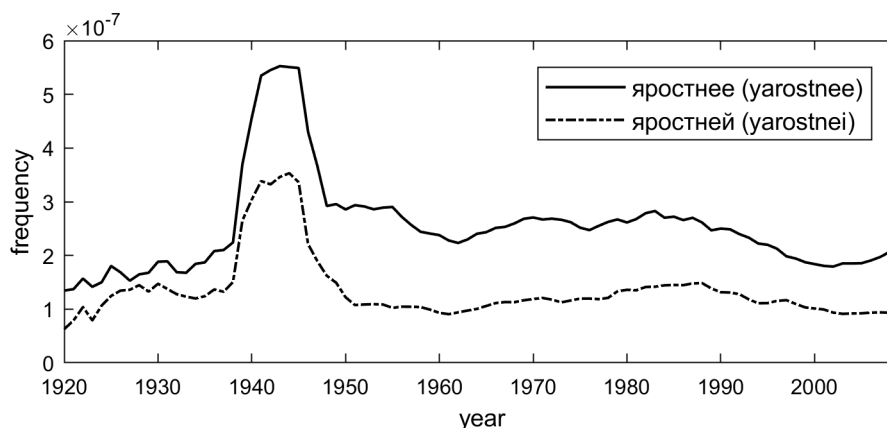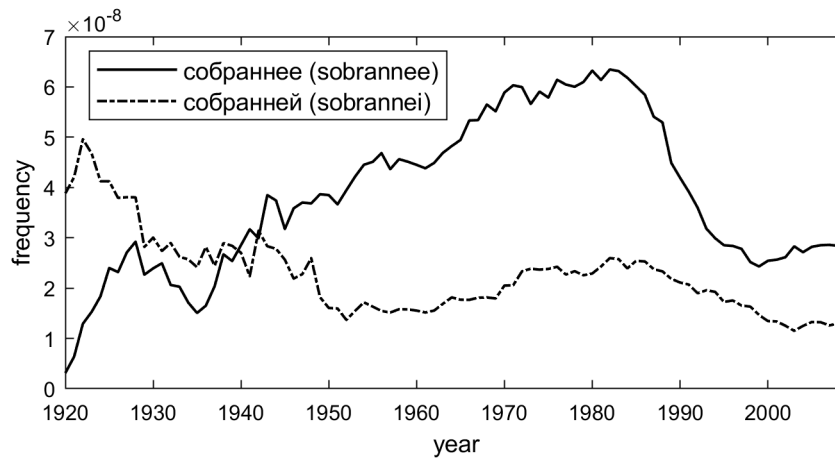

Figure 3 Frequency graph of the adjectives yarostnee/yarostnei (more violent/fierce).

Some pairs show frequency peaks due to historical events. For example, during the Great Patriotic War, there was a sharp increase in adjectives yarostnee/yarostnei (more violently/ fiercely) in Russian press and fiction (see figure 3). It was caused by appeal to defend the motherland.

The predominance of the variant with the suffix -ei was detected in some rare cases at

the beginning of the 20th century. As a rule, in such cases, the initially less frequent variant with the suffix -ee has been used more frequently in recent time (see figure 4).

Figure 4 Frequency graph of the adjectives sobrannee/sobrannei (self-collected).

4 Conclusion

Russian morphology has been an extensively studied area of linguistics. Lots of synchronic studies on Russian words variability were performed in the pre-corpus era when the study material was analyzed manually.

It was previously believed that the choice of a variant form depends not only on the style of the book and an author's idiolect but also on the complexity of the word root and place of stress in the word and prefix. Some researches state [4, 5] that if a root is monosyllabic, the suffix -ee is preferable and if a root is multi-syllabic, the suffix -ei is preferable.

The main feature of our work is that we performed diachronic analysis of the given type of variability for the first time. Google Books Ngram was used as a study material and allowed for obtaining more objective data than that obtained manually from a relatively small number of sources.

The results were the following. Only ten pairs of adjectives (out of 102 pairs) with a monosyllabic root show predominance of the variant with the suffix -ei (10% versus 12% for the whole group). At that, in 10 cases, where the suffix -ei prevails over the competing suffix -ee, the maximum ratio is 75%, while the share of -ee is more than 90% in many cases. Thus, the cases of the predominance of the suffix -ei in the monosyllabic words are less than in the group on average. Moreover, there is no meaningful predominance in these cases.

In other works [8, 9], the variants with the prefix po-, which has the meaning of "a small increase or change in the quality of an adjective" are often mentioned: pobistrei, poslozhnei and posil'nei (faster, more complicated, stronger). In a bookish speech, this additional meaning is conveyed by the word nemnogo (bit): nemnogo bistree, nemnogo slozhnee, nemnogo sil'nee (a bit faster, a bit more complicated, a bit stronger).

The adjective suffix -ee was more frequent in 15 cases out of 190 (less than 8%). Comparing this result with the average value for the whole group (12%), we conclude that the prefix has no influence on the choice of the suffix -ei as a preferable one.

We detected a decrease in the proportion of forms with the suffix -ee in the book speech in the 80s years of the 20th century in comparison with the language situation in the middle

of the 19th century by more than 20%. After the 80s of the 20th century, the proportion of the -ee forms fixed at the level of 75%. This indicates changes in morphology of the bookish speech which can be due to language liberalization and economy.

Thus, the Google Books Ngram corpus, being a source of "hidden" knowledge of bookish speech, allows one to identify language anomalies, as well as qualitatively and quantitatively improves the existing scientific description and interpretation of language features by verifying the previously obtained empirical data.

References

1   Jonathan D. Bobaljik. *Universals in Comparative Morphology: Suppletion, superlatives, and the structure of words*. MIT Press, Cambridge, USA, 2012.

2   V. Bocharov, S. Bichineva, D. Granovsky, N. Ostapuk, and M. Stepanova. Quality assurance tools in the OpenCorpora project. In *Komp'juternaja lingvistika i intellektual'nye tehnologii: Po materialam ezhegodnoj Mezhdunarodnoj konferencii «Dialog»*, pages 101–109. Russian State University for the Humanities, Moskow, 2011.

3   Claire Childs, Christopher Harvey, Karen P. Corrigan, and Sali A. Tagliamonte. Transatlantic perspectives on variation in negative expressions. *English Language and Linguistics*, pages 1–25, 2018. `doi:10.1017/S1360674318000199`.

4   I.A. Es'kova. Obrazovanie sinteticheskikh form stepenei sravneniia v sovremennom russkom literaturnom iazyke. In *Razvitie grammatiki i leksiki sovremennogo russkogo iazyka*. Nauka, Moskow, 1964.

5   L. K. Graudina, V. A. Itskovich, and L. P. Katlinskaia. *Grammaticheskaia pravil'nost' russkoi rechi. Stilisticheskii slovar' variantov*. Nauka, Moskow, 2004.

6   Stefan Th. Gries. Syntactic alternation research: Taking stock and some suggestions for the future. *Belgian Journal of Linguistics*, 31(1):8–29, 2017. `doi:10.1075/bjl.00001.gri`.

7   Benedikt Heller, Benedikt Szmrecsanyi, and Jason Grafmiller. Stability and Fluidity in Syntactic Variation World-Wide: The Genitive Alternation Across Varieties of English. *Journal of English Linguistics*, 45(1):3–27, 2017. `doi:10.1177/0075424216685405`.

8   O.K. Kochineva. Stepeni kachestva bezlichno-predikativnykh slov v sovremennom russkom literaturnom iazyke. *Uch. zap. LGPI im. Gertsena*, 402:59–67, 1968.

9   V.Iu. Koprov. *Variantnye formy v russkom iazyke*. Voronezh state university, Voronezh, 2001.

10 Aet Lees. 4 Synchronic Corpus Study of Object Case Alternation. In *Case Alternations in Five Finnic Languages*, pages 52–92. Brill, Leiden, The Netherlands, 2015. `doi:10.1163/9789004296367_005`.

11 Jean-Baptiste Michel, Yuan Kui Shen, Aviva Presser Aiden, Adrian Veres, Matthew K Gray, Joseph P Pickett, Dale Hoiberg, Dan Clancy, Peter Norvig, Jon Orwant, et al. Quantitative analysis of culture using millions of digitized books. *Science*, 331(6014):176–182, 2011.

12 Terttu Nevalainen, Elizabeth Closs Traugott, and Phillip Wallage. Quantitative evidence for a feature-based account of grammaticalization in English: Jespersen's Cycle, November 2012. `doi:10.1093/oxfordhb/9780199922765.013.0060`.

13 Dirk Pijpops and Freek Van de Velde. Constructional contamination: How does it work and how do we measure it? *Folia Linguistica*, 50(2):543–581, 2016. `doi:10.1515/flin-2016-0020`.

14 Melanie Röthlisberger, Jason Grafmiller, and Benedikt Szmrecsanyi. Cognitive indigenization effects in the English dative alternation. *Cognitive Linguistics*, 28(4):673—-710, 2017. `doi:10.1515/cog-2016-0051`.

15 Uli Sauerland and Jonathan Bobaljik. Syncretism Distribution Modeling: Accidental Homophony as a Random Event. In *Proceedings of GLOW in Asia IX 2012*, pages 31–53. Mie University, Mie, Japan, 2013.

16 A. A. Shakhmatov. *Ocherk sovremennogo russkogo literaturnogo iazyka*. Urait, Moskow, 2018.

Hunting Ancestors: A Unified Approach for Discovering Genealogical Information

José João Almeida [ORCID]
Centro Algoritmi, Universidade do Minho, Portugal
Departamento de Informática, Campus de Gualtar, Universidade do Minho, Portugal
jj@di.uminho.pt

Rui Castro Mendes [ORCID]
Centro Algoritmi, Universidade do Minho, Portugal
Departamento de Informática, Campus de Gualtar, Universidade do Minho, Portugal
azuki@di.uminho.pt

---- **Abstract** --

This paper presents an unified approach for discovering genealogical information. It presents a frameworks for storing information concerning ancestors, locations, dates and documents. It also intends to provide a framework that is able to perform inference concerning dates by using constraints and for handling relations, locations and sources. The DSL presented also aims to help users store information from heterogeneous sources along with the evidence contained therein.

1 Introduction

In the area of digital humanities it is very important to register notes concerning people, genealogy, documents, news, photos, houses, family histories, etc. However, gathering information concerning ancestors is hard. Information is often incomplete including chronological information, lack of records or records containing incomplete or unreadable information and also photographs depicting people or places unknown. Depending on the timeframe or locale, there can be many sources for gathering information. These sources may include parish records, newspapers, and records concerning commerce or school. More recently, there can also be information in photographs. These often involve events where the family took photographs with a large assembly of its members and these photographs can support evidence of family ties that can be hard to find because the location of the photograph and many of the people therein can be unknown.

1.1 Motivation

The goal of this paper is to systematize information concerning sources, evidence and chronology. This will be performed by:

- adding reasoning concerning chronology, like probable dates of birth, christening, attending school, immigration, opening practice and death.
- having information concerning online sources for records and publications including newspapers

⊟ having information concerning documents, their OCR, evidence collected and inference

⊟ having information concerning inference, namely concerning dates, probable locations (e.g., birth, marriage, death), probable ancestors or descendants and family ties

1.2 Information

If we know that a given ancestor posted an ad in a newspaper advertizing a practice as dentist and warning against other practitioners who do not have any sort of education, one can conclude that that ancestor must have some studies in a medical school, and, if he is opening at office, it is probable he is somewhat young.

We want a system were we can add information that could be used to infer interval bounds concerning dates of birth and death, that is capable of providing clues concerning sites where one can search for information (e.g., specialized schools and universities where our ancestor could have learned dentistry).

1.3 Chronology

Information providing hints concerning chronology can be obtained from many sources. In the example of the ad posted in the newspaper, it is possible to infer an interval for that person's date of birth (e.g., it was posted by someone who is starting practice and thus that person must be under forty at the time). This information often only provides us with guesses in the form of interval bounds but provides clues that, when combined with other evidence, can pinpoint chronological information [7, 3, 6].

1.4 Ancestors knowledge base

Registering genealogical information is a challenging task. In order to take sound genealogical notes we are trying to follow an ontology approach. We want to be able to:

1. Define the base concepts (classes), their expected data; properties and the concept taxonomy (by default person, families, photos, places, etc);
2. Define the relations (inverse, domain, range, properties);
3. Populate the genealogy with persons, families, houses, dates, etc;
4. Query using partial information in order to better understand the facts present in documents;
5. Build a versatile reasoner capable of solving constraints about family relations, date intervals, names, etc;
6. Build a name module capable of following human conventions and intuitions;
7. Provide context-aware clues about complementary information sources.

In section 2 we will discuss the ancestor notebook (*ANB*) language to help populate the genealogy. In section 3 we will discuss constraints and inference about ancestors data. Finally, section 4 will present the conclusions and future work.

2 Ancestors notebooks

We will present and discuss an ancestors notebook (ANB) DSL – a textual document where we intent to register genealogical information using a simple wiki-inspired domain specific language [2, 1].

ANB are used to register information about ancestors to be incorporated in our Ancestor Knowledge Base. This information can be seen as an ontology over person concepts like persons, families, houses, places, time, several types of documents, photos, portraits, organizations, and their relations.

ANB include textual documents with inline formal tuples (frequently triples, data properties) using the following structure:

In practical situations, we create a set of ANBs covering different family branches, archives, funds, albums, etc. During the creation process, we often consult previously stored knowledge. We illustrate this by presenting a small extract of an ANB:

```
#/Eduardo Honório de Lima                   // main topic
#= Honório de Lima                          // alias
*1856 +1939

#doc HL1 {                                  // related document
= Honório de Lima street, Porto

- O capitalista portuense #[Eduardo Honório da Lima] (1856-1939),
 colecionador de arte e grande amador de música.
 ( toponymy Archive of Porto)
}

#B Teotónio Augusto de Lima                 // brother

#in-photo DA2-33                            // present in photo DA2-33
#dono fábrica de Cortumes do Bessa          // triples
#mecenas_of Museu Soares dos Reis           //
#lived_in casa HL1
#in-doc EAG1

===
#/photo:DA2-33                              // main topic is photo DA2-33
* c1890                                     // taken in circa 1890
#in-album DiasA2                            // from album ...
#file f-33.jpg                              // filename ...
#desc { family of Honório de Lima .... }    // textual description

===
#/house:casa HL1                            // main topic is house HL1
#= Moradia Honório de Lima I                // alias
#address Rua Cedofeita 492 a 498        // textual relation

- reconstructed em 1910
- it includes a small musical auditorium
```

A ANB can then be added (conciliated) with our ancestors knowledge base (AKB). First we can calculate the list of new instances – to check for typos (`anb-new-items myfamily.ab f.anb` command). And finally commit the new notes: (`anb-commit myfamily.ab f.anb` command). The commit process can produce warnings and errors when constraint violations are detected.

In a simplified way, `anb-commit` extracts triplets and subdocuments from the ANB and adds them to the AKB. There is also a simple script – `anb2triples f.anb` to extract a CSV version. This is the usual way of populating the ontology [4].

3   Inference

3.1   Example

Suppose we find a photograph with the portrait of a gentleman in his twenties. When we turn this photograph, we see the inscription "Jaime, irmão do tio Frederico". The logo and address in the photograph indicates that it was taken by a photographer who worked in that specific address from 1929 to 1935. This photo belonged to an aunt called Blandina Neves. We can summarize the following information:

- This photograph is of a person whose name contains Jaime as a given name;
- This person has a brother called Frederico;
- Frederico is an uncle of the person who wrote the inscription in the back of the photograph;
- Jaime is apparently in his twenties:
- The photograph was taken between 1929 and 1935.

The normal constraints about people longevity, age, people-related intervals (cf. section 3.4) also stand. Using this system, we may write the query:

```
Jaime brother_of Frederico, Frederico uncle_of Blandina Neves,
Jaime:20..30 in_photo F1, F1 photo_taken 1929..1935.
```

This query is rewritten by our system into Prolog:

```
person(P1), name_includes(P1, 'Jaime'),
person(P2), brother_of(P1, P2), name_includes(P2, 'Frederico'),
person(P3), uncle_of(P2, P3), name_includes(P3, 'Blandina Neves'),
photo(F1), in_photo(P1, F1), apparent_age (F1, P1, 20..30),
photo_taken(F1, 1929..1935).
```

`person` is a fact that stores all information concerning a given person. This information can be used by other predicates like `brother_of` or `uncle_of` for checking for family ties or `name_includes` for checking the name of a person. `photo` stores all the information concerning a photograph including the people therein, location and when the photo was taken. When mentioning that a given person appears in a photo, it is possible to add information concerning the apparent age of the person.

Knowing when the photo was taken and the apparent age of the individual, it is possible to provide an estimate of the birth date of Jaime using finite domains [5] and, given that Jaime and Frederico are brothers, of also providing an estimate on his birth date and thus narrow the search scope.

3.2   Dealing with incomplete information

It is possible that the system does not have enough information to answer our query. When this happens, the query concerning, e.g, a person or a photograph, will return a placeholder that will be filled with unknown information. For instance, it is possible that we do not have information concerning Jaime but even so be able to find out who Frederico is. In this case, the system creates a record with an unknown name, inf..sup information for dates of birth and death, unknown location, etc. Then the system instantiates Jaime to be one of the names and subsequently constraints the date of birth knowing that the subject was between 20 and 30 years old somewhere between 1929 and 1935, yielding the interval 1899..1915.

3.3 Dealing with replicated information

During the research for information concerning relatives, it is possible to have more than one record concerning a given person. If this happens, the system should run a query that takes two profiles and computes their similarity. The similarity depends on instantiation. This is performed by comparing names, birth and death dates and data concerning other events, e.g., relations, locations and occupation. The name comparison is performed by performing a name similarity procedure (e.g., it could involve computing the number of names in common and their relative order). If two individuals are deemed similar, the system can show both records to the user and suggest that they may be merged. If the user input is positive, those records are merged.

3.4 Defining rules

It is also possible to define rules concerning constraints regarding relations. It is also possible to define properties regarding relations. These rules will be used for creating relations automatically. For instance, sibling is symmetric and transitive. Thus, if A and B and A and C are siblings, the symmetric and transitive closures of these relations are also created automatically. Furthermore, it is possible to establish constraints that will be applied using constraint propagation [5] for shrinking the time intervals concerning birth or death of people because of family relations.

We present some examples of the system. We state that two siblings cannot be born more than 30 years apart, that a person can only be mother of another with more than 15 years but less that 50 and that it must be alive at the time of birth. If a person appears in an active event, it must be older than 10 and must be alive for the event to take place. If a photo is taken and a person has a given apparent age, then we have more information concerning the person's birth date.

```
spouse: symmetric, anti-reflexive
sibling: transitive, symmetric, anti-reflexive
mother: anti-transitive, anti-symmetric, anti-reflexive
father: anti-transitive, anti-symmetric, anti-reflexive
person(P)    :- P.death >= P.birth, P.death - P.birth <= 100
sibling(A,B) :- abs(A.birth - B.birth) < 30
mother(M, C) :- female(M), C.birth - M.birth > 15,
    C.birth - M.birth < 50, M.death >= C.birth
father(F, C) :- male(F), C.birth - F.birth > 15,
    C.birth - F.birth < 65, F.death + 1 >= C.birth
proof_of_life(E, P):- P.birth <= E.year, P.death >= E.year
is_active(E, P):- P.birth + 10  < E.year, P.death >= E.year
apparent_age(F, P, Apparent) :- (F.year - P.birth) in Apparent
```

3.5 Clues about information sources

Another important information in "hunting ancestors" is what information sources exist: these can contain clues of where, when and what to look for to complement current information about someone.

▶ **Example 1.** We know that someone born in Porto in 1870 is a dentist.
→ The next step could be consulting the anuário of University of Coimbra of (years between 1890..1900) and the registers of Medical-Surgical Academy of Porto (1836-1911).
→ Google("Name" site:repositorio-tematico.up.pt)

▶ **Example 2.** We know that someone emigrated to Brazil, Rio Grande do Sul in 1829.
→ The next step could be consulting the "Lista das cartas de liberdade dos escravos Rio Grande do Sul" (Portalegre, Pelotas, RGS)
→ `http://www.apers.rs.gov.br/arquivos/`[1]

In the current version, this module is in a very initial stage. They can be modelled with rules, in a similar way of what was done in section 3.4.

4   Conclusions

Although in a initial stage, this work already supports the importance of:

- having a constraint solver engine and versatile inference processes;
- providing a simple way of adding relations, their properties and related constraints;
- having domain specific languages for knowledge base population tasks;
- having a sound way of handing name similarity, alias and partial name unification;
- having simple queries dealing with partial names and family kinship (Luísa, sister of Margarida; tio Frederico; avô João).

Concerning future work, we aim to expand this system by creating a better name unification and similarity procedure, expand the DSL to better handle the information necessary and testing the system.

References

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

2   Tomaž Kosar, Nuno Oliveira, Marjan Mernik, Varanda João Maria Pereira, Matej Črepinšek, Cruz Daniela Da, and Rangel Pedro Henriques. Comparing general-purpose and domain-specific languages: An empirical study. *Computer Science and Information Systems*, 2010.

3   Philippe Laborie and Jerome Rogerie. Reasoning with Conditional Time-Intervals. In *FLAIRS conference*, pages 555–560, 2008.

4   J.B. Lamy. Owlready: Ontology-oriented programming in Python with automatic classification and high level constructs for biomedical ontologies. In *Artificial Intelligence In Medicine*, volume 80, pages 11–28, 2017.

5   Markus Triska. The Finite Domain Constraint Solver of SWI-Prolog. In *FLOPS*, volume 7294 of *LNCS*, pages 307–316, 2012.

6   Dongrui Wu and Jerry M. Mendel. Uncertainty measures for interval type-2 fuzzy sets. *Information Sciences*, 177(23):5378–5393, 2007. Including: Mathematics of Uncertainty. `doi:10.1016/j.ins.2007.07.012`.

7   Ran Zhao, Quang Xuan Do, and Dan Roth. A robust shallow temporal reasoning system. In *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies: Demonstration Session*, pages 29–32. Association for Computational Linguistics, 2012.

---

[1]   Download and grep `http://www.apers.rs.gov.br/arquivos/1169142561.Cat_Sel_Cartas_Liberdade_Vol_1.pdf`

SeCoGen – A Service Code Generator

Ricardo Queirós

ESMAD – Polytechnic of Porto, Portugal
CRACS – INESC TEC, Porto, Portugal
http://www.ricardoqueiros.com
ricardoqueiros@esmad.ipp.pt

──── **Abstract** ────

The architectural pattern of micro-services is being increasingly adopted by developers, facilitating the maintenance and scalability of the systems' code. The adoption and consumption of these micro-services are often seen on the front-end code of the Web applications. Nevertheless, this adoption obliges web designers/developers to know where to look for those web services, to read their documentation and to write the request/response code as well to control the corresponding UI rendering. This whole process is time-consuming and error-prone. This article introduces SeCoGen as an interactive code generator for Web service parsing and consumption. The generator benefits from an HTTP request template, a query normalizer and dynamic UI templates. In order, to validate the generator feasibility and usefulness, a REST API to search for countries is used.

1 Introduction

The Web is evolving at a fascinating rate. One of the reasons for its success is a large number of sophisticated frameworks to create responsive, adaptive systems with very high levels of performance. At the same time, the architectural pattern of micro-services has increasingly been adopted in Web development [1], enabling developers to write components using different languages, to facilitate system scalability, and to simplify the deploy of separate system components, among others. However, this architecture also brings a number of challenges. For example, spreading log information and tracking transactions that represent a single context, but are performed across multiple services. Another major difficulty is the search for the most appropriate services, the reading of its documentation (when it exists) and the coding of the request-response circuit and its respective visual rendering in the user's browser. To make things worse, the developer still has to worry about using code that is compatible with the several existent browsers supported by the market.

This article presents SeCoGen as a Web service code generator. The generator acts as an interactive environment organized in 3 steps: the developer 1) searches and selects the desired API; 2) selects the desired endpoint and, if necessary, enters the respective input parameters; 3) choose the template for the response rendering. After these steps, SeCoGen builds and bundles all the code needed to handle the Web service request/response circuit.

In order to evaluate SeCoGen usefulness and feasibility, the environment is tested using the countries API. The main goal is to produce a responsive Web catalog for European countries listing.

The remainder of this paper is organized as follows: Section 2 reviews the current code generation tools and techniques. In Section 3, we present SeCoGen and describe its two use cases: service register and service code generation. In Section 4, we validate the SeCoGen

environment enumerating the steps for the generation of a responsive Web catalog based on the Countries API. Finally, we conclude with a summary of our main contributions and a perspective of future research.

2   Code Generation

Code generation is typically defined as the process by which a compiler's code generator transforms an intermediate representation of source code into other forms of code which can be readily executed by a machine [5].

In the development realm, code generation can be a very important approach to boost productivity in the development process. Regardless of the code generation context, it fosters productivity, simplification, and consistency. In practice, using this approach, the developer writes the generator once and it can be reused as many times as he needs making it significantly faster than writing the code manually. Another important aspect is simplification since you typically generate your code from some abstract description. It means that your source of truth becomes that abstract description and not the code. That description is typically easier to analyze and check compared with the whole generated code [4]. Finally, with code generation, you get always the code you expect. The generated code is designed according to the same principles. With the code written manually instead you can have different developers using different styles and often introducing errors even in the most repetitive code.

There are several types of generation tools available such as: template engines, parser generators, ad-hoc applications and model/database driven design tools. In the following subsections, we detail the first three approaches.

2.1   Template engines

A template engine is basically a compiler that can interpret a template file containing special notations written in a simple template language. The simplest thing it can do is replacing this special notation with the proper data, given at runtime.

The majority of the template engines also supports other programming constructs such as simple flow control commands (e.g., for-loops, if-else-statements). There are several template engines out there. The most notable examples are Pug, Mustache, HandleBars, and EJS.

Pug (formerly known as "Jade") is a high-performance template engine influenced by Haml and implemented with JavaScript for Node.js and browsers. Pug is available via npm. Its rendering process is done by compiling the Pug source code into a JavaScript function that takes a data object as an argument. Finally, calling that function will return an HTML string rendered with the respective data.

Mustache and *Handlebars.js* are both logic-less templating languages which keep the view and the code separated. Both work by expanding tags in a template using values provided in a hash or object.

EJS is a very simple templating language that allows the generation of HTML markup with plain JavaScript. One of its distinguished features is the chance to download a browser build from the latest release and use it in a script tag.

2.2   Parser Generators

A parser generator is a tool that helps you to create parsers. A parser receives a piece of text and transforms it into an organized structure, such as an Abstract Syntax Tree (AST).

The parser generator will produce a new language programming source files (e.g. Java, C/C++, C#, Python, Ruby) which can be divided into several cooperating modules:

- *Lexer*: reads an input character or byte stream (i.e. characters, binary data), splits it into tokens using specified patterns, and generates a token stream as output.
- *Parser*: reads a token stream (often generated by a lexer), and matches phrases in your language via the specified patterns, and typically performs some semantic action for each phrase (or sub-phrase) matched. Each match could invoke a custom action, or generate an Abstract Syntax Tree (AST) for additional processing.

A parser generator takes a grammar (formal description typically written in EBNF format) as input and automatically generates source code that can parse streams of characters using the grammar. The generated code is a parser, which takes a sequence of characters and tries to match the sequence against the grammar [2].

Nowadays, there are several parser generators. The most notable examples are ANTLR, JavaCC and Flex.

ANTLR, ANother Tool for Language Recognition, is a tool which receives a grammar and generates source code files and other auxiliary files [4]. In practice, using ANTLR there are the following set of tasks:

1. define a lexer and parser grammar
2. invoke ANTLR: it will generate a lexer and a parser in your target language (e.g., Java, Python, C#, Javascript)
3. use the generated lexer and parser: you invoke them passing the code to recognize and they return to you an AST

2.3 Ad-Hoc Applications

Other tools were not mentioned since they are not suitable in the presented categories, but their huge importance obliges to create this category such as the scaffolding tools Yeoman, Slush and Lineman. These tools have something in common: they simplify the life of developers, by providing a way to create ready-to-use project optimized for a particular software platform, library or need.

Yeoman is a very popular and generic scaffolding system which generates with one command a new project that automatically implements all the Web best practices. The core of Yeoman is a generator ecosystem, on top of which developers build their own templates (thousands of templates available). Yeoman is written in JavaScript, so developing a generator requires simply to write JavaScript code and using the provided API. The workflow is also very simple: you ask the user information about the project (e.g., its name), gather configuration information and then generate the project [3].

Regarding GitHub stats, Yeoman takes the lead with 9.1K stars and 765 forks against 1.2K of Lineman with 89 forks. The Slush project has low values.

3 SeCoGen

SeCoGen is a Web environment for generating Web service code, namely: the code for the HTTP request (method, URL target, parameters, and body) and all the code needed to handle the response, from the parsing of the service response to the User Interface (UI) rendering.

The generator has two use cases: the service registration and the service code generation. In the former, developers will select the API service and its endpoints, define the placeholders for the input request parameters and define the queries to normalize the response data

according to supported UI templates. In the later, it is expected that anyone, including people without any programming knowledge, to use SeCoGen to automatically generate the service code in simple steps. The following subsections detail both use cases:

3.1   Service Registration

In order to be able to generate the code for a Web service, it must be properly configured and stored in a database. The architecture of SeCoGen service registration is depicted in Figure 1:
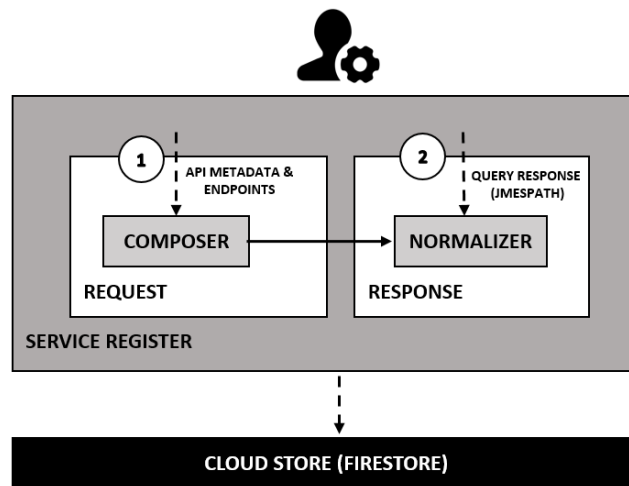
Figure 1 Service registration.

The service creation requires the admin/developer to fill several fields (1): the name of the API, its metadata (API description, main link, tags, etc.) and a list of HTTP messages supported by the API (commonly called of endpoints).

An HTTP request message includes a simple structure and is composed by:

- an HTTP method, a verb (like GET, PUT or POST) which describes the action to be performed;
- the request target, usually a URL;
- an (optional) body. Some requests send data to the server in order to update it: as often the case with POST requests (containing HTML form data).

The following code excerpt shows a generic HTTP message request template:

Listing 1 HTTP request message register.

```
-- HTTP request method and target --
GET http://api.endpoint.com/$1?q1=$2
-- HTTP body (optional) --
field=$3
...
field=$N
```

Both the request URL and body have placeholders that must be described during the registration process. Each placeholder will then be filled by the consumer of the service in order to generate an actual request.

After formalizing the request, it is necessary to analyze the type of response of the service and what is going to be presented visually to the user (2). Thus, SeCoGen provides a configurable template-based rendering system. At the moment, three templates are configured:

- Selector - displays the response of a Web service in an HTML selector. It is the ideal solution for secondary services where the result will be used in a larger process (e.g., web form)
- table - displays the response in tabular format with N columns. It is the ideal scenario to display a list of items, where each item has several textual characteristics (e.g., list of items to buy in an online store)
- Card - displays the response on a responsive card composed by data from different types, such as images, text, and buttons. Ideal for displaying catalogs (e.g., a store's product catalog)

A template is encoded in HTML/CSS. This separation of the request/response code from the UI render code is fundamental to the modularization and scalability of the system. At the same time, it enhances the contribution of different types of users in the generator. In this case, the templates will be fed into the system by Web designers. Here's an excerpt from a Bootstrap Card template to format any service response:

Listing 2 UI response template.

```html
<div class="card">
  <img src="$image" class="card-img-top">
  <div class="card-body">
    <h5 class="card-title">$title</h5>
    <p class="card-text">$subtitle.</p>
    <a href="$linkValue" class="btn btn-primary">more details</a>
  </div>
</div>
```

In order to map the actual response of service to one of the supported templates, it is necessary, in the service registration phase, to define a query that will filter the response data into a normalized format supported by the SeCoGen templates. For this normalization process, we use JMESPath as a query language for JSON. The JMESPath language is described in an ABNF grammar with a complete specification. This ensures that the language syntax is precisely defined. The following excerpt shows the mapping between a random service response and the normalized format supported by SeCoGen templates:

Listing 3 Service response normalization.

```
-- Example of a service response --
product:
[
 {
  img: "p1.jpg",
  product: "HP 15-DA0036NP",
  metadata: {desc:"A great HTP laptop", target:"hp15-DA0036NP.html"},
  stars: "3"
 },
 ...
]
-- JMESPath query --
"product[*].{
    scg_image: @img,
    scg_title: @product,
    scg_subtitle: @metadata.desc,
    scg_link: @metadata.target
}"
```

```
-- Normalized JSON --
[
 {
  "scg_image": "p1.jpg",
  "scg_title": "HP 15-DA0036NP",
  "scg_subtitle": "A great HTP laptop",
  "scg_link": "hp15-DA0036NP.html"
 },
 ...
]
```

3.2   Service Code Generator

With services properly configured and stored in the database, users can access SeCoGen to generate the desired Web service code. The architecture of this use case is presented in Figure 2:
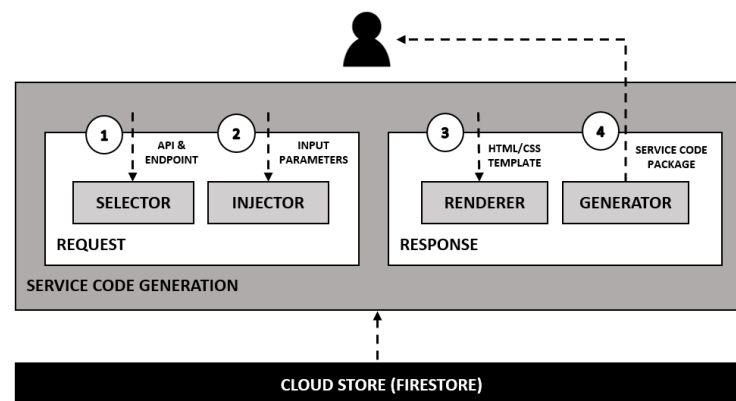
Figure 2 Service register.

The service code generation process is iterative and is organized in two phases: generation of the HTTP request and generation of the response parsing and its rendering. The user begins by selecting the desired API through a search based on tags (1). Next, all the endpoints of the selected API are listed. After choosing an endpoint and, if the endpoint has configurable parameters, the user is invited to define values for all of them (2). In a second step, the user chooses the template (3) that he intends to use to display the service response data (selector, table, or card).

The generator will normalize the service response to the SeCoGen required format and benefit from all the templates presented in the previous subsection to build a package with all the necessary files to represent the request/response circuit of the selected service. The following excerpt shows, in a high level of detail, how all this process is executed:

Listing 4 UI generation based on templates.

```
import * as jmespath from "./jmespath.js"
const responseData = ...
const normalizedJSON = jmespath.search(responseData, @JMESPathQuery)
Secogen.map(normalizedJSON, template)
```

3.3   Data Model

SeCoGen uses a real-time NoSQL database in the cloud called FireStore. Cloud Firestore is a flexible and scalable database for the mobile device, Web and server development from the Firebase and Google Cloud Platform. Like the Firebase Real-time Database, it keeps your data in sync in client applications through real-time listeners. In addition, it offers offline support for mobile and Web devices so you can build responsive applications that work regardless of network latency or Internet connectivity.

The data model of Cloud Firestore is quite flexible. The data is stored in documents that contain field mappings for values. These documents are stored in collections, which are document containers that are compatible with many different data types, from strings and simple numbers to complex and nested objects. You can also create sub-collections within documents and create hierarchical data structures that can be scaled as the database grows.

For the SeCoGen context two collections were defined:

- apis - stores information about APIs and its endpoints. For each endpoint it stores information about the HTTP request message, the conversion query, and other metadata;

- templates - stores all rendering templates which must be independent of the services and their implementation languages.

4   Validation

In order to evaluate SeCoGen usefulness and feasibility, the environment was tested using the countries API [1]. The Countries API allows to get information about countries through a very simple RESTful API.

In the service registration process, the API and some of the most important endpoints were added. In this case, the most important endpoint is the one that lists information from all countries by region. Then a query was defined to map the response with the Card template. On the generation side of the service, first, the user selected the API and its respective endpoint and injected EU (Europe) as the input parameter. Finally, the Card template was selected. The next excerpt shows the main code snippets:

Listing 5 Normalization code for the Countries REST API.

```
-- HTTP message request template
https://restcountries.eu/rest/v2/regionalbloc/$1
-- Normalization query --
"country[*].{
    scg_image: @flag,
    scg_title: @translations.pt,
    scg_subtitle: "Capital: @capital",
    scg_link: "https://pt.wikipedia.org/wiki/@translations.pt"
}"
```

After SeCoGen generated all the required code, the user downloaded the zip file and ran the code in a browser. The final result is shown in Figure 3:

---

[1] Link: https://restcountries.eu
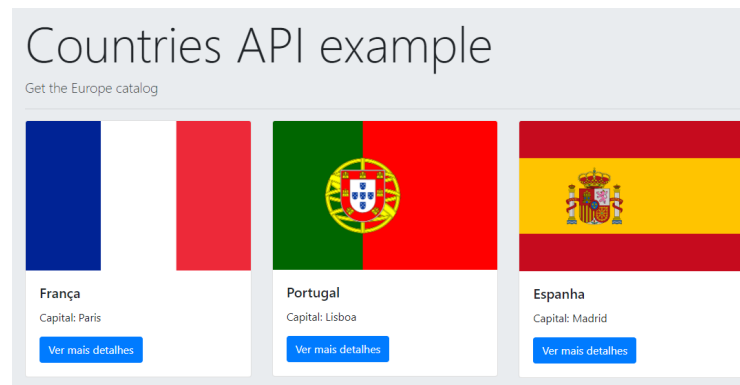
■ **Figure 3** Countries API response render based on the Card template.

5 Conclusion

This article presents SeCoGen as a code generator for the request-response circuit of a Web service. Its ultimate goal is to help anyone to generate code for a Web service without any need of prior domain and programming knowledge. For this purpose, we have explained the service registration process, which entails filling all the details of an API and its endpoints and the code generation process, which consists of three simple steps: API choice, endpoint choice with the respective input parameters injection and the rendering template choice. As a result, SeCoGen generates all necessary files and packages all of them into a ZIP file that can be downloaded by the user and executed in any modern browser.

In order to validate the generator, an API was used to list country data as a responsive Web catalog. The validation process has proven that SeCoGen has enormous potential, yet there are several issues still to be addressed.

In the near future, the following tasks will be tackled:

- support for POST requests and creation of a template based on a web form
- creation of a back office in order to facilitate the registration process of services
- support for GraphQL (an alternative to REST architectures)

References

1   P. D. Francesco. Architecting Microservices. In *2017 IEEE International Conference on Software Architecture Workshops (ICSAW)*, pages 224–229, April 2017. `doi:10.1109/ICSAW.2017.65`.

2   Robert Miller and Max Goldman. Software Construction. `https://ocw.mit.edu/ans7870/6/6.005/s16/classes/18-parser-generators/#reading_18_parser_generators`, 2016. Spring 2016. Massachusetts Institute of Technology: MIT OpenCourseWare.

3   Ricardo Queirós. PROud – A Gamification Framework Based on Programming Exercises Usage Data. *Information*, 10(2), 2019. `doi:10.3390/info10020054`.

4   Gabriele Tomassetti. A Guide to Code Generation. `https://tomassetti.me/code-generation/`, 2018. Accessed: 2018-11-01.

5   Seung-Su Yang, Hyung-Joon Kim, Nam-Uk Lee, and Seok-Cheon Park. Design of Automatic Source Code Generation Based on User Pattern Definition. In James J. Park, Vincenzo Loia, Gangman Yi, and Yunsick Sung, editors, *Advances in Computer Science and Ubiquitous Computing*, pages 1434–1439, Singapore, 2018. Springer Singapore.