5th Symposium on Languages, Applications and Technologies

SLATE'16, June 20–21, 2016, Maribor, Slovenia

Edited by

Marjan Mernik
José Paulo Leal
Hugo Gonçalo Oliveira

Editors

Marjan Mernik
Department of Computer Science
University of Maribor
Marjan.Mernik@um.si

José Paulo Leal
Faculty of Sciences
University of Oporto
zp@dcc.fc.up.pt

Hugo Gonçalo Oliveira
Centre of Informatics and Systems
University of Coimbra
hroliv@dei.uc.pt

ACM Classification 1998
I.2.7 Natural Language Processing, D.3 Programming Languages

ISBN 978-3-95977-006-4

Published online and open access by
Schloss Dagstuhl – Leibniz-Zentrum für Informatik GmbH, Dagstuhl Publishing, Saarbrücken/Wadern, Germany. Online available at http://www.dagstuhl.de/dagpub/978-3-95977-006-4.

Publication date
June, 2016

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

License
This work is licensed under a Creative Commons Attribution 3.0 Unported license (CC-BY 3.0): http://creativecommons.org/licenses/by/3.0/legalcode.

In brief, this license authorizes each and everybody to share (to copy, distribute and transmit) the work under the following conditions, without impairing or restricting the authors' moral rights:

- Attribution: The work must be attributed to its authors.

The copyright is retained by the corresponding authors.

Digital Object Identifier: 10.4230/OASIcs.SLATE.2016.0

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.

Editorial Board

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

ISSN 2190-6807

www.dagstuhl.de/oasics

Contents

5th Symposium on Languages, Applications and Technologies (SLATE'16).
Editors: Marjan Mernik, José Paulo Leal, and Hugo Gonçalo Oliveira
Open Access Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Preface

SLATE, the International Symposium on Languages, Applications and Technologies is an international conference with a long Portuguese tradition. It is rooted in two former conferences, each with a life span of about 10 years: Compilers, Programming Languages, Related Technologies and Applications (CORTA) ; and XML, Applications and Associated Technologies (XATA). The creation of SLATE was part of an effort to promote the internationalization of those conferences, by creating critical mass, attracting foreigners as participants, program committee and steering committee members, and by opening to new venues, specifically Madrid and Maribor.

The current fifth edition has papers from 25 authors, where Portuguese are outnumbered by the non-Portuguese authors. At first glance this could be interpreted a sign of success of the internationalization strategy. Unfortunately this is not the case because, this year, the overall number of submissions did not follow the growing trend of previous editions. In this process, SLATE may have lost contact with its Portuguese backbone, in particular due to the fact that this year's venue was too distant from the Iberian Peninsula.

As in previous editions, SLATE is divided in three main tracks: the processing of languages used to communicate among humans; the processing of languages used by humans to communicate with computers; and the processing of languages used for the communication among computers. These proceedings follow this same structure.

The Human-Human Languages track includes the following contributions:

- In "*Co-bidding graphs for constrained paper clustering*" the authors describe an approach for scheduling the presentations of conference papers on available slots, where papers are clustered based on the similarity of their content and on the reviewers preference to review them;
- In "*A Re-ranking Method Based on Irrelevant Documents in Ad-hoc Retrieval*" the authors describe a new information retrieval method that uses negative feedback to re-rank documents;
- In "*Comparing the Performance of Different NLP Toolkits in Formal and Social Media Text*" the authors compare the performance of the default pre-trained models of several NLP toolkits, freely available and developed in Java or Python, in the tasks of tokenization, part-of-speech tagging, chunking and named entity recognition, both in a newspaper corpus and in social media text;
- In "*Comparing and benchmarking semantic measures using SMComp*" the authors describe an on-line testbed tool for computing well-known or user-defined path-based similarity and relatedness measures, of word pairs or datasets, on different versions of WordNet.

The Human-Computer Languages track includes the following contributions:

- In "*LLLR Parsing: a Combination of LL and LR Parsing*" a new LLLR parsing approach is described, which is a combination of LL and LR parsing. Whenever LL conflict appears it triggers small embedded LR parsers. The approach has been validated on Java 1.0 programming language;
- In "*Locating User Interface Concepts in Source Code*" the authors explore whether strings and concepts displayed in the GUI of a running program can be located in its static source code, too. The study is performed on four Java applications (ArgoUML, FreeMind, PDFsam and Weka);

5th Symposium on Languages, Applications and Technologies (SLATE'16).
Editors: Marjan Mernik, José Paulo Leal, and Hugo Gonçalo Oliveira
Open Access Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

- In "*Declarative Rules for Annotated Expert Knowledge in Change Management*" a declarative domain–specific language for representing expert knowledge in the field of change management is presented. The declarative rules are written as an extension of the well–known deductive database language Datalog;
- In "*A Metamodel for Jason BDI Agents*" the authors describe a meta-model for modelling Belief-Desire-Intention (BDI) agents working on Jason platform;
- In "*Profile detection through source code static analysis*" an approach how to infer a programmer's profile through the analysis of his source code is analysed. The approach can be useful for the continuous evaluation of a student's progress on a programming course;
- In "*Context-Free Grammars: Exercise Generation and Probabilistic Assessment*" the authors deal with a probabilistic assessment whether two context free grammars (CFGs) are equivalent using a system of non-linear equations. The approach is exemplified on a simple CFG example of arithmetic expressions;
- In "*A Model-driven Engineering Technique for Developing Composite Content Applications*" a Domain-Specific Modelling Language (DSML) for composite content applications is proposed. It is then evaluated within an industrial case study.

The Computer-Computer Languages track includes the following contributions:

- In "*Eshu: an extensible web editor for diagrammatic languages*" the authors present a visual language environment for editing diagrammatic languages on the web;
- In "*Sni'per - a Code Snippet RESTful API*" the authors support web based collaboration for programmers, allowing them to share code snippets using a REST API;
- In "*Building a Dictionary using XML Technology*" the authors present an approach to the construction of an on-line dictionary using XML-based tools;
- In "*Automata Serialization for Manipulation and Drawing*" the authors present a visual language environment for handling and visualizing automata.

The articles published here focus very different and interesting areas of languages processing.

In conclusion, it is hoped that all the aforementioned papers will provide readers with some glimpse of research on different and interesting areas of languages processing that are presented at SLATE. Last but not least, we would sincerely like to thank the Program Committee for their assistance in the reviewing process.

Marjan Mernik
José Paulo Leal
Hugo Gonçalo Oliveira

Program Committee

Main Chair

Marjan Mernik
University of Maribor, Slovenia

Track Chairs

Marjan Mernik
(Human-Computer Languages)
University of Maribor, Slovenia

José Paulo Leal
(Computer-Computer Languages)
Universidade do Porto, Portugal

Hugo Gonçalo Oliveira
(Human-Human Languages)
Universidade de Coimbra, Portugal

Publication Chair

Alberto Simões
Universidade do Minho, Portugal

Organization Committee

Marjan Mernik
University of Maribor, Slovenia

Boštjan Slivnik
University of Ljubljana, Slovenia

Alberto Simões
Universidade do Minho, Portugal

Local Organizing Committee

Tomaž Kosar (co-chair)
University of Maribor, Slovenia

Matej Črepinšek (co-chair)
University of Maribor, Slovenia

Marjan Horvat
University of Ljubljana, Slovenia

Miha Ravber
University of Maribor, Slovenia

Martin Kraner
University of Maribor, Slovenia

Program Committee

Salvador Abreu
Universidade de Évora, Portugal

José João Almeida
Universidade do Minho, Portugal

Ana Alves
CISUC, University of Coimbra, Portugal

Jorge Baptista
Universidade do Algarve, Portugal

Fernando Batista
Instituto Universitário de Lisboa, Portugal

Mario Beron
Universidad Nacional de San Luis, Argentina

Barrett Bryant
University of North Texas, USA

João Paiva Cardoso
Universidade do Porto, Portugal

Nuno Carvalho
Universidade do Minho, Portugal

Matej Črepinšek
University of Maribor, Slovenia

Daniela da Cruz
Universidade do Minho, Portugal

Gabriel David
Universidade do Porto, Portugal

Alberto Diaz
Universidad Complutense de Madrid, Spain

Brett Drury
Universidade de São Paulo, Brasil

Luís Ferreira
Inst. Politécnico do Cávado e do Ave, Portugal

Jean-Cristophe Filliâtre
Lab. de Recherche en Informatique, France

Pablo Gamallo
Univ. de Santiago de Compostela, Spain

Alda Lopes Gançarski
Inst. Nat. des Télécommunications, France

5th Symposium on Languages, Applications and Technologies (SLATE'16).
Editors: Marjan Mernik, José Paulo Leal, and Hugo Gonçalo Oliveira
Open Access Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Program Committee

Hugo Gonçalo Oliveira
CISUC, Universidade de Coimbra, Portugal

Xavier Gómez Guinovart
Universidade de Vigo, Spain

Pedro Rangel Henriques
Universidade do Minho, Portugal

Jan Janousek
Czech Technical University, Czech Republic

Geylani Kardas
Ege University, Turkey

Jan Kollar
Technical University of Kosice, Slovakia

Ioannis Korkontzelos
NaCTeM, University of Manchester, UK

Tomaž Kosar
University of Maribor, Slovenia

Eugenijus Kurilovas
Centre of Inf. Tech. in Education, Lithuania

José Paulo Leal
Universidade do Porto, Portugal

António Menezes Leitão
Universidade Técnica de Lisboa, Portugal

Giovani Librelotto
Universidade Federal de Santa Maria, Brasil

João Correia Lopes
Universidade do Porto, Portugal

Ivan Lukovic
University of Novi Sad, Serbia

Paulo Matos
Instituto Politécnico de Bragança, Portugal

Marjan Mernik
Univerza v Mariboru, Slovenia

Kratky Michal
Technical University of Ostrava, Czech Republic

Nuno Oliveira
Universidade do Minho, Portugal

Alexander Paar
TWT GmbH Science & Innovation, Germany

Lluís Padró
Universitat Politècnica de Catalunya, Spain

Thiago Pardo
Universidade de São Paulo, Brasil

Senja Pollak
Jožef Stefan Institute, Slovenia

Jaroslav Porubän
Technická univerzita v Košiciach, Slovenia

Ricardo Queirós
Instituto Politécnico do Porto, Portugal

José Carlos Ramalho
Universidade do Minho, Portugal

Cristina Ribeiro
Universidade do Porto, Portugal

Ricardo Rocha
Universidade do Porto, Portugal

Casiano Rodriguez-Leon
Universidad de La Laguna, Spain

Dietmar Seipel
University of Würzburg, Germany

José Luis Sierra
Universidad Complutense de Madrid, Spain

Alberto Simões
Universidade do Minho, Portugal

Bostjan Slivnik
Univerza v Ljubljani, Slovenia

Peter Sloep
Open Universiteit, Netherlands

Jasmina Smailović
Jožef Stefan Institute, Slovenia

Simão Melo de Sousa
Universidade da Beira Interior, Portugal

Jakub Swacha
University of Szczecin, Poland

Maria João Varanda Pereira
Instituto Politécnico de Bragança, Portugal

List of Authors

Salvador Abreu
Department of Computer Science
University of Évora
Évora, Portugal
spa@di.uevora.pt

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

Mohand Boughanem
University of Toulouse
IRIT lab France
bougha@irit.fr

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

Helder Correia
Faculty of Sciences
University of Porto
Porto, Portugal
up201108850@fc.up.pt

Teresa Costa
Faculty of Sciences
University of Porto
Porto, Portugal
teresa.costa@dcc.fc.up.pt

Ferhat Erata
UNIT Information Technologies R&D Ltd.
ideEge Technology Development Zone
Ege University, Izmir, Turkey
ferhat@computer.org

Miguel Ferreira
Faculdade de Ciências
Universidade do Porto
Porto, Portugal
miguelferreira108@gmail.com

Hale Gezgen
R&D Center
Koçsistem Inform. and Comm. Services Inc.
Üsküdar/Istanbul-Turkey
hale.gezgen@kocsistem.com.tr

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

Eliana Grande
Departamento de Informática
Universidade do Minho
Braga, Portugal
eliana.tiba@ifgoiano.edu.br

Hawete Hattab
Umm Al-qura University
Department of Mathematics
Makkah, KSA
hshattab@uqu.edu.sa

Pedro Rangel Henriques
Departamento de Informática
Universidade do Minho
Braga, Portugal
pedrorangelhenriques@gmail.com

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

Nada Lavrač
Jožef Stefan Institute
Ljubljana, Slovenia
nada.lavrac@ijs.si

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

5th Symposium on Languages, Applications and Technologies (SLATE'16).
Editors: Marjan Mernik, José Paulo Leal, and Hugo Gonçalo Oliveira
Open Access Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

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

Nelma Moreira
Faculdade de Ciências
Universidade do Porto
Porto, Portugal
nam@dcc.fc.up.pt

Falco Nogatz
Department of Computer Science
University of Würzburg
Würzburg, Germany
falco.nogatz@uni-wuerzburg.de

Daniel Ferreira Novais
Departamento de Informática
Universidade do Minho
Braga, Portugal
danielnovais92@gmail.com

Ana Oliveira Alves
CISUC, DEI
University of Coimbra
Coimbra, Portugal
ana@dei.uc.pt

Mehmet Onat
R&D Center
Koçsistem Inform. and Comm. Services Inc.
Üsküdar/Istanbul-Turkey
mehmet.onat@kocsistem.com.tr

José Carlos Paiva
Faculty of Sciences
University of Porto
Porto, Portugal
up201200272@fc.up.pt

Alexandre Pinto
CISUC, DEI
University of Coimbra
Coimbra, Portugal
arpinto@student.dei.uc.pt

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

Ricardo Queirós
ESEIG/IPP & INESC-TEC
Porto, Portugal
ricardoqueiros@eseig.ipp.pt

Rogério Reis
Faculdade de Ciências
Universidade do Porto
Porto, Portugal
rvr@dcc.fc.up.pt

Marko Robnik-Šikonja
University of Ljubljana
Fac. of Computer and Information Science
Ljubljana, Slovenia
marko.robnik@fri.uni-lj.si

Ana Salgado
Instituto de Lexicologia e
Lexicografia da Língua Portuguesa
Academia das Ciências de Lisboa, Portugal
anacastrosalgado@gmail.com

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

Alberto Simões
Centro de Estudos Humanísticos
Universidade do Minho
Braga, Portugal
ambs@ilch.uminho.pt

Tadej Škvorc
University of Ljubljana
Fac. of Computer and Information Science
Ljubljana, Slovenia
ts9675@student.uni-lj.si

Boštjan Slivnik
University of Ljubljana
Fac. of Computer and Information Science
Ljubljana, Slovenia
bostjan.slivnik@fri.uni-lj.si

Georgi Smirnov
Departamento de Matemática
Universidade do Minho
Braga, Portugal
smirnov@math.uminho.pt

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

Baris Tekin Tezel
International Computer Institute
Ege University
Izmir, Turkey
baris.tezel@deu.edu.tr

Mohamed Tmar
Sfax University
Multimedia Information Systems and
Advanced Computing Laboratory
Sfax, Tunisia
mohamedtmar@yahoo.fr

Maria João Varanda Pereira
Dpt. de Informática e Comunicações
Instituto Politécnico de Bragança
Bragança, Portugal
mjoao@ipb.pt

Rüdiger von der Weth
Faculty of Business Administration
Dresden University of Applied Sciences
Dresden, Germany
weth@htw-dresden.de

Alexander Werner
Faculty of Business Administration
Dresden University of Applied Sciences
Dresden, Germany
alexander.werner@htw-dresden.de

Co-Bidding Graphs for Constrained Paper Clustering

Tadej Škvorc[1], Nada Lavrač[2], and Marko Robnik-Šikonja[3]

1 **University of Ljubljana, Faculty of Computer and Information Science, Ljubljana, Slovenia**
`marko.robnik@fri.uni-lj.si`
2 **Jožef Stefan Institute, Ljubljana, Slovenia; and University of Nova Gorica, Nova Gorica, Slovenia**
`nada.lavrac@ijs.si`
3 **University of Ljubljana, Faculty of Computer and Information Science, Ljubljana, Slovenia**
`marko.robnik@fri.uni-lj.si`

Abstract

The information for many important problems can be found in various formats and modalities. Besides standard tabular form, these include also text and graphs. To solve such problems fusion of different data sources is required. We demonstrate a methodology which is capable to enrich textual information with graph based data and utilize both in an innovative machine learning application of clustering. The proposed solution is helpful in organization of academic conferences and automates one of its time consuming tasks. Conference organizers can currently use a small number of software tools that allow managing of the paper review process with no/little support for automated conference scheduling. We present a two-tier constrained clustering method for automatic conference scheduling that can automatically assign paper presentations into predefined schedule slots instead of requiring the program chairs to assign them manually. The method uses clustering algorithms to group papers into clusters based on similarities between papers. We use two types of similarities: text similarities (paper similarity with respect to their abstract and title), together with graph similarity based on reviewers' co-bidding information collected during the conference reviewing phase. In this way reviewers' preferences serve as a proxy for preferences of conference attendees. As a result of the proposed two-tier clustering process similar papers are assigned to predefined conference schedule slots. We show that using graph based information in addition to text based similarity increases clustering performance. The source code of the solution is freely available.

1998 ACM Subject Classification I.2.8 Problem Solving, Control Methods, and Search

Keywords and phrases Text mining, data fusion, scheduling, constrained clustering, conference

Digital Object Identifier 10.4230/OASIcs.SLATE.2016.1

1 Introduction

In many real world situations data can be present in various different formats and can therefore be difficult to understand. In order to efficiently use such data it must first be converted into a common format. Combining data present in different forms is known as data fusion and is a useful technique, particularly in machine learning, where data must be present in the form of feature vectors. One field where data fusion can be useful is conference organization.

© Tadej Škvorc, Nada Lavrač, and Marko Robnik-Šikonja;
licensed under Creative Commons License CC-BY

Scientific conferences, which allow scientists to share new discoveries, are a key to progress in science. Several conferences are very large. Organizing them can be difficult and time consuming. Various tools have been developed to help conference organizers deal with this problem by automating some of conference management tasks. However, scheduling paper presentations is currently still performed manually to a large extent. This can be time consuming, as large conferences often have a lot of presentations that must be grouped together based on similarities and differences between them.

A conference schedule usually consists of multiple time slots which contain semantically similar papers. We propose automated paper scheduling using text mining to find similar papers, grouping similar papers using clustering and assigning them into schedule time slots. Conference organizers may have access to additional metadata describing the papers. Such data is sometimes present in a graph form which can be vectorized and used in clustering.

In general, text documents can be linked within a graph where two documents are connected if they share the same author, if they are published in the same publication or based on citations in the paper's list of references. Such graphs may contain a large amount of information, which is mostly ignored by conventional text mining methods. To make graph data suitable for use with standard machine learning algorithms, a feasible approach is to convert the graphs into a feature vector format. To this end, we have adapted the methodology proposed by Grčar et. al. [6], which converts the information from graphs to feature vectors using the Personalized PageRank (PPR) algorithm [11] and then combines these vectors with text representation in bag-of-words (BOW) vectors. The combined feature vectors can be used by a wide variety of machine learning algorithms for different tasks, such as classification model construction.

We describe how this method can be adapted to categorize and cluster similar conference papers based on the textual content of their abstracts and titles in combination with the reviewers' bidding information collected during the conference evaluation period, where the reviewers select papers they would like to review and the ones they would rather not. This information is used to create a graph, where two papers are connected if the same reviewer expressed the wish to review them. We use this information in combination with the textual information to cluster papers.

The novelty of the proposed methodology is due to combining three previously unrelated approaches: (a) paper clustering using text-based similarity of BOW vectors, (b) paper similarity computation using the co-bidding graph to compute PageRank-based instance weighting, (c) constrained clustering for matching paper clusters to appropriate conference slots, and finally, (d) a user friendly web interface for conference organizers. The utility of the proposed approach was show-cased on the AIME 2013 conference (14th Conference on Artificial Intelligence in Medicine), where the results of automatic approach were nearly as accurate as the manual conference scheduling approach.

The paper is organized into six sections. In Section 2 we present the related work. Section 3 describes the data set we used. In Section 4 we describe the method used to enrich text with metadata present in graphs and how we used this method to group similar papers. In Section 5 we describe a web application used for conference schedule management that supports semi-automated schedule construction. Section 6 concludes the paper.

2 Related work

Several authors present methods of finding similar academic papers using graph-based metadata. Such methods can produce better results than methods using only text similarity

approaches. Huynh et al. [8] and Liang et al. [10] present methods that use graphs constructed from citations to find similar papers. Grčar et al. [6] show how such data can be effectively used in combination with text. They describe a method which fuses graph data with standard bag-of-word vectors into a single feature vector format. These vectors can be used as input to standard machine learning algorithms. Our approach differs from other approaches searching similar papers by using a graph constructed from preferences expressed by reviewers during the conference evaluation period. The information contained in such graphs have yet not been exploited. Additionally, we use this information in constrained clustering with the final aim to construct a useful schedule.

Academic conference recommendation systems help users select talks and presentations they would likely be interested in. Several papers [14, 21, 22] show that information extracted from socially-aware networks can be helpful. Our approach is not oriented towards conference attendees but conference organizers. It applies fusion of text and graph information to conference scheduling. Since a high quality schedule is a prerequisite for conference attendee recommendations, our approach can be viewed as a foundation for user recommendations. Instead of using a socially-aware network constructed during the conference, we construct a graph from the opinions reviewers expressed during the paper review period.

Constrained clustering imposes specific constraints to the results produced by clustering to improve performance. Wagstaff et al. [20] proposes a method of imposing must-link and cannot-link constraints that limit which examples can be in the same cluster. Zhu et al. [23] presents a heuristic method that imposes size constraints to clusters. We use a new approach to limit cluster sizes so they match a predefined conference schedule structure.

3 Automatic conference scheduling

Automating conference scheduling is a hard task. To solve it we first find semantically similar papers and then group them together according to the conference schedule. In this section we present our approach, which uses text-mining enriched with information obtained from the reviewers' co-bidding graph to find similar papers.

3.1 Problem overview

To automate conference scheduling we must solve two separate problems. The first is finding papers that are semantically similar. This is important because conference schedules are usually composed of several *sessions*. Each session contains paper presentations related to a specific research topic and is usually named with the adequate topic name. In automatic scheduling each of these sessions must be filled with similar papers. We solve this problem using clustering. First, each paper is turned into a Bag-of-Words (BOW) vector, and vector components are weighted with tf-idf (term frequency–inverse document frequency) [16]. This vector is constructed from the abstract and title of the paper. We extend the vector with paper similarity components extracted from conference reviewers' co-bidding preferences. After this the papers are clustered using clustering algorithms.

Other methods can also be used to solve this problem. Topic modeling [2] is commonly used to assign text documents into separate topics and could be used to find similar papers. However, such methods only take into account the text of the paper and not the additional graph-based metadata present in our case. Topic modeling also has to be trained on a large corpus before it can be used to assign documents into different topics. For use in a general conference management tool the training corpus would need to encompass papers from a wide variety of topics.

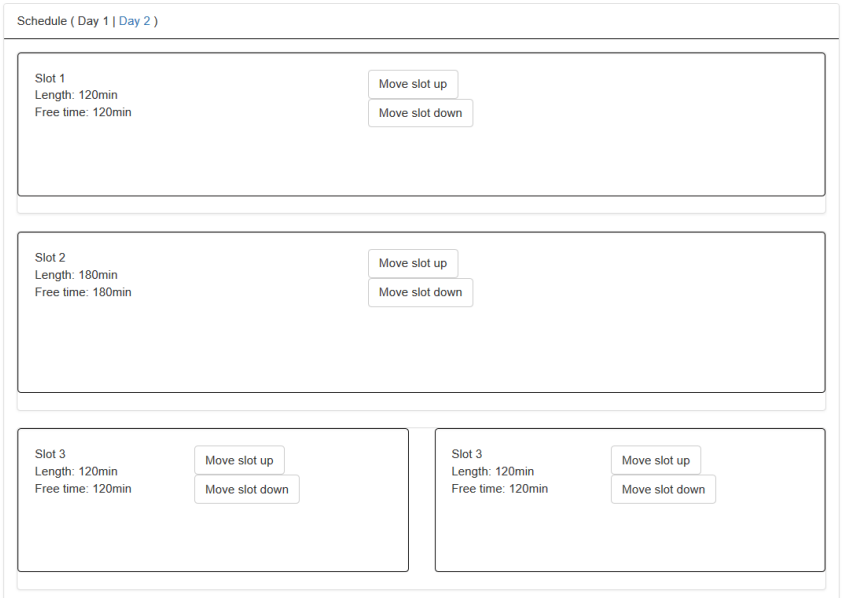


Figure 1 An example of an empty conference schedule. Each slot corresponds to a session that needs to be filled with papers from the same research topic. Sessions in Slots 1 and 2 occur sequentially, while Slot 3 has two parallel sessions. The application allows conference organizers to add new slots, move the slots around and set the duration of the slots. It allows organizers to create a multi-day conference program.

In our approach, clustering returns several groups of papers where each group corresponds to a different research topic. The second problem we need to solve is to construct a conference schedule from these groups. Some aspects of the schedule, such as keynote presentations and lunch breaks, are independent of paper presentations. Because of this we let conference organizers manually construct the conference structure. The structure consists of different blocks with prespecified duration. The blocks can occur either sequentially or in parallel. After the schedule structure is constructed, conference organizers decide which of the blocks will be automatically filled with thematically similar papers and which should be used for other purposes, such as keynote presentations. To integrate the automatic scheduling with manual schedule structure construction we built a user friendly web application. The application allows conference organizers to first manually construct the schedule structure and then automatically place papers into the schedule based on similarities between them. An example of an empty schedule constructed with this application is presented in Figure 1. An overview of the entire process is presented in Figure 2.

3.2 Data set

We tested our approach on a real-world example. Below we give a short description of the data we used. The data set consists of papers from a specific conference. The papers were described by abstracts and titles. We used this data to create feature vectors describing each paper. The BOW vectors were obtained by first removing stop words from the abstracts and titles and then weighting the vector components with tf-idf [16]. The data also included a list of reviewer preferences. For each paper, the reviewers selected one of the following opinions: *"I want to review this paper"*, *"I can review it"*, *"I prefer not to review it"* and *"I have a conflict of interest"*.

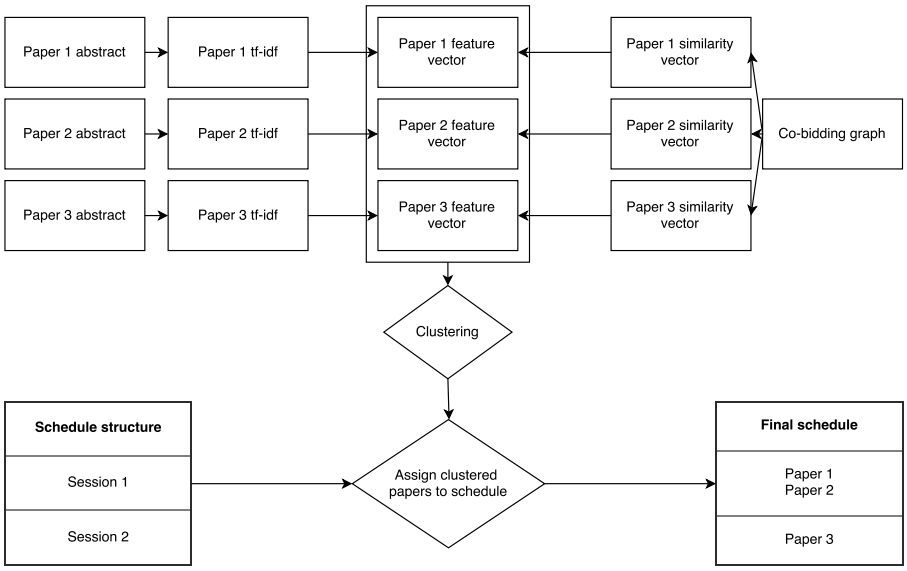


Figure 2 A summary of the entire scheduling algorithm. Each paper is described by the tf-idf vector of its abstract and similarity to other papers, which is obtained from the co-bidding graph. The papers are clustered and assigned to a preconstructed conference schedule.

From these preferences we constructed a graph in which two papers are connected if the same reviewer expressed a wish to review them. Additionally, each connection is weighted. Connections between two papers where the reviewer expressed the opinion *"I want to review this paper"* for both papers were assigned larger weights (we have chosen the weight of 4) than the connections between two papers with opinions *"I want to review this paper"* and *"I can review it"*, which were weighted with the weight of 2. Two abstracts, both with the opinion *"I can review it"*, were weighted even lower, with the weight of 1. Papers with other combinations of expressed preferences were not connected. We created such a graph for each reviewer and combined them into a single co-bidding graph by summing the edge weights from all the reviewers' graphs.

This presents a way of modeling the experts' opinion on the similarity of the articles. Since most conference reviewers tend to focus on a limited number of research fields and they usually review papers from these fields, papers they review tend to be semantically similar. By weighting the graph we can assign larger weights to links between nodes (papers) for which reviewers express stronger preferences, as this indicates that they are likely to be similar and from the same topic. The papers connected by the same reviewer are also likely to be interesting to the same group of conference attendees, of whom the reviewers are an excellent and valuable sample. Even if the reviewers tend to follow more than one topic this is not necessarily bad for the connected papers. First, they might be logically connected and second, the effect of a single reviewer is limited as co-bidding information is merged with textual similarity.

Past research [21] shows that recommendations from other conference participants can be useful for finding similar and interesting papers. Such recommendations cannot be obtained before the start of the conference. With our approach, recommendations from reviewers are used as a substitute, since they are already familiar with some of papers that appear in the conference. The goal of our automatic scheduling is to construct a program schedule

that pleases as many attendees as possible. An important aspect of this is placing paper presentations from the same topic that groups of people would find interesting in the same slot and not to overlap paper presentations with similar audience. It is a reasonable assumption that reviewers want to review papers they find interesting, therefore the constructed co-bidding graph links paper presentations that should be placed close to each other. The co-bidding graph is therefore useful in automatic scheduling.

3.3 Enriching text with co-bidding preferences

The co-bidding graph needs to be converted into a form suitable for clustering algorithms. To this end, we used the Personalized PageRank (PPR) algorithm [11]. This algorithm was designed to assess importance of graphs consisting of web pages. For a given starting page it computes importance of all pages relative to it using a random-walker model. The PageRank R' For a given set of pages can be calculated using the following equation:

$$R'(u) = c \times \sum_{v \in B_u} \frac{R'(v)}{N_v} + cE(u)$$

where B_u is a set of all pages linking to u, N_v is the number of links from v and c is a normalization factor, which ensures that the L_1 norm of R' is equal to 1 and must be maximized. $E(u)$ represents the PageRank source and is defined as follows.

$$E(u) = \begin{cases} 1 & : \text{u is starting page} \\ 0 & : \text{otherwise} \end{cases}$$

The algorithm can be used to convert information from graph structure into a BOW vector that can be used together with the information obtained by text mining [6]. If we assign a unique word to each web page (a node in the graph) and the user randomly navigating the graph writes down that word each time he/she visits the node, we get a textual document describing our graph. Such a document contains a higher frequency of words that are strongly linked to the starting node and can be naturally combined with textual data contained in the node. By using the Personalized PageRank algorithm we essentially get a normalized BOW vector (a vector holding the relative frequencies of individual words) describing such a document. This is useful as we get a description of the graph in the same form as the BOW vectors we constructed from abstracts and titles, and both forms can therefore be merged.

We use this approach to extract information from biddings expressed by the reviewers. In our bidding graph, two papers are connected if a reviewer expressed a wish to review both of them. Two connected papers are likely to contain similar topics. Reviewers are not picking papers to review at random, but rather because they are from the field they are interested in. Consequently, relevant semantically similar papers will be connected. Additionally, since the connections are weighted, reviewers' stronger opinions will have stronger connections. Applying the Personalized PageRank algorithm to our graph, starting from a specific paper, the algorithm will return larger probabilities for papers that are semantically close to that paper. In this way we obtain vectors containing probabilities of other papers being similar to a given paper. The PPR and the BOW vector can be merged and treated with the same approaches in the processing pipeline. Note also that the bidding graph contains semantic similarity information captured from human experts, which is an important added value of our approach.

Figure 3 shows an example graph obtained from reviewers' preferences. If we apply PPR to this graph starting from paper 1, we get the PPR vector $[0.27, 0.20, 0.33, 0.09, 0.06, 0.05]$.

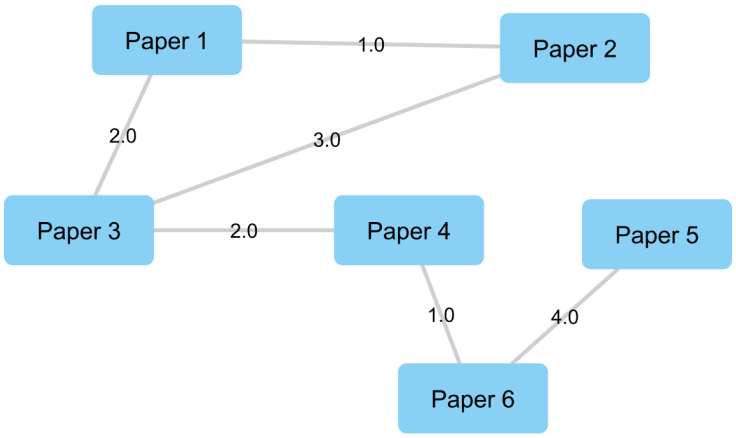


Figure 3 An example graph obtained from reviewers' preferences. By running the Personalized PageRank algorithm starting at a specific node in the graph, we obtain a vector describing how important other papers are for this specific paper.

This vector contains high values for papers 2 and 3, which are closely connected to paper 1. Running the algorithm starting from paper 6 returns vector $[0.03, 0.05, 0.10, 0.10, 0.31, 0.41]$, showing that this paper is closely connected to papers 5 and 4. In our graph edge weights correspond to the number of reviewer's that wanted to review both papers in the paper pair and also indicates the strength of their preferences.

The two vectors (the tf-idf vector of the abstract and title and the PPR vector of the co-bidding graph) are combined into a single feature vector. Since the PPR vectors are normalized, we can treat them as a separate tf-idf vector. We combine the two vectors by multiplying each one by 0.5 and concatenating them. It is possible to weight the vectors differently, which would place more weight on one of the vectors. The result is a single normalized feature vector. This final feature vector is clustered with different algorithms to group similar papers based both on their textual content and on the preferences expressed by the reviewers. This entire process is summarized in Figure 4. The described method combining both graph- and text-based information could also be used in classification.

3.4 Constrained clustering

The final step is to assign similar papers from the same cluster into one of several predefined schedule time slots. To do this we impose constraints on the clustering, as unconstrained clustering returns groups that do not match the schedule structure. The constraints are implemented iteratively, by matching and filling in the largest empty time slot with papers from the largest cluster until it is full. If a cluster large enough to fill the empty time slot does not exist we rerun the clustering with arguments that produce larger clusters. An overview of our approach is presented in Figure 5. We tested the method using *Affinity propagation* [4], *DBSCAN* [3], *Agglomerative clustering* [9], *K-means* [7] and *Mean Shift* clustering [5], as described in Section 5.

4 The conference scheduling application

We implemented the described methodology in a web application that supports program chairs in conference scheduling by automatically grouping similar articles into predefined

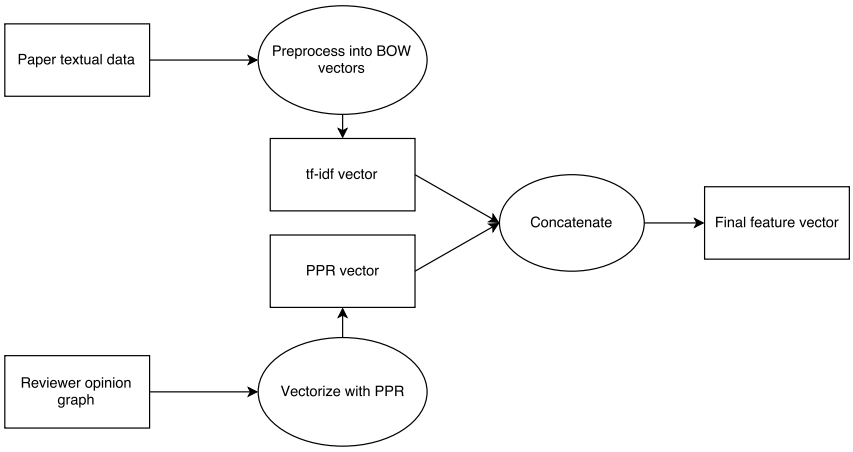


Figure 4 The process to obtain feature vectors. When vectorizing the co-bidding graph with PPR, the starting node of the PPR algorithm corresponds to each paper in turn. This produces a unique PPR vector for each paper. Since both the PPR and tf-idf vectors are normalized, the final feature vector is also normalized.

schedule time slots and also allows manual improvements. The source code of the application is available at `https://github.com/TadejSk/conference-scheduler`. The program chairs first construct the structure of the presentation schedule or import one of the stored schedules. They import papers from a database or manually add them into the application. The application automatically constructs a schedule which is presented to the program chair and allows editing. The application allows chairs to work on multiple conferences at the same time and automatically saves the schedules on the web server running the application. Figure 7 shows an example how paper clusters are visualized by our application and Figure 6 shows the user interface for schedule construction.

5 Evaluation

We evaluated several aspects of our approach. We compared several clustering algorithms to determine which one produces the best results. We tested both the quality of the results as well as the running time of the algorithms. To determine the effect of the co-bidding graph on the final results we compared the results of standard text-mining methods with the results obtained when those methods were fused with the information retrieved from the co-bidding graph. We compared the results by expert analysis and using the silhouette score [15].

5.1 Comparing different clustering methods

As described in Section 3.3, the combined feature vectors produced by our method can be applied to most clustering algorithms. We tested the method on a number of different clustering algorithms implemented in scikit-learn [12] to determine which of them is the most suitable for this type of data. The algorithms we tested were *Affinity propagation*, *DBSCAN*, *Agglomerative clustering*, *K-means* and *Mean Shift* clustering.

For some clusterings examining the visualization of the results was enough to determine that they are not suitable. *Mean Shift clustering* and *DBSCAN* produced clusters that had no clear structure and appeared random on our data set. They required finely tuned parameters to return more than one cluster. A visualization of results returned by Mean Shift is presented

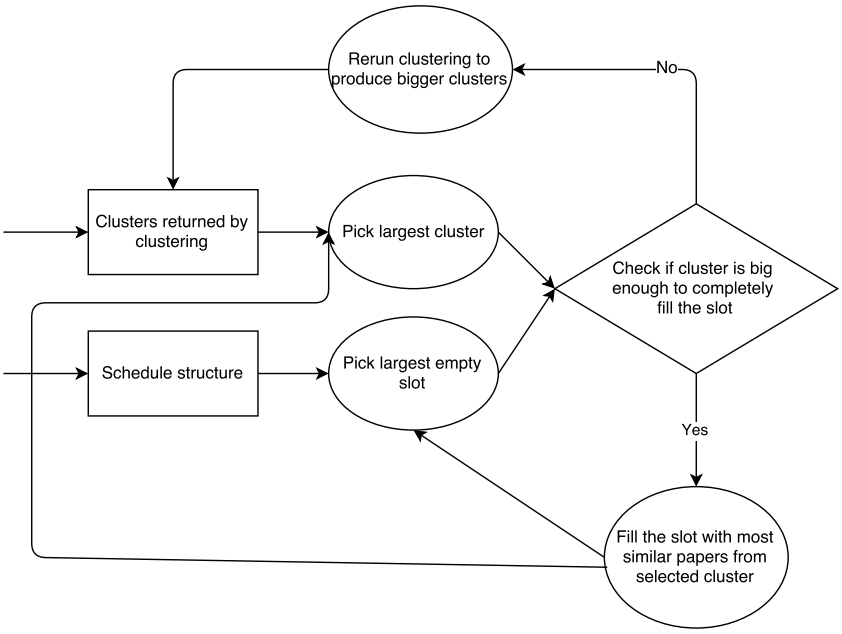


Figure 5 The iterative algorithm that fills an empty conference schedule with clustered papers.

in Figure 8. Visually the best results were obtained by *K-means clustering*, which consistently returned well structured clusters. We also tested the modifications by Sculley [17], which aims to improve the performance of K-means clustering for web applications. The modifications returned similar results while improving the performance. *Agglomerative clustering* returned as visually appealing results as K-means clustering. *Affinity propagation* also returned good results, but does not allow to explicitly set the number of clusters. In practice this is problematic, since the number of conference sessions and the number of clusters should be close.

We also compared the effect of clustering algorithms on the execution time of automatic scheduling. The choice of algorithm had little effect on the overall execution time. We tested the execution time by running the entire automatic scheduling process on an example data set containing 41 papers, using a 2.4 GHz processor. The fastest algorithm was Agglomerative clustering, which finished in 4.31 seconds. Mean Shift was the slowest algorithm and finished in 4.96 seconds. The difference between the fastest and the slowest algorithm was 12%.

5.2 Effect of the co-bidding graph

We tested our approach on papers from the AIME 2013 conference (14th Conference on Artificial Intelligence in Medicine) [13]. The conference schedule consisted of 43 paper presentations scheduled in 9 sessions. We evaluated the approach by comparing the results returned by our method with the actual schedule that was used in the AIME 2013 conference. We also tested the effect of the additional graph data.

We first tested the effect of the co-bidding graph using the silhouette score [15]. The silhouette score is an internal clustering validity measure and measures the quality of produced clusters. It does so by comparing the dissimilarity of an object with other objects in the same cluster to its dissimilarity with objects from a different cluster. If we define $a(i)$ as the average dissimilarity of i to all other objects in its cluster and $b(i)$ as the smallest average dissimilarity of i to all other objects in another cluster, we can compute the silhouette score

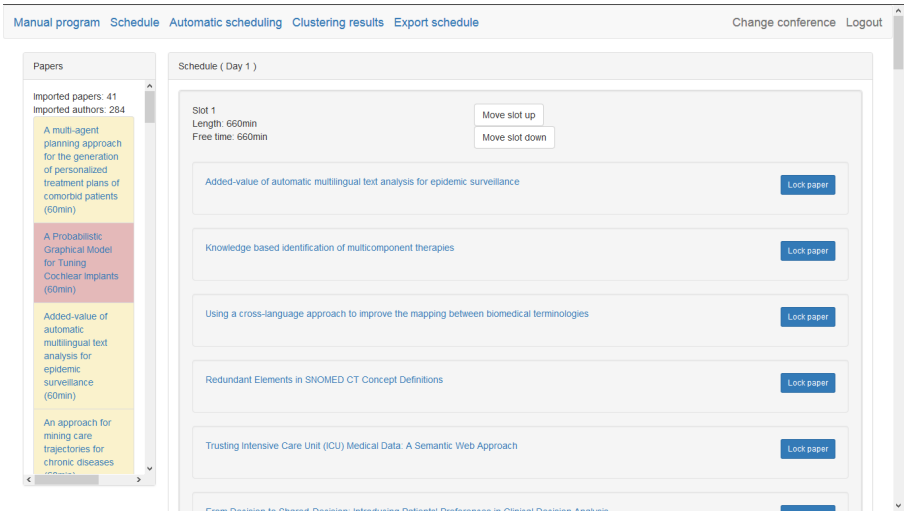


Figure 6 The main user interface of the application. Users can manually assign papers into conference slots, or they can use the application to automatically schedule them.

using the following equation:

$$s(i) = \frac{b(i) - a(i)}{max[a(i), b(i)]} .$$

A high silhouette score indicates that papers within a cluster are more similar among themselves that to papers in other clusters. The method works best on well defined clusters with large distances between clusters. The clustering methods we tested do not return such well defined clusters, since the BOW vectors used in clusterings are both highly dimensional and similar between each other, which leads to clusters that are blurred. Nevertheless, the silhouette score can still be used to compare the produced clusterings. We compared clustering papers by only using BOW vectors and by using BOW vectors fused with graph data. In both cases we repeated the clustering 50 times and averaged the silhouette score. On average, using graph data increased the silhouette score by 4.5%. This shows that additional graph based data is helpful.

5.3 Scheduling evaluation

Evaluating the quality of the automatically constructed schedule with external clustering validity measures such as adjusted Rand index or variation of information [1] can be unreliable. For each conference there exist multiple ways to construct a good schedule. This means that using the actual conference schedule as a ground truth and comparing the automatically constructed schedule with it will not necessary be objective. An automatic schedule that correctly groups together similar papers will be evaluated as inadequate if it groups them in a different way than they were grouped in the actual conference. As ground truth does not exist, similarly to other text mining tasks, e.g., automatic summarization or machine translation, we use subjective opinion of domain experts to measure the quality of (automatically) constructed schedule. Using blind expert evaluation similarity of papers scheduled to the same time slots was evaluated and papers with matching topics were counted. The process was repeated for two schedules: the actually used timetable produced manually by the

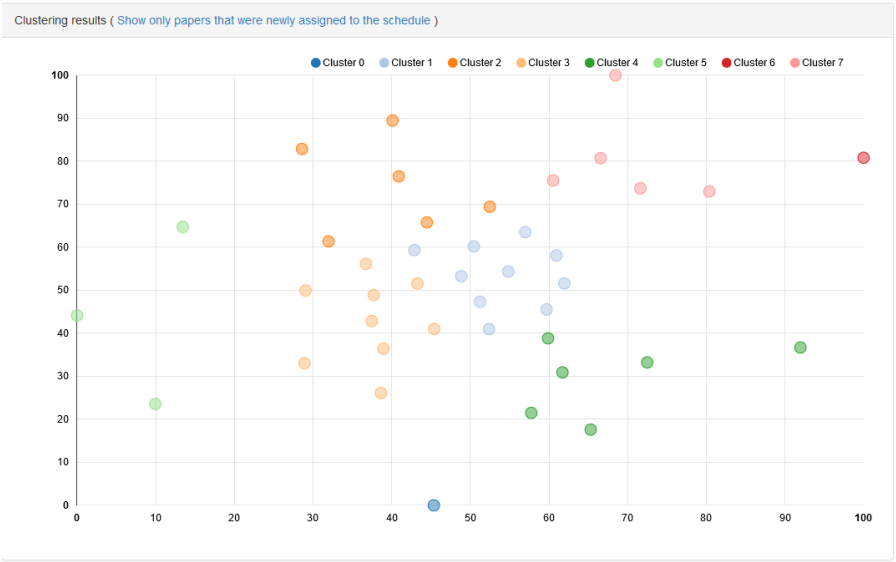


Figure 7 A visualization of clustering papers in our web application. Each cluster contains semantically similar papers and is shown with a unique color. The distance between papers is preserved in the 2-D graph using t-SNE [19]. The user can move the mouse cursor over the dots to view the papers' titles.

conference chair and the automatically constructed schedule. We compared the number of similar papers assigned to the same slot by our method to the number of similar papers in the actual conference schedule. On average, the percentage of similar papers in schedule time slots by our method was 72%, while the actual schedule had a similarity score percentage of 82%. Our method returned useful results and will be used in further improvements.

6 Conclusion

We demonstrate a methodology which is capable to enrich textual information with graph based data and utilize both in an innovative machine learning application of clustering. The proposed solution is helpful in organization of academic conferences and presents a step towards automating one of its time consuming tasks.

We implemented a method that uses data from text documents enriched with additional information extracted from reviewers' co-bidding graphs to group similar documents and assign them to a predefined conference paper presentation schedule. We converted information from reviewers' co-bidding graphs into a BOW-compatible vector using the Personalized PageRank algorithm. We fused this vector with the BOW vector describing the textual data of abstracts and titles to get the final feature vector. We used this vector with various clustering algorithms to get clusters of similar articles. The clusters were assigned to the schedule with iterative clustering implementing predefined constraints and filling in time slots of the conference program. The method is implemented as an open source web application which supports conference chairs in creating the structure of the timetable and allows automatic conference schedule construction. The web application is freely available at `https://github.com/TadejSk/conference-scheduler`.

We evaluated our method using objective and subjective evaluation. Using silhouette score we determined that using additional graph based data increased clustering performance.

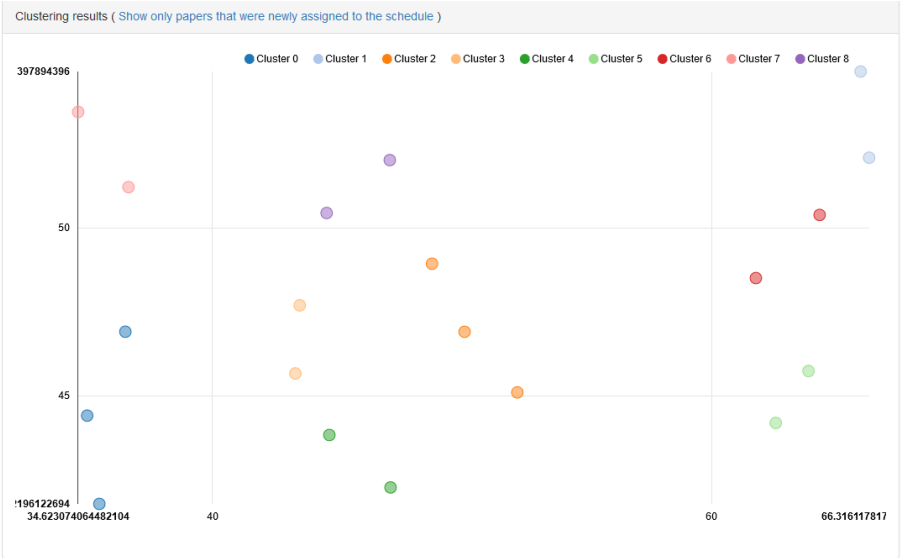


Figure 8 An example of results returned by Mean Shift clustering. There is no clear structure present in the results, and the number of returned clusters is high, with every cluster containing only three or less papers. Compared to Figure 7 the results in this figure are noticeably worse.

Subjective evaluation using an expert's opinion showed that the schedule slots returned by our method contained similar presentations.

Our approach can be further improved. The text analysis could benefit from term extraction and domain specific word weighting. Additionally, domain ontologies have been shown to be useful in semantic text mining [18] and could be used to more effectively find similar papers. The iterative constrained clustering algorithm could an also be improved with additional information extracted from whole documents and their semantic similarity. The approach also needs to be tested on larger conferences.

References

1 Olatz Arbelaitz, Ibai Gurrutxaga, Javier Muguerza, Jesús M Pérez, and Iñigo Perona. An extensive comparative study of cluster validity indices. *Pattern Recognition*, 46(1):243–256, 2013.

2 David M Blei. Probabilistic topic models. *Communications of the ACM*, 55(4):77–84, 2012.

3 Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of Knowledge Discovery and Data Mining*, volume 96, pages 226–231, 1996.

4 Brendan J Frey and Delbert Dueck. Clustering by passing messages between data points. *Science*, 315(5814):972–976, 2007.

5 Keinosuke Fukunaga and Larry D Hostetler. The estimation of the gradient of a density function, with applications in pattern recognition. *IEEE Transactions on Information Theory*, 21(1):32–40, 1975.

6 Miha Grčar, Nejc Trdin, and Nada Lavrač. A methodology for mining document-enriched heterogeneous information networks. *The Computer Journal*, 2012.

7 John A Hartigan and Manchek A Wong. Algorithm AS 136: A k-means clustering algorithm. *Applied Statistics*, pages 100–108, 1979.

8 Tin Huynh, Kiem Hoang, Loc Do, Huong Tran, Hiep Luong, and Susan Gauch. Scientific publication recommendations based on collaborative citation networks. In *International Conference on Collaboration Technologies and Systems (CTS)*, pages 316–321. IEEE, 2012.

9 Stephen C Johnson. Hierarchical clustering schemes. *Psychometrika*, 32(3):241–254, 1967.

10 Yicong Liang, Qing Li, and Tieyun Qian. Finding relevant papers based on citation relations. In *Web-age Information Management*, pages 403–414. Springer, 2011.

11 Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The PageRank Citation Ranking: Bringing Order to the Web. Technical Report 1999-66, Stanford InfoLab, Stanford, CA, November 1999.

12 Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in Python. *The Journal of Machine Learning Research*, 12:2825–2830, 2011.

13 Niels Peek, Roque Marin Morales, and Mor Peleg, editors. *Artificial Intelligence in Medicine: 14th Conference on Artificial Intelligence in Medicine, AIME 2013, Murcia, Spain*, volume 7885 of *Lecture Notes in Artificial Intelligence*. Springer, 2013.

14 Manh Cuong Pham, Dejan Kovachev, Yiwei Cao, Ghislain Manib Mbogos, and Ralf Klamma. Enhancing academic event participation with context-aware and social recommendations. In *Proceedings of IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*, pages 464–471. IEEE, 2012.

15 Peter J Rousseeuw. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, 20:53–65, 1987.

16 Gerard Salton, Anita Wong, and Chung-Shu Yang. A vector space model for automatic indexing. *Communications of the ACM*, 18(11):613–620, 1975.

17 David Sculley. Web-scale k-means clustering. In *Proceedings of the 19th International Conference on World Wide Web*, pages 1177–1178. ACM, 2010.

18 Irena Spasic, Sophia Ananiadou, John McNaught, and Anand Kumar. Text mining and ontologies in biomedicine: making sense of raw text. *Briefings in bioinformatics*, 6(3):239–251, 2005.

19 Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9(2579-2605):85, 2008.

20 Kiri Wagstaff, Claire Cardie, Seth Rogers, Stefan Schrödl, et al. Constrained k-means clustering with background knowledge. In *Proceedings of the International Conference on Machine Learning*, volume 1, pages 577–584, 2001.

21 Feng Xia, Nana Yaw Asabere, Haifeng Liu, Nakema Deonauth, and Fengqi Li. Folksonomy based socially-aware recommendation of scholarly papers for conference participants. In *Proceedings of the Companion Publication of the 23rd International Conference on World Wide Web Companion*, pages 781–786. International World Wide Web Conferences Steering Committee, 2014.

22 Feng Xia, Nana Yaw Asabere, Joel JPC Rodrigues, Filippo Basso, Nakema Deonauth, and Wei Wang. Socially-aware venue recommendation for conference participants. In *Proceedings of the 10th International Conference on Autonomic and Trusted Computing (UIC/ATC)*, pages 134–141. IEEE, 2013.

23 Shunzhi Zhu, Dingding Wang, and Tao Li. Data clustering with size constraints. *Knowledge-Based Systems*, 23(8):883–889, 2010.

A Re-Ranking Method Based on Irrelevant Documents in Ad-Hoc Retrieval

Rabeb Mbarek[1], Mohamed Tmar[2], Hawete Hattab[3], and Mohand Boughanem[4]

1 **Sfax University, Multimedia Information Systems and Advanced Computing Laboratory, Sfax, Tunisia**
rabeb.hattab@gmail.com
2 **Sfax University, Multimedia Information Systems and Advanced Computing Laboratory, Sfax, Tunisia**
mohamedtmar@yahoo.fr
3 **Umm Al-qura University, Department of Mathematics, Makkah, KSA**
hshattab@uqu.edu.sa
4 **University of Toulouse – IRIT lab France, Toulouse, France**
bougha@irit.fr

Abstract

In this paper, we propose a novel approach for document re-ranking, which relies on the concept of negative feedback represented by irrelevant documents. In a previous paper, a pseudo-relevance feedback method is introduced using an absorbing document $\widetilde{d}$ which best fits the user's need. The document $\widetilde{d}$ is orthogonal to the majority of irrelevant documents. In this paper, this document is used to re-rank the initial set of ranked documents in Ad-hoc retrieval. The evaluation carried out on a standard document collection shows the effectiveness of the proposed approach.

1998 ACM Subject Classification H.3.3 Information Search and Retrieval

Keywords and phrases Re-ranking, absorption of irrelevance, vector product

Digital Object Identifier 10.4230/OASIcs.SLATE.2016.2

1 Introduction

A commonly used strategy to improve search results is through feedback techniques, including relevance feedback [14, 15, 16], pseudo-relevance feedback [2, 5, 21] and implicit feedback [17]. A query is difficult if none of the top-ranked documents are relevant. In the case of difficult queries, if we can perform effective negative feedback when a user could not find any relevant document on the first page of the search results, we would be able to improve the ranking of the unseen results in the next few pages. It is clear that in this case of negative relevance feedback, we only have negative (i.e., irrelevant) documents. When a user is unable to reformulate an effective query (which happens often in informational queries due to insufficient knowledge about the relevant documents), negative feedback can be quite beneficial, and the benefit can be achieved without requiring extra effort from users (e.g., by assuming the skipped documents by a user to be irrelevant).

This work investigates the role of irrelevant documents in document re-ranking. In particular, our re-ranking strategy is based on a negative relevance feedback approach which takes into account irrelevant documents in the initial document ranking. The key idea behind our approach is to use the absorbing document [11], which fits the user's need and is orthogonal to the majority of irrelevant documents, to re-rank documents on the ground


© Rabeb Mbarek, Mohamed Tmar, Hawete Hattab, and Mohand Boughanem;
licensed under Creative Commons License CC-BY

of their similarity with respect to the absorbing document. Generally, standard relevance feedback methods are able to handle negative feedback by subtracting information from the original query (for example the Rocchio's model [15]). The key issue of this approach is to quantify the side effect caused by information loss. To deal with this effect, we propose a negative feedback method based on absorbing document that is able to remove only the unwanted aspects pertaining to irrelevant documents. In our approach, documents are represented as vectors in a geometric space in which similar documents are represented close to each other. This space is the classical Vector Space Model (VSM).

We compare our strategy with other approaches. First, with the Baseline Model (the BM25 model [13]). Second, with the approach of Basile et al. [3].

How to identify irrelevant documents is an open question. We use two distinct approaches in our work proposed in [3]: the former exploits documents at the bottom of the rank, while the latter takes the irrelevant documents directly from relevance judgments. These approaches are thoroughly described in Section 3.

The paper is structured as follows. Related work are briefly analyzed in Section 2. Section 3 describes the two strategies used for re-ranking. Experiments performed for evaluating our approach are presented in Section 4. The last section concludes.

2 Related Work

There exist several groups of related work in the areas of document retrieval and re-ranking.

The first category performs re-ranking by using inter-document relationship [6, 7]. The idea is to build a document which represents the ideal response to the user's information need. In [6] documents in the result list are re-weighed according to a relevance function which reflects the distance between documents and the ideal document. Other researchers use inter-document similarities to combine several retrieved lists (see for example [7]). In this case, the idea of similarity is used to give support to documents with similar content highly ranked across multiple result lists.

A second category of work is related to recent advances in structural re-ranking paradigm over graphs. In the language modeling framework, the traditional cluster-based retrieval has been juxtaposed with document language model smoothing in which document representation incorporates cluster-related information [8, 9, 10].

An early attempt to model terms negation in pseudo-relevance feedback by quantum logic operators is due to Widdows [20]. In his work, Widdows has shown that negation in quantum logic is able to remove, from the result set, not only unwanted terms but also their related meaning. The concept of vectors orthogonality is exploited to express queries like *Retrieve documents that contain term A & NOT term B*. Widdows suggested that vectors which represent unrelated concepts should be orthogonal to each other. Indeed, orthogonality prevents vectors from sharing common features.

In [3], Basile et al. proposed a new re-ranking strategy based on a pseudo-relevance feedback approach which took into account both relevant and irrelevant documents in the initial document ranking. The key idea of this approach is to build an ideal document which fits the user's need, and then re-rank documents on the ground of their similarity with respect to the ideal document. The ideal document d^* is built using a geometrical space where d^* is computed as a vector close to relevant documents and unrelated to irrelevant ones. In this space the concept of relevance is expressed in terms of similarity, while the concept of irrelevance is defined by orthogonality (similarity equals to zero). Formally, Basile et al. [3]

computed the ideal document by the following logical operation:

$$d^* = d_1^+ \vee d_2^+ \vee ...d_n^+ \wedge NOT(d_1^- \vee d_2^- \vee ... \vee d_m^-) \quad (1)$$

where $D^+ = \{d_1^+, d_2^+, ..., d_n^+\}$ and $D^- = \{d_1^-, d_2^-, ..., d_m^-\}$ are the subsets of relevant and irrelevant documents respectively. Equation 1 consists in computing a vector which represents the disjunction of the documents in D^+, and then projecting this vector onto the orthogonal spaces generated by the documents in D^-. Disjunction and negation using quantum logic are thoroughly described in [20]. An overview of Quantum Mechanics for Information Retrieval can be found in [4]. The main problem of the approach of Basile et al. is the query drift problem related to the pseudo-relevance feedback approach. Query drift occurs when the documents used for relevance feedback contain few or no relevant documents.

In this paper the orthogonality is defined using the algebraic operator vector product[1]. Using this operator, we build an absorbing document which is orthogonal to the majority of irrelevant documents.

The idea to build a document which represents the response to the user's information need is not new. In [6] documents in the result list are re-weighed according to a relevance function which reflects the distance between documents and the "ideal document".

Whilst relevant documents have been successfully used in several approaches to improve Information Retrieval performance, irrelevant ones seem not to arouse researchers' interest. Singhal et al. [18] achieved an interesting result for the learning routing query problem: they showed that using irrelevant documents close to the query, in place of those in the whole collection, is more effective. Rocchio's original formulation explicitly includes a component of irrelevant documents [15]. In [12, 11], the authors showed that irrelevant documents can be used to extract better expansion terms from the top-ranking k documents. A successful use of irrelevant documents for negative pseudo-relevance feedback has been carried out in [19], where authors point out the effectiveness of their approach with poorly performing queries.

3 A Re-ranking Method Based on Irrelevant Documents

This section describes our re-ranking strategy based on irrelevant documents. The main idea is to build a document vector which attempts to model the absorbing document in response to a user query, and then exploit this vector to re-rank the initial set of ranked documents D_{init}. The absorbing document $\widetilde{d}$ should be orthogonal with each document in the set D^- of irrelevant ones. Identifying relevant documents is quite straightforward: we assume the top ranked documents in D_{init} as relevant, whereas identifying non-relevant documents is not trivial. To this purpose, we propose two strategies: the former relies on documents at the bottom of D_{init}, while the latter needs relevance judgments. The absorbing document vector $\widetilde{d}$ is exploited to re-rank documents in D_{init} on the ground of the similarity between $\widetilde{d}$ and each document in D_{init} in the Euclidean space (vector space equipped with an inner product).

3.1 Vector product

Let E be a vector space of dimension n and let u_1,..., u_{n-1} be $n-1$ vectors of E. For each vector x of E there exists a unique vector w such that:

$$det(u_1, ..., u_{n-1}, x) = w^T.x$$

[1] This operator, in a vector space, naturally models the orthogonality.

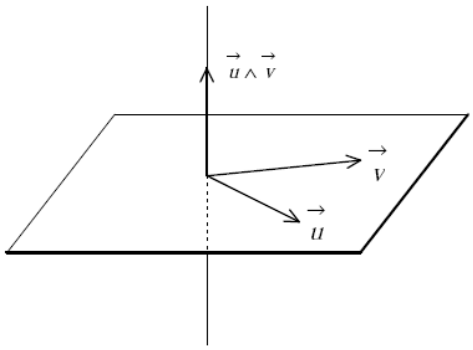


Figure 1 The cross product for $n = 3$.

where *det* is the determinant of n vectors, w^T is the transpose of w and $w^T.x$ is the classical inner product.

w is called *the vector product of* u_1,..., u_{n-1} and is denoted by $u_1 \wedge ... \wedge u_{n-1}$ (for $n = 3$, see Figure 1). We have the following properties:

- the vector $u_1 \wedge ... \wedge u_{n-1}$ is orthogonal to each vector u_i.
- the vector $u_1 \wedge ... \wedge u_{n-1}$ is orthogonal to the subspace F of E generated by the family (u_1,..., u_{n-1}). Indeed, if u is a vector of F, there exists $n-1$ scalars α_1,..., α_{n-1} such that $u = \alpha_1 u_1 + ... + \alpha_n u_{n-1}$.
- $u_1 \wedge ... \wedge u_{n-1} = \overrightarrow{0}$ if and only if u_1,..., u_{n-1} are dependent.
- if u_1,..., u_{n-1} are independent then (u_1,..., u_{n-1}, $u_1 \wedge ... \wedge u_{n-1}$) is a basis of E.

3.2 Scenario

Let n be the dimension of D_{init} as a vector space, n represents the number of indexing terms. Let $m < n$ be the number of linearly independent and representative documents of D^-, and let u_1,...,u_m be these irrelevant ones. We eliminate $n - m - 1$ terms and so the dimension becomes $m + 1$. The *absorbing document* is:

$$\widetilde{d} = u_1 \wedge ... \wedge u_m \tag{2}$$

This document is orthogonal to the majority of irrelevant documents.

3.3 Compute of the absorbing document

To compute $\widetilde{d}$ it suffices to compute the vector product of u_1,...,u_m. Let $A = (u_1, ..., u_m)$ be the matrix of $m + 1$ rows and m columns. Let A_i be the matrix obtained from the matrix A by deleting the ith row ($1 \leq i \leq m + 1$). The vector product of U_1,...., U_m is the vector:

$$u_1 \wedge ... \wedge u_m = \begin{pmatrix} detA_1 \\ -detA_2 \\ ... \\ ... \\ (-1)^m detA_{m+1} \end{pmatrix} \tag{3}$$

The Equation 3 generalizes the definition of vector product of two vectors in dimension 3. In the following, we give an example of vector product of three vectors in dimension 4:

if $u_1 = (1, 0, 1, -1)^T$, $u_2 = (0, 2, 1, 1)^T$ and $u_3 = (1, 3, 1, 0)^T$ are three vectors, then $u_1 \wedge u_2 \wedge u_3 = (4, -1, -1, 3)^T$ and so $(4, -1, -1, 3).(1, 0, 1, -1)^T = (4, -1, -1, 3).(0, 2, 1, 1)^T = (4, -1, -1, 3).(1, 3, 1, 0)^T = 0$.

3.4 An illustrative example

In this example we show how the absorbing document $\widetilde{d}$ help us to extract better expansion terms.

We consider four linearly independent irrelevant documents d_1, d_2, d_3, and d_4, selected from the bottom of the initial ranking of topic 351. These four irrelevant documents indexed by 5 expansion terms t_1, t_2, t_3, t_4 and t_5, selected from the 2-top relevant documents.

$$d_1 = (2, 1, 1, 0, 0)^T \quad d_2 = (1, 0, 2, 0, 0)^T \quad d_3 = (4, 0, 2, 0, 0)^T \quad d_4 = (0, 1, 0, 2, 1)^T .$$

The absorbing document $\widetilde{d}$ is the cross product of d_1, d_2, d_3, and d_4:

$$\widetilde{d} = (2, 1, 1, 0, 0)^T \wedge (1, 0, 2, 0, 0)^T \wedge (4, 0, 2, 0, 0)^T \wedge (0, 1, 0, 2, 1)^T = (0, 0, 0, -6, 12)^T .$$

$\widetilde{d}$ is indexed by the terms t_4 and t_5. Note that d_4 is the only irrelevant document which is indexed by t_4 and t_5.

3.5 Strategies to select irrelevant documents

We use the two strategies proposed in [3] to select the set (D^-) of irrelevant documents:

- BOTTOM, which selects the irrelevant documents from the bottom of the rank; in other words we assume that the user selects the last m linearly independent irrelevant documents;
- RELJUD, which relies on relevance judgments provided by CLEF organizers. This technique selects the top m ranked documents which are irrelevant exploiting the relevance judgments. We use this strategy to simulate the user's explicit feedback; in other words we assume that the user selects the first m linearly independent irrelevant documents.

To select linearly independent irrelevant documents we use the Algorithm 1.

4 Experiments

In this section we give the different experiments and results obtained to evaluate our approach. The goal of the evaluation is to prove that our re-ranking strategy, which relies on the concept of negative feedback represented by irrelevant documents, improves retrieval performance and outperforms other methods. Moreover, we want to evaluate the performance of the BOTTOM strategy and RELJUD strategy.

4.1 Environnement

We set up a baseline system based on the BM25 multi-fields model [13]. The evaluation has been designed using the CLEF 2009 Ad-hoc WSD Robust Task collection [1]. The Robust task allows us to evaluate Information Retrieval System performance even when difficult queries are involved. The CLEF 2009 collection consists of 166, 717 documents which have two fields: HEADLINE and TEXT. Table 1 shows the BM25 parameters, where b is a constant related to the field length, k_1 is a free parameter, and boost is the boosting factor applied to that field.

Listing 1 The set of linearly independent irrelevant documents.

```
Let n be the number of terms
Let A be the n × mm matrix of irrelevant documents
Let m be the rank of A
Let B be the n × m matrix of linearly independent irrelevant documents

for i = 1, ..., n
     b_{i,1} ← a_{i,1}
end for
k ← 1
for j = 2, ..., mm
     Let C be a vector
     for i = 1, ..., n
          c_i ← a_{i,j}
     end for
     for l = 1, ..., n
          b_{l,k} ← c_l
     end for

     Let p be the rank of B
     if p = (k + 1)
          k ← k + 1
     end if
     if k ← n
          break
     end if
end for

return B
```

Table 1 BM25 parameters used in the experiments.

Field	k_1	b	boost
HEADLINE	3.25	0.7	2
TEXT	3.25	0.7	1

In detail, the CLEF 2009 collection has 150 topics. Topics are structured in three fields: TITLE, DESCRIPTION and NARRATIVE. We used only TITLE and DESCRIPTION, because NARRATIVE field is the topic description used by assessors. Moreover, we used different boosting factors for each topic field (TITLE=4 and DESCRIPTION=1) to highlight terms in the TITLE.

For our approach, the experiments consist to re-rank documents (results of the baseline approach) on the ground of their similarity with respect to the absorbing document $\widetilde{d}$ (Equation 2). The retrieved documents are ranked by the inner product done by:

$$< \widetilde{d}, d >= \widetilde{d}^T.d \tag{4}$$

To evaluate the performance of our approach, we executed several runs using the topics provided by CLEF organizers. In particular, we took into account: m (the cardinality of D^-). We selected different ranges for parameter m: $[1, 5, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100]$.

For the approach of [3], the experiments consist to re-rank documents (results of the baseline approach) on the ground of their similarity with respect to the ideal document d^*

(Equation 1). The retrieved documents are ranked by the relevance score computed for each document d in D_{init} done by:

$$S(d) = \alpha * S_{D_{init}}(d) + (1 - \alpha).sim(d, d^*)$$

where $S_{D_{init}}(d)$ is the score of d in the initial rank D_{init}, while $sim(d, d^*)$ is the similarity degree between the document vector d and the ideal document vector d^* computed by cosine similarity.

To evaluate the performance of their approach, Basile et al. [3] executed several runs using the topics provided by CLEF organizers. In particular, they took into account: n (the cardinality of D^+), m (the cardinality of D^-) and the parameter α used for the linear combination of the scores. They selected different ranges for each parameter: n ranges in $[1, 5, 10, 20, 40]$, m ranges in $[0, 1, 5, 10, 20, 40]$, while α ranges in $[0.3, 0.4, 0.5, 0.6, 0.7]$. Table 2. shows the best five runs for BOTTOM and RELJUD strategies with respect to MAP and GMAP. For the both approaches, they set the cardinality of D_{init} to 1000. All the metrics have been computed on the first 1000 returned documents, as prescribed by the CLEF evaluation campaign.

4.2 Results

The experiments and the evaluations are as follow. Comparison between the Baseline Model (the BM25 multi-fields model [13]), the approach of Basile et al. [3], and our approach: re-ranking method using absorbing document (Equation 4), using Mean Average Precision (MAP) and Geometric Mean Average Precision (GMAP) over all the queries.

The results have been grouped by the number of irrelevant documents. Table 2 reports the results of the Baseline Model and the best performance obtained for the approach of Basile et al. [3] (the best five runs for BOTTOM and RELJUD strategies with respect to MAP values). Moreover, this table illustrates the best performance obtained for our approach (the best five runs for BOTTOM and RELJUD strategies where the number of irrelevant documents ranges in $[1, 5, 10, 20, 30, 40, 50, 60, 70]$). Improvements in percentage $\Delta\%$ with respect to the baseline are reported for MAP and GMAP values.

4.3 Analysis of results

Generally, BOTTOM strategy results are not significant improvements. This suggests that the BOTTOM strategy is not able to identify irrelevant documents. For this strategy, the highest MAP value for our approach is 0.476 (GMAP=0.234). Both values (MAP and GMAP) are obtained with 30 irrelevant documents. For the approach of Basile et al., The highest MAP value is 0.4384 (GMAP=0.1928). The MAP value is obtained with five irrelevant documents, while the GMAP is obtained with one irrelevant document.

The method RELJUD obtains very high results. For this strategy, The highest MAP value for our approach is 0.691 (GMAP=0.3328). Both values (MAP and GMAP) are obtained with 70 irrelevant documents. For the approach of Basile et al., The highest MAP value is 0.6649 (GMAP=0.3240). Both values (MAP and GMAP) are obtained with 40 irrelevant documents

For our approach, the performance of the two strategies (BOTTOM and RELJUD) increases if the number of irrelevant documents increases.

The experimental results are very encouraging. For our approach, both methods (BOTTOM and RELJUD) show improvements with respect to the baseline in all the approaches. The comparison between the results of our approach with the use of the two strategies

Table 2 Comparison between our approach, the baseline, and the approach of Basile et al.

Approach	Method	Run	n m α	MAP	Δ%	GMAP	Δ%
-	-	baseline	- - -	0.4139	-	0.1846	-
Basile et al.	BOTTOM	2.B1	1 5 0.6	0.4384	+5.92	0.1923	+4.17
		2.B2	1 10 0.6	0.4379	+5.80	0.1921	+4.06
		2.B3	1 1 0.5	0.4377	+5.75	0.1928	+4.44
		2.B4	1 5 0.5	0.4376	+5.73	0.1926	+4.33
		2.B5	1 20 0.6	0.4372	+5.73	0.1917	+3.85
	RELJUD	2.R1	40 40 0.7	0.6649	+60.64	0.3240	+75.51
		2.R2	40 40 0.6	0.6470	+56.32	0.3156	+70.96
		2.R3	40 40 0.5	0.6223	+50.35	0.3124	+69.23
		2.R4	20 40 0.7	0.6176	+49.21	0.2859	+54.88
		2.R5	20 20 0.7	0.6107	+47.55	0.2836	+53.63
Our approach	BOTTOM	B1	- 1 -	0.4	−3.36	0.17	−7.9
		B2	- 5 -	0.419	+1.23	0.185	+0.21
		B3	- 10 -	0.423	+2.2	0.191	+3.46
		B4	- 20 -	0.442	+6.78	0.212	+14.84
		B5	- 30 -	0.476	+15	0.234	+25.89
	RELJUD	R1	- 20 -	0.601	+45.2	0.272	+47.34
		R2	- 40 -	0.671	+62.11	0.331	+79.3
		R3	- 50 -	0.675	+63.08	0.3325	+80.11
		R4	- 60 -	0.687	+65.98	0.3327	+80.22
		R5	- 70 -	0.691	+66.94	0.3328	+80.28

(BOTTOM and RELJUD), the results of the classic BM25 model, and the results of Basile et al., shows that our approach improves the results of the two other approaches.

5 Conclusion and future work

This paper proposes a novel approach based on negative evidence for document re-ranking. The novelty lies on the use of the absorbing document to capture the negative aspects of irrelevant documents. This method has shown its effectiveness with respect to a baseline system based on BM25 and a re-ranking method based on the approach of Basile et al. [3]. Moreover, the evaluation has proved the robustness of the proposed strategy and its capability to absorb irrelevant documents. On the other hand our approach depends on a single parameter, while the other re-ranking approaches depend on many parameters. Moreover, the absorbing document is modelled by a vector product which is simply computed in a vector space model.

In a future work, we will apply this re-ranking approach with respect to a vector space basis which optimally separates relevant and irrelevant documents.

References

1 Eneko Agirre, Giorgio Maria Di Nunzio, Thomas Mandl, and Arantxa Otegi. CLEF 2009 ad hoc track overview: Robust-WSD task. In Carol Peters, Giorgio Maria Di Nunzio, Mikko Kurimo, Thomas Mandl, Djamel Mostefa, Anselmo Peñas, and Giovanna Roda, editors,

Multilingual Information Access Evaluation I, pages 36–49. Springer, Berlin, Heidelberg, 2010. doi:10.1007/978-3-642-15754-7_3.

2 Rony Attar, Fraenkel, and Aviezri Siegmund. Local feedback in full-text retrieval systems. *Journal of the ACM*, 24(3):397–417, July 1977. doi:10.1145/322017.322021.

3 Pierpaolo Basile, Annalina Caputo, and Giovanni Semeraro. Negation for document re-ranking in ad-hoc retrieval. In Giambattista Amati and Fabio Crestani, editors, *Advances in Information Retrieval Theory*, pages 285–296. Springer, Berlin, Heidelberg, 2011. doi:10.1007/978-3-642-23318-0_26.

4 Garrett Birkhoff and John Von Neumann. The logic of quantum mechanics. *Annals of Mathematics*, 37(4):823–843, 1936.

5 Chris Buckley, Gerard Salton, James Allan, and Amit Singhal. Automatic query expansion using SMART: TREC 3. In *In Proceedings of The third Text REtrieval Conference (TREC-3)*, pages 69–80, 1994.

6 Czesław Daniłowicz and Jarosław Baliński. Document ranking based upon Markov chains. *Information Processing and Management*, 37(4):623–637, July 2001. doi:10.1016/S0306-4573(00)00038-8.

7 Anna Khudyak Kozorovitzky and Oren Kurland. From "identical" to "similar": Fusing retrieved lists based on inter-document similarities. In Leif Azzopardi, Gabriella Kazai, Stephen Robertson, Stefan Rüger, Milad Shokouhi, Dawei Song, and Emine Yilmaz, editors, *Advances in Information Retrieval Theory*, pages 212–223. Springer, Berlin, Heidelberg, 2009. doi:10.1007/978-3-642-04417-5_19.

8 Oren Kurland. Re-ranking search results using language models of query-specific clusters. *Information Retrieval*, 12(4):437–460, 2009. doi:10.1007/s10791-008-9065-9.

9 Oren Kurland and Lillian Lee. Corpus structure, language models, and ad hoc information retrieval. In *Proceedings of the 27th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 194–201, New York, NY, USA, 2004. ACM. doi:10.1145/1008992.1009027.

10 Xiaoyong Liu and W. Bruce Croft. Cluster-based retrieval using language models. In *Proceedings of the 27th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 186–193, New York, NY, USA, 2004. ACM. doi:10.1145/1008992.1009026.

11 Rabeb Mbarek, Mohamed, Hawete Hattab, and Mohand Boughanem. Pseudo-relevance feedback method based on the cross-product of irrelevant documents. *International Journal Web Applications*, 8(1):8–16, March 2016.

12 Karthik Raman, Raghavendra Udupa, Pushpak Bhattacharya, and Abhijit Bhole. On improving pseudo-relevance feedback using pseudo-irrelevant documents. In *Proceedings of the 32nd European Conference on Advances in Information Retrieval*, pages 573–576, Berlin, Heidelberg, 2010. Springer-Verlag. doi:10.1007/978-3-642-12275-0_50.

13 Stephen Robertson, Hugo Zaragoza, and Michael Taylor. Simple BM25 extension to multiple weighted fields. In *Proceedings of the Thirteenth ACM International Conference on Information and Knowledge Management*, pages 42–49, New York, NY, USA, 2004. ACM. doi:10.1145/1031171.1031181.

14 Stephen E. Robertson and Karen Sparck Jones. Relevance weighting of search terms. *Journal of the American Society for Information Science*, 27(3):129–146, 1976. doi:10.1002/asi.4630270302.

15 Joseph J. Rocchio. Relevance feedback in information retrieval. In Gerard Salton, editor, *The SMART retrieval system - experiments in automatic document processing*, pages 313–323. Englewood Cliffs, NJ: Prentice-Hall, 1971.

16 Gerard Salton and Chris Buckley. Improving retrieval performance by relevance feedback. In Karen Sparck Jones and Peter Willett, editors, *Readings in Information Retrieval*, pages 355–364. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1997.

17 Xuehua Shen, Bin Tan, and ChengXiang Zhai. Context-sensitive information retrieval using implicit feedback. In *Proceedings of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 43–50, New York, NY, USA, 2005. ACM. doi:10.1145/1076034.1076045.

18 Amit Singhal, Mandar Mitra, and Chris Buckley. Learning routing queries in a query zone. In *Proceedings of the 20th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 25–32, New York, NY, USA, 1997. ACM. doi:10.1145/258525.258530.

19 Xuanhui Wang, Hui Fang, and ChengXiang Zhai. A study of methods for negative relevance feedback. In *Proceedings of the 31st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 219–226, New York, NY, USA, 2008. ACM. doi:10.1145/1390334.1390374.

20 Dominic Widdows. Orthogonal negation in vector spaces for modelling word-meanings and document retrieval. In *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics*, volume 1, pages 136–143, Stroudsburg, PA, USA, 2003. Association for Computational Linguistics. doi:10.3115/1075096.1075114.

21 Jinxi Xu and W. Bruce Croft. Query expansion using local and global document analysis. In *Proceedings of the 19th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 4–11, New York, NY, USA, 1996. ACM. doi:10.1145/243199.243202.

Comparing the Performance of Different NLP Toolkits in Formal and Social Media Text*

Alexandre Pinto[1], Hugo Gonçalo Oliveira[2], and Ana Oliveira Alves[3]

1 **CISUC, Dept. of Informatics Engineering, University of Coimbra, Coimbra, Portugal**
arpinto@student.dei.uc.pt
2 **CISUC, Dept. of Informatics Engineering, University of Coimbra, Coimbra, Portugal**
hroliv@dei.uc.pt
3 **CISUC, Dept. of Informatics Engineering, University of Coimbra, Coimbra, Portugal; and**
Polythecnic Institute of Coimbra, Coimbra, Portugal
aalves@isec.pt, ana@dei.uc.pt

Abstract

Nowadays, there are many toolkits available for performing common natural language processing tasks, which enable the development of more powerful applications without having to start from scratch. In fact, for English, there is no need to develop tools such as tokenizers, part-of-speech (POS) taggers, chunkers or named entity recognizers (NER). The current challenge is to select which one to use, out of the range of available tools. This choice may depend on several aspects, including the kind and source of text, where the level, formal or informal, may influence the performance of such tools. In this paper, we assess a range of natural language processing toolkits with their default configuration, while performing a set of standard tasks (e.g. tokenization, POS tagging, chunking and NER), in popular datasets that cover newspaper and social network text. The obtained results are analyzed and, while we could not decide on a single toolkit, this exercise was very helpful to narrow our choice.

1998 ACM Subject Classification I.2.7 Natural Language Processing

Keywords and phrases Natural language processing, toolkits, formal text, social media, benchmark

Digital Object Identifier 10.4230/OASIcs.SLATE.2016.3

1 Introduction

The Web is a large source of data, mostly expressed in natural language text. Natural language processing (NLP) systems need to understand the human languages in order to extract new knowledge and perform diverse tasks, such as information retrieval, machine translation, or text classification, among others. For widely-spoken languages, such as English, there is currently a wide range of NLP toolkits available for performing lower-level NLP tasks, including tokenization, part-of-speech (POS) tagging, chunking or named entity recognition (NER). This enables that more complex applications do not have to be developed

* This work was supported by National Funds through the FCT – Fundação para a Ciência e a Tecnologia (Portuguese Foundation for Science and Technology) – within project REMINDS – UTAP-ICDT/EEI-CTP/0022/2014.

© Alexandre Pinto, Hugo Gonçalo Oliveira, and Ana Oliveira Alves;
licensed under Creative Commons License CC-BY

completely from scratch. Yet, with the availability of many such toolkits, the one to use is rarely obvious. Users have also to select the most suitable set of tools that meets their specific purpose. Among other aspects, the selection may consider the community of users, frequency of new versions and updates, support, portability, cost of integration, programming language, the number of covered tasks, and, of course, their performance. During the previous process of selection, the authors of this paper ended up comparing a wide range of tools, in different tasks and kinds of text. This paper reports the comparison of well-known NLP toolkits and their performance in four common NLP tasks – tokenization, POS tagging, chunking and NER – in two different kinds of text – newspaper text, typically more formal, and social network text, often less formal. Although the majority of the tested tools could be trained with specific corpora and / or for a specific purpose, we focused on comparing the performance of their default configuration, which means that we used the available pre-trained models for each tool and target task. This situation is especially common for users that either do not have experience, time or available data for training the tools for a specific purpose. Besides helping us to support our decision, we believe that this comparison will be helpful for other developers and researchers in need of making a similar selection.

The remainder of this paper starts with a brief reference on previous work. After that, the tasks where the toolkits were compared are enumerated, which is followed by the description of the datasets used as benchmarks, all of them previously used in other evaluations. The measures used for comparison are then presented, right before its results are reported and discussed. Although there was not a toolkit that outperformed the others in all the tested tasks and kinds of text, this analysis revealed to be very useful, as it narrowed the range of possible choices and lead to our current selection.

2 Related Work

In academic, official or business contexts, written documents typically use formal language. This means that syntactic rules and linguistic conventions are strictly followed. On the other hand, although typically used orally, informal language has become frequent in written short messages or posts in social networks, such as Facebook or Twitter. In opposition to news websites, where posts are more elaborated, complex and with a higher degree of correctness, in text posted in social networks, it is common to find shorter and simpler sentences that tend to break some linguistic conventions (e.g. proper nouns are not always capitalized, or punctuation is not used properly), make an intensive use of abbreviations, and where slang and spelling mistakes are common. For instance, in informal English, it is common to use colloquial expressions (e.g. “look blue”, “go bananas”, “funny wagon”), contractions (e.g. “ain’t”, “gonna”, “wanna”, “y’all”), clichés (e.g. “An oldie, but a goodie”, “And they all lived happily ever after”), slang (e.g. “gobsmacked”, “knackered”), abbreviations (e.g. “lol”, “rofl”, “ty”, “afaik”, “asap”, “diy”, “rsvp”); the first and the second person, imperative (e.g. “Do it!”) and usually active voices, in addition to the third person and the passive voice, which are generally the only in formal text. Informal language poses an additional challenge for NLP tools, most of which developed with formal text on mind and significantly dependent on the quality of the written text. Given the huge amounts of data transmitted everyday in social networks, the challenge of processing messages written in informal language has received much attention in the later years. In fact, similarly to well-known NLP shared tasks based on corpora written in formal language, including the CoNLL-2000, 2002 or 2003 shared evaluation tasks[25] , tasks using informal text have also been organized, including, for

instance, the Making Sense of Microposts Workshop (MSM 2013)[1] or tasks included in the SemEval workshops (e.g. Sentiment Analysis from Twitter [23]).

González [13] highlights the particular characteristics of Twitter messages that make common NLP tasks challenging, such as irregular grammatical structure, language variants and styles, out-of-vocabulary words or onomatopeias, reminding the fact that there is still a lack of gold standards regarding colloquial texts, especially for less-resourced languages.

Besides comparing different NLP tools, in this work, we also analyze their performance in different types of text, some more formal, from newspapers, and some less formal, from Twitter. Other comparisons have been made by others, including the following. In order to combine different NER tools and improve recall, Dlugolinský et al. [7] assessed selected tools for this task in the dataset of the MSM2013 task. This included the comparison of well-known tools such as ANNIE[2], OpenNLP[3], Illinois Named Entity Tagger[4] and Wikifier[5], OpenCalais[6], Stanford Named Entity Tagger[7] and Wikipedia Miner[8].

Godin et al. [12] also used the MSM2013 challenge corpus and performed similar evaluations oriented to NER web services, such as AlchemyAPI[9], DBpedia Spotlight[10], OpenCalais, and Zemanta[11]. Since the evaluated services use complex ontologies, a mapping between the obtained ontologies and entity types was performed, with good F_1 scores when using AlchemyAPI for the person (78%) and location (74%) type entities, and OpenCalais for the organization (55%) and miscellaneous (31%) entities. Rizzo et al. [20] also evaluated web services, such as Lupedia[12], Saplo[13], Wikimeta[14] and Yahoo Content Analysis (YCa), but with focus on different kinds of well-formed content and varying length, such as TED talks transcripts, New York Times articles and abstracts from research papers. In fact, they evaluated the resulting NER and Disambiguation (NERD) framework, which unified the output results of the aforementioned web services, supporting the fact that tools such as AlchemyAPI, OpenCalais and additionally DBpedia Spotlight perform well in well-formed contents, using formal language. Rizzo et al. also report on the evaluation of datasets with colloquial text, namely Twitter text from the MSM2013 challenge and newspaper text from the CoNLL-2003 Reuter Corpus [21]. They report better NER results when using a combination of the tested tools, achieving F_1 results greater than 80% on he CoNLL-2003 dataset, for all entity types and F_1 results greater than 50% on the MSM-2013 dataset, except for the miscellaneous type that obtained results less than 30%.

Garcia and Gamallo [10] report the development of a multilingual NLP pipeline. To assess the performance of the presented tool, they performed experiments with POS-tagging and NER. The POS-tagger performed slightly better than well-known tools such as OpenNLP and Stanford NER, achieving a precision score of 94% on the Brown Corpus. On the other

[1] http://microposts2016.seas.upenn.edu
[2] https://gate.ac.uk/sale/tao/splitch6.html#chap:annie
[3] https://opennlp.apache.org
[4] https://cogcomp.cs.illinois.edu/page/software_view/NETagger
[5] https://cogcomp.cs.illinois.edu/page/software_view/Wikifier
[6] http://www.opencalais.com
[7] http://nlp.stanford.edu/software/CRF-NER.shtml
[8] http://wikipedia-miner.cms.waikato.ac.nz
[9] http://www.alchemyapi.com
[10] https://github.com/dbpedia-spotlight/dbpedia-spotlight
[11] http://www.zemanta.com
[12] http://dbpedia.org/projects/lupedia-enrichment-service
[13] http://saplo.com
[14] https://www.w3.org/2001/sw/wiki/Wikimeta

hand, the NER module achieved F_1 scores of 76% and 59% on the IEER[15] and SemCor[16] Corpus, respectively.

Rodriquez et al. [22] and Atdag and Labatut [1] compared different NER tools applied to different kinds of text, respectively biographical and OCR texts. Rodriquez et al. used Stanford CoreNLP, Illinois NER, LingPipe and OpenCalais, on a set of Wikipedia biographic articles annotated with person, location, organization and date type entities. Due to the absence of biography datasets, the evaluated corpus was fully designed by the authors, i.e, the evaluated corpus consisted of a series of Wikipedia articles which were annotated with the aforementioned entity types. Although CoreNLP obtained the best F_1 scores (60% and 44%) in two manually-annotated resources, there was not a tool that outperformed all the others in every entity type. They are rather complementary. Atdag and Labatut evaluated OpenNLP, Stanford CoreNLP, AlchemyAPI and OpenCalais using datasets with the entity types person, location and organization manually annotated. They used data from the Wiener Library, London and King's College London's Serving Soldier archive, which consisted of Holocaust survivor testimonies and newsletters written for the crew of H.M.S. Kelly in 1939. Once again, Stanford CoreNLP gave the best overall F_1 results (90%) while OpenCalais only achieved 73%.

3 Addressed Tasks

In order to evaluate how good standard NLP tools perform against different kinds of text, such as noisy text from social networks and formal text from newspapers, we performed a set of experiments where the performance in common NLP tasks was analysed. The addressed tasks were tokenization, POS-tagging, chunking and NER. The following list describes the four evaluated tasks:

- *Tokenization:* usually the first step in NLP pipelines. It is the process of breaking down sentences into tokens, which can be words or punctuation marks. Although it seems a relatively easy task, it has some issues because some words may rise doubts on how they should be tokenized, namely words with apostrophes, or with mixed symbols.
- *Part-of-Speech (POS) Tagging:* given a specific tagset, it determines the part-of-speech of each token in a sentence. In this work, the tags of the Penn Treebank Project [17], popular among the NLP community, are used.
- *Chunking*: also known as shallow parsing, it is a lighter syntactic parsing task. The main purpose is to identify the constituent groups in which the words are organized. This includes at least noun phrases (NP), verb phrases (VP) and prepositional phrases (PP). The sequence of chunks forms the entire sentence. They may also be nested inside each other to form a tree structure, where each leaf is a word, the previous node is the corresponding POS-tag and the head of the tree is the chunk type.
- *Name Entity Recognition/Classification*: deals with the identification of certain types of entities in a text and may go further classifying them into one of given categories, typically PERson, LOCations, ORGanizations, all proper nouns, and sometimes others, such as dates.

These are common NLP tasks, the first step of several more complex NLP applications, and supported by several NLP toolkits for English, including those compared in this work.

[15] `http://www.itl.nist.gov/iad/894.01/tests/ie-er/er_99/er_99.htm`
[16] `http://www.gabormelli.com/RKB/SemCor_Corpus`

Listing 1 Example of the Annotated Data Format.

```
Token         POS       Syntactic Chunk      Named Entity
Only           RB          B-NP                 O
France         NNP         I-NP                 LOC
and            CC          I-NP                 O
Britain        NNP         I-NP                 LOC
backed         VBD         B-VP                 O
Fischler       NNP         B-NP                 PER
's             POS         B-NP                 O
proposal       NN          I-NP                 O
.              .           O                    O
```

4 Used Datasets

In order to evaluate the performance of the different NLP toolkits and determine the best performing ones, the same criteria must be followed, including the same metrics and manually-annotated gold standard data. Testing tools in the same tasks and scenarios makes comparison fair and more reliable. For this purpose, we relied on well-known datasets widely used in NLP and text classification research, not only in the evaluation of NLP tools, but also for training new models. More precisely, we used different gold standard datasets that cover different kinds of text – newspaper and social media. Regarding newspaper text, we used a collection of news wire articles from the Reuters Corpus[17], previously used in the shared task of the 2003 edition of the CoNLL conference. The POS and chunking annotations of this dataset were obtained using a memory-based MBT tagger [5]. The named entities were manually annotated at the University of Antwerp [25].

In order to represent social and more informal text, we first used the annotated data from Alan Ritter's Twitter corpus[18], with manually tokenized, POS-tagged and chunked Twitter posts, also with annotated named entities. The collection of Twitter posts used in the MSM 2013 workshop[19], where named entities are annotated, was also used as a gold standard for social media text.

The POS tags of the CoNLL-2003 dataset follow the Penn Treebank style [20]. Alan Ritter's corpus follows the same format, with the same POS-tags and additional specific tags for retweets, @usernames, #hashtags, and urls. For the chunk tags, the format I|O|B-TYPE is used in both datasets. This is interpreted as: the token is inside (I), in the beginning (B) of a following chunk of the same type or outside (O) of a chunk phrase [18]. The named entities in the CoNLL-2003 dataset are annotated using four entity types, namely Location (LOC), Miscellaneous (MISC), Organization (ORG) and Person (PER). In Alan Ritter's corpus, entity types were not exactly the same, so they had to be converted, as we mention further on this section. The #MSM2013 corpus only contains annotated named entities and their types. To ease experimentation, this corpus was converted to the same format as the other two.

Listing 1 illustrates the annotation format for the experiments. Table 1 shows some numerical characteristics of the used datasets.

[17] http://trec.nist.gov/data/reuters/reuters.html
[18] https://github.com/aritter/twitter_nlp/tree/master/data/annotated
[19] http://oak.dcs.shef.ac.uk/msm2013/challenge.html
[20] https://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html

Table 1 Dataset properties.

Dataset	*Documents*	*Tokens*	*Average Tokens per Document*
CoNLL (Reuter Corpus)	946	203621	215
Twitter (Alan Ritter)	2394	46469	19
#MSM2013	2815	52124	19

Table 2 Datasets with PoS Tags.

	Dataset				
	Twitter (Alan Ritter)		CoNLL (Reuter Corpus)		Description
CC	305	(2.01 %)	3653	(1.79 %)	Coordinating conjunction
CD	268	(1.76 %)	19704	(9.68 %)	Cardinal number
DT	825	(5.43 %)	13453	(6.61 %)	Determiner
IN	1091	(7.18 %)	19064	(9.36 %)	Preposition or subordinating conjunction
JJ	670	(4.41 %)	11831	(5.81 %)	Adjective
MD	181	(1.19 %)	1199	(0.59 %)	Modal
NN	1931	(12.72 %)	23899	(11.74 %)	Noun, singular or mass
NNP	1159	(7.63 %)	34392	(16.89 %)	Proper noun, singular
NNS	393	(2.59 %)	9903	(4.86 %)	Noun, plural
PRP	1106	(7.28 %)	3163	(1.55 %)	Personal pronoun
PRP$	234	(1.54 %)	1520	(0.75 %)	Possessive pronoun
RB	680	(4.48 %)	3975	(1.95 %)	Adverb
RT	152	(1.00 %)	0		Retweet
TO	264	(1.74 %)	3469	(1.70 %)	to
UH	493	(3.25 %)	30	(0.01 %)	Interjection
URL	183	(1.21 %)	0		Url
USR	464	(3.06 %)	0		User
VB	660	(4.35 %)	4252	(2.09 %)	Verb, base form
VBD	306	(2.02 %)	8293	(4.07 %)	Verb, past tense
VBG	303	(2.00 %)	2585	(1.27 %)	Verb, gerund or present participle
VBN	140	(0.92 %)	4105	(2.02 %)	Verb, past participle
VBP	527	(3.47 %)	1436	(0.71 %)	Verb, non-3rd person singular present
VBZ	342	(2.25 %)	2426	(1.19 %)	Verb, 3rd person singular present
Others	908	(5.98%)	10478	(5.15 %)	

It is clear that the Twitter datasets (Alan Ritter and #MSM2013) have a greater number of documents with short sentences. On the other hand, the CoNLL dataset has longer and more complex sentences. Tables 2 and 3 show the distribution of the POS and chunk tags, respectively for Alan Ritter's and CoNLL-2003 corpora. For the POS tags, only those that account for more than one percent at least in one of the two datasets, excluding punctuation marks, are shown. Noun phrases (NP), prepositional phrases (PP) and verbal phrases (VP) are the most common chunks in both datasets.

For the NER evaluation, we stripped the IOB tags from the datasets whenever they were present, and joined them in a single entity tag, i.e, different tags such as B-LOC and I-LOC became simply LOC. Besides making comparison easier, this was made due to some noticed inconsistencies on the usage of I's and B's. Table 4 shows the distribution of the named entities in all of the used datasets.

We recall that the entity types in Alan Ritter's corpus are more and different than the other two. So, in order to enable comparison in the same lines, additional entity types were considered as alternative tags for one of the types covered by the CoNLL-2003 dataset: LOC,

Table 3 Datasets with Chunk Tags.

	Dataset				
	Twitter (Alan Ritter)		CoNLL (Reuter Corpus)		Description
B-ADJP	241	(1.58 %)	2	(0.00 %)	Begins an adjective phrase
B-ADVP	535	(3.52 %)	22	(0.01 %)	Begins an adverb phrase
B-CONJP	2	(0.01 %)	0		Begins a conjunctive phrase
B-INTJ	384	(2.52 %)	0		Begins an interjection
B-NP	3992	(26.24 %)	3777	(1.85 %)	Begins a noun phrase
B-PP	1027	(6.75 %)	254	(0.12 %)	Begins a prepositional phrase
B-PRT	109	(0.72 %)	0		Begins a particle
B-SBAR	103	(0.68 %)	8	(0.00 %)	Begins a subordinating clause
B-VP	1884	(12.39 %)	163	(0.08 %)	Begins a verb phrase
I-ADJP	86	(0.57 %)	1374	(0.67 %)	Is inside an adjective phrase
I-ADVP	66	(0.43 %)	2573	(1.35 %)	Is inside an adverb phrase
I-CONJP	2	(0.01 %)	70	(0.03 %)	Is inside a conjunctive phrase
I-INTJ	124	(0.82 %)	60	(0.03 %)	Is inside an interjection
I-LST	0		36	(0.02 %)	Is inside a list marker
I-NP	2686	(17.66 %)	120255	(59.06 %)	Is inside a noun phrase
I-PP	10	(0.07 %)	18692	(9.18 %)	Is inside a prepositional phrase
I-PRT	0		527	(0.26 %)	Is inside a particle
I-SBAR	5	(0.03 %)	1280	(0.63 %)	Is inside a subordinating clause
I-VP	842	(5.54 %)	26702	(13.11 %)	Is inside verb phrase
O	27646	(20.47 %)	3113	(13.58 %)	Is outside of any chunk.

MISC, ORG and PER. Table 5 shows the new entities distribution after performing the following mapping: FACILITY, GEO-LOC → LOC; MOVIE, TVSHOW, OTHER → MISC; COMPANY, PRODUCT, SPORTSTEAM → ORG; PERSON, MUSICARTIST → PER. This mapping considered the annotation guidelines of the CoNLL-2003 shared task[21].

5 Compared Tools

In order to select a suitable tool for our purpose, many criteria have to be considered. Among other properties, tools can be implemented in different programming languages; have available models that cover different tasks, kinds of text or languages; require different setups; or have different learning curves for simple usage or for integration. The tools compared in this paper were trained for English and are open, well-known and widely used by the NLP community. Moreover, they were developed either in Java or Python, which, nowadays, are probably the two languages more frequently used to develop NLP applications and for which there is a broader range of available toolkits. The compared tools are enumerated in the following list, where they are described and grouped in "standard" toolkits, which means they were developed with no specific kind of text in mind, and social network-oriented tools, which aim to be used in short messages from social networks.

5.1 Standard NLP toolkits

The *NLTK toolkit*[22] is a Python library aimed at individuals who are entering the NLP field. It is divided in independent modules, responsible for specific NLP tasks such as tokenization,

[21] `http://www.cnts.ua.ac.be/conll2003/ner/annotation.txt`
[22] `http://www.nltk.org`

Table 4 Datasets with NER Tags.

	Dataset					
	Twitter (Alan Ritter)		CoNLL (Reuter Corpus)		#MSM2013	
COMPANY	207	(0.45 %)	0		0	
FACILITY	209	(0.45 %)	0		0	
GEO-LOC	325	(0.70 %)	0		0	
LOC	0		8297	(4.07%)	795	(1.53 %)
MISC	0		4593	(2.26%)	511	(0.98 %)
MOVIE	80	(0.17 %)	0		0	
MUSICARTIST	116	(0.25 %)	0		0	
ORG	0		10025	(4.92 %)	842	(1.62 %)
OTHER	545	(1.39 %)	0		0	
PERSON	664	(1.43 %)	11128	(5.47 %)	2961	(5.68 %)
PRODUCT	177	(0.38 %)	0		0	
SPORTSTEAM	74	(0.16 %)	0		0	
TVSHOW	65	(0.14 %)	0		0	
O	44007	(94.70 %)	169578	(83.28 %)	47015	(90.20 %)

Table 5 Dataset with Joint NER Tags.

	Dataset					
	Twitter (Alan Ritter)		CoNLL (Reuter Corpus)		#MSM2013	
LOC	534	(1.15 %)	8297	(4.07 %)	795	(1.53 %)
MISC	690	(1.48 %)	4593	(2.26 %)	511	(0.98 %)
ORG	458	(0.99 %)	10025	(4.92 %)	842	(1.62 %)
PER	780	(1.68 %)	11128	(5.47 %)	2961	(5.68 %)
O	44007	(94.70 %)	169578	(83.28 %)	47015	(90.20 %)

stemming, tree representations, tagging, parsing and visualization. It also comes bundled with popular corpus samples ready to be read. By default, NLTK uses the Penn Treebank Tokenizer, which uses regular expressions to tokenize the text. Its PoS tagger uses the Penn Treebank tagset and is trained on the PENN Treebank corpus with a Maximum Entropy model. The Chunker and the NER modules are trained on the ACE corpus with a Maximum Entropy model [2, 15].

Apache OpenNLP[23] is a Java library that uses machine learning methods for common natural language tasks, such as tokenization, POS tagging, NER, chunking and parsing. Users can either rely on pre-trained models for the previous tasks or train their own with a Perceptron or a Maximum Entropy. The pre-trained models for English PoS tagging and chunking use the Penn Treebank tagset. The Chunker is trained on the CoNLL-2000 dataset. The pre-trained NER models provide cover the recognition of persons, locations, organizations, time, date and percentage expressions. Although there are two POS tagging models available for English, in this work, we used the one based on Maximum Entropy.

The *Stanford CoreNLP*[24] toolkit is a Java pipeline that provides common language processing tasks. The most supported language is English, but other languages are also available [16]. Comparing to other frameworks such as GATE [4] or UIMA [8], CoreNLP is simple to set up and run, since users do not need to learn and understand complex installations and procedures. The CoreNLP performs a Penn Treebank style tokenization and the POS module is an implementation of the Maximum Entropy model using the Penn Treebank tagset. The NER component uses a Conditional Random Field (CRF) model and is trained on the CoNLL-2003 dataset.

[23] `https://opennlp.apache.org`
[24] `http://stanfordnlp.github.io/CoreNLP`

Table 6 Toolkit properties.

System	Programming Language	Target Text	Tokenization	PoS Tagging	Chunking	NER
NLTK	Python	Generic	✓	✓	✓	✓
OpenNLP	Java	Generic	✓	✓	✓	✓
CoreNLP	Java	Generic	✓	✓	✗	✓
Pattern	Python	Generic	✓	✓	✓	✗
TweetNLP	Java	Social Media	✓	✓	✗	✗
TwitterNLP	Python	Social Media	✓	✓	✓	✓
TwitIE	Java	Social Media	✓	✓	✗	✓

Pattern[25] is a Python library that provides modules for web mining, NLP and ML tasks. This library does not provide methods for a single field but rather a general cross-domain and ease-of-use functionality. The PoS tagger uses a simple rule-based model trained on the Brown Corpus [6].

5.2 Social Network-Oriented Toolkits

Alan Ritter's *TwitterNLP*[26] is a Python library that offers a NLP pipeline for performing Tokenization, POS, Chunking and NER. The authors reduced the problem of dealing with noisy texts by developing a system based on a set of features extracted from Twitter-specific POS taggers, a dedicated shallow parsing logic, and the use of gazetteers generated from entities in the Freebase knowledge base, that best match the fleeting nature of informal texts [19].

CMU's *TweetNLP*[27] is Java tool that provides a Tokenizer and a POS Tagger with available models, trained with a CRF model in Twitter data, manually annotated by its authors [11]. In addition to the typical syntactic elements of a sentence, TweetNLP identifies content such as mentions, URLs, and emoticons.

TwitIE[28] is an open-source plugin for GATE. The GATE framework comes already packaged with ANNIE, an information extraction system, and includes resources such as: a Tokenizer, a sentence splitter, gazetteer lists, a PoS tagger and a semantic tagger. TwitIE re-uses some of these components (sentence splitter and gazeteer lists) but adapts the other to the Twitter kind of text, supporting language identification, Tokenization, normalization, PoS tagging and Name Entity Recognition. The TwitIE tokenizer follows the same tokenization scheme as TwitterNLP. The PoS tagger uses an adptation of the Stanford tagger, trained on tweets with the Penn Tree Bank tagset, with additional tags for retweets, URLs, hashtags and user mentions [3]. In our experiments, we used the Text Analytics web service[29] which includes a version of the TwitIE module.

5.3 Tools Summary

Table 6 summarizes additional properties of the aforementioned tools. Java is the most used programming language and only tools such as TweetNLP, TwitterNLP and TwitIE are made

[25] `http://www.clips.ua.ac.be/pages/pattern`
[26] `https://github.com/aritter/twitter_nlp`
[27] `http://www.cs.cmu.edu/~ark/TweetNLP`
[28] `https://gate.ac.uk/wiki/twitie.html`
[29] `http://docs.s4.ontotext.com/display/S4docs/Twitter+IE`

with models adapted to the social domain. In terms of task support, NLTK, OpenNLP and TwitterNLP offer a complete NLP pipeline (Tokenization, PoS, Chunking and NER). Without any additional plugin, CoreNLP, TweetNLP and TwitIE lack support for chunking, while Pattern and TweetNLP do not support NER.

6 Comparison Metrics

The performance of a NLP tool in a certain task can be estimated by the quality of its predictions on the classification of unseen data. Predictions made are either considered Positive or Negative (under some category) and expected judgments are called True or False (again, under a certain category). The following are common metrics used to assess classification tasks [24]:

- *Precision:* The proportion of correctly classified instances (True Positives) among all the classified instances under a certain category (True Positives and False Positives).

$$P_i = \frac{TP_i}{TP_i + FP_i}$$

P_i = Precision under Category i
TP_i = True Positives under Category i
FP_i = False Positives under Category i

- *Recall:* The proportion of correctly classified instances (True Positives) under a certain category (True Positives and False Negatives).

$$R_i = \frac{TP_i}{TP_i + FN_i}$$

R_i = Recall under Category i
TP_i = True Positives under Category i
FN_i = False Negatives under Category i

- *F-measure:* Combines precision and recall, and is computed as the harmonic mean between the two metrics.

$$F_1 = \frac{2 \times P_i \times R_i}{P_i + R_i}$$

F_1 = Harmonic Mean
P_i = Precision under Category i
R_i = Recall under Category i

The previous metrics provide insights on the behavior of the tool. We can go further and compute the previous estimations in different ways such as:

- *Micro Averaging:* the entire text is treated as a single document and the individual correct classifications are summed up.

$$P^{\mu} = \frac{\sum_{i=1}^{|C|} TP_i}{\sum_{i=1}^{|C|} TP_i + FP_i}$$

P^{μ} = Micro Precision
C = Set of Classes
TP = True Positives
FP = False Positives

$$R^{\mu} = \frac{\sum_{i=1}^{|C|} TP_i}{\sum_{i=1}^{|C|} TP_i + FN_i}$$

R^{μ} = Micro Recall
C = Set of Classes
TP = True Positives
FP = False Positives

- *Macro Averaging:* the precision and recall metrics are computed for each document and then averaged.

$$P^M = \frac{\sum_{i=1}^{|C|} P_i}{|C|}$$

P^M = Macro Precision
C = Set of Classes

$$R^M = \frac{\sum_{i=1}^{|C|} R_i}{|C|}$$

R^M = Macro Recall
C = Set of Classes

In addition to the previous averages, the standard deviation is a common dispersion metric that may be computed as follows:

$$\sigma = \frac{1}{N-1} \sum_{i=1}^{|N|} (x_i - \bar{x})^2$$

N = Number of samples
x_i = Result of the i-th measurement
$\bar{x}$ = Arithmetic mean of the N results

These evaluation metrics can give different results. Macro averaging weights each class equally, even if there are unbalanced classes. On the other hand, micro averaging weights the documents under evaluation, but it can happen that large classes dominates smaller classes. Therefore, macro averaging provides a sense of effectiveness on small classes, increasing their importance. Of course that selecting the appropriate metric depends on the requirements of the application.

7 Comparison Results

This section reports on the results obtained when performing the addressed tasks on the gold standard datasets, presented earlier, using each toolkit. Tables 7, 8, 9, 10 and 11 show the precision (P), the recall (R) and the F_1-scores for each scenario. The presented results are macro averages, i.e, we computed the precision, recall and F_1 for each document (tweet or news) and then averaged the results. The standard deviations associated with the computed macro-averages (σ) are also presented. Micro averages were not computed because we were more interested in assessing the toolkits performance in different documents and not to use the whole corpus as a large document, which would lower the impact of less frequent tags.

More precisely, each table targets a different task, lines have the results for each tool and there are three columns per corpus (P, R and F_1). Table 7 targets tokenization, Table 8 POS-tagging, and Table 9 chunking. Tables 10 and 11 show two different NER results: entity identification (NER) only considers the delimitation of a named entity, while entity classification (NEC) also considers its given type. Table 11 has an additional line with the results of the best performing system that participated in the CoNLL-2003 shared task [9], which combined four different classifiers (robust linear classifier, maximum entropy, transformation-based learning and a hidden Markov model), resulting in $F_1 = 89\%$ in named entity classification (NEC).

On the CoNLL dataset, which uses formal language, standard toolkits perform well. OpenNLP excels with $F_1 = 99\%$ in tokenization, 88% in POS-tagging and 83% in chunking. In the NER task, NLTK (89%) and OpenNLP (88%) performed closely. TwitterNLP also performed well in this dataset. This is not that surprising if we add that the CoNLL-2003 dataset was one of the corpora TwitterNLP was trained on [19], and it is probably also tuned for this corpus.

As expected, the performance of standard toolkits, developed with formal text in mind, decreases when used in the social network corpora. This difference is between 5-8% for tokenization, 17% for POS-tagging, 17-40% for chunking, or 5-18% for NER. This is not

Table 7 Tokenization Performance Results.

Task	Tokenization					
Data set	CoNLL			Alan Ritter - Twitter		
Metric / Tool	P ± σ	R ± σ	F1 ± σ	P ± σ	R ± σ	F1 ± σ
NLTK	0.95 ± 0.11	0.96± 0.10	0.95 ± 0.11	0.83 ± 0.14	0.91 ± 0.09	0.87 ± 0.12
OpenNLP	0.99 ± 0.02	0.99 ± 0.01	0.99 ± 0.02	0.92 ± 0.11	0.96 ± 0.06	0.94 ± 0.08
CoreNLP	0.73 ± 0.31	0.73 ± 0.31	0.73 ± 0.31	0.93 ± 0.13	0.95 ± 0.11	0.94 ± 0.12
Pattern	0.42 ± 0.30	0.41 ± 0.29	0.42 ± 0.29	0.76 ± 0.21	0.78 ± 0.20	0.77 ± 0.20
TweetNLP	0.97± 0.05	0.98 ± 0.02	0.98 ± 0.04	0.96 ± 0.07	0.98 ± 0.05	0.97 ± 0.06
TwitterNLP	0.95 ± 0.10	0.97 ± 0.09	0.96 ± 0.10	0.96 ± 0.07	0.97 ± 0.05	0.96 ± 0.06
TwitIE	0.85 ± 0.15	0.93 ± 0.11	0.89 ± 0.14	0.83 ± 0.16	0.89 ± 0.11	0.86 ± 0.13

Table 8 PoS Performance Results.

Task	PoS Tagging					
Data set	CoNLL			Alan Ritter - Twitter		
Metric / Tool	P ± σ	R ± σ	F1 ± σ	P ± σ	R ± σ	F1 ± σ
NLTK	0.65 ± 0.19	0.71 ± 0.18	0.68 ± 0.18	0.65 ± 0.19	0.71 ± 0.18	0.68 ± 0.18
OpenNLP	0.88 ± 0.10	0.88 ± 0.09	0.88 ± 0.10	0.70 ± 0.18	0.73 ± 0.17	0.71 ± 0.17
CoreNLP	0.67 ± 0.29	0.67 ± 0.29	0.67 ± 0.29	0.70 ± 0.19	0.71 ± 0.18	0.71 ± 0.18
Pattern	0.36 ± 0.24	0.35 ± 0.24	0.35 ± 0.24	0.61 ± 0.21	0.62 ± 0.21	0.61 ± 0.20
TweetNLP	0.83 ± 0.10	0.84 ± 0.09	0.84 ± 0.09	0.94 ± 0.08	0.96 ± 0.06	0.95 ± 0.07
TwitterNLP	0.83 ± 0.15	0.84 ± 0.15	0.83 ± 0.15	0.92 ± 0.11	0.93 ± 0.11	0.92 ± 0.11
TwitIE	0.78 ± 0.16	0.85 ± 0.12	0.82 ± 0.14	0.78 ± 0.17	0.84 ± 0.13	0.81 ± 0.14

Table 9 Chunking Performance Results.

Task	Chunking					
Data set	CoNLL			Alan Ritter - Twitter		
Metric / Tool	P ± σ	R ± σ	F1 ± σ	P ± σ	R ± σ	F1 ± σ
NLTK	0.70 ± 0.10	0.71 ± 0.10	0.71 ± 0.10	0.51 ± 0.16	0.56 ± 0.16	0.54 ± 0.16
OpenNLP	0.83 ± 0.13	0.83 ± 0.12	0.83 ± 0.12	0.44 ± 0.34	0.46 ± 0.36	0.45 ± 0.39
CoreNLP	n/a	n/a	n/a	n/a	n/a	n/a
Pattern	0.33 ± 0.22	0.32 ± 0.21	0.33 ± 0.21	0.54 ± 0.21	0.56 ± 0.20	0.55 ± 0.20
TweetNLP	n/a	n/a	n/a	n/a	n/a	n/a
TwitterNLP	0.82 ± 0.13	0.84 ± 0.12	0.83 ± 0.13	0.90 ± 0.12	0.91 ± 0.11	0.90 ± 0.11
TwitIE	n/a	n/a	n/a	n/a	n/a	n/a

Table 10 NER Performance Results.

Task	NER					
Data set	CoNLL			Alan Ritter - Twitter		
Metric / Tool	P ± σ	R ± σ	F1 ± σ	P ± σ	R ± σ	F1 ± σ
NLTK	0.88 ± 0.12	0.89 ± 0.11	0.89 ± 0.11	0.77 ± 0.16	0.84 ± 0.13	0.80 ± 0.15
OpenNLP	0.88 ± 0.09	0.88 ± 0.08	0.88 ± 0.08	0.85 ± 0.14	0.90 ± 0.11	0.87 ± 0.12
CoreNLP	0.70 ± 0.30	0.70 ± 0.30	0.70 ± 0.30	0.87 ± 0.15	0.89 ± 0.14	0.88 ± 0.15
Pattern	n/a	n/a	n/a	n/a	n/a	n/a
TweetNLP	n/a	n/a	n/a	n/a	n/a	n/a
TwitterNLP	0.88 ± 0.11	0.89 ± 0.10	0.88 ± 0.11	0.96 ± 0.07	0.97 ± 0.05	0.97 ± 0.06
TwitIE	0.74 ± 0.16	0.80 ± 0.14	0.77 ± 0.15	0.77 ± 0.17	0.83 ± 0.14	0.80 ± 0.15

Table 11 NEC Performance Results.

Task	NEC					
Data set	CoNLL			Alan Ritter - Twitter		
Metric / Tool	P ± σ	R ± σ	F1 ± σ	P ± σ	R ± σ	F1 ± σ
NLTK	0.84 ± 0.12	0.84 ± 0.12	0.84 ± 0.12	0.75 ± 0.17	0.83 ± 0.14	0.79 ± 0.15
OpenNLP	0.87 ± 0.10	0.87 ± 0.09	0.87 ± 0.09	0.85 ± 0.15	0.89 ± 0.12	0.87 ± 0.13
CoreNLP	0.70 ± 0.30	0.70 ± 0.30	0.70 ± 0.30	0.87 ± 0.16	0.89 ± 0.14	0.88 ± 0.15
Pattern	n/a	n/a	n/a	n/a	n/a	n/a
TweetNLP	n/a	n/a	n/a	n/a	n/a	n/a
TwitterNLP	0.84 ± 0.13	0.85 ± 0.12	0.85 ± 0.12	0.95 ± 0.08	0.96 ± 0.07	0.95 ± 0.08
TwitIE	0.73 ± 0.17	0.80 ± 0.14	0.76 ± 0.16	0.77 ± 0.17	0.84 ± 0.14	0.80 ± 0.15
Florian et al.	0.89	0.89	0.89 ± 0.70	n/a	n/a	n/a

Table 12 NER/NEC Performance Results on the #MSM2013 Data set.

Data set	#MSM2013 - Twitter					
Task	NER			NEC		
Metric / Tool	P ± σ	R ± σ	F1 ± σ	P ± σ	R ± σ	F1 ± σ
NLTK	0.83 ± 0.16	0.83 ± 0.16	0.83 ± 0.14	0.85 ± 0.14	0.85 ± 0.15	0.85 ± 0.13
OpenNLP	0.83 ± 0.14	0.86 ± 0.14	0.85 ± 0.14	0.84 ± 0.14	0.86 ± 0.13	0.85 ± 0.13
CoreNLP	0.73 ± 0.19	0.83 ± 0.16	0.78 ± 0.16	0.73 ± 0.19	0.84 ± 0.16	0.78 ± 0.16
Pattern	n/a	n/a	n/a	n/a	n/a	n/a
TweetNLP	n/a	n/a	n/a	n/a	n/a	n/a
TwitterNLP	0.90 ± 0.12	0.90 ± 0.12	0.90 ± 0.12	0.91 ± 0.11	0.91 ± 0.11	0.91 ± 0.11
TwitIE	0.61 ± 0.20	0.73 ± 0.18	0.66 ± 0.18	0.61 ± 0.20	0.73 ± 0.17	0.66 ± 0.18
Habib et al.	0.72	0.80	0.76	0.65	0.73	0.69

the case of Pattern, which performs poorly in the CoNLL corpus but improves significantly when tokenizing, PoS tagging and chunking the Twitter corpora. Although not developed specifically for Twitter, OpeNLP and CoreNLP still obtain interesting results for tokenization and NER in its corpus ($F_1 > 80\%$).

Also as expected, in the Twitter corpus, the Twitter-oriented toolkits performed better than the others. TweetNLP was the best in the tokenization (97%) and POS-tagging (95%) tasks. TwitterNLP performed closely (96% and 92%). In the case of TwitIE, the difference of performance in different types of text was not relevant. Once again, it should be highlighted that TwitterNLP was trained with the Twitter corpus, so this comparison is not completely fair. This is also why we used an additional corpus, #MSM2013, which covers social network text. The results of the NER task in this corpus, shown in table 12, confirm the good performance of TwitterNLP. In the last line of the previous table, we also present the official results of the best system that participated in the #MSM2013 Concept Extraction Task, Habib et al. [14], which apparently underperformed TwitterNLP. Habib et al. combined Conditional Random Fields with Support Vector Machines for recognition and, for classification, each entity was disambiguated and linked to its Wikipedia article, where the category was extracted from.

8 Conclusions

We presented a set of experiments aiming at comparing the performance of different open-domain NLP toolkits, which were used to perform different NLP tasks on different kinds

of text, namely news (more formal) and social media text (higher proportion of informal documents).

We have shown that, using only the available pre-trained models, there is not one toolkit that overperformed all the others in every scenario. Though, some are more balanced than the others. Even if it cannot be seen as a strong conclusion, the results suggest that OpenNLP is the best choice for news text, and TwitterNLP for social media text. Although the latter result was biased on the TWitter corpus, where TwitterNLP was trained on, we also tested it on another corpus, where it got the best results. It should be noticed that we ended up using datasets that were more appropriate for specific tasks. For instance, although its text of the CoNLL-2003 dataset is tokenized, POS-tagged, and chunked, it was specifically developed for a NER shared task. On the other hand, we did not use the CoNLL-2000, developed for a chunking shared task. Although this dataset was used to train some of the OpenNLP models, we should also consider its results in the future.

As expected, standard toolkits perform better in formal texts, while Twitter-oriented tools got better results in social media text. Besides helping us to make a selection, we believe that these results might be useful for potential users willing to select the most appropriate tools for their specific purposes, especially if they do not have time or expertise to train new models. Of course, we did not use all the available tools, especially those available as web services, but we tried to cover an acceptable range of widely used toolkits that cover several NLP tasks and developed in two programming languages with a large community – Java and Python. We also regard that, with more available manually annotated datasets, either with formal or informal language, we could always re-train some of the available models and possibly increase the performance achieved with most of the tested tools.

References

1. Samet Atdag and Vincent Labatut. A Comparison of Named Entity Recognition Tools Applied to Biographical Texts. In *Systems and Computer Science (ICSCS), 2013 2nd International Conference on*, pages 228–233, Villeneuve d'Ascq, France, August 2013. IEEE.
2. Steven Bird. NLTK: The Natural Language Toolkit. In *Proceedings of the COLING/ACL on Interactive Presentation Sessions*, COLING-ACL'06, pages 69–72, Sydney, Australia, 2006.
3. Kalina Bontcheva, Leon Derczynski, Adam Funk, Mark A. Greenwood, Diana Maynard, and Niraj Aswani. TwitIE: An Open-Source Information Extraction Pipeline for Microblog Text. In *Proceedings of the International Conference on Recent Advances in Natural Language Processing*. Association for Computational Linguistics, 2013.
4. Hamish Cunningham, Diana Maynard, Kalina Bontcheva, and Valentin Tablan. GATE: An Architecture for Development of Robust HLT Applications. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, pages 168–175, Philadelphia, Pennsylvania, 2002.
5. Walter Daelemans, Jakub Zavrel, Peter Berck, and Steven Gillis. MBT: A Memory-Based Part of Speech Tagger-Generator. *arXiv preprint cmp-lg/9607012*, 1996.
6. Tom De Smedt and Walter Daelemans. Pattern for Python. *The Journal of Machine Learning Research*, 13(1):2063–2067, 2012.
7. Štefan Dlugolinský, Peter Krammer, Marek Ciglan, Michal Laclavík, and Ladislav Hluchý. Combining Named Enitity Recognition Tools. In *Making Sense of Microposts (# MSM2013)*, Rio de Janeiro, Brazil, May 2013.
8. David Ferrucci and Adam Lally. UIMA: An Architectural Approach to Unstructured Information Processing in the Corporate Research Environment. *Natural Language Engineering*, 10(3-4):327–348, September 2004.

9 Radu Florian, Abe Ittycheriah, Hongyan Jing, and Tong Zhang. Named Entity Recognition through Classifier Combination. In Walter Daelemans and Miles Osborne, editors, *Proceedings of CoNLL-2003*, pages 168–171. Edmonton, Canada, 2003.

10 Marcos Garcia and Pablo Gamallo. Yet Another Suite of Multilingual NLP Tools. In *Languages, Applications and Technologies – Revised Selected Papers of 4th International Symposium SLATE, Madrid, Spain, June 2015*, CCIS, pages 65–75. Springer, 2015.

11 Kevin Gimpel, Nathan Schneider, Brendan O'Connor, Dipanjan Das, Daniel Mills, Jacob Eisenstein, Michael Heilman, Dani Yogatama, Jeffrey Flanigan, and Noah A Smith. Part-of-speech Tagging for Twitter: Annotation, Features, and Experiments. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies: Short Papers – Volume 2*, pages 42–47, Portland, Oregon, 2011.

12 Fréderic Godin, Pedro Debevere, Erik Mannens, Wesley De Neve, and Rik Van de Walle. Leveraging Existing Tools for Named Entity Recognition in Microposts. In *Making Sense of Microposts (# MSM2013)*, pages 36–39, Rio de Janeiro, Brazil, May 2013.

13 Meritxell González Bermúdez. An analysis of Twitter corpora and the difference between formal and colloquial tweets. In *Proceedings of the Tweet Translation Workshop 2015*, pages 1–7. CEUR-WS. org, 2015.

14 Mena Habib, Maurice Van Keulen, and Zhemin Zhu. Concept extraction challenge: University of Twente at #msm2013. In *Making Sense of Microposts (#MSM2013) Concept Extraction Challenge*, pages 17–20, 2013. URL: `http://ceur-ws.org/Vol-1019/paper_14.pdf`.

15 Edward Loper and Steven Bird. NLTK: The Natural Language Toolkit. In *Proceedings of the ACL-02 Workshop on Effective Tools and Methodologies for Teaching Natural Language Processing and Computational Linguistics – Volume 1*, ETMTNLP'02, pages 63–70, Philadelphia, Pennsylvania, 2002.

16 Christopher D. Manning, Mihai Surdeanu, John Bauer, Jenny Rose Finkel, Steven 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, Baltimore, USA, 2014.

17 Mitchell P. Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. Building a Large Annotated Corpus of English: The Penn Treebank. *Comput. Linguist.*, 19(2):313–330, June 1993. URL: `http://dl.acm.org/citation.cfm?id=972470.972475`.

18 Lance A. Ramshaw and Mitchell P. Marcus. Text Chunking using Transformation-Based Learning. In *Proceedings of the ACL Third Workshop on Very Large Corpora*, pages 82–94, June 1995.

19 Alan Ritter, Sam Clark, and Oren Etzioni. Named Entity Recognition in Tweets: An Experimental Study. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 1524–1534, Edinburgh, Scotland, July 2011.

20 Giuseppe Rizzo, Raphaël Troncy, Sebastian Hellmann, and Martin Bruemmer. NERD meets NIF: Lifting NLP Extraction Results to the Linked Data Cloud. *LDOW*, 937, 2012.

21 Giuseppe Rizzo, Marieke van Erp, and Raphaël Troncy. Benchmarking the Extraction and Disambiguation of Named Entities on the Semantic Web. In *International Conference on Language Resources and Evaluation*, pages 4593–4600, 2014.

22 Kepa Joseba Rodriquez, Mike Bryant, Tobias Blanke, and Magdalena Luszczynska. Comparison of Named Entity Recognition Tools for Raw OCR Text. In *KONVENS*, pages 410–414, Vienna, Austria, 2012.

23 Sara Rosenthal, Preslav Nakov, Svetlana Kiritchenko, Saif Mohammad, Alan Ritter, and Veselin Stoyanov. SemEval-2015 Task 10: Sentiment Analysis in Twitter. In *Proceedings of the 9th International Workshop on Semantic Evaluation (SemEval 2015)*, pages 451–463, Denver, Colorado, June 2015. Association for Computational Linguistics. URL: `http://www.aclweb.org/anthology/S15-2078`.

24 Fabrizio Sebastiani. Machine Learning in Automated Text Categorization. *ACM Computing Surveys*, 34(1):1–47, Mars 2002.

25 Erik F. Tjong Kim Sang and Fien De Meulder. Introduction to the CoNLL-2003 Shared Task: Language-Independent Named Entity Recognition. In Walter Daelemans and Miles Osborne, editors, *Proceedings of CoNLL-2003*, pages 142–147. Edmonton, Canada, 2003.

Comparing and Benchmarking Semantic Measures Using SMComp*

Teresa Costa[1] **and José Paulo Leal**[2]

1 **CRACS & INESC-Porto LA, Faculty of Sciences, University of Porto, Porto, Portugal**
`teresa.costa@dcc.fc.up.pt`

2 **CRACS & INESC-Porto LA, Faculty of Sciences, University of Porto, Porto, Portugal**
`zp@dcc.fc.up.pt`

Abstract

The goal of the semantic measures is to compare pairs of concepts, words, sentences or named entities. Their categorization depends on what they measure. If a measure only considers taxonomy relationships is a similarity measure; if it considers *all* type of relationships it is a relatedness measure.

The evaluation process of these measures usually relies on semantic gold standards. These datasets, with several pairs of words with a rating assigned by persons, are used to assess how well a semantic measure performs.

There are a few frameworks that provide tools to compute and analyze several well-known measures. This paper presents a novel tool – SMComp – a testbed designed for path-based semantic measures. At its current state, it is a domain-specific tool using three different versions of WordNet.

SMComp has two views: one to compute semantic measures of a pair of words and another to assess a semantic measure using a dataset. On the first view, it offers several measures described in the literature as well as the possibility of creating a new measure, by introducing Java code snippets on the GUI. The other view offers a large set of semantic benchmarks to use in the assessment process. It also offers the possibility of uploading a custom dataset to be used in the assessment.

1998 ACM Subject Classification E.1 Graphs and networks, I.2.4 Knowledge Representation Formalisms and Methods, Semantic networks, H.3.5 Online Information Services, Web-based services, H.5.3 Group and Organization Interfaces, Web-based interaction

Keywords and phrases Semantic similarity, semantic relatedness, testbed, web application

Digital Object Identifier 10.4230/OASIcs.SLATE.2016.4

1 Introduction

Semantic measures are an attempt to quantify and compare pairs of concepts, words, sentences or named entities that can be regarded as a kind of semantic distance in a semantic space [12].

* This work is partially financed by the ERDF – European Regional Development Fund through the Operational Programme for Competitiveness and Internationalisation – COMPETE 2020 Programme and by the FCT within project POCI-01-0145-FEDER-006961 and project "NORTE-01-0145-FEDER-000020" financed by the North Portugal Regional Operational Programme (NORTE 2020), under the PORTUGAL 2020 Partnership Agreement and through the European Regional Development Fund (ERDF).

© Teresa Costa and José Paulo Leal;
licensed under Creative Commons License CC-BY

This analysis is difficult since the object of semantic measures is inherently psychological, thus their evaluation rely on datasets that average the human perception of the semantic relationships.

There are two main types of semantic measures: similarity and relatedness. Despite being two different concepts they are commonly confused. Similarity measures the amount of common features between a pair of entities. It depends on the size of the smallest class that contains them. Relatedness depends on any relationship type connecting two elements, including but not restricted to class membership and inclusion. For instance, cat and dog are two similar concepts since they are both mammals; the same can be said about ant and flee since they are both insects. An ant and a dog are also similar insofar they are both animals, but less similar than cat and dog. The reason why is that the class *animals* contains both the classes *mammal* and *insect*. Flees are related to cats and dogs since they parasite them, thus flees are more related to dogs than ants. This happens not because the features they share, and they share because they are animals, but because of other relationships, in this case parasitism. In such way, the similarity of dogs and flees may be the same as the similarity of dogs and ants, but the relatedness between flees and dogs is greater than dogs and ants.

In spite of the clear difference between similarity and relatedness, people often confuse them or value them differently, as the classical example of how people compare a magazine, a pencil and a notepad [9] demonstrates. This difference is particularly noted when the same pair of words is measure using both types: similarity and relatedness.

This paper presents SMComp, a testbed for semantic measures freely available on-line[1]. This web application provides several well-known path based semantic measures and also supports user-defined measures. It has also an assessment mode, using several semantic gold standards or a custom dataset, uploaded by the user.

Section 2 surveys different semantic measures, datasets and libraries. It categorizes the types of semantic measures regarding their source, with focus on knowledge-based measures using the structural approach. This section also provides an overview on other tools used to compute and analyze semantic measures. It identifies also the semantic datasets commonly used in the assessment of semantic measures.

Section 3 introduces SMComp, providing an overview of this testbed. It details on its architecture and its behavior. The validation process was performed using the semantic graphs and measures available in the web application. The results are presented in Section 4. Section 5 summarizes the research present in this paper and highlights its main contribution.

2 Background

A semantic measure is a numerical estimation of how semantically connected two elements are. This evaluation relies on the analysis of information extracted from semantic sources. The type of a semantic measure depends on the type of the semantic source. The unstructured / semi-structured sources (plain texts or dictionaries, for instance) are used by Distributional Measures. Structured sources, such as semantic graphs (WordNet or DBPedia, for instance) are used by Knowledge-based Measures. These measures follow three different approaches: the structural [15, 23, 30] approach, the feature-based [4, 25, 28] approach and the Shannon's Information Theoretical [16, 22, 20, 21] approach. There are also hybrid measures [2, 3, 18, 26] that combine distributional and knowledge-based approaches and take advantage of both representations.

[1] `http://quilter.dcc.fc.up.pt/smcomp/`

This paper focus on structural knowledge-based measures, namely on path-based measures. A common knowledge source used on the assessment of these measures is WordNet[2] [7]. This knowledge base is usually represented as a semantic graph that models the English lexical knowledge. Its structure uses synsets as nodes and they are connected by semantic and lexical relationships.

Path-based measures follow a structural approach. They take advantage of several graph traversal strategies, such as shortest path, random walks and other interaction analysis. These measures are based on the analysis of the interconnections between nodes and use it to estimate the similarity (or relatedness) between them. They rely on the definition of *shortest path* and *least common subsumer*. The measures proposed by Rada et al. [23], Resnik [26], Leacock & Chodorow [15] and Wu & Palmer [30] are examples of path-based similarity measures. Hirst & St-Onge [14] and Strube & Ponzetto [29] proposed several relatedness measures.

There are several tools for the computation and analysis of semantic measures. They can be categorized according to the interface they support. The programming packages are usually general libraries, not focused in any particular semantic graph. They can be used only through their application interface (API) which requires programming just to be used. SML[3] [11] is an example of this kind of tool. It is simultaneously a library and a toolkit designed to the computation of semantic measures, such as similarity, relatedness and distance, supporting a large range of knowledge-based measures.

User packages provide *both* and API and a graphic user interface (GUI), usually deployed on the web. Typically, they are domain-specific since they are designed for a particular semantic graph. WordNet::Similarity [19] is a Perl module that implements several semantic similarity and relatedness measures. The results are based on the information found in a specific version of WordNet. WordNet::Similarity is freely available for download and has also a web interface[4]. WS4J[5] is a Java API for several semantic similarity and relatedness measures. It is freely available for download and has also a web application[6] where one can compute simultaneously all the semantic measures available between a pair of concepts also in a specific version of WordNet. None of these tools provide measure assessment from the GUI.

The assessment of semantic measures usually rely on the correlation of a measure with expected scores of word pairs. This provides an insight of how well they mimic the human capacity to compare things. Most of datasets used in the measure assessment average human ratings for a set of word pairs [12]. The ratings are collected during experiments involving several participants. According to the instructions provided to them, they assign a similarity or relatedness score to the word pairs. Usually, each dataset has its own set of instructions which may influence the notions of similarity and relatedness. Thus, datasets are also categorized as intended for similarity or relatedness. This work uses four different similarity datasets [27, 17, 1, 13] and five relatedness datasets [8, 1, 5, 10, 24].

3 SMComp

SMComp is a testbed for semantic measures, developed with Google Web Toolkit (GWT), that is freely available on-line[7]. It presents a novel approach that combines the GUI and

[2] https://wordnet.princeton.edu/
[3] http://www.semantic-measures-library.org/sml/index.php
[4] http://marimba.d.umn.edu/cgi-bin/similarity/similarity.cgi
[5] https://code.google.com/archive/p/ws4j/
[6] http://ws4jdemo.appspot.com/
[7] http://quilter.dcc.fc.up.pt/smcomp/

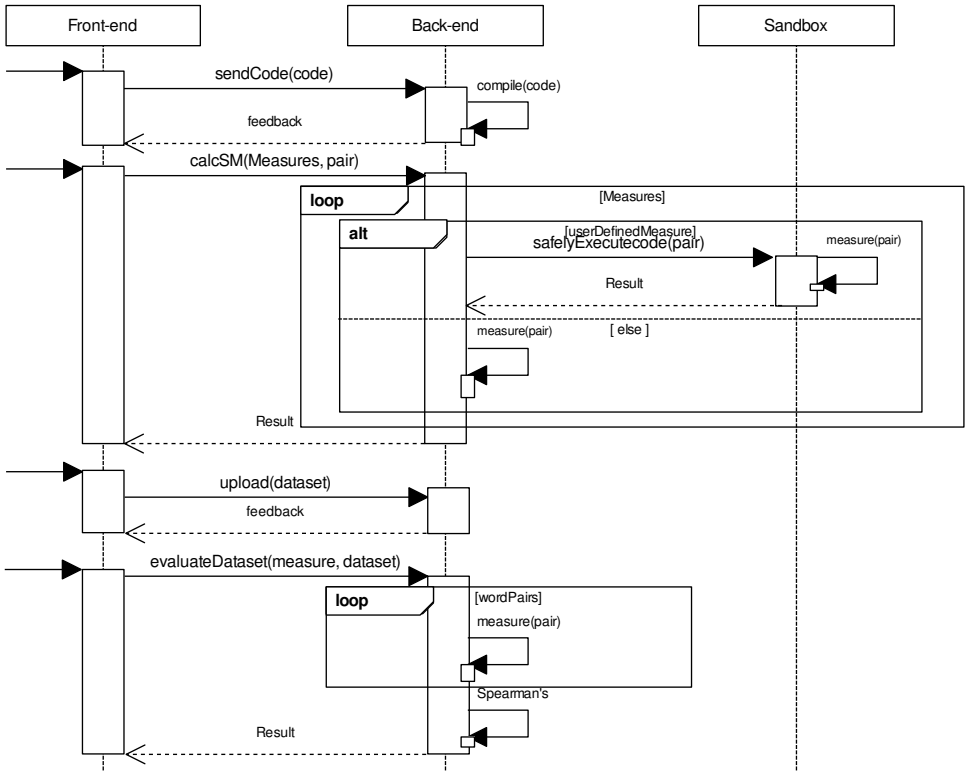


Figure 1 SMComp sequence diagram.

API features. Using its web interface the user can test a range of semantic measures and assess them with several datasets referred in the literature. This GUI also allows the user to provide a snippet of Java code to implement his own semantic measure and quickly assess its performance with relevant benchmarks.

The presented web application has four main distinctive features:

- it compares a semantic similarity measure with its corresponding relatedness version;
- it allows the user to create his own semantic measure and test it with the available semantic sources;
- it allows the user to upload a custom dataset;
- it assesses semantic measures using a set of semantic datasets.

The following Subsections detail on each aspect of SMCom: architecture, front-end, back-end and sandbox.

3.1 Architecture

SMComp is a testbed composed of 3 main components: a *front-end*, running on a web client, responsible for user interaction; a *back-end* running on a server responsible for computing semantic measures; and a *sandbox* responsible for the safe execution of user-provided Java code.

Figure 1 shows an UML sequence diagram summarizing the interaction between components. It has 4 key moments: submit a user-defined method, compute semantic measures, upload a custom dataset and assess a measure. On the first moment, the Java code is sent to the back-end server to be compiled. On the second moment, a selection of measures is sent

for computation. The back-end server iterates over the set of measures. If it includes the user-defined measure, it is sent to the sandbox to be safely executed. On the third moment, the user uploads his dataset to be used in the evaluation process. This process occurs on the final moment, where a single measure is assessed using a selected dataset.

3.2 Back-end

Semantic Graph Processing

The implemented semantic measures follow a structural approach, depending on graph traversal strategies to find paths connecting two words. This strategy can be a very time-consuming process, in particular if a remote source is used. To rely on remote sources would raise some issues: there is no guarantees that they are always available and network performance issues may hinder the execution of a semantic measures, since it requires a massive amount of graph queries.

In order to overcome these constraints, this application relies on a pre-processed graph. Known knowledge-bases, such as WordNet, usually provide dumps of their data. These dumps, in Resource Description Framework (RDF), were previously pre-processed in order to store the graph locally.

The graph pre-processing requires to parse RDF data. This data can be retrieved in several formats, such as Turtle, RDF/XML or N-Triples. To simplify and unify the process, all dumps are converted to N-Triples, the simplest RDF serialization. This conversion relies on the Apache Jena Framework[8].

The SMComp measures are path-based. Thus, the internal representation of semantic graphs in SMComp is designed to simplify the manipulation of paths. A path can be seen as a sort of chain, composed by a sequence of links. Hence, a pre-processed graph in SMComp can be seen as a collection of links ready to form chains. These chain links are called `Transition` and are made of a pair of `Node`. This internal representation is created by loading semantic graph serializations.

A semantic graph in RDF is a collection of triples of the form *subject* - *predicate* - *object*. This data is represented in SMComp using the types depicted in the UML class diagram in Figure 2. Each element of a triple is converted to a `Node<N>`, where the type parameter `N` is the type of the node representation, such as `String` or `Jena Resource`. A *subject* must be a `Concept Node` and the same is true for the *predicate* . The *object* can be either a `Value Node` or a `Concept Node`. `Concept Node` are serialized in RDF by URIs and `Value Node` are serialized as literals. For each *subject*, a `Transition` is created and associated to it, using the values of the *predicate* and *object*.

A `Path` is a more complex data type that has a node at its first position followed by a sequence of transitions. Paths are create by semantic measures using the `ProcessedGraph`.

This mapping process creates a locally stored graph, called `ProcessedGraph`. Figure 2 shows its operations and the relationships with the other types. The use of the `ProcessedGraph` eases the process of adding and retrieving path components, i.e. `Transition` instances.

SMcomp uses the `ProcessedGraph` format to store the available knowledge bases. This conversion process is executed only once, when a new semantic graph is added to SMComp.

[8] `https://jena.apache.org/`

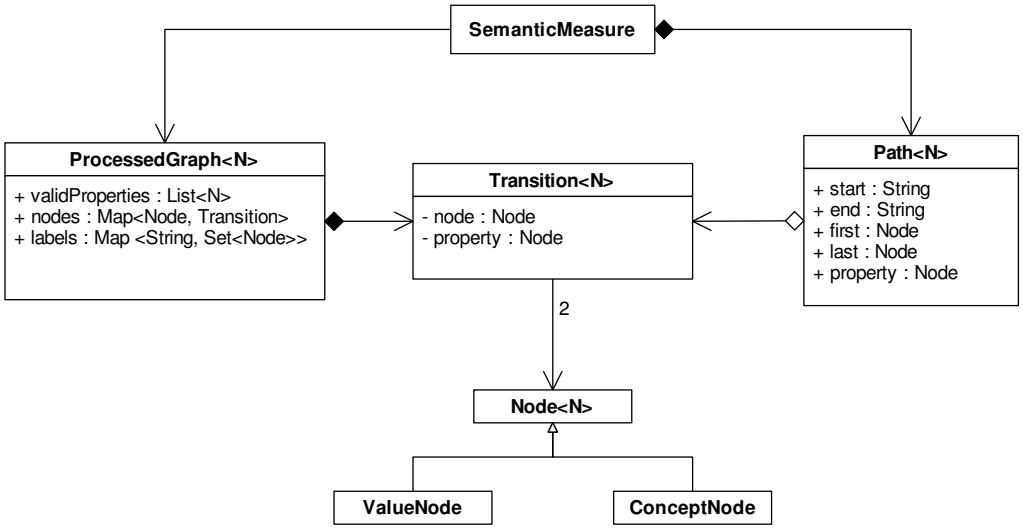


Figure 2 UML class diagram of the ProcessedGraph.

The RDF dumps used are available for WordNet 2.1[9], WordNet 3.0[10] and WordNet 3.1[11]. WordNet 2.1 uses 4.8GB of disk space, WordNet 3.0 uses 6.6GB of disk space and WordNet 3.1 uses 6.3GB.

Semantic Measures

SMcomp implements several path-based semantic measures, namely those described in Section 2. With the exception of the Hirst & St-Onge, the proposed measures were originally designed to estimate semantic similarity. However, they were adapted by Strube & Ponzetto to also measure semantic relatedness.

In addition to these measures, SMComp also implements a Resnik and a Hirst & St-Onge adaptation, enabling measure relatedness with the former and similarity with the later. The approach followed in these adaptations is similar to that used by Strube & Ponzetto [29]. To compute semantic relatedness using the Resnik measure, all the properties must be considered instead of only the taxonomic ones. To compute semantic similarity with Hirst & St-Onge, one must only consider the only taxonomic properties instead of the *allowable*[12] ones.

The implementation process considered the following assumptions:

- the value of a semantic measure between a word and itself it is the maximum value;
- if at least one of the words is not in the semantic graph the value of its measure is the minimum value;
- if the semantic graph has several roots, a new node is inserted to create a semantic tree with a single root node;

[9] `http://sourceforge.net/projects/texai`
[10] `http://semanticweb.cs.vu.nl/lod/wn30/`
[11] `http://wordnet-rdf.princeton.edu/`
[12] The *allowble* relations include: *see also, antonymy, attribute, cause, entailment, holonymy, hypernymy, hyponymy, meronymy, pertinence* and *similarity*.

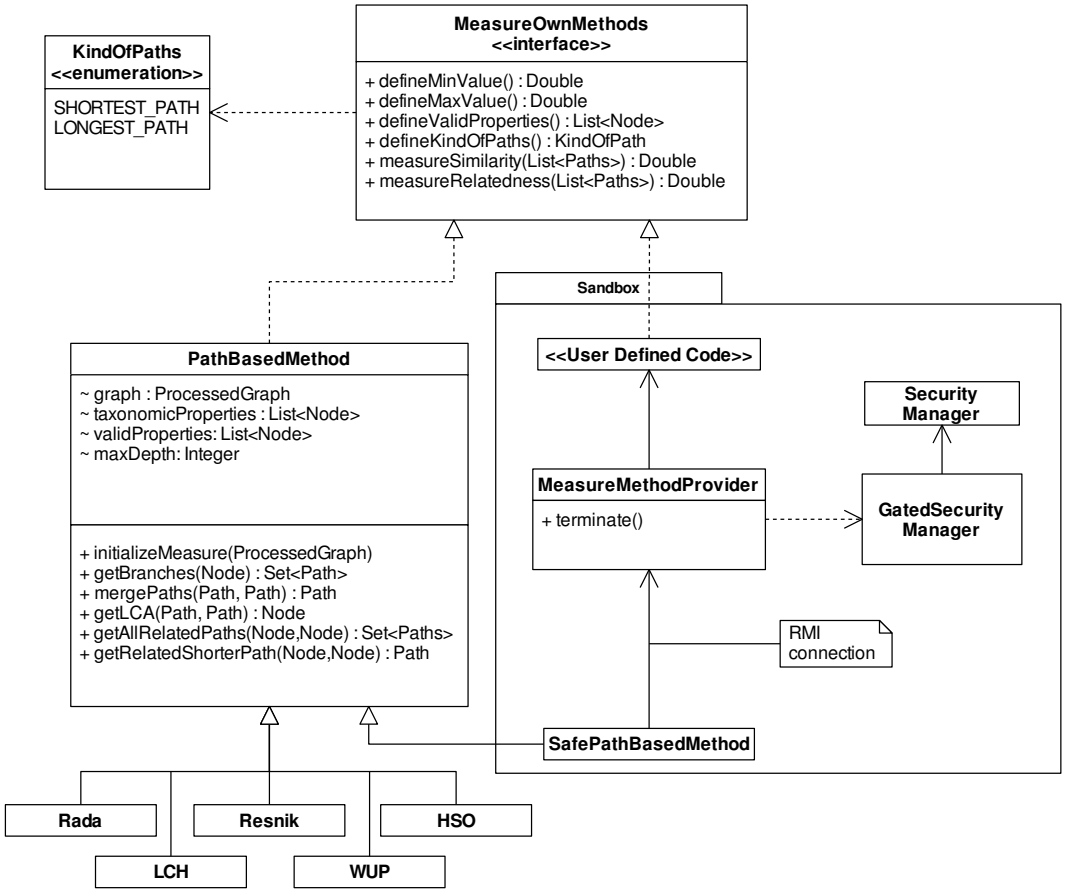


Figure 3 UML class diagram of the path-based semantic measures.

- the disambiguation strategy selects the pair of concepts (derived from the input words) that produces the best measure.

Figure 3 is an UML class diagram for the available methods. An important advantage of the SMComp design is the abstraction of common features used by all measures, exploiting the fact they are all path-based. This simplified the implementation of the literature methods and provided a skeleton to easily implement new measures with only a few lines of code.

All methods follow the same approach. The first step is to define the range of values, which is done by `defineMinValue()` and `defineMaxValue()`. If the measure considers a specific set of properties when computing relatedness, those must be declared using `defineValidProperties()`. A method can consider either the shortest path or a path of any size. To define which kind should be considered the `defineKindOfPaths()` is used. At last, the methods that actually implement the measures are `measureSimilarity` and `measureRelatedness`. The methods receive a list of `Path` and return a numerical value.

3.3 Sandbox

The Java code submitted by the user is compiled by back-end. Compilation errors detected at this stage are reported back to the client and shown on the user interface. For safety

reasons, the execution of this code is performed in a different Java Virtual Machine (JVM). Thus, the back-end can ensure termination of the submitted code.

The sandbox is the component responsible for the execution of submitted code. This component uses its own `Security Manager` to deny the access of user submitted code to any sensitive resources, such as the file system or the network, avoiding the execution of malicious code. It also enforces time limit on the execution of individual methods.

The communication between the back-end and the sandbox relies on Remote Method Invocation (RMI). The sandbox exposes all the methods defined by `MeasureOwnMethods` interface and also the `terminate()` method, which is invoked by the back-end whenever the `user-submitted code` can no longer be executed. The sandbox is also responsible for reporting back to the back-end, and are forward to the front-end, any timeout, execution errors or security violations.

3.4 Front-end

Figure 4 presents a screenshot with an edit view of the SMComp front-end. The web interface is a single window and requires no authentication. An user can select one of two modules: word pairs or datasets. This selection is available by using the tabs on the top of the window.

Both tabs have a similar layout. The major difference is in the area right bellow the tabs: in the word pairs view it has entries for the words to compare; in the datasets view it has a selector with the available benchmarks. The selector of knowledge source is also similar in both tabs. They also have similar grids where the computed results are displayed and a large button labelled "Compute" at the bottom.

All computation results are displayed in a two column grid. Each row corresponds to a semantic measure and the columns to the semantic type (left column is assigned to relatedness and the right column is assigned to similarity). The row and column headers have check boxes where the user selects which measures and variants he wants to compute. Bellow each row label, there is a button with a magnifying glass. When pressed, it shows a dialog box with the Java implementation of the measure. This should be useful as an example for the user implementation.

Bellow the *User Defined* label, the button icon shows a pencil. When pressed, the user enters on edit mode of its measure, as shown on Figure 4. This dialog provides a code editor, where he can enter a Java class with a new method, extending `PathBaseMethod`.

The assessment view using datasets is similar. The computation results are also displayed in a two column grid where each row corresponds to a semantic measure and each column to a semantic variant. The measure and its variant are selected using radio buttons. The available datasets are displayed in a list on the top. It is possible to upload a custom dataset by selecting *USER* from this list. In this case, a dialog box opens where values can be inserted using the format `word, word, value`. Clicking in the button labelled "Upload" dataset is sent to server and is ready to use in the assessment.

Use-case example

The code in Listing 1 is a simple example of the implementation of a user defined measure. The first step is to give a name to the class, which in this case is `UserDefinedMethodExample`. This class implements `MeasureOwnMethods` that provides a skeleton of the measure to complete.

The minimun and maximun numerical value of the measure must be defined. In this example, the minimum value is set to 0 and the maximum is set to the largest double. These

Figure 4 SMComp user interface.

were defined with `defineMinValue()` and `defineMaxValue()` respectively. The next step is to decide which properties should be considered when measuring relatedness since in similarity only *is-a* relationships are considered. If *all* should be considered, `defineValidProperties()` must return a empty list, which is the case.

It is also needed to define the size of paths. One can select between the shortest path between two elements or the longest path. This example uses the shortest path which is the enumerate value defined by `defineKindOfPaths()`.

After these initialization the user can define the measures itself. In this case, the measure takes in consideration the number of edges in the shortest path. The score is given by $1/(number\_of\_edges\_in\_the\_path + 1)$.

4 Validation

This work is part of a comprehensive research on semantic measures that aimed at understanding the misconceptions between similarity and relatedness. SMComp was developed to support its validation process. The detailed results obtained are available on [6].

Listing 1 User defined method example.

```
import pt.up.fc.dcc.SemArachne.datagraphTypes.*;
import pt.up.fc.dcc.SemArachne.methods.MeasureOwnMethods;
import pt.up.fc.dcc.SemArachne.methods.KindOfPaths;
import java.util.*;

public class UserDefinedMethodExample implements MeasureOwnMethods {
   @Override
   public double defineMinValue() {
      return 0.0;
   }

   @Override
   public double defineMaxValue() {
      return Double.MAX_VALUE;
   }

   @Override
   public List<Node<String>> defineValidProperties() {
      return new ArrayList<>();
   }

   @Override
   public KindOfPaths defineKindOfPaths() {
      return KindOfPaths.SHORTEST_PATH;
   }

   @Override
   public double measureSimilarity(List<Path<Node<String>>> paths) {
      int edges = paths.get(0).getProperties().size();
      return score = 1.0 / ((double) edges + 1);
   }

   @Override
   public double measureRelatedness(List<Path<Node<String>>> paths) {
      int edges = paths.get(0).getProperties().size();
      return score = 1.0 / ((double) edges + 1);
   }
}
```

The validation performed executed a cross evaluation using 10 different semantic measures (5 similarity and 5 relatedness) and 9 semantic datasets (4 similarity and 5 relatedness). The three versions of WordNet were used in the process.

Using WordNet 2.1, WUP and HSO similarity measures stood out since they had a better performance with similarity datasets. WS Sim dataset was correctly identified by all measures and MTurk-287 and MEN was always misidentified. With WordNet 3.0, WUP and HSO similarity measures stood out again. WS Sim was once again always identified as a similarity dataset. WordNet 3.1 only WUP stood out for similarity and HSO for relatedness. WS Rel was correctly identified by all semantic measures.

5 Conclusion

This work presents a testbed to compare, analyze and assess path-based semantic measures. SMComp is a novel approach for tools of this kind that couples GUI and API in a web interface. It provides three versions of a widely used semantic knowledge-base (WordNet); implementations of the most referenced path-based measures described in the literature; support for user-provided measures, coded with small snippets on the web interface; and evaluation of semantic measure quality using standard and custom datasets.

A validation of SMComp was performed during an experiment to compare similarity and relatedness measures and datasets. This experiment covered all the measures and datasets supported by SMComp. Using SMComp we were able to show that most of the semantic measures do not have the best performance with datasets of their type. It also allowed to pinpoint some measures and datasets more accurate, namely Wu & Palmer similarity measure and WS-Sim and WS-Rel datasets.

References

1 Eneko Agirre, Enrique Alfonseca, Keith Hall, Jana Kravalova, Marius Paşca, and Aitor Soroa. A study on similarity and relatedness using distributional and wordnet-based approaches. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, NAACL'09, pages 19–27, Stroudsburg, PA, USA, 2009. Association for Computational Linguistics. URL: `http://dl.acm.org/citation.cfm?id=1620754.1620758`.

2 Satanjeev Banerjee and Ted Pedersen. An adapted lesk algorithm for word sense disambiguation using wordnet. In *Computational linguistics and intelligent text processing*, pages 136–145. Springer, 2002.

3 Satanjeev Banerjee and Ted Pedersen. Extended gloss overlaps as a measure of semantic relatedness. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence*, IJCAI'03, pages 805–810, San Francisco, CA, USA, 2003. Morgan Kaufmann Publishers Inc. URL: `http://dl.acm.org/citation.cfm?id=1630659.1630775`.

4 Olivier Bodenreider, Marc Aubry, and Anita Burgun. Non-lexical approaches to identifying associative relations in the gene ontology. In *Pacific Symposium on Biocomputing. Pacific Symposium on Biocomputing*, pages 91–102. NIH Public Access, 2005.

5 Elia Bruni, Nam Khanh Tran, and Marco Baroni. Multimodal distributional semantics. *J. Artif. Int. Res.*, 49(1):1–47, January 2014. URL: `http://dl.acm.org/citation.cfm?id=2655713.2655714`.

6 Teresa Costa and José Paulo Leal. Semantic measures: How similar? How related? in print, 2016.

7 Christiane Fellbaum. *WordNet*. Wiley Online Library, 1999.

8 Lev Finkelstein, Evgeniy Gabrilovich, Yossi Matias, Ehud Rivlin, Zach Solan, Gadi Wolfman, and Eytan Ruppin. Placing search in context: The concept revisited. In *Proceedings of the 10th international conference on World Wide Web*, pages 406–414. ACM, 2001.

9 Yuriy Gorodnichenko and Gerard Roland. Understanding the individualism-collectivism cleavage and its effects: Lessons from cultural psychology. In Masahiko Aoki and Timur Kuran, editors, *Institutions and Comparative Economic Development*, volume 150, page 213. Palgrave Macmillan, 2012.

10 Guy Halawi, Gideon Dror, Evgeniy Gabrilovich, and Yehuda Koren. Large-scale learning of word relatedness with constraints. In *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD'12, pages 1406–1414, New York, NY, USA, 2012. ACM. `doi:10.1145/2339530.2339751`.

11 Sébastien Harispe, Sylvie Ranwez, Stefan Janaqi, and Jacky Montmain. The semantic measures library and toolkit: fast computation of semantic similarity and relatedness using biomedical ontologies. *Bioinformatics*, 30(5):740–742, 2014.

12 Sébastien Harispe, Sylvie Ranwez, Stefan Janaqi, and Jacky Montmain. Semantic similarity from natural language and ontology analysis. *Synthesis Lectures on Human Language Technologies*, 8(1):1–254, 2015.

13 Felix Hill, Roi Reichart, and Anna Korhonen. Simlex-999: Evaluating semantic models with (genuine) similarity estimation. *arXiv preprint arXiv:1408.3456*, 2014.

14 Graeme Hirst and David St-Onge. Lexical chains as representations of context for the detection and correction of malapropisms. *WordNet: An electronic lexical database*, 305:305–332, 1998.

15 Claudia Leacock and Martin Chodorow. Combining local context and wordnet similarity for word sense identification. *WordNet: An electronic lexical database*, 49(2):265–283, 1998.

16 Dekang Lin. An information-theoretic definition of similarity. In *Proceedings of the Fifteenth International Conference on Machine Learning*, ICML'98, pages 296–304, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc. URL: `http://dl.acm.org/citation.cfm?id=645527.657297`.

17 George A. Miller and Walter G. Charles. Contextual correlates of semantic similarity. *Language and Cognitive Processes*, 6(1):1–28, 1991. `doi:10.1080/01690969108406936`.

18 Siddharth Patwardhan, Satanjeev Banerjee, and Ted Pedersen. Using measures of semantic relatedness for word sense disambiguation. In *Proceedings of the 4th International Conference on Computational Linguistics and Intelligent Text Processing*, CICLing'03, pages 241–257. Springer-Verlag, Berlin, Heidelberg, 2003. URL: `http://dl.acm.org/citation.cfm?id=1791562.1791592`.

19 Ted Pedersen, Siddharth Patwardhan, and Jason Michelizzi. Wordnet:: Similarity: measuring the relatedness of concepts. In *Demonstration papers at HLT-NAACL 2004*, pages 38–41. Association for Computational Linguistics, 2004.

20 Giuseppe Pirró. A semantic similarity metric combining features and intrinsic information content. *Data & Knowledge Engineering*, 68(11):1289–1308, 2009.

21 Giuseppe Pirró and Jérôme Euzenat. A feature and information theoretic framework for semantic similarity and relatedness. In *The Semantic Web–ISWC 2010*, pages 615–630. Springer, 2010.

22 Giuseppe Pirró and Nuno Seco. Design, implementation and evaluation of a new semantic similarity metric combining features and intrinsic information content. In *On the Move to Meaningful Internet Systems: OTM 2008*, pages 1271–1288. Springer, 2008.

23 Roy Rada, Hafedh Mili, Ellen Bicknell, and Maria Blettner. Development and application of a metric on semantic nets. *Systems, Man and Cybernetics, IEEE Transactions on*, 19(1):17–30, 1989.

24 Kira Radinsky, Eugene Agichtein, Evgeniy Gabrilovich, and Shaul Markovitch. A word at a time: Computing word relatedness using temporal semantic analysis. In *Proceedings of the 20th International Conference on World Wide Web*, WWW'11, pages 337–346, New York, NY, USA, 2011. ACM. `doi:10.1145/1963405.1963455`.

25 Sylvie Ranwez, Vincent Ranwez, Jean Villerd, and Michel Crampes. Ontological distance measures for information visualisation on conceptual maps. In *On the Move to Meaningful Internet Systems 2006: OTM 2006 Workshops*, pages 1050–1061. Springer, 2006.

26 Philip Resnik. Using information content to evaluate semantic similarity in a taxonomy. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence – Volume 1*, IJCAI'95, pages 448–453, San Francisco, CA, USA, 1995. Morgan Kaufmann Publishers Inc. URL: `http://dl.acm.org/citation.cfm?id=1625855.1625914`.

27 Herbert Rubenstein and John B. Goodenough. Contextual correlates of synonymy. *Commun. ACM*, 8(10):627–633, 1965.

28 Nenad Stojanovic, Alexander Maedche, Steffen Staab, Rudi Studer, and York Sure. Seal: A framework for developing semantic portals. In *Proceedings of the 1st International Conference on Knowledge Capture*, K-CAP'01, pages 155–162, New York, NY, USA, 2001. ACM. doi:10.1145/500737.500762.

29 Michael Strube and Simone Paolo Ponzetto. Wikirelate! Computing semantic relatedness using Wikipedia. In *Proceedings of the 21st National Conference on Artificial Intelligence – Volume 2*, AAAI'06, pages 1419–1424. AAAI Press, 2006. URL: http://dl.acm.org/citation.cfm?id=1597348.1597414.

30 Zhibiao Wu and Martha Palmer. Verbs semantics and lexical selection. In *Proceedings of the 32nd Annual Meeting on Association for Computational Linguistics*, pages 133–138. Association for Computational Linguistics, 1994.

LLLR Parsing: a Combination of LL and LR Parsing

Boštjan Slivnik

University of Ljubljana, Faculty of Computer and Information Science, Ljubljana, Slovenia
bostjan.slivnik@fri.uni-lj.si

Abstract

A new parsing method called LLLR parsing is defined and a method for producing LLLR parsers is described. An LLLR parser uses an LL parser as its backbone and parses as much of its input string using LL parsing as possible. To resolve LL conflicts it triggers small embedded LR parsers. An embedded LR parser starts parsing the remaining input and once the LL conflict is resolved, the LR parser produces the left parse of the substring it has just parsed and passes the control back to the backbone LL parser. The LLLR(k) parser can be constructed for any LR(k) grammar. It produces the left parse of the input string without any backtracking and, if used for a syntax-directed translation, it evaluates semantic actions using the top-down strategy just like the canonical LL(k) parser. An LLLR(k) parser is appropriate for grammars where the LL(k) conflicting nonterminals either appear relatively close to the bottom of the derivation trees or produce short substrings. In such cases an LLLR parser can perform a significantly better error recovery than an LR parser since the most part of the input string is parsed with the backbone LL parser. LLLR parsing is similar to LL(*) parsing except that it (a) uses LR(k) parsers instead of finite automata to resolve the LL(k) conflicts and (b) does not perform any backtracking.

1998 ACM Subject Classification F.4.2 Grammars and Other Rewriting Systems (Parsing), D.3.1 Formal Definitions and Theory (Syntax), D.3.4 Processors (Parsing)

Keywords and phrases LL parsing, LR languages, left parse

Digital Object Identifier 10.4230/OASIcs.SLATE.2016.5

1 Introduction

As the syntax-directed translation represents the core of virtually any modern compiler's front-end, the parser as the implementation of the syntax analysis form an important part of any compiler. Hence, the parser should provide the compiler writer with (a) a solid framework for evaluation of semantic actions during or after parsing and (b) an appropriate support for producing detailed syntax error reports. The two predominant techniques for parsing programming languages are the top-down and the bottom-up parsing. LL parsers represent the core of the first group and LR parsers the core of the second. Both techniques have their advantages and disadvantages. The top-down parsing provides "the readability of recursive descent (RD) implementations of LL parsing along with the ease of semantic action incorporation" while "an LL parser is linear in the size of the grammar"; the bottom-up parsing is regarded highly for "the extended parsing power of LR parsers, in particular the admissibility of left recursive grammar" but "even LR(0) parse tables can be exponential in the size of the grammar" (all quotes are from [12]). The parsers incorporating both top-down and bottom-up parsing (left-corner parsers, etc. [2, 4, 5, 7]) never gained much popularity because of the confusing order in which the semantic actions are triggered.


© Boštjan Slivnik;
licensed under Creative Commons License CC-BY

▶ **Example 1.** Much older XLC(1) and LAXLC(1) parsers [4, 5] are extensions of left-corner parsing and (like LLLR parsing) employ both top-down and bottom-up approaches. However, XLC(1) and LAXLC(1) parsers produce neither left nor right trace of the input string. For instance, if string *abbbbccc* is parsed using an XLC(1) or an LAXLC(1) parser for grammar

$$[A \longrightarrow aBbC], [B \longrightarrow Bb], [B \longrightarrow b], [C \longrightarrow Cc], \text{ and } [C \longrightarrow c],$$

the output

$$[A \longrightarrow aBbC][B \longrightarrow b][B \longrightarrow Bb][B \longrightarrow Bb][C \longrightarrow c][C \longrightarrow Cc][C \longrightarrow Cc]$$

is neither left nor (reverse) right parse of the input string. As the first production printed out expands the start symbol at the top of the derivation tree the evaluation of semantic routines starts at the top. But then the evaluation of semantic routines suddenly changes from the top-down into the bottom up approach since the second production printed out produces the leaf several levels below the root of the tree. Later the evaluation changes direction from bottom-up to top-down and vice-versa a few more times.

In general, during XLC(1) and LAXLC(1) parsing the evaluation of semantic routines might become quite confusing as it changes direction all the time. ◀

Likewise, Packrat parsers [3] used for Parsing Expression Grammars, have not become popular as "... it can be quite difficult to determine what language is defined by a TDPL program." [1]. In recent years the focus has shifted strongly towards the top-down parsing and towards LL parsing in particular [11, 18, 13, 12, 8, 16, 17] – even to the point that yacc was erroneously considered dead [6]. On one hand, a GLL parser capable of parsing a language of any context-free grammar, was formulated [12], but in the worst case it runs in cubic time. Furthermore, an LL(*) parser using (a) finite automata to resolve certain LL(k) conflicts and (b) backtracking to resolve the others, has been implemented in ANTLR [8]. Furthermore, "the LL(*) grammar condition is statically undecidable and grammar analysis sometimes fails to find regular expressions that distinguish between alternative productions" [9].

On the other hand, LR parsers were modified to produce the left parse of its input and thus giving the compiler writer the impression of top-down parsing [11, 18]. However, these parsers either (a) produce the first production of the left parse (and thus trigger the first semantic actions) only after the entire input string has been parsed [11] or (b) they further increase the difference of how much space LR or LL parser needs by introducing two additional parsing tables [18]. One could also use a much stronger GLR parsing but the time complexity is O(np+1) (where p is the length of the longest production). For deterministic grammars, however, both GLL and GLR parsers should run in near-linear time [9].

Finally, ALL(*) parsing performs grammar analysis in parse-time to combine "the simplicity, efficiency, and predictability of conventional top-down LL(k) parsers with the power of a GLR-like mechanism to make parsing decisions" [9]. It achieves almost linear time for majority of grammars used in practice where it is consistently faster than GLL and GLR parsing and can compete with LL or LALR parsing. Nevertheless, its worst case complexity is $O(n^4)$.

The idea behind LLLR parsing is to produce the left parse of the input string by using LL parsing as much as possible and to use small LR parsers only to avoid LL conflicts [19]. Hence, LLLR parsing uses LR parsers where LL(*) and ALL(*) parsing use deterministic finite automata "even though lookahead languages (set of all possible remaining input phrases) are often context-free" [9].

Table 1 LLLR(1) parsing of $bbbbaab \in L(G_{ex2})$. Whenever the LR(1) parser for A stops, it handles the remaining part of the right side of production for A, i.e., symbols that has not yet appeared on the stack as the input has not yet been reduced to these symbols, back to the backbone LL(1) parser (bA in the first instance and a in the second).

	STACK	INPUT	ACTION	OUTPUT
1	$\$S$	$bbbbaab\$$	LL produce $[S \longrightarrow bBab]$	$[S \longrightarrow bAab]$
2	$\$baAb$	$bbbbaab\$$	LL shift b	
3	$\$baA$	$bbbaab\$$	— start the LR parser for A	
4	$\$baq_0$	$bbbaab\$$	LR shift b	
5	$\$baq_0bq_1$	$bbaab\$$	LR reduce on $[B \longrightarrow b]$	
6	$\$baq_0Bq_1$	$bbaab\$$	— stop the LR parser for A	$[A \longrightarrow BbA][B \longrightarrow b]$
7	$\$baAb$	$bbaab\$$	LL shift b	
8	$\$baA$	$baab\$$	— start the LR parser for A	
9	$\$baq_0$	$baab\$$	LR shift b	
10	$\$baq_0bq_1$	$aab\$$	— stop the LR parser for A	$[A \longrightarrow ba]$
11	$\$baa$	$aab\$$	LL shift a	
12	$\$ba$	$ab\$$	LL shift a	
13	$\$b$	$b\$$	LL shift b	
14	$\$$	$\$$	LL accpet	

▶ **Example 2.** Consider the grammar $G_{ex2} \in \text{LR}(1) \setminus \text{LL}(1)$ with the start symbol S and productions

$$[S \longrightarrow bAab], [A \longrightarrow ba], [A \longrightarrow BbA], \text{ and } [B \longrightarrow b].$$

The trace of LLLR(1) parsing of string $bbbbaab \in L(G_{ex2})$ is shown in Table 1. LLLR(1) parsing starts with LL(1) parsing, but to avoid the LL(1) conflict on b for A (line 3), the embedded left LR(k) parser for A is started (line 4). After the 2nd b of the input string is shifted and reduced to B using production $[B \longrightarrow b]$, the 3rd b appears in the lookahead buffer and the embedded left LR(k) parser deduces that productions $[A \longrightarrow bbA]$ and $[B \longrightarrow b]$ are part of the left parse if the input string. It prints out both productions and passes the control back to the backbone LL(1) parser together with suffix bA of the right side of $[A \longrightarrow bbA]$ as these two symbols have not yet been handled by LR(1) parser yet (line 6).

The same pattern repeats in lines 8, 9 and 10 except that the production $[A \longrightarrow ba]$ is printed out and that only suffix a is returned to the backbone LL(1) parser.

The mechanism that enables the embedded LR(1) parser to deduce which production expanding A must be used in either case is explained in [16], the procedure for printing out the left parse instead of the right parse during LR parsing has been described in [11]. ◀

Example 2 contains a simple case that can be handled by, for instance, LL(*) parsing. Harder cases, e.g., (chained) left-recursive nonterminals, etc., must be parsed using LLLR parsing which can be used for parsing any language generated by an LR(k) grammar. The paper is organized in the way an LLLR parser is constructed. After Section 2, which provides the reader with some preliminary issues, Section 3 introduces different kinds of conflicts that can appear during LLLR parsing. Section 4 describes the construction of the LLLR parser while Section 5 provides a method used for eliminating redundant entries in the generated parser tables. Before Conslusions, another section is dedicated to a test case – an illustration how LLLR parsing performs on the grammar for the Java programming language.

2 Preliminaries

An intermediate knowledge of LL and LR parsing is presumed and the standard terminology of formal language theory and parsing is assumed [14, 15]. In addition, let

$$\text{FSTFLW}_k^G(\alpha, A) = \cup_{z \in \text{FOLLOW}_k^G(A)} \text{FIRST}_k^G(\alpha z)$$

and let $T^{*k} = T^0 \cup T^1 \cup \ldots \cup T^k$.

The LLLR parser is composed of (a) an LL parser based on the SLL parsing table and (b) the embedded left LR(k) parser. For the lack of space, only a short overview of both methods are given here.

2.1 The SLL(k) parsing

An LL(k) grammar $G = \langle N, T, P, S \rangle$ can be transformed to an equivalent SLL(k) grammar $\bar{G} = \langle \bar{N}, T, \bar{P}, \bar{S} \rangle$ [15] where the set of nonterminals is defined as

$$\bar{N} = \{ \langle A, \mathcal{F}_A \rangle \,;\, S \Longrightarrow_{\text{lm}}^* uA\delta \wedge \mathcal{F}_A = \text{FIRST}_k^G(\delta) \}$$

and the initial symbol is $\bar{S} = \langle S, \{\varepsilon\} \rangle$; for any nonterminal $\langle A, \mathcal{F}_A \rangle \in \bar{N}$ and any production $[A \longrightarrow X_1 X_2 \ldots X_n] \in P$ the grammar $\bar{G}$ contains production

$$[\langle A, \mathcal{F}_A \rangle \longrightarrow \bar{X}_1 \bar{X}_2 \ldots \bar{X}_n]$$

where

$$\bar{X}_i = \begin{cases} X_i & X_i \in T \\ \langle X_i, \text{FIRST}_k^G(X_{i+1} X_{i+2} \ldots X_n \mathcal{F}_A) \rangle & X_i \in N \end{cases} .$$

Regardless of whether $G \in LL(k)$), (a) $L(\bar{G}) = L(G)$, and (b) every nonterminal A of $\bar{G}$ appears in exactly one right-context $\mathcal{F}$(substrings derived from the nonterminal A are always followed by strings from $\mathcal{F}$). However, $G \in \text{LL}(k) \Longleftrightarrow \bar{G} \in \text{SLL}(k)$.

The most straightforward method for producing the LL(k) parser for an \$-augmented SLL(k) grammar $G = \langle N, T, P, S \rangle$ is a construction of the LL table

$$\text{LL-TABLE}\colon N \times T^{*k} \longrightarrow 2^P$$

where LL-TABLE(A, x) is defined as

$$\{ [A \longrightarrow \alpha] \,;\, \forall [A \longrightarrow \alpha] \in P \colon x \in \text{FSTFLW}_k^G(\alpha, A) \} \quad .$$

The cardinality of LL-TABLE(A, x) indicates different issues. If $|\text{LL-TABLE}(A, x)| = 1$, the LL($k$) table indicates the regular selection of the next production that is to be used during LL(k) parsing when the nonterminal $A \in N$ is on the top of the parser stack and $x \in T^{*k}$ is in the lookahead buffer. $|\text{LL-TABLE}(A, x)| = 0$ indicates the syntax error while $|\text{LL-TABLE}(A, x)| > 1$ represents the LL(k) conflict.

It is assumed that prior the construction of an LLLR(k) parser, a grammar is transformed by the LL-to-SLL transformation described above.

2.2 The embedded left LR(k) parser

Given a grammar $G = \langle N, T, P, S \rangle$, the embedded left LR($k$) grammar for $\alpha \in (N \cup T)^*$ and $\mathcal{F} \in (T \cup \{\$\})^{*k}$ [16, 17] is an LR(k) parser that

1. expects a string starting with a prefix wz, where $z \in \mathcal{F}$ and $\alpha \Longrightarrow_G^* w$, on its input;
2. parses a prefix u of w, i.e., $w = uv$ for some v, without shifting any symbol of z on the stack;
3. returns the left parse $\pi_u \in P^*$ and the viable suffix $\delta^R \in (N \cup T)^*$ found in the derivation

$$\alpha z \Longrightarrow_{G,\mathrm{lm}}^{\pi_u} u\delta z.$$

The embedded left LR(k) parser is the left LR(k) parser [18] for the (reduced) grammar $G(\alpha, \mathcal{F}) = \langle N \cup \{S_1, S_2\}, T, P_{\alpha,\mathcal{F}}, S_1\rangle$ where $S_1, S_2 \notin (N \cup T)$ and

$$P_{\alpha,\mathcal{F}} = P \cup \{[S_1 \longrightarrow S_2 z], [S_2 \longrightarrow \alpha]\,;\, z \in \mathcal{F}\}.$$

The trick is that, if the parser does not stop earlier, the reduction on $[S_2 \longrightarrow \alpha]$ is regarded as a signal to accept the input string w leaving the string z in the lookahead buffer. Furthermore, if $G \in \mathrm{LR}(k)$, then testing whether $G(\alpha, \mathcal{F})$ can be done very efficiently [17].

The embedded left LR(k) parser is called 'embedded' since it parses only a substring of the entire input string and can thus be embedded into other parsers. It is called 'left' because it produces the left parse (instead of the right parse) of the substring it parsed.

3 Identifying conflicts

Computing a set of LL(k) conflicts for a given \$-augmented grammar $G = \langle N, T, P, S\rangle$ is straightforward, but it gets slightly more complicated if these conflicts are to be resolved using small LR(k) parsers embedded into the backbone LL(k) parser.

Suppose that a string $\$uv\$ \in L(G)$ is derived by the leftmost derivation

$$S \Longrightarrow_{G,\mathrm{lm}}^* uB\delta \Longrightarrow_{G,\mathrm{lm}} u\beta_1 A\beta_2\delta \Longrightarrow_{G,\mathrm{lm}}^* uv$$

and that the grammar exhibits an LL(k) conflict on the lookahead string $x \in \mathrm{FIRST}_k^G(A\beta_2\delta\$)$ for the nonterminal symbol A. However, as demonstrated by Example 3, the correct termination of the LR(k) parser for the substrings derived from A is not always guaranteed.

▶ **Example 3.** Consider the grammar $G_{\mathrm{ex3}} \in \mathrm{LR}(1) \setminus \mathrm{LL}(1)$ with the start symbol S' and productions

$$[S \longrightarrow bBab], [B \longrightarrow A], [A \longrightarrow b] \text{ and } [A \longrightarrow Ba].$$

The trace of parsing the input string starting with bba using the backbone LL(1) parser and the embedded LR(1) parser for the LL(1) conflicting nonterminal A is shown in Table 2. The configuration $\$baA|ba\ldots\$$ results in the LL(1) conflict on b for A where the embedded LR(1) parser for A is started (line 4). However, after pushing b and reducing it to A the LR(1) parser cannot determine whether the first a of $bba\ldots$ is derived from A or not (line 7). Namely, if the input string is $bbab$, the LR(1) parser must accept, terminate and handle the control back to the backbone LL(1) parser; otherwise the LR(1) parser must perform the reduction on $[B \longrightarrow A]$ and continue parsing.

As shown in Table 3, the problem remains even if the embedded LR(1) parser for B is used instead (since A is initially derived from B). However, the embedded LR(1) parser for parsing substrings derived from Ba and followed by $b\$$ can terminate correctly and should be used. ◀

Table 2 Parsing a string starting with bba using an LL(1) parser for G_{ex3} and an LR(1) parser for nonterminal symbol A.

	STACK	INPUT	ACTION
1	$\$S$	$bba\ldots\$$	LL produce $[S \longrightarrow bBab]$
2	$\$baBb$	$bba\ldots\$$	LL shift b
3	$\$baB$	$ba\ldots\$$	LL produce $[B \longrightarrow A]$
4	$\$baA$	$ba\ldots\$$	— start a parser for A —
5	$\$baq_0$	$ba\ldots\$$	LR shift b
6	$\$baq_0bq_1$	$a\ldots\$$	LR reduce on $[A \longrightarrow a]$
7	$\$baq_0Aq_2$	$a\ldots\$$	LR reduce on $[B \longrightarrow a]$ or accept?

Table 3 Parsing a string starting with bba using an LL(1) parser for G_{ex3} and an LR(1) parser for nonterminal symbol B.

	STACK	INPUT	ACTION
1	$\$S$	$baa\ldots\$$	LL produce $[S \longrightarrow bBab]$
2	$\$baBb$	$baa\ldots\$$	LL shift b
3	$\$baB$	$aa\ldots\$$	— start a parser for B —
4	$\$baq_0$	$aa\ldots\$$	LR shift a
5	$\$baq_0aq_1$	$a\ldots\$$	LR reduce on $[A \longrightarrow a]$
6	$\$baq_0Aq_2$	$a\ldots\$$	LR reduce on $[B \longrightarrow a]$
7	$\$baq_0Bq_2$	$a\ldots\$$	LR shift a or accept?

Example 3 shows that there are two kinds of conflicting nonterminals in LLLR(k) parsing:

1. *Genuine conflicting nonterminals* are all SLL(k) conflicting nonterminals (symbol A in Example 3).
2. *Induced conflicting nonterminals* are all nonterminals which are not SLL(k) conflicting but must be treated as conflicting so that the embedded LR(k) parser can terminate correctly (symbol B in Example 3).

Algorithm 1 computes the set of all LLLR conflicting nonterminals. It starts with computing genuine conflicting nonterminals (set $C^{(0)}$ in lines 2–3). Afterwards, it adds all induced conflicting nonterminals: if A is a conflicting nonterminal in production $[B \longrightarrow \beta_1 A \beta_2]$, then there should exist a prefix β_2' of β_2 so that the embedded LR(k) parser for $A\beta_2'$ stops correctly – if β_2' does not exist, then A induces B and B is added to the set of conflicting nonterminals (lines 6–9). Finally, the set C contains all conflicting nonterminals (while $C \setminus C^{(0)}$ contains all genuine conflicting nonterminals).

The most time consuming part of Algorithm 1 is testing whether the embedded LR(k) parser can always stop – the condition in line 9. Fortunately, if the grammar G for which the LLLR(k) parser is being made is an LR(k) grammar, the test can be performed quite efficiently [17].

4 Constructing the LLLR parser

After the conflicting nonterminals have been computed, the parsing table of the backbone LL(k) parser can be computed using Algorithm 2. The basic idea is simple: replace all (genuine or induced) conflicting nonterminals appearing on the right side of any production with markers which trigger the appropriate embedded left LR(k) parsers.

Algorithm 1 Evaluation of LLLR conflicting nonterminals.
INPUT: A grammar G after the LL-to-SLL transformation.
OUTPUT: A set $C^{(0)}$ of genuine conflicting nonterminals and a set C of all conflicting nonterminals.

1: $i \leftarrow 0$
2: $C^{(0)} \leftarrow \{A\,;\, \exists[A \longrightarrow \alpha_1], [A \longrightarrow \alpha_2] \in P\colon \mathrm{FSTFLW}_k^G(\alpha_1,A) \cap \mathrm{FSTFLW}_k^G(\alpha_2,A) \neq \emptyset\}$
3: **repeat**
4: $\quad i \leftarrow i+1$
5: $\quad C^{(i)} \leftarrow C^{(i-1)} \cup$
6: $\qquad \{B\,;\, \exists[B \longrightarrow \beta_1 A \beta_2] \in P, A \in C^{(i-1)}\colon$
7: $\qquad\quad \forall \beta_2', \beta_2''\colon \big(\beta_2'\beta_2'' = \beta_2 \Longrightarrow G(A\beta_2', \mathrm{FSTFLW}_k^G(\beta_2'',B)) \notin \mathrm{LR}(k)\big)\}$
8: **until** $C^{(i)} = C^{(i-1)}$
9: $C = C^{(i)}$

More precisely, if A is the only conflicting nonterminal in $[B \longrightarrow \beta_1 A \beta_2]$, then

$$\text{LL-TABLE}(B,x) = [B \longrightarrow \beta_1 [\![A\beta_2', \mathrm{FSTFLW}_k^G(\beta_2'', B)]\!] \beta_2'']$$

where β_2' is the shortest prefix of $\beta_2 = \beta_2'\beta_2''$ such that the embedded left LR(k) parser for $A\beta_2'$ can stop on any string $x \in \mathrm{FSTFLW}_k^G(\beta_2'', B)$. The newly created nonterminal symbol $[\![A\beta_2', \mathrm{FSTFLW}_k^G(\beta_2'', B)]\!]$ acts as a trigger for starting the embedded left LR(k) parser. If there are more conflicting nonterminals on the right hand side of a production, then the described substitution is performed from left to right. Furthermore, the embedded left LR(k) grammar for $G(A\beta_2', \mathrm{FSTFLW}_k^G(\beta_2'', B))$ must be constructed regardless of whether B is a conflicting nonterminal or not.

The left-to-right scan of the sentential form on the right side of a production being transformed is performed in lines 3–17 of Algorithm 2; the computation of the shortest sentential form a particular embedded left LR(k) parser is to be made for, is computed in lines 8–15.

There are two key insights into the correctness of Algorithm 2: (a) the loop in lines 8–15 terminates successfully, i.e., by finding an appropriate prefix $X_i X_{i+1} \ldots X_j$ and exiting after reaching line 11, and (b) no element of LL-TABLE contains more than one production. For if either of these two statements were false, the nonterminal A should have been considered a conflicting nonterminal by Algorithm 1.

▶ **Example 4.** Consider a grammar G_{ex4} with the start symbol S and productions

$[S \longrightarrow aAabCb]$,
$[A \longrightarrow aEB]$, $[A \longrightarrow Aaa]$, $[C \longrightarrow aaD]$, $[C \longrightarrow abE]$,
$[B \longrightarrow bb]$, $[D \longrightarrow aa]$, $[E \longrightarrow a]$ and $[E \longrightarrow Aa]$.

As symbols A, C and E are conflicting nonterminals, the LL(1) parsing table contains the following entries:

$$\begin{aligned}
\text{LL-TABLE}(S,a) &= [S \longrightarrow a[\![Aa, \{b\}]\!] b [\![C, \{b\}]\!] b] \\
\text{LL-TABLE}(B,b) &= [B \longrightarrow bb] \\
\text{LL-TABLE}(D,a) &= [D \longrightarrow aa]
\end{aligned}$$

The backbone LL(1) parser is augmented with the embedded left LR(1) parsers for the following grammars:

$G(Aa, \{b\})$, $G(C, \{b\})$ and $G(E, \{b\})$.

Algorithm 2 Construction of the backbone LL(k) parser.
INPUT: A grammar G after the LL-to-SLL transformation and a set C of all conflicting nonterminals for G.
OUTPUT: The LL parsing table for the backbone LL parser.

```
1: for [A ⟶ X1X2 ... Xn] ∈ P ∧ A ∉ C do
2:   α ← ε ; i ← 1
3:   while (i ≤ n) do
4:     if Xi ∉ C then
5:       α ← αXi ; i ← i + 1
6:     else
7:       j ← i
8:       while (i ≤ j) do
9:         F = FSTFLW_k^G(Xj+1 ... Xn, A)
10:        if G(Xi ... Xj, F) ∈ LR(k) then
11:          α ← α⟦Xi ... Xj, F⟧ ; i ← j + 1
12:        else
13:          j ← j + 1
14:        end if
15:      end while
16:    end if
17:  end while
18:  for x ∈ FSTFLW_k^G(X1 ... Xn, A) do
19:    LL-TABLE(A, x) ← {[A ⟶ α]}
20:  end for
21: end for
```

LLLR(1) parsing of string *aaaabbaaabaaaab* is shown in Table 4:

1. The parsing starts with printing out the the first production of the left parse, i.e., $[S \longrightarrow aAa\ bCb]$ (line 1) and shifting the first symbol.
2. When $[\![Aa, \{b\}]\!]$ appears on the top of the stack, the embedded LR(1) parser for $G(Aa, \{b\})$ is started. While parsing the substring *aaabbaaa* derived from Aa, the left parse $\pi_{Aa} = [A \longrightarrow Aaa][A \longrightarrow aEB][E \longrightarrow Ea][E \longrightarrow a][B \longrightarrow bb]$ is accumulated on the stack (lines 3–17). When b is in the lookahead buffer, the left parse π_{Aa} is printed out and the viable suffix ε returned to the backbone LL(1) parser (line 17).
3. After shifting b, the embedded parser for $G(C, \{b\})$ is started. After shifting a, the lookahead buffer contains another a and the parser recognizes the production $[C \longrightarrow aaD]$ (line 22). It prints this production out and returns the viable suffix aD – the part of the production not yet matched against the input.
4. Afterwards, the backbone parser finishes the job.

Hence, the LLLR(1) parser prints the left parse

$$[S \longrightarrow aAabCb][A \longrightarrow Aaa][A \longrightarrow aEB]$$
$$[E \longrightarrow Ea][E \longrightarrow a][B \longrightarrow bb][C \longrightarrow aaD].$$

As the symbols A and E are left-recursive, the parsers for $G(Aa, \{b\})$ and $G(E, \{b\})$ must always parse the entire string generated by either grammar. However, the parser for $G(C, \{b\})$ always shifts only the first a and returns the viable suffix aD (if a is in the lookahead buffer) or the viable suffix $b[\![E, \{b\}]\!]$ (if b is in the lookahead buffer).

Table 4 Parsing a string starting with bba using an LL(1) parser for G_{ex3} and an LR(1) parser for B. (All LR(1) states of both embedded left LR(1) parsers are presented generically with q; the subscripts denote left parses accumulated on stack during left LR parsing [11].)

	STACK	INPUT	ACTION
1	$\$S$	$aaaabbaaabaaaab\$$	LL produce $[S \longrightarrow a[\![Aa, \{b\}]\!]b[\![C, \{b\}]\!]b]$
2	$\$b[\![C]\!]b[\![Aa]\!]a$	$aaaabbaaabaaaab\$$	LL shift a
3	$\$b[\![C, \{b\}]\!]b[\![Aa, \{b\}]\!]$	$aaabbaaabaaaab\$$	LL starts the LR parser for grammar $G(Aa, \{b\})$
4	$\$b[\![C, \{b\}]\!]b\langle q_\varepsilon\rangle$	$aaabbaaabaaaab\$$	LR shift a
5	$\$b[\![C, \{b\}]\!]b\langle q_\varepsilon\rangle a\langle q_\varepsilon\rangle$	$aabbaaabaaaab\$$	LR shift a
6	$\$b[\![C, \{b\}]\!]b\langle q_\varepsilon\rangle a\langle q_\varepsilon\rangle a\langle q_\varepsilon\rangle$	$abbaaabaaaab\$$	LR reduce on $p_1 = [E \longrightarrow a]$
7	$\$b[\![C, \{b\}]\!]b\langle q_\varepsilon\rangle a\langle q_\varepsilon\rangle E\langle q_{p_1}\rangle$	$abbaaabaaaab\$$	LR shift a
8	$\$b[\![C, \{b\}]\!]b\langle q_\varepsilon\rangle a\langle q_\varepsilon\rangle E\langle q_{p_1}\rangle a\langle q_\varepsilon\rangle$	$bbaaabaaaab\$$	LR reduce on $p_2 = [E \longrightarrow Ea]$
9	$\$b[\![C, \{b\}]\!]b\langle q_\varepsilon\rangle a\langle q_\varepsilon\rangle E\langle q_{p_2p_1}\rangle$	$bbaaabaaaab\$$	LR shift b
10	$\$b[\![C, \{b\}]\!]b\langle q_\varepsilon\rangle a\langle q_\varepsilon\rangle E\langle q_{p_2p_1}\rangle b\langle q_\varepsilon\rangle$	$baaabaaaab\$$	LR shift b
11	$\$b[\![C, \{b\}]\!]b\langle q_\varepsilon\rangle a\langle q_\varepsilon\rangle E\langle q_{p_2p_1}\rangle b\langle q_\varepsilon\rangle b\langle q_\varepsilon\rangle$	$aaabaaaab\$$	LR reduce on $p_3 = [B \longrightarrow bb]$
12	$\$b[\![C, \{b\}]\!]b\langle q_\varepsilon\rangle a\langle q_\varepsilon\rangle E\langle q_{p_2p_1}\rangle B\langle q_{p_3}\rangle$	$aaabaaaab\$$	LR reduce on $p_4 = [A \longrightarrow aEB]$
13	$\$b[\![C, \{b\}]\!]b\langle q_\varepsilon\rangle A\langle q_{p_4p_2p_1p_3}\rangle$	$aaabaaaab\$$	LR shift a
14	$\$b[\![C, \{b\}]\!]b\langle q_\varepsilon\rangle A\langle q_{p_4p_2p_1p_3}\rangle a\langle q_\varepsilon\rangle$	$aabaaaab\$$	LR shift a
15	$\$b[\![C, \{b\}]\!]b\langle q_\varepsilon\rangle A\langle q_{p_4p_2p_1p_3}\rangle a\langle q_\varepsilon\rangle a\langle q_\varepsilon\rangle$	$abaaaab\$$	LR reduce on $p_5 = [A \longrightarrow Aaa]$
16	$\$b[\![C, \{b\}]\!]b\langle q_\varepsilon\rangle A\langle q_{p_5p_4p_2p_1p_3}\rangle$	$abaaaab\$$	LR shift a
17	$\$b[\![C, \{b\}]\!]b\langle q_\varepsilon\rangle A\langle q_{p_5p_4p_2p_1p_3}\rangle a\langle q_\varepsilon\rangle$	$baaaab\$$	LR stops: lm parse $= p_5p_4p_2p_1p_3$ remaining suffix $= \varepsilon$
18	$\$b[\![C, \{b\}]\!]b$	$baaaab\$$	LL shift b
20	$\$b[\![C, \{b\}]\!]$	$aaaab\$$	LL starts the parser for grammar $G(C, \{b\})$
21	$\$b\langle q_\varepsilon\rangle$	$aaaab\$$	LR shift a
22	$\$b\langle q_\varepsilon\rangle a\langle q_\varepsilon\rangle$	$aaab\$$	LR stops: lm parse $= p_4$ remaining suffix Da
23	$\$bDa$	$aaab\$$	LL shift a
24	$\$bD$	$aab\$$	LL produce $[D \longrightarrow aa]$
25	$\$baa$	$aab\$$	LL shift a
26	$\$ba$	$ab\$$	LL shift a
27	$\$b$	$b\$$	LL shift b
28	$\$$	$\$$	LL accept `:-)`

Finally, note that not all LL-TABLE entries generated by Algorithm 2 are needed – LL-TABLE(B, b) is not needed as substrings derived from B will always be parsed by a parser for $G(Aa, \{b\})$. Likewise, if C is eliminated from the grammar, the parser for $G(E, \{b\})$ is not needed anymore for the same reason. ◀

An LLLR(k) parser can be constructed for any LR(k) grammar G: if not otherwise, the start symbol gets included into the set of induced conflicting symbols and hence the entire input string is parse by a single embedded left LR(k) parser. However, this is not the advised use of LLLR parsing.

5 Reducing the LLLR Parser

As illustrated by Example 4, the construction of the LLLR parser described in Section 4 might produce some redundant LL-TABLE entries or even some redundant embedded parsers.

Since it it obvious that LL-TABLE entries for the start symbol of the grammar are not redundant, it is trivial to compute what entries and parsers are needed by the backbone LL(k) parser and thus the rest of this section is dedicated to identifying all symbols that can form a viable suffix returned by an embedded left LR(k) parser. Once this symbols are known, it is again straightforward to see which entries and parsers are needed on the basis of embedded parsers.

The key insight comes from the understanding of how the embedded left LR(k) parser computes the viable suffix. Namely, that parser uses an additional parsing table LEFT that maps certain parser states and lookahead strings to LR(0) items. More precisely, if

$$\text{LEFT}(q, x) = [A \longrightarrow \alpha \bullet \beta],$$

then for each viable prefix γ the state q is associated with, i.e., for every LR(k) stack contents where q is at the top, there exists a single γ-path of items in the nondeterministic LR(0) machine starting with the initial LR(0) item and terminating with item $[A \longrightarrow \alpha \bullet \beta]$ [18, 17]. The viable suffixes consist from the sentential forms found on the right side of $\bullet$ in this LR(0) items and thus these sentential forms must be checked to find symbols that can form a viable suffix returned by an embedded left LR(k) parser. This in fact is done by Algorithm 3 which must be run for every embedded parser.

6 A test case: the Java Language

To test the new parsing method against the real programming language, Java 1.0 has been chosen. There are two reasons for this: (a) there is an official LALR grammar for Java 1.0 available (Gosling et. al, The Java™ Language Specification, 1996), and (b) choosing a language primarily made for parsing in a top-down manner would be unfair.

As the official Java 1.0 grammar is LALR, it contains a lot of left-recursive nonterminals. However, one must distinguish between two kinds of left-recursive nonterminals:

1. *Essential left-recursive nonterminals* are those where left recursion is used to achieve the proper form of derivation trees.

   The best examples of essential left-recursive nonterminals are those describing arithmetic expressions as the left recursion is needed to emphasize the left associativity of various arithmetic operators. For instance, productions

   AdditiveExpression ⟶ MultiplicativeExpression
   AdditiveExpression ⟶ AdditiveExpression + MultiplicativeExpression
   AdditiveExpression ⟶ AdditiveExpression – MultiplicativeExpression

Algorithm 3 Enumeration of all symbols that can form a viable suffix returned by an embedded left LR(k) parser.
INPUT: An embedded left LR(k) parser.
OUTPUT: A set of symbols $\mathcal{L}$.

enumerate all items needed to compute the viable suffix:
1: $i \leftarrow 0$
2: $\mathcal{I}^{(0)} \leftarrow \{\ \langle [A \to \alpha \bullet \beta, x], q \rangle\ ;$
3: $\quad \mathrm{LEFT}(q, z) = [A \to \alpha \bullet \beta]$
4: $\quad \wedge\, [A \to \alpha \bullet \beta, x] \in q$
5: $\quad \wedge\, z \in \mathrm{FIRST}_k^G(\beta x)\ \}$
6: **repeat**
7: $\quad i \leftarrow i + 1$
8: $\quad \mathcal{I}^{(i)} \leftarrow \mathcal{I}^{(i-1)} \cup \{\ \langle [A' \to \alpha' \bullet A\beta', x'], q' \rangle\ ;$
9: $\quad \langle [A \to \alpha \bullet \beta, x], q \rangle \in \mathcal{I}^{(i-1)}$
10: $\quad \wedge\, \hat{\delta}(q', \alpha) = q$
11: $\quad \wedge\, x \in \mathrm{FIRST}_k^G(\beta' x')\ \}$
12: **until** $\mathcal{I}^{(i)} = \mathcal{I}^{(i-1)}$
13: $\mathcal{I} = \mathcal{I}^{(i)}$
enumerate all symbols
14: $\mathcal{L} = \emptyset$
15: **for** $\langle [A \to \alpha \bullet X_1 X_2 \ldots X_n, x], q \rangle \in \mathcal{I}^{(0)}$ **do**
16: $\quad \mathcal{L} \leftarrow \mathcal{L} \cup \{X_i, X_{i+1}, \ldots, X_n\}$
17: **end for**
18: **for** $\langle [A \to \alpha \bullet A' X_1 X_2 \ldots X_n, x], q \rangle \in \mathcal{I} \setminus \mathcal{I}^{(0)}$ **do**
19: $\quad \mathcal{L} \leftarrow \mathcal{L} \cup \{X_i, X_{i+1}, \ldots, X_n\}$
20: **end for**

describing the structure of additive expressions should not be changed as otherwise the programmer needs to modify the derivation tree manually.

2. *Nonessential left-recursive nonterminals* are those where the left recursion is used to make a grammar more suitable for a particular parsing algorithm.
   For instance, the Java 1.0 grammar includes a number of non-essential left recursion like

   $$\begin{aligned} &\text{ClassBodyDeclarations} \longrightarrow \text{ClassBodyDeclaration} \\ &\text{ClassBodyDeclarations} \longrightarrow \text{ClassBodyDeclarations ClassBodyDeclaration} \end{aligned}$$

   that can easily be rewritten to

   $$\begin{aligned} &\text{ClassBodyDeclarations} \longrightarrow \text{ClassBodyDeclaration ClassBodyDeclarations}_{\text{opt}} \\ &\text{ClassBodyDeclarations}_{\text{opt}} \longrightarrow \varepsilon \\ &\text{ClassBodyDeclarations}_{\text{opt}} \longrightarrow \text{ClassBodyDeclaration ClassBodyDeclarations}_{\text{opt}} \end{aligned}$$

   using the established method for immediate left recursion elimination.

Nonterminal symbol ClassBodyDeclarations describes the entire contents of a class – all declarations within a class together. If it is left-recursive (the original productions), virtually entire Java source is parsed by the embedded left LR(1) parser – everything except the package header, import declarations and the class header. If the non-left-recursive productions are used, then every declaration within a class can be parsed separately using

either LL or LR parsing, depending on (a) the structure of a particular declaration and (b) what other left-recursive productions are replaced by non-left-recursive ones.

By eliminating the left recursion (using the well-known recipe illustrated above) in productions for only

ImportDeclarations, TypeDeclarations,
InterfaceMemberDeclarations, ClassBodyDeclarations, and
BlockStatements

the percentage of the input that is parsed using the embedded LR(1) parsers drops significantly: (typically) from 98 % to 54 % (but one should be warned that by using the right set of Java sources, especially the second number can be tailored to whatever value one chooses). Instead of parsing the entire contents of an interface or a class using the embedded left LR(1) parser, individual interface members or statements within a method can be reached using the backbone LL(1) parser – and at the same time, expressions can be parsed using the left-recursive productions producing the most appropriate derivation trees. Furthermore, approx. 54 % of the input is not parsed using one embedded LR(1) parser as a single substring but rather as a number of short substrings parsed by several instances of embedded LR(1) parsers.

Reducing the length of the substrings that need to be parsed using the individual embedded LR(k) parser is crucial for efficient LLLR parsing. In case of Java 1.0, these substrings encompass mostly expressions (which are seldom longer than a few dozen symbols) and prefixes of the field or method declarations.

Finally, as the left parse of the Java program is produced during LLLR(1) parsing, semantic routines can be inserted at every position within a grammar, i.e., on both sides of every symbol found on the right side of some production. This is also true for several other parsing methods like SLL, LL(*) and ALL(*), but not for left-corner parsing and its derivatives like XLC(1) and LAXLC(1) parsing. For instance, methods like the one described in [10] can handle semantic routines in only 41.95 % of all positions within a grammar.

7 Conclusions

The main advantage of LLLR parsing (which can be used to parse any LR(k) language) over left-corner parsing is that it produces the left parse of the input string while left-corner parsing produces the mixed-strategy parse. If compared with LL(*) and ALL(*) parsing, LLLR parser runs in linear time and does not involve backtracking.

However, it must be admitted once again that there is no silver bullet in parsing. The LLLR(k) parsing works well for certain LR(k) grammars, i.e., typically for grammars where LL(k) conflicting nonterminals appear close to the bottom of the derivation trees. Otherwise, if LL(k) conflicting nonterminals are found near the top of large subtrees of the derivation tree, LLLR(k) parsing is not to be advised – one should either modify the grammar or use some other parsing method (the situation is similar to LR(k) parsing where, if an LR(k) grammar is right-recursive, the parser works but consumes a lot of stack unnecessarily).

There are basically two issues left undone in regard to LLLR parsing. First, a combination of LL and LR parsing using lookahead buffers of different sizes is worth investigating. Second, a clear theoretical formulation of LLLR parsing using the combination of the canonical LL(k_1) and the canonical LR(k_2) machines would be a challenging task.

References

1. Alfred V. Aho and Jeffrey D. Ullman. *The Theory of Parsing, Translation and Compiling*, volume Volume I: Parsing. Prentice-Hall, Englewood Cliffs, N.J., USA, 1972.
2. Alan J. Demers. Generalized left corner parsing. In *Proceedings of the 4th ACM SIGACT-SIGPLAN symposium on Principles of programming languages POPL'77*, pages 170–182, Los Angeles, CA, USA, 1977. ACM, ACM.
3. Bryan Ford. Parsing expression grammars: a recognition-based syntactic foundation. In *Proceedings of the 31st ACM SIGACT-SIGPLAN symposium on Principles of programming languages POPL'04*, pages 111–122, Venice, Italy, 2004. ACM, ACM.
4. R. Nigel Horspool. Recursive ascent-descent parsers. In Dieter Hammer, editor, *Compiler Compilers, Third International Workshop CC '90, Schwerin, FRG*, volume 477 of *Lecture Notes in Computer Science*, pages 1–10. Springer-Verlag, 1990.
5. R. Nigel Horspool. Recursive ascent-descent parsing. *Journal of Computer Languages*, 18(1):1–16, 1993.
6. Matthew Might and David Darais. Yacc is dead. Available online at Cornell University Library (arXiv.org:1010.5023), 2010.
7. Mark-Jan Nederhof. Generalized left corner parsing. In *Proceedings of the sixth conference on European chapter of the Association for Computational Linguistics EACL'93*, pages 305–314, Stroudsburg, PA, USA, 1993. Association for Computational Linguistics.
8. Terence Parr and Kathleen Fischer. LL(*): The foundation of the ANTLR parser generator. *ACM SIGPLAN Notices - PLDI'10*, 46(6):425–436, 2011.
9. Terence Parr, Sam Harwell, and Kathleen Fischer. Adaptive LL(*) parsing: The power of dynamic analysis. In *Proceedings of the 2014 ACM SIGPLAN International Conference on Object-Oriented Programming Systems Languages and Applications (OOPSLA'14)*, volume 579–598, Portland, OR, USA, 2014. ACM, ACM.
10. Paul Purdom and Cynthia A. Brown. Semantic routines and LR(k) parsers. *Acta Informatica*, 14(4):299–315, 1980.
11. James P. Schmeiser and David T. Barnard. Producing a top-down parse order with bottom-up parsing. *Information Processing Letters*, 54(6):323–326, 1995.
12. Elizabeth Scott and Adrian Johnstone. GLL parsing. *Electronic Notes in Theoretical Computer Science*, 253(7):177–189, 2010.
13. Elizabeth Scott, Adrian Johnstone, and Rob Economopoulos. BRNGLR: a cubic Tomita-style GLR parsing algorithm. *Acta Informatica*, 44(6):427–461, 2007.
14. Seppo Sippu and Eljas Soisalon-Soininen. *Parsing Theory, Volume I: Languages and Parsing*, volume 15 of *EATCS Monographs on Theoretical Computer Science*. Springer-Verlag, Berlin, Germany, 1988.
15. Seppo Sippu and Eljas Soisalon-Soininen. *Parsing Theory, Volume II: LR(k) and LL(k) Parsing*, volume 20 of *EATCS Monographs on Theoretical Computer Science*. Springer-Verlag, Berlin, Germany, 1990.
16. Boštjan Slivnik. The embedded left LR parser. In *Proceedings of the Federated Conference on Computer Science and Information Systems*, pages 871–878, Szczecin, Poland, 2011. IEEE Computer Society Press.
17. Boštjan Slivnik. LL conflict resolution using the embedded left LR parser. *Computer Science and Information Systems*, 9(3):1105–1124, 2012.
18. Boštjan Slivnik and Boštjan Vilfan. Producing the left parse during bottom-up parsing. *Information Processing Letters*, 96(6):220–224, 2005.
19. Boštjan Slivnik. LLLR parsing. In *Proceedings of the 28th Annual ACM Symposium on Applied Computing SAC'13*, pages 1698–1699, Coimbra, Portugal, 2013. ACM.

Locating User Interface Concepts in Source Code*

Matúš Sulír[1] and Jaroslav Porubän[2]

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

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

Abstract

Developers often start their work by exploring a graphical user interface (GUI) of a program. They spot a textual label of interest in the GUI and try to find it in the source code, as a straightforward way of feature location. We performed a study on four Java applications, asking a simple question: Are strings displayed in the GUI of a running program present in its source code? We came to a conclusion that the majority of strings are present there; they occur mainly in Java and "properties" files.

1998 ACM Subject Classification D.2.3 Coding Tools and Techniques, H.5.2 User Interfaces: Graphical User Interfaces

Keywords and phrases Source code, graphical user interfaces, feature location

Digital Object Identifier 10.4230/OASIcs.SLATE.2016.6

1 Introduction

Developers understand a program only when they are able to mentally connect structures in the program with real-world concepts [2]. Naturally, this connection can be established much more easily if the vocabulary used in the source code resembles the domain terms displayed in the GUI of a program.

One of the most frequent activity performed by a programmer is feature location – finding an initial source code location implementing a given functionality [5]. To perform it, developers rarely use complicated feature location tools and plugins [8], and rely on simple textual search instead [4].

Consider a developer trying to fix a bug in a program he does not know. He will probably start with an exploration of a running UI (user interface) relevant to the bug. He will start to concentrate on particular GUI items, like buttons and menu items causing the bug to manifest. Then, he will try to search for the labels of these GUI items (button captions, menu names) in the source code of the program, using standard search functionality of an IDE (integrated development environment).

The GUI of a program is displayed to an end user – often a paying customer. For this reason, it must contain terms from the problem domain. On the other hand, the source code is rarely shown to a customer. The use of correct domain concepts in the source code is only a recommended practice, often not enforced.

* This work was supported by the FEI TUKE Grant no. FEI-2015-16 "Evaluation and metrics of domain usability".

© Matúš Sulír and Jaroslav Porubän;
licensed under Creative Commons License CC-BY

Table 1 The applications used in the study.

Application	*Java LOC*	*GUI strings*	*GUI words*
ArgoUML 0.34	195,363	307	2,391
FreeMind 1.0.1	67,357	353	1,050
PDFsam 2.0.0	23,774	65	168
Weka 3.6.13	275,036	592	904

We formulate our main hypothesis and two smaller research question for this paper as follows:

- *Hypothesis*: Strings and concepts displayed in the GUI of a running program are located in its static source code, too.
- *RQ1*: When yes, mainly in what types of files are these strings located?
- *RQ2*: If no, what are the most common reasons?

2 Method

To prove our hypothesis, we will automatically extract strings from a running GUI of a few applications and try to search for these strings (and their parts) in the source code of the corresponding program.

In Table 1, there is a summary of the studied objects. We selected three desktop Java applications from the SF110 [6] corpus of open source projects: FreeMind[1] is mind-mapping software, PDFsam[2] splits and merges PDF files, and Weka[3] is machine learning software. Additionally, ArgoUML[4] – a UML modeling tool – was selected as a popular, medium-sized project. The "Java LOC" column in Table 1 denotes the number of source code lines in Java files, measured by the the CLOC[5] program.

2.1 GUI Scraping

Before running the experiment, we ensured English localization was set in all applications, since the source code is written in English and a mismatch between the code and GUI language would produce skewed results. In the case of the FreeMind application, language adjustment in the settings was necessary, all other programs had the language already set correctly.

Every application was fed to a GUI ripper [10] which is a part of the project GUITAR [12]. The GUI ripper fully automatically opens all available windows in the program, checks all check-boxes, clicks the menus, etc., in a systematic way. The properties of all widgets are written in a form of an XML file.

From the XML file, a text and title was extracted for all recorded widgets. The number of unique strings for each application is in Table 1, column "GUI strings". Examples of these strings include button labels and tooltips, text-area contents, items in combo-boxes and many more. We excluded strings shorter than two characters, as they do not represent realistic search queries for further analysis.

[1] http://sourceforge.net/projects/freemind/
[2] http://sourceforge.net/projects/pdfsam/
[3] http://sourceforge.net/projects/weka/
[4] http://argouml.tigris.org/
[5] http://github.com/AlDanial/cloc

Table 2 The occurrence counts of whole strings from GUIs in the source code.

	Occurrences of GUI strings in code				
Application	0	1	$[2, 10)$	$[10, 100)$	$[100, \infty)$
ArgoUML	20.5%	10.4%	42.3%	12.1%	14.7%
FreeMind	7.9%	2.8%	60.6%	13.0%	15.6%
PDFsam	13.8%	13.8%	50.8%	4.6%	16.9%
Weka	8.1%	12.0%	14.0%	30.2%	35.6%

Table 3 The occurrence counts of individual words from GUIs in the source code.

	Occurrences of GUI words in code				
Application	0	1	$[2, 10)$	$[10, 100)$	$[100, \infty)$
ArgoUML	4.8%	3.1%	13.6%	29.4%	49.2%
FreeMind	0.7%	0.1%	26.0%	32.9%	40.4%
PDFsam	4.8%	7.7%	4.2%	18.5%	64.9%
Weka	6.6%	0.7%	5.9%	32.5%	54.3%

2.2 Analysis

Some of the strings contain multiple words, or even lines of text. For this reason, we broke them into individual words. We define a word as a sequence of three or more letters. The number of unique words for each application is in the column "GUI words" in Table 1.

For each string contained in the GUI, we searched it in the source code files of the corresponding project. The same process was repeated for individual words.

Regarding the source code, we used tarballs of the same versions as the binaries. The PDFsam tarball contained also automatically generated JavaDoc API documentation, which we removed, since such files should not be included in source archives.

The searching was performed fully automatically, via a script. We decided to perform a case-sensitive search, which should be more precise, especially to locate whole GUI strings. On the other hand, in practice, case-insensitive search is probably the preferred way, as it is often default in IDEs.

3 Quantitative Results

3.1 Occurrence Counts

First, we would like to simulate a situation when a programmer tries to find a whole GUI string in the source code. For each string, we determine a number of occurrences in the project – essentially the number of search results he would get in an IDE. For example, the string "Generate Data" (a button label in the Weka application) has 2 occurrences in the source code of the Weka project. Only text files were searched – this behavior is consistent with the majority of common IDEs which ignore binary files when searching.

In Table 2, we can see what portion of all GUI strings has no occurrence in the source code, exactly one occurrence, from 2 to 10 occurrences, etc.

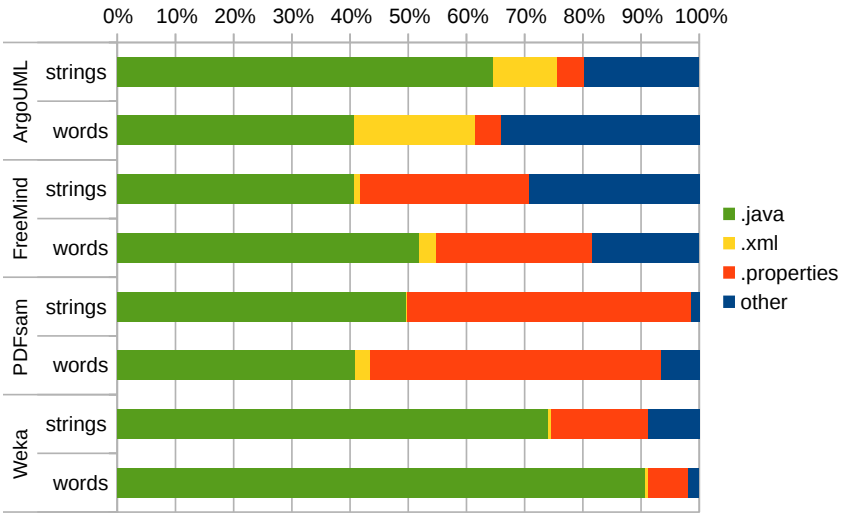


Figure 1 Occurrence counts of strings/words divided by file types.

Similar statistics, but for individual words, are located in Table 3. This represents the situation when the programmer is unhappy about the results and starts searching for smaller parts of the given string – usually words.

An ideal situation arises when a search gives exactly one result. It can (in theory) mean that the programmer found the sole piece of code relevant to the GUI widget. The higher is the number of occurrences, the longer he must sift through search results to find the relevant code. However, a situation when a searched string is not found in the source code is unfavorable, as the developer would have no idea where to start searching for an implementation of the given feature.

3.2 File Types

Ideally, the search results point to Java source files (`*.java`). This way, it is possible to directly find the code of interest. However, the source code of a project usually contains many kinds of files – not just Java source files.

For each project, we took all GUI string (and word) occurrences in all files and divided them by an extension of a file they are located in. In Figure 1, there is a graphical representation of the results.

In ArgoUML, 65% of all occurrences of GUI strings are contained directly in Java source code files. Although the project uses `*.properties` files for internationalization, GUI concepts are often used also as identifiers in Java code. For example, a GUI label "Notation" is present many times in the source code in the form of the class name `Notation`.

The FreeMind project uses "properties" localization files, too.

PDFsam uses a system where each key in a `*.properties` file is the original English string, and the value is the translated one. Therefore, the GUI strings are located both in `*.java` and `*.properties` files.

While Weka also uses `*.properties` files for localization, many GUI strings are also contained in special DSL [7] (domain specific language) files with an extension `.ref`.

4 Qualitative Results

While the numbers gave us some insight about the presence of strings from GUIs in the source code, it is necessary to know exact reasons why some strings were not found at all. Furthermore, we will show on a few samples that the strategy of searching GUI strings in the source code can sometimes be an effective way of finding the relevant code.

4.1 Strings Not Present in Code

Many GUI strings were not found in the source code of the application because they were a part of a universal dialog supplied by the GUI toolkit. For example, color-related terms like "Saturation" were not found in the FreeMind source tree because they are a component of the standard Java Swing color chooser dialog. Examples of such strings in ArgoUML and Weka are "File Name:" – a part of a file selection dialog and "One Side" – a term from the printing dialog.

The string "http://simplyhtml.sf.net/" displayed in FreeMind was located only in a class file inside a third-party JAR archive. Therefore, it is invisible for a standard textual search.

The string "Mode changed to MindMap Mode" was not found in the FreeMind source code as a whole because it was instantiated at runtime from the template "Mode changed to {0}". This forces the developer to find smaller portions of a string until a match appears, as we mentioned earlier.

The label "Show Icons Hierarchically" was not found in the source code of FreeMind because it was written as "Show Icons &Hierarchically" to specify the keyboard shortcut Alt+H.

Examples of strings which are not present in the code as a whole, but their parts can be found there, are help texts, logs and exception stack traces.

Regarding PDFsam, the label "Thumbnail creator:" was not found in the code because the colon was programatically concatenated.

4.2 Strings Present in Code

First, we tried to search for a string which is present exactly once in the FreeMind code: "Change Root Node". It was located in a localization file, as a value of a key named "accessories/plugins/ChangeRootNode.properties_name". Opening the file "accessories/plugins/ChangeRootNode.java" revealed that this Java class is really relevant to the feature.

Next, we searched for a string present 35 times in the code – "Bubble". It represents a node format in the mind-mapping software. This time, the exploration of the results took more time and we required multiple iterations using different search terms, even with some dead ends, until we finally found the relevant source code.

Finally, we searched the string "Export" (a menu item), present 1,988 times in the source code. Just skimming through such a long list is a lengthy activity. Therefore, other strategies are necessary to efficiently find the feature of interest in the code. For example, the programmers can reformulate their search queries, use structured navigation (tools like Call Hierarchy) or debugging techniques [4].

We conclude that in some cases, simple textual searching is a feasible way to find code relevant to a GUI element. Ideally, a GUI string should be located exactly once (or just a few times) in the source code, to allow easy finding of source code relevant to a GUI feature. Furthermore, finding an occurrence in a non-Java file makes it more difficult to find relevant source code than finding it directly in a Java file.

5 Threats to Validity

We will now look at the threats to validity of our study. Construct validity is concerned with the correctness of the measures. External validity discusses whether the findings can be generalized. Reliability denotes whether similar results would be obtained by another researcher replicating the study [13].

5.1 Construct Validity

While the GUI ripper in the GUITAR suite gives good results when scraping the GUI, it is not perfect and it could miss some of the strings visible in the user interface.

During an automated search for whole GUI strings in the code, also long texts like exception stack traces were included. It is not probable that a programmer will actually try to search for a whole stack trace in the code textually, as-is. Instead, he will directly look at some of the methods mentioned in the trace.

As was already mentioned, we performed a case-sensitive search, which has both advantages and disadvantages. In the future, a case-insensitive search should be also performed to better reflect the manual searching behavior of programmers.

5.2 External Validity

All four applications in our study were desktop Java programs, using the Swing GUI widget toolkit. However, common contemporary applications have Web and mobile front-ends. Scraping Web applications could have produced much different results. For example, they often display texts downloaded from external databases. This could be one of the reasons for non-presence of GUI strings in the code of these applications.

Even in the world of Java Swing applications, the selected ones represent just a small sample. However, they are representative of common Java projects, as three of them were included in the standard SF110 [6] benchmark.

5.3 Reliability

The quantitative results were produced chiefly by an automated script. Therefore, the subjectivity of a researcher is eliminated. The strings presented in the qualitative part were selected manually, but we tried to select representative samples.

6 Related Work

6.1 GUI Ripping

To rip GUIs, we used the tool GUI ripper [10] which is a part of the GUITAR [12] suite. Swing UIs are one of the best supported technologies, however, there is a partial support for SWT, Web, and Android. To crawl highly dynamic Web applications, Crawljax [11] could be used.

The DEAL method [1] creates a DSL from a GUI. However, the process is not automated and a user must manually traverse the user interface.

6.2 Feature Location Using GUIs

Of course, finding a string from a UI using IDE's textual search is not a sole option to perform feature location. GUITA [14] allows to take a snapshot of a running GUI widget. The snapshot is associated with a method which was last called on the given widget.

Another approach, UI traces [15], splits a long method trace into smaller ones, each associated with a graphical snapshot of the GUI in the given state.

6.3 Feature Location in General

There exists a large number of feature location techniques – see [5] for a survey. An example of a method utilizing source code comments and identifiers is presented by Marcus et al. [9]. Carvalho et al. [3] use a combination of static and dynamic analysis, specifications and ontologies to map problem domain concepts to source code elements. However, none of these approaches use labels directly from GUIs of running programs.

6.4 Other Studies

Václavík et al. [16] analyzed words used in names of identifiers in the source code of Java EE application servers and web frameworks. They tried to determine what portion of these words are meaningful according to the WordNet database. The more words from the source code of a project are meaningful, the more understandable it should be. We could perform a similar experiment, but use a dictionary built from the GUI of an application instead of WordNet.

7 Conclusion

In this article, a simple study was performed: We scraped all strings contained in a GUI of four open-source Java desktop applications and tried to automatically find them (as a whole and words contained in them) in the static source code.

Regarding the main *hypothesis*: The vast majority of GUI strings were found in the code. However, 11.2% of them were not found at all.

Answering *RQ1*, the GUI strings are often located in Java source code files, `*.properties` localization files, XML and custom DSL files.

To answer *RQ2*, the main reasons of a non-presence of a GUI string in the source code were: a string was a part of a standard dialog, a third-party library, or the string was dynamically generated at runtime.

If we consider a set of GUI words the application's problem domain dictionary, the percentage of GUI words present in the source code can be regarded as a measure of code understandability. For example, if the code contains too few concepts from the GUI, it can be considered obfuscated.

We found out that the approach of simple textual code search of strings displayed in the GUI is useable and it can actually find code relevant to a feature, unless there are too many results, or none at all.

As a future work, there is a potential in creation of a tool which would automatically assign GUI strings to corresponding source code fragments, e.g., using annotations. This way, a simple textual search would be sufficient to quickly find any code related to a given GUI element. Also, we can replicate this study on Web applications, or study logs in addition to GUIs.

References

1 Michaela Bačíková, Jaroslav Porubän, and Dominik Lakatoš. Defining domain language of graphical user interfaces. In José Paulo Leal, Ricardo Rocha, and Alberto Simões, editors, *2nd Symposium on Languages, Applications and Technologies*, volume 29 of *OpenAccess Series in Informatics (OASIcs)*, pages 187–202, Dagstuhl, Germany, 2013. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/OASIcs.SLATE.2013.187.

2 Ted J. Biggerstaff, Bharat G. Mitbander, and Dallas Webster. The concept assignment problem in program understanding. In *Proceedings of the 15th International Conference on Software Engineering*, ICSE'93, pages 482–498, Los Alamitos, CA, USA, 1993. IEEE Computer Society Press. URL: http://dl.acm.org/citation.cfm?id=257572.257679.

3 Nuno Ramos Carvalho, José João Almeida, Pedro Rangel Henriques, and Maria João Varanda Pereira. Conclave: Ontology-driven measurement of semantic relatedness between source code elements and problem domain concepts. In *Computational Science and Its Applications – ICCSA 2014*, pages 116–131. Springer International Publishing, 2014. doi:10.1007/978-3-319-09153-2_9.

4 Kostadin Damevski, David Shepherd, and Lori Pollock. A field study of how developers locate features in source code. *Empirical Software Engineering*, 21(2):724–747, 2016. doi:10.1007/s10664-015-9373-9.

5 Bogdan Dit, Meghan Revelle, Malcom Gethers, and Denys Poshyvanyk. Feature location in source code: a taxonomy and survey. *Journal of Software: Evolution and Process*, 25(1):53–95, 2013. doi:10.1002/smr.567.

6 Gordon Fraser and Andrea Arcuri. A large-scale evaluation of automated unit test generation using EvoSuite. *ACM Trans. Softw. Eng. Methodol.*, 24(2):8:1–8:42, December 2014. doi:10.1145/2685612.

7 Tomaž Kosar, Nuno Oliveira, Marjan Mernik, Maria João Varanda Pereira, Matej Črepinšek, Daniela da Cruz, and Pedro Rangel Henriques. Comparing general-purpose and domain-specific languages: An empirical study. *Computer Science and Information Systems*, 7(2):247–264, April 2010. doi:10.2298/CSIS1002247K.

8 Walid Maalej, Rebecca Tiarks, Tobias Roehm, and Rainer Koschke. On the comprehension of program comprehension. *ACM Trans. Softw. Eng. Methodol.*, 23(4):31:1–31:37, September 2014. doi:10.1145/2622669.

9 Andrian Marcus, Andrey Sergeyev, Václav Rajlich, and Jonathan I. Maletic. An information retrieval approach to concept location in source code. In *Reverse Engineering, 2004. Proceedings. 11th Working Conference on*, pages 214–223, Nov 2004. doi:10.1109/WCRE.2004.10.

10 Atif Memon, Ishan Banerjee, and Adithya Nagarajan. GUI ripping: reverse engineering of graphical user interfaces for testing. In *Reverse Engineering, 2003. WCRE 2003. Proceedings. 10th Working Conference on*, pages 260–269, Nov 2003. doi:10.1109/WCRE.2003.1287256.

11 Ali Mesbah, Arie van Deursen, and Stefan Lenselink. Crawling AJAX-based Web applications through dynamic analysis of user interface state changes. *ACM Trans. Web*, 6(1):3:1–3:30, March 2012. doi:10.1145/2109205.2109208.

12 Bao N. Nguyen, Bryan Robbins, Ishan Banerjee, and Atif Memon. GUITAR: an innovative tool for automated testing of GUI-driven software. *Automated Software Engineering*, 21(1):65–105, 2013. doi:10.1007/s10515-013-0128-9.

13 Per Runeson and Martin Höst. Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering*, 14(2):131–164, 2009. doi:10.1007/s10664-008-9102-8.

14 André L. Santos. GUI-driven code tracing. In *Visual Languages and Human-Centric Computing (VL/HCC), 2012 IEEE Symposium on*, pages 111–118, Sept 2012. doi:10.1109/VLHCC.2012.6344495.

15 Andrew Sutherland and Kevin Schneider. UI traces: Supporting the maintenance of interactive software. In *Software Maintenance, 2009. ICSM 2009. IEEE International Conference on*, pages 563–566, Sept 2009. doi:10.1109/ICSM.2009.5306389.

16 Peter Václavík, Jaroslav Porubän, and Marek Mezei. Automatic derivation of domain terms and concept location based on the analysis of the identifiers. *Acta Universitatis Sapientiae. Informatica*, 2(1):40–50, 2010. URL: http://www.acta.sapientia.ro/acta-info/C2-1/info21-4.pdf.

Declarative Rules for Annotated Expert Knowledge in Change Management

Dietmar Seipel[1], Rüdiger von der Weth[2], Salvador Abreu[3], Falco Nogatz[4], and Alexander Werner[5]

1 **Department of Computer Science, University of Würzburg, Würzburg, Germany**
dietmar.seipel@uni-wuerzburg.de
2 **Fac. of Business Admin., Dresden University of Applied Sciences, Dresden, Germany**
weth@htw-dresden.de
3 **LISP and Department of Computer Science, University of Évora, Évora, Portugal**
spa@di.uevora.pt
4 **Department of Computer Science, University of Würzburg, Würzburg, Germany**
falco.nogatz@uni-wuerzburg.de
5 **Fac. of Business Admin., Dresden University of Applied Sciences, Dresden, Germany**
alexander.werner@htw-dresden.de

Abstract

In this paper, we use declarative and domain–specific languages for representing expert knowledge in the field of change management in organisational psychology. Expert rules obtained in practical case studies are represented as declarative rules in a deductive database. The expert rules are annotated by information describing their provenance and confidence. Additional provenance information for the whole – or parts of the – rule base can be given by ontologies.

Deductive databases allow for declaratively defining the *semantics* of the expert knowledge with rules; the evaluation of the rules can be optimised and the inference mechanisms could be changed, since they are specified in an abstract way. As the logical syntax of rules had been a problem in previous applications of deductive databases, we use specially designed domain–specific languages to make the rule *syntax* easier for non–programmers.

The semantics of the whole knowledge base is declarative. The rules are written declaratively in an extension DATALOG* of the well–known deductive database language DATALOG on the data level, and additional DATALOG* rules can configure the processing of the annotated rules and the ontologies.

1998 ACM Subject Classification H.2 Database Management, I.2.1 Applications and Expert Systems, I.2.4 Knowledge Representation Formalisms and Methods

Keywords and phrases declarative, DATALOG, PROLOG, domain-specific, change management

Digital Object Identifier 10.4230/OASIcs.SLATE.2016.7

1 Introduction

There have been many rule–based approaches for knowledge representation, but the declarative approach of *deductive databases* appears very promising. Especially when armed with


© Dietmar Seipel, Rüdiger von der Weth, Salvador Abreu, Falco Nogatz, and Alexander Werner;
licensed under Creative Commons License CC-BY

domain–specific languages, it becomes much easier to incorporate domain expertise into the development process of an information system.

The knowledge in organisations plays an important role in day–to–day business, even more if routines and organisational structures have to be changed. Knowledge in organisations is represented in many ways. Some rules are documented explicitly in rules but there are also informal procedural rules, which are mostly undocumented. A reason why it is so difficult to compare these implicit and explicit rules is because they exist fragmented in different sources (e.g. individuals, groups) and can not be collected in a standardised manner. An approach to solve this problem is to transfer differently collected information about procedures in a standard format using content analysis, which allows to manage the rules in a deductive database system. The system DDBASE [17] allows to analyse rules during input and to link them in the evaluation process. Both a graphical visualiser interface and automated reasoning facilitate the linking of conclusions and help to detect contradictions. An expandable rule base enables the extension of an overall model and its validation across methodologically different studies. Having a semantically sound and functionally rich declarative rule language is an enabling factor when presented as a domain–specific language (DSL). These results pave the way for the comparison of informal knowledge and official rules as well as documenting and modelling their history accurately.

We have described a rule concept for knowledge in change management and some methods for querying and visualizing the rules from the point of view of organizational psychology in [21]; the syntax of the rules has been modelled using operator precedences yielding an internal DSL in PROLOG. In the present paper, we give a context–free grammar for the rule language, which might be textended later, and we provide a theory about the semantics and evaluation of declarative rule bases following concepts from DATALOG and logic programming in general. Moreover, it is necessary to include provenance information, that can be given by ontologies, and confidence annotations of the rules.

Organization of the Paper

The rest of this paper is organised as follows: Section 2 recalls some basic ideas from declarative programming, domain–specific languages, and deductive databases. Section 3 is about declarative rule bases; it introduces rules for change management and defines syntax, semantics, and evaluation using deductive databases and logic programming. Section 4 shows how the rule base can be represented using a suitable DSL. The analysis of rule bases is investigated in Section 5; ideas for queries and the visual analysis in interactive rule editors are given. Section 6 shows how the knowledge base can be augmented by contextual information given in ontologies and how rules can be annotated with confidence information. The paper is concluded with some final remarks.

2 Background Concepts

In this section, we recall some concepts useful for reading the rest of the article, namely declarative programming, domain–specific languages, deductive databases and logic programming. We will mainly summarize and highlight some of the general statements found in literature.

2.1 Declarative Programming

Declarative programming is a programming paradigm that expresses the logic of a computation in an abstract way, without having to describe its control flow. Thus, the *semantics* of a

declarative language becomes easier to grasp for domain experts. Declarative programming offers, e.g., the following advantages for data and knowledge engineering: security, safety, and shorter development times, as known from information systems with relational databases. There exists a plethora of results about query optimisation in relational and deductive databases, e.g., [7, 4, 19, 20, 14]. For instance, Minker and his students have interesting results in the field of semantic query optimisation [5]: evaluation plans can be derived and cached results can be included.

Languages which claim to be declarative usually attempt to minimise or eliminate side effects by describing *what* the program must accomplish in terms of the problem domain rather than describing *how* to accomplish it as a sequence of the programming language instructions – the how is implicitly left to be decided by the implementation of the language. In contrast, imperative (or procedural) languages require algorithms to be implemented in explicit steps. Declarative programming often considers programs as theories of a formal logic, and computations as deductions, or proofs. Declarative programming may greatly simplify writing parallel programs, as it does away with explicit control. Examples of declarative languages include database query languages (e.g., SQL, DATALOG, XQuery), regular expressions, logic and constraint programming (e.g. PROLOG), and functional programming.

2.2 Domain Specific Languages

A domain–specific language (DSL) is a computer language specifically tailored to a particular application domain, in contrast to a general–purpose language (GPL), which aims for broad applicablility across domains. The *syntax* of a DSL is meant to be intelligible for domain experts. According to Martin Fowler [9], *DSLs are small languages, focused on a particular aspect of a software system*, i.e. they cannot be used to write a whole program, although it is frequent to resort to multiple DSLs in a single system, which is basically written in a GPL.

DSLs can take on two forms: *external* or *internal*. The first form is parsed independently of the host GPL, for instance CSS or regular expressions. Internal DSLs, also known as *embedded* DSLs, are a dialect of a host programming language, and are intrinsically part of the host syntax; they amount to an API in a general–purpose language.

DSLs are appreciated because, for its target domain, a DSL is much easier to wield than either a GPL or a traditional library. The outcome is increased programmer productivity, which is always welcome. Having a DSL also improves communication with the domain experts. In short, a DSL raises the level of abstraction required to program an application, allowing non–computer savvy experts to work more productively.

There are DSLs for numerous areas of application, such as, e.g., expert rules, business rules, configuration rules/constraints, and queries to databases. A systematic mapping study has been given in [12].

2.3 Deductive Databases and Logic Programming

A deductive database (DDB) is a database which may carry out *deductions* based not only on facts but also on rules which are also stored in the database itself [4]. DDBs combine logic programming languages and relational databases, as they share the querying flexibility of the former while retaining the performance and scalability of the latter.

DDBs commonly use variants of the logic programming language DATALOG, whose syntax restricts the standard logic programming language PROLOG [3, 22], and whose declarative bottom–up semantics, given in Section 3, is closer to relational databases.

Typically DDBs will operate on data which are more restrictive than that of Prolog, yet more general than that which may structurally be accessed with Sql. Deductive database languages have been used in many applications, such as data integration, computer networking, program analysis or security [1, 14].

3 Declarative Rule Bases

Besides relational databases, ontologies have played an important role for building intelligent information systems. Currently, ontology languages like Owl are extended by rule–based elements and links. We have built tools for managing and analysing relations, ontologies, and rules. Techniques from deductive databases and logic programming can integrate hybrid knowledge bases with structured knowledge. We will show how domain–specific languages and declarative languages can offer a clear syntax and semantics to modern information systems. Nowadays, semantic web technology including linked data (JSON–LD) is also very important. Data and knowledge engineering can clearly benefit from the declarative approach provided by logic programming.

3.1 Declarative Rules in Change Management

Currently, the results obtained in psychological studies of organisations are not collected in a uniform data format. The data are kept in proprietary systems, which so far only serve for persistent storage without trying to obtain new insights. Some databases are used, but joining the data records and the underlying research results remained almost impossible for lack of integration. The field of *change management* offered a perfect case study for an integrated rule management [21]. The following types of rules have been considered in organisations: explicit, official business processes, and informal rules. Often, the sources of the rules are fragmented, distributed, and hybrid. To collect and manage the rules systematically, we have used the Prolog–based deductive database system DDbase [17]. We have investigated the analysis, evaluation, and visualisation of the rule base as well as reasoning techniques. Since we use a deductive database system, we are able to do a *continuous integration* of further rules. Based on the facilities of DDbase, we could perform a comparison of informal and official rules.

We are developing a textual, logic–based rule format, which tries to represent the rules – as far as possible – in a natural language syntax. We have modelled the emotional processes in connection with projects for introducing new software. Currently, we have about 50 rules including the rules shown below. The rules are relevant for companies that are considering to introduce an ERP system. E.g., the second rule states that the acceptance of an ERP system is decreasing if other software exists and the functionality and the acceptance of other Software is increasing.

```
if 'Processes in ERP System' = partly
then 'Processes in other Software' = partly .

if 'Existence of other Software' = yes
and 'Functionality of other Software' = increasing
and 'Acceptance of other Software' = increasing
then 'Acceptance of ERP System' = decreasing .

if 'Use of other Software' = increasing/constant
```

```
then 'Acceptance of ERP System by Users' = decreasing .

if 'Test of ERP System by Users' = yes
then 'Discovery of ERP Function by Users' = yes .
```

We have developed a DSL for intuitively representing the business rules, which maps to PROLOG in simple manner. We have also implemented mechanisms for analysing and visualising the rule base. We use the deductive database system DDBASE, which works with an extension of DATALOG, a logic programming language extending the well–known relational query language SQL.

3.2 Deductive Databases and Logic Programming

A comprehensive description of the syntax and semantics of deductive databases and logic programming is given, e.g., in [14]. A logic program $\mathcal{P}$ is a set of rules, which are range–restricted implications $A \leftarrow \beta$, where A is an atom and β can be any formula over atoms built with the junctors $\vee$, $\wedge$, and not (default negation). Some extensions can also handle literals with classical negation ($\neg$), rather than just atoms in the rules. We allow for function symbols and an arbitray use of the junctors in the rules, whereas frequently in deductive databases $\beta = B_1 \wedge \ldots \wedge B_m \wedge \mathsf{not}\ C_1 \wedge \ldots \wedge \mathsf{not}\ C_n$ is just taken as a conjunction of atoms B_i or default negated atoms not C_i without function symbols. A is called the head, and β is called the body of the rule. The property *range–restricted* means that every variable symbol in the head must also occur in the body, where variable symbols within default negated formulas not ϕ are not counted. Facts A are rules with an empty body and thus correspond to tuples in a relational database; rules $A \leftarrow \beta$ are implications. For instance, the well–known transitive closure rules can be expressed as:

$$tc(X,Y) \leftarrow arc(X,Y),$$
$$tc(X,Y) \leftarrow arc(X,Z) \wedge tc(Z,Y).$$

Semantics and Evaluation

In logic programming, terms are defined inductively: terms can be variable symbols or constant symbols or of the form $f(t_1,...,t_n)$, where f is a function symbol and $t_1,...,t_n$ are terms themselves. An atom is of the form $p(t_1,...,t_n)$, where p is a predicate symbol and $t_1,...,t_n$ are terms. A ground atom is an atom without variable symbols. E.g., $arc(a,b)$ is a ground atom with the predicate symbol arc, where the ground terms $t_1 = a$ and $t_2 = b$ are constants (strings starting with a lower case character), and the atom $arc(X,Y)$ contains the variable symbols X and Y (strings starting with an upper case character). The Herbrand base $HB_{\mathcal{P}}$ is the set of all ground atoms over the logic program $\mathcal{P}$, i.e. their predicate, function, constant and variable symbols must occur in P. An Herbrand interpretation I is a subset of $HB_{\mathcal{P}}$.

Consequences and Evaluation

Assuming the standard definition, we write $I \models \beta$, if I models β. Here, $I(\mathsf{not}\ \phi) = \neg I(\phi)$ and $I(\phi_1 \odot \phi_2) = I(\phi_1) \odot I(\phi_2)$, for formulas ϕ, ϕ_1, ϕ_2, and junctors $\odot = \vee, \wedge$. A ground rule $A \leftarrow \beta \in gnd\,(\mathcal{P})$ is obtained by substituting all variable symbols of a rule by ground terms. The immediate consequence operator $\mathcal{T}_{\mathcal{P}}$ derives all ground atoms A, such that there exists a ground rule in $gnd\,(\mathcal{P})$, where I models its body:

$$\mathcal{T}_{\mathcal{P}}(I) = \{\ A \in HB_{\mathcal{P}} \mid A \leftarrow \beta \in gnd\,(\mathcal{P}),\ I \models \beta\ \}.$$

Since the rules are range–restricted, $\mathcal{T}_\mathcal{P}(I)$ will be finite, if I is finite. E.g., for the transitive closure rules together with the facts $arc(a,b), arc(b,c), arc(c,d)$, the *bottom–up evaluation* derives the following monotonically increasing sequence of interpretations by repeatedly applying the rules to the already derived facts:

$$\begin{array}{lcl} I_0 & = & \emptyset, \\ I_1 & = & \{\, arc(a,b),\, arc(b,c),\, arc(c,d) \,\}, \\ I_2 & = & I_1 \cup \{\, tc(a,b),\, tc(b,c),\, tc(c,d) \,\}, \\ I_3 & = & I_2 \cup \{\, tc(a,c),\, tc(b,d) \,\}, \\ I_n & = & I_3 \cup \{\, tc(a,d) \,\},\ \text{for all } n \geq 4. \end{array}$$

For $n \in I\!N_0$, the interpretation $I_n = \mathcal{T}_\mathcal{P}^n$ is obtained by the repeated application of $\mathcal{T}_\mathcal{P}$, starting with $I_0 = \mathcal{T}_\mathcal{P}^0 = \emptyset$, i.e. $\mathcal{T}_\mathcal{P}^{n+1} = \mathcal{T}_\mathcal{P}(\mathcal{T}_\mathcal{P}^n)$. The least fixpoint of the consequence operator – here I_4 – is also the unique minimal model of the logic program. Observe, that the least fixpoint is $I_\omega = \mathcal{T}_\mathcal{P}^\omega = \cup_{n=0}^\omega \mathcal{T}_\mathcal{P}^n$. In theory, it can be infinite, if the Herbrand base is infinte since $\mathcal{P}$ contains function symbols. In practice, the rules have to ensure that the iteration terminates after finitely many steps with a finite fixpoint. The consequence operator and its iteration provide one proof–theoretic (operational) semantics of a logic program without default negation, i.e., an evaluation method.

Semantics

In general, the *semantics* of a logic program with default negation is given by its answer sets, cf. [14]. For our purposes, however, it is sufficient to consider logic programs with a limited use of default negation, so–called stratified programs, where there is no recursion through default negation. The evaluation of stratified programs can be based on logic programs without default negation at all. These programs – the transitive closure program above is an example – can be evaluated bottom–up using hyperresolution in an efficient bottom–up style. Then, *declarativity* is given by the fact that without default negation, three semantics coincide: model, proof, and fixpoint theory. In general, the answer set semantics of logic programs with unlimited default negation is defined by a fixpoint theory. This can also be extended to handle literals with classical negation ($\neg$), rather than just atoms. In non–monotonic reasoning with answer sets, we distiguish between true literals in an answer set and literals derived by the inference process. Intuitively, a default negated literal $\mathsf{not}A$ is considered true in an answer set, if the atom A cannot be derived, whereas a classically negated literal $\neg A$ is considered true, if the negated atom $\neg A$ can be derived.

During the first phase of the case study in change management, we needed only basic semantic concepts, since the main focus had been on the language and the structural analysis and visualisation of the rule base. For the future, it will be important that the semantics of the target language is precisely defined, a characteristic we inherit from logic programming and DATALOG.

Renaissance of Datalog

For several years, we can observe what is sometimes called a *renaissance* of DATALOG [1]. Zaniolo started with LDL at Austin; subsequently, many DDBs have been developed. New DATALOG applications have been developed, e.g., at Berkeley, where Boom and Bloom handle distributed computing, parallelism, and concurrency. New DATALOG companies have been created and became successful: the company LogicBlox provides a unified database foundation for the next generation of smart analytical and transactional applications; the

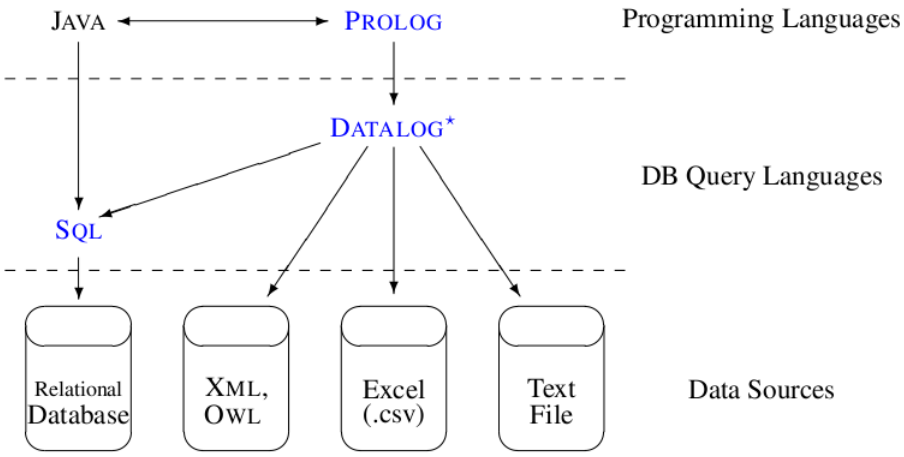


Figure 1 Architecture of DDBASE.

company Lixto on declarative web data extraction and annotation exists since the beginning of the millennium, and Gottlob's database group has recent publications with Oracle. SAP uses SWI–PROLOG [22] in its Cloud Platform HANA (configuration with source repository git/gerrit), just to name a few.

3.3 The Deductive Database System DDbase

The deductive database system DDBASE uses the extension DATALOG* with function symbols [16, 17]. Rule bodies can contain embedded PROLOG calls and default negation. In DDBASE, it is possible to have bottom–up and top–down evaluation in one system. DATALOG* can evaluate logic programs with PROLOG syntax (extended DATALOG programs) in a bottom–up style. Thus, the main evaluation method of DATALOG* is bottom–up; but DATALOG* is designed to evaluate embedded PROLOG calls in a top–down manner. In DDBASE, a logic program can be abstracted by a predicate dependency or a rule predicate graph, and a derivation can be visualised by its proof tree, cf. [4].

The architecture of DDBASE is given in Figure 1, which also shows that DDBASE can access hybrid data sources.

DDBASE is part of the DDK, the DISLOG developers' kit, a collection of PROLOG libraries written in SWI–PROLOG [22] including features from data and knowledge engineering, databases (relational, XML, and deductive), ontologies, and non–monotonic reasoning. It can be obtained from `http://www.ddbase.de`.

4 A Domain Specific Language for Rules

For expressing rules, we chose to follow the general form:

if *Condition* then *Consequence*.

Notwithstanding the previous schematic statement form, every rule is required to *end with a dot*. The dot allows the language to behave as an embedded DSL for PROLOG. With minor changes, it could be embedded into other host languages such as Python or Javascript.

The DSL has been conveniently defined in Prolog by a collection of suitable operator precedences. After the keywords `if` and `then`, a *Condition* and *Consequence*, respectively, is expected. Both are so–called junctions of findings. If *Condition* is empty, the rule is also called a *fact*. A finding always has the form: `Feature = Value`, where `Feature` and `Value` should generally be included in quotation marks. Only in strings that neither contain spaces nor start with a capital letter, the quotes can be omitted. The values are not limited to `yes` and `no`, for instance other literal descriptions as in the finding `'Acceptance' = increases` or numerical information, which can assume significant practical importance, are possible. Besides equality, additional comparators may be used.

4.1 Syntax of Formulas

Several findings in `Condition` and `Consequence` can be linked to formulas by connectives. For this, the keywords `and`, `or`, and `neg` are available. If F and G be formulas, then the following are also allowed formulas: `neg` F, F `and` G, F `or` G. Note that conjunction binds stronger than disjunction, and classical negation `neg` binds the most strongly. An extension would be to allow for default negation (`not`) to occur in `Condition`, but not in `Consequence`. For representing arbitrary formulas, subformulæ can be included in brackets.

Our domain–specific rule language could be described by a context–free grammar, which we could also implement using the following *definite clause grammar* (DCG) in Prolog.

```
rule --> "if ", formula, " then ", conjunction.
formula --> conjunction | disjunction | classical_negation.
conjunction --> literal | literal, " and ", formula.
disjunction --> literal | literal, " or ", formula.
classical_negation --> "-", formula.
literal --> finding | "not(", finding, ")".
finding --> feature, "=", value.
```

By further rules, we can define that features and values are certains strings without the character "=". This DCG can be used for verifying that a rule is in the language. The grammar formalism can help to clarify the syntax for people who are not experts in logic programming or Prolog. At the moment, however, we are not using the DCG. Instead, we have embedded the rule DSL internally into Prolog by providing suitable operator defintions for the junctors `if`, `then`, `and`, `or`, `neg`, etc. Thechnically, the rules `if Formula then Conjunction` can be parsed by Prolog into Prolog structures `then(if(Formula), Conjunction)`. From these structures, the rule base of our system can be derived easily and analyzed by our tool. In future work, we might need a more powerful rule language that cannot be an internal DSL in Prolog. In that case, will try to use refinements and extensions of the given DCG formalism to define a suitable external DSL.

W.r.t. the syntax given in Section 3.2, the findings `A=V` are the atoms in the rules; they have the binary predicate symbol "`=`". More general findings `A`$\odot$`V` can use other comparator predicate symbols "$\odot$", which could be, e.g., one of `=,<,>,=<,>=`. Moreover, also other atoms are possible in Datalog*, e.g. for embedded calls with built–in predicates. In Datalog*, the rules are evaluated bottom–up, and the embedded calls are evaluated in a top–down style, as it is common in logic programming approaches, cf. Section 3.3. Within a formula `not` F with default negation "`not`", no other default negation is allowed, but classical negation "$\neg$" is allowed; i.e., F can only be built using the classcical junctors $\wedge$, $\vee$, and $\neg$.

An Example from Change Management

As a more complex example, consider the following statement:

> *In small business, work processes are comprehensible without frequent team meetings, and no abundance of information arises.*

The same applies to large companies with frequent meetings. In natural language, this may be formulated as follows:

> *If either the size of the company is small or the meetings are frequent, then the transparency of the work processes increases and there is no information overload.*

The syntax of the rules must follow the form `if` *Condition* `then` *Consequence*. The statement above is only true in the case of exclusive ors: either the company is small – then no meetings are needed – or it is so large that team meetings are needed to track work processes without an excess of information. If both premises are met (i.e., there would be frequent business meetings in a small business), then the consequence `'information overload' = no` would not be true.

The given example illustrates that the formal recording of statements by predicate logic formulas can sharpen the uniqueness of the resulting statements. In the case of the more general F `or` G instead of `either` F `or` G, the application of the rule along with other statements might lead to inconsistencies. The detection of such situations is part of the functionality of our tool.

The syntax for the rule storage allows the basic conjunction `and` and disjunction `or`. Moreover, classical negation is supported, which is denoted by `neg` in rules. The exclusive or, $F \oplus G$ (either F or G), can be expressed as $(F \wedge \neg G) \vee (\neg F \wedge G)$ using elementary connectives. Thus, the statement sketched in the example above can be modeled using elementary connectives as follows:

```
if neg 'Company Size' = small and 'Meeting' = often
   or 'Company Size' = small and neg 'Meeting' = often
then 'Traceability of Work Processes' = rises
   and 'Information Overload' = no.
```

In this case, we do not need any brackets. The precedences ensure that `and` binds before `or` and `if` and `then`, that `or` binds before `if` and `then`, and finally that `if` binds before `then`. In Prolog, we can declare this more compactly by assigning increasing precedences to the operators in the sequence `=`, `and`, `or`, `if`, `then`. In general, using brackets we can express a formula where `or` should bind before `and`.

Variants of Rules and the DSL

The proposed notation for rules complies with the syntax of Prolog, which facilitates its usage as an embedded DSL. With the definition of `if`, `then`, `neg`, `and` and `or` as operators, the established rules become valid Prolog structures. Thus, it is possible to create an externally–backed rule base file including all known statements. As it conforms to user–readable syntax, the rule base may even be updated with a text editor. It may be gradually expanded by adding new rules. The end result is an incremental rule storage containing all statements found from research results to be analysed later.

We allow for formulas linking findings by the connectives `and` and `or`. If `Consequences` is a conjunction, then the rule can be normalized to several rules using macro expansion

techniques in PROLOG. More general rules over the junctors `and` and `or` can be transformed to several rules with disjunctive `Consequences`. So far, the domain experts have not used disjunctive `Consequences` in applications; at the moment, they are not accustomed to use disjunctions in rule heads. In the future, we will try to introduce that new feature into applications. Especially the handling of confidence values together with disjunctive `Consequences` will be an interesting research field.

4.2 The Integrated Development Environment (IDE)

The current tool will allow sloppy input that gets normalized. More powerful rule *editors* are planned for the future, so as to avoid having to directly alter the rule file. These advanced editors will facilitate the insertion of rules, to embody the intention that rules should look like natural language statements. Additionally, the editing tool will try to correct frequently occurring syntactic errors, for example, forgetting the dot at the end of the rule or the wrong notation and use of the junctors; these are potential sources of easy–to–correct errors.

Having a graphical integrated development environment will also facilitate the reuse of previously existing features and values. It is clear that conclusions from the given statements are only possible if the same names are consistently used for the same features.

At present, the tool implements a text–based approach for the handling and a declarative approach for the analysis and the visualisation of the rule base.

5 Design Analysis of the Rule Base

The individual records of the rule memory are usually stored in the memory of DDBASE and analysed with our tool. It is possible to read rules as PROLOG source code, to inspect the rules and even directly query them.

5.1 Declarative Queries

Using this deductive knowledge base, it is possible to answer the following exemplary questions with our tool:

- Which factors affect the acceptance of the new ERP system?
- Which constellation of findings is necessary to derive another finding?
  Are there findings, which are a particularly common cause of a change?
  Are there any *killer* findings, that block many developments?
- What are the necessary conditions for a finding? Which ones are optional?
- If a different value is assigned to a single feature, how does this affect the overall structure?
- Are there any redundant rules?
  Can some individual rules be expressed by more accurate rules?
- Where do some findings form opposite or even contradictory relationships?

The questions above underline the diversity of queries than can be asked. The current prototype is already supporting queries for conditions and consequences of individual findings. For example, by means of the predicate `depends_on`, the following query can be formulated in DDBASE (we do not show the encoding here); we can iterate through all answers. [1] The predicate `depends_on` is built to work also for pairs of features instead of just findings.

[1] This can be done by entering a semicolon "`;`" after each answer, standard procedure in a PROLOG top–level interpreter.

```
?- F1 = finding:Consequence, F2 = finding:Condition,
   depends_on(F1, F2).

Consequence = ('Emergence of ERP Knowledge in Employees' = yes),
Condition = ('Existence of ERP Knowledge in Employees' = yes) ;
Consequence = ('Emergence of ERP Knowledge in Employees' = yes),
Condition = ('Cooperation/Communication between Employees
   and Employees with ERP Knowledge' = yes) ;
...
```

Here, not only the contents of individual rules is returned, but also derived knowledge gets computed. If the consequence `'Emergence of Knowledge about ERP System in Employees' = yes` is a prerequisite for a further consequence, then the existence of an employee with knowledge about the ERP System is output as being a prerequisite. Since we use the system DDBASE, is it also possible to immediately determine all causes of an individual finding. For doing this, the consequence can be an argument in the following predicate, as this happens to determine the causes of a conflict:

```
?- F1 = finding:'Emergence of Conflicts' = yes,
   F2 = finding:Precondition,
   depends_on(F1, F2).

Precondition =
   ('Acceptance of ERP System at the Beginning' = partly) ;
Precondition =
   ('Feedback' = no).
```

For simple values, the tool can also handle classical negations, i.e., in the example above the two findings `neg 'Feedback' = no` and `'Feedback' = yes` are equivalent.

5.2 Visualisation of Dependency Graphs

We have already indicated the advantages of an interactive rule editor. Besides facilitating the entry of rules and the dynamic formulation of queries, the rule editor should be used for visualising the statements stored in the rule base.

In a similar form, this had been implemented with the tool VISUR, cf. [18]. The tool visualises a given rule base and thus allows for a graphical interpretation of the rule base. It had been developed for and used by AI people for the analysis and visualisation of rules in medical diagnosis. Thus findings, which are a prerequisite for a variety of consequences, can also been rendered visually. VISUR has, among other applications, been used for the visualisation of medical diagnoses, whose rules assign symptoms to a diagnosis. We are extending the tool for visualising change management rules from studies in organisational psychology. This provides a schematic representation of the findings: from the features (grey circles), consequences can be visualised depending on the values (which are not shown here). An example application is given in Figure 2, which illustrates the features and the relevant rules on which the feature `'acceptance of ERP system by employee'` depends transitively. The other nodes (shown by grey circles) are features, which can themselves be influenced by further features.

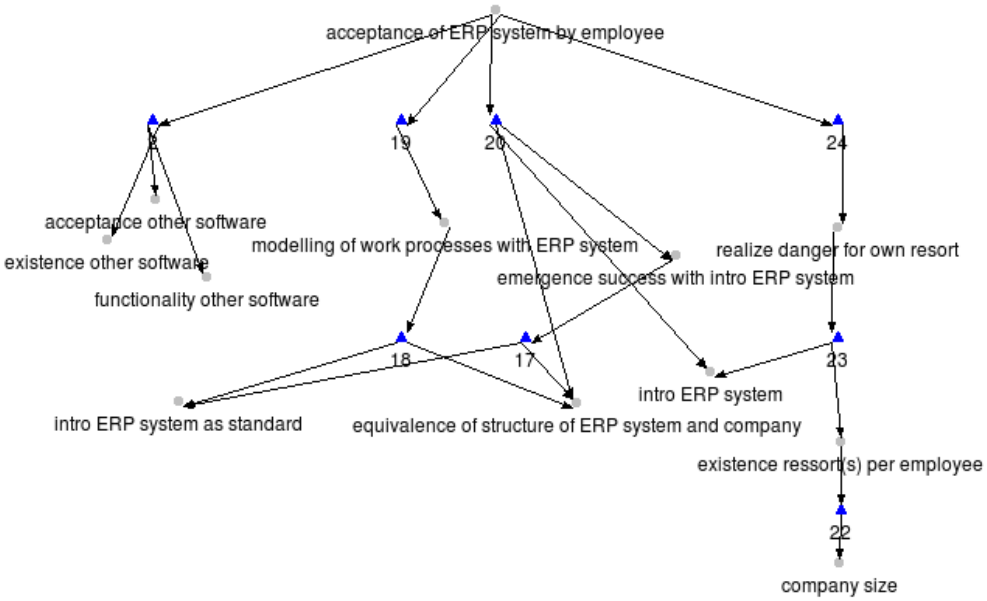


Figure 2 A dependency graph for the schematic representation of the transitive preconditions and the relevant rules (shown by the blue triangles labelled by `2, 17-20, 22-24`) for deriving the feature `'acceptance of ERP system by employee'`.

6 Extensions of the Knowledge Base

As already suggested, besides the pure features, the rules should be annotated by contextual information, such as their source, the method of achieving the rules, the time period of the investigation, and the confidence. The annotations can be used to deduce further constraints and implications from the rule base; the resulting statements about findings can be annotated with confidence values. For reasoning about the queries, provenance information is very important and can influence the usage of the interactive rule editor; for instance, unexpected interaction with other parts of the knowledge base can resort to provenance information to influence whether and how we accept or reject the new knowledge. The extensions can be expressed in DATALOG*, and thus DDBASE can integrate them with the evaluation in a consistent reasoning system.

6.1 Provenance Information in Ontologies

Provenance is information about entities, activities, and people involved in producing a piece of data, which can be used to assess its quality, reliability or trustworthiness. For collaborations across disciplines, hybrid information systems using data and techniques from many different sources with no preexisting agreement about the semantics of the processes or data, it is important to be able to express provenance. The infrastructure must provide general purpose mechanisms for annotating (i.e., making assertions about), discovering, and reasoning about processes and data.

The Open Provenance Model (OPM)

Some of the *inferences* require additional reasoning beyond that supported by OWL and SWRL. We assume that the reader is familiar with the basic concepts from ontologies; we do not get formal in this section; a general description of the semantic web rule language SWRL can, e.g., be found in [15, 2]. Also, structures such as the *provenance graph* are very useful for representing causal relationships. The *Open Provenance Model (OPM)* defines logical constraints on the provenance graph [15]. Some constraints cannot be expressed in OWL, but can be expressed based on SWRL rules. The Open Provenance Model provides a way to use semantic web technology and rules to implement semantic metadata. [15] discusses a binding of the OPM written in OWL with rules written in SWRL. This allows for the development of hybrid systems that use OWL, SWRL, and other semantic web software, interoperating through a shared space of RDF triples. PROV is a specification that provides a vocabulary to interchange provenance information. It defines a core data model for the interchange of *provenance* on the web; it allows for building representations of the entities, people and processes involved in producing a piece of data in the domain. The provenance of digital objects represents their origins; the records of a PROV specification can describe the entities and activities involved in producing and delivering or otherwise influencing a given object. Provenance can be used for many purposes, such as understanding how data was collected so it can be used meaningfully, determining ownership and rights over an object, making judgements about information to determine whether to trust it, verifying that the process and steps used to obtain a result complies with some given requirements, and reproducing how something was generated.

Example in Turtle Syntax

For example, the provenance of a conference paper could be described as follows: The paper was written by author `abc`. The final version of the paper is based on an earlier draft. Some professors made comments on the draft. The author cites prior work from a book. The paper includes a table that was generated by a program. This may be expressed in the so–called turtle syntax, which is a special language that could also be considered as a DSL. Note that the property "`a`" means "*is a*".

```
ex:draft
   a prov:Entity ;
   a abc:Manuscript ;
   dcterms:title "Latest results" .
ex:article a prov:Entity ;
   a abc:ConferencePaper ;
   dcterms:title "Results from case study" .
ex:dataset a prov:Entity ;
   a abc:Dataset .
ex:book
   a prov:Entity ;
   a abc:Thesis .
ex:result a prov:Entity ;
   a abc:Table .
ex:comment a prov:Entity ;
   a abc:Review .
```

Evaluation in Datalog*

Several of the key constraints and inferences of the OPM cannot be expressed in Owl and Swrl, due to fundamental limitations of the semantics of these languages. E.g., it is not possible to modify the value of an asserted property, or to write a rule to determine the number of times an artifact is used, or to detect a cycle in the provenance graph. Storing the OPM records in triples makes it possible to use other reasoning engines or languages such as Prolog or Datalog to implement queries or inferences. Owl and Swrl's Rdf representations provide a simple and well–understood means of exchanging provenance information with other tools, such as Rdf databases or *declarative* programming languages.

The hybrid system DDbase shows that semantic web technologies are not only useful for provenance information but also provide a base level of interoperability that can enable loosely–coupled tools with varying levels of capability and expressiveness. We do not need specialized reasoners for different knowledge bases. Instead, the rules are encoded in Datalog* and the provenance information is given in ontologies. Further Datalog* rules can encode the profile of the ontology. E.g., [10] study the controlled query evaluation for Datalog and Owl 2 profile ontologies. Then, we obtain a Datalog* knowledge base that can be evaluated in DDbase. Observe, that standard Datalog rules would not be sufficient here, since we need function symbols and embedded calls to Prolog.

6.2 Annotation of the Rule Base

The infrastructure must provide universal mechanisms for the annotation of rules for argumenting about processes and data. Similarly, the treatment of confidence values can be achieved. Frequently, collected values can be ambiguous. In the example above, our rule base contains the value `partly` in addition to `yes` and `no`. A more precise value in the form of *relative frequencies* could derive a more accurate form of knowledge.

Annotated Findings

The following simple example is a general annotated rule, where findings are annotated by values in the form `X:A=V`; the higher precedence of "`=`" in our domain–specifc language binds the finding `A=V` before it is annotated with the confidence value `X` by "`:`":

```
if A:'Existence of other Software' = yes
and B:'Functionality of other Software' = increasing
and C:'Acceptance of other Software' = increasing
and accumulate(conjunction_independence, [A,B,C], D)
then D:'Acceptance of ERP System' = decreasing .
```

The symbols A, B, C, and D in the rule represent logical variables, which always begin with a capital letter – which is common in the logic programming language Prolog. They are distinguishable from normal character strings, since they are included in quotation marks as 'emergence conflicts'. The variables in the rule body are universally quantified; i.e., the statement is assumed to hold for all suitable findings. The variables are in this case attached to the actual values, so that the unconditional probability can be calculated. Thus, our domain–specific language makes use of logical variables, and follows the syntax and semantics of predicate logic and its refinements in answer set programming.

In the case of stochastic independence, the predicate `accumulate` can be implemented as follows:

```
accumulate(conjunction_independence, Xs, X) :-
   multiply(Xs, X).
```

Annotated Datalog Rules

In general, the handling of annotated DATALOG rules has also been investigated by Lakshmanan, Subrahmanian, Kifer, et. al. [13, 11]. We have implemented the rules of Subrahmanian in DDBASE. Observe, that the implementation of `accumulate` above works for arbitrary lists `Xs` of values to obtain the product. In DDBASE, `multiply` is implemented using PROLOG meta–predicates. We have also implemented other forms of accumulating lists `Xs` of values, such as, e.g., positive and negative correlation. They can be used within the same knowledge base in DATALOG*.

Our approach is specified in terms of declarative rules that are given in domain–specifc languages. Rather than changing the inference engine in various forms, we keep the inference engine of DDBASE and change the declarative knowledge base of rules. We are thinking of adding a mode for specifying how to interpret variables as matching or evaluate. By analysing the individual rules, their dependencies, and the number of occurrences of individual features or findings, it is possible to determine the approximate impact of a single features, findings, or rules.

In short, having a good match with the underlying general purpose language while retaining a convenient user–friendly syntax and clear semantics is useful for an application–oriented DSL. This is clearly the case with DDBASE and PROLOG.

7 Final Remarks

Deductive databases allow for declaratively defining the *semantics* of the expert knowledge with rules. For representing the *syntax* of the rules, we use concepts from domain–specific languages – trying to remain usable within an adequate host language. In a case study for change management in organisational psychology, we have demonstrated the usefulness of the proposed approach in a practical situation. The analysis and visualisation of rules is also used successfully by AI people for medical diagnosis.

In the future, we are planning to apply knowledege engineering techniques, such as refactoring approaches [8], to the deductive rule bases. We will also incorporate further aspects of hybrid information sources and contextual annotations by, e.g., uncertainty and provenance information. Regarding the latter, it could be useful to model confidence and uncertainty with concepts from annotated logic programming [13, 11] and probabilistic–enabled logic programming languages, such as ProbLog [6], and to analyse and support the knowledge engineering and reasoning process for hybrid knowledge bases including these concepts.

Other extensions might deal with uncertain knowledge in the form of *disjunction* in the rule heads (conclusions), as described in, e.g., [14]. We expect that, especially, the combined handling of confidence values and disjunctive rules will be an interesting research field.

References

1 Serge Abiteboul. DATALOG: La renaissance. `http://www.college-de-france.fr/site/serge-abiteboul/course-2012-05-09-10h00.htm`, 2012.

2 Joachim Baumeister and Dietmar Seipel. Anomalies in ontologies with rules. *Journal of Web Semantics, Science, Services and Agents on the World Wide Web*, 8(1):55–68, 2010.

3 Ivan Bratko. PROLOG *Programming for Artificial Intelligence*. Addison–Wesley Longman, 4th edition, 2011.

4 Stefano Ceri, Georg Gottlob, and Letizia Tanca. *Logic Programming and Databases*. Springer, Berlin, 1990.

5 Upen S. Chakravarthy, Dan H. Fishman, and Jack Minker. Semantic query optimization in expert systems and database systems. In *Proceedings of the 1st International Workshop on Expert Database Systems*, pages 659–674, 1986.

6 Luc De Raedt, Angelika Kimmig, and Hannu Toivonen. ProbLog: A probabilistic Prolog and its application in link discovery. In *Proceedings of the 20th International Joint Conference on Artifical Intelligence*, pages 2468–2473, 2007.

7 Ramez Elmasri and Shamkant Navathe. *Fundamentals of Database Systems*. Benjamin Cummings, 7th edition, 2015.

8 Martin Fowler. *Refactoring – Improving the Design of Existing Code*. Addison–Wesley, 1999.

9 Martin Fowler. *Domain–Specific Languages*. Addison–Wesley, 2011.

10 Bernardo Cuenca Grau, Evgeny Kharlamov, Egor V. Kostylev, and Dmitriy Zheleznyakov. Controlled query evaluation for DATALOG and OWL 2 profile ontologies. `arXiv:1504.06529`, 2015.

11 Michel Kifer and V. S. Subrahmanian. Theory of generalized annotated logic programming and its applications. *Journal of Logic Programming*, 12(4):335–368, 1992.

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

13 Laks V. Lakshmanan and Fereidoon Sadri. On a theory of probabilistic deductive databases. *Theory and Practice of Logic Programming*, 1:5–42, 2001.

14 Jack Minker, Dietmar Seipel, and Carlo Zaniolo. Logic and databases: History of deductive databases. In *Handbook of the History of Logic*, volume 9, Computational Logic. North Holland, 2014.

15 Robert E. McGrath and Joeg Futrelle. Reasoning About Provenance with OWL and SWRL Rules. In *AAAI Spring Symposium*, 2008.

16 Dietmar Seipel. Practical applications of extended deductive databases in DATALOG*. In *Proceedings of the 23rd Workshop on Logic Programming (WLP 2009)*, September 2009.

17 Dietmar Seipel. Knowledge engineering for hybrid deductive databases. In *Proceedings of the 29th Workshop on Logic Programming (WLP 2015)*, September 2015.

18 Dietmar Seipel, Joachim Baumeister, and Marbod Hopfner. Declaratively querying and visualizing knowledge bases in XML. In *Proceedings of the 15th International Conference on Applications of Declarative Programming and Knowledge Management*, LNAI 3392, pages 16–31. Springer, 2005.

19 Jeffrey D. Ullman. *Principles of Database and Knowledge-Base Systems, Volume I*. Computer Science Press, 1988.

20 Jeffrey D. Ullman. *Principles of Database and Knowledge-Base Systems, Volume II*. Computer Science Press, 1989.

21 Rüdiger von der Weth, Dietmar Seipel, Falco Nogatz, Katrin Schubach, Alexander Werner, and Franz Wortha. Modellierung von handlungswissen aus fragmentiertem und heterogenem rohdatenmaterial durch inkrementelle verfeinerung in einem regelbanksystem. *Journal Psychologie des Alltagshandelns*, 2016.

22 Jan Wielemaker. An overview of the SWI–Prolog programming environment. In *Proceedings of the 13th International Workshop on Logic Programming Environments (WLPE)*, pages 1–16, 2003.

A Metamodel for Jason BDI Agents

Baris Tekin Tezel[1], Moharram Challenger[2], and Geylani Kardas[3]

1 **International Computer Institute, Ege University, Izmir, Turkey**
`baris.tezel@deu.edu.tr`
2 **International Computer Institute, Ege University, Izmir, Turkey**
`moharram.challenger@mail.ege.edu.tr`
3 **International Computer Institute, Ege University, Izmir, Turkey**
`geylani.kardas@ege.edu.tr`

Abstract

In this paper, a metamodel, which can be used for modeling Belief-Desire-Intention (BDI) agents working on Jason platform, is introduced. The metamodel provides the modeling of agents with including their belief bases, plans, sets of events, rules and actions respectively. We believe that the work presented herein contributes to the current multi-agent system (MAS) metamodeling efforts by taking into account another BDI agent platform which is not considered in the existing platform-specific MAS modeling approaches. A graphical concrete syntax and a modeling tool based on the proposed metamodel are also developed in this study. MAS models can be checked according to the constraints originated from the Jason metamodel definitions and hence conformance of the instance models is supplied by utilizing the tool. Use of the syntax and the modeling tool are demonstrated with the design of a cleaning robot which is a well-known example of Jason BDI architecture.

1998 ACM Subject Classification I.6.5 Model Development, I.2.11 Multiagent systems

Keywords and phrases metamodel, BDI agent, multi-agent system, Jason

Digital Object Identifier 10.4230/OASIcs.SLATE.2016.8

1 Introduction

In agent-oriented software engineering (AOSE), many metamodels (e.g. [1, 18, 19, 20, 13, 2, 7, 10]) exist for describing the intelligent software agents and multi-agent systems (MASs) which are composed of these agents. A group of these metamodels (e.g. [1, 18, 2]) only provides the definition of traditional AOSE methodologies [14] while another group (e.g. [19, 12, 13, 7, 10]) aims at more general specification of agent systems from different aspects varying from agent internals to MAS organizations. They present abstract syntaxes for domain-specific MAS modeling languages (such as [17, 12, 8, 11]) and hence enable the model-driven development (MDD) of MAS [15]. Taking into account the OMG's Model-driven architecture (MDA) specifications[1], it is proper to indicate that above metamodels support the platform-independent MAS modeling which is abstract from the underlying agent implementation and/or execution platforms.

On the other hand, platform-specific modeling of agent systems is also possible by using metamodels. AOSE researchers propose metamodels which are specific to real MAS

[1] OMG Model Driven Architecture, `http://www.omg.org/mda/`.


© Baris Tekin Tezel, Moharram Challenger, and Geylani Kardas;
licensed under Creative Commons License CC-BY

implementation platforms such as JACK[2] , JADE[3] or JADEX[4]. For instance, [13] defines the metamodels of JADE and JACK platforms while [16] and [8] consider the development of MAS on JADEX platform. Definition of MAS metamodels at these different abstraction levels and application of model transformations between these platform-independent and platform-specific MAS metamodels enable the implementation of modeled agent systems in abovementioned MAS platforms.

This paper introduces our ongoing work on the platform-specific modeling of Belief-Desire-Intention (BDI) agents [22] according to the specifications and features of Jason platform[5] which provides the implementation of agents using a Prolog-like logic programming language called AgentSpeak [21]. We discuss the derivation of a metamodel for Jason and construction of a graphical modeling tool in which agent developers can model Jason systems conforming to the introduced metamodel. We believe that the work presented herein contributes to the abovementioned MAS metamodeling efforts by taking into account another BDI agent platform which is not considered in the existing platform-specific MAS modeling approaches. The work introduced in [9] is the single exception which also aims at the modeling of Jason agents. However, the given metamodel does not support the reusability of same concepts like beliefs, events, actions plans, goals and rules for different agents of a MAS if required. The metamodel proposed in this study supports the reusability of all related entities inside the whole MAS model.

Rest of the paper is organized as follows. In section 2, Jason platform is briefly discussed. Section 3 introduces the metamodel we propose for Jason. Section 4 covers the definition of the concrete syntax, implementation of a modeling environment for Jason agents and exemplification of its usage. Section 5 concludes the paper.

2 Jason

Jason [3] is a Java-based interpreter for an extended version of a Prolog-like logic programming language called AgentSpeak [21]. AgentSpeak is based on the well-known BDI architecture [22] for software agents which originates from Bratman's human practical reasoning theory [5]. In BDI architecture, agents constantly monitor their environment and respond instantly to the changes in the environment. This reaction depends on agent's mental attitudes. An agent has three types of mental attitudes which are belief, desire and intention.

Beliefs are information about an agent's itself, other agents and the environment that the agent is located. Desires express all possible states of affairs which might be achieved by an agent. One desire is a potential trigger for an agent's actions. Simply, desires are often considered as options for an agent. Finally, intentions represent the states of affairs which have been decided to work towards by the agent. Intentions may be delegated goals or results of considered options.

Simply, an AgentSpeak agent is defined by a set of beliefs, rules and plans. Beliefs represent initial knowledge of an agent. Rules are logic expressions or mathematical equations. Plans constitute the actions and/or subgoals to achieve the current goal.

A plan of an AgentSpeak agent consists of a triggering event, a context and a body element. The triggering event specifies the events for which that plan is suitable. The context

[2] JACK Autonomous Software, `http://aosgrp.com/products/jack/`.
[3] JAVA Agent DEvelopment Framework, `http://jade.tilab.com/`.
[4] JADEX Active Components, `https://www.activecomponents.org/#/project/news`.
[5] Java-based interpreter for an extended version of AgentSpeak, `http://jason.sourceforge.net/wp/`.

represents whether the plan is applicable according to the beliefs of the agent. Body is a sequence of basic actions and/or subgoals.

Logic programming based on AgentSpeak [4] serves a computationally efficient capability for BDI agent development. With providing a Java-based interpreter, Jason extends the expressiveness of AgentSpeak during implementation of cognitive agents.

3 A metamodel for Jason

A metamodel, which may provide an abstract syntax of a modeling language for Jason platform, is presented in this section. As shown in Figure 1, the metamodel provides the meta-entities and their relations required for both the internal BDI agent architecture and MAS organization on Jason platform. Derivation of the main elements and their associations is mainly based on the definitions given in [4]. Moreover, the metamodel complies with the Extended Backus-Naur Form (EBNF) of Agent Speak Language [4] with the required level of abstractions. During the discussion below, the meta-entities of the proposed Jason metamodel are given in the text with italic font.

A *Multi-agent System* (MAS) is mainly composed of agents. However, the metamodel includes five different entities other than the *Agent* entity for the complete representation of the MAS composite structure. These are *BeliefBase*, *EventSet*, *RuleSet*, *PlanLibrary*, *ActionSet* and *GoalSet*. Each of them is denoted by individual meta-elements which are owned by the agents. This helps the metamodel's support on the reusability for the agent developers.

BeliefBase is consisted of possible beliefs which may be adopted by agents in order to use for initial beliefs or context of plans. But, above mentioned *BeliefBase* of a MAS is different from the belief base of individual agents. It has a static structure while belief base of an agent can be changed during the execution of reasoning cycles. Also, each belief represents knowledge about an agent or the environment. *EventSet* contains events within the environment. In Jason platform, each event is represented by a triggering event which is also a meta-element (*Triggering_Event*) included in the proposed metamodel. Events happen as a consequence of belief or goal changes inside an agent's mind. *RuleSet* is composed of rules. Each *Rule*, which allows arriving at a judgement based on beliefs of an agent, can simplify making certain conditions used in the context of plans. Rule concept is directly related with logic programming especially in Prolog.

In the metamodel, *ActionSet* stores actions which can be used by all agents. Actions, which are included in the body of plans, represent what an agent is capable of performing. The Jason metamodel provides two kinds of actions which are internal actions and external actions. Internal actions (represented by the *Internal_Action* meta-entity) are executed inside an agent's mind. So, they cannot change the environment. Communication actions of an agent, which are called as *Messages* in the Jason metamodel, are also internal actions. On the other hand, external actions (represented by *External_Action* meta-entities) directly change the environment.

An agent has *Goal*s to achieve. Bringing together all candidate goals for each agent within the environment creates the *GoalSet*. In fact, goals are the desired states, which are achieved by the executed plans. There are two types of goals, including achievement goals and test goals. An achievement goal represents a state of the environment which is desired to be achieved by an agent. A test goal is used to retrieve knowledge from beliefs of an agent or to check something expected what is actually believed by the agent, while executing a plan body. Those agent types are included in the metamodel as being attributes

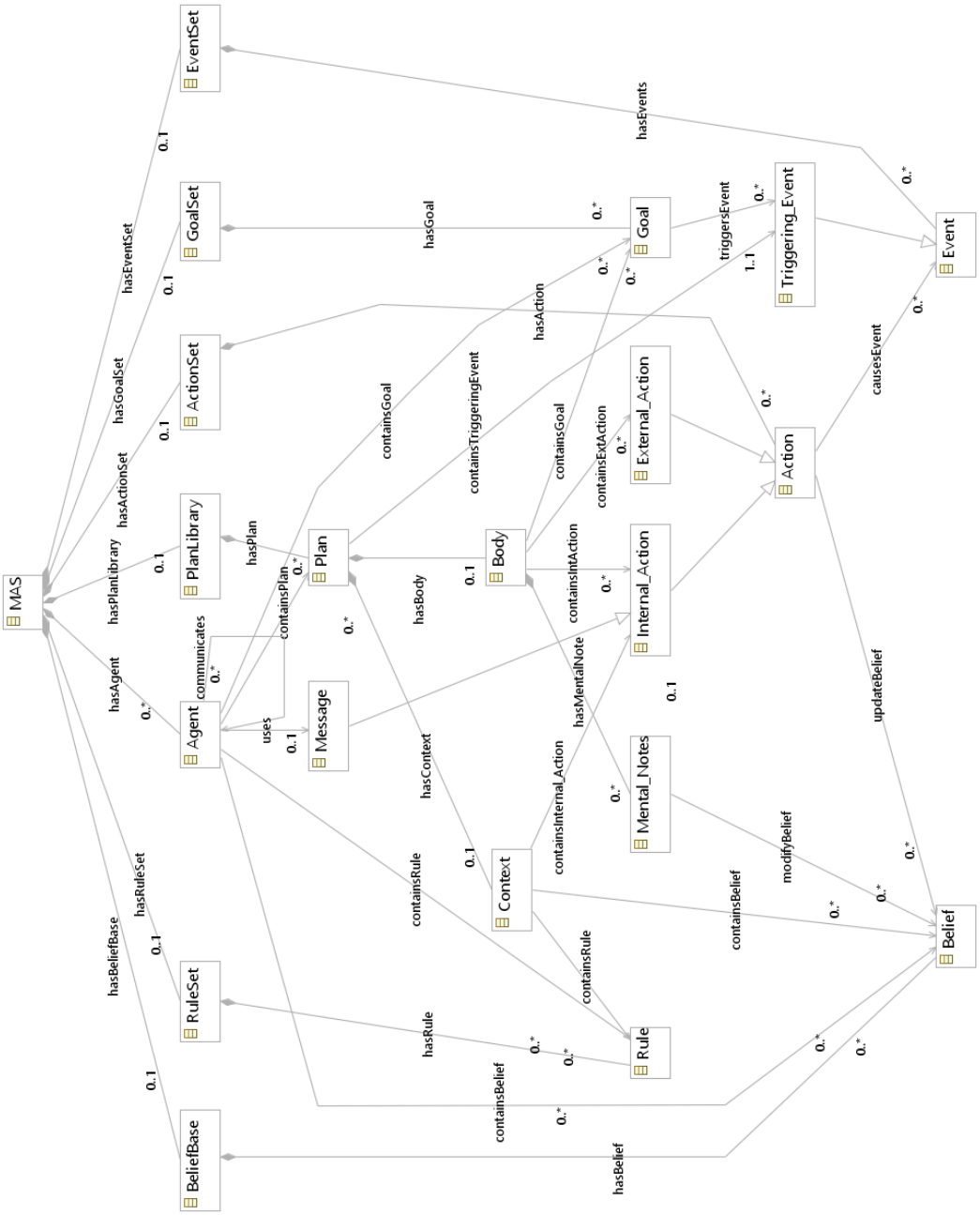


Figure 1 The Jason metamodel.

of *Goal* meta-entity. However, these attributes can not be shown in Figure 1 since attribute compartments are closed in the metamodel given in the figure due to space limitations.

An agent reacts against events. Those reactions of an agent are represented in the metamodel by *Plans*. As previously mentioned, a plan, which represents the skills of an agent, has three distinct parts. These are the triggering event, the context and the body. *Triggering_Event* is the post-condition of a plan. *Context* is the pre-condition of a plan and composed of beliefs and rules which allow deducing based on the knowledge. The *Body* of a plan is simply a list of actions. Besides, the body has sub-goals and *Mental_Notes* which modify the belief base. All possible plans which can be used by an agent, are covered in a *PlanLibrary* meta-element. Finally, an agent has initial beliefs, plans and pursuing goals. Initial beliefs and pursing goals are adopted from the related sets defined in the metamodel (e.g. *BeliefBase*, *GoalSet*). Also, plans of an agent are selected from the *PlanLibrary*.

4 Concrete Syntax

Whilst the specification of abstract syntax inside a metamodel includes those concepts that are represented in the language and the relationships between those concepts, concrete syntax definition provides a mapping between meta-elements and their representations for models. In fact, the concrete syntax is the set of notations which facilitates the presentation and construction of the language. This section discusses the graphical concrete syntax which maps the Jason platform's abstract syntax elements presented in the previous section to their graphical notations.

We use Epsilon EuGENia[6] for constructing the concrete syntax from the Jason metamodel. Providing a tool for implementing a graphical modeling editor from a single annotated Ecore metamodel and hence facilitating the construction of a concrete syntax of a metamodel caused us to prefer EuGENia in this study. After setting the graphical notations for the abstract syntax meta-elements, we employ EuGENia to tie notations to the domain concepts. The graphical notations are listed in Table 1. Meta-elements which are not used directly during the creation of a Jason instance model, do not have notations in the table. For instance, the Action meta-element, which is inherited by internal action and external action, is not needed in an instance model. But, internal action and external action have to be existed in an instance model. Moreover, some composition relations, which are modeled with compartments in their holder elements, do not have graphical notations. On the other hand, rest of the composition relations which are modeled with connections to their holder elements, have the same graphical notations with their owner.

Use of the derived concrete syntax inside the EuGENia-based modeling editor can impose some restrictions/controls for the conformance of designed models to the specifications of our Jason metamodel. Benefiting from adopting the Ecore as the meta-metamodel while the production of the graphical modeling tool, the graphical concrete syntax succeeds in providing the check of constraints such as compartment, number of relationships between model elements and source and destination elements in a relationship. Finally, defined inheritance relationship in the Jason metamodel obligates users while modelling. Inheritance relationship constraint in question checks whether a sub-entity in a Jason instance model includes all of the attributes and relationships of its super-entity.

In order to demonstrate the use of the proposed concrete syntax during modeling a Jason MAS, let us consider the design of a cleaning robot. Cleaning robot is one of the well-known

[6] Epsilon Eclipse GMT Component: EuGENia, `http://www.eclipse.org/epsilon/doc/eugenia/`.

Figure 2 Instance model for the cleaning robot example designed inside the provided graphical Jason modeling editor.

Table 1 Jason concepts and their notations provided for the graphical concrete syntax.

Concept	*Notation*	*Concept*	*Notation*
Agent		Rule	
Belief Base		Plan	
Event Set		Context	
Rule Set		Body	
Plan Library		Internal Action	
Action Set		External Action	
Goal Set		Message	
Belief		Goal	
Triggering Event		Mental Note	

examples of Jason BDI architecture which is provided by Bordini and Hubner [3] for the scenario described in [6].

The screenshot from our EuGENia-based modeling tool, given in Figure 2, shows the instance model conforming to Jason metamodel designed for the cleaning robot example. The cleaning robot example consists of robots (agent instances) that collect and burn garbage. In our example, *robot1* is responsible for collecting garbage. In order to collect garbage, *robot1* has different plans. The agent adopted each plan from the plan library again given in the instance model. Also, each plan mainly covers the required action which is adopted from the action set. So, when we add an action to the action set of the model, it is available for any plan. This idea works not only for actions but also works for beliefs, goals, events and rules. The model given in Figure 2 shows that *robot1* agent's main goal is checking slots. The agent continues to search until finding garbage. When it finds garbage, it desires to carry garbage toward *robot2* which burns garbage. Whole system runs until there is no garbage in the environment.

5 Conclusion

A metamodel, which can be used for modeling Jason BDI agents, has been introduced in this paper. The metamodel provides the modeling of agents with including their belief bases, plans, and sets of events, rules and actions respectively. Derivation of a graphical concrete syntax based on this metamodel also enabled us to develop a graphical modeling tool for the MDD of Jason agents. In this tool, MAS models can be checked according to the constraints originated from the Jason metamodel definitions and hence conformance of the instance models is supplied.

Our next work will include automatic code generation from the instance models designed with the tool introduced in this paper. Hence, agent developers will achieve executables for Jason platform via MAS modeling. Following this work, our aim is to integrate Jason metamodel and its modeling tool into the tool set of our existing domain-specific MAS modeling language called SEA_ML [8]. Hence, it will be possible to model BDI agents in the platform-independent level and after executing a chain of model-to-model and model-to-text transformations, agent developers can automatically achieve the implementation of their MAS models in the Jason platform.

References

1 Carole Bernon, Massimo Cossentino, Marie-Pierre Gleizes, Paola Turci, and Franco Zambonelli. A study of some multi-agent meta-models. In *5th Int'l Workshop on Agent-Oriented Software Engineering*, volume 3382, pages 62–77, 2005.

2 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.

3 Rafael H. Bordini and Jomi F. Hübner. BDI agent programming in Agentspeak using Jason. In *Proceedings of the 6th International Conference on Computational Logic in Multi-Agent Systems*, pages 143–164, Berlin, Heidelberg, 2006. Springer-Verlag. doi:10.1007/11750734_9.

4 Rafael H. Bordini, Jomi Fred Hübner, and Michael Wooldridge. *Programming Multi-Agent Systems in AgentSpeak Using Jason*. John Wiley & Sons, 2007.

5 Michael E. Bratman. *Intention, Plans, and Practical Reason*. Harvard University Press, Cambridge, Massachusetts, 1987.

6 Lars Braubach, Alexander Pokahr, Daniel Moldt, and Winfried Lamersdorf. Goal representation for bdi agent systems. In *Proceedings of the Second International Conference on Programming Multi-Agent Systems*, pages 44–65, Berlin, Heidelberg, 2005. Springer-Verlag. doi:10.1007/978-3-540-32260-3_3.

7 MMoharram Challenger, SSebla Demirkol, SSinem Getir, and Geylani Kardas. A domain specific metamodel for semantic web enabled multi-agent systems. In *1st International Workshop on Domain Specific Engineering*, volume 83, pages 177–186, 2011.

8 Moharram Challenger, Sebla Demirkol, Sinem Getir, Marjan Mernik, Geylani Kardas, and Tomaž 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.

9 Massimo Cossentino, Antonio Chella, Carmelo Lodato, Salvatore Lopes, Patrizia Ribino, and Valeria Seidita. A notation for modeling Jason-like BDI agents. In *Complex, Intelligent and Software Intensive Systems (CISIS), 2012 Sixth International Conference on*, pages 12–19, July 2012. doi:10.1109/CISIS.2012.203.

10 Iván Garcia-Magarino. Towards the integration of the agent-oriented modeling diversity with a powertype-based language. *Computer Standards & Interfaces*, 36:941–952, 2014.

11 Enyo José Tavares Goncalves, Mariela Inés Cortes, Gustavo Augusto Lima Campos, Yrleyjander Salmito Lopes, Emmanuel Sávio Ssilva Freire, Viviane Torres da Silva, Kleiner Silva Farias de Oliveira, and Marcos António de Oliveira. MAS-ML2.0: Supporting the modelling of multi-agent systems with different agent architectures. *The Journal of Systems and Software*, 108:77–109, 2015.

12 Christian Hahn. A domain specific modeling language for multiagent systems. In *7th Int'l Conf. on Autonomous agents and Multi-agent systems (AAMAS 2008)*, pages 223–240, 2008.

13 Christian Hahn, Cristián Madrigal-Mora, and Klaus Fischer. A platform-independent metamodel for multiagent systems. *Autonomous Agents and Multi-Agent Systems*, 18(2):239–266, 2009.

14 Brian Henderson-Sellers and Paolo Giorgini. *Agent-oriented Methodologies*. Idea Group Publishing, 2005.

15 Geylani Kardas. Model-driven development of multiagent systems: a survey and evaluation. *The Knowledge Engineering Review*, 28(4):479–503, 2013.

16 Geylani Kardas, Erdem Eser Ekinci, Bekir Afsar, Oguz Dikenelli, and N. Yasemin Topaloglu. Modeling tools for platform specific design of multi-agent systems. In Lars Braubach,

Wiebe van der Hoek, Paolo Petta, and Alexander Pokahr, editors, *Multiagent System Technologies: 7th German Conference, MATES 2009*, pages 202–207. Springer, Berlin, Heidelberg, 2009. doi:10.1007/978-3-642-04143-3_20.

17 Uirá Kulesza, Alessandro Garcia, Carlos Lucena, and Paulo Alencar. A generative approach for multi-agent system development. In Ricardo Choren, Alessandro Garcia, Carlos Lucena, and Alexander Romanovsky, editors, *Software Engineering for Multi-Agent Systems III: Research Issues and Practical Applications*, pages 52–69. Springer, Berlin, Heidelberg, 2005. doi:10.1007/978-3-540-31846-0_4.

18 Ambra Molesini, Enrico Denti, and Andrea Omicini. MAS meta-models on test: UML vs. OPM in the SODA case study. In *Proceedings of the 4th International Central and Eastern European Conference on Multi-Agent Systems and Applications*, pages 163–172, Berlin, Heidelberg, 2005. Springer-Verlag. doi:10.1007/11559221_17.

19 James Odell, Marian Nodine, and Renato Levy. A metamodel for agents, roles, and groups. In James Odell, Paolo Giorgini, and Jörg P. Müller, editors, *Agent-Oriented Software Engineering V*, pages 78–92. Springer, Berlin, Heidelberg, 2005. doi:10.1007/978-3-540-30578-1_6.

20 Andrea Omicini, Alessandro Ricci, and Mirko Viroli. Artifacts in the A&A meta-model for multi-agent systems. *Autonomous Agents and Multi-Agent Systems*, 17(3):432–456, 2008.

21 Anand Rao. AgentSpeak(L): BDI agents speak out in a logical computable language. In *7th European Workshop on Modelling Autonomous Agents in a Multi-Agent World, Eindhoven, The Netherlands, Lecture Notes in Computer Science*, volume 1038, pages 42–55, 1996.

22 Anand S. Rao and Michael P. Georgeff. Decision procedures for BDI logics. *Journal of Logic and Computation*, 8(3):293–343, 1998.

Profile Detection Through Source Code Static Analysis

Daniel Ferreira Novais[1], Maria João Varanda Pereira[2], and Pedro Rangel Henriques[3]

1 **Departamento de Informática & Centro Algoritmi, Universidade do Minho, Braga, Portugal**
`danielnovais92@gmail.com`

2 **Departamento de Informática e Comunicações & Centro Algoritmi, Instituto Politécnico de Bragança, Bragança, Portugal**
`mjoao@ipb.pt`

3 **Departamento de Informática & Centro Algoritmi, Universidade do Minho, Braga, Portugal**
`pedrorangelhenriques@gmail.com`

Abstract

The present article reflects the progress of an ongoing master's dissertation on language engineering. The main goal of the work here described, is to infer a programmer's profile through the analysis of his source code. After such analysis the programmer shall be placed on a scale that characterizes him on his language abilities. There are several potential applications for such profiling, namely, the evaluation of a programmer's skills and proficiency on a given language or the continuous evaluation of a student's progress on a programming course. Throughout the course of this project and as a proof of concept, a tool that allows the automatic profiling of a Java programmer is under development. This tool is also introduced in the paper and its preliminary outcomes are discussed.

1998 ACM Subject Classification D.2.8 Metrics, D.3.4 Processors

Keywords and phrases Static analysis, metrics, programmer profiling

Digital Object Identifier 10.4230/OASIcs.SLATE.2016.9

1 Introduction

Proficiency on a programming language can be compared to proficiency on a natural language [7]. Using, for example, the *Common European Framework of Reference for Languages: Learning, Teaching, Assessment* (CEFR) method[1] it is possible to classify individuals based on their proficiency on a given foreign language. Similarly, it may be possible to create a set of metrics and techniques that allow the profiling of programmers based both on proficiency and abilities on a programming language.

In [6], the main inspiration behind this project, Pietrikova explores techniques aiming the evaluation of Java programmers' abilities through the static analysis of their source code. Static code analysis may be defined as the act of analysing source-code without actually executing it, as opposed to dynamic code analysis which is done on executing programs. It's usually performed with the goal of finding bugs or ensure conformance to coding guidelines.

[1] `http://www.coe.int/t/dg4/linguistic/cadre1_en.asp`


© Daniel Ferreira Novais, Pedro Rangel Henriques, and Maria João Varanda Pereira;
licensed under Creative Commons License CC-BY

For the present paper, static analysis will be used to extract metrics from source-code related with language usage practices.

Building on the referred paper, the goal is to further explore the discussed techniques and introduce new ones to improve that evaluation, with the ultimate goal of creating a tool that automatically profiles a programmer.

The basic idea is to statically analyse a Java programmer's source code and extract a selection of metrics that can either be compared to *standard solutions* (considered *ideal* by the one willing to obtain the profiles) or, using machine learning techniques, subjected to a classification model in order to be assigned the appropriate profile. The attributes or metrics that will allow us to infer a profile based on sets of previously classified programs can be defined a-priori by hand (using intuition) or can be extracted through data-mining techniques, as Kagdi et al. explored [5]. However this last approach requires the availability of huge collections of programs assigned to each class.

The programmers will be classified generically as for their language proficiency or skill, for example, as novice, advanced or expert. Other relevant details are also expected to be provided, such as the classification of a programmer on his code readability (indentation, use of comments, descriptive identifiers), defensive programming, among others.

Below are some source-code elements that can be analysed to extract the relevant metrics to appraise the code writer's proficiency:

- Statements and Declarations
- Repetitive patterns
- Lines (code lines, empty lines, comment lines)
- Indentation
- Identifiers
- Good practices

Code with errors will not be taken into consideration for the profiling. This is, only correct programs producing the desired output will be used for profiling.

To build the system discussed in this paper we intend to develop a metric extractor program, to evaluate the set of parameters that we chose for the profiling process. However this process will be complemented with the use of a tool, called PMD[2], to get information of the use of good Java programming practices. PMD is a source code analyser that finds common programming flaws like unused variables, empty catch-blocks, unnecessary object creation, and so forth. For these reasons it is a tool that may prove to be very useful.

The rest of the paper is organized as follows. In Section 2 we will review the area and present related work, in order to identify techniques and tools commonly used to deal with this problem. Section 3 is devoted to present our proposal for an automatic programmer profiling system based on source code analysis. The analyzer implemented and the set of metrics extracted at the present project stage will be introduced in Section 4. In Section 5 we will discuss the profiling results so far inferred correlating the values provided by the comparison between the programmer's code metrics and the standard solution. The paper is closed at Section 6 with some conclusions and future work.

[2] `https://pmd.github.io/`

2 Programmer Profiling: approaches and tools

As mentioned before, the main motivation for this project came from the study [6] of Pietriková and Chodarev. These authors propose a method for profiling programmers 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 refers to the actual knowledge necessary to handle those tasks. It can be viewed as a target or a model to follow.

The profile is generated by counting language constructs and then comparing the numbers to the ones of previously developed optimal solutions for the given tasks. Through that comparison it's possible to find gaps in language knowledge. The authors agree that the tool is promising, but there is still a lot of work that can be done on the subject. To compare programs against models or ideal solutions, by counting language constructs is a common feature between this work and our project. Despite that, in this work, the object profile is optional. The subject profile can be inferred analysing the source code, using as base the language grammar. Considering the language syntax, a set of metrics are extracted from the source code. This can be done to conclude about the complexity of a program or to perform some statistics when analysing a set of programs of one programmer. In our case, we are not concerned with the complexity level of the programs but we analyse the way each programmer solves a concrete problem. So, almost all metrics that we extract only make sense when compared with a standard solution.

In another paper [9], Truong et al. suggest a different approach. Their goal is the development of a tool, to be used throughout a Java course, that helps students learning the language. Their tool provides two types of analysis: Software engineering metrics analysis and structural similarity analysis. The former checks the students programs for common poor programming practices and logic errors. The latter provides a tool for comparing students' solutions to simple problems with model solutions (usually created by the course teacher). Despite having several limitations, teachers have been giving this tool a positive feedback. As stated before, this thesis will be taking a similar approach to this software engineering metrics. However, the tool above mentioned was only used on an academic context while the purpose of this project is to develop a tool that can also be applied in another contexts.

Flowers et al. [1] and Jackson et al. [4] discuss a tool developed by them, *Gauntlet*, that allows beginner students understanding Java syntax errors committed while taking their Java courses. This tool identifies the most common errors and displays them to students in a friendlier way than the Java compiler. *Expresso tool* [3] is also a reference on Java syntax, semantic and logic error identification. Both tools have been proven to be very useful to novice Java learners but since they focus mainly on error handling, they will not be very useful for this project.

Hanam et al. explain [2] how static analysis tools (e.g. *FindBugs*) can output a lot of false positives (called unactionable alerts) and they discuss ways to, using machine learning techniques, reduce the amount of those false positive so a programmer can concentrate more on the real bugs (called actionable alerts). This study may prove to be very useful to this work since there is an intention of exploring similar machine learning and data mining techniques on the analysis.

3 Programmer Profiling: Our proposal

3.1 Programmer Profiles

Programmer profiling is an attempt to place a programmer on a scale by inferring his profile. As Poss stated [7], we can compare proficiency on a programming language with proficiency on a natural language, and like the CEFR has a method of classifying individuals based on their proficiency on a given foreign language, it is believed that the same can be done for a programming language.

CEFR defines foreign language proficiency at six levels: A1, A2, B1, B2, C1 and C2 (A1 meaning the least proficient and C2 the most proficient). A similar method for classifying programmers was considered at first, but due to the fact that the levels were not very descriptive, a more self-described scale was preferred.

Sutcliffe presents [8] a classification for programmer categorization: naive, novice, skilled and expert. A similar scale was agreed upon, with what it is believed to be a good starting point for the profiling:

Novice
- Is not familiar with all the language constructs
- Does not show language readability concerns
- Does not follow *good programming practices*

Advanced Beginner
- Shows variety in the use of language constructs and data-structures
- Begins to show readability concerns
- Writes programs in a safely[3] manner

Proficient
- Is familiar with a great variety of language constructs
- Follows *good programming practices*
- Shows readability and code-quality concerns

Expert
- Masters a great variety of language constructs
- Focuses on producing efficient code (minimizing resources or lines of code) without readability concerns

The following (a little bit exaggerated) example (see Listing 1) may help to shed some light on what is meant by the previous scale. Each method has the same goal: to calculate the sum of the values of an integer array, and has features of what may be expected for each class. It's hard to represent all 4 classifications on such a small example, so the Advanced Beginner was left out.

The novice has little or no concern with code readability. He will also show lack of knowledge of language features. In the example we can see that by the way he spaces his code, writes several statements in one line or gives no meaning in variable naming. He also shows lack of advanced knowledge on assignment operators (he could have used the *add and assignment* operator, `+=`).

The expert, much like the novice, shows no concern for language readability, but unlike the latter, he has more language knowledge. That means that the expert has a different kind

[3] e.g. writes `if (cond==0)` instead of `if (!cond)` as is done by people that have more self-confidence and usually have a not so obvious way of programming.

Listing 1 Examples of programs corresponding to different Profile Levels.

```
int novice (int[] list) {
    int a=list.length;
    int b;int c= 0;
    for (b=0;b<a;b++) {
      c=c+list[b];}
    return c;
}

//Sums all the elements of an array
int proficient (int[] list) {
    int len = list.length;
    int i, sum = 0;
    for (i = 0; i < len; i++) {
        sum += list[i];
    }
    return sum;
}

int expert (int[] list) {
    int s = 0;
    for (int i : list) s += i;
    return s;
}
```

of bad readability. The code can be well organized but the programming style is usually more compact and it's harder to understand. As an example of language knowledge the expert uses the *extended for loop*, making his code a lot smaller.

Finally, the proficient will show the skill and knowledge of an expert programmer while keeping concern with code readability and appearance. The code will feature advanced language constructs while maintaining readability. His code will be clear and organized, variable naming has meaning and code will have comments for better understanding.

3.2 System architecture

As mentioned previously, this project will be complemented with a tool, developed in Java, that intends to put into action the theory behind the project.

Figure 1 shows the block diagram that represents the expected final implementation of the system. The tool will be named *Programmer Profiler* (PP).

The programmer's Java source code is loaded as PP input. Then, the code goes through two static analysis processes: the analyser implemented (PP-Analyser) using AnTLR with the goal of extracting a set of metrics and the PMD-Analyser, an analyser that resorts to the PMD Tool to find a set of predefined metrics regarding poor coding practices.

Both analysis' outcomes will feed two other modules: A *Metrics Visualizer* (a generator of HTML pages [4]) which will allow us to make a manual assessment of the source code to infer the programmer's profile; and a *Profile Inference Engine* whose goal is to make the profile assignment an automatic process.

[4] `http://www4.di.uminho.pt/~gepl/PP/`

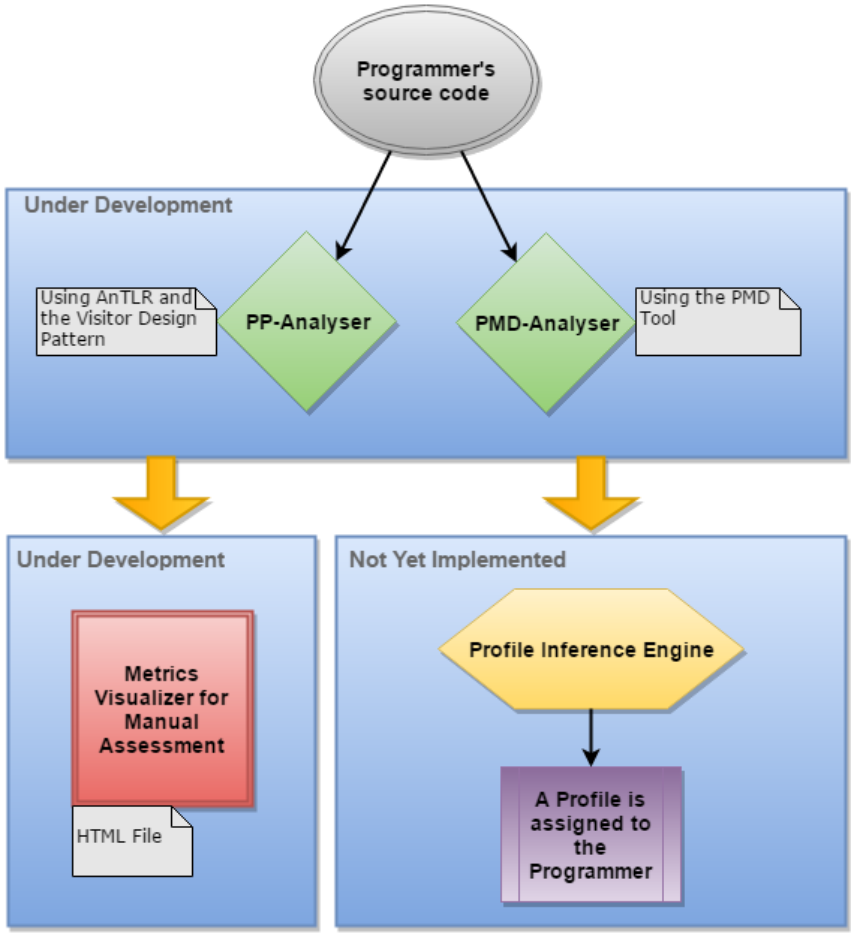


Figure 1 PP Block Diagram.

Making the profiling an automatic assignment will be the most interesting, challenging and complex part of this project. The goal is not to assign an absolute value that characterizes a programmer's proficiency on the Java language, but instead to give a general classification in regards to a resolution of a given problem or task.

3.3 Tools being used

To implement PP some tools were very helpful throughout the development process. Below we describe the two most relevant: AnTLR and PMD.

3.3.1 AnTLR

As taken from the website[5]:

> ANTLR (ANother Tool for Language Recognition) is a powerful parser generator for reading, processing, executing, or translating structured text (...). From a grammar, ANTLR generates a parser that can build and walk parse trees.

[5] `http://www.antlr.org/`

Using only a Java grammar, AnTLR will generate a parser that will read any syntactically correct Java source file. This allows us to easily manipulate the information that a source file contains. This means that AnTLR allows us to extract, with more or less ease, any kind of metric needed from the Java files.

As mentioned on the previous chapter, instead of implementing features that search for poor Java practices and novice programming mistakes, PMD, a very useful source code analyser, was selected.

3.3.2 PMD

PMD[6] is a source code analyser. It finds common programming flaws like unused variables, empty catch-blocks, unnecessary object creation, and so forth.

PMD executes a thorough analysis over source-code (it supports several languages) and reports back the possible programming flaws in the form of violations. PMD looks for dozens of poor programming practices, nonconformity to conventions and security guidelines, being a promising asset to this project.

Below are some of the main PMD Rulesets[7] that may be the most useful to our goals:

Unused Code
The Unused Code Ruleset contains a collection of rules that find unused code

Optimization
These rules deal with different optimizations that generally apply to performance best practices

Basic
The Basic Ruleset contains a collection of good practices which everyone should follow

Design
The Design Ruleset contains a collection of rules that find questionable designs

Code Size
The Code Size Ruleset contains a collection of rules that find code size related problems

Naming
The Naming Ruleset contains a collection of rules about names - too long, too short, and so forth

Braces
The Braces Ruleset contains a collection of braces rules

4 Metrics extraction: source code analysis

As mentioned before the *Programmer Profiler* (PP) tool will consist of two performed analysis. An implemented PP-Analyser, which resorts to metrics extraction, with the goal of comparing with model solutions and a PMD-Analyser, which uses the PMD tool to detect poor programming practices in an absolute manner (not resorting to comparison).

[6] `https://pmd.github.io/`
[7] `https://pmd.github.io/pmd-5.4.1/pmd-java/rules/index.html`

Listing 2 Violation detected by PMD.

```
<violation beginline="274" endline="276" begincolumn="33"
endcolumn="33" rule="CollapsibleIfStatements" ruleset="Basic"
package="(...)" class="IMC" method="MT"
externalInfoUrl="https://pmd.github.io/pmd-5.4.0/pmd-java/rules/
java/basic.html#CollapsibleIfStatements" priority="3">
These nested if statements could be combined</violation>
```

4.1 PP-Analyser

The PP-Analyser, which extracts metrics that allow comparing possible solutions with optimal ones was implemented using AnTLRv4 and so far a variety of metrics is being extracted:

1. Class hierarchy
2. Class and method names and sizes
3. Variable names and types
4. Number of files, classes, methods, statements
5. Number of lines code and comment
6. Control Flow Statements (if, while, for, etc)
7. Advanced Java Operators (Bitshift, Bitwise, etc)
8. Other relevant Java Constructs

Metrics 1, 3, 4, 6 and 7 can be used to compare a given solution of a problem to an optimal solution, and that way know if a solution was that of a programmer with more or less expertise in Java. Metrics 2, 3 and 5 can be use to find concerns with code understanding and readability.

4.2 PMD-Analyser

As mentioned before, PMD is a source-code analyser that looks for poor practices usually adopted by beginner programmers (e.g. several returns in one method or leaving empty catch-blocks).

To illustrate PMD behaviour, listing 2 below shows an example of a violation detected and reported by the tool.

Each violation found contains a good amount of information about the violation itself and where it was found. A large-sized project, with a lot of rulesets being examined, could return hundreds of violations which may prove very helpful in the profiling of programmers.

5 Correlating metrics with profiles

Correlating metrics with profiles has proved to be a challenging task. After much consideration, we came up with a proposal, presented below, that we think to be as accurate as possible.

To classify the abilities of a programmer regarding his knowledge about a language and the way he uses it, we considered two profiling perspectives, or group of characteristics: language skill and language readability.

- *Skill* is defined as the language knowledge and the ability to apply that knowledge in a efficient manner.
- *Readability* is defined as the aesthetics, readability and general concern with understandability of code.

Table 1 Proposed correlation.

Profile	*Skill*	*Readability*
Novice	−	−
Advanced Beginner	−	+
Expert	+	−
Proficient	+	+

Of the metrics extracted, some show a tendency towards classifying Skill while others towards classifying Readability. Here's a breakdown of where each metric may fall:

Skill

- Number of statements
- Control flow statements (If, While, For, etc)
- Advanced Java Operators
- Number and datatypes used
- Some PMD Violations (e.g. Optimization, Design and Controversial rulesets)

Readability

- Number of methods, classes and files
- Total number and ratio of code, comments and empty lines
- Some PMD Violations (e.g. Basic, Code Size and Braces rulesets)

These two groups contain enough information to obtain a profile of a programmer, regarding a given task. Then, for each group, and according to the score obtained by the programmer, Table 1 gives a general idea of how programmers can be profiled. (+) means a positive score, while (−) means a negative one.

What constitutes a lower and a higher score on each group must be defined. For every programmer, the goal is to compare each metric's value to the *standard solution*, which is by default considered a proficient solution (high skill and readability), and then, assemble a mathematical formula which allows to combine the metrics' results into a grade for each one of the two groups. With those two grades and resorting to Table 1 we can easily infer the programmer's profile in regards to the subject problem.

The exercise: "Read a given number of integers to an array, count how many are even", as proposed to two programmers. A Java and OOP teacher and an advanced Java programmer (master student). Listings 3 and 4 show both solutions.

After running both solution through the PP-Analyser we get the results shown in Table 2.

In Table 3 we compare the metrics of the obtained solution with the ones of the standard solution. In this table, for each metric analysed, the programmer gets 1, 0 or -1 whether his value on that metric is better, the same level or worst when compared to the standard solution, respectively. In this case the programmer got +3 points in skill -6 in readability when comparing to a proficient solution, making him an expert according to Table 1. As a general rule of thumb, for the readability group, more is better. Of course the score was obtained in a very naive way. As mentioned previously a mathematical formula which takes into consideration the importance of the metrics is expected to be developed to make this classification as precise a possible.

Another problem that is yet to be tackled is how to automatically compare some complex metrics like control flow statements and variable declarations, but we already know that it will be important to classify *CFS* and the datatypes as common or not so common in order to evaluate the programming language knowledge level.

Listing 3 Teacher Solution.

```
package ex1_arrays;

import java.util.Scanner;

/**
 * Escreva um algoritmo que leia e mostre um vetor de n elementos
 * inteiros e mostre quantos valores pares existem no vector.
 *
 * @author Paula
 */
public class Ex1_Arrays {

    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);

        int cont = 0, N;

        N = in.nextInt();

        int vec[] = new int[N];

        for (int i = 0; i < N; i++) {
            vec[i] = in.nextInt();
        }

        for (int i = 0; i < N; i++) {
            if (vec[i] % 2 == 0) {
                cont = cont + 1;
            }
        }

        System.out.println(cont);
    }

}
```

The goal for the *Programmer Profiler*, and especially the *Profile Inference Engine* is to be able to automatically make that classification and that way infer the profile of the programmer.

The (alpha version) PP-Analyser has already been applied on source-code developed by programmers on different levels of Java proficiency to start acquiring the values (metrics) that characterise the profiles. The code analysed was of moderate diversity, ranging in size and programming background (teachers, students and professional programmers).

6 Conclusion

Along this article, it was presented a proposal to develop a system (called Programmer Profiler) that allows to profile a programmer through the analysis of his source code. The hypothesis is that such profile inference is possible.

Listing 4 Advanced Programmer Solution.

```
import java.util.Scanner;
public class Even {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        int n = in.nextInt();
        int[] numbers = new int[n];
        int result = 0;
        for (int i = 0; i < n; i++) {
            int input = in.nextInt();
            numbers[i] = input;
            result += (input & 1) == 0 ? 1 : 0;
        }

        System.out.println(result);
    }
}
```

Table 2 PP-Analysis of two solutions.

Metric	*Teacher*	*Expert*
Total Number Of Files	1	1
Number Of Classes	1	1
Number Of Methods	1	1
Number Of Statements	6	3
Lines of Code	17 (48,6%)	12 (75%)
Lines of Comment	3 (8.3%)	0
Empty Lines	10 (28.6%)	1 (6.3%)
Total Number Of Lines	35	16
Control Flow Statements	{FOR=2, IF=1}	{FOR=1, IIF=1}
Not So Common CFSs	0	1
Variety of CFSs	2	2
Number of CFSs	3	2
Not So Common Operators	{}	{BIT_AND=1}
Number of NSCOs	0	1
Variable Declarations	{Scanner=1, int[]=1, int=4}	{Scanner=1, int[]=1, int=4}
Number Of Declarations	6	6
Number Of Types	3	3
Relevant Expressions	{SYSOUT=1}	{SYSOUT=1}

Until now, the main contributions of this work consist in: defining a set of possible profiles and their main characteristics; constructing the architecture of the system and the used tools; and performing experiments that allowed us to manually profile a programmer.

Currently, there is a working implementation that can be used to visualize [8] extracted metrics, both by the implemented PP-Analyser and the PMD Tool. That generated data is also being properly stored.

[8] http://www4.di.uminho.pt/~gepl/PP/

Table 3 Comparing to standard solution.

Metric Name	*Skill*	*Readability*
Number Of Files	X	0
Number Of Classes	X	0
Number Of Methods	X	0
Number Of Statements	+1	X
Number Of Lines of Code	X	−1
% Code	X	−1
Number Of Lines of Comment	X	−1
% Comment	X	−1
Number Of Empty Lines	X	−1
% Empty	X	0
Total Number Of Lines	X	−1
Control Flow Statements	+1	X
Variable Declaration	0	X
Total Number Of Declarations	0	X
Total Number Of Types	0	X
Advanced Operators	+1	X
PMD	N/A	N/A
Total	*+3*	*−6*

a) (+1) – better than the standard solution
b) (0) – same level as the standard solution
c) (−1) – worst than the standard solution
d) (X) – metric no related to this group
e) PMD results were not considered in this example

Some manual assessments are already being made with the objective of finding patterns and correlations that will make the PP a fully automatic tool.

We intend to go on conducting more and more experimental case studies to extract as much data as possible to refine the conclusions so far attained to improve our inference process aiming at finding a set of rules to automatically profile programmers.

References

1 Thomas Flowers, Curtis Carver, James Jackson, et al. Empowering students and building confidence in novice programmers through gauntlet. In *Frontiers in Education, 2004. FIE 2004. 34th Annual*, pages T3H–10. IEEE, 2004.

2 Quinn Hanam, Lin Tan, Reid Holmes, and Patrick Lam. Finding patterns in static analysis alerts: improving actionable alert ranking. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, pages 152–161. ACM, 2014.

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 James Jackson, Michael Cobb, and Curtis Carver. Identifying top java errors for novice programmers. In *Frontiers in Education, 2005. FIE'05. Proceedings 35th Annual Conference*, pages T4C–T4C. IEEE, 2005.

5 Huzefa Kagdi, Michael L Collard, and Jonathan I Maletic. A survey and taxonomy of approaches for mining software repositories in the context of software evolution. *Journal of Software Maintenance and Evolution: Research and Practice*, 19(2):77–131, 2007.

6 Emília Pietriková and Sergej Chodarev. Profile-driven source code exploration. *Computer Science and Information Systems (FedCSIS), pp. 929-934, IEEE.*, 2015.

7 Raphael 'kena' Poss. How good are you at programming? – a CEFR-like approach to measure programming proficiency, July 2014. URL: `http://science.raphael.poss.name/programming-levels.html`.

8 Alistair Sutcliffe. *Human-computer interface design*. Springer, 2013.

9 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.

Context-Free Grammars: Exercise Generation and Probabilistic Assessment

José João Almeida[*1], Eliana Grande[†2], and Georgi Smirnov[3]

1 **Departamento de Informática, Universidade do Minho, Braga, Portugal**
`jj@di.uminho.pt`
2 **Departamento de Informática, Universidade do Minho, Braga, Portugal**
`eliana.tiba@ifgoiano.edu.br`
3 **Departamento de Matemática, Universidade do Minho, Braga, Portugal**
`smirnov@math.uminho.pt`

Abstract

In this paper we present a metagrammar based algorithm for exercise generation in the domain of context-free grammars. We also propose a probabilistic assessment algorithm based on a new identity theorem for formal series, a matrix version of the well-known identity theorem from the theory of analytic functions.

1998 ACM Subject Classification F.4.2 Grammars and Other Rewriting Systems, F.4.3 Formal Languages, Classes defined by grammars or automata, F.2.1 Numerical Algorithms and Problems

Keywords and phrases Exercise generation, context-free grammars, assessment

Digital Object Identifier 10.4230/OASIcs.SLATE.2016.10

1 Introduction

This paper deals with (1) exercise generation and (2) assessment in the theory of context-free grammars. We describe a process of exercise generation based on metagrammars and an algorithm of assessment allowing one to decide, with high probability, if two context-free grammars are equivalent or not. It is well-known that the equivalence of two context-free grammars is an undecidable problem [9]. This is true if we consider the problem as an algebraic one. In this paper we show that the context-free grammars equivalence problem admits a solution if considered as a problem of analysis. Such a situation is not new. For example the fundamental theorem of algebra [7] has no algebraic proves but admits a simple proof using methods of analysis. The main idea of the assessment algorithm is the following. We associate to any context-free grammar a system of nonlinear equations. The system admits a solution in the form of formal series [10]. We substitute the symbols of terminal alphabet by 2×2-matrices and numerically solve the system of nonlinear matrix equations. This amounts to calculating of the value of the corresponding matrix series. We prove a new identity theorem. This theorem claims that if the sums of two formal power series coincide for all possible substitutions of 2×2-matrices, then the series are identical. From the practical point of view this implies that it suffices to check the equality between the sums of two series for a sufficiently big number of different substitutions, in order to decide if the grammars are equivalent or not. This leads us to a probabilistic assessment algorithm.

* The work of J.João Almeida was partially supported by COMPETE: POCI-01-0145-FEDER-007043 and FCT – Fundação para a Ciência e Tecnologia within the Project Scope: UID/CEC/00319/2013.
† The work of Eliana Grande was supported by PIQ IF Goiano through the fellowship no. 02/2013.

© José João Almeida, Eliana Grande, and Georgi Smirnov;
licensed under Creative Commons License CC-BY

This algorithm allows one not only to distinguish between two different languages but also to distinguish between grammars with different ambiguity.

2 Exercise generation and assessment

The exercise generation in the field of formal grammars can be based on several methods:

- Completely hand written,
- Example based methods (sets of positive and negative examples)
- Automata or regular expression-based generation,
- Grammar based generation.

As a result one pretends to obtain the statement of the problem, its solution and, if possible, an assessment method.

In this paper we consider a method based on a generative grammars.

2.1 Metagrammars for exercise generation

We define an attribute grammar which generates an exercise statement and the respective solution (a grammar). We refer to this attribute grammar as *metagrammar* [12, 11, 6]. The right-hand side (rhs) of any production-rule of the metagrammar is composed of two parts:

- a traditional grammar rhs to describe a grammar;
- a template to build an exercise statement, using subparts returned by the elements of the first part (attributes).

Schematically, an exercise generating metagrammar production has the following form:

$$\boxed{\text{NonTerminal}} \rightarrow (\text{"}GrammarComponent\text{"} \mid \boxed{\text{NonTerminal}})^* \quad \{TemplateWithAttributes\} \tag{1}$$

2.2 Example

Consider the following metagrammar example:

$$\begin{array}{lcl}
\boxed{\text{Mg}} & \rightarrow & \text{"}S \rightarrow \text{"} \boxed{\text{B}} \text{"} \mid A;\ A \rightarrow \text{"} \boxed{\text{C}} \text{"} \mid Int;\text{"} \\
 & & \{Give\ a\ context-free\ unambiguous\ grammar\ for\ the\ language\ of\ the \\
 & & arithmetic\ expressions\ including\ integers,\ \$2\ operator\ and\ \$4\ .\} \\
\boxed{\text{B}} & \rightarrow & \text{"}S - A\text{"}\{infix\ subtraction\} \\
 & \mid & \text{"}S + A\text{"}\{infix\ addition\} \\
 & \mid & \text{"}S * A\text{"}\{infix\ multiplication\} \\
 & \mid & \text{"}S/A\text{"}\{infix\ division\} \\
\boxed{\text{C}} & \rightarrow & \text{"}(S)\text{"}\{round\ brackets\} \\
 & \mid & \text{"}f(S)\text{"}\{prefix\ unary\ functions\}
\end{array} \tag{2}$$

In the templates \$2 and \$4 stand for attribute expansions following the Yacc[1], Bash, Perl syntax.

An example of generated exercise is:

▶ **Example 1.** Give a context-free unambiguous grammar for the language of the arithmetic expressions including integers, infix subtraction operator and round brackets.

The generated correct grammar for this exercise is the following:

$$\begin{array}{rcl} S & \to & S - A \\ & | & A \\ A & \to & (S) \\ & | & Int \end{array} \tag{3}$$

2.3 Assessment

The comparison of languages may take different flavor depending on the situation:

- If we are dealing with regular languages, we can use minimal automata comparison [3, 2];
- It is possible to use a set of tests (positive and negative sentences) to verify if two grammars generate the same language;
- In some situations it is possible to directly compare the grammars.

But these approaches are only applicable to special cases or do not completely solve the problem. Below we propose a new assessment method. It consists of the following steps:

1. The transformation of a context-free grammar into a system of formal nonlinear equations [9, 10];
2. The substitution of the terminal alphabet letters by randomly generated 2×2-matrices;
3. Creation of a numerical solution of the system of nonlinear matrix equations;
4. Iterating the previous steps k times.

The identity theorem proved in the next section says that if the sums of two formal power series coincide for all possible substitutions of 2×2-matrices, then the series are identical. Motivated by this result we use a probabilistic approach to compare two grammars. Namely, if the solutions of the respective systems of matrix equations coincide for k successive random matrix substitutions, we conclude that the grammars generate the same language.

Note that the use of 2×2-matrices is a key point of the approach. The multiplication of matrices is not commutative, and this fact allows one to distinguish between two words composed of the same terminal symbols collocated in different order. Obviously the use of numbers (instead of matrices) would not solve this problem.

3 Context-free grammars, formal power series, and nonlinear matrix equations

Any language can be defined in terms of a formal powers series in associative but noncommutative variables [9, 10]. Let V_T be a terminal alphabet, $W(V_T)$ be the set of words over V_T, and Z_+ be the set of nonnegative integers. A map $\phi : W(V_T) \to Z_+$ defines a formal power series

$$s = \sum_{P \in W(V_T)} \phi(P)P \tag{4}$$

with nonnegative integer coefficients. Let μ be a map from V_T to the set $R^{2\times 2}$ of 2×2-matrices. By $\mu(P)$ we will denote the matrix obtained substituting the letters $a_i \in P$ by the matrices $\mu(a_i)$ and calculating the respective matrix product. If the series

$$s(\mu) = \sum_{P \in W(V_T)} \phi(P)\mu(P) \tag{5}$$

converges, its sum is a 2×2-matrix. (If the series is divergent, we set $s(\mu) = \infty$.) The assessment of exercises is base on the following *identity* theorem

▶ **Theorem 2.** *Let s_1 and s_2 be two formal power series. If $s_1(\mu) = s_2(\mu)$ for all $\mu : V_T \to R^{2\times 2}$, then $s_1 = s_2$.*

Proof. It suffices to consider the class of matrices

$$\mu = \begin{pmatrix} u & v \\ 0 & 1 \end{pmatrix}.$$

By induction it is easy to prove the equalities

$$\prod_{i=1}^{n} \mu_i = \prod_{i=1}^{n} \begin{pmatrix} u_i & v_i \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} P & Q \\ 0 & 1 \end{pmatrix}, \tag{6}$$

where

$$P = \prod_{i=1}^{n} u_i \text{ and } Q = \sum_{m=2}^{n} \left(\prod_{i=1}^{m-1} u_i \right) v_m + v_1. \tag{7}$$

(It is allowed $u_i = u_j$, and $v_i = v_j$.) The polynomial P allows one to identify the letters appearing in the word $a_1 a_2 \dots a_n$, while the polynomial Q makes it possible to uniquely reconstruct the order of the letters. ◀

▶ Remark. It suffices to check the equality of the series sums for matrices with a sufficiently small norm. This will guarantee the convergence of the series.

It is well-known [9, 10] that for any context-free grammar there exists a system of nonlinear equations allowing one to obtain, by an iterative procedure, the formal power series whose terms are the words of the respective language. This correspondence between the series and systems of equations makes it possible to effectively calculate the sums of the series for all possible substitutions of 2×2-matrices.

Let X_i, $i = \overline{0, m}$, be the nonterminals of a context-free grammar and let P_j^i, $i = \overline{0, m}$, $j = \overline{1, l_i}$, be the words appearing on the right-hand sides of the production with the left-hand side X_i. Then the system of formal equations corresponding to the grammar reads

$$\begin{aligned} X_1 &= P_1^1 + \ldots + P_{l_1}^1, \\ &\vdots \\ X_m &= P_1^m + \ldots + P_{l_m}^m. \end{aligned} \tag{8}$$

Substituting the letters of the terminal alphabet, $a_i \in P_j^i$, by matrices $\mu(a_i)$, we get a system of nonlinear (matrix) equations. This system, $X = F(X)$, can be solved using an iterative procedure: $X^{k+1} = F(X^k)$, $X^0 = 0$, or using the Newton method. The latter is much more faster. Note that, in view of the Identity Theorem, it suffices to consider only matrices of the form

$$\mu = \eta \begin{pmatrix} u & v \\ 0 & 1 \end{pmatrix}$$

with $u, v \in [0, 1]$ and sufficiently small η. The convergence of the iterations can be guaranteed for grammars in Chomsky and Greibach normal forms. If we deal with regular languages, then system (8) is linear.

4 Example

Let us go back to the example considered in Sec. 2. Assume that we generate the following exercise:

▶ **Example 3.** Give a context-free unambiguous grammar for the language of the arithmetic expressions including integers, infix subtraction operator and round brackets.

The generated correct grammar for this problem is:

$$\begin{array}{rcl} S & \to & S - A \\ & | & A \\ A & \to & (S) \\ & | & Int \end{array} \tag{9}$$

To avoid confusion in notation we set $a = -$, $b = Int$, $c = ($, and $d =)$. Thus the right solution is given by

$$\begin{array}{rcl} S & \to & SaA \\ & | & A \\ A & \to & cSd \\ & | & b \end{array} \tag{10}$$

The corresponding system of matrix equations reads [9, 10]

$$\begin{array}{l} S = SaA + A, \\ A = cSd + b. \end{array} \tag{11}$$

Let us consider other three possible answers.

Alternative correct solution. The following grammar is different but generates the same language:

$$\begin{array}{rcl} S & \to & AaS \\ & | & A \\ A & \to & cSd \\ & | & b \end{array} \tag{12}$$

and the corresponding system of matrix equations is

$$\begin{array}{l} S = AaS + A, \\ A = cSd + b. \end{array} \tag{13}$$

Wrong solution. The following grammar does not generate the same language (does not generate the sentence $cbabd$):

$$\begin{array}{rcl} S & \to & SaA \\ & | & A \\ A & \to & cAd \\ & | & b \end{array} \tag{14}$$

The corresponding system of matrix equations is

$$\begin{array}{l} S = SaA + A, \\ A = cAd + b. \end{array} \tag{15}$$

Ambiguous solution. The following grammar generates the same set of sentences but is ambiguous (the sentence *babab* possesses more then one derivation):

$$\begin{array}{rcl} S & \to & SaS \\ & | & A \\ A & \to & cSd \\ & | & b \end{array} \tag{16}$$

and the corresponding system of matrix equations is

$$\begin{aligned} S &= SaS + A, \\ A &= cSd + b. \end{aligned} \tag{17}$$

Generating random matrices a, b, c, and d with random elements uniformly distributed in the interval $[0, 0.1]$ and solving systems (11), (13), (15), and (17) (the iteration process starts at $S = A = 0$) we see that the difference between S components of solutions for systems (11) and (13) is zero, while for the pair (11) and (15) or (11) and (17) is of the order $10^{-4} \div 10^{-5}$. This allows one to clearly distinguish between the correct solution and the wrong one.

The interval where the elements of the random matrices are generated must be (a) sufficiently small in order to guarantee the convergence of iterations, (b) big enough to distinguish between two different languages.

5 Probabilistic assessment

It is clear that to distinguish between the correct solution and a wrong solution it suffices to find a matrix substitution $\mu(a_i)$, $a_i \in V_T$, giving different solutions to systems (8) corresponding to two different grammars. But how many substitutions are needed to conclude that two (unambiguous) grammars generate the same language? The answer to this question can be given in a probabilistic form. Probabilistic approach is not new in program testing [4] or assessment of math exercises [8] and showed good results.

Consider the following thought statistical experiment. We randomly generate a pair of grammars with the same terminal alphabet (see Sec. 2) and solve, for an infinite sequence of matrix substitutions $\mu(a_i)$, $a_i \in V_T$, the corresponding systems (8) of matrix equations. The number of substitutions giving the same result for two systems of equations is a random variable ξ. It is natural to assume that ξ is distributed according with the Poisson law:

$$P\{\xi = k\} = \frac{\lambda^k e^{-\lambda}}{k!},$$

where λ is the average number of events. Our simulations show that $\lambda \ll 1$. Thus the probability to have k times equal results for two different languages (recall that we consider languages with different ambiguity as different languages) is

$$P\{\xi = k\} = \frac{\lambda^k e^{-\lambda}}{k!} < \frac{1}{k!}.$$

From the practical point of view this implies that if we have the same result for two systems of equations in, for example, $k = 10$ successive random matrix substitutions $\mu(a_i)$, $a_i \in V_T$, then the grammars generate the same language.

6 Application to real size grammars

In the context of exercise generation, the grammars are rather small, because the aim is to deal with one thing at a time. Typical examples of such grammars have half a dozen of nonterminal symbols, and a dozen of productions. In this situation we had no problems with the presented approach.

Comparing two real size grammars, we may fall in situations of non-convergence, or in situations where it is impossible to distinguish between two different grammars due to insufficient computer accuracy. In order to test our approach for real size grammars we used a C-like grammar [5] (44 nonterminals, 104 production rules). We compared this grammar with an equivalent one and with a slightly modified grammar generating a different language. Generating random matrices with components in the interval $[0, 0.01]$, we could correctly solve the respective systems of equations and to establish the coincidence of the languages generated by the equivalent grammars and to distinguish between the correct and the wrong grammars.

7 Conclusion and future research

In this paper we demonstrated that numerical methods can be an effective tool to compare context-free grammars.

We presented:

1. An algorithm for exercise generation in the domain of formal languages, namely context-free grammars. The algorithm makes use of metagrammars;
2. A probabilistic assessment algorithm allowing one to decide, with high probability, if two context-free grammars are equivalent or not.

The algorithms allow one to automatically generate and assess exercises involving context-free grammars.

In the future research we plan to address the following issues:

1. Determination of the range of random matrices guaranteeing the convergence of numerical methods used to solve the systems of nonlinear matrix equations;
2. A profound study of statistical properties of the random variable ξ defined in section 5.

Acknowledgments. The authors would like to thank the reviewers for their comments, insights and corrections.

References

1 Alfred V. Aho, Ravi Sethi, and Jeffrey D. Ullman. *Compilers: Principles, Techniques, and Tools*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1986.

2 Marco Almeida, Nelma Moreira, and Rogerio Reis. Testing equivalence of regular languages. *Journal of Automata, Languages and Combinatorics*, 15(1/2), 2010.

3 Marco Almeida, Nelma Moreira, and Rogerio Reis. Finite automata minimization algorithms. In Jiacun Wang, editor, *Handbook of Finite State Based Models and Applications, Discrete Mathematics and Its Applications*, pages pp.145–170. Chapman and Hall/CRC Press, 2012.

4 Richard Demillo and Richard Lipton. A probabilistic remark on algebraic program testing. *Information Processing Letters*, 7(4):pp. 193–195, 1978.

5 Robert Heckendorn. A grammar for the C- programming language. Technical Report Version S16, Department of Computer Science – University of Idaho, 2016. URL: http://marvin.cs.uidaho.edu/Teaching/CS445/c-Grammar.pdf.

6 Daan Leijen and Erik Meijer. Parsec: Direct style monadic parser combinators for the real world. Technical report, Department of Computer Science, University of Utrecht, 2001. URL: http://research.microsoft.com/pubs/65201/parsec-paper-letter.pdf.

7 Jerrold E. Marsden and Michael J. Hoffman. *Basic Complex Analysis*. W. H. Freeman, third edition, 1999.

8 Minh Luan Nguyen, Siu Cheung Hui, and Alvis C. M. Fong. Probabilistic equivalence verification approach for automatic mathematical solution assessment. In *23rd International Joint Conference on Artificial Intelligence (IJCAI2013)*, pages pp. 1352–1356, 2013.

9 Arto Salomaa. *Formal Languages*. Academic Press, 1973.

10 Arto Salomaa and Matti Soittola. *Automata-theoretic aspects of formal power series*. Springer, 1978.

11 S. Doaitse Swierstra. Combinator parsers: From toys to tools. *Electronic Notes in Theoretical Computer Science*, 41, 2001. doi:10.1016/S1571-0661(05)80545-6.

12 Harald Heinz Vogt, S. Doaitse Swierstra, and Matthijs F. Kuiper. Higher order attribute grammars. *SIGPLAN Not.*, 24(7):131–145, June 1989. doi:10.1145/74818.74830.

A Model-Driven Engineering Technique for Developing Composite Content Applications*

Moharram Challenger[1], Ferhat Erata[2], Mehmet Onat[3], Hale Gezgen[4], and Geylani Kardas[5]

1 **R&D Department, UNIT IT R&D Ltd., Izmir, Turkey, and International Computer Institute, Ege University, Izmir, Turkey**
moharram.challenger@unitbilisim.com
2 **R&D Department, UNIT IT R&D Ltd., Izmir, Turkey, and International Computer Institute, Ege University, Izmir, Turkey**
ferhat.erata@unitbilisim.com
3 **R&D Center, Koçsistem Information and Communication Services Inc., Üsküdar/Istanbul, Turkey**
mehmet.onat@kocsistem.com.tr
4 **R&D Center, Koçsistem Information and Communication Services Inc., Üsküdar/Istanbul, Turkey**
hale.gezgen@kocsistem.com.tr
5 **International Computer Institute, Ege University, Izmir, Turkey; and R&D Center, Koçsistem Information and Communication Services Inc., Üsküdar/Istanbul, Turkey**
geylani.kardas@ege.edu.tr

Abstract

Composite Content Applications (CCA) are cross-functional process solutions built on top of Enterprise Content Management systems assembled from pre-built components. Considering the complexity of CCAs, their analysis and development need higher level of abstraction. Model-driven engineering techniques covering the use of Domain-specific Modeling Languages (DSMLs), can provide the abstraction in question by moving software development from code to models which may increase productivity and reduce development costs. Hence, in this paper, we present MDD4CCA, a DSML for developing CCAs. The DSML presents an abstract syntax, a concrete syntax, and an operational semantics, including model-to-model and model-to-code transformations for CCA implementations. Use of the proposed language is evaluated within an industrial case study.

1998 ACM Subject Classification D.1.7 Visual Programming, D.2.6 Programming Environments, Graphical environments, D.2.11 Software Architectures, Domain-specific Architectures

Keywords and phrases Domain-specific modelling languages, composite content applications, model transformation, code generation

Digital Object Identifier 10.4230/OASIcs.SLATE.2016.11

1 Introduction

An enterprise content management system (ECM) organizes documents, contacts and records related to the processes of a commercial organization [3]. ECM aims to make the management

* This work is financially supported by the Scientific and Technological Research Council of Turkey (TUBITAK) Technology and Innovation Funding Programs Directorate (TEYDEB) under grant no. 3110712.

© Moharram Challenger, Ferhat Erata, Mehmet Onat, Hale Gezgen, and Geylani Kardas;
licensed under Creative Commons License CC-BY

of corporate information easier through simplifying storage, security, version control, process routing, and retention. However, considering all capabilities and components of ECM, its architecture is rather huge and complex. Composite Content Applications[1] (CCA) are cross-functional process solutions (built on top of an ECM system) assembled from pre-built components, e.g. an integration of a Customer Relation Management (CRM), Forms, ECM, and Business Process Modelling (BPM). This integration adds more structural complexity to the system architecture.

Considering this complexity, CCA analysis and development needs new domain engineering and software development techniques. One possible approach to cope with this complexity is to increase the abstraction level using models [6] [8], in other words applying Model Driven Engineering (MDE), which moves software development from code to models [11] and may increase productivity [5] and reduce development costs [13]. One of the approaches to realize MDE is developing a Domain-specific Modelling Language (DSML) [7]. A DSML allows end-user programmers (domain experts) to describe the essence of a problem with abstractions related to a domain specific problem space.

In this paper, we present MDD4CCA, a domain specific modelling language for composite content applications. The DSML covers abstract syntax, concrete syntax, and operational semantics (including model-to-model and model-to-code transformations). In this study, the target platform is Microsoft Sharepoint. Using MDD4CCA tool, a user can model the system from various aspects, such as Form, Navigation, Content, and Workflow. As result the functional architectural code will be generated by the tools which is fully functional in target platform. Furthermore, the proposed language is evaluated using a real industrial use case and the results are reported in this paper.

There are some frameworks in industry to address the difficulty of CCA development, such as Nintex[2], AgilePoint[3], and K2[4] Frameworks. Nintex is a workflow software to automate business processes. AgilePoint enables business users to manage their processes. K2 is a platform for creating business applications and provides forms and workflows. However, there is no language behind any of these frameworks. So, they can only convert the model to code in a idiosyncratic way and they offer no semantics check and constraint control possibilities. A DSML, not only provides the concrete syntax for elements in the language, but also it presents abstract syntax including meta elements and their relations. Moreover, DSML allows providing controls both in modelling and interpretation time. Having these capabilities was our main motivations for developing MDD4CCA. In addition, the above-mentioned frameworks mainly stress on Forms and Workflow views. However, in our approach MDD4CCA uses various views in the form of different packages namely, Content, Navigation, Workflow, Form, and User packages.

Rest of the paper is organized as follows: In Section 2, the methodology applied in developing MDD4CCA is presented. In the next section, development of MDD4CCA is elaborated. Section 4 presents the industrial use case in which MDD4CCA is used to develop the software. Finally, the paper is concluded in Section 5.

[1] http://www.gartner.com/it-glossary/composite-content-applications-ccas/
[2] http://www.nintex.com/
[3] http://agilepoint.com/
[4] http://www.k2.com/

2 Methodology

To perform this study, we fulfilled three main steps: Requirement analysis, development of the language and finally its application in a case study. The overall procedure of steps one and two are depicted in Figure 1 and the case study is discussed in Section 4. The DSML development procedure includes the problem space (PS) modelling, solution space (SS) modelling, and solution space code generation [10]. PS covers issues for the modelling the problem and related tools independent of the details of the underlying frameworks, while SS deals with the modelling and code generation for the target framework.

Considering the requirement engineering for MDD4CCA, initially a concept dictionary is provided including the terms of a CCA. We have used a feature model [9] as a formal method to represent the specifications. As a tool to realize this, we have used Clafer [1] with which it is possible to perform model checking. After checking the feature models, the feature model is transformed into a meta-model in EMF [12] automatically by transformation rules given in Xtend [2] and Java. The meta-model is divided into four viewpoints including: Content, Form, Navigation, and Workflow.

The provided PS meta-model in EMF is the main input by which problem space modelling and its tool are provided. We developed a graphical editor in GMF[5] with which an end-user can model a problem according to the related business domain. Also, we provided Form model with an endogenous model to model transformation which is required for a composite content model. Using the Form model, we generate entity data models using bidirectional transformations which provides round-trip functionality and any modification in each side can be converted to the other side automatically. In the solution space modelling, the entity data model is generated.

Finally, from entity data model, entity classes are generated by using entity data model tool. On the other hand, from solution space models, the codes for server side pages, and other required files such as XML files are generated by model to text transformations via Xpand[6], Xtend, and Java. With a similar transformation, the required library code is configured based on the solution space model.

3 Development of MDD4CCA

As depicted in Figure 1, MDD4CCA development started with the requirement engineering providing a concept dictionary including terms (from YAWL[7], UWE[8], BPMN[9], and so on) and their relations with other concepts. The dictionary includes 537 concepts which are used as CCA requirements. To present these specifications in a formal way, we mapped the requirements into feature model. Also, to cope with the complexity and size of the resulted feature model (considering large number of concepts and their inter-relations), the models are divided into several viewpoints. It is worth noting that the User view is integrated in Content model. We used Clafer to implement the feature model with which we can also use Alloy[10] to do model checking. In this way, a clean room software engineering is realized with preventing some defects in the requirement engineering level using Alloy model checking

[5] http://www.eclipse.org/gmf-tooling/
[6] http://wiki.eclipse.org/Xpand
[7] http://www.yawlfoundation.org/
[8] https://en.wikipedia.org/wiki/UML-based_Web_Engineering
[9] http://www.bpmn.org/
[10] http://alloy.mit.edu/alloy/

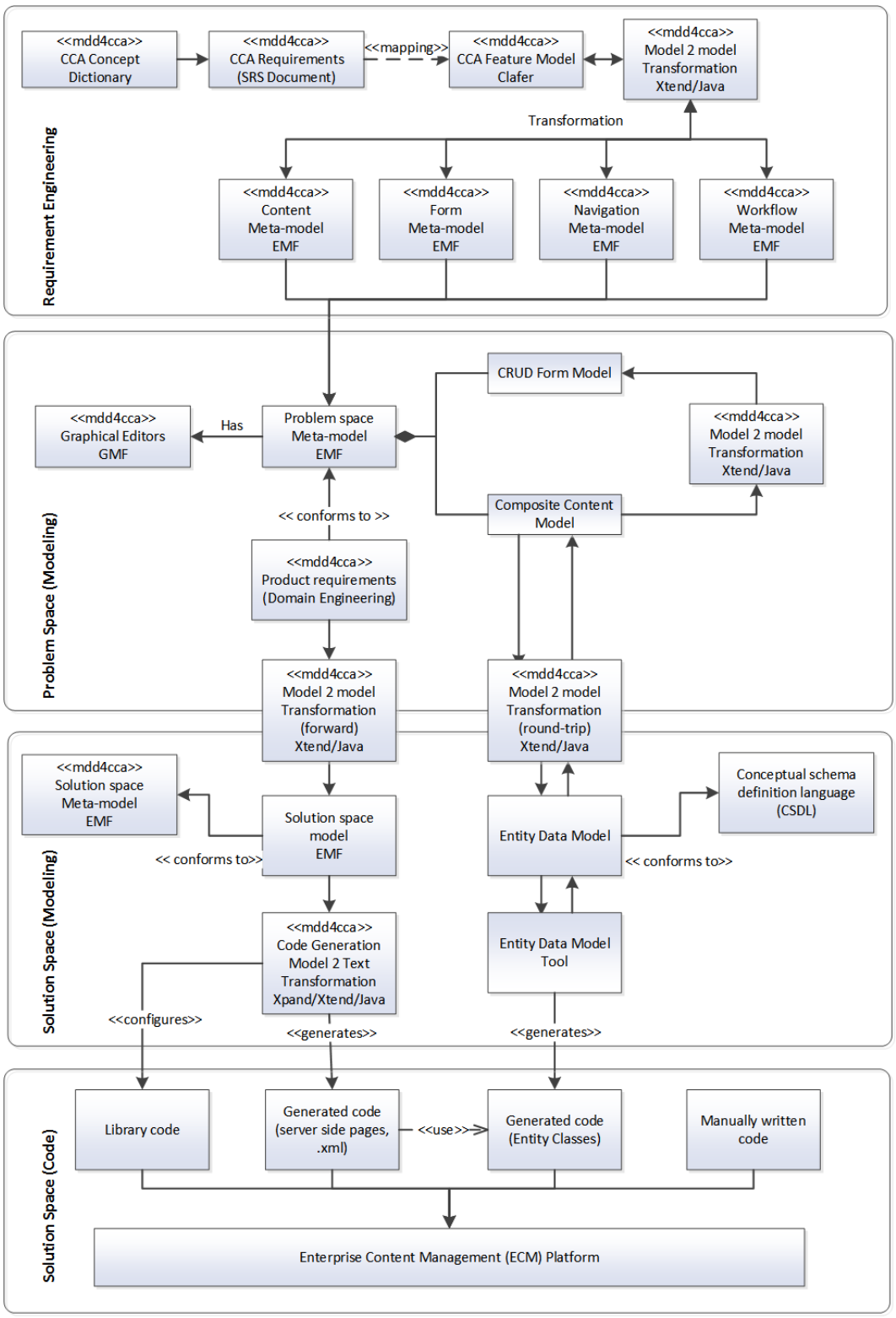


Figure 1 The process of analysing and developing MDD4CCA.

on feature model. Then, we transform feature models in Clafer format (XML-based) to meta-model [4].

3.1 Problem Space Modelling

Modelling of PS addresses the problem independent of details inside the artefacts of target framework. Based on the requirement analysis, the problem space modelling covers the domain with four viewpoints discussed in the previous section. Considering the space limitations of this paper, we focus on Content viewpoint which includes User viewpoint and also can generate main forms of Form viewpoint. Of course, an end-user's custom forms will be added to the Form viewpoint to complete the modelling.

To develop a DSML for modelling the PS, i.e. MDD4CCA, there is need for an abstract syntax. We generated the required meta-model automatically from the PS feature model. Part of this meta-model, related to Content Viewpoint, is depicted in Figure 2. This meta-model is used to provide the graphical editor for the Content viewpoint in MDD4CCA.

As can be seen in this figure, the features and the hierarchy structure in the feature model is transformed to the meta-elements and the relations between them in the meta-model. For example, a Web item can have several sub Web items and each of them can have Pages, Lists, SiteCollections and so on.

In addition to the abstract syntax, the graphical concrete syntax is provided for the MDD4CCA's graphical editor. To this end, we mapped the abstract syntax elements of MDD4CCA to the graphical notations. In this study, we have used Eclipse GMF to provide the graphical editor of the DSML. However, the complexity of configuring GMF forced us to use a higher level tool on top of GMF called Eclipse Epsilon[11] with which we could generate GMF components from an Ecore like language called EMF and a tool called Emfatic[12]. As result, we have provided a fully functional tool for MDD4CCA.

MDD4CCA's syntax tools can impose some restrictions/controls during the user's modelling. One part of these controls comes from the PS meta-model and the remaining originates from the graphical tool itself. These constraint controls help the end-user to design an accurate model by presenting mistakes such as wrong element connection, avoiding empty attribute, controlling number of required relations for a specific element, and so on. These constraints are implemented in Epsilon Validation Language (EVL)[13]. Part of the contraint control for Content Type is given in Listing 1.

3.2 Solution Space Modelling

There is a difference between modelling a CCA independent of the underlying frameworks (PS) and its modelling based on the platforms (SS). Therefore, we needed to separate level of modelling by providing two different meta-models for problem and solution spaces. To prepare a meta-model for SS, we analysed the commonality and variability. The result was a meta-model including elements from different related technologies, e.g. Entity Framework, Server Side pages and Forms, Cache layer, and object oriented language concepts.

Finally, a (instance) model of solution space meta-model is transformed to a platform specific code (or API) which is discussed in the next section.

[11] http://www.eclipse.org/epsilon/
[12] http://www.eclipse.org/epsilon/doc/eugenia/
[13] http://www.eclipse.org/epsilon/doc/evl/

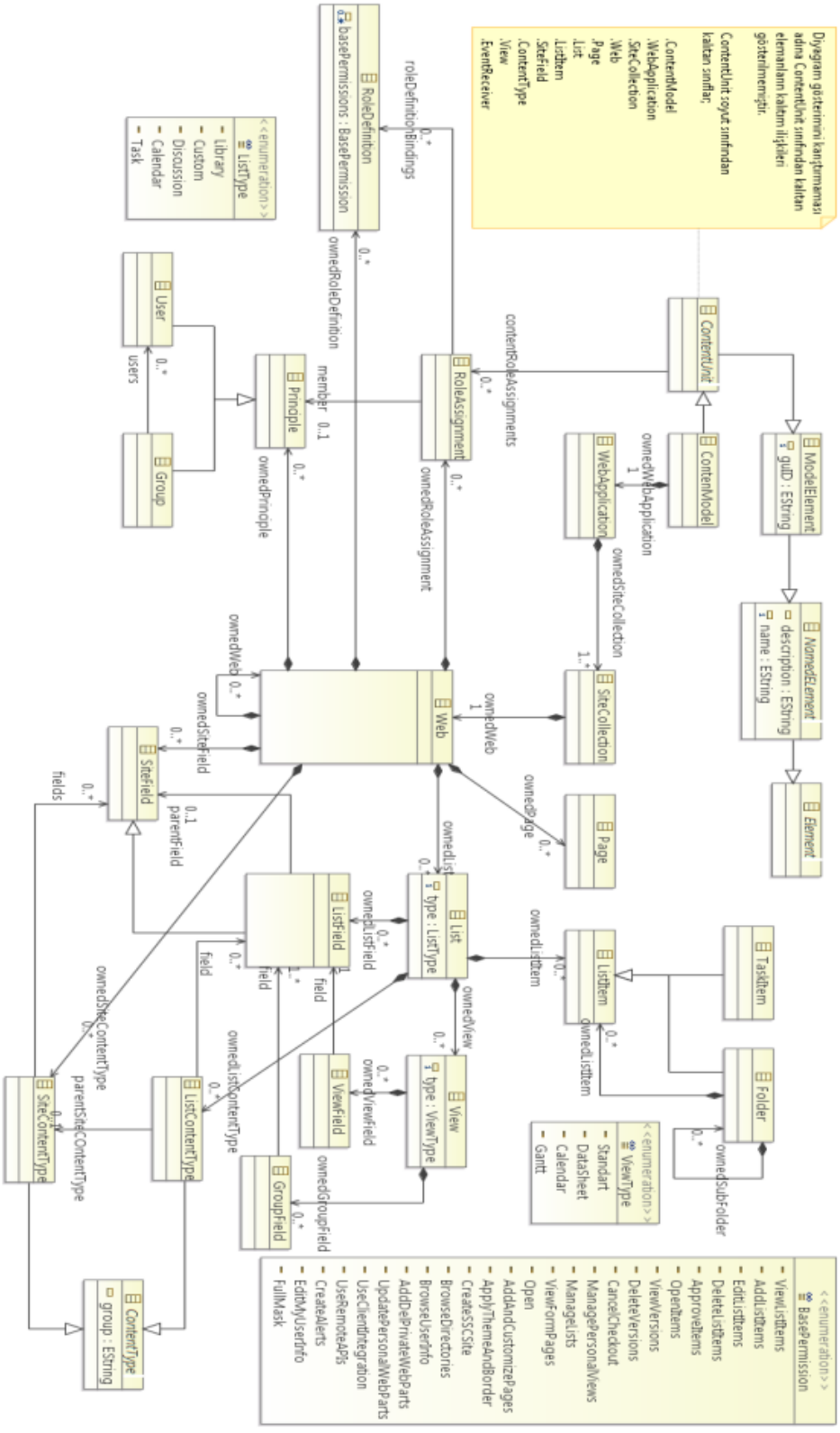


Figure 2 The Abstract Syntax of the Content Viewpoint.

Listing 1 Constraint control for Content Type element in EVL.

```
import ecore : 'http://www.eclipse.org/emf/2002/Ecore#/';
package modelgen : modelgen =
 'http://www.mdd4cca.com/msf/modelgen/Modelgen'{
 ...
 class File{
  attribute name : String[?];
  attribute extension : String[?] = '.cs';
  attribute folderPath : String[?];
 }
 class EnumMProperty extends MProperty{
  property enumType : EnumClass[?];
 }
 class MNavigationProperty{
  attribute name : String[?];
  property type : Cache[?];
  attribute multi : Boolean[?];
 }
 class MClass extends File{
  attribute usings : String[*] { ordered };
  attribute namespace : String[?];
  attribute constructorBody : String[*] { ordered };
 }
 ...
}
```

Table 1 Model Transformations in MDD4CCA.

Direction	*Source MM*	*Target MM*	*Type*	*# of Rules*
Forward	Problem Space	Problem Space	M2M	38
Forward	Problem Space	Solution Space	M2M	71
Forward/Backward	Problem Space	Microsoft CSDL[14]	M2M	24
Forward	Solution Space	code and xml files	M2C	204 Xpand templates

3.3 Model Transformations

It is not sufficient to complete a DSML definition by only specifying the notions and their representations. The complete definition requires that one provides semantics of language concepts in terms of other concepts whose meanings are already established. In this study, four types of transformations are fulfilled to realize the mentioned methodology, see Table 1. These transformations are Clafer to Ecore transformation (in requirement engineering phase), Content viewpoint to Form viewpoint transformation (in problem space modelling phase), problem space model to solution space model transformation, and finally solution space model to target platforms' code transformation. These transformations are used to respectively generate CRUD forms from content models; weave and translate content and form models into a solution space model; transform MDD4CCA content model (EDM) from/to EDMX (Entity Framework); and code generation from the SS models. The transformations are implemented using Java and Xpand in an integrated way.

4 Industrial Use Case: TUPRAS TPY Project

Taking into account the high cost of developing a DSML, there is a need to have the Return On Investment (ROI), balancing the expected benefits and productivity improvement against the cost of development and future maintenance of the tools before moving to a new development environment. Therefore, in this section we present one of the industrial projects which is implemented using MDD4CCA for one of the corporate customers of UNIT Company. The purpose of this section is to briefly report our experiences of developing a DSML based on the use case.

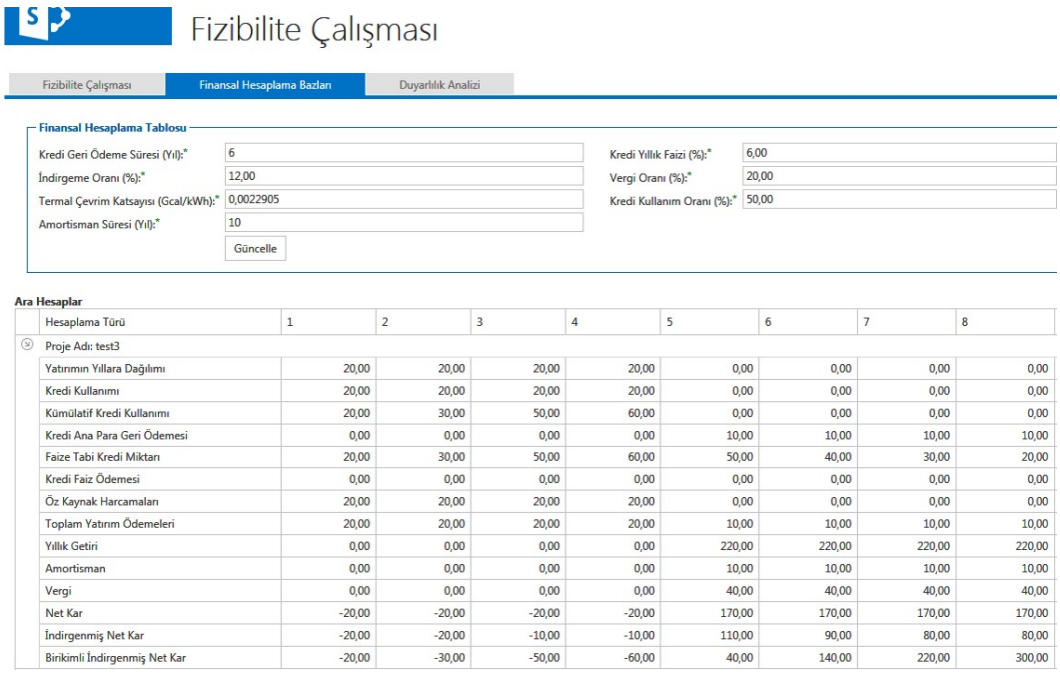
The industrial project discussed here is called TPY (acronym for the Turkish translation of "Tupras Project Management"). Turkish Petroleum Refineries Corporation (TUPRAS)[15] is currently the biggest enterprise in Turkey according to Turkey's Fortune 500 list[16]. TPY project is for managing the newly defined projects such as designing a new refinery in Tupras. The use case is modelled in MDD4CCA in 4 viewpoints (Content, Form, Fork-flow, and Navigation) with many diagrams. For example, there are several diagrams modelled for Form viewpoint considering the forms in different parts of the project, such as Activity form, Feasibility form, and so on.

After modelling the project in different viewpoints with diagrams for each part of the project using MDD4CCA, the transformations are applied using transformation engine of MDD4CCA and other models are generated from which the architectural code is generated. By adding delta code with developers, the system is fully functional. Altogether, this project has 79 work-flows, 195 transitions in work-flows, 43 roles, 137 tasks, 107 tables of databases, and 403 web pages for forms. The project has several important parts such as pre-feasibility, feasibility, pre-discovery, discovery, yearly investment planing, and Progress. For instance the screenshot of automatically generated form of the TPY's "Feasibility Page" is shown in Figure 3. Due to the confidentially issues, only the general parts are demonstrated in the figure. Besides, both the interface and the content of the form are in Turkish since Tupras aimed at using TPY only nation-wide at this stage. In the figure, feasibility study of a new project inside the refinery is considered. In the active tab given in the figure, some of the financial calculation values such as yearly interest for the credit, tax ratio, yearly distribution of the new project investments are given inside the form which is achieved by the execution of the MDD4CCA transformations.

In this case study, totally 96 model to model transformation rules and 151 model to code transformation templates are used. The model transformation from the problem space to the solution space is fully transformed automatically. In the scope of this case study, most of the code also is generated automatically. The resulting architectural code is functional in the target platform. Considering the underlying web-based content management platform, the language generated around 65% of the code. The other 35% of the code is in fact the part which has high variability among the projects.

The resulting software product has been used in TUPRAS since mid 2014 (about 2 years until the time this paper was prepared). The software is used by the staff from different TUPRAS departments. 120 workers from the project management departments of four TUPRAS refineries are using the software. In addition, 80 engineers are benefiting from the modeling environment of the given tool during project proposal preparation.

[15] http://www.tupras.com.tr/masterpage.en.php
[16] http://aa.com.tr/en/economy/tupras-tops-turkey-s-fortune-500-list/30989



Hesaplama Türü	1	2	3	4	5	6	7	8
Proje Adı: test3								
Yatırımın Yıllara Dağılımı	20,00	20,00	20,00	20,00	0,00	0,00	0,00	0,00
Kredi Kullanımı	20,00	20,00	20,00	20,00	0,00	0,00	0,00	0,00
Kümülatif Kredi Kullanımı	20,00	30,00	50,00	60,00	0,00	0,00	0,00	0,00
Kredi Ana Para Geri Ödemesi	0,00	0,00	0,00	0,00	10,00	10,00	10,00	10,00
Faize Tabi Kredi Miktarı	20,00	30,00	50,00	60,00	50,00	40,00	30,00	20,00
Kredi Faiz Ödemesi	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
Öz Kaynak Harcamaları	20,00	20,00	20,00	20,00	0,00	0,00	0,00	0,00
Toplam Yatırım Ödemeleri	20,00	20,00	20,00	20,00	10,00	10,00	10,00	10,00
Yıllık Getiri	0,00	0,00	0,00	0,00	220,00	220,00	220,00	220,00
Amortisman	0,00	0,00	0,00	0,00	10,00	10,00	10,00	10,00
Vergi	0,00	0,00	0,00	0,00	40,00	40,00	40,00	40,00
Net Kar	-20,00	-20,00	-20,00	-20,00	170,00	170,00	170,00	170,00
İndirgenmiş Net Kar	-20,00	-20,00	-10,00	-10,00	110,00	90,00	80,00	80,00
Birikimli İndirgenmiş Net Kar	-20,00	-30,00	-50,00	-60,00	40,00	140,00	220,00	300,00

Figure 3 The running application of TPY use case (Feasibility Page).

5 Conclusion and Future Work

In this paper, a domain specific modelling language, called MDD4CCA is developed for Composite Content Applications. The language covers all model driven components including the abstract syntax, the concrete syntax, model to model and model to code generation. The language is used in the development of an industrial project which is reported as a case study again in this paper. The result shows that utilization of the proposed model-driven technique and MDD4CCA, the 65% of the application code is generated automatically in average. Our next work is to continue the evaluation of the DSML by using it for the development of new real industrial CCAs.

References

1 Kacper Bąk, Zinovy Diskin, Michał Antkiewicz, Krzysztof Czarnecki, and Andrzej Wąsowski. Clafer: unifying class and feature modeling. *Software & Systems Modeling*, pages 1–35, 2014.

2 Lorenzo Bettini. *Implementing Domain-Specific Languages with Xtext and Xtend*. Packt Publishing, 2013.

3 Bob Boiko. *Content Management Bible*. John Wiley & Sons, 2005.

4 Ferhat Erata, Moharram Challenger, Serhat Gezgin, Argün Demirbaş, Mehmet Önat, and Geylani Kardas. A methodology for supporting the synchronization between capability models and metamodels in software product lines. In *8th Turkish National Software Engineering Symposium*, volume 1221, pages 2–13, 2014.

5 Tomaž Kos, Tomaž Kosar, Jure Knez, and Marjan Mernik. From DCOM interfaces to domain-specific modelling language: A case study. *Computer Science and Information Systems*, 8(2):361–378, 2011.

6 Ivan Lukovic, Vladimir Ivancevic, Milan Celikovic, and Slavica Aleksic. DSLs in action with model based approaches to information system development. In Marjan Mernik, editor, *Formal and Practical Aspects of Domain-Specific Languages: Recent Developments*, pages 502–532. IGI Global, 2013.

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

8 Aleksandar Popovic, Ivan Lukovic, Vladimir Dimitrieski, and Verislav Djukic. A DSL for modeling application-specific functionalities of business applications. *Computer Languages, Systems and Structures*, 43(C):69–95, October 2015. doi:10.1016/j.cl.2015.03.003.

9 Roger Pressman. *Software Engineering: A Practitioner's Approach.* McGraw Hill, 2000.

10 Awais Rashid, Jean-Claude Royer, and Andreas Rummler. *Aspect-Oriented, Model-Driven Software Product Lines - The AMPLE Way.* Cambridge publications, 2011.

11 Douglas C. Schmidt. Guest Editor's Introduction: Model-Driven Engineering. *Computer*, 39(2):25–31, February 2006.

12 Dave Steinberg, Frank Budinsky, Marcelo Paternostro, and Ed Merks. *EMF: eclipse modeling framework.* Pearson Education, 2008.

13 Antonio Vallecillo. A journey through the secret life of models. In Uwe Aßmann, Jean Bézivin, Richard Paige, Bernhard Rumpe, and Douglas C. Schmidt, editors, *Perspectives Workshop: Model Engineering of Complex Systems (MECS)*, number 08331 in Dagstuhl Seminar Proceedings, Dagstuhl, Germany, 2008. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany.

Eshu: An Extensible Web Editor for Diagrammatic Languages*

José Paulo Leal[1], Helder Correia[2], and José Carlos Paiva[3]

1 **CRACS & INESC-Porto LA, Faculty of Sciences, University of Porto, Porto, Portugal**
zp@dcc.fc.up.pt

2 **CRACS & INESC-Porto LA, Faculty of Sciences, University of Porto, Porto, Portugal**
up201108850@fc.up.pt

3 **CRACS & INESC-Porto LA, Faculty of Sciences, University of Porto, Porto, Portugal**
up201200272@fc.up.pt

Abstract

The corner stone of a language development environment is an editor. For programming languages, several code editors are readily available to be integrated in Web applications. However, only few editors exist for diagrammatic languages. Eshu is an extensible diagram editor, embeddable in Web applications that require diagram interaction, such as modeling tools or e-learning environments. Eshu is a JavaScript library with an API that supports its integration with other components, including importing/exporting diagrams in JSON. Eshu was already integrated in a pedagogical environment with automated diagram assessment, configured for extended entity-relationship diagrams, that served as basis for an usability evaluation.

1998 ACM Subject Classification D.2.6 Programming Environments; Interactive environments

Keywords and phrases Diagram assessment, language environments, automated assessment, e-learning

Digital Object Identifier 10.4230/OASIcs.SLATE.2016.12

1 Introduction

Programming is about languages, but not all languages used in programming are textual. Visual languages are typically used when it is necessary to abstract complex concepts and highlight relationships among them. Visual languages replace named types by simple shapes, such as rectangles or ellipses, that our brain grasps immediately. They also replace identifiers used in textual representations by lines connecting occurrences of these concepts. Still, visual languages have drawbacks when compared to textual languages. The number of types that can be represented with simple shapes is limited and the same goes for the number of connections that a person perceives using lines.

* This work is partially financed by the ERDF – European Regional Development Fund through the Operational Programme for Competitiveness and Internationalisation – COMPETE 2020 Programme and by the FCT within project POCI-01-0145-FEDER-006961 and project "NORTE-01-0145-FEDER-000020" financed by the North Portugal Regional Operational Programme (NORTE 2020), under the PORTUGAL 2020 Partnership Agreement and through the European Regional Development Fund (ERDF).

© José Paulo Leal, Helder Correia, and José Carlos Paiva;
licensed under Creative Commons License CC-BY

Visual languages in programming are mainly diagrams used for modelling[1]. The most popular example is arguably the (Extended) Entity-Relationship diagrams (EER) used for modelling databases. Another example are Deterministic Finite Automaton (DFA) diagrams used for modelling simple computations. The Unified Modelling Language (UML) also specifies a number of diagram types, including the popular Class Diagram, that can be seen as an extension of EER, and the state transition diagram that is an extension of DFA.

A diagram editor is the core component of an environment for diagrammatic languages. The goal of Eshu – the library described in this paper – is to provide diagram editing to Web based applications. Typical clients of Eshu are e-learning systems for diagrammatic languages, in particular those with automated evaluation of diagrams, that provide feedback to the student and need it to be displayed as diagram fragments (inserted and removed nodes and edges) overlaying the original diagram.

There are several programs for editing diagrams, including e-learning systems for certain diagrammatic languages. A survey of these systems is presented in Section 2. This survey covers also libraries for displaying and editing diagrams on web applications, with features in common with Eshu. Nevertheless, they lack the concept of language, both for supporting guided edition and to exchange diagrams with other tools, such as evaluators.

The concept of language is essential in Eshu. It supports the basic concepts of diagrams, such as nodes and edges, and basic operations on them. Nodes and edges may have their own visual syntax according to their type. Restrictions on the type and number of edges connected to a type of node can also be defined. Diagrammatic languages aggregate a collection of types of edges and nodes. Eshu has its own JSON based format for exchanging diagrams of any kind. This format can also be used for transmitting differences that Eshu is able to display over the current diagram. The features of Eshu as well as its design and implementation are detailed in Section 3.

Eshu was integrated in Enki, a web environment for learning computer science languages. The integration is described in Section 4 and served as a validation of the proposed library. It validated both the interoperability features of Eshu, in particular with the graph evaluation module [13], and the usability of the user interface. Section 5 summarizes the main contributions of this work and discusses the opportunities for future research.

2 State of the Art

The most used diagram editors are rich client applications, such as Microsoft Visio or Dia. A few libraries embeddable in web applications, such as mxGraph or GoJS, are also available. This section surveys the features of the systems that served as inspiration for the design of Eshu.

Microsoft Visio[2] is an application for creating diagrams in Windows. The strengths of Visio are the professional and technical diagrams with vector images, which can be magnified and manipulated. Visio can be used to create diagrams of various types, such as organization charts, flowcharts, data modeling (using UML or other graphical notation), network diagrams, floor plans, posters, etc.. Besides the ability to export diagrams into a variety of different formats (for example: PDF, SVG, DWG), it also enables a publishing model in Web format and export/import in XML formats. There are only versions of Visio for Windows. Also, it is a commercial software and it lacks diagram validation.

[1] There are also visual programming languages, but these are seldom used in computer science.
[2] `https://msdn.microsoft.com/en-us/library/office/ff604964(v=office.14).aspx`

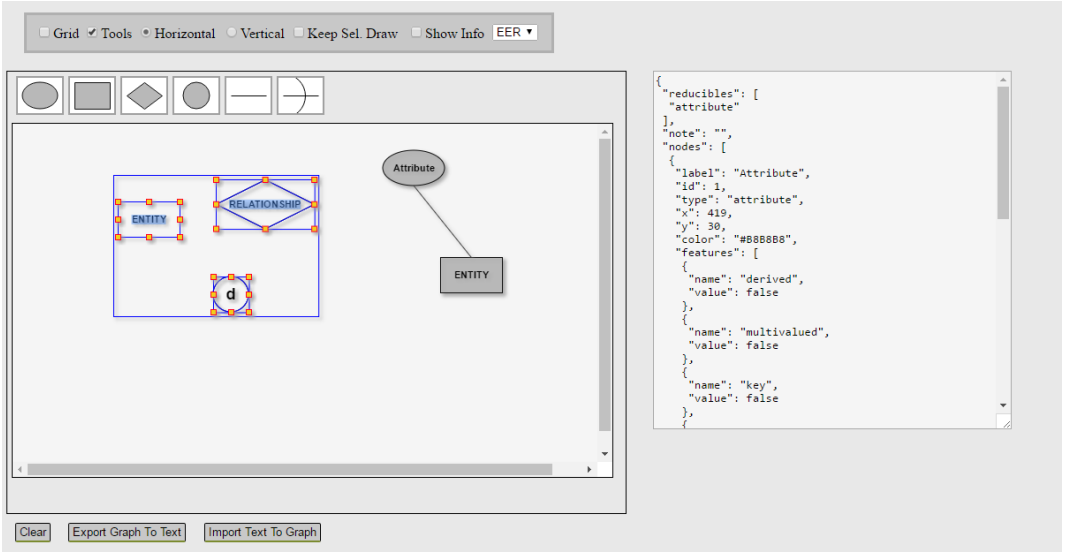


Figure 1 Eshu test application.

Inspired by the Windows program Visio, Dia[3] is a free diagram drawing software running on GNU/Linux, MacOS X, Unix, and Windows. Most diagram objects have handles to which lines can be connected to form graph structures. When the objects are moved or resized, the connections follow their respective objects. Dia has stencils for a wide variety of use cases such as user interface layout, organizational chart, entity relationship diagrams, UML diagrams, network diagrams and flowcharts. It can load and save diagrams to a custom XML format, save files in various formats such as EPS, SVG, xfig, WMF and PNG, among others. As a drawback for educational purposes, Dia allows nodes and connectors from various categories to be mixed together in the same diagram.

The product family library mxGraph [7] provides resources to applications that display interactive charts and graphs with implementations in various technologies. The JavaScript implementation supports several features such as file I/O using XML and JSON, dynamic styling of nodes and edges, event processing, folding of subtrees in acyclic graphs and node grouping.

GoJS [11] is a pure JavaScript library for the implementation of interactive diagrams in browsers. It offers advanced features for user interactivity, such as drag, copy and paste the text editing site, automatic layouts, data binding and templates, event handlers and select groups of nodes. Usually, it runs completely in a Web browser without any need for JavaScript libraries or frameworks, facilitating its integration. It supports diagram serializing in JSON.

Most of the available automatic diagram assessment systems were designed for a specific diagram type and use tailor made diagram editors . Examples of these single diagram types addressed by existing systems are deterministic finite automata (DFA) [2, 10], UML class diagrams [1, 12], UML use case diagrams [14], Entity-Relationship diagrams [3], among others.

[3] `https://wiki.gnome.org/Apps/Dia`

3 Design and implementation

Eshu is a diagram editor running on a web browser, written in JavaScript and using the HTML 5 canvas. This section details the main design points, starting with the user interface, covering also its interoperability and extensibility features, and ending with a few implementation details.

3.1 User interface

Eshu is not a complete application, it is a complex widget comparable to a rich text editor. Its user interface appears in the context of a host application. During the development of Eshu, a test application was developed to mock a host application. Figure 1 presents a screenshot of the test application. In fact, the user interface of Eshu is just the central rectangle displaying a diagram, including the toolbar for selecting nodes and edges. All the other widgets, such as check boxes, buttons and the text window, are part of the test application. These widgets interact with Eshu through its API, in a similar manner as a host application would do.

Eshu is started by the host application like any widget provided by a framework. When started, it has a certain width and height, that by default is also the size of the canvas where the user draws the diagram. If the user drags an object to the right or to the bottom then the canvas will automatically enlarge. The size of the Eshu widget will remain the same and scroll bars will be added to allow the navigation on the enlarged diagram.

The types of diagram elements, nodes and edges, available for the current diagrammatic language are displayed in the toolbar, right above the canvas. As mentioned previously, this panel is an integral part of Eshu. The toolbar may be positioned both vertically and horizontally, or even hidden, by using the appropriate calls to the API. After selecting an element type, the user may insert a new instance anywhere on the diagram and it may even overlap other elements.

Newly inserted nodes are selected, meaning that they will be the focus of the subsequent editing operations. If the user types, the label will be replaced, and if he drags its handles (the small squares on the borders), the node will be resized. Node size is also automatically adjusted according to the label text.

Eshu can also display feedback. Figure 2 presents an example of the feedback shown to the user. In the feedback, green nodes are modifications, blue nodes are additions and red nodes are deletions.

3.2 Interoperability

The main purpose of Eshu is to be integrated in web applications, hence interoperability is a particularly important facet of its design. Eshu has an extensive API that allows the host application to control it. Some of the most important methods of the API require a diagram serialization, to import or export them. Eshu uses its own JSON based serialization, specified using JSON-Schema [4].

Figure 3 presents, as UML classes, the data schema used by the API. As expected, a diagram is composed by a set of `Node` and a set of `Edge`; nodes have a position and a dimension; edges connect a source and a target node. Instances of `Node` or `Edge` are also instances of `Element`, containing an identifier and a type.

An `Element` may also have a temporary status given by an enumerated value, one of `DELETE`, `INSERT` or `MODIFY`. These values are used for stating differences between diagrams,

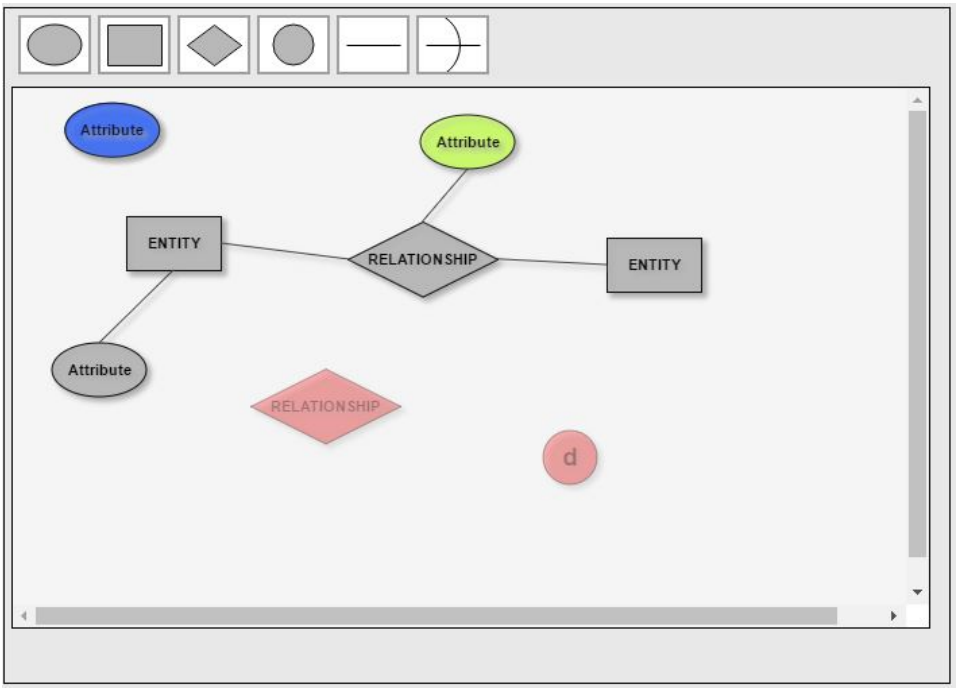


Figure 2 Example of feedback provided by Eshu.

and are used for returning feedback. For instance, an `Element`, either a `Node` or an `Edge` marked as `DELETE` should be removed of the current diagram. Eshu displays in red, blue and light green respectively the elements marked as `INSERT`, `DELETE` and `MODIFY`.

An `Element` may also have a number of instances of `Feature`, which are name-value pairs. Features depend on the type of element, thus cannot be rigidly specified in the schema. For instance, an Attribute is a type of node in an Entity-Relationship diagram which can have different features, such as being a key or multivalued. Thus, an Attribute may have a core graphical representation, a labelled ellipsis, with variants controlled by its features: in a key attribute the label is underlined while a multivalued attribute has a double line ellipsis.

Some features are actually a kind of sub-types. Encoding node characteristics as types or as features is mostly a matter of convenience. One could consider the types "Attribute-Key" and "Attribute-Non-Key" for instance, each with its own graphical representations. Of course, this would lead to a proliferation of types by combining different features that could be rather cumbersome.

Having a large number of types may be relevant for some operations, such as assessing diagrams. Previous research [13] has shown that the assessment of diagrams (represented as graphs) is more efficient when a larger number of types is considered. However, it does not always make sense to use all the features as sub types and it also depends largely on the diagrammatic language being considered. The `reducibles` property of `Diagram` class identifies the names of the types that can be reduced in this fashion, wich ultimately depends on the language.

The diagram serialization is needed by some of the methods provided by the application interface (API) of Eshu. The basic methods are `setGraph()` and `getGraph()`. The former receives a diagram represention as argument while the latter returns one. The method `importGraph()` also receives a diagram serialization but only with differences to the current

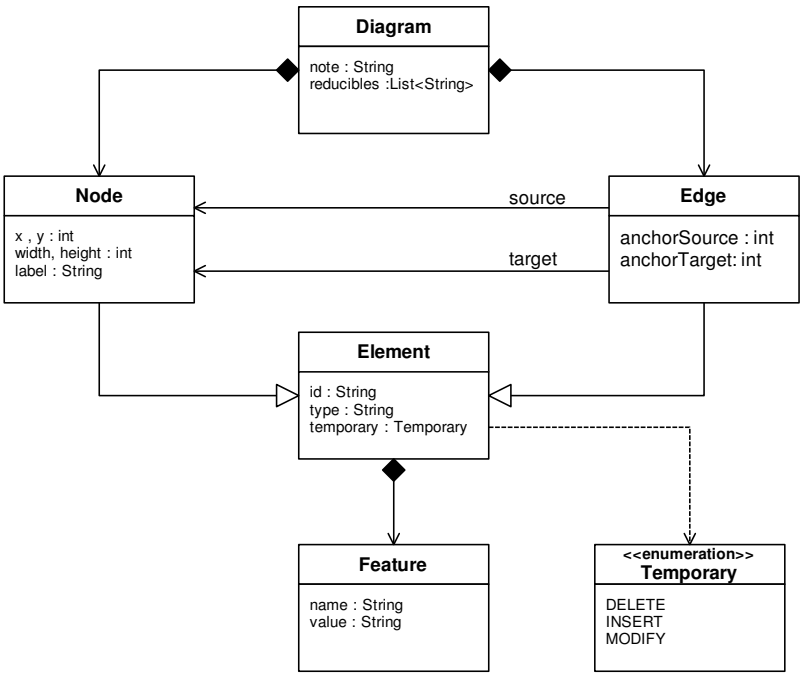


Figure 3 UML class diagram of the diagram serialization used for import/export.

diagram. The diagram serializations imported with this method typically have elements with a `Temporary` value assigned to them and they are displayed accordingly.

The import and export methods are not the only ones in Eshu's API. A partial list of the available methods is presented in Table 1. These methods can be used on an instance of Eshu to configure it. An example of a configuration is the definition of the diagrammatic language, using the `setLanguage()`. An example of binding functions to the user interface would be assigning the copy, cut and paste operations to menu entries or to accelerator keys. As another example, the widgets shown in Figure 1 are all bound to methods of the API.

3.3 Extensibility

Eshu was designed for being extensible, to be able to incorporate new diagrammatic languages, with their own nodes and edges, and enforce restrictions on how these can be connected. These extensions require new JavaScript definitions that have to be integrated with Eshu's source code.

To ease the integration of these definitions, Eshu's code follows an object-oriented approach. JavaScript is not in itself an object-oriented language, but supports to a certain extent a few key concepts of this methodology. For instance, object instances are created with the keyword `new` and a function that acts as a constructor. This function is itself an object with a prototype. The properties of the prototype are shared by all objects it creates, thus these properties can be seen as fields or methods, when they hold values or functions, respectively. It is even possible to simulate inheritance by calling the constructor function of a parent "class".

The new nodes and edges extend abstract definitions provided by the core of the library. These define a number of essential fields, such as position, dimension and label; and also methods for label editing, node resizing, feature selection, moving and dragging, among

Table 1 The application interface (API) of Eshu.

Name method	*Description*
setLanguage	defines the language of the graph
getGraph	returns the graph in json
setGraph	create a new graph using a json text
importDiff	import the differences to the graph
resizeGraph	resize the graph
setPositionHorizontal	change the position of the toolbar to horizontal
setPositionVertical	change the position of the toolbar to vertical
copy	mark to copy a node or a node group
cut	cut a selected node
paste	paste a node or node group that were marked to copy

Listing 1 Entity constructor.

```
function Entity(x, y,id) {
        Vertice.call(this, x, y,id); // extend Vertice
        this.type="entity";
        this.label="ENTITY";
        this.weak=false;
        this.cursorPosition=this.label.length-1;
        this.listTypeCanBeconnected=[
                "attribute", "simpleNode",
                "relationship", "espGenCat"];
}
```

others. However, a few methods have to be explicitly provided, such as the method that displays the element. JavaScript does not have means to enforce the implementation of methods. Thus, mandatory methods are defined by the abstract classes, but the default implementation simply raises an exception unless they are overriden/implemented.

The code snippet in Listing 1 is the constructor of an `Entity` that inherits from `Vertice` all the generic definitions of nodes. The constructor receives a position and an identifier that are passed to the constructor of `Entity`. The constructor also initializes, in this object, specific properties of `Entity`, such as `weak` (by default it is not a weak entity). The last instruction defines a list of node types that can be connected to an `Entity`. Note that in this language two entities cannot be connected directly (only through a `Relationship`), hence the type being defined is absent from this list.

The definition of `Entity` is incomplete without a `draw()` method, as presented in Listing 2. This method receives a graphic context as argument, which the method uses to draw a rectangle, and, optionally, another argument that defines if it is a weak entity or not. Drawing the label at the center is common in most node types, hence this definition can resort to the method `showLabel` inherited from `Vertice`.

The creation of an edge type follows a similar pattern. As shown on Listing 3, the new constructor must extend the `Edge` class which already defines the basic fields and methods required by an edge. Edge takes as argument a source node, a target node, the source anchor index, the index of the anchor and the target node id. It is necessary to define a method `draw()`.

The nodes in the graph are connected with other nodes through edges, but not all nodes can connect with each other. One can set a list of restrictions with objects that check if the

Listing 2 Drawing method of an `Entity`.

```
Entity.prototype.draw = function(ctx) {
        ctx.beginPath();
        ctx.rect(this.x, this.y,this.width, this.height);
        if(this.weak){
                ctx.rect(this.x+2, this.y+2,
                        this.width-4, this.height-4);
        }
        ctx.fill();
        ctx.stroke();

        this.showLabel(ctx);
}
```

Listing 3 Creating a new `Edge`.

```
  function EEREdge1(source,target,handleSource,handleTarget,id) {
        this.links= [
                new SimpleNodeOther(),
                new EntityAttribute(),
                new EntityRelationship(),
                new AttributeAttribute(),
                new AttribuiteRelationship(),
                new RelationshipRelationship(),
                new EntityEspGenCat()];
        Edge.call(this,source,target,handleSource,handleTarget,id);
  }
```

pair of connected nodes has acceptable types. Thus Eshu avoids having links in the graph that are not allowed in the language.

After creating nodes and edges it is necessary to relate them to a language. As shown in Listing 4, the method `setLanguage()` assigns to a language named "eer" a list of previously defined nodes and edge types. This setting has as side effect the creation of a new toolbar in Eshu's user interface.

3.4 Implementation

The implementation of the design described in the previous subsections has some points that need to be detailed. These points are related to the efficiency, automatic layout and integration.

In general, the number of nodes and edges in a diagram is fairly small, since large diagrams are difficult to understand. However, small is relative and for some computers a few dozens of nodes and edges may have impact on efficiency. The most demanding operation from an efficiency point of view is node and edge selection. The iteration over a list of nodes, with a computational complexity of $O(n)$, proved to be too inefficient. Instead, Eshu uses quadtrees, that with a complexity of $O(log(n))$ find nodes and edges more efficiently. The main drawback of using quadtrees for indexing nodes and edges is that it requires reindexing them when they are moved, but the efficiency gain when searching for selected elements pays off.

Listing 4 Defining language EER.

```
 var graph = new Graph(div,700,400);
graph.setLanguage(Language( "err",
        ["attribute","entity","relationship","espGenCat"],
        ["line","lineEGC"]);
```

As a diagram editor, Eshu does not need to layout nodes. The users insert nodes where they please. However, the API allows the host application to send feedback in the form of changes to the existing diagram. If these changes are deletions or modifications, they can be rendered by displaying the existing nodes and edges with a different color. If the difference is a node insertion then it has to be positioned by Eshu.

The layout of these new nodes is computed using a force-directed algorithm [5]. In this approach, nodes repel each other according to Coulomb's law as if they were electrically charged particles with the same signal, and edges bind them together as springs following Hooke's law. One of the advantages of a force directed algorithm is that it adjusts to changes, either changes of window's dimension or changes in the number of nodes.

Eshu is a pure JavaScript library, hence it can be integrated in most JavaScipt based web applications. However, some frameworks, such as the Google Web Toolkit (GWT), use different languages to code the web interfaces; in this case Java. To enable the integration of Eshu in GWT applications, a binding to this framework was also developed. The binding is composed of a Java class (that is converted to JavaScript by GWT) with methods for all API methods, such as those listed in Table 1, implemented using the JavaScript Native Interface (JSNI) of GWT.

4 Validation

This section presents an acceptability evaluation of Eshu. To carry out this evaluation, Eshu was integrated in a web integrated environment for learning computer science languages – Enki [9] – to create a diagram assessment module. The assessment of diagrams was provided by a graph-based evaluator [13]. Then, an experiment was conducted with undergraduate students in the laboratory classes of an undergraduate Databases course at the *Department of Computer Science of the Faculty of Sciences of the University of Porto (FCUP)*, from the 1st to the 18th of March, 2016.

Enki is one of the GUIs of Mooshak 2.0 (the new version of Mooshak [6]), a framework for automated assessment of computer science languages. Enki was designed for a wide range of use cases, from introductory high school or college courses, to massive online open courses. It assumes that the students may have little or no help from a teaching assistant and that they may not have the necessary tools installed on their computers. It was developed using Google Web Toolkit (GWT), an open source software development framework that allows a fast development of AJAX applications in Java.

The diagram assessment component compares 2 diagrams using a graph based serialization obtained from Eshu's API. Given two graphs, a solution and an attempt of a student, it computes a mapping between the node sets of both graphs that maximizes the student's grade, as well as a description of the differences between them. These differences are converted to Eshu's diagram serialization and are shown to the student as feedback.

The Enki course created for the experiment contains resources of two types: expository and evaluative. The expository resources are PDFs describing EER languages. The evaluative

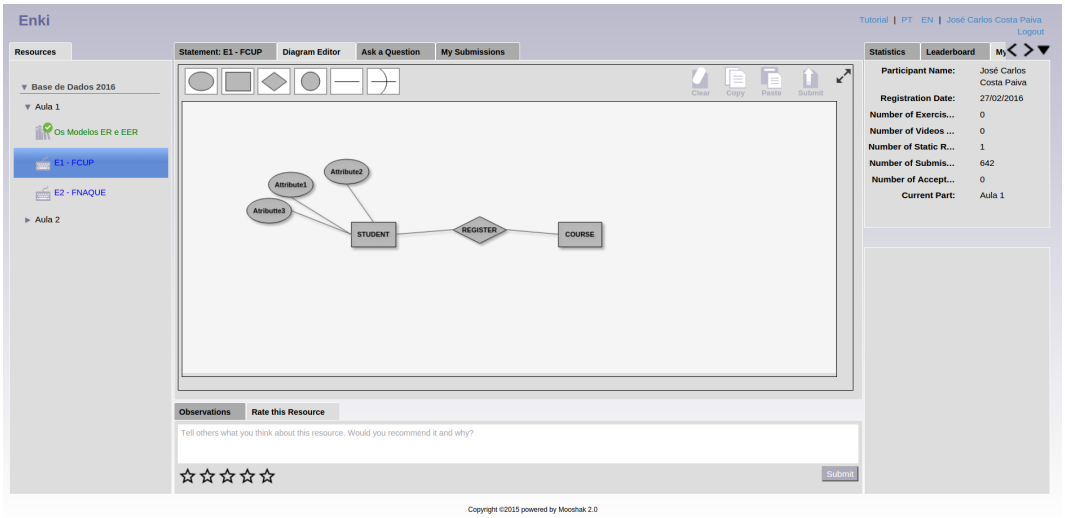


Figure 4 Screenshot of Eshu integrated in Enki.

resources contain the statements describing the database requirements, and an instance of Eshu to create an EER diagram for that database. Figure 4 presents a screenshot of Enki's GUI with Eshu as the editor.

The integration of Eshu in Enki requires a frequent communication between the two, controlled by Enki. Eshu has been designed as a widget, with a binding for GWT, exposing several methods for managing it. An important part of this integration happens when a student sends his diagram for evaluation. Figure 5 presents an activity diagram of this situation.

Firstly, Enki gets the graph from Eshu as a JSON serialization and stores it in a temporary variable. Then, Enki executes an asynchronous request to the graph evaluator to evaluate the graph serialized in JSON. The graph evaluator returns the feedback, observations and a grade of this attempt to solve the exercise. The feedback is a JSON serialization with differences between the attempt and a possible solution to the exercise. These differences are imported temporarily to Eshu, until the student confirms that he is ready to continue from his last attempt. Finally, Enki imports the JSON serialization of the last submission (stored in the temporary variable) to Eshu.

After the experiment the students were invited to fill-in an online questionnaire based on the Nielsen's model [8], using Google Forms. It includes questions on the usefulness of the integration of Enki with Eshu, i.e. on its utility and usability. Utility is the capacity of the system to achieve a desired goal. Usability is defined by Nielsen as a qualitative attribute that estimates how easy is to use an user interface. The survey was completed by 8 students, of which 3 were females.

Figure 6 shows the results grouped by Nielsen's heuristics. The collected data is shown in a bar chart, with heuristics sorted in descending order of user satisfaction.

On the positive side the results prove that the compatibility, consistence and visibility were the heuristics with higher satisfaction. The respondents also selected the ease of use as one of the strongest points of Eshu. On the negative side the results highlighted deficiencies in three areas: speed, reliability and flexibility. Students complained about the difficulty of connecting edges to nodes. This is due to the size of the connecting area of the node. Other students stated that the delay when they validate or submit their programs was too high.

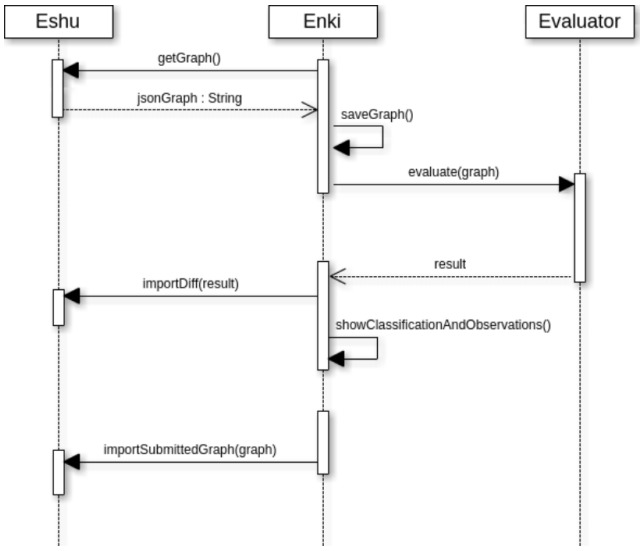


Figure 5 Communication diagram of Eshu with Enki upon a submission to the evaluator.

Most of these issues were already fixed as result of the students' feedback.

The questionnaire finalizes with an overall classification of Eshu in a 5 values Likert-type scale (very good, good, adequate, bad, very bad). Many of the students (37.5%) classified Eshu as an adequate tool and others (37.5%) stated Eshu as a good or a very good tool. Some students (25%) found it either bad or very bad.

5 Conclusions

The creation of a language based environment requires an editor. Many editors are available for textual languages, but the offer is much more limited for diagram editors, in particular for embeddable in web based e-learning environments. Eshu is a JavaScript library that offers diagram editing to web based applications, with an emphasis on the production of valid diagrams within a given language, and in the interoperability with other software components.

Eshu was designed for interoperability and extensibility. It has a diagram definition language for data interchange with other components, which includes complete diagrams and diagram differences used for reporting feedback. This data is serialized in JSON and its schema is formalized in JSON Schema. Although implemented in JavaScript, Eshu follows an object-oriented methodology, which simplifies the introduction of new node and edge types by extending core classes.

To validate the proposed library, Eshu was integrated in Enki, the pedagogical interface of Mooshak 2.0. Eshu is one component of the diagram language module, where it interacts with a graph based evaluation component. This integration validated the interoperability features of Eshu and was also the base for an usability evaluation. To complete the validation of the extensibility features, new languages must also be supported.

The candidates to new languages in Eshu are deterministic finite automata (DFA) and UML diagrams. The former should be straightforward for Eshu (although more demanding from an evaluation point of view). UML diagrams are a bit more complex due to their variety and profusion of features, particularly in class diagrams.

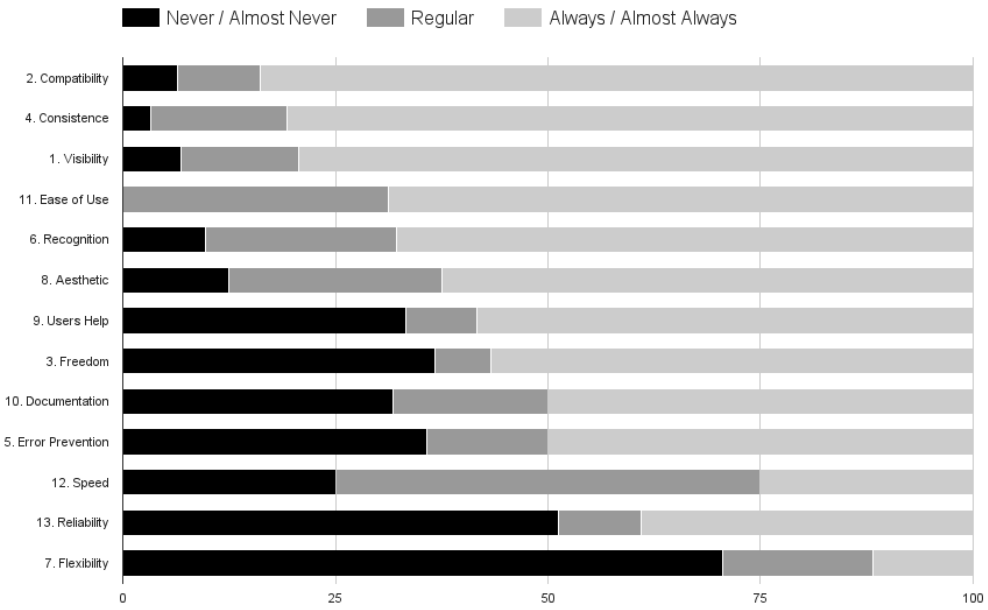


Figure 6 Eshu acceptability evaluation.

References

1 Noraida Haji Ali, Zarina Shukur, and Sufian Idris. A design of an assessment system for uml class diagram. In *Computational Science and its Applications, 2007. ICCSA 2007. International Conference on*, pages 539–546. IEEE, 2007.

2 Rajeev Alur, Loris D'Antoni, Sumit Gulwani, Dileep Kini, and Mahesh Viswanathan. Automated grading of dfa constructions. In *Proceedings of the Twenty-Third international joint conference on Artificial Intelligence*, pages 1976–1982. AAAI Press, 2013.

3 Firat Batmaz and Chris J. Hinde. A diagram drawing tool for semi–automatic assessment of conceptual database diagrams. In Myles Danson, editor, *10th CAA International Computer Assisted Assessment Conference*, pages 71–84. Loughborough University, 2006.

4 Kris Zyp Francis Galiegue and Gary Court. Json schema: core definitions and terminology. Technical report, Internet Engineering Task Force, 2013.

5 Stephen G. Kobourov. Spring embedders and force directed graph drawing algorithms. *CoRR*, abs/1201.3011, 2012. URL: `http://arxiv.org/abs/1201.3011`.

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

7 JGraph Ltd. Build interactive web diagramming apps. Accessed: 2016-03-18. URL: `https://www.jgraph.com/`.

8 Jakob Nielsen and Thomas K Landauer. A mathematical model of the finding of usability problems. In *Proceedings of the INTERACT'93 and CHI'93 conference on Human factors in computing systems*, pages 206–213. ACM, 1993.

9 José Carlos Paiva, José Paulo Leal, and Ricardo Queirós. Enki: A pedagogical services aggregator for learning programming languages. (in press), 2016.

10 Zarina Shukur and Nurul F Mohamed. The design of adat: A tool for assessing automata-based assignments. *Journal of Computer Science*, 4(5):415, 2008.

11 Northwods Software. Gojs – interactive diagrams for javascript and html. Accessed: 2016-03-18. URL: `http://gojs.net/`.

12 Josep Soler, Imma Boada, Ferran Prados, Jordi Poch, and Ramon Fabregat. A web-based e-learning tool for uml class diagrams. In *Education Engineering (EDUCON), 2010 IEEE*, pages 973–979. IEEE, 2010.

13 Rúben Sousa and José Paulo Leal. A structural approach to assess graph-based exercises. In José-Luis Sierra-Rodríguez, José-Paulo Leal, and Alberto Simões, editors, *Languages, Applications and Technologies*, pages 182–193. Springer International Publishing, 2015. doi:10.1007/978-3-319-27653-3_18.

14 Vinay Vachharajani and Jyoti Pareek. A proposed architecture for automated assessment of use case diagrams. *International Journal of Computer Applications*, 108(4):35–40, December 2014. Full text available.

Sni'per: a Code Snippet RESTful API*

Ricardo Queirós[1] and Alberto Simões[2]

1 **ESEIG/IPP & INESC-TEC, Porto, Portugal**
`ricardoqueiros@eseig.ipp.pt`
2 **Centro de Estudos Humanísticos & Centro Algoritmi, Universidade do Minho, Braga, Portugal**
`ambs@ilch.uminho.pt`

Abstract

Today we use the Web for almost everything, even to program. There are several specialized code editors gravitating on the Web and emulating most of the features inherited from traditional IDEs, such as, syntax highlight, code folding, autocompletion and even code refactorization. One of the techniques to speed the code development is the use of snippets as predefined code blocks that can be automatically included in the code. Although several Web editors support this functionality, they come with a limited set of snippets, not allowing the contribution of new blocks of code. Even if that would be possible, they would be available only to the code's owner or to the editors' users through a private cloud repository. This paper describes the design and implementation of Sni'per, a RESTful API that allows public access for multi-language programming code-blocks ordered by popularity. Besides being able to access code snippets from other users and score them, we can also contribute with our own snippets creating a global network of shared code. In order to make coding against this API easier, we create a client library that reduces the amount of code required to write and make the code more robust.

1998 ACM Subject Classification D.3 Programming Languages; D.3.3 Language Constructs and Features

Keywords and phrases Programming languages, interoperability, web services, code snippets

Digital Object Identifier 10.4230/OASIcs.SLATE.2016.13

1 Introduction

With the evolution of the Internet, the act of programming, so far exclusive to standalone IDEs, moved to the Web taking advantage of its full potential regarding ubiquity and speed. This transition fostered code sharing and collaboration in real time, whether the goal is pair programming or just get some help from a friend.

In this context, several Web programming editors appeared, in the last decade, such as ICEcoder, CodeAnywhere, CodeMirror, ACEditor, and many others. These editors are often used by web applications for different purposes: 1) computer programming learning, 2) code development and testing, and 3) programming contests.

Regardless of their purposes, they include a set of features, inherited from traditional IDEs, that aims to facilitate and speed up the coding process [5, 2], such as syntax highlight and checking, tab support, indentation, bracket matching, code folding, keyword shortcuts, spell checking, code generation, refatorization, etc.

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

© Ricardo Queirós and Alberto Simões;
licensed under Creative Commons License CC-BY

One of the most promising features is the code snippets. Code snippets is a well-known IDE feature that increases productivity by reducing the amount of time spent by programmers typing repetitive code or searching for samples [4]. In fact, snippets might be used for standard file skeletons, class and function definitions, HTML tables, and much more.

Despite their importance, most editors do not benefit from the potential that the Web provides in order to leverage code snippets implementation. For instance, there are few editors that support code snippets. Those who support, do not allow the contribution of new snippets and provide only a small range of code snippets based on a private user account or on a specific editor's code repository.

This paper presents the design and implementation of an API – called Sni'per – that allows code snippets management. The RESTful API supports the access to multi-language code blocks and the submission of new snippets to authenticated users. These same users can also score the snippets affecting the results of subsequent requests.

The remainder of this paper is organized as follows. Section 2 analyses several of existing code snippets formats highlighting both their differences and their similar features. In the same section, various snippets API are also covered and a comparative survey is presented based on several criteria. The following section provides details on the design and implementation of Sni'per, including its architecture, the service API, the data model and a client library. The final section summarizes the main contributions of this research and plans futures developments of this service.

2 Related work

The standardization of content and communication is the key for the interoperability on the Web. In this section, we survey the approaches used to formally represent and share snippets over the Web.

2.1 Snippets languages

Code snippets are small blocks of reusable code that can be inserted into a source file to speed up coding. Usually, the representation of a snippet resource is encoded in XML or JSON formats and include the text of the snippet and some metadata for easy discovery. The content of the snippet can be categorized in two groups:

Static snippets: plain text inserted by the user into the source code. The user is not able to specify anything else.

Dynamic snippets: plain text combined with dynamic elements. The user may specify both the content of the dynamic elements, as well as their position relative to the plain text. Good examples are variables such as the current system date or the input from the user that is supplied via a GUI.

Dynamic snippets use *placeholders* to define dynamic elements. These elements are supplied by the user (through graphical user interface and modal dialog boxes) or other external process. Placeholders have special markup syntax that allows the editor to identify the boundaries of placeholders relative to the plain text. There are two typical actions made with placeholders: duplication and transformation. The former allows the user to indicate that the value supplied for one placeholder should be replicated in multiple places, relative to the entire text of the programmable snippet. The later, allows the user to indicate that one or more values supplied for a placeholder should be replicated and transformed in other places within the text of the programmable snippet. For instance, the user may supply a

Listing 1 HelloWorld VS code snippet.

```
<?xml version="1.0" encoding="utf-8"?>
<CodeSnippets
  xmlns="http://schemas.microsoft.com/VisualStudio/2005/CodeSnippet">
  <CodeSnippet Format="1.0.0">
    <Header>
      <Title>Hello World VB</Title>
      <Shortcut>HelloWorld</Shortcut>
      <Description>Inserts code</Description>
      <Author>MSIT</Author>
      <SnippetTypes>
        <SnippetType>Expansion</SnippetType>
        <SnippetType>SurroundsWith</SnippetType>
      </SnippetTypes>
    </Header>
    <Snippet>
      <Declarations>
        <Literal>
          <ID>expression</ID>
          <ToolTip>Expression to switch on.</ToolTip>
          <Default>World</Default>
        </Literal>
      </Declarations>
      <Code Language="VB">
        <![CDATA[Console.WriteLine("Hello, $expression$!")]]>
      </Code>
    </Snippet>
  </CodeSnippet>
</CodeSnippets>
```

document title in one part of the snippet, and specify that the document title should be repeated in other places, with the first instance being all uppercase and every other instance being lowercase.

The previous categorization can include also *scriptable snippets* that consists of runnable segments of code in a scripting language. Although its flexibility, it depends on the programming languages supported by the text editor.

There are several languages to represent code snippets. We detail two of the most popular: VS Code Snippet and TextMate snippet syntax.

2.1.1 VS Code Snippet

VS Code Snippets[1] are predefined pieces of code (e.g. Visual Basic, Visual C#, or XML) that are ready to be inserted into source code within Visual Studio. The VS Code Snippet language uses an XML Schema to formalize a code snippet. Listing 1 shows a simple XML instance that reproduces a "Hello World" snippet:

The root element `CodeSnippets` aggregates two elements: Header and Snippet. The former includes metadata about the snippet, more precisely, its title, author, date, description,

[1] Code Snippets Schema Reference: `https://msdn.microsoft.com/en-us/library/ms171418.aspx`

Listing 2 For Loop snippet for JavaScript using TextMate language snippet.

```
"For Loop": {
  "prefix": "for",
  "body": [
   "for(var ${index} = 0; ${index} < ${array}.length; ${index}++) {",
     "\tvar ${element} = ${array}[${index}];",
     "\t$0",
   "}"
  ],
  "description": "For Loop"
},
```

shortcut and type of the snippet. The type, represented by a set of `SnippetType` elements, can be one of the following values:

- SurroundsWith: allows the code snippet to be placed around a selected piece of code;
- Expansion: allows the code snippet to be inserted at the cursor;
- Refactoring: specifies that the code snippet is used during Visual C# refactoring.

The Snippet element contains the Code element that defines the placeholders and the code snippet. To define a placeholder you must include the Literal element that defines the literals of the code snippet that you can edit and add three sub-elements: ID – specifies a unique identifier for the literal; ToolTip – describes the expected value and usage of the literal; and Default – specifies the literals' default value when you insert the code snippet. Later in the element Code you should signal the placeholder with $. The element has an attribute Language where the language of the code snippet is defined.

2.1.2 Textmate Code Snippet

TextMate snippets[2] are pieces of text which can be inserted into the source code at the current location via a context-sensitive key stroke or tab completion. The text can be, in the simplest case, plain text that you do not want to type systematically (either because you type it a lot, or because the actual text to insert is hard to remember) or dynamic text with interpolated shell code, placeholders and transformations.

The Textmate language uses JSON as the format for the snippet syntax. Each snippet is defined under a snippet name and has a prefix, body and description. The prefix is what is used to trigger the snippet and the body will be expanded and inserted. Possible variables are: $1, $2 for tab stops and $id and $id:label and $1:label for variables variables. Listing 2 shows a For Loop snippet for JavaScript.

Variables with the same id are connected. In the previous example, the $index variable (after set) will reflect the same value for all variables with the same name.

2.2 Code snippets API

With the evolution of Computer Science, several front-end development tools and code snippets repositories appeared having in mind code sharing and collaboration.

There are several tools focused on front-end development, usually coined as live pastebin apps, such as jsbin, jsfiddle or codepen. In this context, Github Gists assumes a leading role.

[2] `https://manual.macromates.com/en/snippets`

Listing 3 Snippet creation with Bitbucket API.

```
$ curl -u {username}
       -X POST https://api.bitbucket.org/2.0/snippets/ \
       -F file=@mySnippet.txt
```

GitHub Gists are a type of pastebin apps that hosts code snippets adding version control, easy forking, and SSL encryption for private pastes.

Regarding code snippets repositories, the most prominent are GistBox, CSnipp, SnippetSource.net, Snipplicious, Bootsnipp, Snip2code and Tagmycode. Most of these repositories offer a GUI to create and manage public or private snippets. If they are public some of the repositories allow the possibility to comment and grade snippets. All of them allow the snippets search based on tags and the "copy & paste" action for the users code editor.

Nevertheless, these are generic tools and do not allow their easy use as a snippet API service. An API for code snippets is not a novel approach. There are a few solutions on the Web that provide access to pieces of code through an API. Most solutions use REST as the software architectural style in order to benefit from its main features (e.g. caching, self-descriptiveness and hypermedia) and JSON as the data format used for the asynchronous browser/server communication. The next subsections details four code snippets API: Bitbucket, Glot, SnippetStash and TagMyCode.

2.2.1 Bitbucket Snippet API

Bitbucket is a Web-based hosting service, owned by the Atlassian group, for projects using revision control systems (e.g. Mercurial, Git). It is similar to GitHub, which primarily uses Git. The Bitbucket cloud hosts several REST API to build third party applications. The snippets API[3] allows you to create, retrieve and delete snippets in the Bitbucket Cloud as well information about them. Snippets can be either public (visible to anyone on Bitbucket Cloud, as well as anonymous users), or private (visible only to the owner, creator and members of the team in case the snippet is owned by a team). Beyond the possibility of managing snippets (e.g. add, update, get and delete), the API allows users to comment and watch for existing snippets. For instance, creating a snippet from a local file is just a single curl command as shown in Listing 3: The API uses JSON as the standard format to exchange data between the server and the browser.

2.2.2 Glot Snippet API

The Glot Snippets API[4] provides an HTTP API for storing and managing snippets. This API enables users to create, update, get, list and delete snippets. Snippets can be saved as either public or secret. Public snippets appear in `/snippets` the endpoint and can be found by search engines. Secret snippets can only be accessed by those who know the URL. In order to create a private snippet an API token is required. Listing 4 shows how to create an anonymous snippet:

This Snippets API uses CouchDB as the datastore engine and JSON for exchanging data.

[3] https://confluence.atlassian.com/bitbucket/snippets-endpoint-719095086.html
[4] https://github.com/prasmussen/glot-snippets/tree/master/api_docs

Listing 4 Snippet creation with Glot API.

```
$ curl --request POST \
       --header 'Content-type: application/json' \
       --data '{
        "language": "python",
        "title": "test",
        "public": true,
        "files": [{"name": "main.py", "content": "print(42)"}]}' \
       --url 'https://snippets.glot.io/snippets'
```

Listing 5 Snippet creation with Glot API.

```
$ curl -u my@email.com:d53a1fb463c7e3180f3ac0f1479ec7daffa2b9b7 \
  http://www.snippetstash.com/snippets.xml
```

2.2.3 SnippetStash API

The SnippetStash API[5] is a REST-based API which allows users to integrate snippets in an easy and uniform way. The SnippetStash has a *public and a private API* hosted in the main endpoint `http://www.snippetstash.com`. The *public API* has three endpoints:

- `/snippets/latest.xml` – obtains the latest twenty snippets in XML format;
- `/tags.xml` – retrieves all existent tags;
- `/tags/[tag].xml` – retrieves all the snippets associated with [tag].

The responses to these requests use an ad-hoc XML representation.

The *private API* allows users to access and manage personal snippets. Since SnippetStash supports authentication via OpenID and basic password authentication, it is necessary to use an API key to access the API functions. After registration, the API key is sent by e-mail to the registered user. To access the API, you simply provide the email address and the API key on every request. For instance, to get your snippets returned as XML use the code included in Listing 5:

Other actions supported by this API are creating, editing and sharing snippets.

2.2.4 TagMyCode Snippet API

The TagMyCode API[6] is a RESTful service that enables users to access to snippets and tags. Firstly, you need to register a new application to get a consumer id and secret. Most of the TagMyCode API calls needs authentication. OAuth 2 is the only way to authenticate your requests. After the authentication, you can execute several actions based on the main endpoint `https://api.tagmycode.com/`, such as:

- `/account?access_token=YOUR_TOKEN` – get logged user information;
- `/languages` – list languages;
- `/snippets` – list snippets;
- `/snippets/:id` – get single snippets;

[5] `http://www.snippetstash.com/api`
[6] `https://tagmycode.com/`

Table 1 Snippet API comparision.

Features	Bitbucket	Glot	SnippetStash	TagMyCode
Web Service	REST	REST	REST	REST
Authentication	HTTP Basic	HTTP Basic	OpenID & Basic	OAuth
Storage	–	CouchDB	–	–
Response format	JSON	JSON	XML	JSON
CRUD actions	YES	YES	YES	YES
Social	Comment	NO	Share & Unshare	NO
GUI	NO	NO	YES	YES
# public snippets	45	87	199	1350
# languages	6	17	19	59

- `/snippets` – create a new snippet; this is a HTTP POST request with the following parameters: the id of the language, the title of the snippet, the description of the snippet (optional), the tags of the snippet (comma or space separated) and the optional boolean private that specifies if the snippet is private;
- `/snippets/:id` – edit a specific snippet; this is a PUT request with the same parameters as the previous endpoint;
- `/snippets/:id` – removes a specific snippet; this is a DELETE request with the identification of the snippet to remove;
- `/search` – search for snippets based on a search keyword.

2.2.5 Snippet API comparision

In this section, we present a comparative table that surveys the four snippets API previously presented. Table 1 compare the four snippets API based on several criteria.

All the snippets API use REST as the software architectural style because of its simplicity.

Regarding authentication, basic access authentication is the preferred method providing a username and password when making a request.

JSON is the elected exchange format. In fact, JSON has a much smaller grammar when compared with XML and maps more directly onto the data structures used in modern programming languages such as JavaScript.

All the API support common actions over snippets, such as, the creation of snippets through the HTTP POST, their edition using the HTTP PUT method, getting a specific snippet or all snippets through the HTTP GET and remove a specific snippet using the HTTP DELETE method. Moreover, snippets can be found by using filters based on tags. Regarding social features, Bitbucket supports comments on snippets and SnippetStash allow users to share snippets with others.

Beyond the documentation of the API, some API (SnippetStash and TagMyCode) have a graphical interface where users can perform all the CRUD actions and others such as embed the snippet in a HTML page or share it on social networks (e.g. Facebook, Twitter).

Lastly, to assess the use and diversity of content API, we obtained data on the number of public snippets and programming languages represented. Based on the numbers, it can be concluded that the SnippetStash and TagMyCode are the API with more activity.

3 Sni'per

This section describes the details on Sni'per server implementation.

Table 2 Snippet API.

Function	REST syntax
List languages	`GET /langs`
List language snippets	`GET /`*lang-id*
List snippets per language/type	`GET /`*lang-id*`/`*keyword*
Retrieve snippet	`GET /snippet/`*snippet-id*
Register snippet	`POST /`*lang-id*`/`*keyword*`/`*username* < *auth-hash, snippet*
Retrieve user info and snippets	`GET /user/`*user-name*
Delete snippet	`DELETE /`*lang-id*`/`*keyword*`/`*username*`/`*auth-hash*
Register	`POST /user/`*user-name* < *email, password*
Authenticate	`POST /user/`*user-name* < *password*
Vote on snippet	`POST /snippet/`*snippet-id*`/`*voting-user* < *auth-hash*

3.1 Architecture

Sni'per is a complete JSON REST service, implemented in Perl, using Dancer 2 Web Framework. This server interacts with a MongoDB NoSQL server, using the Perl Moo OO module. Thus, requests are intercepted by the web server, that translate the request route in a query for the object model. The object model queries the database, and retrieves an answer that is later serialised to JSON.

As Sni'per uses a RESTful approach [1], where HTTP verbs are used to indicate the type of each operation, and without storing any information between each request. This means that each client needs to authenticate itself for each operation that requires it. In order to allow clients not to store the user password, and send it over and over for each request, Sni'per uses a token-based authentication. The client performs its authentication sending the user name and password. It receives a token that identifies that user in his/her consequent requests.

3.2 Service API

Table 2 lists the most important routes available in Sni'per API. Note that for simplicity we decided not to include the full URI, but just the route. Routes requiring extra explanation are:

- The registration and authentication that use the same route but behave differently accordingly with the supplied arguments.
- Users can have only one snippet per language/keyword pair. Note that this is the usual behaviour for most text editors. If the user wants to have different snippets for the same language keyword, they name it differently (for example, `for` and `fori` where the first cycles over the elements of a collection, and the second creates a typical index-based cycle).
- The voting route is used both to vote or remove a vote. Unlike other websites, like *Stack Overflow*, we decided to have a vote-only approach (or the starring mechanic). If the user likes a snippet it can vote or star it. If the user tries to like it again, the vote is removed.
- Although there is nothing against the definition of a body to a `DELETE` request, some web servers and proxies drop that information. Therefore, our `DELETE` routes include, explicitly, all required parameters for authentication.

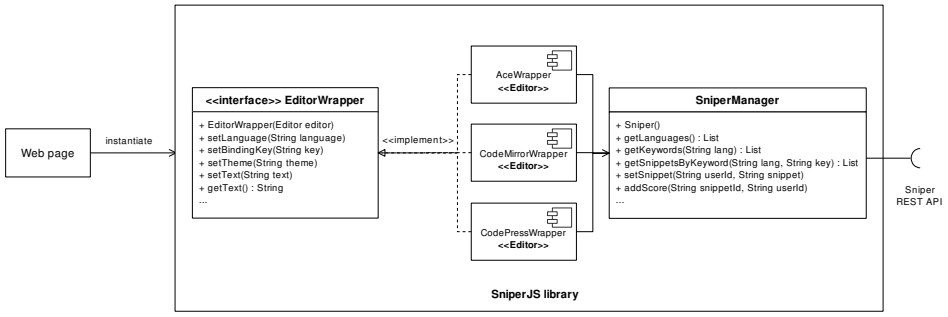


Figure 1 Components diagram of the SniperJS library.

As pointed before, the database uses a NoSQL approach, storing information in a MongoDB database [6]. At the moment, the database consists of a set of collections, for users, languages, snippets and votes. This requires the system logic to be implemented in the OO model representation. In the future we expect to use more features from MongoDB, reducing the complexity of the system logic.

3.3 Client library

In order to foster the use of the Sni'per API on Web environments, we implement a JavaScript library called SniperJS. Figure 1 shows the component diagram of the library. The cornerstone of the library is the interface EditorWrapper that defines the actions that a specific editor must follow. Thus, to support a specific editor, it is necessary to implement this interface through the creation of a wrapper. The interface has the following functions:

- EditorWrapper([Editor] editor) – instantiates a new wrapper;
- setLanguage(String language) – defines the language of the editor;
- setBindingKey(String key) – defines the key combination that will trigger the Sni'per management modal box;
- setTheme(String theme) – defines the theme of the editor;
- setText(String text) – includes new text on the editor;
- getText() : String – obtains text from the editor.

In the HTML page, we instantiate a EditorWrapper passing a reference for the specific editor. Listing 6 shows the initialization of the Ace editor. In turn, the wrapper is responsible for communicating directly with the Snippet API through the SniperManager class that provides a set of functions to facilitate the process of interaction with the Snippet API.

This architecture promotes and facilitates the use of Sni'per API. If a programmer does not find the desired wrapper, he/she only has to create a new one, that implements the EditorWrapper interface and, then, use the SniperManager class to interact with the API without the need to handle HTTP requests or even parsing JSON reponses.

The GUI for the Sni'per library had two design requirements: be simple and responsive. Based on these requirements we opt for using Bootstrap for the creation of the graphical interface of the Sni'per JS library. Bootstrap is an open source frontend Web framework. Since version 2.0 the framework supports responsive web design. This means the layout of web pages adjusts dynamically, taking into account the characteristics of the device used (desktop, tablet, mobile phone). Figure 2 shows the modal box that allows users to select the desired snippet.

Listing 6 Instantiation of the Sni'per library.

```
<body>
...
<div id="editor"></div>
...
<script>
    // Ace editor initialization
    var aceSniper = new EditorWrapper(ace.edit("editor"));
    aceSniper.setLanguage("c_cpp");
    aceSniper.setTheme("ace/theme/cobalt");
    aceSniper.setBindingKey("Shift-Return");
</script>
...
</body>
```

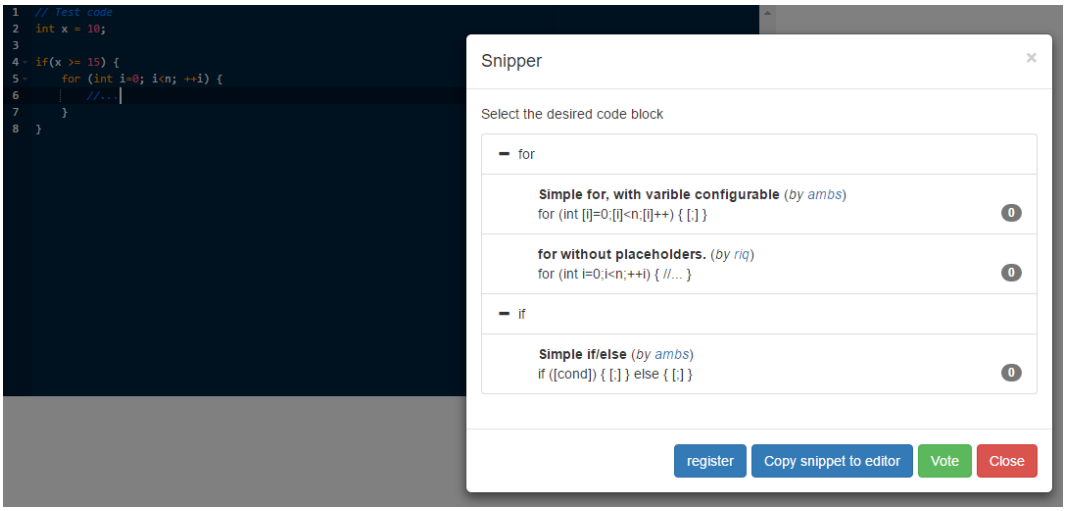


Figure 2 Sni'per library graphical interface.

4 Conclusions

The goal of the research presented in this paper is to promote the interoperability among Web code editors through the use of a standard API for code snippets management. The proposed approach is a service to manage code snippets on-the-fly. The contribution of this research is twofold, the service definition and a client library implementation that will use the service. The service definition comprehends the modular design of the snippet service based on a RESTful API with a set of functions for snippets management such as getting snippets based on a keyword, submit a new snippet of a specific language or vote on a single snippet. The client of the Sni'per service is a library implemented in JavaScript with a Bootstrap-based GUI. Currently, Sni'per supports only a few languages with few snippets each. The idea was initially to have a functional proof of concept. The API can be accessed in the following link `sniper.zbr.pt`[7]. As future work the authors will:

[7] To be available soon.

- increase the number of snippets on the cloud snippets repository
- extend the Sni'per compliance to other snippet formats based on the implementation of injectors of other snippets DSLs
- use source code context to enhance snippet retrieval and parameterization [8], [3]
- improve dynamic snippets using placeholders to define dynamic elements with duplication and transformation actions [7]

References

1 Roy T. Fielding and Richard N. Taylor. Principled design of the modern web architecture. In *Proceedings of the 22Nd International Conference on Software Engineering*, ICSE '00, pages 407–416, New York, NY, USA, 2000. ACM. doi:10.1145/337180.337228.

2 Joel Galenson, Philip Reames, Rastislav Bodik, Björn Hartmann, and Koushik Sen. Codehint: Dynamic and interactive synthesis of code snippets. In *Proceedings of the 36th International Conference on Software Engineering*, ICSE 2014, pages 653–663, New York, NY, USA, 2014. ACM. doi:10.1145/2568225.2568250.

3 Tihomir Gvero, Viktor Kuncak, and Ruzica Piskac. Interactive synthesis of code snippets. In Ganesh Gopalakrishnan and Shaz Qadeer, editors, *Computer Aided Verification: 23rd International Conference, CAV 2011, Snowbird, UT, USA, July 14-20, 2011. Proceedings*, pages 418–423. Springer Berlin Heidelberg, 2011. doi:10.1007/978-3-642-22110-1_33.

4 Miryung Kim, Lawrence Bergman, Tessa Lau, and David Notkin. An ethnographic study of copy and paste programming practices in oopl. In *Empirical Software Engineering, International Symposium on*, pages 83–92, Aug 2004. doi:10.1109/ISESE.2004.1334896.

5 Torben Lorenzen, Lee Mondshein, Abdul Sattar, and Seikyung Jung. A code snippet library for cs1. *ACM Inroads*, 3(1):41–45, March 2012. doi:10.1145/2077808.2077822.

6 Zachary Parker, Scott Poe, and Susan V. Vrbsky. Comparing nosql mongodb to an sql db. In *Proceedings of the 51st ACM Southeast Conference*, ACMSE '13, pages 5:1–5:6, New York, NY, USA, 2013. ACM. doi:10.1145/2498328.2500047.

7 Naiyana Sahavechaphan and Kajal Claypool. Xsnippet: Mining for sample code. *SIGPLAN Not.*, 41(10):413–430, October 2006. doi:10.1145/1167515.1167508.

8 Doug Wightman, Zi Ye, Joel Brandt, and Roel Vertegaal. Snipmatch: Using source code context to enhance snippet retrieval and parameterization. In *Proceedings of the 25th Annual ACM Symposium on User Interface Software and Technology*, UIST '12, pages 219–228, New York, NY, USA, 2012. ACM. doi:10.1145/2380116.2380145.

Building a Dictionary Using XML Technology

Alberto Simões[1], José João Almeida[2], and Ana Salgado[3]

1 **Centro de Estudos Humanísticos, Universidade do Minho, Braga, Portugal**
ambs@ilch.uminho.pt
2 **Centro Algoritmi, Universidade do Minho, Braga, Portugal**
jj@di.uminho.pt
3 **Instituto de Lexicologia e Lexicografia da Língua Portuguesa, Academia das Ciências de Lisboa, Lisbon, Portugal**
anacastrosalgado@gmail.com

Abstract

In this article we describe the workflow implemented to convert a dictionary saved as a PDF file into an XML document and posterior importation into an XML aware database, and the process to edit, add and delete new entries. The conversion process was challenging given the format of the PDF file, and the fine grained detail of the XML schema that was used. For that, an iterative filtering approach was used. To store the dictionary we decided to use an XML aware database (eXist-DB), that stores each dictionary entry as a separate resource. It can be queried used a web interface developed using XQuery. The lexicographers can edit entries using the oXygen XML editor, reading and storing them directly in the database. In order to guarantee incremental backups, it was defined a mechanism to import the XML database into a GIT repository. Finally, a couple of programs were created in order to prepare regular reports on the dictionary revision process, as well as to backup it in a GIT repository.

1998 ACM Subject Classification I.7.2 Document Preparation / Markup languages

Keywords and phrases XML databases, dictionaries, XQuery, PDF files

Digital Object Identifier 10.4230/OASIcs.SLATE.2016.14

1 Introduction

After the release of the *Dicionário da Academia das Ciências de Lisboa* (DACL) in 2001 [1], our goal is to recover that work, update the dictionary data and publish it both on the Internet, as a web application, and as a conventional paper dictionary.

We do not intend to publish the dictionary, or even make it available to the public, as it is. Our aim is to manually revise the full dictionary, fixing known errors, detecting others, and including new terms. That is, we want to create a new version of the dictionary to be made available in the web for free by the end of the next year.

The main problem arose when it was found that the only source for the dictionary itself, was from a PDF file. [1]. There was no time nor money to allow the full transcription of the document. This required an automated process to recover the data from the PDF file. In order to achieved this, our previous recovering other dictionaries [3, 5] was crucial, and allowed this process to be faster.

[1] This PDF file was created from a Word document, that was generated from a Microsoft Access file, but none of these files were still available.


© Alberto Simões, José João Almeida, and Ana Salgado;
licensed under Creative Commons License CC-BY

This process resulted in a text file with font face and size information. Then, a set of rewriting rules were written, to convert this information into a basic XML structure that was later enriched using an iterative filtering approach, as we will describe.

The resulting XML was split into smaller documents, one for each dictionary entry. These documents were later imported into an XML aware database. In this case, we chose eXist-DB [4].

In order to allow the revision and enrichment of the dictionary, it was needed an interface or some other mechanism to allow linguists to edit the dictionary records. With that in mind, we created an application based on XQuery to allow the navigation of the dictionary, and rendering of links using the XMLDB protocol, that oXygen XML editor[2] could understand, making it possible to edit an entry using a single mouse click.

Finally a set of reporting and export tools were developed in order to monitor the pace of the dictionary revision, and the export of the dictionary into different formats.

In the next section we will focus on the task of understanding the PDF document format and exporting it into XML TEI [7]. Follows Section 3 that explains how the dictionary was imported and indexed into a XML-aware database. Section 4 present the developed tools to help maintaining and validating the dictionary. Finally, we conclude in Section 5 with some insights on next steps on the development of the new dictionary.

2 Rewriting PDF into XML

In previous projects we had already applied techniques in order to convert dictionaries encoded in different formats into some XML schema. Each one of these data conversion tasks resulted in very different challenges. The same holds true for the PDF format conversion of the *Dicionário da Academia das Ciências de Lisboa*.

As described earlier, the PDF file was generated from a Microsoft Word document. The format is a two column template, using mostly text in a same font-size, but changing its style, including normal, italic, bold and small capitals. The tools we tried to convert the PDF to text did not provide satisfactory results:

- *pdftotext* command line utility loses all formatting, which means that all the information codified as different font styles is completely lost.
- *pdftohtml* as similar problems, including the fact that it also lost all non-ASCII characters, namely the phonetic transcriptions (using modified IPA) and the etymological information which could include Greek words. Note that *pdftohtml* includes an option to output XML, but it is unable to detect some user-defined fonts. Although not tested, we expect *pdf2xml*[3] to behave similarly, given it uses, just like *pdftohtml*, the *Xpdf* library. It was not tested as its source code is not prepared for easy installation, lacking the usual configuring mechanism present on most OpenSource tools[4].
- OCR tools, like the free *Tesseract* OCR, and commercial tools, like *Omnipage Pro* and *Abbyy Fine Reader* converted the vectorial PDF files into images, and then tried to recognize the text. Their results were also far from satisfactory even when using high quality images. Also, during this process, when specifying Portuguese as the main language, they would miss the detection of non Portuguese words.

[2] `https://www.oxygenxml.com/`
[3] `https://github.com/eliask/pdf2xml`
[4] There are some binaries for the *pdf2xml* tool in SourceForge, but it is known that SourceForge was hacked more than once, and therefore binaries available there are not to be trusted.

```
<text font="KIKNHC+Garamond-Redondo"  size="9.548">v</text>
<text font="KIKNHC+Garamond-Redondo"  size="9.548">o</text>
<text font="KIKNHC+Garamond-Redondo"  size="9.548">g</text>
<text font="KIKNHC+Garamond-Redondo"  size="9.548">a</text>
<text font="KIKNHC+Garamond-Redondo"  size="9.548">l</text>
<text font="KIKNHC+Garamond-Redondo"  size="9.548"> </text>
<text font="KIKNHC+Garamond-Redondo"  size="9.548">c</text>
<text font="KIKNHC+Garamond-Redondo"  size="9.548">e</text>
<text font="KIKNHC+Garamond-Redondo"  size="9.548">n</text>
<text font="KIKNHC+Garamond-Redondo"  size="9.548">t</text>
<text font="KIKNHC+Garamond-Redondo"  size="9.548">r</text>
<text font="KIKNHC+Garamond-Redondo"  size="9.548">a</text>
<text font="KIKNHC+Garamond-Redondo"  size="9.548">l</text>
```

Figure 1 XML as generated by PDFBox to the "*vogal central*" string.

Apache PDFBox tools[5], on other hand, proved to be very effective.. This tool is able to process a PDF file and generate a (large) XML file, where each character in the PDF file is encoded into an entity with the typeface and font size, as shown in figure 1. This XML file takes 2.7GB of disk space, with 28.5 million `text` tags.

This XML was then simplified, joining all text with he same typeface and font size into single elements. Then, a list of different pairs $(font, size)$ was created, and mapped to different tag elements. For the common typeface and font size adding a bold, italic or small caps were used the elements *b*, *i* or *sc*. For other, less clear strings, other *ad-hoc* elements were defined.

A set of rewrite rules were then written in order to process this file and identify what role each string played in the dictionary, determine what was the entry term, its phonetic transcription, sense identifiers, synonyms, quotes, etc. The techniques employed here were similar to the ones applied previously [5].

After this step, our main challenge was the granularity of the desired markup, which was more fine-grained than in previous works which made the XML processing time step take too much time. A simple addition of a rewrite rule could make the entire process to take up to 30 minutes. This kind of approach was hard to maintain, as it was not easy to perform simple tests. Also,

At some point we noticed that some entries were already correctly formatted, and we were concerned that writing rules for more complex ones could affect the correct entries. At this point, we implemented an iterative filtering mechanism. First, the XML file was divided into multiple files, one for each entry. Then, the following algorithm was applied:

1. define a basic DTD for simple entries;
2. test all single entry files against the DTD and move the valid ones into another folder;
3. include a set of new rules to process the remaining files in order to annotate extra information;
4. update the DTD to support the new elements and structure, while maintaining the validity of previous validated files;
5. go to the step 2.

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

This approach was quite useful, not only because we guaranteed that new rules would not damage entries already correctly formatted, but also allowed us to define the minimum needed DTD: elements, their arity and attributes were added only when there was an entry needing for them.

In the end, a list of about two hundred entries were not validated by the DTD. At this point we noticed that most of the errors were related to mistakes (incoherences, mostly) of the original edition. As the original dictionary was created based on a Microsoft Access database[6] without integrity constraints on the entry contents, it was not possible to guarantee the quality of the work. Due to this, mistakes that were found after producing the dictionary Word document were correct manually on the document itself. Also, some characters were added or removed in order to guarantee some graphical format but, at the same time, break the consistency with the dictionary entry format that was enforced by the Word document generation process. These entries were edited manually using a tool developed specifically for this purpose. This tool allowed the user to edit an entry and save it after being validated.

This iterative process made it possible for all 69,428 entries of this dictionary to be validated against the created DTD.

3 Indexing the XML dictionary

The obtained dictionary has some errors, such as words broken in two, given the transliteration not being detected by our previous tools, or some of the phonetic and Greek letters not being detected. Unfortunately most of these errors will need to be fixed manually.

If we had only one lexicographer working on this dictionary, it could be possible to use a single file, and allow to edit it at once in an XML aware editor. With the intent of allowing more than one person to edit the dictionary, and given that the complete dictionary is, formally, a sequence of the same element (*entry*), we split the file into 69,428 smaller XML files. This also allows the editor to open and validate each entry much faster, than analyzing the full file. In order to allow the better understanding of the discussion that follows, listing 1 presents the content of the dictionary entry for the terms *vassoura/vassoira*.

A dictionary entry can have more than one headword. As a simple example, the term "*vassoura*" (broom) can also be written as "*vassoira*". So, it is not possible to rename each one of the XML files to the term it defines (at least, not for every entry, or we would have duplicated entries, saved with different names). To suppress this problem, we decided to use the first orthographic form of the entry. In case of polysemous words (words appearing as the first orthographic form in more than one entry) we concatenated the word name with the acception number (separated by an underscore).

This process makes it easier to find an entry. But for some situations it is still difficult. How will the lexicographer know if a specific word is the first orthographic form of its entry, or even, if a specific word has more than one meaning registered in the dictionary? Therefore, we needed to develop a mechanism to allow the lexicographer to search for specific terms, and looking to their entries' contents, choose the one to be edited.

Instead of developing an in-house solution for this problem, we decided to give a try with an XML-aware database. The choice was the well known eXist-DB [4], not just because it is free and open-source, but also because it has extensive documentation.

Our long term plan is to develop a simple interface to allow the maintenance of the dictionary, based on the XForms [2] standard. Meanwhile, while that is not possible, and in order to allow lexicographers to start their work revising the dictionary, we adopted oXygen

[6] Unfortunately this database is lost, explaining why all the work on re-engineering the PDF document.

Listing 1 Entry for the word *vassoura/vassoira* encoded in XML.

```
<entry [...]>
  <term>
    <orth>vassoura</orth> <orth>vassoira</orth>
    <pron>vɐs'oɾɐ</pron>  <pron>vɐs'ojɾɐ</pron>
  </term>
  <gramGrp>s. f.</gramGrp>
  <etym>Do lat. <mentioned>*versoria</mentioned>, de <mentioned>versus
</mentioned>, part. pas. de <mentioned>verrĕre</mentioned> 'varrer'</etym>
  <sense n="1">
    <def> Utensílio doméstico formado por um cabo longo ou curto ao qual é
fixado, numa das extremidades, um feixe de folhas de palma, piaçaba,
sorgo, pêlos naturais ou artificiais... e que serve para varrer o lixo.
<quote type="example">O cabo da vassoura partiu-se. Deitou fora a vassoura
porque tinha os pêlos gastos.</quote></def>
[...]
  <sense n="7"><usg type="geo">Bras.</usg>
    <def>Pessoa que ganha sempre ou quase sempre em sorteios, jogos de
azar... </def></sense>
[...]
</entry>
```

XML Developer[7]. This choice was backed by the tight cooperation between the developers of eXist-DB and oXygen. Example of that cooperation is the wizard available to connect to eXist-DB from oXygen. With some extra work it was also possible to tweak Mozilla Firefox to open URIs using the `oxygen://` protocol. This allows the lexicographers to do a query in the eXist web pages, search for the entry to edit, and open it with a simple click on a link.

Finally, to make the 69K documents searchable, the eXist-DB collection was configured to be indexed by Lucene[8]. Note that Lucene is part of eXist-DB and does not need to be installed separately. The only requirement is the creation of a configuration file, to enable full text search, and specifying which elements of the XML documents are to be indexed. For the dictionary we created two different indexes with two very distinct goals:

- The first is an index for the entries' orthographic form. These index allows the lexicographers to search for a specific term entry. Given the index is only over the content of an element, it is quite small and efficient.
- The second index allows the reverse-search [6] of the dictionary. This type of search mechanism is quite interesting when analyzing a dictionary, as it allows to search for entries not by their head word, but using their definition. This index is quite large, as it contains all the dictionary text.

4 Developed Tools

When choosing eXist-DB to store our XML documents we ended up choosing a complete solution for web application development. Although the database can be used using different APIs, like REST or xmldb protocols, eXist-DB suggest users to create applications on top of it. eXist-DB allows the development of web applications using standard W3C protocols, like XQuery and XForms. It also includes a full web Integrated Development Editor that allows the programmer to run queries but also to edit the application code.

[7] `http://oxygenxml.com/`
[8] `https://lucene.apache.org/`

Listing 2 XQuery script to validate the dictionary collection.

```
xquery version "1.0";
declare namespace validate="http://exist-db.org/xquery/validation";

<reports> {
   for $doc in collection("/db/academia")
   let $file := fn:base-uri($doc)
   return <file uri="{ $file }"> {
       validate:jaxp-report(doc($file),true())
   } </file>
} </reports>
```

Instead of developing our tools in an external programming language, we decided to test how far we could go with eXist-DB. Not just for the sake of analysis of the tool, but also for portability. It would be much easier to port the dictionary application to the final servers if it uses just one technology.

The next list describes briefly the tools we implemented to help in the management of the dictionary. Note that some are simple XQuery scripts, to be run in the terminal, while other are end-user interfaces, developed for the web.

Validator
: The first task was the development of a script to validate every entry in the database. Although eXist-DB supports different kind of validations, we preferred to create a standalone tool. This allows us to remove liberty points in the schema, turning it more restrict, and test how many entries would be affected. As simple as this script may seem, it took some time before it could validate all the collection entries in a reasonable time. Therefore, listing 2 presents our XQuery script. This script takes about 3 minutes to validate the 69K entries in a Quad-core Xeon 2.40GHz, outputting an XML document with existing errors.

Search
: As explained before, the lexicographers use a web application to query the dictionary and obtain the filename where the entry is encoded. This script allows both the search by an orthographic form (searching entries in the `orth` element) or doing reverse search (looking up in every PCDATA section of the XML document). The XQuery script returns the complete entries, in the original XML format. A Cascading Style Sheet is then used to make the content adequate to be viewed in a web browser. Figure 2 shows the *vassoura/vassoira* entry as presented currently in the web application[9].

New entry
: As stated earlier, at the moment the lexicographers are using oXygen to edit the dictionary. To create from scratch a dictionary entry is not a simple thing, specially when the user XML competences are not strong. To simplify this process, a small XQuery script that, based on the term, validates if it already exists, and in case it does not, create a boilerplate XML document in the database, that can then be edited.

List domains
: Although lexicographers will review all dictionary entries, before the dictionary publication, the definitions of some words are prepared by specialists in different areas (like mineralogy or astronomy experts). For those, the entries are exported as rendered in the browser, so

[9] Note that currently the CSS is hiding the phonetic transcription, and that the last part of the entry, with the diminutive form of the word, is being wrongly considered part of the last definition, and therefore, appearing in the wrong position.

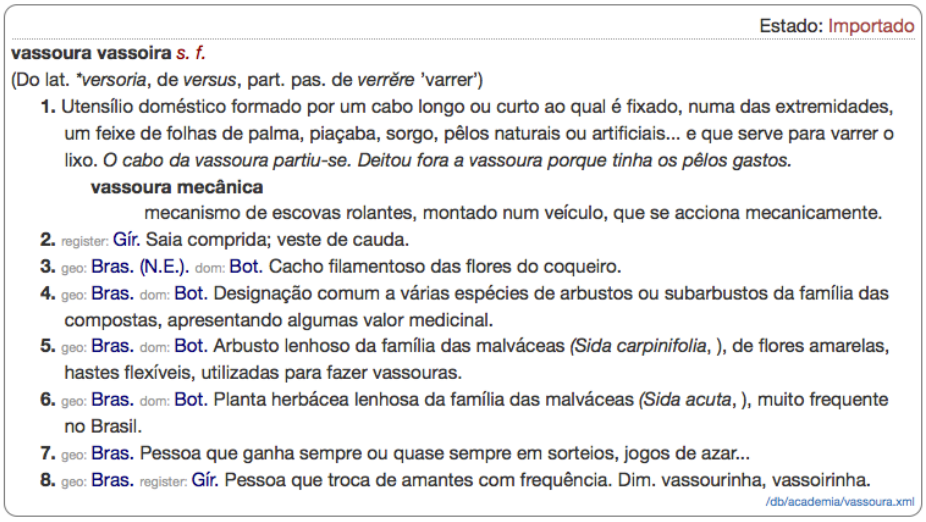


Figure 2 Entry for *vassoura/vassoira* as presented to the lexicographer in the browser.

they can validate them. As most of them are elder and prefer not to use the computer, these lists can be printed. So, a XQuery script was prepared that list all available areas of knowledge, and allow the visualization of all related entries.

Change reports

Another developed tool is the creation of Changes Reports. This XQuery scripts extract the entries that were edited or created in the last week, creating a list of these entries. This script is just a search looking into all documents last write access time.

Backup system

We use a quite original approach for backing up an eXist Database. eXist includes tools to export an entire collection either as a ZIP file, including all the collection documents, or exporting these documents to a folder in the disk (or a complete folder structure). To backup or dictionary we have a regular job, being executed every night, to export the collection to a folder. Then, this folder is committed to a GIT repository. This allows us to have a regular backup, but also a quite incremental system (using less disk space), and easy to replicate (at the moment we are pushing this repository into BitBucket[10]).

5 Conclusions and Future Work

In this article we presented our approach in the process of reverse engineering a dictionary published in PDF, in order to convert it to a fine-grained XML document. We discuss not just the process of reverse engineering (a task that is not new, although it was the first time we did it from a PDF document), but also why and how we store it in an XML aware database.

With the goal of making the dictionary available for editing and validation by different lexicographers, we split the dictionary into various XML documents, one for each dictionary entry. Also, as the process of searching these documents was not easy, a web application

[10] `http://bitbucket.org/`

was developed to search the document collection, and create links that allow the immediate access to each file using the oXygen XML Developer editor.

Having the dictionary being edited by lexicographers, a set of other tools required our attention. For those, we wrote small XQuery scripts that run on top of eXist and allow very different kinds of resources to be built.

Nevertheless, a set of other scripts need to be developed:

- Instead of creating HTML reports of each week work, we intend to create daily and weekly reports of editions, generated as XML documents, imported into another collection. This is a very interesting resource to have, in order to monitor the activity in the dictionary, and having a log on all performed changes.
- A paper dictionary can be born, developed, printed and die. But a dictionary to be available on the Internet needs to be dynamic, allowing the dictionary to evolve following the language and culture. Editing directly the XML file is versatile, but not easy to use. So, we expect to develop a user-friendly editor.
- Currently our web application is restricted to authenticated users. In the future an open interface needs to be available to end-users. Although the simple mechanisms to search for entries are already developed (although restricted), we think there is a couple of other interesting approaches. For example, the synonyms and antonyms annotation can be used to present the dictionary as a graph/WordNet-like structure.
- Although we will make the dictionary available on-line, we still want to be able to create other media, like eBooks or even printed books. For that we expect to create a set of exporting tools.

References

1 João Malaca Casteleiro, editor. *Dicionário da Língua Portuguesa Contemporânea*. Academia das Ciências de Lisboa, Verbo, 2001.

2 Micah Dubinko. *XForms Essentials*. O'Reilly Media, Inc., August 2003.

3 Xavier Gómez Guinovart and Alberto Simões. Retreading Dictionaries for the 21st Century. In José Paulo Leal, Ricardo Rocha, and Alberto Simões, editors, *2nd Symposium on Languages, Applications and Technologies*, volume 29 of *OpenAccess Series in Informatics (OASIcs)*, pages 115–126, Dagstuhl, Germany, 2013. doi:10.4230/OASIcs.SLATE.2013.115.

4 Wolfgang Meier. exist: An open source native xml database. In Akmal B. Chaudhri, Mario Jeckle, Erhard Rahm, and Rainer Unland, editors, *Web, Web-Services, and Database Systems: NODe 2002 Web- and Database-Related Workshops Erfurt, Germany, October 7–10, 2002 Revised Papers*, pages 169–183. Springer, Berlin, Heidelberg, 2003. doi:10.1007/3-540-36560-5_13.

5 Alberto Simões and José João Almeida. Processing XML: a rewriting system approach. In Alberto Simões, Daniela da Cruz, and José Carlos Ramalho, editors, *XATA 2010 – 8ª Conferência Nacional em XML, Aplicações e Tecnologias Associadas*, pages 27–38, 2010.

6 Alberto Simões, Álvaro Iriarte, and José João Almeida. Dicionário-Aberto: Construção semiautomática de uma funcionalidade codificadora. In Alain Lemaréchal, Peter Koch, and Pierre Swiggers, editors, *Actes du XXVIIe Congrès international de linguistique et de philologie romanes*, Nancy, 15-20 july 2013 2014. ALTIF. Section 16 : Projets en cours; ressources et outils nouveaux.

7 Edward Vanhoutte. An Introduction to the TEI and the TEI Consortium. *Literary and Linguistic Computing*, 19(1):9–16, 2004. doi:10.1093/llc/19.1.9.

Automata Serialization for Manipulation and Drawing*

Miguel Ferreira[1], Nelma Moreira[2], and Rogério Reis[3]

1 **CMUP & DCC, Faculdade de Ciências da Universidade do Porto, Porto, Portugal**
`miguelferreira108@gmail.com`
2 **CMUP & DCC, Faculdade de Ciências da Universidade do Porto, Porto, Portugal**
`nam@dcc.fc.up.pt`
3 **CMUP & DCC, Faculdade de Ciências da Universidade do Porto, Porto, Portugal**
`rvr@dcc.fc.up.pt`

Abstract

GUItar is a GPL-licensed, cross-platform, graphical user interface for automata drawing and manipulation, written in C++ and Qt5. This tool offers support for styling, automatic layouts, several format exports and interface with any foreign finite automata manipulation library that can parse the serialized XML or JSON produced. In this paper we describe a new redesign of the GUItar framework and specially the method used to interface GUItar with automata manipulation libraries.

1998 ACM Subject Classification D.2.2 State diagrams

Keywords and phrases automata, serialization, visualization

Digital Object Identifier 10.4230/OASIcs.SLATE.2016.15

1 Introduction

Software environments for symbolic manipulation of formal languages and models of computation are widely recognized as important tools for theoretical and practical research, as well as pedagogical tools for teaching automata theory and formal languages. Examples include Grail+ [18, 12], OpenFST [13], FAdo [6, 1] and Vaucanson [11, 10]. The visualisation and interactive drawing of the diagrams of the computational models is also an important component, but few tools are available. Namely, JFLAP [15, 14] is mainly used for pedagogical purposes (and includes its own symbolic manipulator) and other alternatives include the use of generic graph visualization tools such as Graphviz [17].

The GUItar project aims to develop an extensible graphical environment for several combinatorial objects and models of computation, such as finite automata, pushdown automata, transducers, Turing machines, etc. Its functionalities include visualisation and interactive editing, i.e. automatic and assisted diagram drawing; and export/import filters that allow the interaction with several symbolic manipulators. Comparing with generic graph visualization tools several requirements are distinct and are analised in the following.

* This work was partially supported by CMUP (UID/MAT/00144/2013), which is funded by FCT (Portugal) with national (MEC) and european structural funds through the programs FEDER, under the partnership agreement PT2020.

© Miguel Ferreira, Nelma Moreira, and Rogério Reis;
licensed under Creative Commons License CC-BY

Automatic graph drawing has been a very active research area and several (mainly) commercial software packages are now available for general and specific applications (data base design, information systems, bioinformatics, social networks,etc). In contrast, automata diagrams (alike labelled multi-digraphs) require additional aesthetics and graphical constraints: left-to-right reading, initial states on the left and final states on the right, edge shapes and label placements, etc. Another important issue is the visualisation of only some parts of a larger automata.

For the interactive editing it should define constraints that correspond to boolean functions of manipulators. For instance, if we are editing a deterministic finite automaton (DFA) no multiple transitions with the same label should be allowed; or a state can only be deleted if the resulting recognised language is the same.

For the interaction with the different symbolic manipulators (filters), it should be allowed the dynamic definition of actions that can be invoked, as well as conversions between the objects of the graphical framework and ones of the manipulators.

A first version of the software tool GUItar, was developed together with the FAdo system [1, 2, 3]. That version was implemented in wxPython [16] and included the visualisation and interactive editing for various types of automata. FAdo is mainly implemented in Python and currently includes most standard operations for the manipulation of regular languages and regular transductions, as well as several uniform random generators for these objects.

In this paper we present a new redesign of the GUItar framework that allows the interaction with several automata manipulators. In Section 2 we describe the main GUItar features within its interface and the several algorithms we use to layout the manipulated automata. The communication process between a library and GUItar is shown in Section 3. For the communication between these two layers to be possible, we needed to serialize the automaton. This is accomplished by the XML/JSON grammar presented in Section 4, along with the description of style manipulation of the automaton and examples of conversion to portable export formats. Section 5 concludes with some future work.

2 Graphical Interface for Automata Manipulation

The GUItar [8] is a graphical interface for automata drawing and manipulation, written in C++ and Qt5 [5]. The program is licensed with the GNU General Public License version 3.

It includes functionalities such as:

- Interactive "point and click" creation of automata;
- XML and JSON description of the automaton structure, styling and geometry of the states and transitions;
- Socket communication for obtaining and drawing automata represented on the GUItar XML or JSON language;
- Layout algorithms for state positioning;
- Embedded ipython shell for real time command line interaction;
- Exporting for several formats: PDF, PGF/Tikz, GraphViz (dot).

Its main window is composed of a QGraphicsScene widget used as a canvas, a terminal for interfacing with the scene automaton through command lines and selection buttons: insert state, insert transition and select items.

Using the same mechanism for automata manipulation through serialization described in Figure 1, GUItar supports file saving and opening, restoring all automaton style and geometry properties.

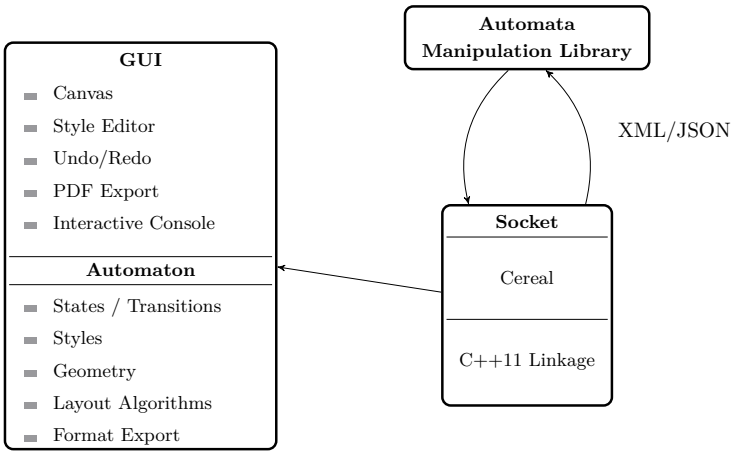


Figure 1 GUItar organization.

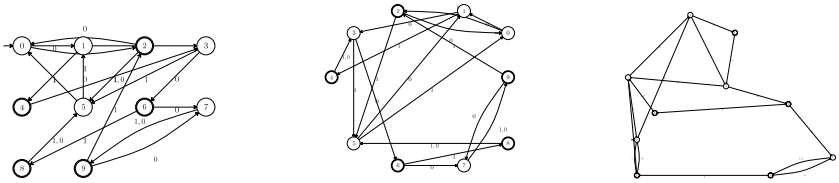


Figure 2 GUItar square, circle and spring layout of an automaton.

Comparing with the previous version of GUItar, the whole interaction and communication use a new paradigm (including the embedded shell), consisting on the complete separation of the symbolic manipulation program and the display program. Additional layouts and export formats where added.

2.1 Layouts

GUItar includes several implementations of graph layout algorithms, such as:

- Square - states are distributed in a $\lceil\sqrt{n}\rceil \times \lceil\sqrt{n}\rceil$ virtual grid where n is the number of states.
- Circle - we calculate a circle radius as a function of the number of states and then each state s_i gets positioned at $radius \times (\cos(\Theta \times i), \sin(\Theta \times i))$ where Θ is $\frac{2\pi}{n}$ and n is the number of states.
- Two circle - based on the JFLAP implementation, we separate the automaton into two parts: the inner circle, which includes all states with degree greater than 2, and the outer circle which includes the remaining states.
- Barycenter [4] - an iterative algorithm that given a fixed set of states, the remaining states move towards the barycenter proportionally to its degree.
- Spring energy - another iterative algorithm that simulates a force system where states repel each other and transitions attract their states, working as springs. This algorithm can run a user-defined number of iterations or until the particle system's kinetic energy is below a certain threshold.

An example of some of these layouts is presented in Figure 2.

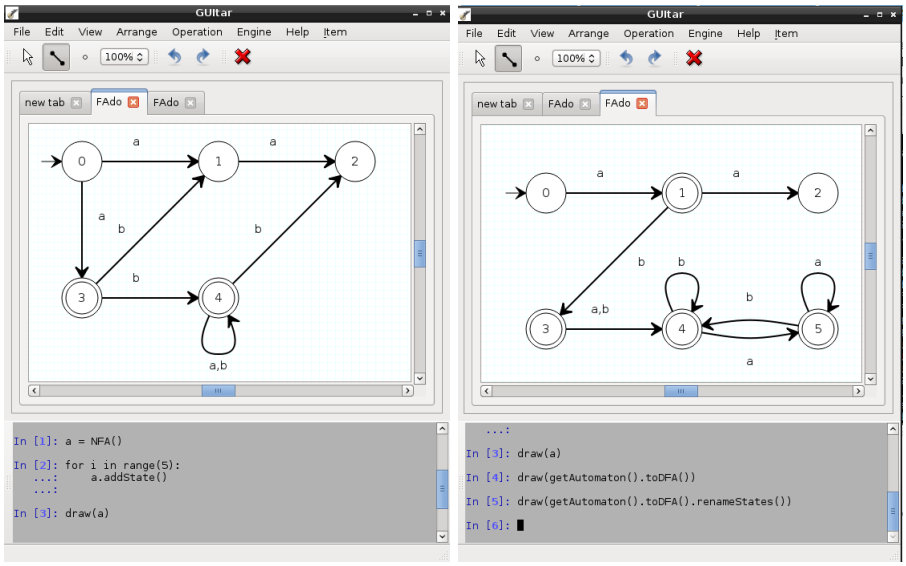


Figure 3 Example manipulation of an automaton using GUItar and FAdo.

3 Interfacing with Automata Manipulation Libraries

Communication with the GUItar is made through local sockets. For the sake of simplicity on the automata manipulation library side, we added support for the communication through the GUItar binary itself, as this is a single instance application.

There is also support for communication with a command line embedded inside the GUItar. This terminal has an ipython shell with the FAdo library loaded for a more natural way of handling the visible automaton.

The action for obtaining the currently seen automaton consists of sending the string "GET" through the GUItar socket and expecting a JSON/XML response. It is possible to draw using an analogous approach with the string "PUT" followed by the JSON/XML representation of the automaton, expecting empty answer in case of success and validation of the input against a schema.

This simplistic approach allowed us to quickly interface with known libraries such as FAdo and Vaucanson, by writing library-side methods for interpreting the JSON/XML GUItar format. An example using FAdo is shown in Figure 3.

4 Automata Serialization and its XML Grammar

GUItar uses the cereal [7] serialization library to produce JSON and XML representations of its automata to be understood by manipulation libraries. We chose this approach instead of other alternatives based on other serialization methods not only for the sake of code simplicity and parsing efficiency, but to allow any automata manipulation library programmer to easily interact with GUItar, be it through JSON, XML or even binary form.

The C++11 library cereal allowed us to transform our structural representation into a XML/JSON language that later can be parsed and validated using a schema. A fragment of the language schema is shown in Listing 2.

The structure is completely passed back and forth between the GUItar and the library through sockets and it is the library responsibility to maintain any state, if necessary, within

Listing 1 Partial JSON serialization example for the GUItar automata class.

```
{
 "automaton": {
  "title": "Automaton 1",
  "type": "",
  "states": [
   {
    "name": "1461439542",
    "label": "s2",
    "output": "",
    "initial": false,
    "final": true
   },
  ],
  "trans": [
   {
    "name": "14614395790",
    "orig_name": "1461439542",
    "dest_name": "1461439542",
    "label": "a",
    "weight": "",
    "compounds": []
   }
  ],
 "alphabet": ["a"]
 }
}
```

Listing 2 XML Relax NG Compact grammar for the GUItar automata class.

```
start = element cereal {
 element automaton {
  element title { text },
  element type { text },
  element states {
   element state {
    element name { xsd:integer },
    element label { text },
    element output { text },
    element initial { xsd:boolean },
    element final { xsd:boolean }
   }*
  },
  element trans {
   element transition {
    element name { xsd:integer },
    element orig_name { xsd:integer },
    element dest_name { xsd:integer },
    element label { text },
    element weight { text },
    element compounds {
     element compound {
      element key { text },
      element value {text}
     }*
    }
   }*
  },
  element alphabet {
    element symbol {text}*
  }
}
```

the extra branch of the grammar. This branch is guaranteed by GUItar to be returned exactly as it was sent.

For different automaton types, we allow special transitions with compounds, in which a format string is defined on the label field in the form of $\$compound_1 \cdots \$compound_n$, and the values of the compounds are stored as key-value pairs on its branch.

4.1 Example of a XML Automaton with Styles

On this implementation of GUItar, we include support for styling inside the JSON/XML grammar. Each drawable object can have style properties in GUItar defined for each object class. There can be style templates defined inside the GUI besides the default one.

The correspondence between style XML, automaton visualization and style form can be seen in Listing 3 and Figure 4.

4.2 Exporting to Visualization Formats

Our automata objects in GUItar can be exported to several known formats. At the moment, it is possible to export to Vaucanson-G [9] and PGF/Tikz maintaining some level of similarity between geometry and style. We can also export the automaton with some level of styling to the GraphViz layout program in dot language.

Listing 3 An example of XML GUItar.

```
<?xml version="1.0" encoding="utf-8"?>
<cereal>
 <automaton>
  <title>FAdo</title>
   ...
  <states size="dynamic"> ... </states>
  <trans size="dynamic"> ... </trans>
 </automaton>
 ...
 <style>
  <states size="dynamic">
   <value0>
    <key>1458486283455</key>
    <value>
     <shape>0</shape>
     <lineWidth>5</lineWidth>
     <lineStyle>0</lineStyle>
     <lineRgb>255</lineRgb>
     <fillStyle>0</fillStyle>
     <fillRgb>16777215</fillRgb>
    </value>
   </value0>
  </states>
 </style>
</automaton>
```

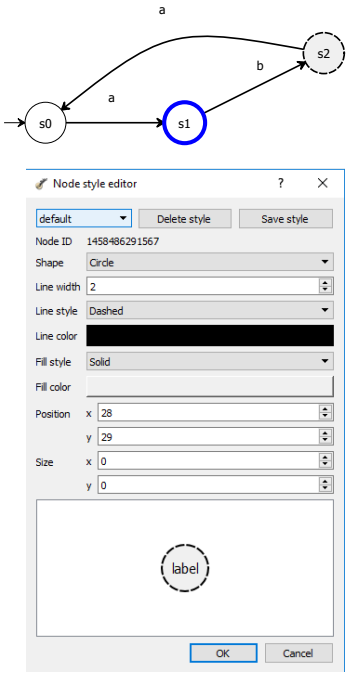


Figure 4 Automaton state styling.

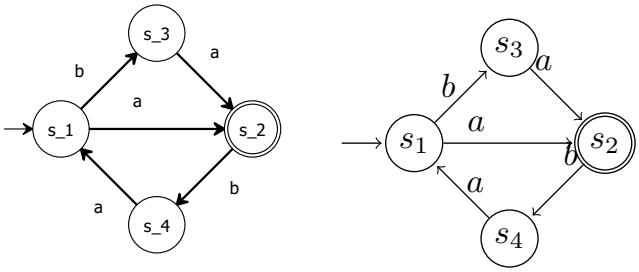


Figure 5 GUItar native export and PGF/Tikz comparison.

It is possible to export to a PDF file, maintaining an exact layout, the visible drawing in GUItar through Qt framework methods.

5 Conclusion

In this paper we presented a new implementation of the GUItar program for visually manipulating automata and interacting with libraries. This new version, although using some ideas of the previous one, consisted in a new redesign of most of the features and addition of new ones. Major improvement was the possibility of communicate with several automata symbolic manipulators.

This project is still under continuous development. The simplistic socket interface combined with the serialization procedure provides an almost transparent communication with automata manipulation libraries.

There are still features to develop and add to this project, such as GraphML export format, more layout algorithms focused on automata drawing and constrained edition driven by manipulators functions.

Acknowledgements. We want to thank to the anonymous reviewers for their comments that helped to improve this paper.

References

1 André Almeida, Marco Almeida, José Alves, Nelma Moreira, and Rogério Reis. FAdo and GUItar: tools for automata manipulation and visualization. In Sebastian Maneth, editor, *14th International Conference on Implementation and Application of Automata, CIAA 2009. Proceedings*, volume 5642, pages 65–74, Sidney, July 2009. Springer.

2 André Almeida, Nelma Moreira, and Rogério Reis. GUItar and FAgoo: Graphical interface for automata visualization, editing, and interaction. In Luís S. Barbosa and Miguel P. Correia, editors, *Inforum, Simpósio de Informática*, pages 317–328, Braga,Portugal, 9-10 Setembro 2010.

3 José Alves, Nelma Moreira, and Rogério Reis. XML description for automata manipulations. In Alberto Simões, Daniela Cruz, and José Carlos Ramalho, editors, *Actas XATA 2010, XML: aplicações e tecnologias associadas*, pages 77–88, ESEIG, Vila do Conde, 2010.

4 Giuseppe Di Battista. *Graph drawing: algorithms for the visualization of graphs*. Pretice Hall, 1999.

5 The Qt Company. Qt, Access date:1.12.2015. URL: `http://www.qt.io`.

6 Project FAdo. FAdo: tools for formal languages manipulation, Access date:1.11.2015. URL: `http://fado.dcc.fc.up.pt/`.

7 Shane Grant and Randolph Voorhies. cereal – A C++11 library for serialization, Access date:4.14.2016. URL: `http://uscilab.github.io/cereal/`.

8 Project GUItar. GUItar, Access date:1.06.2016. URL: `http://guitar.dcc.fc.up.pt/`.

9 S. Lombardy and J. Sakarovitch. Vaucanson-G, Access date:1.12.2015. URL: `http://igm.univ-mlv.fr/~lombardy/Vaucanson-G/`.

10 Sylvain Lombardy, Yann Régis-Gianas, and Jacques Sakarovitch. Introducing Vaucanson. *Theor. Comput. Sci.*, 328(1-2):77–96, 2004. `doi:http://dx.doi.org/10.1016/j.tcs.2004.07.007`.

11 Sylvain Lombardy and Jacques Sakarovitch. Vaucanson, Access date:1.12.2015. URL: `http://vaucanson-project.org`.

12 Darrell Raymond and Derick Wood. Grail: A C++ Library for automata and expressions. *J. Symb. Comp.*, 17(4):341–350, 1994.

13 Michael Riley. OpenFst, Access date:1.3.2016. URL: `http://www.openfst.org`.

14 Susan Rodger and Thomas Finley. *JFLAP: An Interactive Formal Languages and Automata Package*. Jones and Bartlett Publishers, 2006.

15 Susan H. Rodger. JFLAP, Access date:1.12.2015. URL: `http://www.jflap.org`.

16 Julian Smart, Robert Roebling, Vadim Zeitlin, and Robin Dunn. *wxWidgets 2.6.3: A portable C++ and Python GUI toolkit*, 2006. URL: `http://wxpython.org`.

17 Graph Visualization Software. Graphviz, Access date:1.12.2015. URL: `http://graphviz.org/`.

18 Sheng Yu and Cezar Campeanu. Grail+, Access date:1.3.2016. URL: `http://www.csit.upei.ca/~ccampeanu/Grail`.