Frontiers in Artificial Intelligence and Applications

CONTROLLED NATURAL LANGUAGE

Proceedings of the Sixth International Workshop, CNL 2018, Maynooth, Co. Kildare, Ireland, August 27-28, 2018

Edited by
Brian Davis
C. Maria Keet
Adam Wyner

IOS Press

CONTROLLED NATURAL LANGUAGE

Controlled natural languages (CNLs) are based on natural language and apply restrictions on vocabulary, grammar, and/or semantics. They fall broadly into 3 groups. Some are designed to improve communication for non-native speakers of the respective natural language; in others, the restrictions are to facilitate the use of computers to analyze texts, for example, to improve computer-aided translation; and a third group of CNLs are designed to enable reliable automated reasoning and formal knowledge representation from seemingly natural texts.

This book presents the 11 papers, selected from 14 submitted, and delivered at the sixth in the series of workshops on Controlled Natural Language, (CNL 2018), held in Maynooth, Ireland, in August 2018. The papers cover a full spectrum of controlled natural languages, ranging from human oriented to machine-processable controlled languages and from more theoretical results to interfaces, reasoning engines, and the real-life application of CNLs.

The book will be of interest to all those working with controlled natural language, whatever their approach.

ISBN 978-1-61499-903-4 (print)
ISBN 978-1-61499-904-1 (online)
ISSN 0922-6389 (print)
ISSN 1879-8314 (online)

CONTROLLED NATURAL LANGUAGE

Frontiers in Artificial Intelligence and Applications

The book series Frontiers in Artificial Intelligence and Applications (FAIA) covers all aspects of theoretical and applied Artificial Intelligence research in the form of monographs, selected doctoral dissertations, handbooks and proceedings volumes. The FAIA series contains several sub-series, including 'Information Modelling and Knowledge Bases' and 'Knowledge-Based Intelligent Engineering Systems'. It also includes the biennial European Conference on Artificial Intelligence (ECAI) proceedings volumes, and other EurAI (European Association for Artificial Intelligence, formerly ECCAI) sponsored publications. The series has become a highly visible platform for the publication and dissemination of original research in this field. Volumes are selected for inclusion by an international editorial board of well-known scholars in the field of AI. All contributions to the volumes in the series have been peer reviewed.
The FAIA series is indexed in ACM Digital Library; DBLP; EI Compendex; Google Scholar; Scopus; Web of Science: Conference Proceedings Citation Index – Science (CPCI-S) and Book Citation Index – Science (BKCI-S); Zentralblatt MATH.

Volume 304

Recently published in this series

ISSN 0922-6389 (print)
ISSN 1879-8314 (online)

Controlled Natural Language

Proceedings of the Sixth International Workshop, CNL 2018,
Maynooth, Co. Kildare, Ireland, August 27–28, 2018

Edited by

Brian Davis

Maynooth University, Ireland

C. Maria Keet

University of Cape Town, South Africa

and

Adam Wyner

Swansea University, United Kingdom

IOS Press

Amsterdam • Berlin • Washington, DC

ISBN 978-1-61499-903-4 (print)
ISBN 978-1-61499-904-1 (online)
Library of Congress Control Number: 2018953109

Publisher
IOS Press BV
Nieuwe Hemweg 6B
1013 BG Amsterdam
Netherlands
fax: +31 20 687 0019
e-mail: order@iospress.nl

For book sales in the USA and Canada:
IOS Press, Inc.
6751 Tepper Drive
Clifton, VA 20124
USA
Tel.: +1 703 830 6300
Fax: +1 703 830 2300
sales@iospress.com

PRINTED IN THE NETHERLANDS

Preface

CNL 2018 is the sixth in the series of workshops on Controlled Natural Language (CNL), which was first organised in 2009. The 2018 edition was organised in Co. Kildare, Ireland, on 27 and 28 August.

As with previous editions of the workshop, this year's papers cover the wide spectrum of the area of Controlled Natural Languages, ranging from human oriented to machine processable controlled languages, and from more theoretical results to interfaces, reasoning engines, and real-life applications of CNLs.

This year we invited both long and short papers to be submitted to the workshop, and we received a total of 14 submissions. All papers were peer-reviewed by at least three, and in most cases four, members of the workshop's Program Committee. Based on the reviews, 11 papers were accepted, out of the 14. As per usual the outcomes remain truly international, where PC members represented organisations from 13 countries and authors' affiliations of accepted papers came from 10 countries on 3 continents.

In addition to the presentation of the accepted papers, the programme includes 3 invited speakers: Claire Gardent (Loria, France), Albert Gatt (University of Malta, Malta) and Teresa Lynn (Dublin City University, Ireland) and a late-breaking results posters & demos session. We would like to thank the Programme Committee for their reviews and feedback, the authors for their contributions and the invited speakers for accepting our invitation to present their work in the workshop. Furthermore, we would also like to thank Maynooth University for hosting the workshop and the workshop sponsors Science Foundation Ireland, Maynooth University and Digital Grammars Gothenburg AB.

Brian Davis
C. Maria Keet
Adam Wyner

July 13, 2018
CNL'18, Maynooth, Co Kildare, Ireland

This page intentionally left blank

Program Committee

Krasimir Angelov	Digital Grammars Gothenburg AB, Sweden
Mihael Arcan	Insight Centre for Data Analytics, NUI Galway, Ireland
John Camilleri	Digital Grammars Gothenburg AB, Gothenburg, Sweden
Brian Davis	Maynooth University, Ireland
Ronald Denaux	Expert System, Madrid, Spain
Ramona Enache	Microsoft Trondheim, Norway
Esra Erdem	Sabanci University, Istanbul, Turkey
Sebastien Ferre	IRISA, Université Rennes 1, France
Norbert Fuchs	University of Zurich, Swizerland
Normunds Grūzītis	University of Latvia, Latvia
Siegfried Handschuh	University of Passau, Germany
Kaarel Kaljurand	Nuance Communications, Austria
Maria Keet	University of Cape Town, South Africa
Peter Koepke	University of Bonn, Germany
Tobias Kuhn	Vrije Universiteit Amsterdam, the Netherlands
John P. McCrae	Insight Centre for Data Analytics, NUI Galway, Ireland
Sharon O'Brien	ADAPT Centre, Dublin City University, Ireland
Gordon Pace	University of Malta, Malta
Laurette Pretorius	University of South Africa, South Africa
Mike Rosner	University of Malta, Malta
Rolf Schwitter	Macquarie University, Sydney, Australia
Silvie Spreeuwenberg	LibRT, Amsterdam, the Netherlands
Irina Temnikova	Qatar Computing Research Institute, Qatar
Camilo Thorne	University of Stuttgart
Adam Wyner	Swansea University, United Kingdom

This page intentionally left blank

Contents

This page intentionally left blank

Controlled Natural Language
B. Davis et al. (Eds.)

doi:10.3233/978-1-61499-904-1-1

Editing with Search and Exploration for Controlled Languages

Krasimir ANGELOV [a], and Michal Boleslav MĚCHURA [b]
[a] *University of Gothenburg and Digital Grammars AB, Sweden*
[b] *Masaryk University, Czech Republic*

Abstract. We present an editor for controlled languages which is a combination of a syntax editor and a predictive editor. It shows a bird's-eye view which lets the user to explore what is possible in the language. Still, unlike the syntax editors the user is not expected to understand the underlying abstract syntax or ontology behind the language. It also lets the user to enter arbitrary phrases from which the editor finds the phrases which are the closest match.

Keywords. syntax editor, predictive editor, closest match

1. Introduction

Controlled languages have the virtue of being formal languages. Their clearly defined syntax and semantics makes them attractive in applications where the reliability of the computer-human or human-human interaction is a must. However, this virtue comes with a cost. A controlled language is usually a subset of an existing natural language and the end-user must learn how to stay within its boundaries.

There are a number of editors [2,3,5,7,8], which help the user to work with a controlled language. All of them, however, have important limitations, which we will point out shortly. The editors are of roughly two types. The first is the so called syntax editors which let the user manipulate a logical structure, while the actual text is just a by-product. For example in [3] and [7] the user edits the abstract syntax of the text. Similarly in [8] the user edits ontological data, and the software renders it in one or more natural languages. The second kind is called predictive editors [2,5], which opt to work directly on the text level and guide the user by showing the set of possible continuations.

The two types of editors have different pros and cons. The syntax editors are appropriate when the intermediate logical structure actually makes sense for the end user. For example in [3] the abstract syntax is also a type-theoretical proof object which is of primary interest for the logician. In [8] again the intermediate representation is an ontology which is what the knowledge engineer needs. By using an intermediate representation, these kinds of editors let the user control parameters that may not be visible in the natural language. For example in many languages it is not enough to select pronouns like *I* and *you*, since the rendering will be different depending on whether we mean a male or a female subject. However, if the current target language is a language like English, then there will be absolutely no difference. The difference will only show up in another

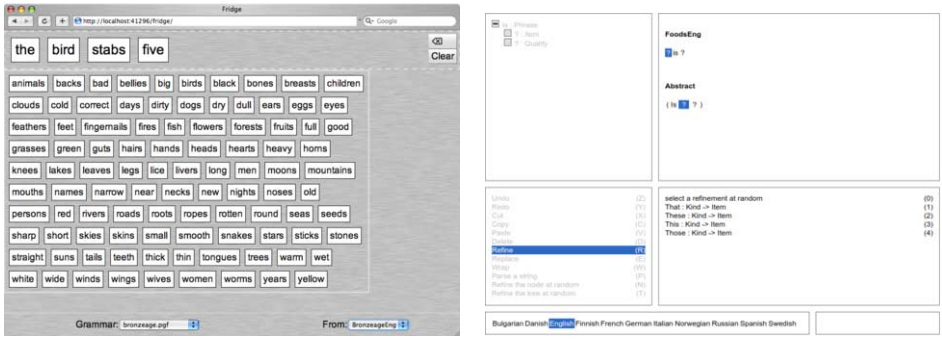


Figure 1. Screenshot of a predictive editor [2] and a syntax editor [7].

language which the user may not understand. On the level of the internal representation, however, they should be distinguished which will force the user to make the decision.

The disadvantage of the syntax editors is that they are not suitable when the end-user is not expected to be familiar with the internal representation. A further limitation is that apart from making decisions about the content of the text, the user might have to make linguistic decisions about the surface structure. For example the user might have to choose whether to use a relative clause or whether simple or continuous tense is needed. Naturally the users are capable of using language but may not be familiar with the linguistic terminology for describing the language.

The disadvantage of the syntax editors is overcomed by using predictive editors. There the internal representation is hidden and the user sees only the target text. Instead of typing text directly, however, the user is asked to select words predicted by the grammar. By only clicking words, it is guaranteed that the final sentence will be in scope.

The problem with the predictive editors is still that the user cannot see how the phrase that he/she wants to say can be expressed. Figure 1 shows the predictive editor from [2]. Obviously it is clear what are the immediate continuations of the sentence, but the overall picture of what is possible is still missing. The predictive editor offers only the perspective of an ant. It shows a set of relevant words, but it does not show what kinds of sentences are possible. Using a predictive editor feels like a maze where the user needs to find his way through by seeing only one word at a time. It would have been much easier if there was also a bird's-eye view.

The absence of a bird level is a problem even for syntax editors. The difficulty there is mainly in the beginning where the user starts with an empty intermediate representation and then he/she has to find the way through by finding the appropriate refinements. An example is in Figure 1 where in the current state it is obvious that the target sentence will use the copula as the main verb but what the sentence might be about is a unknown.

We propose a new interface which is a combination of a syntax editor and a predictive editor. The editor works directly with the controlled language, so the intermediate representation is never shown. Still just like with the syntax editor, the refinements are done by using a system of menus. The menus are rendered in the controlled language so again there is no need for the user to see the underlying structure.

In contrast to to the syntax editors, the user never starts from an empty state. Instead there is a list of example sentences which illustrates what is possible (the bird's-eye view). After selecting an example, the user can customize it by using the menus.

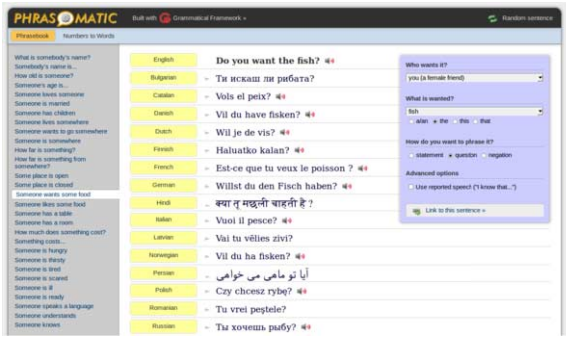


Figure 2. Screenshot of Phrasomatic.

Going through the list of examples to find the best starting point might still be tedious. Our alternative is to let the user just type/speak whatever he/she wants to say. The editor then searches through the grammar of the language to find the phrases that are the best match for the input. Once a candidate is selected it could still be edited through the menus.

2. Origins: Phrasomatic and Parlira

An early prototype of the idea was Phrasomatic[1], created in 2013. Phrasomatic is a website which generates sentences in 19 languages from the Phrasebook grammar [11]. The user starts by selecting a semantically vacuous sentence template with placeholders, such as 'someone wants some food', and then continues to gradually refine the sentence by interacting with a user interface (making selections from menus): who wants the food, what food, is it a statement or a question, etc. From the user's input, the tool constructs an abstract syntax tree (AST), a language-independent representation of the user's intended meaning. The abstract syntax tree is then sent to the GF [9] web service for linearization into all available languages.

Phrasomatic was a prototype rather than a final product because building the user interface involved a large amount of hand-coding in HTML, CSS and JavaScript. Each element where the user makes a selection, for example selecting a person (the 'who' part of a sentence) or selecting a food type (the 'what food' part of a sentence), is a small hand-coded HTML page embedded within the application as an IFRAME. Each such IFRAME provides a rudimentary JavaScript interface which can be called by the application to "harvest" the current state of the user's choices and to return (as a string) a fragment of an abstract syntax tree. The application then assembles the fragments into an abstract syntax tree for the entire sentence.

There are, essentially, three types of elements in the Phrasomatic user interface where the user can make a selection:

1. Elements which supply values for placeholders, for example selecting a person or selecting a food type. The choices the user makes in these elements are either simple ones (for example, selecting 'you' as a person), or complex ones where

[1] http://www.phrasomatic.net/

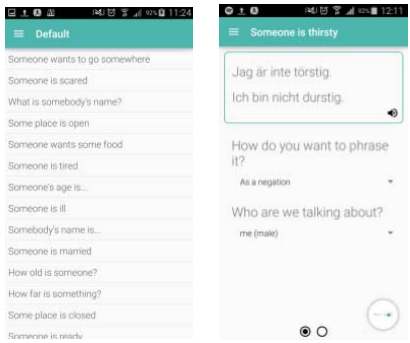


Figure 3. Screenshot of Parlira.

making a choice implies having to make further choices (for example, selecting 'someone's wife' and then having to select 'whose' wife).

2. Elements which control the grammatical 'shape' of the sentence but do not go into any particular placeholder, for example selecting whether the sentence should be a statement, a question, or a negative statement.
3. Elements which 'wrap' the sentence inside a larger sentence, for example turning it into reported speech (going from 'someone wants some food' to 'someone knows that someone wants some food'). Again, some of the choices made in such an element are complex because making one choice implies having to make further choices (for example, having to choose 'who' knows).

The logic of choices, how they depend on one another, and how they come together into an abstract syntax tree for the entire sentence, has been hard-coded into the user interface. This means that the user interface is specific to the Phrasebook application grammar and cannot easily be re-used for other applications. The obvious next step would be to devise a formalism or scheme where we could generate the user interface from grammar itself or from some intermediate representation.

Based on the same ideas as Phrasomatic, the Parlira project [4] created an Android app. Both Phrasomatic and Parlira use the Phrasebook grammar [11] and both suffer from being tied to it. The initial goal of the Parlira project was to relax that limitation, but that was not fully realized. In this paper we present a fully customizable solution which finally overcame that limitation and has been tested with two grammars, [11] and [10] (the later is an app developed by Digital Grammars AB for communication between patient and ambulance personnel in several of the immigrant languages of Sweden).

Both Phrasomatic and Parlira did, however, play a role as experimental prototypes for multilingual point-and-click authoring with GF. The advantage of such a tool over machine translation is obvious: the application does not have to 'guess' (ie. parse) what the user may have meant because the input is unambiguous, while the output is guaranteed to be correct (both in content and linguistically), making it reliable for production-oriented situations where the user does not speak the target language. The disadvantage is obvious too: the coverage is tiny and limited to one domain.

The fundamental challenge of tools like Phrasomatic and Parlira, however, is ergonomical: the user experience is slow-moving, involves 'too much clicking' (a comment from an actual user) and can come across as unintuitive because most people are not used to expressing their intended meaning in anything other than natural language.

This limitation can perhaps be overcome by combining point-and-click authoring with natural-language search: the user's natural-language input would be used for an initial search of candidate sentences, while conceptual authoring would come in later for refinement and disambiguation.

3. Editing

In this section we will tackle the key issue explained in the previous section, i.e. how to make the editor reconfigurable. We assume that the underlying representation of a sentence is a tree. This fits perfectly with the Grammatical Framework (GF [9]), but it can also fit other frameworks. For example the sentence:

I am an ambulance nurse

in [10] has the representation:

```
statementPhrase
   (professionStatement (mainPerson iMainPerson)
                        ambulanceNurseProfession)
```

The key components of the phrase are obviously in the representation, i.e. the profession `ambulanceNurseProfession` and the subject `iMainPerson`. What the presentation does not say is how exactly these phrases are rendered. In GF this is defined by the grammar and can be different for each language while sharing the same representation. For example, from the same tree the Spanish grammar produces:

Soy enfermera de ambulancia

Apart from the essential content of the sentence, the representation also contains pieces which are irrelevant for the semantics but are helpful, for instance, in making the grammar better structured. For example the functions `statementPhrase` and `mainPerson` are only coercions from one category to another. In a typical syntax editor the user might have to insert those coercions manually while working directly with the tree. These, however, are just artefacts of the grammar engineering and the user should not be bothered to make decisions whose purpose might not even be obvious for him/her.

Formally, we expect that there is a grammar which converts abstract trees into natural language. The syntax for trees is defined as:

$$\begin{array}{ll} \text{Expression } e & ::= f\ e_1\ e_2\ \ldots\ e_k, \qquad k \leq |f| \\ \text{Function} & f, g, h \end{array}$$

In other words, there is a set of functions where every function f takes $|f|$ number of arguments i.e. other trees. In order for a tree to be rendered in a natural language, every function in the tree must be fully applied. However, in the intermediate steps when we build a tree, it is helpful if we allow partial function applications. Because of that in the definition above we wrote $k \leq |f|$.

In order to describe how the menus in the editor work we also need the notion of a choice configurations. The formal definition is:

$$\begin{aligned}\text{Choice } c ::= {} & \mathit{option}(\iota,k)\ \ c_1\ c_2\ \ldots\ c_n \\ \mid {} & \mathit{function}(\iota,e)\ c_1\ c_2\ \ldots\ c_{|e|} \\ \mid {} & \mathit{call}(\iota)\ c_1\ c_2\ \ldots\ c_s \\ \mid {} & \mathit{arg}(k)\end{aligned}$$

The first configuration $\mathit{option}(\iota,k)$ specifies that the user can choose one of the options from c_1 to c_n and the default option is k. The second configuration $\mathit{function}(\iota,e)$ says that a tree is built by applying the tree e to the result of the choices from c_1 to $c_{|e|}$. In both cases there is also the parameter ι which must be an unique id which is different for every configuration instance. An example is:

$$\begin{aligned}&\mathit{function}(\iota_1,\mathit{mainPerson})\ (\mathit{option}(\iota_2,0)\ \ \mathit{function}(\iota_3,\mathit{youMainPerson}) \\ &\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\ \ \mathit{function}(\iota_4,\mathit{youFemMainPerson}))\end{aligned}$$

It says that the user can choose between two options which results in using either function *youMainPerson* or function *youFemMainPerson*. On top of that is applied function *mainPerson*. The final result is that one of these two trees will be produced:

mainPerson youMainPerson
mainPerson youFemMainPerson

By using this kind of choice configurations it is possible to describe the entire set of trees in the grammar. The place and the granularity of the user's choices is explicitly specified. In the example every option consists of only a single function but in general it is possible to insert a complex tree as a single unit. From the user's point of view every option will be rendered as an item in a menu. The text of the menu is generated by rendering the attached tree by using the grammar. Thanks to that the abstract representation is never visible. Sometimes the tree that is used for rendering the controlled language is not appropriate for use in the menus. For example both *youMainPerson* and *youFemMainPerson* will simply generate *you* in English. Instead we would like to see something like *you (man)* and *you (woman)*. For that purpose, the designer of the user interface can choose to attach "description trees" which are also rendered to natural language but in a more descriptive way than the trees that produce the controlled language.

Note also that coercion functions like *mainPerson* are automatically inserted without user's involvement.

The two other constructions $\mathit{call}(\iota)$ and $\mathit{arg}(k)$ turn our configurations into a small functional language. This is handy since one and the same sequence of choices might be needed in different situations. $\mathit{call}(\iota)$ encodes a call into a choice with unique id ι which is applied to arguments c_1 to c_s. $\mathit{arg}(k)$ retrieves and uses an argument with index k.

The choice configurations represent the semantics of the XML file that we use in the actual implementation. The above example will look like:

```
<function id="you_main_person" name="mainPerson">
    <option>
        <function abs_desc="mainPersonItem youMainPerson"    name="youMainPerson"/>
        <function abs_desc="mainPersonItem youFemMainPerson" name="youFemMainPerson"/>
    </option>
</function>
```

The pseudo code for the configuration interpreter is in Figure 4. It is a function $E[\![c|\alpha;\beta;\gamma]\!]$ with four arguments:

$$
\begin{array}{lrl}
E[\![option(\iota,k)\ \ c_1\ c_2\ \ldots\ c_n\ |\ & \langle \iota\ ,l\rangle\alpha;\beta;\gamma\ ]\!] &= E[\![c_l|\alpha;\beta\langle\iota,l\rangle;\gamma]\!] \\
E[\![option(\iota,k)\ \ c_1\ c_2\ \ldots\ c_n\ |\ & \langle \iota',l\rangle\alpha;\beta;\gamma\ ]\!] &= E[\![c_k|\varepsilon;\beta\langle\iota,r\rangle;\gamma]\!], \qquad \iota\neq\iota' \\
E[\![option(\iota,k)\ \ c_1\ c_2\ \ldots\ c_n\ |\ & \varepsilon;\beta;\gamma\ ]\!] &= E[\![c_k|\varepsilon;\beta\langle\iota,r\rangle;\gamma]\!] \\
E[\![function(\iota,e)\ c_1\ c_2\ \ldots\ c_s\ |\ & \alpha;\beta;\gamma\ ]\!] &= (e\ e_1\ e_2\ \ldots\ e_s,\ \beta_s) \\
 & & \quad \text{where } (e_1,\ \beta_1) = E[\![c_1|\alpha;\beta\ ;\gamma]\!] \\
 & & \qquad\quad (e_2,\ \beta_2) = E[\![c_2|\alpha;\beta_1;\gamma]\!] \\
 & & \qquad\quad \vdots \\
 & & \qquad\quad (e_s,\ \beta_s) = E[\![c_s|\alpha;\beta_{s-1};\gamma]\!] \\
E[\![call(\iota)\ c_1\ c_2\ \ldots\ c_s & \ |\ \alpha;\beta;\gamma\ ]\!] &= E[\![c\ |\alpha;\beta;c_1\ c_2\ \ldots\ c_s\gamma]\!], \qquad id(c)=\iota \\
E[\![arg(k) & \ |\ \alpha;\beta;c_1\ c_2\ \ldots\ c_k\gamma\ ]\!] &= E[\![c_k\ |\alpha;\beta;c_1\ c_2\ \ldots\ c_k\gamma]\!]
\end{array}
$$

Figure 4. An interpreter for choice configurations

- c - a choice configuration
- α - a list of choice alternatives taken at the latest user interaction
- β - a list of choice alternatives to be made for the new user interaction
- γ - a stack for keeping track of the *call* arguments.

The arguments α and β are those that represent how the user interface looks and how it should be updated. At every single point the interface shows a list of options. What is currently selected in them is encoded in α. At this point the user can change the selection of any of the options which is recorded again in α. After that the interpreter is executed. A change in one place in α might require making other choices and showing new options. This results in computing a new sequence of options which is computed in β. Both α and β are sequences of pairs like $\langle\iota,l\rangle$, where ι is the unique id of an $option(\iota,k)$ configuration and l is the index of the selection currently taken. In the beginning $l=k$ but after that the user might change it, so we store the actual index in α.

The first rule of the interpreter says that if there is an $option(\iota,k)$ configuration and if α starts with $\langle\iota,l\rangle$ then we should proceed by processing c_l. The pair $\langle\iota,l\rangle$ is also copied to β to specify that this option should continue to be available in the user interface with the same selection.

The second rule is again about $option(\iota,k)$ but it handles the case where on top of α there is a different id ι' which does not match the current configuration. This happens when the user changes somewhere the current selection and this affects the set of consecutive options. In this case we just disregard the rest of α. For the current option we select choice c_k.

The third rule of the interpreter is similar and it handles the case where α is an empty sequence (ε). In that case we again make the default choice. When the user interface is first started it always starts with empty α. This lets the interpreter to choose the default version of the sentence to be produced. The user then makes changes through the menus.

The fourth rule builds the trees. It applies the tree e to the results computed from c_1 to c_s. Usually e is just a function name, but more generally it could be a tree like $f\ e_1\ e_2 \ldots e_p$. The constraint is that $p+s=|f|$. This guarantees that the tree can be linearized according to the semantics in GF [9].

The last two rules implement function calls. The rule for $call(\iota)$ finds the configuration with key ι and proceeds with that while pushing the arguments in the stack γ. Similarly the rule for $arg(k)$ retrieves the appropriate argument from the stack.

4. Search

As we discussed, roaming through the set of menus might still not be straightforward. For that purpose we also implemented search. The idea is to see the grammar as a finite description of a potentially infinite set of sentences. In information retrieval a set of sentences (documents) is called a corpus and the task of the retrieval is to find the sentence which matches best the search query using measures like cosine similarity. With cosine similarity the corpus sentence and the query are represented as vectors in a high-dimensional space and the cosine of the angle between them measures the similarity [6].

Despite that a grammar represents an infinite set of sentences the same information retrieval techniques can be applied. We just use the grammar as a finite proxy for the infinite set. This method is already available as a service in the GF Runtime [1]. For example with the grammar in [10], if the user searches for *pain* (Figure 5) the runtime finds the trees:

statementPhrase (*symptomStatement* ? *painSymptom*)
statementTimePhrase (*symptomStatement* ? *painSymptom*) ?
questionPhrase (*whetherQuestion* (*symptomStatement* ? *painSymptom*))

This are partial trees where the question mark is a placeholder for the missing bits. If the trees are refined they could represent phrases like:

```
You are in pain.
You are in pain today.
Are you in pain?
```

As we can see all the phrases contain the search query *pain*. These are also the phrases in the grammar which contain the least number of additional words that are not already in the query. The partial trees have holes for *you* and *today* because there is no evidence in the query for what kind of subject/adverb should be used. However, if the query is *she pain*, then runtime will automatically fill in the subject with *she*.

What the editor does is to take the trees computed by the runtime, and to match them with the choice configurations. The parts of the tree that are already known are used to find the correct path through the choice options. When a placeholder is encountered, we instead use the existing strategy to take the default option as described in the previous section. This gives us for instance *you* as the default subject. The user can then use the menus as usual to make changes.

5. User Feedback

The Ambulance app was tested twice by members of the ambulance service in Gothenburg. The first test was done early on by a representative of the management staff. At that point the app had the menus for editing but still did not have the search facility and instead a traditional predictive editing mode was available. There was also a parser which can be used in combination with keyboard or speech-to-text input.

The feedback was that as it is the app is difficult to use without some prior training. The reason is that finding the right phrase through the menus takes time unless if the

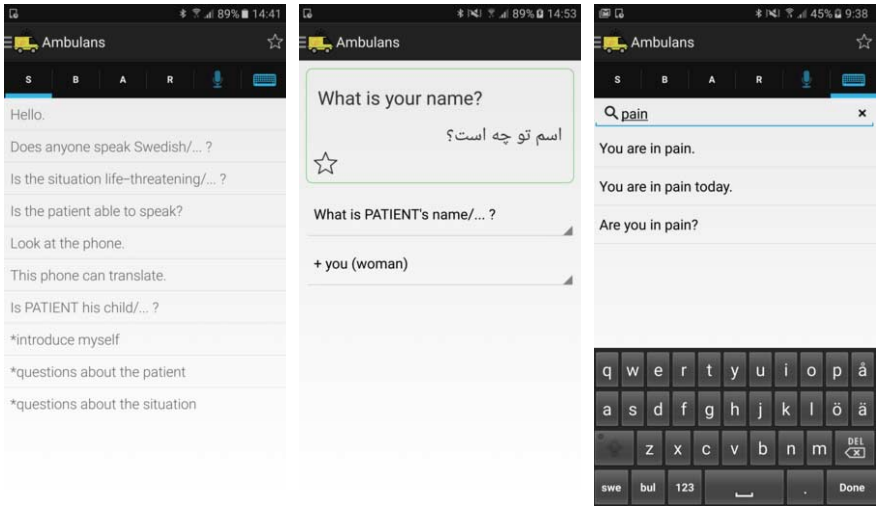


Figure 5. Screenshots of the Ambulance app

user more or less knows where to look for. Since the app is supposed to be used in an emergency situation the staff must be trained if they are supposed to use the menus. On the other hand the menus are great for exploring what is possible. Using the predictive editor is not a good option either since the user tend to just say what they want to say which often falls outside of the grammar's coverage, although during the test we showed that a very similar phrase is indeed available.

The second time we organised half a day session where we presented the purpose of the app to actual ambulance nurses and let them use it. At that time the search facility was in place and we also added a "Favorites" feature which allows quick access to frequently used phrases. We got a very positive feedback since at that time the app allowed both exploration and quick search. The purpose of the session was to acquaint the staff with the app, so we did not do more formal evaluation.

Although we wanted to do actual doctor–patient evaluation as early as possible, at the time of writing, this has not been done yet. The reason is that we had to wait for decision from the management. The current plan is to deploy the app during the summer of 2018. The expectation is that the feedback after the deployment will be as much about the editing functionality as well as for the coverage of the grammar in different real-world situations.

6. Conclusion

The editor as it is serves its purpose. It lets us to explore the coverage of a controlled language and to create new content in that language. It is reconfigurable and was tested with two different grammars. Our experience shows that it is much user-friendly than the existing predictive and syntax editors.

References

[1] Krasimir Angelov. *The Mechanics of the Grammatical Framework*. PhD thesis, Chalmers University of Technology, 2011.

[2] Krasimir Angelov and Aarne Ranta. Implementing controlled languages in GF. In *Proceedings of the 2009 conference on Controlled natural language*, CNL'09, pages 82–101, Berlin, Heidelberg, 2010. Springer-Verlag.

[3] Thomas Hallgren and Aarne Ranta. An extensible proof text editor. In Michel Parigot and Andrei Voronkov, editors, *Logic for Programming and Automated Reasoning*, pages 70–84, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg.

[4] Björn Hedström, Matilda Horppu, and David Michaëlsson. Parlira. An interactive phrasebook for Android devices. Technical report, Chalmers University of Technology, 2016.

[5] Tobias Kuhn. Acewiki: A natural and expressive semantic wiki. *CoRR*, abs/0807.4618, 2008.

[6] Huma Lodhi, Craig Saunders, John Shawe-Taylor, Nello Cristianini, and Chris Watkins. Text classification using string kernels. *Journal of Machine Learning Research*, 2(Feb):419–444, 2002.

[7] Moisés Salvador Meza Moreno and Björn Bringert. Interactive multilingual web applications with grammatical framework. In *GoTAL '08: Proceedings of the 6th international conference on Advances in Natural Language Processing*, pages 336–347, Berlin, Heidelberg, 2008. Springer-Verlag.

[8] Richard Power and Donia Scott. Multilingual authoring using feedback texts. In *Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics - Volume 2*, ACL '98, pages 1053–1059, Stroudsburg, PA, USA, 1998. Association for Computational Linguistics.

[9] Aarne Ranta. *Grammatical Framework: Programming with Multilingual Grammars*. CSLI Publications, Stanford, 2011. ISBN-10: 1-57586-626-9 (Paper), 1-57586-627-7 (Cloth).

[10] Aarne Ranta, Krasimir Angelov, Robert Höglind, Christer Axelsson, and Leif Sandsjö. A mobile language interpreter app for prehospital/emergency care. In *Medicinteknikdagarna, Västerås Sweden, October 10-11, 2017*, 2017.

[11] Aarne Ranta, Ramona Enache, and Grégoire Détrez. Controlled language for everyday use: the molto phrasebook. In *Controlled Natural Language*, pages 115–136. Springer, 2010.

Controlled Natural Language
B. Davis et al. (Eds.)

doi:10.3233/978-1-61499-904-1-11

A Controlled Natural Language for Financial Services Compliance Checking[1]

Shaun AZZOPARDI [2], Christian COLOMBO and Gordon J. PACE
Department of Computer Science, University of Malta, Malta

Abstract. Controlled natural languages have long been used as a surface form for formal descriptions, allowing easy transitioning between natural language specifications and implementable specifications. In this paper we motivate the use of a controlled natural language in the representation and verification of financial services regulations. The verification context is that of payment applications that come with a model of their promised behaviour and which are deployed on a payments ecosystem. The semantics of this financial services regulations controlled natural language (FSRCNL) can produce compliance checks that analyse both the promised model and/or monitor the application itself after it is deployed.

Keywords. financial regulations, controlled natural language, compliance checking, regulation formalisation

1. Introduction

Financial services exist under a highly regulated legal regime, given the high risk of disruptive behaviour such as money laundering. Financial institutions usually have dedicated compliance departments that aim to reduce legal liability by ensuring legal compliance, but ensuring compliance is difficult and thus results in substantial *a priori* investment. Particularly, this need for compliance may discourage such institutions to act as service providers to financial programmes (e.g. by performing transactions as directed by payment applications), especially when they are developed and managed by newcomers to the field, given the element of risk involved.

Automated methods to compliance checking could partially circumvent this problem, allowing for confidence that payment applications are behaving correctly. In the context of compliance of applications one can make use of a multitude of already existing analysis techniques to enforce compliance. However, a particular problem with such an approach is that there is a gap between the formal specification languages of existing analysis tools and the language of the regulations to be checked or enforced. Moreover, legal experts cannot be expected to understand the former, making such specifications not amenable to validation by domain experts.

[1]This research has received funding from the European Union's Horizon 2020 research and innovation programme under grant number 666363.
[2]Corresponding Author: Shaun Azzopardi, Department of Computer Science, University of Malta, Msida, Malta; E-mail: shaun.azzopard@um.edu.mt.

In this paper we discuss our experience in tackling this problem by developing a controlled natural language (CNL) to allow the specification of legal compliance checks in the context of financial services regulations. The *Financial Services Regulation Controlled Natural Language* (FSRCNL) is a subset of the English language, allowing for easy uptake, while it has an unambiguous semantics. We developed this CNL in the context of an industrial project, the *Open Payments Ecosystem* (OPE) [4], and used it mainly to encode the Gibraltar and UK transpositions of the Payment Services, the E-Money, and the Anti-Money Laundering EU Directives. A novel feature of this CNL is that it consists of two sublanguages which are given semantics in different logics. Although not visible from the point of view of the user, since the structure of the language is uniform, the use of different logics allows the specification of regulations that are verifiable at different phases of a payment application's lifetime.

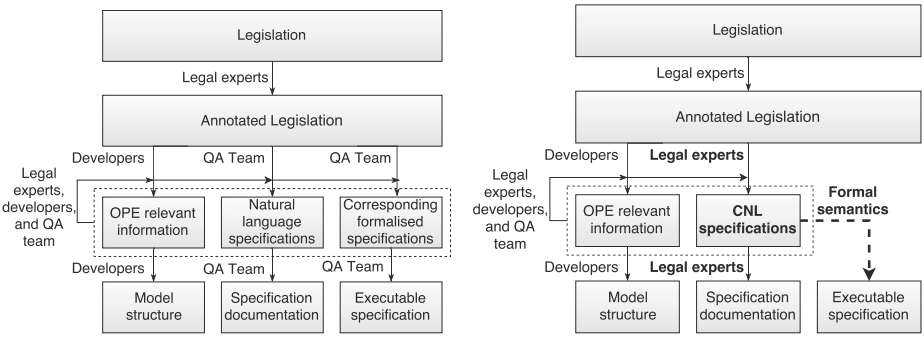
In this paper we detail our experience with this CNL, starting in Section 2 with an overview of the iterative process taken with experts to elicit the meaning and the language of the regulations. We then discuss FSRCNL by giving an overview of its grammar in Section 3 and the corresponding formalisation in Section 4. In Section 5 we discuss qualitatively the CNL and the lessons learnt from its use. In Section 6 we consider related work, before concluding in Section 7.

2. Analysing Financial Services Regulations

Identifying the relevant pieces of legislation which regulate a given system and formalising them is a challenging process. In particular, doing this manually can be prohibitively expensive, given their length and density. General approaches exist to aid such a process by automating the translation of natural language legal documents into a formal representation e.g. [17,5], with varying success. In the narrower scope of building a compliance checking engine for a particular system and domain, this problem is less acute, even though the interaction between the legal and technical experts still requires substantial effort. In our context, the compliance engine had to be incorporated within a general framework (the Open Payments Ecosystem, or OPE) to be used by developers to construct financial applications, which led to two major challenges: (i) the compliance engine had to be developed to work for any system communicating with the OPE; (ii) the legislation is different from one country to another, and changes on a regular basis, and thus a solution which supports easier update and transposition was required.

2.1. Identifying Automatically Verifiable Regulations

One of the major challenges in using automated analysis of legal texts is that the domain covered by legislation is much wider than the one for which compliance systems usually work — also true when restricting oneself to relevant legislative chapters e.g. the OPE does not implement all the payment services regulated by legislation. This means that pruning of legislation to relevant parts is a crucial first step. We thus undertook a manual process of requirements elicitation that exploited legal experts' intimate knowledge of the regulations. A law firm was consulted, were we had direct contact with three lawyers, but with other lawyers being consulted in the process from within the firm. This process started with lawyers identifying which clauses in the legislation were relevant to the business



(a) The formalisation process without FSRCNL. (b) The formalisation process with FSRCNL.

Figure 1. Alternate formalisation processes.

process of the system. Some of these clauses were however found not to be verifiable in an automatic manner. For instance, consider our challenge of incorporating a compliance engine within a library for building financial applications. A regulation may require that relevant terms and conditions should be presented to the financial application user before any transactions are carried out. However this cannot be easily verified since (i) displaying of text is performed outside the scope of the library; and (ii) ensuring that the (free) text of the terms and conditions makes sense is not possible with current natural language technology. Such clauses were thus deemed unverifiable by the compliance engine even though directly relevant to the domain. Thus, the lawyer-identified *relevant clauses* were further filtered by the system's developers to acquire a smaller set of *verifiable clauses*.

In order to support the communication between the teams involved, it was realised that we needed to be able to maintain versions of these automatically verifiable regulations in three formats (other than their representation in the regulations): informally (for the lawyers to ensure that the interpretation is in compliance with the actual law), formally (for the quality assurance team to ensure the meaning of the informal representation), and using an executable representation (to be able to verify the formal regulations). However, developing three versions of the specification in parallel (as illustrated in Figure 1a.) would require an inordinate amount of replication of work. Furthermore, if these parallel versions are developed separately, misunderstandings and ambiguities in interpretation of terms between legal experts and developers could arise. For instance, the rule stating that "*Only prepaid instruments can be used with e-money*" was misunderstood by developers, since they associated a different meaning with the word *instrument* than that meant by lawyers. This could easily have led to a mismatch between what should have been checked and what was actually checked. Iterative meetings between the different teams uncovered such discrepancies.

To avoid such problems in the future we opted to construct a CNL with a semantics that is more aligned to what a legal expert expects. It can be used to specify regulations with little prior knowledge of the system and allows executable specifications to be automatically generated. Using this CNL as the primary source of compliance checks places the duty of specification back on the actual domain expert, the legal experts. The process aided with the use of the CNL, is illustrated in Figure 1b.

Regulation	Acronym	Verifiable Clauses
The Electronic Money Regulations 2011 (SI 2011/99)	EMR	11
The Payment Services Regulations 2009 (SI 2009/209)	PSR	14
The Money Laundering Regulations 2009 (SI 2009/209)	MLR	4
Fourth Money Laundering Directive (EU) 2015/849	MLD4	0
European Commission's Proposal for a Directive Amending MLD4	MLD5	2

Table 1. List of UK regulations considered.

2.2. The Language of the Regulations

Payment services are regulated tightly by the EU, with several relevant directives being in force. The OPE is planned to be initially deployed in the UK but later to be extended to cover other countries. We thus limited ourselves to the UK-specific implementations of these directives, but covering also minor differences with the legislation of Gibraltar. Table 1 illustrates the main UK regulations considered (and directives that were not yet transposed at the time), their associated acronym as used here, and the amount of clauses identified as verifiable (and specified using our CNL) from each of them. These regulations cover programmes that perform some form of payment service (e.g. issuing payment cards and other payment instruments), with each of these services performed by a certain service provider licensed in an appropriate country. In this section we consider the language of these clauses through examples.

Regulatory documents are normative documents, specifying what regulated entities can and cannot do, starting with definitions of the domain-specific jargon, for example:[3]

EMR2(1) *"electronic money" means electronically (including magnetically) stored monetary value as represented by a claim on the electronic money issuer which (a) is issued on receipt of funds for the purpose of making payment transactions; [...]*

These terms are then used to specify obligations and prohibitions restricting behaviour:

EMR45 *An electronic money issuer must not award (a) interest in respect of the holding of electronic money; or (b) any other benefit related to the length of time during which an electronic money holder holds electronic money.*

These kind of clauses identify the responsible entity (the issuer), the kind of norm (a prohibition), and the relevant action or actions (awarding interest). To be able to enforce this we need to at least be able to represent the forbidden behaviour or state. Syntax from deontic logics can be used to model these norms e.g. using F to denote a prohibition, one can write $F_{issuer}(awardInterest)$. A regulation may also be conditioned on some limits holding, both with respect to some time-frame or a monetary value:

ML13(7)(d)(ii) *[...] if the device can be recharged, a limit of 2,500 euro is imposed on the total amount transacted in a calendar year, except when an amount of 1,000 euro or more is redeemed in the same calendar year by the bearer [...]*

In summary, a representation to encode these regulations must then allow for specifying what should take place, and negation to specify what should not. Articles in the legislation tend to be limited to particular situations and contexts, motivating the need for a way to represent the conditions necessary for a certain state to hold. Also, the representation should include finance-specific notions and constructs, while allowing for both time and monetary qualifiers.

[3]The following and all subsequent legal texts are taken from UK legislation.

3. A Financial Services Regulations Controlled Natural Language

The *Financial Services Regulations Controlled Natural Language* (FSRCNL) was designed to allow legal experts to specify compliance checks for payment services specific entities (e.g. *transaction*s, *instrument*s, *service provider*s). The vocabulary of the language also includes verbs that can be used to specify relations between these entities (e.g. programme *p is regulated in the UK*). Apart from these, the language includes (i) temporal notions allowing for the expression of when something occurred (e.g. instrument *i expired less than 12 months ago*), and (ii) monetary expressions specifying limits on a value (e.g. transaction *t deals with exactly 500 EUR*). These can be combined to specify conditions on aggregate values (e.g. *the amount redeemed from i within a calendar year is less than 1000 EUR*). Some interesting aspects of the language are the following:

Types The regulations define several payment-specific objects, as well as certain roles for entities providing payment services. To handle the several laws regulating the behaviour of entities taking such roles, we include the possibility of declaring quantified variables over such types (roles) in FSRCNL (e.g. *programme p*).

⟨TYPE-NAME⟩ ::= programme | service provider | instrument | transaction | country
⟨DECLARATION⟩ ::= [(⟨TYPE-NAME⟩,)* or]?⟨TYPE-NAME⟩ ⟨VARIABLE⟩

Payment relations/Sentences Basic relations between variables over the defined payment types serve as the basis for sentences. These are of the subject-verb-object form, and can be negated. They are used to specify both conditions limiting a clause and the state that must hold given the conditions, as in:

For each instrument i and programme p, where *i is an instrument of p* , *p is regulated in the UK* ,
then *e-money in i is redeemed without fees.*

Monetary Expressions Monetary expressions can act as objects of a relation, e.g. one can specify that a transaction *t deals with exactly 500 EUR*, allowing also for specifying *less than* or *more than* a certain value.

Temporal and Country Qualifiers Relations can be further qualified with some temporal, or location constraint, refining a relation to hold only in a certain country (e.g. given service provider *sp*, and programme *p*, *sp deploys p in the UK*). Furthermore, we can also refine sentences to a time period before or after a certain point in time (e.g. instrument *i expired less than 12 months ago*).

⟨QUALIFIED-SENTENCE⟩ ::= ⟨SENTENCE⟩
| ⟨SENTENCE⟩ ⟨TEMPORAL-QUALIFIER⟩
| ⟨SENTENCE⟩ ⟨COUNTRY-QUALIFIER⟩
⟨TEMPORAL-QUALIFIER⟩ ::= in less than ⟨TIME⟩ | in more than ⟨TIME⟩
⟨COUNTRY-QUALIFIER⟩ ::= in ⟨COUNTRY⟩ | not in ⟨COUNTRY⟩

Guarded Declarations and Guards A FSRCNL specification can have a list of variable declarations, which can possibly be guarded by a compound sentence (e.g. *service provider sp, and programme p where sp deploys p in the UK*).

⟨GUARDED-DECLARATION⟩ ::= ⟨DECLARATION-LIST⟩ ⟨GUARD⟩
⟨GUARD⟩ ::= ε | where ⟨COMPOUND-SENTENCE⟩
⟨COMPOUND-SENTENCE⟩ ::= [(⟨QUALIFIED-SENTENCE⟩,)* and]⟨QUALIFIED-SENTENCE⟩
| [(⟨QUALIFIED-SENTENCE⟩,)* or]⟨QUALIFIED-SENTENCE⟩

Quantifiers The variables declared can be universally or existentially quantified over.

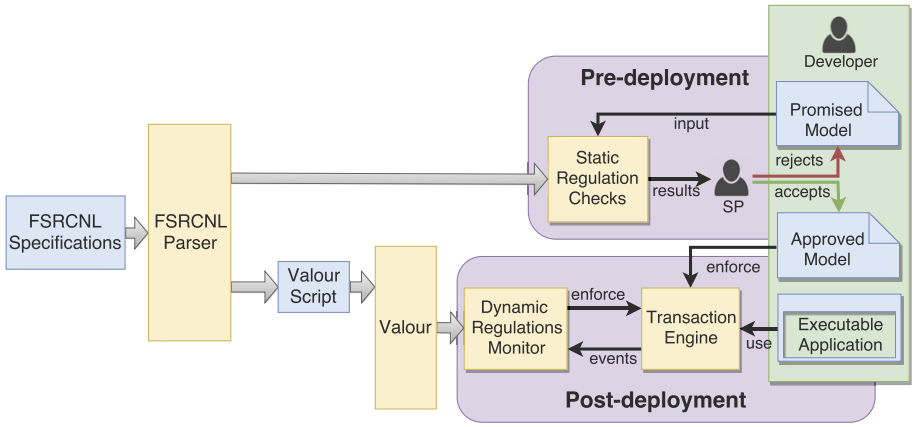


Figure 2. OPE payment application business process with verification facilitated by FSRCNL.

⟨QUANTIFIED-PROPOSITION⟩ ::= For each ⟨GUARDED-DECLARATION⟩ then ⟨COMPOUND-SENTENCE⟩
| For at least one ⟨GUARDED-DECLARATION⟩ then ⟨COMPOUND-SENTENCE⟩

In FSRCNL, **EMR45** would be represented as follows:

> *For each programme p, and instrument i, where i is an instrument of p, p is regulated in the UK, and i deals with e-money, then i does not give time-based rewards.*

The language and a parser for FSRCNL was built using Haskell and the *parsec* package, using around 700 lines of code. The language was built from scratch, in a compositional manner so as to allow for easy extension when the need arises.

4. Verifying FSRCNL Rules

We used FSRCNL in the context of an ecosystem that provides common functionality needed for payment applications. The OPE was designed in such a manner that third party payment applications are seen as black boxes interacting with service providers through the OPE. While observing the runtime behaviour of applications enables the OPE to ensure compliance, ideally non-compliant applications are not allowed to use the platform upfront. Thus, developers are required to provide a *promised model* specifying assurances about the application's behaviour (e.g. the model can specify that the application will only perform transactions within the UK). The compliance verification is divided into two stages: (i) pre-deployment the OPE uses the developer-provided model of the application, by statically verifying it against the formulated legislation; while (ii) post-deployment verification involves the analysis of the dynamic features of an application in order to ensure that it is working according to the constraints of the legislation and according to the promised model that was used for the pre-deployment analysis (e.g. transaction values). In this context, FSRCNL relations can either be linked to artifacts known pre-deployment (statically) or to the actual application's behaviour post-deployment (dynamically).

Syntax and Semantics The sentence form in FSRCNL is similar to a quantified predicate logic implication, we can give static rules their semantics in terms of such a

logic. For **EMR45** we can give its semantics as follows, assuming a programme p: $\forall i \in \texttt{instruments}(p) \cdot regulatedIn(p, UK) \wedge emoney(i) \implies noTimeRewards(i)$. Given such a semantics of the CNL, it is straightforward to take the payment application model (which essentially provides information about the resource flow of the financial instruments which the application will enable) and confirm that it complies to these quantified formulae.

Some relations in the FSRCNL cannot be linked to attributes known before deployment, and thus have to be pushed post-deployment. Recall regulation **ML13(7)(d)(ii)** having guard: *the amount redeemed from i within a calendar year is less than 1000 EUR*. This amount can only be known at runtime, and thus we postpone checking this rule to post-deployment using a runtime monitor. The semantics of the post-deployment logic is given in terms of an event-based language, as required for the runtime verification tool VALOUR [2].

For instance, from regulation **ML13(7)(d)(ii)**, VALOUR generates executable code for each instrument (i) keeping a running total of amounts redeemed from transactions (on that instrument), reset upon the beginning of a calendar year; and (ii) setting up a guarded event to trigger when the running total exceeds €1000, signalling a violation of the regulation.

These checks are applied directly to the OPE (via the Java code generated from the static analysis checks and the code generated by VALOUR), adding a safety layer around the applications use of the API on the OPE side — ensuring that any compliance breach is reported. Figure 2 illustrates the executable specification generation process through the use of FSRCNL, and their verification at the different stages of an application's lifetime as explained in this section.

5. Discussion and Lessons Learnt

In this section we discuss several issues related both to the semantics and syntax of the language, and lessons learnt from the process we undertook.

When analysing the regulations we found ambiguity between the terms used by the lawyers and the developers, e.g. the term *instrument* was used to mean different but similar things for both of them which created some problems in the formalisation early on. Deciding to use the system-level constructs at a high-level could have led to behaviour unintended by the legal expert authors. When designing a CNL, we can avoid such a situation by taking into account the *target authors* of the language and their preconceptions about its vocabulary, and putting that above any other considerations. In fact we geared the language of our CNL to keywords and phrases as found and used by the legal experts, hiding the verified system's constructs in the semantics of the language.

The CNL however still uses somewhat unnatural structures more reminiscent of logic rather than legal text, particularly explicit quantification and variables. These are used to remove the ambiguities of the anaphora common in natural language text. Although we aimed for the CNL to be as natural as possible, removing any avenues for ambiguity necessarily results in a somewhat formulaic structure [10]. The effect of this on the usability of the language has not yet been evaluated with legal experts. Development of FSRCNL only started after the process illustrated in Figure 1a was largely over, and given the nature of the project, the legal experts were not available for evaluation.

In the process of requirements elicitation, as common with natural language texts, we found the laws to be ambiguous or underspecified at times, e.g. one clause specifies

that issuing of e-money should be done *without delay*[4]. With these kind of regulations the lawyers depended on guidance from relevant authorities and their experience, e.g. *without delay* was interpreted as meaning that a certain delay was acceptable, namely the amount of time needed to process such a request, and an approximate numeric value was agreed with the developers. Thus the semantics we gave to FSRCNL encode a specific interpretation of the regulations.

Given the narrow scope of the language, the variety of notions in the language, and the limited kind of clauses that are automatically verifiable, we do not expect the language will need substantial change in structure in the face of new regulations. This was validated by our experience when constructing the language. The language was based on two sets of regulation documents (the e-money and payment services regulations). We then further extended it with respect to the money laundering (ML) related regulations. This extension of the language only involved the introduction of new ML-specific verbs. In this case the general structure of the language was found to be adequate to represent the previously unseen regulations.

In [14] Kuhn introduces a classification scheme for CNLs, namely the PENS scheme, standing for precision, expressiveness, naturalness, and simplicity. Each dimension is restricted to five classes where, e.g., for precision (or lack of ambiguity), P^1 denotes the lowest possible precision and P^5 the highest. According to the definitions given by Kuhn, we would classify FSRCNL as $P^4E^3N^3S^4$. The language is not maximally precise since the semantics of its basic sentences depend on the underlying system, in our case the OPE, while expressiveness is limited since it does not include second-order universal quantification. Naturalness of the language is reduced by the use of variable declarations (which is not something every English user would feel to be natural) and by the repetitive nature of the specifications. Simplicity is defined in terms of the length of an "exact and comprehensive description", S^4 denotes that FSRCNL can be described in not more than ten pages.

We consider briefly the application of the language to verification of applications. In verification although we ideally want to prove the application correct pre-deployment, this may not always be possible (e.g. its source code is not available, or Turing-completeness of the code makes this difficult), and we may have to leave some work for after deployment [3]. FSRCNL is a key part in our approach to this problem in the context of the OPE — the parser we developed for FSRCNL is able to automatically classify specifications into those that can be proven fully pre-deployment, and those that have to be left for runtime.

6. Related Work

CNLs have been applied to serve as an interface to verification tools before [15,13,12]. In particular, Grover et. al. in [12] specify a language for the verification of hardware models that translates into a temporal logic. This work proposes allowing CNLs to have some ambiguity, since some words may be inherently ambiguous, while managing it by presenting the different interpretations to the user for feedback. In our work we opted for a CNL without ambiguity for ease of processing. Future work could include considering different modalities of FSRCNL depending on the kind of user using it (legal expert or

[4]UK Electronic Money Regulations 2011 Regulation 39(a).

developer), and letting the user choose between different options of semantics for some terms (e.g. to vary the processing time allowed for "without delay").

Similar to our paper, [15] presents a CNL for the specification of regulations to be verified against a model but in the context of railways. Like other CNLs intended for legal source representation (e.g. [1,7,16,8]), it makes explicit use of deontic notions (i.e. obligation, permission, and prohibition). We found the current form of FSRCNL to be enough to specify what should be specified, but the introduction of explicit deontic notions can be considered if it will facilitate a legal expert's understanding of the language.

A popular English CNL that can be mapped to first-order logic is Attempto Controlled English (ACE [11]). Subsets of it has been mapped to logics for different domains, e.g. a privacy policy language [9]. Using ACE allows the use of different already existing tools to reason with the specification. We opted to develop our own CNL to be fully in control of its syntax.

Previous CNLs by authors of this paper dealing with analysis took the form of imperative commands [6,10], in that they were used to instruct the system what actions to take. On the other hand, FSRCNL sentences are declarative, with the semantics engine inferring the instructions for checking for the specified conditions, thus acting as a filter.

7. Conclusions

We have reported on our experience of developing a controlled natural language (CNL) for the compliance checking of financial services regulations in an industrial project. The language was developed to replace a manual process of developing these checks, involving multiple stakeholders and artifacts. Our intention is to use the *Financial Services Regulations CNL* to allow legal experts to specify rules that both document and automatically generate implementable compliance checks. We hypothesise that using FSRCNL will streamline this process. The next step in this work is to evaluate this language with legal experts in terms of ease of use, upon the introduction of new applicable legislation.

References

[1] Krasimir Angelov, John J. Camilleri, and Gerardo Schneider. A framework for conflict analysis of normative texts written in controlled natural language. *The Journal of Logic and Algebraic Programming*, 82(5):216 – 240, 2013. Formal Languages and Analysis of Contract-Oriented Software (FLACOS'11).

[2] Shaun Azzopardi, Christian Colombo, Jean-Paul Ebejer, Edward Mallia, and Gordon J. Pace. *Runtime Verification using* VALOUR. EPiC Series in Computing, Volume, 2016.

[3] Shaun Azzopardi, Christian Colombo, and Gordon Pace. *A Model-Based Approach to Combining Static and Dynamic Verification Techniques*, pages 416–430. Springer International Publishing, Cham, 2016.

[4] Shaun Azzopardi, Christian Colombo, Gordon J. Pace, and Brian Vella. *Compliance Checking in the Open Payments Ecosystem*, pages 337–343. Springer International Publishing, Cham, 2016.

[5] Shaun Azzopardi, Albert Gatt, and Gordon J. Pace. Integrating natural language and formal analysis for legal documents. In *10th Conference on Language Technologies and Digital Humanities 2016*, 2016.

[6] Aaron Calafato, Christian Colombo, and Gordon J. Pace. A controlled natural language for tax fraud detection. In *Proceedings of the 5th International Workshop on Controlled Natural Language - Volume 9767*, CNL 2016, pages 1–12, New York, NY, USA, 2016. Springer-Verlag New York, Inc.

[7] John J. Camilleri, Gordon J. Pace, and Michael Rosner. Controlled natural language in a game for legal assistance. In Michael Rosner and Norbert E. Fuchs, editors, *Controlled Natural Language*, pages 137–153, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[8] John J. Camilleri, Gabriele Paganelli, and Gerardo Schneider. A cnl for contract-oriented diagrams. In Brian Davis, Kaarel Kaljurand, and Tobias Kuhn, editors, *Controlled Natural Language*, pages 135–146, Cham, 2014. Springer International Publishing.

[9] Juri Luca De Coi, Philipp Kärger, Daniel Olmedilla, and Sergej Zerr. Using Natural Language Policies for Privacy Control in Social Platforms. Heraklion, Greece, Jun 2009. CEUR-WS.org.

[10] Christian Colombo, Jean-Paul Grech, and Gordon J. Pace. A controlled natural language for business intelligence monitoring. In Chris Biemann, Siegfried Handschuh, André Freitas, Farid Meziane, and Elisabeth Métais, editors, *Natural Language Processing and Information Systems*, pages 300–306, Cham, 2015. Springer International Publishing.

[11] Norbert E. Fuchs, Kaarel Kaljurand, and Tobias Kuhn. *Attempto Controlled English for Knowledge Representation*, pages 104–124. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[12] Claire Grover, Alexander Holt, Ewan Klein, and Marc Moens. Designing a controlled language for interactive model checking. In *Proceedings of the 3rd International Workshop on Controlled Language Applications (CLAW 2000)*, 2000.

[13] Kristofer Johannisson. *Natural Language Specifications*, pages 317–333. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.

[14] Tobias Kuhn. A survey and classification of controlled natural languages. *Comput. Linguist.*, 40(1):121–170, March 2014.

[15] Bjørnar Luteberget, John J. Camilleri, Christian Johansen, and Gerardo Schneider. Participatory verification of railway infrastructure by representing regulations in railcnl. In Alessandro Cimatti and Marjan Sirjani, editors, *Software Engineering and Formal Methods*, pages 87–103, Cham, 2017. Springer International Publishing.

[16] Gordon J. Pace and Michael Rosner. A controlled language for the specification of contracts. In Norbert E. Fuchs, editor, *Controlled Natural Language*, pages 226–245, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[17] Adam Z. Wyner and Guido Governatori. A study on translating regulatory rules from natural language to defeasible logics. In *Joint Proceedings of the 7th International Rule Challenge, the Special Track on Human Language Technology and the 3rd RuleML Doctoral Consortium, Seattle, USA, July 11 -13, 2013*, 2013.

Controlled Natural Language
B. Davis et al. (Eds.)

doi:10.3233/978-1-61499-904-1-21

Responsive and Flexible Controlled Natural Language Authoring with Zipper-Based Transformations

Sébastien FERRÉ [1]
Univ Rennes, CNRS, IRISA
Campus de Beaulieu, 35042 Rennes cedex, France
ferre@irisa.fr

Abstract Controlled natural languages (CNL) have the benefits to combine the readability of natural languages, and the accuracy of formal languages. They have been used to help users express facts, rules or queries. While generally easy to read, CNLs remain difficult to write because of the constrained syntax. A common solution is a grammar-based auto-completion mechanism to suggest the next possible words in a sentence. However, this solution has two limitations: (a) partial sentences may have no semantics, which prevents giving intermediate results or feedback, and (b) the suggestion is often limited to adding words at the end of the sentence. We propose a more responsive and flexible CNL authoring by designing it as a sequence of sentence transformations. Responsiveness is obtained by having a complete, and hence interpretable, sentence at each time. Flexibility is obtained by allowing insertion and deletion on any part of the sentence. Technically, this is realized by working directly on the abstract syntax, rather than on the concrete syntax, and by using Huet's zippers to manage the focus on a query part, the equivalent of the text cursor of a word processor.

Keywords. Controlled Natural Languages, Authoring, User Interaction, Abstract Syntax, Huet's Zippers, Focus

1. Introduction

An important issue in the Semantic Web [8], and knowledge-based systems in general, is to fill the gap between the natural language (NL) of users, and the formal languages (FL) of those systems (e.g., OWL, SPARQL, Answer Set Programming). Formal languages make data processable by machines but they also constitute a language barrier to the production and consumption by end users. Controlled Natural Languages (CNL) [12] offer an interesting solution in that they combine the readability of natural languages, and the accuracy of formal languages. An input sentence respecting the syntactic and semantic constraints of the CNL can be parsed non-ambiguously into a formal expression (e.g. a query), and hence can be automatically interpreted (e.g. computing query results). CNLs also offer a good *adequacy* between what can be expressed respectively by the

[1]This research is supported by ANR project PEGASE (ANR-16-CE23-0011-08).

natural language and the formal language. In contrast, the spontaneous natural language approach generally adopted in Question Answering (QA) [13] suffers from problems of ambiguity and adequacy.

While CNLs are much easier to read compared to FL, they remain difficult to write because of the constrained syntax. Syntactic and semantic errors can be frequent and frustrating, especially for users who are new to the CNL and/or the domain vocabulary. The latter is related to the well-known *habitability* problem [11], which occurs when users do not known exactly what falls in the scope of the knowledge-based system, and what is out of scope. A common solution in CNL editors is to use an *auto-completion* mechanism (e.g., ACE [10], Ginseng [11]) that suggests the next possible words in a sentence. Those suggestions are derived from the CNL grammar and the domain vocabulary, and possibly predicted from usage statistics [6,15]. The major advantages of auto-completion are to prevent syntactic errors, and to alleviate the habitability problem. However, we here identify two limitations of auto-completion for CNL authoring. First, as auto-completion is designed to complete a sentence word after word, it results that at most steps the current sentence is not complete, and hence cannot be translated as such to the formal language, and therefore cannot be interpreted. For example, in the case of a query, results can be computed and returned only once the sentence expressing it is complete. When the query happens to have no result, the user will only detect it at the end even if the constraints expressed in the first half of the query were already sufficient to make the results empty. Second, auto-completion is limited to adding words at the end of the partial sentence. In comparison, a text editor allows for insertions and deletions at any position, and has a cursor to let the user control that position. For a user, expressing her knowledge or information needs is generally not a linear process, and goes through rectifications, insertions, substitutions, etc. There is a need to combine suggestions with a flexible authoring process.

We propose an alternative solution to guide users in the authoring of CNL sentences, by following the principles of the `N<A>F` design pattern [3]. In our proposal, the authoring process is based on sequences of transformations at the abstract syntax level, rather than on the addition of words at the concrete syntax level. As a consequence, the concrete form is obtained by NL generation from the abstract form, rather than the reverse by syntactic parsing as usually done in CNL and QA. The formal expression is obtained as usual by translation from the abstract form, generally using compositional semantic techniques such as Montague grammars [2]. We claim that our solution improves auto-completion in terms of *responsiveness* and *flexibility*. It is responsive because transformations are designed to take a complete sentence as input, and to return a complete sentence as output. Therefore, at each step, the current sentence is complete, and can therefore be interpreted to give the user intermediate results and feedback. Our solution is also flexible because it allows for insertions and deletions on any part of the sentence, where a part is defined as a node of the abstract syntax tree. We use Huet's zippers [9] as a technique to represent that part, called *focus*, which plays a role similar to the text cursor of a word processor. We illustrate our approach on a small yet expressive query language whose target FL is SPARQL [17].

The paper is structured as follows. Section 2 shortly recalls the principles of the `N<A>F` design pattern, and Huet's zippers. Section 3 describes our approach in detail through a concrete use case. Section 4 reports on the implementation and application of our approach to several tasks.

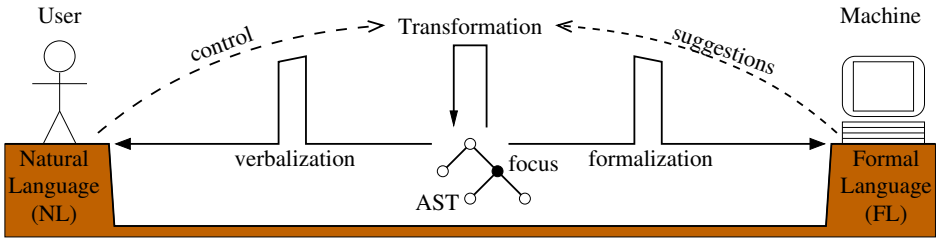


Figure 1. Principle of transformation-based authoring for bridging the gap between NL and FL

2. Preliminaries

2.1. *The* `N<A>F` *Design Pattern*

The principle of the `N<A>F` design pattern [3] is schematized as a "suspended bridge over the NL-FL gap" in Figure 1. The central pillar is made of Abstract Syntax Trees (AST) with a focus on one AST node. The nature of the intermediate language abstracted by the ASTs depends on the application: e.g., queries, descriptions, logical axioms. The AST is initialized by the system, and modified by users applying structural *transformations*, not by direct textual input. For that reason, it is important to design a *complete* set of transformations, so that every AST is reachable through a finite sequence of transformations. Conversely, only *safe* transformations should be suggested to users, so as to avoid syntactic and semantic errors. In the case of querying, a semantic error could be applying a transformation that leads to an empty result. In the case of ontology construction, it could be constructing an axiom that leads to inconsistency.

In order for the AST structure to be understood by both the user and the machine, *verbalization* translates ASTs to NL, and *formalization* translates ASTs to FL. In addition to those translations, verbalization supports user control by showing suggested transformations in NL, and formalization supports the computation of suggestions by taking into account the semantics of the AST. A key issue in the design of ASTs is to make the two translations semantically transparent, and simple enough. First, the AST structure should reproduce the syntactic structure of NL (e.g., sentences, noun phrases, verb phrases), while abstracting as many details as possible. Indeed, starting with a flat representation like SPARQL, it is possible to produce a NL version [14], but it is difficult to make it stable across transformations. Second, the AST structure should semantically align with the target FL. Indeed, every AST that can be obtained by a sequence of transformations must have a semantics that is expressible in FL. The design pattern supports multi-lingualism because only verbalization depends on the chosen NL. In particular, the NL can be changed at any time in the course of a user session. Moreover, NL generation is known to be easier than NL understanding so that it is easier to support more languages.

In this paper, we concentrate on the representation of the abstract syntax, the focus, and the transformations. We assume classical techniques for the verbalization to NL (e.g., Grammatical Framework [16]), and for the translation to FL (e.g., Montague grammars [2]).

2.2. Huet's Zippers

In his "Functional Pearl" [9], Huet introduced the *zipper* as a technique for traversing and updating a data structure in a purely functional way, and yet in an efficient way. Pure functional programming completely avoids modification in place of data structures, and makes it much easier to reason on program behaviour, and hence to ensure their correctness [18]. We here use zippers for the incremental construction of ASTs. A simple and illustrative example is on simply chained lists, their traversal, and the insertion of elements. Given a base type *elt* for list elements, the *list* datatype is defined with two constructors: one for the empty list, one for adding an element at the head of another list.

$$list ::= \mathbf{Nil} \mid \mathbf{Cons}(elt, list)$$

The AST of list $[1,2,3]$ is $\mathbf{Cons}(1,\mathbf{Cons}(2,\mathbf{Cons}(3,\mathbf{Nil})))$. The zipper idea is to keep a location in the list such that it is easy and efficient to insert an additional element at that location, and also to move that location to the left or to the right. A location (e.g. at element 2 in the above list) splits the data structure in two parts: the *sub-structure* at the location ($[2,3]$), and the surrounding *context* ($[1,\_]$). It has been shown that the context datatype corresponds to a data structure with one hole, and can be seen as the *derivative* of the structure datatype [1]. We therefore name $list'$ ("list-prime") the context datatype for lists, and define it as follows.

$$list' ::= \mathbf{Root} \mid \mathbf{Cons}'(elt, list')$$

That definition says that a list occurs either as a root list or as the right-argument of constructor **Cons**, which has in turn its own context. For example, the context at location 3 of list $[1,2,3]$ is $\mathbf{Cons}'(2,\mathbf{Cons}'(1,\mathbf{Root}))$. In fact, a list context is the reverse list of the elements before the location. Finally, a zipper data structure combines a structure and a context: $zipper ::= \mathbf{List}(list, list')$. A zipper contains all the information of a data structure plus a location in that structure. That location is also called "focus".

A zipper makes it easy to move the location to neighbour locations, and to apply local transformations such as insertions or deletions. For example, to insert an element x in a list zipper $\mathbf{List}(l,l')$ is as simple as returning the zipper $\mathbf{List}(\mathbf{Cons}(x,l),l')$. Given a zipper $\mathbf{List}(l,\mathbf{Cons}'(e,l'))$, the location can be moved to the left by returning the zipper $\mathbf{List}(\mathbf{Cons}(e,l),l')$.

3. Detailed Use Case: SPARQL-based Querying

In this section, we describe in detail our approach taking SPARQL-based querying as a use case[2]. Our abstract intermediate query language covers the basic graph patterns, and their composition with union (UNION) and negation (NOT EXISTS).

3.1. AST Zippers

The following datatype definitions describe the abstract syntax of our query language. Given base types for RDF nodes (*node*), RDFS classes (*class*), and RDFS properties

[2]The complete source code of an extension of this use case is available online in the OCaml programming languages. Visit http://www.irisa.fr/LIS/ferre/pub/CNL2018.ml

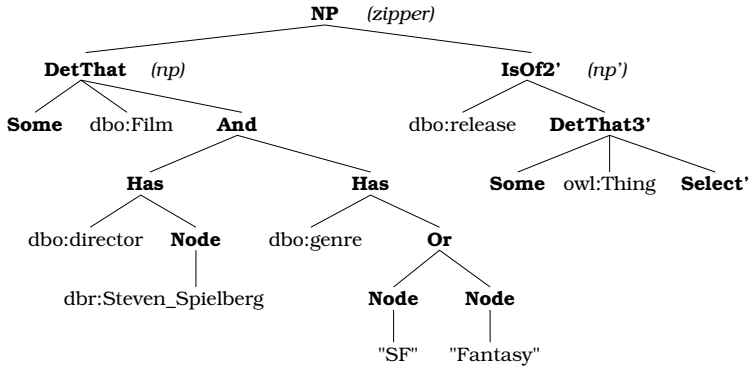


Figure 2. Example zipper made of a sub-structure (np), and a context (np')

($prop$), we define abstract sentences (s), noun phrases (np), determiners (det), and verb phrases (vp).

$$
\begin{aligned}
s &::= \mathbf{Select}(np)\\
np &::= \mathbf{Node}(node) \mid \mathbf{DetThat}(det, class, vp)\\
&\quad \mid \mathbf{And}(np, np) \mid \mathbf{Or}(np, np) \mid \mathbf{Not}(np)\\
det &::= \mathbf{Some} \mid \mathbf{Every} \mid \mathbf{No}\\
vp &::= \mathbf{IsA}(class) \mid \mathbf{Has}(prop, np) \mid \mathbf{IsOf}(prop, np)\\
&\quad \mid \mathbf{True} \mid \mathbf{And}(vp, vp) \mid \mathbf{Or}(vp, vp) \mid \mathbf{Not}(vp)
\end{aligned}
$$

Note that this is very similar to the definition of abstract syntax in Grammatical Framework [16], with *categories* here in italic, and *functions* here in bold. Node np in Figure 2 points to the tree representation of an example AST. It has type np, and specifies *"any film directed by Steven Spielberg and whose genre is SF or Fantasy"*. The AST definitions reflect NL syntax with noun phrases and verb phrases, but is indeed abstract because all sorts of syntactic distinctions are ignored. The type vp is used to represent relative clauses (occurence of vp in **DetThat**) because the two have the same semantics. The two constructors **Has** and **IsOf** account for the traversal direction of a property, but not whether the property is verbalized with a verb, a noun, or a transitive adjective. Note that Boolean connectors are defined on both NPs and VPs.

The following datatype definitions describe the structure of AST contexts. They are automatically obtained as the derivatives of the AST datatypes (see Section 2.2). When a constructor has several arguments (e.g., **And**), the derived constructors are indexed by the position of the focus (e.g., $\mathbf{And}'_2$ for a focus on the second argument).

$$
\begin{aligned}
np' &::= \mathbf{Select}' \mid \mathbf{Has}'_2(prop, vp') \mid \mathbf{IsOf}'_2(prop, vp')\\
&\quad \mid \mathbf{And}'_1(np', np) \mid \mathbf{And}'_2(np, np') \mid \mathbf{Or}'_1(np', np) \mid \mathbf{Or}'_2(np, np') \mid \mathbf{Not}'(np')\\
vp' &::= \mathbf{DetThat}'_3(det, class, np')\\
&\quad \mid \mathbf{And}'_1(vp', vp) \mid \mathbf{And}'_2(vp, vp') \mid \mathbf{Or}'_1(vp', vp) \mid \mathbf{Or}'_2(vp, vp') \mid \mathbf{Not}'(vp')
\end{aligned}
$$

Node np' in Figure 2 points to the tree representation of an example AST context. It has type np', and specifies the one-hole AST *"select the release date of _"*, where the underscore (hole) gives the location of the zipper sub-structure. Note that the path in the context AST from the root to $\mathbf{Select}'$ is the reverse of the path in the one-hole AST from the root to the hole. Finally, the following datatype definition describes a AST zipper, combining an AST and an AST context.

$$zipper ::= \mathbf{NP}(np, np') \mid \mathbf{VP}(vp, vp')$$

Therefore, when editing a query, the focus can be put on any noun phrase (i.e. on any entity involved in the query) or on any particular verb phrase (i.e. on any description of any entity in the query). Figure 2 displays the tree representation of an example zipper that represents the NL question *"Give me the release date of films directed by Steven Spielberg and whose genre is SF or Fantasy"*, where the focus is on films. Its formalization in SPARQL is: `SELECT ?x1 ?x2 WHERE {?x2 dbo:releaseDate ?x1. ?x2 rdf:type dbo:Film. ?x2 dbo:director dbr:Steven_Spielberg. {?x2 dbo:genre "SF"} UNION {?x2 dbo:genre "Fantasy"}}`.

3.2. A Complete Set of Zipper Transformations

Because ASTs are only built by the successive and interactive application of transformations, it is important to define one or several initial zippers and a set of zipper transformations that makes the building process complete.

Definition 1 (completeness)**.** *A set of initial zippers and a set of zipper transformations are* complete *w.r.t. AST datatypes iff every AST zipper can be reached by applying a finite sequence of transformations starting with an initial zipper.*

We start by defining an initial AST x_0 for each AST datatype x: $s_0 := \mathbf{Select}(np_0)$, $np_0 := \mathbf{DetThat}(\mathbf{Some}, \texttt{owl:Thing}, \mathbf{True})$, $vp_0 := \mathbf{True}$. An initial zipper can then defined as $zipper_0 := \mathbf{NP}(np_0, \mathbf{Select}')$. It corresponds to the totally unconstrained query that returns the list of everything. A menu of application-specific initial zippers can be added according to frequent types of queries, and so that the users have less transformations to perform.

We continue by defining a number of zipper transformations. A zipper transformation is formally defined as a partial mapping from zipper to zipper. Transformations are denoted by all-uppercase names, and are defined by unions of mappings from a zipper pattern to a zipper expression. We strive to define our transformations in terms of the domain vocabulary rather than in terms of the AST constructs, so as to avoid exposing users to the latter. Therefore, our first transformations insert the primitive elements (i.e. nodes, determiners, classes, and properties) depending on the current focus.

$$
\begin{aligned}
\mathit{NODE}(n) &:= \mathbf{NP}(np, np') \rightarrow \mathbf{NP}(\mathbf{Node}(n), np') \\
\mathit{DET}(d) &:= \mathbf{NP}(\mathbf{DetThat}(det, class, vp), np') \rightarrow \mathbf{NP}(\mathbf{DetThat}(d, class, vp), np') \\
\mathit{CLASS}(c) &:= \mathbf{NP}(\mathbf{DetThat}(det, class, vp), np') \rightarrow \mathbf{NP}(\mathbf{DetThat}(det, c, vp), np') \\
&\quad \mid \mathbf{VP}(\mathbf{True}, vp') \rightarrow \mathbf{VP}(\mathbf{IsA}(c), vp') \\
\mathit{PROP}(p) &:= \mathbf{NP}(\mathbf{DetThat}(d, c, \mathbf{True}), np') \rightarrow \mathbf{VP}(\mathbf{Has}(p, np_0), \mathbf{DetThat}'_{\mathbf{3}}(d, c, np')) \\
&\quad \mid \mathbf{VP}(\mathbf{True}, vp') \rightarrow \mathbf{VP}(\mathbf{Has}(p, np_0), vp') \\
\mathit{PROP}^{-}(p) &:= \text{(same as } \mathit{PROP}\text{, replacing } \mathbf{Has} \text{ by } \mathbf{IsOf}\text{)}
\end{aligned}
$$

Other transformations allow the introduction of the Boolean connectors between noun phrases and verb phrases. They are defined in a generic way below, where $\mathbf{X}/x$ stands for both $\mathbf{NP}/np$ and $\mathbf{VP}/vp$. Transformations $\mathit{AND}, \mathit{OR}$ coordinate a substructure x with an initial AST x_0, and move the focus to x_0. Transformation NOT toggles the application of negation.

$$
\begin{aligned}
\mathit{AND} &:= \mathbf{X}(x, x') \rightarrow \mathbf{X}(x_0, \mathbf{And}'_{\mathbf{2}}(x, x')) \\
\mathit{OR} &:= \mathbf{X}(x, x') \rightarrow \mathbf{X}(x_0, \mathbf{Or}'_{\mathbf{2}}(x, x')) \\
\mathit{NOT} &:= \mathbf{X}(\mathbf{Not}(x), x') \rightarrow \mathbf{X}(x, x') \\
&\quad \mid \mathbf{X}(x, x') \rightarrow \mathbf{X}(\mathbf{Not}(x), x')
\end{aligned}
$$

$T(\mathbf{Node}(n))$	$:= NODE(n)$
$T(\mathbf{DetThat}(d,c,vp))$	$:= DET(d);CLASS(c);DOWN;T(vp);UP$
$T(\mathbf{IsA}(c))$	$:= CLASS(c)$
$T(\mathbf{Has}(p,np))$	$:= PROP(p);DOWN;T(np);UP$
$T(\mathbf{IsOf}(p,np))$	$:= PROP^{-}(p);DOWN;T(np);UP$
$T(\mathbf{True})$	$:= ID$
$T(\mathbf{And}(x_1,x_2))$	$:= T(x_1);AND;T(x_2);UP$
$T(\mathbf{Or}(x_1,x_2))$	$:= T(x_1);OR;T(x_2);UP$
$T(\mathbf{Not}(x))$	$:= T(x);NOT$

Figure 3. Recursive definition of the transformation sequence $T(x)$ from x_0 to AST x

A generic *DELETE* transformation can also be defined to undo insertions.
$DELETE := \mathbf{X}(x,x') \rightarrow \mathbf{X}(x_0,x)$

In order to allow transformations at an arbitrary focus, it is important to allow moving the focus through the AST. To this purpose, we define four transformations *UP*, *DOWN*, *LEFT*, *RIGHT* to move the focus respectively up to the parent AST node, down to the leftmost child, to the left sibling, and to the right sibling. We only provide the definitions for constructor $\mathbf{And}(np,np)$ as other constructors work in a similar way.

$$
\begin{aligned}
DOWN &:= \mathbf{NP}(\mathbf{And}(np_1,np_2),np') \rightarrow \mathbf{NP}(np_1,\mathbf{And}'_1(np',np_2))\\
UP &:= \mathbf{NP}(np_1,\mathbf{And}'_1(np',np_2)) \rightarrow \mathbf{NP}(\mathbf{And}(np_1,np_2),np')\\
&\;\;\mid\; \mathbf{NP}(np_2,\mathbf{And}'_2(np_1,np')) \rightarrow \mathbf{NP}(\mathbf{And}(np_1,np_2),np')\\
RIGHT &:= \mathbf{NP}(np_1,\mathbf{And}'_1(np',np_2)) \rightarrow \mathbf{NP}(np_2,\mathbf{And}'_2(np_1,np'))\\
LEFT &:= \mathbf{NP}(np_2,\mathbf{And}'_2(np_1,np')) \rightarrow \mathbf{NP}(np_1,\mathbf{And}'_1(np',np_2))
\end{aligned}
$$

Theorem 1 (completeness)**.** *The initial zipper* $zipper_0 = \mathbf{NP}(np_0,\mathbf{Select}')$ *and the above set of transformations is* complete*, assuming transformation* $NODE(n)$ *is suggested for all nodes of the target RDF graph, and similarly for transformations* $DET(d)$*,* $CLASS(c)$*,* $PROP(p)$*, and* $PROP^{-}(p)$*.*

Proof. First, it is easy to show that the moving transformations give access to all zippers of an AST. Therefore, to prove completeness, it is enough to prove that every zipper in the form $\mathbf{NP}(np,\mathbf{Select}')$ is reachable. Figure 3 defines a recursive function $T(x)$ that returns a transformation sequence from an initial AST x_0 to any AST x, where x stands for any AST datatype. For each case, it can be proved that the transformation sequence $T(x)$ indeed leads from x_0 to x, and that every recursive call is well defined (sub-structure x_0 at focus). *ID* is the identity transformation. □

The above proof provides an algorithm T for computing the transformation sequence leading to any AST x. The example zipper given in Figure 2 can be reached with the following sequence:

DET(**Some**); *CLASS*(`owl:Thing`); *DOWN*;
PROP^{-}(`dbo:release`); *DOWN*; *DET*(**Some**); *CLASS*(`dbo:Film`); *DOWN*;
PROP(`dbo:director`); *DOWN*; *NODE*(`dbr:Steven_Spielberg`); *UP*;
AND; *PROP*(`dbo:genre`); *DOWN*; *NODE*(`"SF"`); *OR*; *NODE*(`"Fantasy"`); *UP*; *UP*....

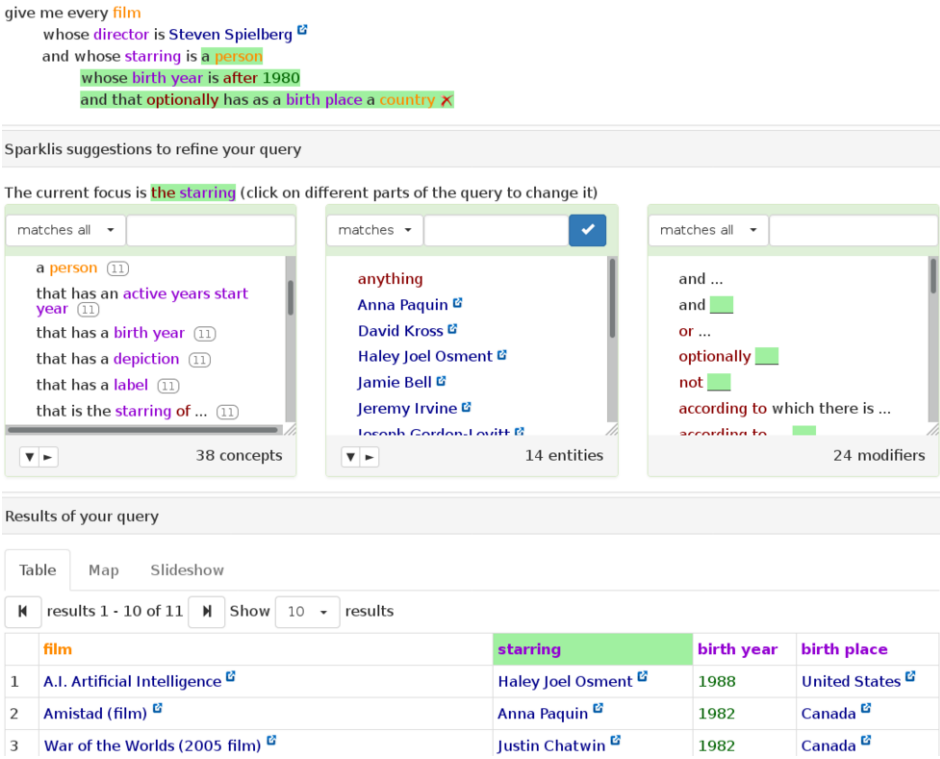


Figure 4. Screenshot of SPARKLIS

In practice, it appears useful to tune transformations so as to minimize the number of interaction steps for users: e.g., moving down after inserting a property, moving up after inserting a node, moving down when inserting a property at a noun phrase, avoiding the insertion of default elements (e.g., determiner **Some**, class `owl:Thing`). Applying those rules reduces the number of steps in the example from 19 to 9:

PROP^{-}(`dbo:genre`); *CLASS*(`dbo:Film`);
PROP(`dbo:director`); *NODE*(`dbr:Steven_Spielberg`);
AND; *PROP*(`dbo:genre`); *NODE*(`"SF"`); *OR*; *NODE*(`"Fantasy"`).

4. Implementation, Applications, and Evaluation

Our approach of authoring based on zippers and transformations is implemented in a functional programming language (OCaml), and has already been used in three applications: SPARKLIS[3] [5], UTILIS [7], and PEW [4]. Those applications target different tasks and formal languages (FL). SPARKLIS targets semantic search with SPARQL; UTILIS targets descriptions of RDF nodes; and PEW targets ontology design and completion with OWL class expressions. All include the conjunctive subset of the intermediate language defined in previous section, and extends it with task-specific constructs. In

[3]Online version at `http://www.irisa.fr/LIS/ferre/sparklis/` with examples and screencasts.

each application, the user interface has three parts (see Figure 4 for a screenshot): the current sentence and focus (top), the suggested transformations (center), and the semantic interpretation of the sentence (bottom). The latter is made of query answers for SPARKLIS, similar nodes for UTILIS, and class instances for PEW. The suggested transformations are selected so as to avoid semantically misformed sentences: e.g., empty results in SPARKLIS or inconsistent ontology in PEW. The suggested transformations are generally grouped in three categories: (a) classes and properties, (b) entities and values, (c) Boolean connectors and other transformations. The focus is highlighted in the sentence, and can be moved freely by clicking the relevant parts of the sentence.

User studies have been conducted in all three applications, and are reported in the related papers. SPARKLIS is online since April 2014, and more than 100,000 navigation steps have been performed by more than 1000 unique users. It has recently been officially adopted as a SPARQL query builder by two French institutions: Persée[4] and INIST[5]. A user study comparing UTILIS to Protégé has shown that users prefered the fine-grained suggestions of UTILIS to the static entity lists of Protégé. We have also observed that those suggestions improve consistency across RDF descriptions without the rigidity of a prescriptive schema. Another user study comparing PEW to Protégé has demonstrated promising results in terms of quantity, precision, and recall of the produced axioms, and in terms of usability. A notable result is the increase in recall, from 24% with Protégé to 56% with PEW, where 100% would mean a complete OWL formalization of the domain knowledge for the selected OWL fragment. Overall, our approach requires less background knowledge, and is more productive and safe compared to the direct use of formal languages or to the use of Semantic Web tools such as Protégé. The main difficulty appears to be in the understanding of the focus, and its impact on the suggested transformations. Some training is necessary for new users but most of them progress rapidly.

5. Conclusion and Perspectives

We have presented an alternative solution to the problem of CNL authoring. Like the auto-completion solution, it guides the user in the authoring process, and prevents the construction of misformed sentences. The novelty lies in the fact that the system suggestions are abstract syntax transformations rather than concrete words. As a consequence, the translation between the concrete and the abstract syntaxes goes backward, i.e. from the abstract to the concrete through a verbalization process that also applies to the suggested transformations. There are two benefits: (1) constructed sentences are complete at any step of the authoring process, allowing for intermediate results and feedback (*responsiveness*), and (2) editions can apply at any *focus*, i.e. any position or part of the sentence (*flexibility*). We have shown how to use Huet's zippers on the abstract syntax in order to represent the current *focus* and efficiently apply transformations. An interesting perspective is to implement this approach in Grammatical Framework (GF), in order to offer an alternative authoring process in applications. GF already offers the definition of abstract syntax, and verbalization (called *linearization* in GF). The definitions of contexts and zippers can be automatically derived from the abstract syntax, and zipper transformations can be naturally defined as functions on zippers.

[4] http://data.persee.fr/
[5] https://www.loterre.fr/category/explorer/

Acknowledgements. We thank the anonymous reviewers for their valuable feedback, and insightful suggestions, as they helped to improve this paper.

References

[1] ABBOTT, M., ALTENKIRCH, T., MCBRIDE, C., AND GHANI, N. ∂ for data: differentiating data structures. *Fundamenta Informaticae 65*, 1 (2005), 1–28.

[2] DOWTY, D. R., WALL, R. E., AND PETERS, S. *Introduction to Montague Semantics*. D. Reidel Publishing Company, 1981.

[3] FERRÉ, S. Bridging the gap between formal languages and natural languages with zippers. In *Extended Semantic Web Conf. (ESWC)* (2016), H. Sack et al., Eds., Springer, pp. 269–284.

[4] FERRÉ, S. Semantic authoring of ontologies by exploration and elimination of possible worlds. In *Int. Conf. Knowledge Engineering and Knowledge Management* (2016), LNAI 10024, Springer.

[5] FERRÉ, S. Sparklis: An expressive query builder for SPARQL endpoints with guidance in natural language. *Semantic Web: Interoperability, Usability, Applicability 8*, 3 (2017), 405–418.

[6] GUY, S., AND SCHWITTER, R. Architecture of a web-based predictive editor for controlled natural language processing. In *Controlled Natural Language* (2014), B. Davis, K. Kaljurand, and T. Kuhn, Eds., Springer, pp. 167–178.

[7] HERMANN, A., FERRÉ, S., AND DUCASSÉ, M. An interactive guidance process supporting consistent updates of RDFS graphs. In *Int. Conf. Knowledge Engineering and Knowledge Management (EKAW)* (2012), A. ten Teije et al., Eds., LNAI 7603, Springer, pp. 185–199.

[8] HITZLER, P., KRÖTZSCH, M., AND RUDOLPH, S. *Foundations of Semantic Web Technologies*. Chapman & Hall/CRC, 2009.

[9] HUET, G. Functional pearl – the zipper. *J. Functional Programming 7*, 5 (1997), 549–554.

[10] KALJURAND, K., AND KUHN, T. A multilingual semantic wiki based on attempto controlled english and grammatical framework. In *The Semantic Web: Semantics and Big Data*. Springer, 2013, pp. 427–441.

[11] KAUFMANN, E., AND BERNSTEIN, A. Evaluating the usability of natural language query languages and interfaces to semantic web knowledge bases. *J. Web Semantics 8*, 4 (2010), 377–393.

[12] KUHN, T. A survey and classification of controlled natural languages. *Computational Linguistics* (2013).

[13] LOPEZ, V., UREN, V. S., SABOU, M., AND MOTTA, E. Is question answering fit for the semantic web?: A survey. *Semantic Web 2*, 2 (2011), 125–155.

[14] NGOMO, A.-C. N., BÜHMANN, L., UNGER, C., LEHMANN, J., AND GERBER, D. Sorry, I don't speak SPARQL: translating SPARQL queries into natural language. In *WWW* (2013), pp. 977–988.

[15] PALMAZ, S., CUADROS, M., AND ETCHEGOYHEN, T. Statistically-guided controlled language authoring. In *Controlled Natural Language* (2016), B. Davis, G. J. Pace, and A. Wyner, Eds., Springer, pp. 37–47.

[16] RANTA, A. Grammatical framework. *Journal of Functional Programming 14*, 02 (2004), 145–189.

[17] SPARQL 1.1 query language, 2012. W3C Recommendation.

[18] TURNER, D. Functional programming and proofs of program correctness. In *Tools and Notions for Program Construction: An Advanced Course*, D. Néel, Ed. Cambridge University Press, 1982, pp. 187–209.

Controlled Natural Language
B. Davis et al. (Eds.)

doi:10.3233/978-1-61499-904-1-31

Automating Question Generation and Marking of Language Learning Exercises for isiZulu

Nikhil GILBERT[a] and C. Maria KEET[a,1]
[a] *University of Cape Town*

Abstract. Increase in isiZulu language learning is hampered by the predominantly manual approach to creating and marking homework and test exercises. Extant computer-assisted language learning platforms cannot handle the intricacies of agglutination in isiZulu and related languages. We seek to address this by designing a controlled natural language-based exercise generator and marker for isiZulu. This consists of question and answer sentence templates for exercise types, reusable algorithm snippets as grammar library, a small corpus of words and sentences to be used by the system, a constrained sentence generator to combine the right type of words, and finally the exercise creation and automated marking system. The preliminary evaluation shows encouraging results.

Keywords. Language learning; IsiZulu; Controlled Natural Language; Computer-Assisted Language Learning; Corpus

1. Introduction

Africa is culturally a diverse continent with hundreds of languages. Interactions between various African cultures throughout history have enriched the languages with various nuances and similarities [11]. South Africa has 11 official languages [20] and many residents are bilingual or multilingual, either due to home circumstances or through learning another language at school [5]. IsiZulu is the most widely spoken language of South Africa with 23% of the population speaking it as a first/home language [14] and about half of the population speaks it to some degree. Recently, the University of KwaZulu-Natal has introduced a compulsory introductory course in isiZulu for all its first-year students, out of a total enrollment of about 45000 students (i.e., roughly 9000 students per year). Other universities require this for selected degree programmes only, also for a very closely related language, such as isiXhosa, amounting to some 500-1000 students per year per university in South Africa.

There are only very limited learning resources for isiZulu, however, which are predominantly classroom and paper-based. The lack of computational resources for learning isiZulu resulted in the curriculum having been developed around teaching isiZulu using outmoded methods to assess students [5]. This also makes it difficult to find a means of widely distributing any new developments of language learning techniques. The manual marking of language learning exercises for such large amounts

[1] C. Maria Keet, Department of Computer Science, University of Cape Town, South Africa; E-mail: mkeet@cs.uct.ac.za.

of students has shown to cause many problems. The principal issues are: 1) prone to errors in marking, 2) loss of scripts, 3) time taken to return the work to students, and 4) limited options to assess the students' progression in language learning [18, 25]. In addition, many of the currently existing isiZulu corpora are dated or do not provide adequate material for properly studying a language [16, 28]. Automation may alleviate these issues. This requires a system that 1) controls, or limits, the language appropriately for the level of language learning, and 2) provides structured exercises that can be marked automatically. Research and tools for computer-assisted language learning (CALL) exercises focus more on either just vocabulary and sentence grammar [6, 7], advanced tasks such as reading comprehension, essay writing, and meta-cognition of the grammar [3, 8, 31], or crowd-sourced user input, such as on Duolingo. A common characteristic of the NLP-mediated approaches is that those languages are well-resourced, such as having POS taggers, parsers, analyzers, and grammars for sentence construction, to generate questions and answers for automated marking. Reuse of such existing tools is infeasible, not just because isiZulu is still underresourced, but because isiZulu is a Bantu language characterized by agglutination and its characteristic noun class system (NCS). That is, it has a very different morphology and grammar from Indo-European languages and it is the NCS that causes the steep learning curve for beginners: one needs to know the NCS even before looking up words in the dictionary.

We seek to address these issues by developing a controlled natural language (CNL)-based question generation and automated marking system that is inspired by pen and paper-based exercises used by teachers to date. This is realized by, mainly, embedding extant computational models of isiZulu grammar and morphology and repurposing isiZulu natural language generation algorithms of [2,15] to compute the question answers. The modular approach to question and answer template specification can enable teachers to easily assess students with varied exercises. The current system can generate almost 40000 unique question sentences, thanks to the templates and words taken from a newly designed small relevant corpus. From a language learning viewpoint, the questions principally address the issue of form exposure and practice, which, given the centrality of the NCS, is crucial for advancing to an intermediate level.

The remainder of the paper is structured as follows. We summarize some basic aspects of isiZulu and related works on CALL systems in Section 2. Section 3 describes the design of the CNL-based CALL system. Section 4 contains an evaluation. We reflect on the system in Section 5 and conclude in Section 6.

2. Background and Related Work

We first describe salient aspects of isiZulu, which helps contextualizing the related work on CALL systems and the requirements for them.

2.1. Some aspects of (Computational) Linguistics for isiZulu

There are a few algorithms for generating isiZulu words or text, being pluralisation of the isiZulu noun [2] and a CNL for ontologies that also includes conjugation [15]. Pluralising nouns is based on the noun class (NC) that a noun is classified in; e.g., the singular *indlovu* 'elephant' in NC9 is pluralized as *izindlovu* in NC10. The 17 isiZulu noun classes are listed by singular/plural pair in Table 1, from which rules were

developed that also include various deviations and exceptions. Due to sameness of prefixes for some NCs, the pluralizer uses both a morphological and semantic approach, through analyzing the first few letters of a noun in combination with the NC stored with the noun (not doing so reduces the accuracy to a mere 50% [2]). This also hints at a difficulty in isiZulu language learning, which is to figure out which noun class the noun is of, especially in those cases where the prefix is the same for the noun in the singular but different for the plural or vice versa; e.g., NC1 and NC3 have the same prefix *um(u)-* and NC10 can have either NC9 or NC11 as singular, which have different prefixes. There are also nouns that have the same stem but a different prefix and therewith obtain different meanings; e.g., *umuntu* 'human' and *ubuntu* 'humanity'.

After the NCS, the next main step at the introductory level is conjugation of the verb, which is governed by the NC of the subject by means of the so-called subject concord (SC), among other possible constituents that can be seen as a slot system [14]. Each NC has its own SC. For instance, with the concordial agreement underlined, *umuntu uhamba* 'the human goes' and, in the plural, *abantu bahamba*, but if, say, the elephant goes, it is *indlovu ihamba* (plural *izindlovu zihamba*). The negation is governed by the negative SC, which is also specific to each NC; e.g., *umuntu akahamba* 'the human does not go' and *indlovu ayihamba*. There are a few cases where there is the same negative SC for different NCs, which ups the exercise difficulty level; e.g., NC4 and NC9 both take *ayi-* [14]. Algorithmically, once the NC of the noun is fetched, adding the SC to the verb uses a NC-driven lookup table and checks for phonological conditioning required for the few vowel-commencing verb roots.

The verb can take various extensions, such as certain prepositions that are inserted in the verb and the wh-questions that always go at the end [14]. For instance, 'to work for' *ukusebenzela* (*uku-* 'to', *-sebenz-* verb root of 'work', *-el-* is the reciprocal extension, *-a* the final vowel) and *uvelaphi?* 'where [do you/does (s)he] come from?'. The NCS further governs prefixes of, among others, adjectives and possessives into a comprehensive concordial agreement system that is taught at an intermediate level. Therefore, an excellent command of the NCS is of paramount importance to advance in isiZulu and related languages that use a NCS.

While the multitude of all components are still being investigated, it is clear that this lends itself well for a modular building up of a CNL with a set of templates of varying complexity for learning exercises, where the teacher would be able to select which grammatical components should be included as well as which noun classes.

NC	Prefix	Examples	NC	Prefix	Examples
1 2	um(u) aba	*umuntu* 'human' *abantu*	9a (6)	i ama	*ivazi* 'vase' *amavazi*
1a 2a	u o	*ugogo* 'grandmother' *ogogo*	9 10	i(N) izi(N)	*indlovu* 'elephant' *izindlovu*
3a (2a)	u o	*ushizi* 'cheese' *oshizi*	11 (10)	u(lu) izi(N)	*uphawu* 'mark' *izimphawu*
3 4	um(u) imi	*umfoloko* 'fork' *imifoloko*	14	ubu	*ubuhle* 'beauty'
5 6	i(li) ama	*igama* 'name' *amagama*	15	uku	*ukuhamba* 'to go'
7 8	isi izi	*isilwane* 'animal' *izilwane*	17	ku	(locatives)

Table 1. isiZulu noun classes with prefixes for each noun class with examples (based on [2]); the odd-numbered classes are the singular and even numbers are their plural and the 'N' denotes 'n', 'm', or ''.

2.2. Computer-Assisted Language Learning

Principles of integrative CALL are described in [17], integrating reading, writing, speech, and listening to deliver lessons. Random generation of questions is an important part of CALL systems, as repeated texts suffer from a phenomenon known as the "practice effect" [23] when the same material is presented to a user repeatedly.

There are typically three components of a CALL system: a grammar rule set [26] and sentence generator, the contextually relevant corpus [10, 17], and the language learning exercises that may have hints and answers [26]. The rule set of the language's grammar has to be established first, as the number of language learning exercises may be constrained by it [26]. The grammar rules need to consist of, at least, a series of computationally modelled parts of speech (POS), especially the noun and verb [4]. With a 'slot system' to add and remove components, a learning platform would then be only limited to a teacher's ingenuity.

The language learning exercise question set should be independent of the rule base so as to avoid repetition in the code and avoid confusion in future parallel development [13]. The questions need to challenge learners also in the basic and mandatory contexts [1], yet also respect the scope of pen-and-paper exercises, such as [29] for isiZulu.

The variability in questions normally comes from a sufficiently large corpus from which to draw words and sentence fragments. It is important that the corpus is contextually relevant and contains words of modern communication that can be used in every day speech [17, 27]. The corpus has to be annotated with the correct POS tags, which is subsequently used by the grammar rules [21]. This is a non-trivial step for isiZulu: the Ukwabelana corpus is tagged with a gold standard [28] but contains outdated texts [22], whereas the news item corpus with recent texts [16] is not POS tagged and there is no fully functional isiZulu POS tagger.

Natural language generation has been used in CALL systems. Closest to our scope, there is a French grammar and question bank for fill-in-the-blank questions and sentence shuffle [24], and question/answer sentence pairs with SemTAG (a Feature-Based Lexicalised Tree Adjoining Grammar) and transformation rules for whole sentences [7]. NLG also has been used to assist with essay writing [8]. There are more distantly related efforts, such as CNL-based question generation from texts for grammar concept questions [3] and reading understanding questions [31]. These types of questions are at a much more advanced level of language learning than the contents of currently available textbooks for isiZulu. Also, the demand for a computational approach for those type of questions is not nearly as pressing as the questions for thousands of students at the introductory level of language learning.

Currently, very few CALL systems for isiZulu (and Bantu languages more generally) are available that use an integrative learning strategy [12, 19]. We thus also consulted current paper-based systems [29] to transfer into automated templates.

3. Specification and Design

The platform was designed with a layered architecture, with each layer feeding either information, or services to the next, as can be seen in Fig. 1. A layered architecture is claimed to have the benefits of ease of scalability and the convenience of concurrent development [9]. The underlying idea of the present design is similar to [7] in the sense of source and target sentences and transformations, but it is specified as a CNL with

word component exercises and it uses algorithms to obtain the transformations, rather than tree transformations with a grammar (a formalisation of sentence grammar does not exist for isiZulu). We discuss each component in the following sections.

3.1. Grammar Rule Sets

Because isiZulu is an underresourced language, any resource developed should be easy to reuse in other systems. Therefore, the system has a separate support library for isiZulu grammar, which also includes support functions such as the randomized sentence generator. This ensures that no recoding is required for each new CALL exercise, as the set of methods and classes allow flexible reconfiguration of the constituents of the words and sentences.

The grammar rule set of the system consists of the following components, (harmonized into Java): a new singularizer by 'reversing' the pluraliser algorithm; the pluralizer of [2]; the verb grammar represented as a CFG [14] with a substantial new set of verb roots; the algorithm for verb conjugation from a CNL for ontologies [15]. These are sufficient to create basic sentences for the exercises and their answers (see below). Further, each morpheme can be obtained so that exercises may be constructed with individual grammar components only as well; e.g., getVerbGrammar("SC") will return a list of all subject concords. There are further functions, notably to generate a specific verb form, which allows the user to stipulate the morphological specificities of the verb they would like to generate, and to check consistency of the verb form with the CFG. For instance, if a verb has both a positive and negative subject concord, it violates the CFG, so the function will then only append affixes consistent with the positive subject concord. Current algorithms for isiZulu sentence generation are not perfect [15], which is largely due to the pluralizer that hovers between 92-100% accuracy depending on the test set [2].

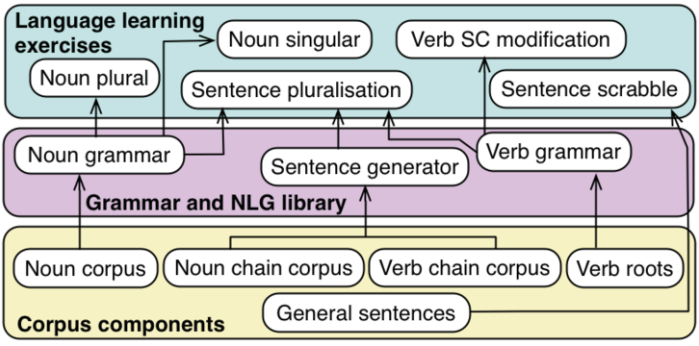


Figure 1. Architecture of the CNL-driven CALL system. The arrows indicate which upper layer components make use of the lower layer components.

3.2. Corpus Specifications

The two existing isiZulu corpora turned out to be unsuitable for the task. Ukwabelana [28] was contextually inadequate for it consists of the bible and novels, it contains outdated isiZulu, or did not contain words that were simple enough for novice language learning. The portion of the isiZulu National Corpus (INC) [16] turned out to be too

variant with the number of words. We therefore constructed a small corpus consisting of five lists of manually POS-tagged words and sentences, adhering to consistency in the style of annotation with other corpora. The corpus' content was also inspired from a beginner's language learning book in order to use words that were deemed most relevant for conversational context [29]. The corpus includes: a list of common nouns (n=231) and verbs (n=59) typical for language learning, such as *umfundi* 'learner', *ikhaya* 'home', *-enza* 'do', *-hamba* 'go', *thenga* 'like'; common phrases spoken in isiZulu (n=60); a morphologically analysed and tagged word list (n=10040); and the noun and verb chain lists for sentence generation that is described in the next section.

3.3. Sentence Generation Algorithm

We created a sentence generation algorithm that draws from the annotated corpus. The possible sentence structures supported in the system are either of the high-level pattern "<noun> <verb>" or "<noun> <verb> <noun>". In order to string the right words together, it uses two chained lists of words. The Noun Chain List is a list of 231 nouns, each of which has a list of one or more suitable verbs that can follow it; e.g., animals can 'eat' and 'drink' but a radio cannot. The generator will select a noun at random from the list, and then select verb at random from that chain. For each verb in Verb Chain List, there is a list of suitable nouns that also can be selected at random. It continues to follow these chains until a sentence with the specified structure is built. The list is annotated with whether the verb needs an object or not, as the latter can be used in the "<noun> <verb>" pattern, but the former cannot (indicated with <t>). For instance, given the examples in Figure 2, the algorithm selects *ubaba* (NC1), and picks *washa* 'wash' from the list of verbs in the Noun Chain List. Then in the Verb Chain List, *washa* lists what objects can be washed, such as *imoto* 'car' and thus can be used in the "<noun> <verb> <noun>" pattern (but need not).

Noun chain list	Verb chain list
ubaba <1a> washa;sula;faka;khuluma	washa <t> imoto;umshini;umnyango
umzali <1;s> ALL_v;e_dumisa;e_cisha	sula <> ifasitela;imoto;ipuleti
	khuluma <t> ALL_1;ALL_1a

Figure 2. Illustration of the Noun Chain and Verb Chain lists options: nouns may take some verbs, all verbs "ALL_v" with or without exceptions "e_", and verbs may go with some specified classes (e.g., "ALL_1") or with nouns in specific noun classes only, such as all people (NC1, NC1a) 'speak' *-khuluma*.

3.4. Language Learning Exercises

The language learning exercises are built using the corpus and grammar library and are alike existing pen-and-paper based isiZulu language learning challenges [29] and it draws inspiration from Duolingo's sentence scrabble [30]. The types of exercises include reordering scrabbled sentences, pluralizing and singularizing nouns with the correct prefixes, and modifying the verb's SC in a sentence based on the noun as well as changing positive/negative that requires learners to change the morpheme of the verb to be in agreement with the correct verb grammar rules. Figure 3 illustrates some of these exercises with questions and computed (correct) answers. For instance, questions of type 1 and 2 help the learner to spot wrong concordial agreement, provide a hint as to what the plural might be, and assist in discovering sound agreement between the noun prefixes and concords (e.g., *aba- ba-*, *isi- si-*).

1. Pluralise the subject:
Q: Umfowethu bayaphuza
A: Abafowethu bayaphuza
Q: Indlovu zidla ihlamvana
A: Izindlovu zidla ihlamvana

2. Pluralise the verb:
Q: Oyihlo utheleka
A: Oyihlo batheleka

3. Pluralise the sentence:
Q: Isitshulu simnandi
A: Izitshulu zimnandi
Q: Umfowethu usula inkomishi
A: Abafowethu basula izinkomishi

4. Positive/negative subject concord:
Q: Batotoba
A: Abatotobi

Figure 3. Illustration of sample exercises (questions and answers) the system is able to produce.

Regarding the templates, recall that the two sentence patterns are "<noun> <verb>" and "<noun> <verb> <noun>". The actual exercises templates have <noun> constructed from prefix[SG/PL] + stem and the verb is composed of [Negative]Subject Concord + VerbRoot + [Negative]FinalVowel, taking into account phonological conditioning in the agglutination. The examples in Figure 3 are based on the following template pairs, with the change for the answer underlined and the wrong concordial agreement indicated in italics in the question, where the answer is computed with the algorithms mentioned in Section 3.1:

1. Q: <*prefixSG*+stem> <*PLSC*+VerbRoot+FV>
   A: <prefixPL+stem> <PLSC+VerbRoot+FV>
   Q: <*prefixSG*+stem> <*PLSC*+VerbRoot+FV> <prefixSG+stem>
   A: <prefixPL+stem> <PLSC+VerbRoot+FV> <prefixSG+stem>

2. Q: <*prefixPL*+stem> <*SGSC*+VerbRoot+FV>
   A: <prefixPL+stem> <PLSC+VerbRoot+FV>

3. Q: <prefixSG+stem> <SGSC+VerbRoot+FV>
   A: <prefixPL+stem> <PLSC+VerbRoot+FV>
   Q: <prefixSG+stem> <SGSC+VerbRoot+FV> <prefixSG+stem>
   A: <prefixPL+stem> <PLSC+VerbRoot+FV> <prefixPL+stem>

4. Q: <PLSC+VerbRoot+FV>
   A: <PLNEGSC+VerbRoot+NEGFV>

The system also has question on making the noun singular given the plural (in a short sentence) and to turn a negated verb into the positive, i.e.:

5. Q: <*prefixPL*+stem> <*SGSC*+VerbRoot+FV>
   A: <prefixSG+stem> <SGSC+VerbRoot+FV>

6. Q: <PLNEGSC+VerbRoot+NEGFV>
   A: <PLSC+VerbRoot+FV>

Thanks to the modularized approach of the grammar to facilitate flexible question generation, one also could integrate in a template, say, pluralisation of the sentence combined with positive/negative subject concord; i.e., question/answer pair templates:

Q: <prefixSG+stem> <SGSC+VerbRoot+FV> <prefixSG+stem>
A: <prefixPL+stem> <PLNEGSC+VerbRoot+NEGFV> <prefixPL+stem>

Such a question sentence may be, e.g., *umfowethu uwasha inkomishi* '(my) brother washes the cup' and the requested negative plural is then *abafowethu abawashi izinkomishi* '(my) brothers do not wash the cups'.

In order to separate the core engine from any interface, the system provides the POS tags and noun class tags with the question and computed model answer. This allows further flexibility in display, and, moreover, answer hints. For instance, it could be used to reveal the noun class of the noun to help the learner in figuring out what the plural/singular prefix or subject concord is for that noun, or use aforementioned getVerbGrammar("SC") to provide a hint listing all the possible subject concords.

4. Implementation and basic evaluation

The current system can generate 39501 unique question sentences of two or three words and compute their answers, and scrabble 60 general common conversational sentences. The source code is available at http://www.meteck.org/sw/callCodeZU.zip.

To test the accuracy of the system's output of generating the controlled sentences, we used an oracle, i.e., an isiZulu speaker, to check the appropriate aspects of the text that was generated (pluralization and conjugation have been evaluated [2,15]). A linguist was consulted in the detailed analysis of the results. The meaningfulness of the sentences and the grammatical correctness were the two proxies to determine whether a generated sentence was valid. Accuracy testing was carried out by generating 30 sentences (15 singular, 15 plural) covering each type of template and evaluating its outcome, from which the percentage was calculated, weighing each sentence equally. A sentence received a point only if it was completely free of errors of the particular category being assessed, any semantic or grammatical errors would be grounds for inadequacy. There was space for comments on each sentence.

The raw results were 100% semantically meaningful and 96% grammatically correct for two-words sentences, and (at a first pass) 63% semantically meaningful and 58% grammatically correct for three-word sentences. The single 'error' of a two-word sentence was due to omission of ticking the box of grammatical correctness (as it was checked as semantically correct). The primary reasons for the lower accuracy in the three-word sentences were the words in the corpus and the ported pluraliser and conjugator. For instance, *ushukela* 'sugar' does not have a plural and *-enza* 'do' requires additional phonological condition, which are regular exceptions that are correctly handled in [2, 15]. Others are due to the limited rules in the pluraliser; e.g., NC2 normally takes *aba-*, except when the noun refers to groups of some ethnicities or culture (then the plural prefix is *abe-*), which was not covered in [2]. These issues affected 5 sentences. Debatable words from the corpus in the test sentences are, e.g., *-sheka*, of which it is unclear whether it exists in its own right (meaning: defecate, to be scared, or to commit something) or is an acceptable (or not) colloquial contraction of *shiyeka* 'stay behind', and whether *udadewenu* 'your sister' should be spelled as such

or as *udade wenu*. These issues count for 7 cases, which a CALL system is not expected to resolve, but is for linguists and speakers of isiZulu to decide upon. Thus, the CNL templates function exactly as intended, the underlying algorithms perform mostly well, and the word chaining process also works well.

5. Discussion

The NV and NVN template structures with pluralization and negation may look simple from the perspective of isolating languages. For instance, in English, negation amounts to simply 'does not' or 'do not' regardless who or what the subject is and regardless the morphology of the verb. For verb negation in isiZulu, there are 12 singular NCs + 9 plural NC combinations with singulars + 6 personal pronouns = 27 negative SCs to consider and then to remember a set of phonological conditioning rules. Put differently, the range of templates may seem small, but the variability of what can possibly be slotted in is much higher. This thus also entails that canned questions and answers are not feasible even with the current 6 types and 12 templates (aside from shortcomings of not being able to extend it easily).

Overall, the system is a step in the direction of providing many more exercises to the thousands of isiZulu language learners. It already solves the resource issues of limited question sets, of time to mark (instantly cf. weeks), and script loss. We have conducted preliminary experiments with assigning difficulty levels to the exercises—which is integrated in the system presented here—that aims to contribute to assessing the learner's level and progress.

6. Conclusions

Computer-assisted language learning exercises for isiZulu were designed based on novel templates, a small corpus, and algorithms to compute the answers that adhere to the specified answer templates. The agglutinative nature of isiZulu lends itself well for a modular approach so that new templates easily can be configured from existing components of both template elements and of the algorithm snippets for a particular morphological unit. The system has 100% semantic accuracy for two-word isiZulu sentences, but leaves some room for improvement for three-word sentences.

Exercise extensions include the object concord and past tense, a larger corpus, and more comprehensive testing.

Acknowledgements
We thank Zola Mahlaza and Langa Khumalo for their feedback on the exercises and vocabulary, and a section of the INC. This work is based on the research supported in part by the National Research Foundation of South Africa (Grant Number 93397).

References

[1] S. Braun. From pedagogically relevant corpora to authentic language learning contents. *ReCALL*. **17** (2005), 47-64.

[2] J. Byamugisha, C. M. Keet and L. Khumalo, Pluralizing Nouns in isiZulu and Related Languages. In *Proc. of CICLing'16.* Springer LNCS 9623 (2018), 271-283.
[3] M. Chinkina and D Meurers. Question generation for language learning: from ensuring texts are read to supporting learning. *Proc of BEA-12*, Copenhagen, Denmark, Sept 8, 2017.
[4] L. M. da Costa, F. Bond, and X. He, Syntactic Well-Formedness Diagnosis and Error-Based Coaching in Computer Assisted Language Learning using Machine Translation. In *Proc. of NLPTEA2016*. (2016), 107-116.
[5] P. Diab, M. Matthews, and R. Gokool, Medical students' views on the use of video technology in the teaching of isiZulu communication, language skills and cultural competence. *African Journal of Health Professions Education* **8** (2016), 11.
[6] G. Duman, G. Orhon, and N. Gedik, Research trends in mobile assisted language learning from 2000 to 2012. *ReCALL* **27** (2015), 197-216.
[7] C. Gardent and L. Perez-Beltrachini. Using FB-LTAG Derivation Trees to Generate Transformation-Based Grammar Exercices. *Proc. of TAG+11*, Sep 2012, Paris, France. pp117-125, 2012.
[8] K. Harbusch, G. Itsova, U. Koch and C. Kühner. The Sentence Fairy: a natural-language generation system to support children's essay writing. *Computer Assisted Language Learning*, **21** (2008), 339-352.
[9] D. Hatley, P. Hruschka and I. Pirbhai, Process for system architecture and requirements engineering. Addison Wesley. (2013), 45-47.
[10] M. A. Hearst, Can Natural Language Processing Become Natural Language Coaching? In *Proc of ACL 2015 and IJCNLP 2015 (Volume 1: Long Papers)*. ACL (2015) **1**, 1245-1252.
[11] B. Heine and D. Nurse, *African languages: An introduction.* Cambridge University Press, Cape Town, 2000.
[12] HelloTalk, Free Language Exchange, *Language Partners: 2017.* https://www.hellotalk.com/. Accessed: 2017- 09- 30.
[13] C. Henderson, *Building scalable web sites*. O'Reilly California, Sebastopol, 2006.
[14] C. M. Keet and L. Khumalo, Grammar rules for the isiZulu complex verb. *Southern African Linguistics and Applied Language Studies*, **35**(2) (2017), 183-200.
[15] C. M. Keet and L. Khumalo, Toward a knowledge-to text controlled natural language of isiZulu. *Language Resources and Evaluation* **51** (2017), 131-157.
[16] L. Khumalo, Advances in Developing corpora in African languages. *Kuwala*, **1** (2015), 21-30.
[17] L. Kuang-wu, English teachers' barriers to the use of computer-assisted language learning. *The Internet TESL Journal* **6** (2000), 1-8.
[18] C. Lai and W. Kristonis, The advantages and disadvantages of computer technology in second language acquisition. *National Journal for Publishing and Mentoring Doctoral Student Research*, **3**(1) (2006), 6p.
[19] Language-Learning Software and Online Courses, *Transparent Language: 2017.* https://www.transparent.com/. Accessed: 2017-10-01.
[20] S. Makoni, From misinvention to disinvention of language: Multilingualism and the South African Constitution. *Black linguistics: Language, society, and politics in Africa and the Americas.* (2003), 132-155.
[21] F. Mishan, Authenticating corpora for language learning: a problem and its resolution. *ELT Journal* **58** 3 (2004), 219-227.
[22] B. Ndaba, H. Suleman, C.M. Keet and L. Khumalo. The Effects of a Corpus on isiZulu Spellcheckers based on N-grams. *IST-Africa 2016*. IIMC. May 11-13, 2016, Durban, South Africa.
[23] H. Pashler and G. Baylis, Procedural learning: I. Locus of practice effects in speeded choice tasks. *Journal of Experimental Psychology: Learning, Memory, and Cognition* **17** (1991), 20-32.
[24] L. Perez-Beltrachini, C. Gardent, and G. Kruszewski. Generating Grammar Exercises. *7th Workshop on Innovative Use of NLP for Building Educational Applications, NAACL-HLT*, Jun 2012, Montreal, Canada. pp.147-157, 2012.
[25] M. Prabitha, 2010. Computer assisted language learning: Benefits and barriers. *Journal of Literature, Culture and Media Studies* **2** (2010), 59-71.
[26] A. M. G. Sanz, M. Levy, F. Blin and D. Barr, *WorldCALL: Sustainability and computer-assisted language learning*. Bloomsbury Publishing. London. 2015.
[27] J.M. Sinclair (Ed.). How to use corpora in language teaching. *John Benjamins Publishing* **12** (2004).
[28] S. Spiegler, A. Van Der Spuy and P.A Flach, August. Ukwabelana: An open-source morphological Zulu corpus, *In Proc. of COLING 2010*, ACL, 1020-1028.
[29] N.S Turner, *isiZulu Sokuzwana*. University of KwaZulu-Natal. Durban, South Africa, 2011.
[30] R. Vesselinov and J. Grego, J, *Duolingo effectiveness study*. Technical report, City University of New York, 2012.
[31] Y. Xu, A. Goldie and S. Seneff. Automatic question generation and answer judging: a Q&A game for language learning. *Proc of SLaTE 2009*. Warwickshire, UK, Sept 3-5, 2009.

Controlled Natural Language
B. Davis et al. (Eds.)

doi:10.3233/978-1-61499-904-1-41

Controlled Natural Languages for Hazard Analysis and Risk Assessment

Paul CHOMICZ [a], Armin MÜLLER-LERWE [b], Götz-Philipp WEGNER [b], Rainer BUSCH [b], and Stefan KOWALEWSKI [a]
[a] *RWTH Aachen University, Lehrstuhl Informatik 11 – Embedded Software*
[b] *Ford Research & Innovation Center Aachen*

Abstract. The hazard analysis and risk assessment (HARA) is a safety activity, which is performed during the concept phase of the functional safety standard ISO 26262. The results of this activity are usually documented by using a natural language. On the one hand, natural languages are expressive and powerful, but on the other hand, they are also ambiguous and complex. The usage of controlled natural languages (CNLs) is a means to reduce the drawbacks of natural languages. In this paper, we introduce controlled natural languages for the rationales of the three risk parameters: severity, exposure, and controllability to extend our set of CNLs for the HARA. In the first place, the application of controlled languages leads to more harmonized descriptions and rationales. Subsequently, an automatic processing based on these languages shall be implemented to enable the detection of inconsistencies across different HARA documentations.

Keywords. Controlled Natural Language, Hazard Analysis and Risk Assessment, Functional Safety, ISO 26262, Controllability, Exposure, Severity, Rationale, Risk Parameter

1. Introduction

The ISO 26262 is an international standard for functional safety of electrical or electronic systems within road vehicles that was published in 2011 [1]. One of the first activities according to the safety lifecycle of the ISO 26262 is the hazard analysis and risk assessment (HARA) [2].

The HARA is divided into three steps. It starts with the identification of all hazards that could be caused by a potential malfunctioning behavior. Then, all relevant operational situations and operation modes are determined in which the identified hazards could possibly occur. The combination of a certain hazard and the situation in which the hazard could occur is called hazardous event. The second step is the assessment of the hazardous event regarding its risk. It comprises the determination of the risk parameters: severity, exposure, and controllability. The rating of the severity reflects the estimated potential harm caused by the hazardous event. The exposure describes the probability of being in the corresponding situation, and the controllability reflects the ability of the driver or other traffic participants to avoid the potential harm. The last step is the assignment of the automotive safety integrity level (ASIL) and the definition of a safety goal. Based on the ratings for severity, exposure, and controllability, a corresponding ASIL

will be assigned to the hazardous event. The ASIL specifies the necessary level of risk reduction with ASIL A being the lowest and ASIL D the highest level. Additionally, the class quality management (QM) can be assigned. It states that no safety requirements have to be managed under the ISO 26262 safety lifecycle for this hazardous event. For every hazardous event with an ASIL assigned to it, a safety goal has to be defined. The safety goal is a top-level safety requirement defining how to prevent or mitigate the risk of the hazardous event [2].

The ISO 26262 standard defines the three risk parameters in a qualitative way that leaves room for interpretation. As a consequence and based on the fact that new systems share the same actuators causing the same malfunctions and similar hazards, it is challenging to ensure consistency of the risk assessments along with their rationales between HARAs developed by different teams.

To describe and record the identified hazardous events and the rationales for the risk classifications, a natural language is usually used. Natural languages are complex and ambiguous. Therefore, it might be the case that same or similar hazardous events or risk parameter rationales could be described using different wordings and phrases. This makes it more difficult to verify the consistency across several HARAs.

Our approach to tackle these problems is to apply controlled natural languages (CNLs) for the documentation. A CNL is based on a natural language and restricts the grammar or the vocabulary of it [3]. Therefore, it is a subset of a natural language. The restrictions intend to reduce or eliminate ambiguity and to improve machine processing [4]. The usage of controlled natural languages for the hazard analysis and risk assessment shall reduce the possibility to write similar or same hazardous events or rationales with different wordings and phrases. Furthermore, the languages might be used to enable an automatic check to detect inconsistencies between different HARAs.

The remainder of this paper is structured as follows. The next section describes related work and previous work that was made to achieve a controlled natural language for the hazardous event descriptions. Afterwards, the formalization process along with the resulting CNLs for the rationales of the three risk parameters severity, exposure, and controllability are presented. The paper concludes with an evaluation of the created languages and an outlook on future work.

2. Related Work

In a previous work [5], we have already developed a controlled natural language for the description of hazardous events. Along with means to perform a situation analysis [6], the CNL can be used to record the outcomes of the first step of the hazard analysis and risk assessment in a more formal way than only using a natural language.

The grammar of the controlled natural language for the description of hazardous events only allows to write the description in a bullet-point manner. Noun phrases (NPs) are used to describe an event, a situation, or a characteristic of a system. The headword of the phrase is a noun, and it can contain additional adnominals. Certain prepositions (IN) and conjunctions are used to connect several noun phrases to create more complex descriptions. The usage of pronouns, clauses, and verbs is prohibited to further reduce the complexity. In the depicted example in Fig. 1, two simple noun phrases are connected with a preposition resulting in a noun phrase and a prepositional phrase (PP) [13].

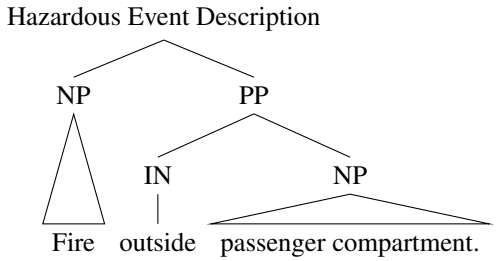


Figure 1. Hazardous event description written in the CNL [5]

The developed controlled natural language is based on a representative set of already finished HARAs that were documented using the English language. In a bottom-up and iterative approach, the documents were analyzed to find a common structure for the descriptions. From this common structure the CNL was created. The newly created language was applied in two ways. Firstly, it was used to describe all hazardous events which already exist in the HARAs that were analyzed. A large portion (51.9 %) of the existing descriptions was already compliant to the controlled language. The other portion could be translated by domain experts into semantically equivalent descriptions that are in line with the language. Secondly, the CNL was prototypically utilized to three newly created HARAs at the Ford Research & Innovation Center Aachen. After extending the vocabulary with domain-specific terms, all hazardous events could be described using the new language.

Several controlled natural languages have been developed for various domains and purposes [4,7]. Among these, a great number has been designed for requirements engineering like [8] or [9]. Requirements are rather specifying in contrast to the hazardous events and the risk parameter rationales which have a rather descriptive and justifying character.

Such kind of languages have been developed or used for technical documentations, like Caterpillar Technical English [10] or Bull Controlled English [11]. These languages usually consist of a simplified set of rules and a controlled vocabulary to improve understandability and translatability. During the creation of the new controlled natural languages for the hazard analysis and risk assessment, the presented best-practice features and the models for developing a new controlled natural language for technical documents were considered and applied [12].

3. Formalization Process

For the formalization of the rationales, the same process was applied as for the formalization of the hazardous event descriptions [5]. In two iterations, different sets of HARAs were analyzed to find a common structure for each rationale. At first, the single rationales were extracted from the documents, and duplicates were removed. Then, the rationales were analyzed in detail by determining word frequency statistics, the used parts of speech, and sentence structure statistics based on the part of speech tagging. All operations, like the part of speech tagging, were performed computer-assisted and double-checked manually.

The severity rationales have a similar structure as the hazardous event descriptions [5]. Therefore, the structure can be divided into two categories. The first category contains rationales written in a bullet-point manner, and for the second category, full sentences were used to formulate the rationales. In the first iteration, 166 different severity rationales were extracted from the given documents. The biggest part of the rationales belongs to the first category (59 %). Only a small portion (9.7 %) uses full sentences for the rationales. The remaining rationales were written in a mixed version of using full sentences and bullet-points (31.3 %).

In the second iteration, 114 additional severity rationales were extracted. For both iterations, the same documents were used as for the formalization of the hazardous event descriptions [5]. In this set, the portion of bullet-point rationales is bigger (62.9 %), and again, only a few were written using full sentences (14.9 %). The categorization was performed manually in both iterations.

An exemplary set of severity rationales is shown in Table 1 that represents how the rationales are currently described using the English language. The rationales 1 and 3 are written in a bullet-point manner, the rationale 2 is written as a full sentence, and the last rationale is described in a mixed version.

Table 1. Exemplary set of severity rationales

No.	*Severity Rationale*
1	Potential collision with surrounding traffic at low speed.
2	Pedestrian may be overrolled.
3	Potential lane departure due to unexpected yaw behavior. Potential crash into pedestrians or obstacles beside the road or side collision with oncoming traffic.
4	Vehicle may be moved into the path of oncoming traffic. Side collision with velocities greater than 35 kph possible.

Furthermore, we analyzed the exposure rationales and the controllability rationales in the same way and divided the single rationales into the two categories. In total, 351 different exposure rationales were extracted from 16 different HARAs and categorized. Nearly the half (45.6 %) is formulated in bullet-point manner, and a third (32.5 %) is written using full sentences. The remaining 22.9 % are stated in a mixed version. Table 2 shows an exemplary set of exposure rationales.

Table 2. Exemplary set of exposure rationales

No.	*Exposure Rationale*
1	Service situation. Frequency rated.
2	Failure can potentially occur during any driving situation.
3	Launch on a hill on μ-split occurs a few times a year for the great majority of drivers.
4	Stopped at intersection (E4); however lower probability with non-motorist crossing the road (E3), 1-10 % of average operating time.

The results of the analysis of the controllability rationales indicate an even stronger tendency for using full sentences to justify the chosen controllability value. From the 16 HARA documents, overall, 410 different controllability rationales were extracted. Only 7.8 % of the rationales are written in a bullet-point manner. The major part is stated in full sentences (84.6 %). Again, in the remaining rationales, bullet-point phrases and full

sentences are used together (7.6 %). The Table 3 contains some controllability rationale examples.

Since the focus has changed on using full sentences, we also analyzed the structure of the sentences in more detail for the exposure rationales and the controllability rationales. For both, the total number of rationales is significantly bigger than the number of severity rationales or hazardous event descriptions. This indicates a bigger variety.

Table 3. Exemplary set of controllability rationales

No.	*Controllability Rationale*
1	Difficult to control for an average driver.
2	Most of drivers will brake/steer to avoid collision.
3	The driver can reduce the acceleration request and/or is able to increase the steer angle.
4	Driver has to steer slightly and/or reduce throttle to control this situation. Situation is slightly better to control than with 2WD.

In the following, only the results of the sentence structure analysis of the controllability rationales are presented. Each sentence was considered separately, and in total 461 different sentences were analyzed. One fourth of the sentences contains at least one subordinate clause (25.6 %). The majority of the sentences is formulated in present tense (92.4 %). Only a small portion used the present progressive, the past tense, or the future tense. In addition to that, much more active voice (88.7 %) is used than passive voice. In 66.4 % of the sentences, modal verbs are used to express obligations or abilities.

Every sentence contains at least a subject and a predicate. In our case, the predicate of a sentence corresponds to the main verb and any auxiliaries (e.g. modal verbs or adverbs) that accompany it [13]. The dependents of the predicate were analyzed separately. In 80.5 % of the sentences, an object follows the predicate. The type of the object (e.g. direct object or prepositional object) was not further determined. Modifiers as the dependent of the predicate are used in 25 % of the analyzed sentences. Additionally, one fourth of the sentences contains an infinitive phrase (26.5 %) [14].

The results of the sentence structure analysis of the exposure rationales are nearly the same. The biggest differences were identified in the usage of modal verbs and infinitive phrases. Only in 20 % of the exposure rationales modal verbs are used and in 7.6 % infinitive phrases. The differences of the other values do not exceed a portion of 10 %.

4. Controlled Natural Languages

As a result of the formalization process, the controlled natural languages for the rationales of the three risk parameters are introduced in this section.

4.1. Severity Rationale

The severity parameter gives an estimation on the potential harm or damage that could be caused by the hazard in a specific operational situation. The structure of the rationales is similar to the hazardous event descriptions. Therefore, we decided to use the same grammar with minor extensions for the controlled natural language of the severity rationales. The rationales are written in a bullet-point manner. Nominal phrases are the cen-

tral elements of the grammar. They can be combined with conjunctions or prepositions to formulate more complex rationales. The usage of verbs and pronouns is prohibited.

The Listing 1 contains a simplified version of the grammar definition for the hazardous event descriptions and the severity rationales.

Listing 1: Hazardous event and severity grammar

```
 bulletPoints → (initPhrases (Conjunction phrases)* '.')+
  initPhrases → nominalPhrase adjunct*
      phrases → (nominalPhrase | adjunct) adjunct*
nominalPhrase → nounPhrase | gerundPhrase
      adjunct → prepoPhrase | compPhrase
   nounPhrase → Determiner? adjPhrase? Noun nominal*
 gerundPhrase → Determiner? adjPhrase? Gerund nominalPhrase?
    adjPhrase → Adverb* Adjective+ (Conjunction adjPhrase)*
      nominal → Noun | Gerund
  prepoPhrase → 'not'? Preposition nominalPhrase
   compPhrase → asPhrase | thanPhrase
     asPhrase → 'not'? 'as' adjPhrase
   thanPhrase → adjPhrase 'than' (adjPhrase | nominalPhrase)
```

The production rules of the grammar are written with a small initial letter and lexer rules with the first letter capitalized. Lexer rules contain a set of terminal words. In our case, a lexer rule contains all words of a part of speech that are contained in the vocabulary of the controlled natural languages. Terminal symbols are surrounded by single quotation marks.

A single description or rationale contains at least one noun or gerund phrase and can be extended by additional prepositional phrases or comparative phrases. Furthermore, it is possible to conjoin additional phrases with conjunctions to create longer sentences. A noun phrase contains at least one noun, and additional modifiers or a determiner can be subjoined. A gerund phrase might also have an optional determiner and an optional adjective phrase (AP) that are put in front of a gerund. After the gerund, an additional nominal phrase may be attached. The adjective phrase contains at least one adjective (JJ) which can be modified with preceded adverbs (RBs). Furthermore, it is possible to conjoin several adjective phrases with conjunctions.

Two different types of comparative phrases are part of the controlled language. The first type uses the word 'as' followed by an adjective phrase. The other type starts with an adjective phrase in comparative form followed by the word 'than' and either an adjective phrase, a noun phrase, or a gerund phrase [14].

The severity rationales 1 and 3 of Table 1 already conform to the grammar of the controlled natural language. Figure 2 shows the classification of the first example into the parts of speech [15].

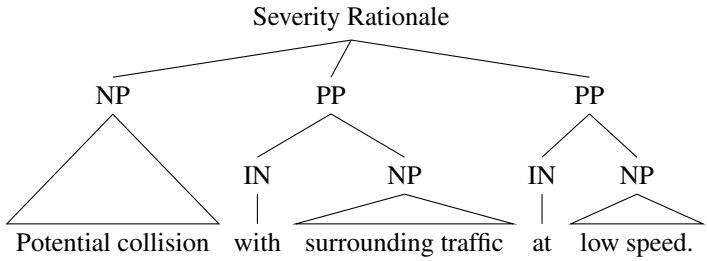


Figure 2. Severity rationale written in the CNL

4.2. Controllability Rationale

The controllability describes the ability of the driver or other traffic participants to retain sufficient control of the hazardous event to prevent the resulting harm. The major part of the rationales were formulated by using full sentences. The structure of the single sentences was analyzed resulting in the simplified grammar definition in Listing 2.

Listing 2: Controllability grammar

```
     sentences → (sentence (Conjunction sentence)* '.')+
      sentence → subject verbPhrase (infPhrase | compPhrase)*
       subject → subjectElement (Conjunction subjectElement)*
subjectElement → nominalPhrase prepoPhrase*
    verbPhrase → verb (object | adjPhrase)?
          verb → (ModalVerb | AuxiliaryVerb)? Adverb* Verb
        object → objectElement (Conjunction objectElement)*
 objectElement → (nominalPhrase | prepoPhrase) prepoPhrase*
     infPhrase → 'to' BaseFormVerb object? adjPhrase?
    compPhrase → adjPhrase 'than' adjPhrase? object
```

A rationale contains at least one sentence, but it is also possible to compound more sentences. Each sentence starts with a subject and a predicate. A subject element is a nominal phrase with optional prepositional phrases. The subject contains at least one subject element, and additional subject elements can be conjoined with conjunctions. The predicate is a verb phrase (VP) that starts with a verb. The verb might have an auxiliary verb or a modal verb to formulate abilities or obligations, and additionally it is possible to modify the verb with adverbs. Moreover, it is possible to extend the verb phrase with an object or an adjective phrase. An object element is a nominal phrase or a prepositional phrase that can be extended with additional prepositional phrases. Object elements can be conjoined with conjunctions.

Infinitive phrases (IP) and a comparison phrases (CP) can be appended to the sentence to construct more complex expressions. The infinitive phrase starts with the word "to" followed by a verb in its base form. Then, an optional object and an optional adjective phrase can be appended. The comparison phrase starts with an adjective phrase in

comparative form followed by the word ‘than’, another optional adjective phrase, and an object.

Figure 3 shows an example of a controllability rationale according to the controlled natural language. The example contains a single sentence with an object and an infinitive phrase.

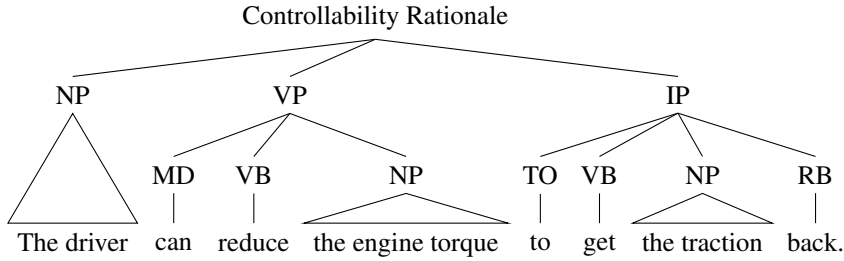


Figure 3. Controllability rationale written in the CNL

4.3. Exposure Rationale

The exposure parameter describes the estimation of the probability of being exposed to the hazard in terms of time and location. The rationale for justifying the selected value can be formulated using a combination of both created controlled natural languages.

Listing 3: Exposure grammar

```
rationale → (bulletPoints | sentences)+
```

Figure 4 shows an example of an exposure rationale that is according to the controlled natural language. The example contains a bullet-point phrase followed by a single sentence with an adjective phrase and an infinitive phrase.

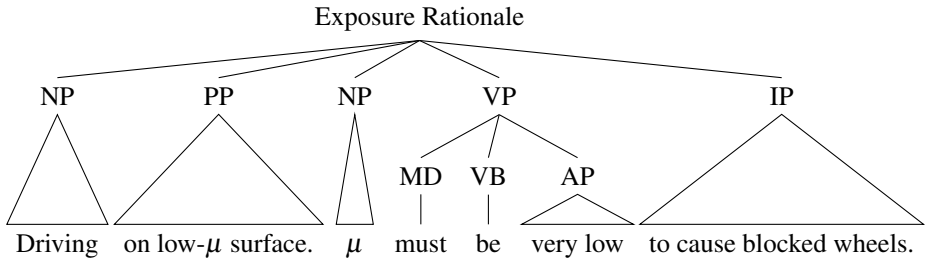


Figure 4. Exposure rationale written in the CNL

4.4. Vocabulary

The controlled languages for the hazardous event descriptions and for the three risk parameter rationales share the same vocabulary. It arose from the used terms in the given HARA documents. In a first version all used terms were added to the vocabulary. Afterwards, the vocabulary was partitioned into sets of terms with equivalent semantics [16]. For every set, only one representative was selected to remain as part of the vocabulary. Synonyms and terms with meanings that overlap widely enough are regarded as equivalent. The first part was determined by the help of existing thesauri, and the domain-specific meanings were specified by domain experts. The terms 'big' and 'large' are synonym, and for example the terms 'non-motorist' and 'child' are also equivalent in our domain-specific case. The reason for this is that the distinction between traffic participants should only be made between motorized and non-motorized participants. Further details like the age of the participants should not be taken into account while rating the hazardous event.

5. Evaluation

The newly created controlled natural languages for the rationales have been evaluated against the provided data to show that the languages are highly related to the already written rationales.

More than a half of the severity rationales (62.9 %) was written in a bullet-point manner. 116 out of these 176 rationales were compliant to the CNL (65.9 %). It was possible to translate another 45 severity rationales into a correct form by replacing a synonym with the corresponding word that is part of the vocabulary (25.6 %). The major part of the controllability rationales was already formulated using full sentences (84.6 %). Again, after replacing the synonyms, 126 out of 347 rationales are in line with the created CNL (36.3 %). More than the half of the 351 exposure rationales is already conform to the new controlled language (54.7 %). The remaining rationales of the three risk parameters can all be translated into a semantically equivalent version.

The translations were performed manually, and in the following, two examples are shown for rationales written in a bullet-point manner and using full sentences. The severity rationale in row 4 of Table 1 is not conform to the controlled natural language. One possibility is to translate each sentence separately resulting in the rationale "Vehicle movement into the path of oncoming traffic. Possible side collision with speed greater than 35 km/h.". Another possibility is to describe the relation between these two sentences in more detail. The second sentence is a consequence of the first one, and therefore, the translation "Possible side collision with speed greater than 35 km/h due to vehicle movement into the path of oncoming traffic." might be better.

Row 1 of Table 3 contains a controllability rationale that is not correct with respect to the new CNL. The rationale is not a complete sentence since the subject and the verb are missing. Adding the missing parts results in the correct sentence "The situation is difficult to control for an average driver.".

In addition, the new languages were prototypically applied in hazard analyses and risk assessments for new systems within the domains steering, fuel cell, and powertrain to make first experiences in a productive usage just like the CNL for the hazardous event

descriptions [5]. The same results were made for the languages of the rationales. After extending the vocabulary, it was possible to write the rationales conform to the CNLs.

6. Conclusion and Outlook

The formalization of the rationales for the severity, exposure, and controllability classification extends the set of controlled natural languages for the hazard analysis and risk assessment activity according to ISO 26262. During the analysis of the severity rationales, it turned out that the structure of the rationales is similar to the hazardous event descriptions. Therefore, it was possible to reuse the controlled natural language for the hazardous event descriptions [5].

The controllability rationales differ in their structure comparing to the severity rationales and hazardous event descriptions. The major portion is written in full sentences. The single sentences were further analyzed to determine a common structure. Based on these results, a new controlled natural language for justifying the chosen controllability parameter was developed.

The structure of the exposure rationales is bipartite. Bullet-point phrases and full sentences were nearly equally used to reason the exposure value. Thus, it was possible to use a combined version of the two controlled natural languages.

Altogether, two different controlled natural languages were developed. The bullet-point manner controlled natural language (BP-CNL) is used to describe the hazardous events and severity rationales. The full sentence controlled natural language (FS-CNL) enables to formulate the controllability rationale in a structured way. A combination of both languages serves as the formalization of the exposure rationales. The languages share the same vocabulary, which evolved from the given HARA documents and needs to be extended beyond this scope.

During the evaluation, it was possible to translate every rationale of the provided HARA documents into the respective controlled natural language as exemplarily shown. The manually performed translation example of the severity rationale shows that it is possible to translate it into two different correct versions depending on the understanding. The second translation connects the two sentence in a semantically way, whereas the first translation keeps the two sentences unrelated. This example shows that further means need to be developed to be able to determine the similarity for sentences of the controlled natural language. In this case, the used words are nearly the same in the two rationales, which might be a first simple and suitable method to calculate a similarity score.

In a next step, the set of CNLs shall be implemented within a prototype tool to simplify the usage. The prototype tool can then be used to further examine and improve the concept of the languages. Furthermore, a case study should be performed to gather more user experiences and to show the benefits of the concept.

References

[1] International Organization for Standardization: ISO 26262: Road Vehicles – Function Safety (2011)
[2] International Organization for Standardization: ISO 26262-3: Road Vehicles – Function Safety – Part 3: Concept Phase (2011)

[3] Kittredge, R. I.: Sublanguages and Controlled Languages. In: The Oxford Handbook of Computational Linguistics, pp. 403–447. 2nd edition (2003)
[4] Kuhn, T.: A Survey and Classification of Controlled Natural Languages. Computational Linguistics, 40(1):121–170 (2014)
[5] Chomicz, P., Müller-Lerwe, A., Wegner, G.-P., Busch, R., Kowalewski, S.: Towards the Use of Controlled Natural Languages in Hazard Analysis and Risk Assessment. Automotive-Safety & Security-Sicherheit und Zuverlässigkeit für automobile Informationstechnik, pp. 163–174 (2017)
[6] Jang, H. A., Kwon, H. M., Hong, S.-H., Lee, M. K.: A Study on Situation Analysis for ASIL Determination. Journal of Industrial and Intelligent Information, 3(2):152-157 (2015)
[7] Pool, J.: Can Controlled Languages Scale to the Web?. In: Proceedings of the 5th Int. Workshop on Controlled Language Applications (2006)
[8] Tommila, T., Antti, P.: Controlled Natural Language Requirements in the Design and Analysis of Safety Critical I&C Systems. SAFIR2014 Reference Group 2 (2014)
[9] Luo, Y., van den Brand, M.G.J., Kiburse, A.: Safety Case Development with SBVR-based Controlled Language. In: Model-Driven Engineering and Software Development. pp. 3–17 (2015)
[10] Kamprath, C., Adolphson, E., Mitamura, T., Nyberg, E.: Controlled Language for Multilingual Document Production: Experience with Caterpillar Technical English. In: Proceedings of the Second International Workshop on Controlled Language Applications, pp. 51–61 (1998)
[11] Lee, A.: Controlled English with and without Machine Translation. In: Aslib Proceedings, 46(5):131–133 (1994)
[12] Crabbe, S.: Controlling Language in Industry: Controlled Languages for Technical Documents. Palgrave Macmillan (2017)
[13] Meyer, P. G. et al.: Descriptive English Linguistics. Gunter Narr Verlag (2008)
[14] Radford, A.: An Introduction to English Sentence Structure. Cambridge University Press (2009)
[15] Brill, E.: Part-of-Speech Tagging. Handbook of Natural Language Processing, pp. 403–414 (2000)
[16] Svenonius, E.: Design of Controlled Vocabularies. Encyclopedia of Library and Information Science, 45(10):82–109 (1989)

Controlled Natural Language
B. Davis et al. (Eds.)

doi:10.3233/978-1-61499-904-1-52

Using the AIDA Language to Formally Organize Scientific Claims

Tobias KUHN
Department of Computer Science, Vrije Universiteit Amsterdam, The Netherlands

Abstract. Scientific communication still mainly relies on natural language written in scientific papers, which makes the described knowledge very difficult to access with automatic means. We can therefore only make limited use of formal knowledge organization methods to support researchers and other interested parties with features such as automatic aggregations, fact checking, consistency checking, question answering, and powerful semantic search. Existing approaches to solve this problem by improving the scientific communication methods have either very restricted coverage, require formal logic skills on the side of the researchers, or depend on unreliable machine learning for the formalization of knowledge. Here, I propose an approach to this problem that is general, intuitive, and flexible. It is based on a unique kind of controlled natural language, called AIDA, consisting of English sentences that are atomic, independent, declarative, and absolute. Such sentences can then serve as nodes in a network of scientific claims linked to publications, researchers, and domain elements. I present here some small studies on preliminary applications of this language. The results indicate that it is well accepted by users and provides a good basis for the creation of a knowledge graph of scientific findings.

Introduction

It is increasingly difficult for scientists to keep up with the rapidly growing literature, and science is in a genuine communication crisis [1]. Text mining has been the favorite approach to address this problem, but results remain far from perfect even for basic tasks, such as entity recognition [2] and extraction of simple relations [3], and such approaches have therefore failed to mimic human capacity to understand texts describing complex results. As an alternative solution, annotation approaches [4] have been proposed that require humans in the loop to manually attach formal representations and links to existing articles. Such annotations are, however, quite complicated to create, typically apply restrictions on what can be expressed, and can often be understood only in the larger context of the underlying article text.

In earlier work [5], I sketched an alternative approach that goes beyond annotation, is simple and intuitive, is fully general, and leads to representations that can be linked to scientific articles but are completely independent entities.

Specifically, this approach is based on the simple idea that we can use single English sentences to capture scientific findings and hypotheses, which then form the nodes in a network of scientific knowledge. My previous work introduced the concept of AIDA sentences [5], which are defined as English sentences that are: **Atomic:** a sentence describing one thought that cannot be further broken down in a practical way; **Independent:** a sentence that can stand on its own, without external references like "this effect" or "we"; **Declarative:** a complete sentence ending with a full stop that could (at least in theory) be true or false; and **Absolute:** a sentence describing the core of a claim ignoring the (un)certainty about its truth and ignoring how it was discovered (without phrases such as "probably" or "evaluation showed that").

The language of AIDA sentences thereby forms a Controlled Natural Language (CNL) [6]. Its approach is based on the assumption that virtually every scientific hypothesis can be represented as such a sentence. In the future, we could ask from researchers to publish their results in such a format from the start, and these AIDA sentences can then serve as nodes in a growing network of scientific claims and as a basis for the formal representation of their content, following our proposed vision of genuine semantic publishing [7].

Some examples of AIDA sentences are shown here:

- "A combination of system and searcher biases lead search engine users to settle on the incorrect answer to yes/no-questions around half of the time." (from https://doi.org/10.1145/2484028.2484053)
- "Teenagers reply on average faster to emails than adults." (from https://doi.org/10.1145/2736277.2741130)
- "Deep learning is a powerful and accurate method for automatic speech recognition." (from https://doi.org/10.1109/ASRU.2011.6163930, https://doi.org/10.1109/MSP.2012.2205597, and https://doi.org/10.1109/ICASSP.2013.6639347)

These examples illustrate the benefits of the different requirements of the AIDA approach. Atomicity ensures that each sentence is as short and concise as possible. Independence allows us to interpret and understand these sentences without further context (the sentences above can be understood without looking at the references we provided). Declarativeness gives them a common form and allows us to categorize them as true or false, or any degree of uncertainty in between. Absoluteness, finally, contributes to normalizing the sentences, thereby allowing us to use the same identifier (i.e. AIDA sentence) for reported results that only differ in their uncertainty or method of discovery, as exemplified by the last sentence above. These degrees of certainty and these methods of discovery are of course important too, but they are relatively easy to record with classical formal methods and formally linked to AIDA sentences. Various ontologies have in fact been proposed for these aspects (e.g. [8] and [9]).

These properties make an AIDA sentence highly reusable, and we can treat it as an anchor to formally link, for example, papers that claim or refute it. We can also link the sentences among each other, such as stating that a given AIDA sentence is more specific or more general than another one, or has the same meaning. Moreover, we can allow for these AIDA sentences themselves to be

partially or fully specified in a formal logic language like RDF, thereby allowing for a full continuum from informal over semi-formal to fully formalized statements, as proposed in our earlier work [10].

In previous work, we also presented two studies on the manual and automatic creation, respectively, of AIDA sentences in the biomedical field [5]. These studies showed that manual creation of AIDA sentences by untrained researchers as well as their automatic creation from an existing biomedical data source can be performed in either case in an effective and accurate manner. In both cases, about 70% of the created AIDA sentences received a perfect quality score. In a follow-up study, we worked on the extraction of AIDA sentences from paper abstracts with a simple rule-based approach [11].

AIDA is certainly not the first controlled natural language that aims to improve the way how science is conducted and communicated. In fact, very first English-based CNL, Basic English, was designed around 1930 to improve the global communication in science [12], among other spheres such as politics and economy. However, Basic English had no relation to formal methods and automatic knowledge organization, but only dealt with inter-human communication. Formally precise CNLs have been proposed more recently to improve the communication of scientific results [13] and to provide intuitive yet powerful query interfaces to researchers [14,15]. These approaches have however quite narrow application ranges that are limited by the expressiveness of the underlying logic formalism and the coverage of the used vocabularies or ontologies. AIDA is unique in the sense that it aims to support formal knowledge representation, while focusing on expressiveness rather than precision [6].

Data

In past few years, I have been building a small dataset of hand-curated AIDA sentences, which serves as the basis for the small studies to be introduced below. The two studies in the biomedical field introduced above [5] provide us with the first batch of AIDA sentences. The manual AIDA extraction study created 51 AIDA sentences and the automatic extraction created another 189 of them, therefore 240 AIDA sentences in total. They all come with an identifier of the associated publication they were extracted from.

The next sets of AIDA sentences come from two small case studies on meta-reviews in different domains. The first of these meta-reviews is a Cochrane Library report entitled "Cholinesterase inhibitors for Alzheimer's disease" (https://doi.org/10.1002/14651858.CD005593), summarizing evidence and findings from a number of publications on the topic. I manually created a network of AIDA sentences and the relevant publications based on the information found in the meta-review report. Figure 1 shows the main part of the resulting network, with three levels of AIDA sentences (more general ones at the top), and the lowest level being linked to the individual publications. The full data contains 62 AIDA sentences and can be found online.[1]

[1] https://github.com/tkuhn/aida/blob/master/usecases/alzheimers.md

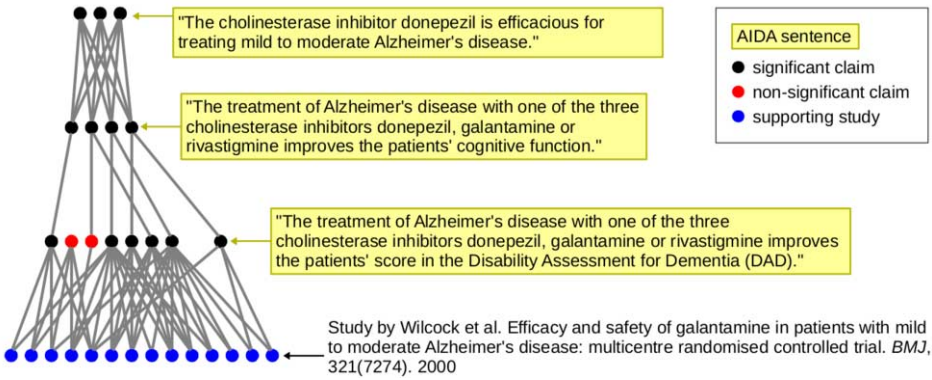


Figure 1. Part of the network of AIDA sentences and publications from the Alzheimer's case study. The links between AIDA claims connect a more specific claim (bottom) to a more general one (top).

The second case study targeted a less formal meta-review report published by SPARC Europe[2] on the question of whether Open Access publications enjoy a citation advantage. This general question can be expressed as the AIDA sentence "Open Access publications receive on average more citations than similar publications that are not Open Access". Most covered publications, however, investigate a narrower claim such as "Open Access publications *in astronomy and physics* receive ...". Overall, this second case study created AIDA sentences for 70 publications (only one paper could not be found and had to be excluded). The details of this study can be found online as well.[3]

Finally, I have been building a personal collection of AIDA sentences for some of the scientific publications I have read. This collection consists at the moment of 287 AIDA sentences from a wide variety of scientific disciplines. The examples shown in the introduction of this paper are from this collection.

The combined collection therefore consists at the moment of 659 AIDA sentences (650 at the time the network study to be described below was conducted; I have added nine entries to my personal collection since). All these AIDA sentences were manually curated. Automatically extracted sentences were only added after a manual check for accuracy and AIDA compliance.

User Study

I felt that AIDA sentences could turn out to be a beneficial technology also in the classroom setting. I have therefore used AIDA sentences for a Master course that I have been giving at VU Amsterdam during the fall semesters since 2015. In this course, entitled "Knowledge and Media", students are required to read 20 given papers on topics around knowledge organization. In order to help them remember and understand what they have read, I provided them with AIDA

[2] https://sparceurope.org/what-we-do/open-access/sparc-europe-open-access-resources/open-access-citation-advantage-service-oaca/

[3] https://github.com/tkuhn/aida/blob/master/usecases/openaccess.md

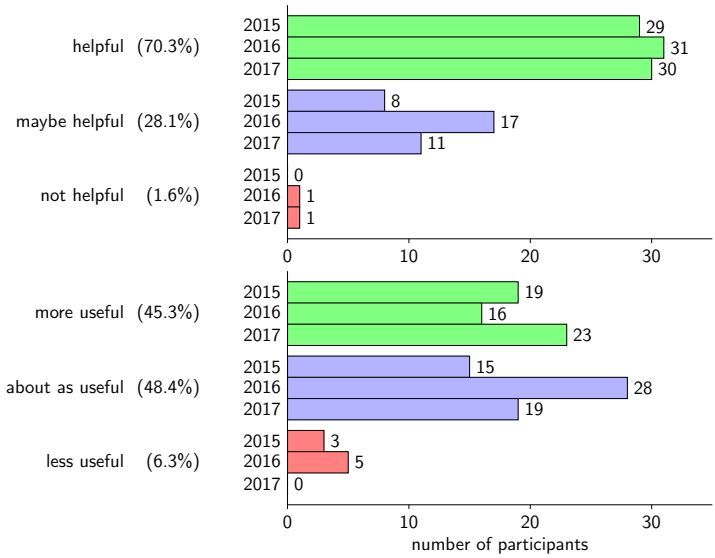


Figure 2. Responses from the participants on whether AIDA sentences were helpful (top) and on how AIDA sentences compared to classical text summaries (bottom)

sentences for each of these 20 publications and for some additional publications that I mentioned in the lecture slides. These sentences are also part of the personal AIDA collection introduced above.

In the end of the course, I asked the students to give me feedback on their opinions with respect to AIDA sentences. Specifically, I asked them the following questions with the shown answer options:

1. AIDA Sentences: Were the AIDA sentences, as presented during the lectures and on the slides, helpful for you to understand and remember the content of the papers?

- Yes, the AIDA sentences were helpful.
- Maybe. I am not sure whether the AIDA sentences were helpful.
- No, the AIDA sentences were not helpful.

2. AIDA sentences compared to classical text summaries: Did you find the AIDA sentences, as presented during the lectures and on the slides, to be more or less useful than classical text summaries?

- I found the AIDA sentences to be more useful than classical text summaries.
- I found the AIDA sentences to be about as useful as classical text summaries.
- I found the AIDA sentences to be less useful than classical text summaries.

In total, 128 students participated over the three years since 2015. The exact set of AIDA sentences varied slightly between the years as I tried to optimize the mix of papers and removed unpopular ones from the list. The students' responses are shown in Figure 2.

We see that 70.3% of all participating students thought the AIDA sentences were helpful to understand and remember the content of the papers. There is some

variation over the years, but the *helpful* group formed a clear majority in every single year, and over the three years only two students (out of 128) responded with *not helpful*. If we try to boil this down to a single number by assigning positive answers the value +1, *maybe* answers the value 0, and negative answers the value −1, we get an average response of +0.69, which is far in the positive range.

These results indicate that AIDA sentences are indeed helpful to a certain extent, but they do not tell us whether this extent is large or small. The answers of the students to the second question, comparing AIDA sentences to classical text summaries, can give us some insights on this. The positive answers (i.e. that AIDA sentences are more useful) still form the majority in the years 2015 and 2017, but not in 2016 and the overall dataset, at 45.3% of the overall responses. The majority in 2016 as well as the overall majority responded with the neutral answer (i.e. that AIDA sentences are about as useful as classical summaries). On the other hand, only eight out of the 128 respondents would have preferred classical text summaries. If we quantify the overall effect again in a single number in the same manner as before, we get a value of +0.39, which is still clearly in the positive range, even though considerably less so than for the first question.

These results indicate that AIDA sentences are indeed an intuitive and accurate method to structure scientific findings. Importantly, a vast majority of students found the AIDA sentences to be not less useful than classical summaries, despite the fact that they did not even get to experience some of the core benefits of AIDA sentences, namely advanced knowledge access powered by the formal interlinking features.

Linking and Network Study

While AIDA sentences do not require us to formally represent their domain-level content, they can serve as a tool and structure to support this process. The benefit of the AIDA approach is that this formalization does not have to happen right away (or at all), can be done by a different person (or algorithm) at a later point in time, and allows for any degree of partial formalization. And in the meantime, formal links on the meta-level of statements can be established. The study presented in this section investigates to what extent a simple kind of such post-hoc partial formalization and linking can lead to a connected knowledge graph of scientific findings.

The aim of this study is to automatically link the AIDA sentences from the datasets above to the Linked Open Data cloud [16]. For that, I applied DBpedia Spotlight [17], which is an annotation tool automatically linking text in natural language to DBpedia [18] identifiers (which map to Wikipedia pages). I used the DBpedia Spotlight API[4], with the default confidence parameter of 0.5. This annotation process of the 650 AIDA sentences of our combined dataset led to overall 1726 DBpedia mappings, i.e. on average 2.66 per AIDA sentence.

In order to assess the quality of these links, I first performed a manual evaluation of a random sample of 10% of the resulting annotations (173 out of 1726). The result showed that these annotations were correct in 94.2% of the cases, i.e. a

[4] http://www.dbpedia-spotlight.org/api

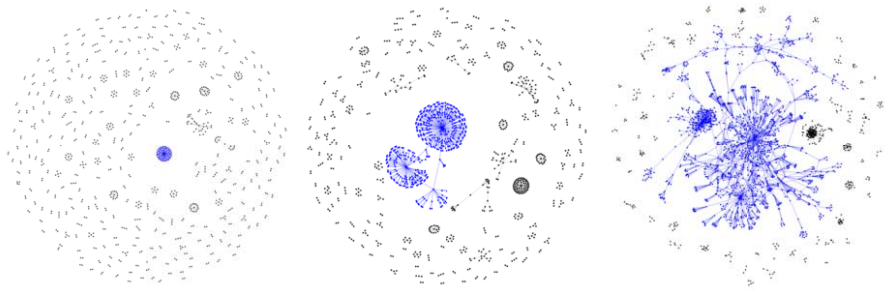


Figure 3. Visualization of the network structure of AIDA sentences with their associated publications (left), additionally augmented with some known domain identifiers (middle), and with automatically detected DBpedia connections (right). The largest connected component is shown in blue.

bit more than one error per twenty annotations. The errors range from minor differences, such as annotating the word "Japanese" in "Japanese population" with the DBpedia term for "Japanese language", to completely unrelated concepts, such as mapping the mention of the gene identifier "CHES-1" to the DBpedia entry for the "Workshop on Cryptographic Hardware and Embedded Systems", which happens to share the same acronym.

The correct annotations also show a broad range, with general and straightforward annotations on the one end, such as "probability" in this example from the Alzheimer's study:

> The treatment of Alzheimer's disease with one of the three cholinesterase inhibitors donepezil, galantamine or rivastigmine has a higher probability of at least one adverse event of anorexia before the end of the treatment as compared to a placebo treatment.

On the more specific and more complex end, the phrase "cholinesterase inhibitors" of the example above is correctly mapped to the DBpedia term "Acetylcholinesterase inhibitor", which is a slightly different name for the same thing. Moreover, also "donepezil", "galantamine", "rivastigmine", "adverse event", "anorexia", and "placebo" in the example above are correctly mapped to their DBpedia identifiers.

As a next step, we can look into the degree to which we achieved the goal of a connected network of scientific findings. Well-connected networks allow for effectively browsing, searching, aggregating, and clustering this conceptual space of scientific knowledge.

We can first look at the structure of the network consisting of AIDA sentences and the publications they link to. Unsurprisingly, this leads to a network of many small disconnected components, as shown on the left hand side of Figure 3. The network consists of 989 nodes (615 of them unique AIDA sentences, with identical sentences merged from the initial set of 650 sentences) forming 332 network components (i.e. internally connected sub-networks without connections to other components). The AIDA sentences from the Alzheimer's study form the largest component with 62 AIDA nodes (10.1%), which are connected via the node for the respective meta-review publication. Already before applying our automatic

linking, some of the AIDA sentences came with links to domain entities: The sentences from our previous study on the automatic extraction in the biomedical domain came with formal links to gene and organism identifiers. Adding these links to the network produces a large new component, comprising of 149 AIDA sentences (24.2%), shown in the middle part of Figure 3. The network is, however, still dominated by many small disconnected components.

On this background, we can now assess the impact of our automatic DBpedia linking. It introduced additional 711 nodes to the network, representing 711 distinct DBpedia concepts. The biggest component has now grown to include almost half of all AIDA nodes (48.1%, or 296 out of 615) and the number of components is reduced to 66. 80.1% of the components of the initial network therefore have been merged by the added links. The network starts to show a more complex structure, as can be seen on the right hand part of Figure 3. About half of the AIDA sentences, therefore, can now be found by browsing through the largest component. These network results do not form a direct proof but can be seen as an indication that a dense and useful knowledge structure starts appearing when applying such automatic linking methods.

All data, code, and results from the presented studies can be found online.[5]

Conclusions

AIDA sentences are designed to improve organization and communication of scientific knowledge by establishing an intuitive and general formalism to identify and link scientific claims. This formalism, in turn, can be used as a basis for further formalization in the "upward" as well as "downward" direction. Upward formalization can establish the different kinds of relationships and aggregations with formal relations expressing things like "is more general than" or "follows from". Downward formalization can take AIDA sentences as a anchor to attach partial or complete domain-level representations, for example involving relations like "is a disease affected by gene" or "correlates in human populations with". Formal representations get complicated at the very low as well as the very high level. The AIDA approach boils down to the idea that the most practical method might be to start from the middle layer of individual statements and to grow the formalization from there in both directions, and then to see how far we get.

The preliminary results presented here indeed indicate that the approach is promising. Students confirmed that AIDA sentences summarize research findings in a useful manner. Our linking and network study moreover showed that such findings can be automatically connected to Linked Data identifiers at good accuracy and that this process leads to a dense and broad network of scientific findings.

As future work, I plan to work on providing a publishing infrastructure for such AIDA sentences, based on the concept and infrastructure of a Linked Data format called nanopublications [19]. With these techniques, AIDA sentences could be published, corrected, reviewed, linked, searched, and aggregated in a fully decentralized and open manner. Researchers could then publish their latest find-

[5] https://github.com/tkuhn/aida

ings as AIDA sentences right away, and link it to the existing body of scientific findings.

References

[1] H. Bastian, P. Glasziou, and I. Chalmers, "Seventy-five trials and eleven systematic reviews a day: how will we ever keep up?" *PLoS medicine*, vol. 7, no. 9, p. e1000326, 2010.

[2] D. Nadeau and S. Sekine, "A survey of named entity recognition and classification," *Lingvisticae Investigationes*, vol. 30, no. 1, pp. 3–26, 2007.

[3] C.-H. Wei, Y. Peng, R. Leaman, A. P. Davis, C. J. Mattingly, J. Li, T. C. Wiegers, and Z. Lu, "Overview of the BioCreative V chemical disease relation (CDR) task," in *Proceedings of the fifth BioCreative challenge evaluation workshop*. Sevilla Spain, 2015, pp. 154–166.

[4] P. Ciccarese, M. Ocana, L. J. G. Castro, S. Das, and T. Clark, "An open annotation ontology for science on web 3.0," in *Journal of biomedical semantics*, vol. 2, no. 2. BioMed Central, 2011, p. S4.

[5] T. Kuhn, P. E. Barbano, M. L. Nagy, and M. Krauthammer, "Broadening the scope of nanopublications," in *Proceedings of ESWC*. Springer, 2013, pp. 487–501.

[6] T. Kuhn, "A survey and classification of controlled natural languages," *Computational Linguistics*, vol. 40, no. 1, pp. 121–170, March 2014. [Online]. Available: http://www.mitpressjournals.org/doi/abs/10.1162/COLI_a_00168

[7] T. Kuhn and M. Dumontier, "Genuine semantic publishing," *Data Science*, vol. 1, no. 1–2, 2017.

[8] A. De Waard and J. Schneider, "Formalising uncertainty: An ontology of reasoning, certainty and attribution (ORCA)," in *Proceedings of the Joint 2012 International Conference on Semantic Technologies Applied to Biomedical Informatics and Individualized Medicine-Volume 930*. CEUR-WS. org, 2012, pp. 10–17.

[9] M. C. Chibucos, C. J. Mungall, R. Balakrishnan, K. R. Christie, R. P. Huntley, O. White, J. A. Blake, S. E. Lewis, and M. Giglio, "Standardized description of scientific evidence using the evidence ontology (ECO)," *Database*, vol. 2014, 2014.

[10] T. Kuhn and M. Krauthammer, "Underspecified scientific claims in nanopublications," in *Proceedings of the Workshop on the Web of Linked Entities (WoLE 2012)*. CEUR-WS, 2012, pp. 29–32.

[11] T. Jansen and T. Kuhn, "Extracting core claims from scientific articles," in *Benelux Conference on Artificial Intelligence*. Springer, 2016, pp. 32–46.

[12] C. K. Ogden, *Basic English: a general introduction with rules and grammar*. London: Paul Treber & Co., 1930.

[13] T. Kuhn, L. Royer, N. E. Fuchs, and M. Schroeder, "Improving text mining with controlled natural language: A case study for protein interactions," in *International Workshop on Data Integration in the Life Sciences*. Springer, 2006, pp. 66–81.

[14] C. Hallett, D. Scott, and R. Power, "Composing questions through conceptual authoring," *Computational Linguistics*, vol. 33, no. 1, pp. 105–133, 2007.

[15] T. Kuhn and S. Höfler, "Coral: Corpus access in controlled language," *Corpora*, vol. 7, no. 2, pp. 187–206, 2012.

[16] C. Bizer, T. Heath, and T. Berners-Lee, "Linked data-the story so far," *International journal on semantic web and information systems*, vol. 5, no. 3, pp. 1–22, 2009.

[17] P. N. Mendes, M. Jakob, A. García-Silva, and C. Bizer, "DBpedia spotlight: shedding light on the web of documents," in *Proceedings of the 7th international conference on semantic systems*. ACM, 2011, pp. 1–8.

[18] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. Ives, "Dbpedia: A nucleus for a web of open data," in *The semantic web*. Springer, 2007, pp. 722–735.

[19] T. Kuhn, C. Chichester, M. Krauthammer, N. Queralt-Rosinach, R. Verborgh, G. Giannakopoulos, A.-C. N. Ngomo, R. Viglianti, and M. Dumontier, "Decentralized provenance-aware publishing with nanopublications," *PeerJ Computer Science*, vol. 2, p. e78, 2016.

Controlled Natural Language
B. Davis et al. (Eds.)

doi:10.3233/978-1-61499-904-1-61

Putting Control into Language Learning

Herbert LANGE [a] Peter LJUNGLÖF [a]
[a] *Computer Science and Engineering*
University of Gothenburg

Abstract Controlled Natural Languages (CNLs) have many applications including document authoring, automatic reasoning on texts and reliable machine translation, but their application is not limited to these areas. We explore a new application area of CNLs, the use of CNLs in computer-assisted language learning. In this paper we present a a web application for language learning using CNLs as well as a detailed description of the properties of the family of CNLs it uses.

1. Introduction

Controlled Natural Languages (CNLs) are an active field of research in Computational Linguistics. Their definition usually is vague, but the general consensus is that they are constructed languages placed somewhere between full natural languages on the one hand and formal languages on the other. Kuhn discusses the definition in [1] and presents typical applications for CNLs, such as machine translation and document authoring. He also expects other applications without going into detail about these possibilities. In this paper we present one new application of CNLs, the use in computer-assisted language learning (CALL).

We present the MUSTE Language Learning Environment (MULLE)[1], a tool for learning languages as a second or foreign language which can use CNLs for the description of learning objectives, and automatically generate translation exercises from formal grammars of these languages. The application itself uses methods similar to conceptual authoring [2] to edit sentences in natural languages in order to make them proper translations of each other. By using CNL grammars as the basis for the exercises, and using textbook lessons as the basis of the grammars, it is possible to create a learning environment that complements the traditional classroom setting.

This article is structured as follows: In Section 2 we present related work, both in CNLs and CALL. Section 3 describes the language learning application we designed. It describes the interface provided to the user and gives details about the grammars used. In Section 4 we sketch an experimental method to evaluate our system and describe a small-scale pilot. Finally in Section 5 we discuss further questions and conclude the article in Section 6 with a look to possible future work.

[1] `https://github.com/MUSTE-Project/MULLE`

2. Related Work

The use of CNLs for language learning seems to be a new application in this field that has not yet been broadly discussed. For this reason, the amount of directly related work is rather limited with the exception of [3]. However, this application can also be viewed as a combination of two tasks that have been popular among the CNL community: reliable machine translation and user support for text edition and creation.

2.1. Related CNL work

Quite often the motivation for the use of Controlled Natural Language is their proximity to formal language. This allows, e.g. automatic reasoning within and reliable translation of these languages. To guarantee that the language used by an author is covered by a CNL, special editors have been proposed. Two approaches are conceptual editing [2] and predictive editing [4,5]. Predictive editing uses chart parsers and compatible grammars to suggest only valid continuations in the process of writing. Some systems only support a fixed vocabulary while other systems support the extension of the vocabulary while the document is authored.

Conceptual editing instead refines the underlying representation by manipulating the surface presentation, i.e. the natural language sentence. In this process so called "holes" are created and filled.

Angelov and Ranta [5] not only present a predictive editor, but also suggest a translation system based on Attempto Controlled English (ACE) [6]. It provides reliable machine translation via abstract syntax trees in the Grammatical Framework (GF) [7,8].

Some CNLs were designed to aid learning languages at a time before computers were considered a tool for it. One example is Basic English [9], an auxiliary language created to help people learn English as a second or foreign language, invented at the beginning of the 20th century.

2.2. Related work within language learning

In the field of language learning applications several approaches can be observed. They range from finite-state technology for morphology training [10] over annotated text data, or semantic resources combined with rule-based algorithms [11,12,13] to user-generated content in combination with machine learning [14]. The aim and scope of theses systems also varies broadly, including the learning environment they target. The systems [12,13] target a closed classroom setting with specific language classes while [10,11] aim at a broader learning environment and are applicable outside a specific course. Modern general-purpose systems like Duolingo[2] target independent language learners.

Reliability also varies between these systems. The smallest scale systems provide the most reliable examples while the most general systems being the least reliable.

[2] `https://www.duolingo.com`

3. Application: Language Learning using CNLs

In this paper we describe a web-based language learning tool which takes advantage of CNL grammar features. It is intended for use in a closed, classroom-related learning environment and provides reliable translation exercises by using fully formalised grammars. The tool presents exercises grouped into lessons where the user is presented with sentences in two different languages.

Usually two languages are involved in language learning: one language the user already knows is used for instructions in the language classes (the meta language) and one language the user is learning (the object language) by discussing it in the meta language . So the meta language and object language in the classroom determine the two languages used in our exercises.

The task of the user is to edit one of the sentences to make it a proper translation of the other. Currently, this means that the underlying GF abstract syntax trees for both sentences have to be the same. As a future development, we imagine an extension where we consider not single trees but sets of trees, in order to be able to handle ambiguous parses. In this case it would be sufficient to have at least one abstract tree in the intersection of the two sets.

The tool has been used in an introductory course in Latin for Swedish students. So in all examples given here Swedish acts as the meta language and Latin as the object language. We implemented the first four lessons of [15] and conducted the pilot of a user study that is described in more detail in Section 4.

3.1. The editing interface

Our application uses a method for word-based text-editing [16], which is in principle related to conceptual editing. It uses syntax trees as formal representations and provides editing operations like insertion, deletion, and substitution on the surface by mapping them onto tree operations. In our application we only look at complete syntax trees, that means we do not use "holes" for incremental creation, but instead modify complete syntax trees.

The editing operations work in the following way: the user clicks on a word in the sentence or on the gap between two words in the sentence. The click position is translated to the node in the syntax tree in which the word is introduced or the closest node covering the space that was clicked. Based on the subtree below this node, all subtrees with the same root category are computed and their linearisations, i.e. their surface representations, are collected and presented to the user.

To clarify this process a set of screenshots with the corresponding syntax tree can be seen in Figures 1–5. In the screenshot one can see the two sentences in different languages. The sentence at the top is fixed and the sentence at the bottom can be changed by the user. The syntax tree beside the screenshot describes the sentence at the bottom. Figure 1 shows the start of the system before any click. Clicks on words in the sentence are then translated to pointers into the tree. For example clicking on the word "Gallien" will set the pointer (circled node) to the node introducing this string, in this case the rightmost PN node (Figure 2). Then the category in the focus of the pointer is used to compute all similar trees of

```
Prima scripta Latina

[...] Imperium imperatorem habet. Imperator imperium tenet. Caesar
Augustus imperator Romanus est. Imperium Romanum tenet. Multas
civitates externas vincit. Saepe civitates victae provinciae deveniunt.
[...]
```

Figure 6. Sample from the text fragment in the first lesson in [15]

with the same category in the root. These trees are used to suggest potential replacements which are shown to the user in the form of a menu. Clicking on the same word several times moves the pointer up in the tree which changes the root category of the candidate trees from PN to NP, VP and so on, and suggests different changes to the sentence (Figure 3–4) before finishing in the root of the tree with category Cl (Figure 5) where the menu does not change anymore. In this way larger phrases can be changed at the same time of more global features like sentence negation can be modified. When instead the user clicks on a different position than before, then the system is reset and the pointer again points to the node indicated by the new click.

3.2. Lessons and exercises

The application provides of a set of lessons, each consisting of a number of exercises. A lesson is defined by a multilingual GF grammar which is derived from a part of a textbook. These parts usually consist of text fragment (a sample can be seen in Figure 6), a vocabulary list, some explanation of grammatical phenomena, as well as some exercises that are supposed to be solved on paper.

We adopt this structure in our system by using the same text fragments presented in the textbook lessons and formalising them in separate grammars that cover the vocabulary as well as the syntactic constructions used in the corresponding parts of the textbook.

Given a lesson grammar, an exercise consists of two syntax trees and their surface representations. With the task described before a score is calculated based on both the number of clicks and the time spent before finishing the exercise. After finishing a certain number of exercises, the lesson is considered finished.

3.3. Creating the lesson grammars

We create a GF grammar for each textbook lesson in the following way. This work should be automated as much as possible but the basic procedure can also be executed manually.

1. The first step is to adapt a lexicon for the textbook lesson, which is given as an explicit vocabulary list in the book. We can use existing reliable lexical resources or GF smart paradigms [17] to implement it in our grammar.
2. The next step is to create syntax trees for all sentences in the text (Figures 7–8). This can either be done manually or semi-automatically. To automate this process, we parse each sentence using the GF resource gram-

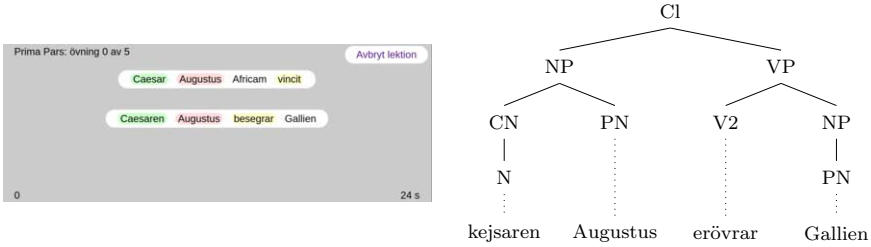


Figure 1. System before any click

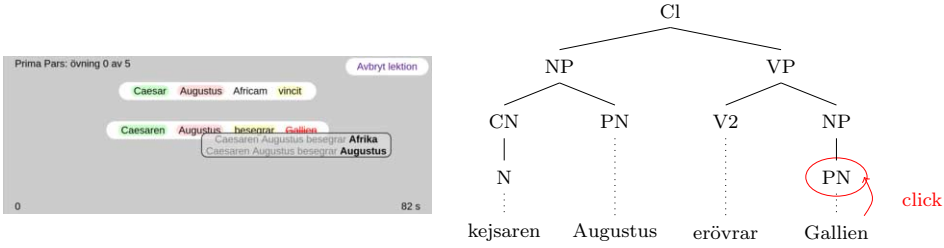


Figure 2. System after one click on "Gallien"

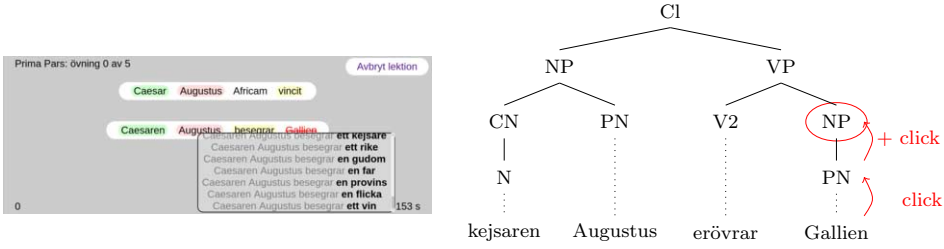


Figure 3. System after second click on "Gallien"

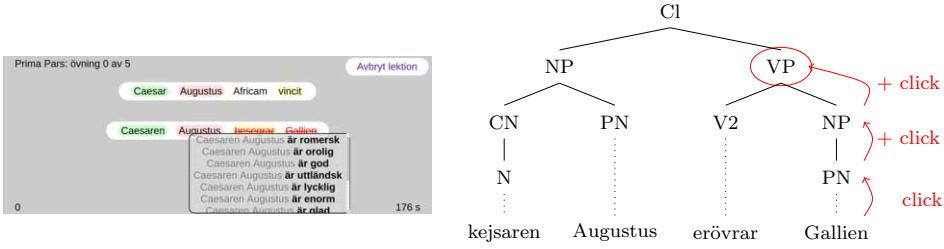


Figure 4. System after third click on "Gallien"

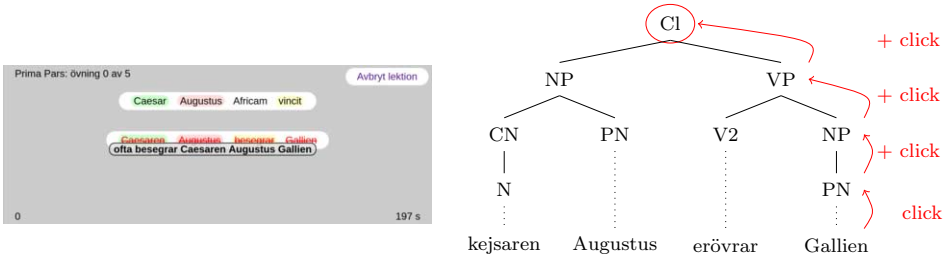


Figure 5. System after fourth click on "Gallien"

mar library [18] augmented with the lexicon from step 1. Because of syntactic ambiguities this might result in several possible trees, so afterwards we have to manually select the correct syntax tree, i.e. the desired analysis of the sentence. This involves some linguistic knowledge from the person creating the grammar.

3. Finally, we formulate the grammar that describes the text fragment in the textbook. This can be done straightforwardly by reading off the grammar rules from the internal nodes in the trees (see Figure 9). This will usually result in an over-generating grammar, so we use different techniques to reduce the over-generation such as merging two or more generated rules into one.

```
Caesar Augustus
imperator Romanus
est.
```

Figure 7.: Example sentence

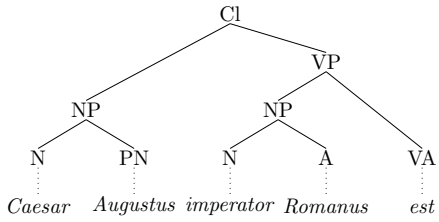


Figure 8.: Syntax tree derived from sentence in Figure 7

Abstract:

```
NP —> N PN
NP —> A N
VP —> VA NP
Cl —> NP VP
S  —> Cl
...
```

Figure 9.: Derived abstract grammar

3.4. Making the lesson grammars ungrammatical

The above process yields a grammar which only accepts syntactically correct sentences. However, we also wish to train the morphology in the object language, e.g. noun-verb agreement, number/gender inflection, affixation, etc.

It is possible to semi-automatically transform a lesson grammar into a grammar that accepts some grammatical errors, e.g. sentences where nouns and verbs disagree in number. What has to be done manually is to indicate which inflection parameter(s) in which grammar rule(s) should be loosened. Then it is possible to automatically transform the grammar into a grammar that accepts sentences where that specific inflection parameter is violated.

The user's task still is to edit a sentence in the object language to make it a translation of the meta language, but now they will have the additional complexity of allowing ungrammatical sentences. It is possible to control the level of ungrammaticality by deciding how many inflection parameters should be loosened.

3.5. Characterisation of the lesson grammars

We identified relevant criteria that characterise grammars that are suitable to be used by our application. The two most relevant criteria are a layer of semantics in the grammars as well as making some implicit features of the syntax explicit.

Grammars in GF usually are distinguished between resource grammars and application grammars. The main difference is that resource grammars just describe the syntax of a language without any semantic considerations while application grammars are used for a specific application and include semantic aspects necessary for that domain.

The grammars that are used in our system also have a strong focus on syntax, but require at least some semantic restrictions. From a purely syntactic point of view, adjectives can be combined with any noun, but language use only allows certain combinations. This can be solved by including semantic knowledge in different ways, of which the inclusion of FrameNet-style semantics in the grammars [19] seems the most natural.

The second point is, that natural languages might employ syntactic features that are not visible on the surface. One example are romance languages that allow the dropping of pronouns in the subject position because the relevant information is already present in the verb form. But to keep the grammars multilingual, these pronouns still have to be present in the grammar as empty tokens, which is an obstacle in language learning. This has to be changed in a way to make this information explicit on the surface for our application to be useful.

3.5.1. PENS classification

Given this description of the family of grammars we designed we want to render this definition more precisely in the PENS classification scheme [1]. PENS stands for Precision, Expressiveness, Naturalness, and Simplicity and is typically used to classify CNLs. Each of these scales ranges from 1 to 5 with 1 being the least and 5 the highest level of strictness possible.

Our grammars are fully specified in a computational grammar formalism and each sentence can be mapped to a set of abstract syntax trees. We do not insist on completely unambiguous interpretations but the set of interpretations will always be finite. According to the PENS classification that places our languages in the field of *Deterministically interpretable languages* (P^4).

The sentences generated by our grammars should be syntactically correct and in this aspect be considered as *Languages with natural sentences* (N^4). Because we focus just on sentences in our application, an extension to *Languages with natural texts* is not necessary.

The scope of our grammars is very limited, both by the text fragments and the explicit vocabulary, which makes it possible to formalise the language fragments in compact grammars. Even though they rely on external resources in the form of the RGL, the fragments can be considered as *Languages with short descriptions* (S^4).

The only problematic dimension is Expressivity, because we do not really focus on a translation to a specific logic interpretation, but remain on the level of the abstract syntax tree and its expressiveness. But because this is not relevant for our application we decide to ignore this dimension and set it to E^- That places our languages in the family of $P^4E^-N^4S^4$ languages.

This classification is not just some characterisation of the grammars we use in our system now and the languages defined by them, but instead a general requirement for all grammars and languages that can be used in our framework.

4. Evaluation

To test the acceptance of our application, as well as the desired change in learning outcome and learner motivation, we designed an empirical evaluation which we partially conducted as a pilot in connection with an introductory course for Latin at university level.

The full experiment consists of a prepared set of four lessons with a runtime of about four weeks. At the beginning, the students are asked to answer a questionnaire to control for aspects of the learner background and give some insight into the motivation at the beginning of the course. A simple timed placement test with eight exercises, four from each lesson, estimates the language skills before taking the class. The participants then are split into one treatment group and one or several control groups. In the following four weeks the students in the treatment group get access to lessons matching the progress in class while one control group only gets access to the traditional learning material in the text book. After the experiment period the students are given a slightly modified version of the questionnaire from the beginning to test for a change in motivation and a second placement test to see if there is a change in speed to solve the exercises.

In the pilot, due to lack of students, we could only ask for general feedback without gaining relevant insight into change in learner motivation or learning outcome. From ten students in class six volunteered to try the application and answer the first questionnaire. But due to a general drop out from this class, only four students were present in the end, of which only two had volunteered to participate. Still, the general feedback from both teachers and students was very positive which encourages us to aim for a full scale version of the evaluation.

5. Discussion

In the related work we pointed out several different technical approaches for systems in the field of foreign and second language learning. The different systems differ not only in the expressivity of the underlying technology but also in their intended use case.

Systems which employ technology with limited expressivity like finite-state technology aim at a closed setting in a very specific classroom setting but provide a high reliability. Other systems that employ very expressive machine learning methods can be used in a very open and classroom-independent setup but suffer from a lack of reliability.

With our system, which uses a very expressive syntax formalism, we currently target a closed classroom setting where we can profit from the reliability of our grammar-based approach but we also believe it is possible to widen the focus to provide a completely open language-learning application.

We claim that our system employs controlled natural languages for language learning. Some might disagree, and we admit that the PENS classification fails in the point of expressivity. But the application we sketch is grammar-agnostic, which means one can use almost any multilingual grammar to generate translation exercises from it. The grammars we used so far might not really be seen as

controlled languages because they are defined too implicitly, even though textbook lessons usually are created with clear concepts in mind. Still it can be used with any grammar that fulfils the PENS requirements we identified as characteristic for our grammars. This also gives a chance for further research looking into the application of CNLs in CALL far beyond the scope of this work.

Finally it is possible to discuss the combination of the underlying technology with other CNLs to build different applications. We think that there is some potential, especially given the similarity of conceptual editing and the word-based text editing, to have a fruitful exchange between the CNL community and other disciplines.

6. Conclusions and Future Work

We presented a working application usable for language learning that uses fully formalised grammars to define language learning lesson. According to some definition of controlled natural languages these lessons can be seen as CNLs, even there might be problems with this claim.

In the future we want to investigate how the design of the grammars influences the learning experience. This mostly concerns the structure of the grammars with varying focus on syntax and semantics. But that also includes additional ideas for different kinds of language learning exercises.

Another relevant topic of research is automatic generation of "good" exercises. This in entangled with the questions which kind of exercises besides translation exercises we want to include in our application. It also seems connected to a different topic, the selection of good examples in the creation of lexica [20], even though features of good translation exercises are not exactly the same as for good lexicon examples. Still this would give an opportunity for further interdisciplinary research.

References

[1] T. Kuhn, "A survey and classification of controlled natural languages," *Computational Linguistics*, vol. 40, pp. 121–170, March 2014.

[2] C. Hallett, D. Scott, and R. Power, "Composing questions through conceptual authoring," *Computational linguistics*, vol. 33, no. 1, pp. 105–133, 2007.

[3] E. Abolahrar, "Multilingual grammar-based language training: Computational methods and tools," Master's thesis, Chalmers University of Technology, 2011.

[4] T. Kuhn and R. Schwitter, "Writing support for controlled natural languages," in *Proceedings of the Australasian language technology association workshop 2008*, pp. 46–54, 2008.

[5] K. Angelov and A. Ranta, "Implementing controlled languages in GF," in *Proceedings of the 2009 Conference on Controlled Natural Language*, CNL'09, (Berlin, Heidelberg), pp. 82–101, Springer-Verlag, 2010.

[6] N. E. Fuchs, S. Höfler, K. Kaljurand, F. Rinaldi, and G. Schneider, "Attempto Controlled English: A Knowledge Representation Language Readable by Humans and Machines," in *Reasoning Web, First International Summer School 2005, Msida, Malta, July 25–29, 2005, Revised Lectures* (N. Eisinger and J. Małuszyński, eds.), no. 3564 in Lecture Notes in Computer Science, Springer, 2005.

[7] A. Ranta, "Grammatical Framework: A multilingual grammar formalism," *Language and Linguistics Compass*, vol. 3, no. 5, pp. 1242–1265, 2009.
[8] A. Ranta, *Grammatical Framework: Programming with Multilingual Grammars*. Stanford: CSLI Publications, 2011.
[9] C. K. Ogden, *Basic English : A general introduction with rules and grammar*. London: K. Paul, Trench, Trubner & co., ltd, 1930.
[10] H. Kaya and G. Eryiğit, "Using finite state transducers for helping foreign language learning," in *Proceedings of the 2nd Workshop on Natural Language Processing Techniques for Educational Applications*, pp. 94–98, 2015.
[11] M. Moritz, B. Pavlek, G. Franzini, and G. Crane, "Sentence shortening via morpho-syntactic annotated data in historical language learning," *Journal on Computing and Cultural Heritage (JOCCH)*, vol. 9, no. 1, p. 3, 2016.
[12] L. N. Michaud, "King Alfred: A translation environment for learners of Anglo-Saxon English," in *Proceedings of the Third Workshop on Innovative Use of NLP for Building Educational Applications*, (Columbus, Ohio), pp. 19–26, Association for Computational Linguistics, June 2008.
[13] H. Redkar, S. Singh, M. Somasundaram, D. Gorasia, M. Kulkarni, and P. Bhattacharyya, "Hindi shabdamitra: A WordNet based e-learning tool for language learning and teaching," in *Proceedings of the 4th Workshop on Natural Language Processing Techniques for Educational Applications (NLPTEA 2017)*, pp. 23–28, Asian Federation of Natural Language Processing, 2017.
[14] A. K. Horie, "Rewriting Duolingo's engine in Scala." `http://making.duolingo.com/rewriting-duolingos-engine-in-scala`, January 2017. accessed 04.04.2017.
[15] S. Ehrling, *Lingua Latina novo modo – En nybörjarbok i latin för universitetsbruk*. University of Gothenburg, 2015.
[16] P. Ljunglöf, "Editing syntax trees on the surface," in *Nodalida'11: 18th Nordic Conference of Computational Linguistics*, (Rīga, Latvia), 2011.
[17] G. Détrez and A. Ranta, "Smart paradigms and the predictability and complexity of inflectional morphology," in *Proceedings of the 13th Conference of the European Chapter of the Association for Computational Linguistics*, EACL '12, (Stroudsburg, PA, USA), pp. 645–653, Association for Computational Linguistics, 2012.
[18] A. Ranta, "The GF Resource Grammar Library," *Linguistic Issues in Language Technology*, vol. 2, no. 2, 2009.
[19] N. Gruzitis and D. Dannélls, "A multilingual FrameNet-based grammar and lexicon for controlled natural language," *Language Resources and Evaluation*, vol. 51, pp. 37–66, Mar 2017.
[20] A. Kilgarriff, M. Husák, K. McAdam, M. Rundell, and P. Rychlý, "Gdex: Automatically finding good dictionary examples in a corpus," in *Proceedings of the 13th EURALEX International Congress* (E. Bernal and J. DeCesaris, eds.), (Barcelona, Spain), pp. 425–432, Institut Universitari de Linguistica Aplicada, Universitat Pompeu Fabra, jul 2008.

Controlled Natural Language
B. Davis et al. (Eds.)

doi:10.3233/978-1-61499-904-1-71

Automated Program Synthesis from Object-Oriented Natural Language for Computer Games

Michael S. Hsiao
Electrical and Computer Engineering, Virginia Tech, Blacksburg, VA 24061

Abstract. A prototype object-oriented natural-language programming system for computer/video games is described, in which sentences written in object-oriented English is automatically converted to a functional, executable game code in Javascript. In addition, new attributive words are automatically learned while converting the text to code. Any syntactic or semantic errors are reported during the compilation to help users to debug and fine-tune the game. With less than 20 plain English sentences, up to 1800 lines of game code can be generated.

Keywords. Object-oriented, video game, natural language, code synthesis

1. Introduction

Natural Language (or Naturalistic) Programming [1, 2] in the user's native language attempts to directly convert the instructional text to a computational program. The full realization of natural language based programming (NLBP) for general-purpose programming is considered extremely challenging. This is because not only does the synthesis/compilation system need to detect syntactic and grammatical errors, they also need to detect *ambiguity* errors. In this work, we offer a CNL that is precise, yet sufficiently natural and expressive to the user. The key constraints we place into the language is that every clause should involve *objects*. An object is an identifiable entity used in a video game, such as the specific characters, score, etc. Note that while these objects are nouns in NL, not every noun in NL is a valid object in a video game. Hence, we call such a CNL object-oriented natural language, or simply OONL.

We choose video games as the target domain because today's children have grown up with the Internet and are familiar with computer games. Moreover, the learner can readily see their code in action, which forms a positive feedback to the learning process. A prototype has been constructed for converting OONL to program code for video games. A 20-sentence OONL text can be converted to more than 1800 lines of code. If there are any ambiguity or unclear sentences, error messages and hints on how to fix the errors will be given as well. We note that this system can be extended to beyond video games.

2. The OONL Language Model for Coding

For a specific application domain, the first step is to construct the language model with which the user can communicate the ideas. Succinctly, the language, L, is a tuple: $L=(E, A, T, P, S, G)$. Each of the parameters is described below. E: the set of entities (or objects), A: the set of actions, T: the set of attributes, P: the set of predicates, optionally, S: the set

of selectors, and finally *G*: the underlying grammar rules binding the words and phrases. The items in sets, *E, A, T, P*, and *S* are either words or word-phrases. The extent to which these sets encompass will also determine the expressiveness of the resulting OONL. To understand the sentences, we need a grammar, *G*, that takes in a sentence in which a varying number of phrases from each of the five aforementioned sets is comprised.

For the video game domain, the set of entities, *E*, contains the characters involved in the game, such as *rabbits, foxes, carrots*, etc. The set of actions, *A*, may include *chase, flee, wander, stop, jump*, etc. Third, the set of attributes, *T*, includes the color, speed, etc. associated with the characters. Note that the user can add new attributes on the fly. Next, the set of predicates, *P*, may include see(), reach(), touch(), catch(), etc. Finally, the set of selectors, S, allows the user to say something like "*the fox whose speed is 5*".

The aforementioned sets are all dynamic, in the sense that new terms can be learned. For example, the sentence "*When a rabbit sees a fox, it becomes scared. When a rabbit is scared, it …*" The attributive term '*scared*' needs to be learned and associated with the behavior at run-time, which will be explained in the next section.

The set of possible valid sentences can be determined by a set of grammatical rules. Alternatively, the semantics can be obtained using a classification system. In either case, an intermediate representation (IR) for each sentence is generated. Finally, whenever incorrect or ambiguous sentences are entered, the proposed system can also offer suggestions on how to re-write the sentence.

3. Methodology

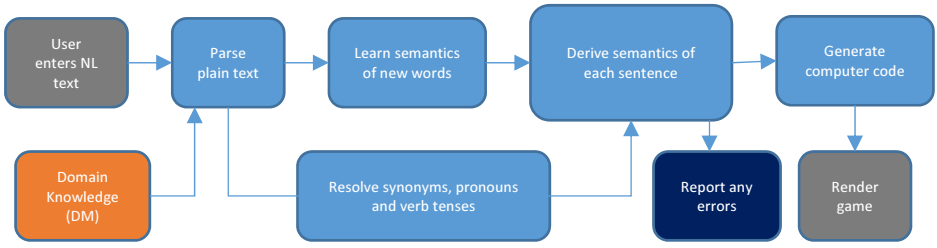


Figure 1. Overall flow of the system.

Figure 1 illustrates the high-level flow of the proposed NLBP system.

Step 1: Parse and learn any new terms. After the entire text is captured, one sentence is retrieved from the text at a time, while remembering the knowledge from any previous sentences. Any word not found in the lexicon is regarded as a potential new term that may describe new behaviors. Such words will be identified. Clearly, if any new term is used as an adjective multiple times, it will define new semantics. For example, consider the following.

"When a rabbit eats a diamond, it becomes *empowered*. … When a fox sees an *empowered* rabbit, the fox runs away."

In this example, the term "*empowered*" is learned. In our method, we first check if the new term is found in at least one antecedent clause of some sentence and one consequent clause of another sentence. Let A_i and C_i denote the antecedent and consequent clauses of sentence *i*, respectively. Then, if word *w* is found in A_i and C_j for sentences *i* and *j*, *w*

will be marked. Furthermore, A_j describes the condition for which w is set, and C_j describes the action that will result when w is true. Note that C_j may contain a conjunction of conditions, one of which is w.

Step 2: Resolve synonyms, related terms, verb tenses, modifier clauses and pronouns. Parallel to Step 1, this step resolves the synonyms, verb tenses, modifier clauses and pronouns. The order of the resolution is not important.

There may be many different ways that the user can describe how two objects collide, such as "the rabbit *hits* the fox", "the rabbit *touches* the fox", etc. All related terms and synonyms are resolved and replaced by a representative term. We note that this step does not need to consider the full grammatical structure of the sentence, but just the local context around the synonym in question. The represented related terms are stored in a set in this work. However, other data structure can be used as well.

For resolving pronouns, we use a very light-weight method as the compilation must be performed in real-time for potentially large number of users. However, we note that any standard anaphora resolution can be used.

Next, clauses such as "that is not empowered" needs to be bound to the corresponding object. Such clauses can be represented as n-grams that immediately follows a preceding character. We note that the modifier can also come before the character, such as "an empowered rabbit". Thus, "an empowered rabbit" and "a rabbit that is empowered" are semantically equivalent.

Finally, verb tenses can be handled similarly as the synonyms. In our approach, we convert all passive verbs phrases into active phrases to ease translation. At the end of this step, all identified terms are tokenized.

Step 3: N-grams of tokens and grammar to determine semantics. Given the sanitized token stream with all synonyms, related-terms, modifiers, verbs and pronouns resolved, we move on to determining the semantics of this token stream. We need a way to convert various alternatives to a single representative token. For example, the phrase "when a rabbit collides with a fox" can very well be said as "when a rabbit and a fox collide." For example, "A collides B" may be chosen as the representative way to describe how to objects collide, replacing all occurrences of "A and B collide", etc. Thus, an *unordered* n-gram (or bag of word-tokens) {collide, A, B} can be used to detect that objects A and B have collided; however, the order in which the three tokens appear does not matter. One may constrain this by adding that certain orders of 'collide', 'A', and 'B' are not allowed. Consider another example, "A eats B". Here, the order of the words matter. So an *ordered* n-gram {A, eat, B} is used. These n-grams form predicates, such as collide(A, B), or actions eat(X, Y), etc. With these predicates and actions, the grammar is defined over them. A subset of production rules is shown below, where **ant1**, **ant2**, etc. are predicates and **cons1**, **cons2** are actions obtained from bag-of-words and n-grams.

cond_sent → when_if antecedent consequent | consequent when_if antecedent
antecedent → ante_phrase | ante_phrase and ante_phrase
ante_phrase → ant1 | ant2 | ant3 | …
consequent → cons_phrase | cons_phrase and cons_phrase
cons_phrase → cons1 | cons2 | cons3 | …
when_if → when | if

Due to space considerations, the complete set of production rules is omitted here.

Step 4: Code generation. The final step of the process is to generate executable code from the processed NL text. We note that our approach can generate code for any target programming language. For example, the sentence "when a rabbit collides with a fox, the rabbit dies" will be converted to the IR 'if collide(rabbit, fox), die(rabbit).' The implementation of both collide() and die() functions are pre-written for the programming language of choice. Likewise, the objects used the game are pre-designed. Finally, conversion of IR to target code is straight-forward.

Lastly, we also produce an initialization subroutine that allocates memory for all objects and initializes them. Similarly, we add a starting game screen, in which the title of the video game, the author of the game, and some of the characters involved are shown.

Error Handling: There are various ways to handle errors. Error message generation rules are also factored into the system during n-gram detection and production-rule checking. Consider the erroneous sentence, "A eats pushes B", the system has four choices: (a) interpret it as "A eats B", (b) interpret it as "A pushes B", (c) ignore and skip the sentence, and (d) ask the user to modify the sentence. Depending on the intent of the system and the severity of the problem, one can choose any one of the above. In all cases, warnings will be displayed to the user, should she wish to go back and re-write the text.

4. Results of Sample OONL Games

The system generates Javascript code, working with the ProcessingJS library [3] to render polygons and other Processing-specific functions. We note that a wide genre of video games can be generated, including 2D platformer, 2D sports, shoot'em ups, etc. Below is an example OONL game, with a screen shot for the game generated.

"Thirty carrots are scattered all over the field. There is one rabbit. There are two foxes near the top of the area. The rabbit initially is near the bottom of the playing area. You control the rabbit with keyboard. When an arrow key is pressed, the rabbit moves in the same direction as the arrow. When the rabbit encounters a carrot, the carrot is eaten by the rabbit and the rabbit is empowered for 2 seconds. When the rabbit eats a carrot, score increments. When the fox sees the rabbit and rabbit is not empowered, the fox starts chasing the rabbit. When the fox sees the rabbit and rabbit is empowered, the fox starts flees the rabbit. When the fox does not see the rabbit, it wanders. When fox reaches border, it reverses. When the fox catches the rabbit, the game is over. When score is 30, you win."	

This platform [4] has been piloted in more than 20 classes in Virginia and North Carolina, with very positive responses from the users. The users have particularly liked the ease of creating a game while learning the underlying computational design process.

References

[1] A. W. Biermann and B. W. Ballard, "Toward natural language computation," Comput. Linguist., vol. 6, no. 2, pp. 71–86, 1980.
[2] O. Pulido-Prieto and U. Juarez-Martínez, "A survey of naturalistic programming technologies," ACM Computing Surveys (CSUR) Surveys, vol. 50, no. 5, November 2017.
[3] http://processingjs.org/
[4] Game Changineer: https://gc.ece.vt.edu

Controlled Natural Language
B. Davis et al. (Eds.)

doi:10.3233/978-1-61499-904-1-75

Understanding Texts in Attempto Controlled English

Norbert E. Fuchs
Institute of Computational Linguistics
University of Zurich

Abstract. Understanding texts in Attempto Controlled English (ACE) is considered undemanding, nonetheless hides some problems. To deal with these problems I propose an experiment based on Kuhn's ontographs that tests the understanding of simple ACE texts. Furthermore, I suggest to compare the relation between authors and readers with human verbal conversations. My conclusion is that the correct understanding of an ACE text is possible, but requires contributions from both authors and readers, quasi their cooperation.

Keywords. Attempto Controlled English, ACE, ontographs, human conversation, cooperative principle, Grice's maxims, repair, deduction, RACE, ACE comments

1. Can Texts in Attempto Controlled English Be Understood by Anybody?

We, the authors of Attempto Controlled English (ACE), claim that "once written, ACE texts can be read and understood by anybody"[1].

Is our claim really justified?

ACE was originally developed in the context of requirements engineering with the goal to make formal methods available to people who are not familiar with them. We have evidently achieved this goal as the current version of ACE is a rich and powerful knowledge representation language in the form of a subset of English that can automatically be translated into many formal, mostly logic, languages.

On top of that requirement engineering suffers from another problem that is related to our claim above. Requirement specifications, generally written in unconstrained natural language, are often not understood in the same way by the various people involved in the software development process [1,2]. This can lead – and has led – to misunderstandings with serious consequences for the software project. The misunderstandings are to a large part caused by the use of unconstrained natural language that is prone to ambiguity, vagueness, inconsistency, incompleteness, and hidden context dependencies. Other misunderstandings are due to human faults and to organisational shortcomings.

The question arises whether ACE – being a controlled language with a logical basis – can avoid, or at least alleviate, communication problems as those exhibited in requirements engineering. For a question so general, the answer must be no since ACE does not offer any assistance for organisational shortcomings. However for cases not involving organisational aspects, ACE seems to be the perfect answer since it eliminates most linguistic issues and provides means to curtail human faults.

---

[1] http://attempto.ifi.uzh.ch/site/description/

Returning to our claim above, I will investigate the following concrete question: Does a reader of an ACE text – familiar with ACE or not – understand the ACE text as the writer intended? Note that this question involves only linguistic issues and the behaviour of the writer and the reader, but no organisational entanglements.

I will present two approaches to answer this question, one based on Kuhn's ontographs [3], another one modelled on human conversation [4]. The first approach is basically a test for understanding, while the second one is a careful examination of the roles of the writer and the reader of an ACE text. This examination leads to necessary conditions for our claim: understanding an ACE text is not solely the responsibility of the reader, but both the writer and the reader have to contribute, quasi to collaborate, to achieve a common understanding.

2. Kuhn's Ontograph Experiments

In his thesis [3] Kuhn investigates the understandability of ACE by performing two experiments with test persons not familiar with ACE. Before I describe Kuhn's experiments and their results, a clarification is necessary to avoid a possible confusion. The word "understanding" used by me in this paper has another meaning than the word "understanding" as used by Kuhn. While Kuhn addresses the understanding of the language ACE itself, I investigate whether and to which extent a reader understands an ACE text written by somebody else.

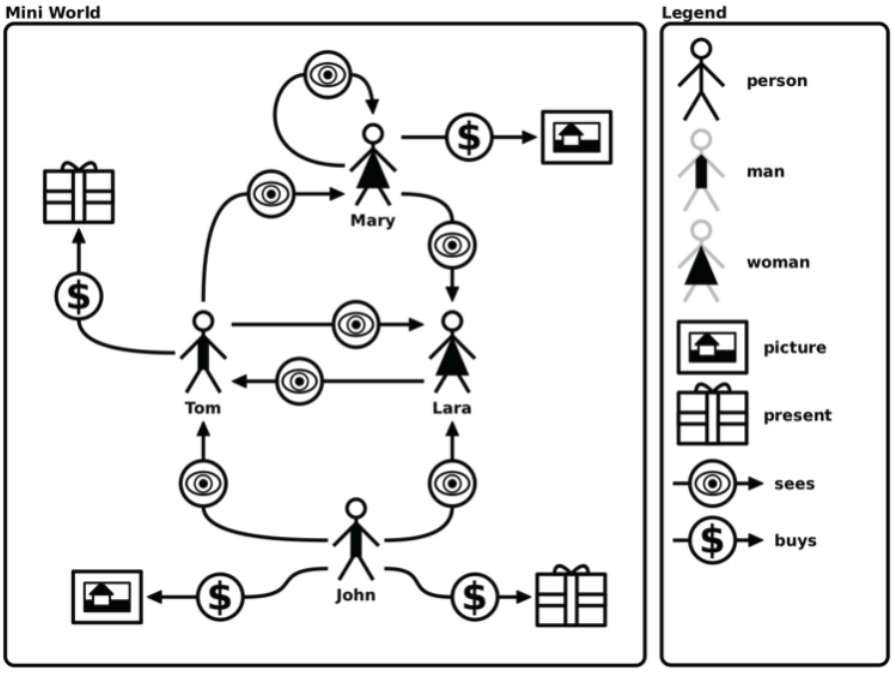


Figure 1. Ontograph

For his experiments Kuhn introduces ontographs, graphical notations of scenes called mini worlds together with a legend. The legend introduces types and (binary) relations, while the mini world shows individuals, their types and their relations. The prime quality of ontographs is that they provide concise, unambiguous and nonverbal descriptions of situations. Figure 1 shows an example.

In his first experiment Kuhn investigates the understandability of ACE by combining an ontograph with a set of 20 ACE sentences of which about half are true with respect to the depicted mini world, while the rest is false. Following is the set of sentences for the ontograph shown in figure 1.

John sees Tom. Lara sees Mary. Mary does not see Tom. Tom does not see Lara. Tom buys a picture. John buys a present. John sees no woman. Mary sees no man. Tom sees every woman. Lara sees every man. Tom sees nothing but women. John sees nothing but men. Lara buys nothing but presents. Lara buys nothing but pictures. No woman sees herself. No man sees himself. Every woman buys nothing but pictures. Every man buys nothing but presents. No man who buys a picture is seen by a woman. No woman who buys a picture is seen by a man.

The task of the 15 test persons consisted in identifying which of the 20 ACE sentences associated with the ontograph is true or is false. The test was performed with four ontographs of increasing complexity. The experiment showed that overall 83% of the ACE sentences were classified correctly. If three logically difficult sentences were not taken into account then the correct classification rose to almost 88%.

In a second experiment Kuhn compared the understandability of ACE with the understandability of a modified version of OWL's Manchester Syntax[2], a visibly formal language called MLL [3]. Each ontograph is accompanied by 20 ACE sentences or, alternatively, 20 logically equivalent MLL statements, both again about half correct and half incorrect with respect of the depicted mini world. The results with 64 test persons working with four diverse ontographs showed clearly, and statistically significantly, that ACE is better understood than MLL.

Kuhn concludes that his experiments clearly show that ACE is generally well understood by persons not familiar with ACE, and that ACE is understood significantly better than MLL.

3. Testing the Understanding of ACE Texts with Kuhn's Ontographs

As it happens, Kuhn's ontographs – by their very nature of being concise and unambiguous nonverbal descriptions of situations – can also be used to investigate the question whether a reader of an ACE text will understand the text as intended by its author.

Here is the outline of an experiment that is based on Kuhn's first experiment and that distinguishes between the role of the author and the role of the reader. The author takes an ontograph and generates a set of true ACE sentences describing the ontograph. For concreteness, let us assume that the author chooses the ontograph of figure 1 and the "true" subset of its associated ACE sentences, namely

John sees Tom. Mary does not see Tom. John buys a present. Mary sees no man. Tom sees every woman. Tom sees nothing but women. Lara buys nothing but presents. Lara

[2] https://www.w3.org/TR/owl2-manchester-syntax/

buys nothing but pictures. No man sees himself. Every woman buys nothing but pictures. No man who buys a picture is seen by a woman.

The reader – who acts as test person – is now shown the above ACE sentences together with a small set of ontographs including the one that is described by the ACE sentences. The reader's task is then to identify the correct ontograph.

For lack of resources I did not do the experiment, but – given the excellent results of Kuhn's two experiments – I am highly confident that it would show that the test persons can successfully relate a set of true ACE sentences to the correct ontograph.

However, a positive outcome of the experiment should only be considered as a supportive argument, but not as a proof, that the reader of an ACE text will understand it in the way that the author intended. I have two reasons for this reservation. First and most importantly, the ACE texts associated with ontographs consist only of individual, unrelated sentences, they do not constitute one coherent text. Second, Kuhn's texts use only a subset of ACE, namely those ACE constructs that can be mapped to the semantic web language OWL. Some commonly occurring constructs are missing, for example explicit if-then sentences, anaphoric references interrelating sentences, intransitive and ditransitive verbs, modifiers of nouns and verbs.

When present, these ACE constructs lead to more structured, more connected and more involved texts that go beyond what can be expressed by ontographs. Alternative approaches are needed to investigate the understanding of these texts. One such alternative is offered by human conversation.

4. Understanding ACE Texts in the Context of Human Conversations

Conversations are interactive verbal communications between two or more people that underlie explicit and implicit social rules, linguistic rules and timing constraints [4]. Most important in our context is the mutually expected and required cooperation between speakers and listeners to achieve a successful communication, i.e. a clear and immediate understanding of the information exchanged. Clark [5] and Brennan [6] introduced the concept of grounding in conversations that comprises the collection of "mutual knowledge, mutual beliefs, and mutual assumptions" that is essential for the communication between people. Successful grounding in communication requires all parties "to coordinate both the content and process". Conversations usually consist of a series of utterances and possibly clarifying questions, where speakers and listeners take turns in order to gradually complete the communication, but it is also possible that a single exchange takes place between one speaker and one or more listeners.

I contend that a form of cooperation, a form of grounding, is also required between the author and the reader of an ACE text to make sure that the reader understands the intentions of the author. This cooperation, too, requires explicit and implicit rules to be successful, details of which we will see in the following. This motivates me to consider the relation between author and reader as a form of conversation. The analogy turns out to be fruitful with respect to the question whether and how far a reader understands the authors intentions, though it has its limitations and peculiarities: the communication is not verbally, but in written form; there is no taking turns of author and reader, but only one exchange of a complete text; a reader usually cannot ask an author clarifying questions that are often an integral part of a verbal conversation.

Continuing with the analogy I define a conversation of this kind as successful if the reader understands the ACE text as its author intended.

Next we will have a closer look at the necessary contributions of the author and the reader to achieve this goal.

5. The Contributions of the Author

The task of authors to write clearly and to keep their readers in mind has been spelled out since ancient times by many teachers of good writing style. Here I will focus on those aspects that are related to controlled languages, specifically to ACE. I will assume that the author is sufficiently familiar with ACE.

The ACE Trouble Shooting Guide[3] gives some general hints on the construction and interpretation of an ACE text that can be summarised as "Though the ACE parser will unravel every syntactically correct sentence, however complex, you [and your readers] may have problems to do so. Thus keep your sentences short and simple."

These hints are similar to the writing rules defined for Simplified Technical English STE[4], but are not as detailed and do not radically restrict the length of certain language constructs. The decisive difference between ACE and STE is that the grammatical correctness of ACE texts can be checked by parsers while there are no checkers for STE. The hints of the ACE Trouble Shooting Guide also recall the Guidelines on Transparency defined for the EU General Data Protection Regulation (GDPR).

The major part of the ACE Trouble Shooting Guide is dedicated to trouble shooting per se. The use of each ACE construct is examined in great detail, possible pitfalls and their avoidance are considered, and solutions for often occurring problems suggested.

Though hints, guidelines and writing rules contribute to the understanding of the author's text they do not address the essential and most critical point, namely the content that is to be communicated to the reader.

Effective communication requires – as I stated above – a cooperation between the author and the reader. In the context of verbal conversations Grice introduced the cooperative principle[5] that quite naturally can also be applied to the author-reader case. In the author-reader case, the cooperative principle binds foremost the author as the one who produces the text, but also the reader who relies on the author to adhere to the principle. From the cooperative principle Grice derived four maxims that substantiate it. I rephrase Grice's maxims to be suitable for the author-reader case and for ACE, and add some clarifying comments.

- Maxim of Quality
  The text should be true. Though using an English syntax, ACE is a formal logic language. Thus an ACE texts can be given a model-theoretic or a proof-theoretic meaning [7]. Considered model-theoretically, an ACE text is the description of a modelled domain in which case – as Kuhn's ontograph experiment shows – some sentences can describe the domain correctly, while others do not. This approach is also called truth-conditional since the correct sentences are labelled true, the others false. Considered proof-theoretically, the sentences of an ACE text are not

[3] http://attempto.ifi.uzh.ch/site/resources/

[4] http://www.asd-ste100.org

[5] https://en.wikipedia.org/wiki/Cooperative_principle

interpreted with respect to a modelled domain, but are assumed to describe a true state of affairs. The meaning of a sentence is accordingly no longer described in terms of truth-conditions but in terms of the logical inferences that can be drawn from that sentence. It remains the responsibility of the writer to avoid contradictions. For computational purposes – for instance to answer questions from the ACE text – the proof-theoretic approach is preferred since it supports logical deduction on a syntactic not on a semantic basis. The Attempto reasoner RACE that will be presented later as a tool for the reader interprets an ACE text proof-theoretically.

- Maxim of Quantity
  The text should be complete, but should not contain additional information. As the experience with requirement specifications and instruction manuals shows this maxim is hard to fulfil, and often it is not. This maxim makes the most exacting demand on the author of an ACE text.
- Maxim of Relevance
  The text should be relevant to the topic described. For ACE texts this maxim seems to coincide with the maxim of quantity.
- Maxim of Manner
  The text should be perspicuous. This maxim should restrain the author from creating convoluted texts and from showing off. Following this maxim is made easier by the unambiguousness of ACE and by the guidelines of ACE's Trouble Shooting Guide.

There are potential pitfalls in ACE relevant for the above maxims. ACE's handling of anaphoric references and ambiguities can lead even people familiar with ACE into a conflict between the reading enforced by ACE's interpretation rules and a reading suggested by common sense. The author of an ACE text must be aware of these pitfalls that, if not avoided, could also negatively affect the reader of the ACE text.

Here is an – admittedly contrived – case of misleading resolutions of anaphoric references. The author wants to express that a manager calls a subordinate to give the subordinate some order, and writes

A manager calls a subordinate. He gives him some orders.

In standard English this would be understood as intended since we use context and common sense to correctly resolve the anaphoric references *he* and *him*. However, according to the ACE interpretation rules the pronoun *he* of the second sentence refers to the *subordinate* and the pronoun *him* to the *manager*, thus incorrectly stating that the subordinate gives orders to his superior. The author can easily avoid this unintended meaning using more explicit anaphoric references, for example

A manager calls a subordinate. The manager gives the subordinate some orders.

Next a potentially misleading case of ambiguity and anaphora resolution – even more contrived.

A girl owns a telescope. A girl owns a camera. A man sees the girl with the telescope.

Reading this in standard English we most probably would interpret the third sentence to mean that the man sees the girl that owns the telescope. However, according to the ACE interpretation rules the third sentence has quite another meaning: the anaphoric reference *the girl* refers to the textually closer *a girl* in the second sentence, the one that owns a camera, the prepositional phrase *with the telescope* modifies the verb *see* not *the girl*, and *the telescope* refers to *a telescope* of the first sentence. To express the "common sense meaning" the author could, for instance, write

A girl owns a telescope. A girl owns a camera. A man sees the girl that owns the telescope.

Of course, contriving cases where the "standard English meaning" and the "ACE meaning" clash does not properly depict the reality of using ACE where such clashes are rare and – as demonstrated – easily avoided.

6. The Contributions of the Reader

I assume that also the reader is familiar with ACE. Furthermore, I assume that the reader cannot contact the author, but has only the author's complete ACE text available.

An important concept in verbal conversation is repair [4], elsewhere called "establishing a common ground" [5,6]. Repair can be performed by the speakers who correct themselves while speaking, but in most cases it is the responsibility of the listeners to contribute to the success of the conversation by asking clarifying questions or by showing signs of not understanding.

Defining the relation between the author of an ACE text and its reader as conversation, I must also redefine the concept "repair" given that the reader cannot ask clarifying questions or express lack of understanding. How can repair – understood as an attempt to better understand the ACE text – be defined in these circumstances?

My answer is logical deduction for which ACE is supported by several automated tools. Best suited as a tool for repair is RACE [8,9], a first-order reasoner with equality that can show the (in-) consistency of an ACE text, deduce one ACE text from another one and answer ACE questions on the basis of an ACE text. All three capabilities of RACE – consistency checking, deduction, question answering – can help the reader to better understand the given ACE text, to "repair" the author-reader conversation in the newly defined sense.

To begin with, I will focus on RACE questions that can extract explicit and implicit information from the ACE text. RACE offers altogether 11 forms of questions that we will use to investigate the following example ACE text.

John's red cat catches a mouse in a garden. Some black cats of John sleep silently. No red cat is a black cat. Every cat is an animal. Every mouse is an animal.

Most general are *yes/no* questions that just ask for the existence or non-existence of facts, for instance

Is there a black cat that sleeps? Does a mouse sleep? Is a cat a mouse?

If a *yes/no* question can be answered then RACE will show all answers and for each answer all sentences of the ACE text needed for that answer. For lack of space, I will omit here and in the following the – mostly obvious – answers generated.

Detailed information can be gained by *wh*-questions, i.e. questions with query words. If a *wh*-question can be answered then RACE will show all ACE sentences needed for the answer and, most importantly, the substitutions for the query words. Again, there can be more than one answer.

- Asking for subjects and objects of sentences is done by *who*-questions, *whose*-questions, *what*-questions and *which*-questions.

  Who catches what in a garden? Whose cats sleep? Which cat sleeps?
- Asking for adverbs or prepositional phrases that modify verbs is done via *how*-questions, *where*-questions and *when*-questions.

  Where does a cat catch a mouse? How does a black cat sleep?
- Asking for the cardinality of countable objects or for the amounts of mass

objects is performed by *how-many*-questions and *how-much*-questions.
There are how many cats? There are how many animals? How many cats sleep?

- Finally, asking for verbs can be done via *do*-questions.
What does a red cat do? What do some black cats do? What does a cat do?

In human conversations presuppositions[6] – implicit assumptions based on shared common knowledge – and implicatures[7] – information that is only suggested by a conversation – play an important role. Thus also the reader of an ACE text may be tempted to interpret the text beyond what it concretely states, respectively seems to logically imply.

Actually, this behaviour is justified in a few cases. RACE relies for its deductions on built-in auxiliary axioms that express domain-independent common knowledge, for example the relation between singular and plural nouns, or the laws of arithmetic. Some of the deductions using auxiliary axioms have the quality of presuppositions or implicatures since they add information that is not explicit in the ACE text. Using the above example text RACE can, for instance, derive

- for a noun its existence expressed by *there is* or *there are*
There is a red cat. There are some cats. There is a mouse. There is a garden.
- for a noun its replacement by *somebody* or *something*
There is somebody. Somebody's cats sleep. Somebody catches something.
- from possessives the ownership expressed by the verb *have*
John has some black cats.
- from *some X* either *one X* or *at least one X* or *more than one X* or *two X* or *at least two X*
At least one black cat sleeps. More than one black cat sleeps.

Other presuppositions or implicatures are not supported by RACE because they are either domain-dependent or too specific.

To sum up, RACE – offering a method of "repair" of the author-reader relation – provides the reader with powerful methods to investigate the ACE text at hand, and ultimately to understand it better.

7. The Role of ACE Comments

RACE offers many powerful forms of questions, but leaves out one highly important for the understanding of the ACE text, namely the question "why?" that would reveal the intentions and decisions of the author.

The question "why?", and similar ones, cannot be answered by the ACE text since they would have to provide information about the text itself, that is meta-information for which there are no – and there cannot be – ACE language constructs.

Being privy to authors meta-information must be provided by them. One way to do so is offered in the form of ACE comments that so far have not found their proper attention as an important means of information transfer between authors and readers.

---

[6] https://en.wikipedia.org/wiki/Presupposition

[7] https://en.wikipedia.org/wiki/Implicature

ACE comments have the advantage that they can contain any information using the less constrained standard English – which, alas, could also be the cause of further misunderstandings.

To the above example ACE text the author could add a comment to explain why the sentence *No red cat is a black cat.* is necessary to correctly count the cats.

...
 # Red cats and black cats must be declared distinct to correctly count all cats.
No red cat is a black cat.
...

Comments can also be used by authors for other forms of meta-information that need to be transmitted to the reader, for instance clarifying examples. Furthermore, comments can provide helpful structuring information of the ACE text, like titles of chapters and sections.

8. The Cooperation between Authors and Readers Ensures Understanding

The question whether and to which extent the reader of an ACE text understands the text as the author intended is a difficult one.

To find an answer to this question I first suggest an experiment based on Kuhn's ontographs. Users have to identify the ontograph described by a given ACE text. This experiment is instructive, but – using a subset of ACE in a restricted way – has only limited evidence with respect to the question of understanding.

My second approach is based on the analogy of the author-reader relation to human verbal conversations. This analogy turns out to be fruitful since it allows me to show that the understanding of an ACE text does not concern the reader alone, but necessarily requires contributions from both the author and the reader. For authors I derive guide-lines for the presentation and for the content of the ACE text. I also offer solutions to possible pitfalls. For readers I suggest logic deduction as a powerful tool to increase the understanding of the ACE text.

Furthermore, I advocate ACE comments for the transmission of important meta-information between author and reader.

To summarise: My investigations validate our claim that a given ACE text can be understood by any reader – however only provided that both the author and the reader of the ACE text cooperate to ensure a successful communication.

Acknowledgements

I would like to thank the Institute of Computational Linguistics, University of Zurich, for its hospitality. Furthermore, I very much appreciate the critical and constructive comments of the four anonymous reviewers of a previous version of this paper.

References

1. A. Al-Rawas, S. Easterbrook, Communication Problems in Requirement Engineering: A Field Study, Proc. First Westminster Conference on Professional Awareness in Software Engineering, London, 1996.
2. L. Li, *Desiree: a Refinement Calculus for Requirements Engineering*, PhD Thesis, DISI, Università degli Studi di Trento, 2016.
3. T. Kuhn, *Controlled English for Knowledge Representation,* Doctoral thesis, Faculty of Economics, Business Administration and Information Technology of the University of Zurich, 2010.
4. N. J. Enfield, *How we talk. The inner workings of conversation,* Basic Books, 2017.
5. H. H. Clark, S. E. Brennan, Grounding in communication. In L. B. Resnick et al (eds.), *Perspectives on socially shared cognition,* Washington, DC: American Psychological Association (1991) 127–149.
6. S. E. Brennan, The Grounding Problem in Conversations With and Though Computers. In S. R. Fussell, R. J. Kreuz (eds.), *Social and cognitive psychological approaches to interpersonal communication,* (1998), 201-225.
7. U. Schwertel. *Plural Semantics for Natural Language Understanding – A Computational Proof-Theoretic Approach*, PhD thesis, University of Zurich, 2004.
8. N. E. Fuchs, *First-Order Reasoning for Attempto Controlled English*, Proceedings of the Second International Workshop on Controlled Natural Language (CNL 2010), Springer, 2012.
9. N. E. Fuchs, *Reasoning in Attempto Controlled English: Non-Monotonicity,* Proceedings of the Fifth International Workshop on Controlled Natural Language (CNL 2016), Springer, 2016.

Controlled Natural Language
B. Davis et al. (Eds.)

doi:10.3233/978-1-61499-904-1-85

Rewriting Simplified Text into a Controlled Natural Language

Hazem Safwat, Manel Zarrouk, Brian Davis
Insight Centre for Data Analytics,
National University of Ireland,
Galway, Ireland
{*hazem.abdelaal, manel.zarrouk, brian.davis*}*@insight-centre.org*

Abstract. While machine processable Controlled Natural Languages (CNLs) as a natural language interface have proven a popular, unambiguous and user friendly method for non experts to engineer formal knowledge-bases, human-oriented CNLs however remain under-researched despite having found favor within industry over many years. Whether such human orientated CNLs like the machine processable counterparts can be captured automatically as formal knowledge remains an open question. In addition, rewriting all or most of a human-oriented CNL into a machine-oriented CNL could unlock significant silos of general purpose domain knowledge, contained within existing human-oriented CNL content for exploitation by knowledge based systems. This paper explores the feasibility of rewriting a human-orientated CNL represented in Simplified English into a well know machine-oriented CNL represented in ACE CNL and describes preliminary results.

Keywords. Controlled Natural Language, Natural Language Processing, Knowledge Extraction, Semantic Web

1. Introduction

Knowledge extraction from text is still a problem that is not fully solved. There are a variety of advantages that makes solving this problem crucial in the community. Most of these advantages lie within the number of applications that can benefit from the extracted knowledge. These applications include but are not limited to generating quality linked data and ontologies from text, and generating machine-readable content to support Semantic Web Technologies. Although there are a lot of benefits, there are also many challenges with respect to the current approaches to ontology learning and population from text [1]. These limitations varies from the time needed to train non-expert users, to the time and effort needed to prepare the generated linked data to fit with different schemas and languages such as OWL. Some requirements presented in [2], are defined to help solve these problems. The requirements are related to mapping natural language to a well defined model representation (i.e. OWL), representing complex relations in text, and reduce or eliminate the need for training of non expert users for ontology engineering. However aside from explicit knowledge gathering and engineering activities machine orientated CNLs offer little incentive to the average user to create formal knowledge. A subcategory of CNL which offers a middle ground of reduced ambiguity

for language processing but less restriction than a machine-oriented CNL is a human-oriented CNL [8]. Their origins are motivated for the purposes of language learning, and unambiguous communication between humans in a domain context. An example of a human-oriented CNL is Simple English used to author Simple Wikipedia[1].

In this paper, we argue that rewriting a human orientated CNL (Simplified English) into a machine processable CNL allows us to reap the benefits of machine processable while simultaneously circumventing the barrier with respect to uptake of machine processable CNLs by users working outside of the knowledge engineering context. Finally, rewriting all or most of a human-oriented CNL content into a machine-oriented CNL could unlock significant rich silos of implicit general purpose formal knowledge, contained within existing human-oriented CNL content such as Simple Wikipedia.

The paper is structured as follows: Section 2 describes related work, Section 3 presents the approach, processing and analysis. In Section 4, we discuss some initial results with examples, and finally, Section 5 offers a conclusion.

2. Related Work

Attempto Controlled English (ACE) [3] is a widely known CNL that is mainly designed for knowledge representation. ACE texts are both human readable and machine processable that can be unambiguously mapped into different formal language such as Discourse Representation structures (DRS). The DRS is a variant of first order logic and its output of ACE texts can be bidirectionally translated into Web Ontology Language (OWL). Beside all the previous reasons, we choose ACE for testing our rewriting system as it provides open access to its tools and resources. One of these resources is the ACE parsing engine - APE[2], which we used for our system validation and for generating the DRS and OWL outputs.

The work from [5] proposed a sentence rewriting system based on semantic parsing that can rewrite a sentence into a new form while keeping the same target logical form. The rewriting system is trying to resolve the problem of vocabulary mismatch between natural language and ontology. This mismatch happens due to the various expressions in natural language and the fact that one can express the same meaning using different expressions and sentence structures. The system is supported by a ranking approach to select the best rewriting using the features of the semantic parsing and rewriting. Our work is similar to them as we also rewrite sentence into a new form that have the same target logical form. However, the difference is that our efforts are directed towards generating a machine processable text in the form of CNL that can be an alternative to the simplified text and can be mapped into triples.

In [4] the authors present an approach which is based on text rewriting for the aim of automatically generating labeled data that can be used for model training. The approach is implemented after analysing Simple Wikipedia and their parallel Wikipedia texts and extracting some rewrite rules. These rules are then used to produce different structures of sentences that are annotated with gold standard labels. Our work is similar to them as we used Simple Wikipedia and their parallel Wikipedia abstracts for analysing their texts and we used rewrite rules. However our analysis was performed to measure the overlap

[1] https://simple.wikipedia.org/wiki/

[2] https://github.com/Attempto/APE

between the properties of both texts and CNLs, and our rewrite rules are implemented to produce CNL alternatives of the simplified text for the aim of extracting OWL triples.

Finally, with respect to human-oriented CNLs, ASD Simplified Technical English[3] was developed to improve the readability and comprehensibility in technical documents. Another example is Boeing Technical English to improve the communication between people for air traffic control [9]. The development and planning of these CNLs appears often community driven, like Simplified Wikipedia, to have been based on Basic English [10].

3. Methodology

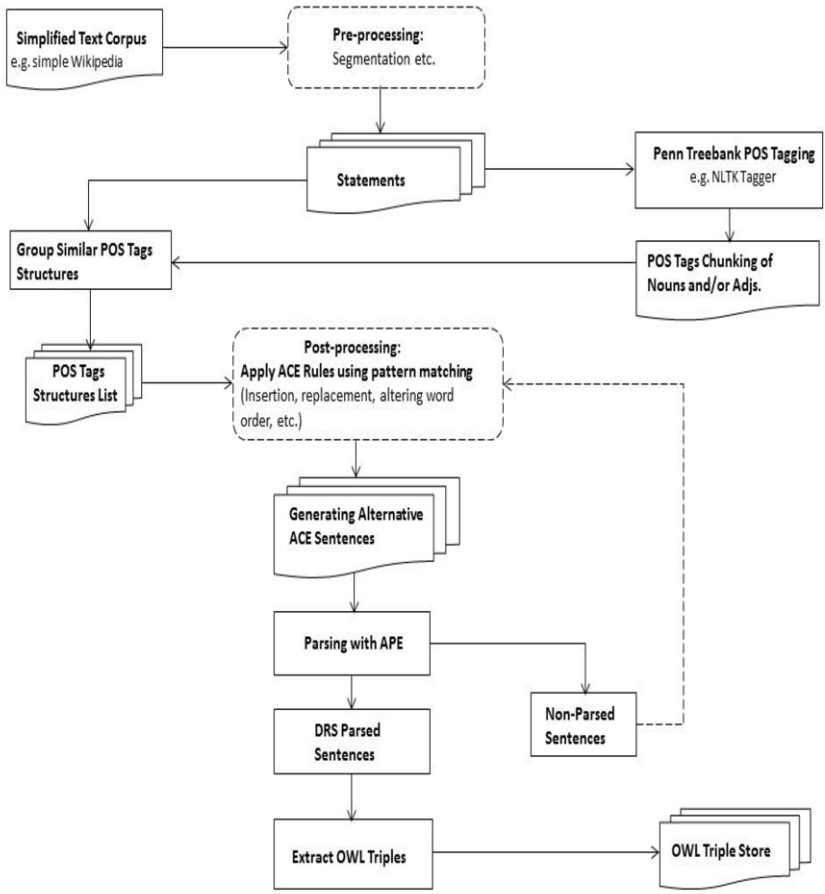


Figure 1. System architecture for rewriting simplified text into ACE CNL to extract OWL Triples.

The first step in our experiment involved collecting the dumps from both the Simple Wikipedia abstracts and its parallel Wikipedia abstracts. The main aim is to analyse the

[3] http://www.asd-ste100.org/

Table 1.: The most 5 common Chunked POS structures with an example of grouped POS structures (Descending) .

Chunked POS tags structures	Examples of grouped POS tags structures	Total No. of grouped POS tags structures	Total No. of sentences
CHUNK VBZ DT CHUNK IN CHUNK	• (NNP NNS) VBZ DT NN IN NN • (NNP NN) VBZ DT NN IN NNS	366	2664
CHUNK VBZ DT CHUNK IN DT CHUNK IN CHUNK	• NNP VBZ DT NN IN DT (JJ NN) IN NNP • (NNP NNP) VBZ DT NN IN DT NN IN NNP	224	926
DT CHUNK VBZ DT CHUNK IN CHUNK	• DT (NNP NNP) VBZ DT NN IN NNP • DT NNP VBZ DT NN IN (JJ NN)	194	469
CHUNK VBZ DT CHUNK IN DT CHUNK	• NNP VBZ DT NN IN DT (NNP NNP) • (NNP NNP) VBZ DT NN IN DT NNP	158	318
CHUNK VBZ DT CHUNK	• (NNP NNP) VBZ DT NN • NNP VBZ DT (JJ NN)	125	1399

text properties of each one and see if there is an overlap with the CNLs properties in [6]. The total number of sentences in Simple Wikipedia was 87k sentences and 39.2k in the parallel Wikipedia sentences. As shown from the system architecture in figure 1, text passes by a pre-processing module that involves text cleansing using regular expressions, then segmentation to split abstracts into sentences and tokenization to split sentences into tokens. The NLTK POS tagger that uses Penn Treebank [7] is applied on the sentences to analyse POS structures in each corpus. Our intuition is that, if the authors followed the guidelines in [6] to write the Simple Wikipedia abstracts, then there should be many similar POS tags structured sentences in the Simple Wikipedia corpus. From our analysis, we found that the sentences in the Simple Wikipedia abstracts overlap with the CNL properties more than the sentences written by authors in their parallel Wikipedia abstracts for instance the maximum tokens/sentence in the Simple Wikipedia corpus is always less than or equal 20 tokens as recommended in the CNL properties. In addition the use of passive voices is found only in approximately one third of the sentences, in contrary to the Wikipedia corpus where passive voices were found in more than half of the corpus.

The next step is extracting all the abstracts that follow the CNL rules defined in [6], which represent a total number of 36.5% sentences. Then, analyse the POS structures of the sentences to see to what extent the authors of Simple Wikipedia abstracts used the sentence forms such as `Subject-Verb-DirectObject`,

Table 2.: Examples of simplified text sentences after their rewriting into alternative ACE and generating their equivalent DRS and OWL outputs.

Simplified text sentences	Alternative ACE sentences	DRS Parser	OWL Parser
• Parametric statistics is a branch of statistics. • Herbs zoster is a disease in Humans.	• Parametric-statistics is a branch of Statistics. • Herbs-zoster is a disease in Humans.	Accepted	Accepted
• Lucca is a city in the Italian region of Tuscany. • Rio Cuarto is a city in the center of Argentina.	• Lucca is a city in the n:Italian-region of Tuscany. • Rio-Cuarto is a city in the center of Argentina.	Accepted	Accepted
• The Moscow Zoo is a zoo in Moscow. • The Manx is a breed of domestic cat.	• The n:Moscow-Zoo is a zoo in Moscow. • The Manx is a breed of the n:domestic-cat.	Accepted	Accepted
• Lubbock is a city in the United States. • North Holland is a province of the Netherlands.	• Lubbock is a city in the n:United-States. • North-Holland is a province of the Netherlands.	Accepted	Accepted
• Guilford County is a county. • Lyriel is a German band.	• Guilford-County is a county. • Lyriel is a n:German-band.	Accepted	Accepted

and `Subject-Verb-IndirectObject`[4]. The analysis showed that around 12.6k sentences can be grouped into 629 POS tags structures, which represent 34.5% from the total number of Simple Wikipedia sentences that can be mapped to ACE CNL. Hence we would need to implement rules that can cover the 629 different POS structures. We concluded that the percentage is not high, if we compared this to the total number of sentences which is 36.5k sentences. A deeper analysis is performed on the POS structures, where a large percentage from the POS structures could be grouped together as they contain noun and/or adjective clusters. So, after implementing a noun/adjective chunker, a new group of POS structures are created , where a total number of 19.2k sentences are grouped together under 300 POS structures.

From the previous analysis we can conclude that, if we can implement rules to rewrite the 300 POS tags structures into ACE CNL, then we can generate around 19.2k sentences into ACE CNL format and consequently into the DRS formal representation and exported to OWL Triples.

[4] `https://simple.wikipedia.org/wiki/Wikipedia:How_to_write_Simple_English_pages`

4. Initial Results

The system represented in the architecture shown in Figure 1 is implemented and applied on the 19.2k sentences that follow the CNL rules. In Table 1, we show the top five common chunked POS structures in the corpus, ranked in descending order by the total number of grouped POS structures under each chunked POS structure. The first column represents chunked POS structures, where the POS tag `CHUNK` refers to a one or more nouns and/or adjectives. Since, nouns can be found in different forms e.g singular, plural, noun phrases..etc, all of these forms are taken into consideration in the chunking process. The second column in the table shows a couple of examples of the original POS structures that are grouped under the chunked POS structure, with their total number shown in the third column. Finally, the last column shows the total number of sentences in the corpus that are grouped under this chunked POS structure.

Some examples that extend table 1 are shown in table 3, such that each row in table 3 is an extension of the same row in table 1. The *n:* prefix represents the chunked nouns performed by the chunker and the rewriting rules.

The first column in Table 3 contains some example Simplified Text sentences from the corpus. In the second column we show the sentences after implementing and applying the architecture in figure 1. The third and fourth columns, confirm the generated ACE alternatives are accepted and parsed by the ACE parser and the DRS and OWL outputs are generated, as shown in Table 3.

Table 3.: The DRS and OWL outputs from the ACE parser for an example sentence.

Metric	Result
Simplified text	Parametric statistics is a branch of statistics.
Equivalent ACE text	Parametric-statistics is a branch of Statistics.
DRS	[A,B]object(A,branch,countable,na,eq,1)-1/4 relation(A,of,named(Statistics))-1/5 predicate(B,be,named(Parametric-statistics),A)-1/2
OWL XML	$<Ontologyxml:base="http://www.w3.org/2002/07/owl\#"$ $xmlns="http://www.w3.org/2002/07/owl\#"$ $ontologyIRI="http://attempto.ifi.uzh.ch/ontologies/owlswrl/test">$ $<ObjectPropertyAssertion>$ $<ObjectPropertyIRI="http://attempto.ifi.uzh.ch/ontologies /owlswrl/test\#branch"/>$ $<NamedIndividualIRI="http://attempto.ifi.uzh.ch/ontologies /owlswrl/test\#Statistics"/>$ $<NamedIndividualIRI="http://attempto.ifi.uzh.ch/ontologies /owlswrl/test\#Parametric-statistics"/>$ $</ObjectPropertyAssertion></Ontology>$

5. Conclusion

This paper presented some initial results of our work towards rewriting simplified text (Simple Wikipedia) into a human-readable and machine-processable text (ACE CNL). Our initial results showed that, the features of simplified text are close to the features of CNL than unstructured text when users follow the authoring guidelines. We showed also that simplified text can be rewritten into a machine processable format (CNL) and can be used for knowledge extraction by extracting DRS and OWL triples. The approach

could be exploited to generate formal background knowledge for variety of applications automated reasoning, decision support or ontology aware NLP applications. The next steps will be applying the approach on all the abstracts of Simple Wikipedia to generate DRS and OWL triples, where this extracted structured knowledge could also be linked to a knowledge base such as DBpedia[5].

Acknowledgment

This work has emanated from research supported in part by a research grant from Science Foundation Ireland (SFI) under Grant Number SFI/12/RC/2289 and in part by the SSIX Horizon 2020 project (grant agreement No 645425).

References

[1] Wong, Wilson, Wei Liu, and Mohammed Bennamoun. "Ontology learning from text: A look back and into the future." ACM Computing Surveys (CSUR) 44.4 (2012).

[2] Draicchio, Francesco, Aldo Gangemi, Valentina Presutti, and Andrea Giovanni Nuzzolese. "Fred: From natural language text to rdf and owl in one click." Extended Semantic Web Conference. Springer, Berlin, Heidelberg, 2013.

[3] Fuchs, Norbert E., Kaarel Kaljurand, and Tobias Kuhn. "Attempto controlled english for knowledge representation." Reasoning Web. Springer, Berlin, Heidelberg, 2008. 104-124.APA

[4] Woodsend, Kristian, and Mirella Lapata. "Text rewriting improves semantic role labeling." Journal of Artificial Intelligence Research 51 (2014): 133-164.

[5] Chen, Bo, et al. "Sentence rewriting for semantic parsing." Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). Vol. 1. 2016.

[6] OBrien, Sharon. "Controlling controlled english. an analysis of several controlled language rule sets." Proceedings of EAMT-CLAW 3 (2003): 105-114.

[7] Marcus, Mitchell P., Mary Ann Marcinkiewicz, and Beatrice Santorini. "Building a large annotated corpus of English: The Penn Treebank." Computational linguistics 19.2 (1993): 313-330.

[8] Hujisen WO. 1998. *Controlled language: an introduction*. Second International Workshop on Controlled Language Applications (CLAW), Pittsburgh.

[9] Wojcik, Richard H., Heather Holmback, and James Hoard. 1998. *Boeing Technical English: An extension of AECMA SE beyond the aircraft maintenance domain*. Second International Workshop on Controlled Language Applications (CLAW), Pittsburgh.

[10] Ogden, Charles Kay. 1944. *Basic English: A general introduction with rules and grammar*. Vol. 29. K. Paul, Trench, Trubner.

[5] `http://wiki.dbpedia.org/`

Controlled Natural Language
B. Davis et al. (Eds.)

doi:10.3233/978-1-61499-904-1-92

Modelling Negation of the Afrikaans Declarative Sentence in GF

Laurette PRETORIUS [a,1], Laurette MARAIS [b]
[a] *University of South Africa, Pretoria, South Africa*
[b] *Meraka Institute, CSIR, Pretoria, South Africa*

Abstract. The correct modelling of negation in a computational grammar is essential in order for the grammar to be useful in natural language processing, and controlled language development and applications. An important and unique feature of Afrikaans is how it deals with negation. We present an exposition of a substantial fragment of negation in Afrikaans and discuss our implementation thereof in GF. Examples are given as illustration of the relevant issues. The paper is concluded with a discussion of results and plans for future work.

Keywords. Controlled language, Afrikaans, syntax, negation, Grammatical Framework

1. Introduction

The correct modelling of negation in a computational grammar is essential in order for the grammar to be useful in natural language processing, controlled language development and their applications. A controlled natural language (CNL) is "a constructed language that is based on a certain natural language, being more restrictive concerning lexicon, syntax, and/or semantics while preserving most of its natural properties" [6]. The development and use of controlled natural languages has been dominated by English [6]. However, the interest in multilingual CNLs continues to grow as the Semantic Web and its expectations in terms of knowledge representation and reasoning in and across the many world languages increase [9,10].

Grammatical Framework (GF), a computational grammar development environment [8], has become a *de facto* standard for conceptualising, developing and using multilingual CNLs in multilingual applications. Any given GF grammar is by definition a CNL, and so-called GF resource grammars may be used by GF application grammars to implement various domain specific CNLs. GF allows a clear distinction between so-called syntactic and semantic grammars. A syntactic (resource) grammar models the morphological and syntactic structure of a given language, while GF semantic grammars may use resource grammar modules. This enables developers of multilingual CNLs and their applications to focus on the domain specific and semantic aspects of the application. GF resource grammars form part of GF's Resource Grammar Library (RGL). At the time of writing 32 languages were represented in the GF RGL.[2]

[1] Corresponding Author: Laurette Pretorius, PO Box 392, University of South Africa, Pretoria 0003, South Africa; E-mail: pretol@unisa.ac.za.

[2] `https://www.grammaticalframework.org/lib/doc/synopsis.html`

Afrikaans is one of the eleven official languages of South Africa. It is the mother tongue of 13.5% of the population of approximately 57 million. There exists a GF resource grammar (RG) for Afrikaans, as well as one for Dutch[3], its mother language. We adapted the Afrikaans RG from the existing Dutch in 2011, which in turn was adapted from the German. This repeated adaptation, as well as possible limitations in the Dutch and German RGs at the time, resulted in an RG that did not model the properties of Afrikaans systematically. As a result, this Afrikaans RG contained various inaccuracies, especially with regards to negation, and it was also cumbersome to debug. Furthermore, Afrikaans negation is significantly different from that of Dutch, and indeed all other languages [1]. Since negation is a core aspect of a GF RG, we decided to build on a next version of the Afrikaans RG "from scratch" that would be lean, accurate, easier to maintain and would provide the best possible representation of standard Afrikaans in GF. This article concerns the modelling of the declarative sentence in Afrikaans, with specific focus on its negation. To the best of our knowledge, no other attempts have been made to implement a resource grammar for Afrikaans, and indeed, no computational grammars exist which attempt to model negation in Afrikaans in a systematic, general way.

A universal and fundamental property of natural language is that every language is able to express *negation* and therefore has some means to reverse the truth value of a given sentence or utterance. Consequently, providing support for negation is a core function of a GF resource grammar. It is well known that languages may differ as to how they express this negation. Not only do languages vary with respect to the position of negative elements, also the form of negative elements and the interpretation of sentences that consist of multiple negative elements are subject to broad crosslinguistic variation [11]. It is therefore clear that modelling negation in a computational grammar is crucial to the quality and usefulness of the grammar.

Negation is considered one of the most complex aspects of Afrikaans syntax [3,7,4]. Afrikaans is also rather unique in how it expresses negation (see for example [2]). Indeed, it has been shown that with respect to negation, Afrikaans fills a certain "typological gap" that arises in Zeijlstra's feature-based analysis of negation systems [1]. Consequently, any Afrikaans CNL, based on GF, that must express negation would benefit greatly from a resource module that accurately implements this complex aspect of Afrikaans.

In this paper, we focus on the *declarative sentence in standard Afrikaans* and briefly describe the fragment of Afrikaans negation that forms the basis of our computational model. We discuss its implementation in GF and conclude with a discussion of results obtained and work still to be done.

2. Afrikaans negation

Afrikaans exhibits two mechanisms by which a single semantic negation can be expressed, namely the use of two negative markers (*nie ... nie*), and the use of a negative indefinite and a negative marker (*nooit .. nie* etc.).

In Afrikaans, negative indefinites, or n-words, are the nominal and adverbial negative elements *niemand* (nobody), *geeneen* (not one), *nooit* (never), *nêrens* (nowhere), *niks* (nothing), as well as the negative determiners *geen, g'n* (no). In standard Afrikaans these elements always co-occur with a marker of negation, *nie*.

[3]See https://www.grammaticalframework.org

We refer to the two mechanisms for expressing negation as *sentential negation* and *n-word negation*, respectively, and we illustrate them in Example 1.a and 1.b.

(1) a. *Ek sien nie die boom nie.*
I see NEG the tree NEG

'I do not see the tree'

b. *Niemand sien die boom nie.*
No-one sees the tree NEG

'No-one sees the tree'

We follow the analysis of [1], which in the case of sentential negation posits the presence of an abstract negative operator. The negation is syntactically realised by means of the two negative markers *nie ... nie* neither of which in itself carries overt negative meaning. In the case of n-word negation, the n-word *niemand* carries overt negative meaning and requires a concordial negative marker *nie*.

A double negation (DN) interpretation[4] results from either a co-occurrence of two n-word negations in a sentence (see Example 2.a), or the co-occurrence of sentential negation and n-word negation (see Example 2.b). In any case "where two negative markers are spelled out adjacent to one another in the same prosodic domain [...] only one *nie* survives" [1].

(2) a. *Ek sien nooit niemand nie (*nie).*
I see never no-one NEG NEG

'I never see no-one', i.e. 'I always see someone'

b. *Ek sien nie niemand nie (*nie).*
I see NEG no-one NEG NEG

'I don't see no-one', i.e. 'I see someone'

Multiple negations may be expressed in other ways as well. Examples 3.a and 3.b illustrate how only a single *nie* survives in the case where both the main clause and the embedded clause contain the second negative marker.

(3) a. *Ek weet nie dat hy nie die boom sien nie (*nie).*
I know NEG that he NEG the tree see NEG NEG

'I do not know that he does not see the tree'

b. *Niemand weet dat hy nie die boom sien nie (*nie).*
No-one know that he NEG the tree see NEG NEG

'None-one knows that he does not see the tree'

Evidently, an important aspect of modelling Afrikaans negation is to prevent the occurrence of adjacent second negative markers *nie*.

[4]Any proposition S has the same truth value as $\neg(\neg(S))$.

For sentential negation the placement of the verb is specifically relevant. The following aspects concerning Afrikaans word order are therefore notable.

- Although Afrikaans displays a word order asymmetry between subject-initial main clauses (SVO) and subject-initial embedded clauses (SOV), it is generally assumed to be an underlying SOV language with finite verb movement, or Verb Second, in main clauses.
- In main clauses that contain auxiliary verbs, the auxiliaries appear in the second position and the finite verb or past participle in the sentence-final position.
- In embedded clauses both finite verb or past participle and auxiliaries appear sentence-finally. In the case of the temporal auxiliary *het*, this auxiliary occurs in the sentence-final position, directly preceded by the past participle, while in the case of modal auxiliaries, the infinitive appears in the sentence-final position, preceded by the modal auxiliary.

A description of Afrikaans grammar is not within the scope of this paper. For a concise and clear summary, see [5][pp. 21-27].

3. Afrikaans negation in GF

A *GF syntactic grammar* consists of two main kinds of modules, namely a single abstract grammar module and one or more concrete grammar modules. The abstract grammar module defines language independent syntax categories and rules, while a concrete grammar module defines how such categories and rules are realised in a particular language. Specifically, the abstract grammar module contains *categories* and *functions*, which together give rise to abstract syntax trees, while the concrete grammar modules contain the corresponding *linearisation categories* (or *lincats*), which take the form of records containing fields, and *linearisation functions*, which are the means by which syntax trees are linearised as strings.

In implementing a new Afrikaans RG, we follow the typical proposed *modus operandi*, which consists of first implementing an Afrikaans concrete grammar for the so called miniature resource grammar (MRG). The MRG is a GF resource grammar that makes provision for a select number of categories and functions that display the key grammatical structures relating to declarative and question sentences [8][pp. 199-218 and 237]. Our focus is on the declarative sentence and its negation in particular. We use the MRG as a framework for our implementation of negation in the Afrikaans declarative sentence in order to limit the presence of other aspects that might cloud the issues pertaining to negation.

The MRG includes the `Utt` category as its starting category, that is, the category type at the root of each tree structure in the grammar. Table 1 lists the most pertinent syntactic categories used to construct declarative sentences. A clause, which has variable tense and polarity, together with a value for tense and for polarity, yields a sentence. For the Afrikaans concrete grammar module we follow a typical strategy in which the lincat of the clause is essentially a table which provides correct linearisations for each tense and polarity combination.[5] When constructing a sentence, the correct entry is simply chosen

[5]In reality, the lincat for clauses in Afrikaans also has a word order dimension. A detailed discussion of this linguistic feature and its implementation is beyond the scope of this paper.

Categories	Description
`S`	Declarative sentences, with fixed tense and polarity.
`Tense`, `Pol`	The MRG implements four basic tenses, namely the present, past, future and perfect tenses. These happen to be the most important tenses in Afrikaans, and hence the MRG provides an appropriate framework for modelling the negation of declarative sentences. Polarity is a feature of sentences, and may be positive or negative.
`Cl`	A clause for a declarative sentence is constructed from a subject noun phrase and a verb phrase, and has variable tense and polarity.
`VP`	A verb phrase contains the predicate of a declarative sentence, and may be constructed using one-place verbs (`V`; typically intransitive verbs eg. "walk"), two-place verbs (`V2`; such as transitive verbs eg. "see"), verb-phrase-complement verbs (`VV`; eg. "want") and sentence-complement verbs (`VS`; eg. "know").
`NP`	A noun phrase may be constructed from a pronoun, a proper noun or a common noun with a determiner. In the MRG, pronouns are constructed directly as noun phrases. Pronouns such as *niemand* (no-one) are n-words.
`Adv`	Adverbs modify verb phrases. Adverbs such as *nooit* (never) are n-words.
`CN`, `N`, `Det`	These categories (namely common nouns, nouns and determiners) are used to construct noun phrases. Determiners such as *geen* (no) are n-words.

Table 1. Pertinent syntactic categories used to construct declarative sentences

from the table. Note that the negation achieved by negative-valued polarity, i.e. sentential negation, is a feature of the abstract syntax tree. On the other hand, n-words are found within noun phrases (the `NP` category) in the form of pronouns and determiners that are semantically negative, such as *niemand* ('no-one') and *geen* ('no'), as well as adverbs (the `Adv` category) such as *nooit* ('never'). Consequently, their negative semantics must be recorded in the relevant lincats in the MRG, namely `NP`, `Det` and `Adv`, as shown in Figure 1. Each lincat contains a field `p` of type `TPol`, which has possible values `TPos` (positive) and `TNeg` (negative). The value of `p` is determined when the lincat is constructed: for example, when `i_NP` of type `NP` is constructed to represent the pronoun *ek* ('I'), `p` has value `TPos`, while in the case of the pronoun *niemand* ('no-one'), i.e. `no_one_NP`, `p` has value `TNeg`. Hence, this kind of negation is not a feature of the abstract syntax tree. Indeed, n-word negation may occur in sentences that have either a positive or a negative polarity value, so that sentential and n-word negation are orthogonal mechanisms for expressing negation. This gives rise to the typology illustrated in Table 2. Therefore, modelling negation in Afrikaans in the GF MRG consists essentially of accurately modelling the interaction between these two mechanisms so that the correct negative markers are always present.

Sentence polarity	Positive	Negative
Without n-words	*Ek sien die man.* 'I see the man.'	*Ek sien nie die man nie.* 'I do not see the man.'
With n-words	*Niemand sien die man nie.* 'No-one sees the man.'	*Niemand sien nie die man nie.* 'No-one does not see the man.'

Table 2. Orthogonal nature of sentential and n-word negation

3.1. Adjacent second negative markers

Both n-words and negative sentence polarity require a second negative marker *nie*, but as mentioned above, only one such marker should survive. Furthermore, sentence-

```
NP = { s : Case => Str ;
       a : Agr ;
       isPron : Bool ;
       p : TPol } ;

Det = { s : Str ;
        n : Number ;
        p : TPol } ;

Adv = { s : Str ;
        p : TPol } ;
```

```
VP = { v  : Verb ;
       n2a, n2b, subCl : Str ;
       adv : Str ;
       filled : Bool ;
       nword : Bool ;
       finNie : Bool ;
       compV : VForm => Str } ;
```

Figure 1. Lincats for `NP`, `Det`, `Adv` and `VP`

complement verbs may result in the presence of such a marker in the verb phrase if the embedded sentence[6] contains it. Due to the compositional nature of GF, this means that, on the one hand, the requirement of the second negative marker due to the presence of n-words must be kept track of using the lincats of the concrete grammar module, while on the other hand, whether the verb phrase already contains such a negative marker must similarly be kept track of. This is also shown in Figure 1: the boolean field `nword` keeps track of whether the object noun phrase contains an n-word, while `finNie` keeps track of whether a final *nie* occurs in an embedded sentence. Then, when the table representing the clause is constructed, these values inform the decision of whether to render the second negative marker in the case of both positive and negative sentence polarity. That is, there is an interaction between the requirement of a second negative marker due to the presence of n-words, the possibility that such a marker is already present in an embedded sentence, and the polarity in which the clause will eventually be rendered as a sentence. A detailed exposition of the implementation of the relevant functions requires discussion of the implementation details of various other aspects of Afrikaans grammar that are not pertinent to negation *per se*. While such a discussion is beyond the scope of this article, implementation details may be found in the source code of the grammar.[7]

However, we may illustrate this interaction by considering, for example, the abstract syntax tree in Figure 2, which is the result of parsing the sentence *Niemand sien die man nie* ('No-one sees the man'). The clause is constructed from the subject noun phrase consisting of the negative pronoun *niemand* and the verb phrase. Hence, even though the sentence has positive polarity, the second negative marker is present due to the n-word *niemand* in the subject noun phrase. On the other hand, consider the abstract syntax tree in Figure 3, which is the result of parsing the sentence *Niemand weet dat hy nie die man sien nie* ('No-one knows that he does not see the man'). The polarity of the main sentence is positive and the subject noun phrase is an n-word as in the previous example. However, in contrast to the previous example, the verb phrase contains a sentence that ends with *nie*. Note that although the embedded sentence has an SOV word order, the negative marker is still sentence final, and in the positive present tense, no other elements from

[6]For the sake of clarity in the context of GF, we refer to main sentences and embedded sentences (instead of the usual main clauses and embedded clauses) in the case where tense and polarity are fixed.

[7]The interested reader is referred to the source code of the Afrikaans MRG at `https://github.com/laurette/GFAfrikaans`.

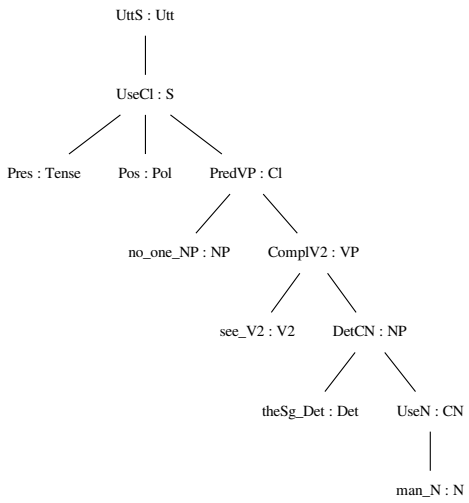


Figure 2. Abstract syntax tree: *Niemand sien die man nie*

the main clause follow it. Hence, when constructing the main clause table, no additional second negative marker is rendered for the positive present tense.

3.2. Adjacent first and second negative markers

In general, whenever sentential negation occurs, at least two negative markers must be present in the sentence. A common exception, however, is certain present tense sentences, namely those with unmodified intransitive non-phrasal verbs, e.g. *Ek loop nie* ('I do not walk'), and unmodified transitive non-phrasal verbs where the object is a pronoun or a proper noun, e.g. *Ek sien hom nie* ('I do not see him'). Both are exceptions due to the fact that the second negative marker would occur adjacent to the first. In the first case, this is simply due to the absence of any other words in the sentence which might occur between the two negative markers. In the latter case, the object noun phrase occurs in a different part of the sentence than it normally would, and consequently, in the absence of other words between the two negative markers, they would occur adjacent to each other.

This kind of exception is correctly handled by our concrete grammar module by keeping track of whether a noun phrase consists of a pronoun or proper noun and whether a verb phrase constructed from a non-phrasal transitive or intransitive verb is modified by an adverb or adverbial phrase. Figure 1 shows that the `NP` lincat contains the boolean field `isPron`, which is set to true if the noun phrase is constructed from a pronoun or proper noun, while the `VP` lincat contains the boolean field `filled`, which is set to true when the verb phrase uses a phrasal verb or when it is modified by an adverb or adverbial phrase. Furthermore, the `VP` lincat contains two possible fields for the object noun phrase string, namely `n2a` and `n2b`. When verb phrases are constructed with an object noun phrase, `isPron` determines into which field the object noun phrase string is added. Thus,

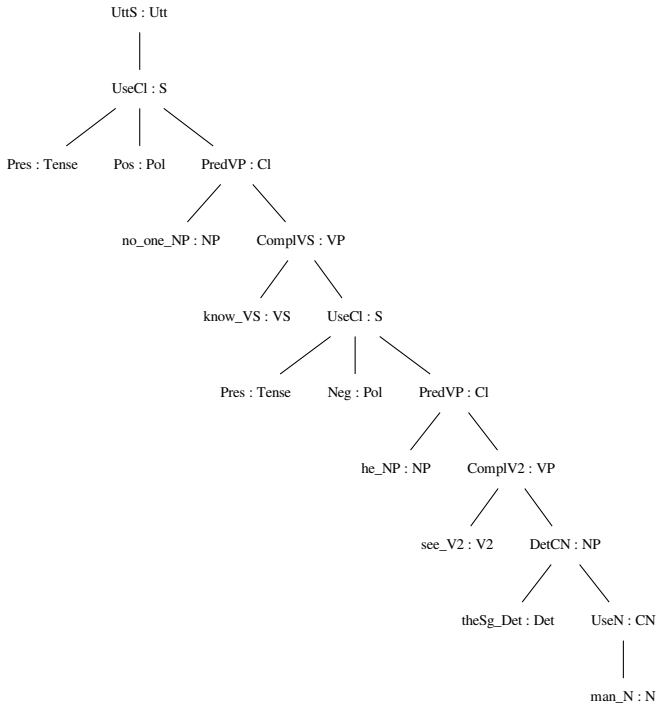


Figure 3. Abstract syntax tree: *Niemand weet dat hy nie die man sien nie (*nie)*

all the necessary information is captured in the lincats in order to determine the way in which the negative markers are rendered.[8]

3.3. Results

Our concrete grammar module correctly handles negation in declarative sentences containing one-place verbs, two-place verbs, verb-phrase-complement verbs and sentence-complement verbs in all four major tenses of Afrikaans. This is true regardless of the depth of the abstract syntax tree. For a representative selection of examples, see Table 3.

4. Conclusion and future work

The implementation of an Afrikaans concrete grammar module of the MRG as discussed in this paper covers the two mechanisms of negation, namely sentential and n-word negation, for declarative sentences in the four major tenses in Afrikaans. Consequently, it constitutes notable progress towards the implementation of an accurate Afrikaans GF re-

[8]Once again, the interested reader is referred to the source code of the Afrikaans MRG at `https://github.com/laurette/GFAfrikaans`.

Verb type	Positive sentence polarity	Negative sentence polarity
One-place verb	*Ek loop altyd.* 'I always walk.' *Ek loop nooit nie.* 'I never walk.' *Niemand loop vandag nie.* 'No-one walks today.'	*Ek loop nie altyd nie.* 'I don't always walk.' *Ek loop nie nooit nie.* 'I don't never walk.' *Niemand loop nie vandag nie.* 'No-one doesn't walk today.'
Two-place verb	*Ek sien altyd die man.* 'I always see the man.' *Ek sien nooit die man nie.* 'I never see the man.' *Niemand sien vandag die man nie.* 'No-one sees the man today.'	*Ek sien nie altyd die man nie.* 'I don't always see the man.' *Ek sien nie nooit die man nie.* 'I don't never see the man.' *Niemand sien nie vandag die man nie.* 'No-one doesn't see the man today.'
Verb-phrase-complement verb	*Ek wil vandag loop.* 'I want to walk today.' *Ek wil nooit loop nie.* 'I never want to walk.' *Niemand wil vandag loop nie.* 'No-one wants to walk today.'	*Ek wil nie vandag loop nie.* 'I don't want to walk today.' *Ek wil nie nooit loop nie.* 'I don't never want to walk.' *Niemand wil nie vandag loop nie.* 'No-one doesn't want to walk today.'
Sentence-complement verb	*Ek weet dat hy vandag loop.* 'I know that he walks today.' *Ek weet dat hy nooit loop nie.* 'I know that he never walks.' *Ek weet dat hy nie vandag loop nie.* 'I know that he doesn't walk today.' *Niemand weet dat hy vandag loop nie.* 'No-one knows that he walks today.' *Niemand het geweet dat hy nie sal kan loop nie.* 'No-one knew that he wouldn't be able to walk.'	*Ek weet nie dat hy vandag loop nie.* 'I don't know that he walks today.' *Ek weet nie dat hy nooit loop nie.* 'I don't know that he never walks.' *Ek weet nie dat hy nie vandag loop nie.* 'I don't know that he doesn't walk today.' *Niemand weet nie dat hy vandag loop nie.* 'No-one doesn't know that he walks today.' *Niemand het nie geweet dat hy nie sal kan loop nie.* 'No-one didn't know that he wouldn't be able to walk.'

Table 3. Selected sentences covered by the grammar

source grammar. This will ensure that any Afrikaans CNL, built with GF, will be able to handle Afrikaans negation correctly.

One aspect not addressed in this paper is the semantic scope of negation, which is affected by the placement of adverbs and adverbial phrases in different positions in a sentence. However, this variability does not affect the syntactic realisation of negation as discussed in this paper. Therefore, we include only one such adverbial position in our implementation. A complete implementation of adverbs and adverbial phrases forms part of future work.

Two features of the MRG abstract grammar module that are subject to negation but which are excluded from our current concrete grammar module for Afrikaans are questions and subjunctions. In these cases, additional complexities with respect to word order arise, which have an influence on the rendering of the second negative marker. Furthermore, certain constructions subject to negation are not included in the MRG abstract grammar module, such as imperatives and relative clauses. Addressing these constructions is future work.

References

[1] T. Biberauer and H. Zeijlstra. Negative concord in Afrikaans: Filling a typological gap. *Journal of Semantics*, 29(3):345–371, 2012.

[2] T. J. R. Botha, F. A. Ponelis, J. G. H. Combrinck, and F. F. Odendal. *Inleiding tot die Afrikaanse Taalkunde*. J. L. van Schaik, Pretoria, 1994. ISBN: 0868743518.

[3] W. A. M. Carstens. *Norme vir Afrikaans: Enkele Riglyne by die Gebruik van Afrikaans*. J. L. van Schaik, Pretoria, 1994. ISBN: 0627019536.

[4] L. G. de Stadler. *Afrikaanse Semantiek*. Johannesburg, 1989. ISBN: 868122077.

[5] K. Huddlestone. *Negative Indefinites in Afrikaans*. PhD thesis, Netherlands National Graduate School of Linguistics, Janskerkhof 13, 3512 BL Utrecht, The Netherlands, 2010. ISBN: 9789460930331.

[6] T. Kuhn. A survey and classification of controlled natural languages. *Computational Linguistics*, 40(1):121–170, 2014.

[7] F. A. Ponelis. *Afrikaanse Sintaksis*. J. L. van Schaik, Pretoria, 1979. ISBN: 0627010350.

[8] A. Ranta. *Grammatical Framework: Programming with Multilingual Grammars*. CSLI Publications, Stanford, 2011. ISBN-10: 1-57586-626-9 (Paper), 1-57586-627-7 (Cloth).

[9] H. Safwat and B. Davis. A brief state of the art of cnls for ontology authoring. In B. Davis, K. Kaljurand, and T. Kuhn, editors, *Controlled Natural Language*, pages 190–200, Cham, 2014. Springer International Publishing. ISBN: 9783319102238.

[10] H. Safwat and B. Davis. Cnls for the semantic web: a state of the art. *Language Resources and Evaluation*, 51(1):191–220, Mar 2017. ISSN: 15740218.

[11] H. Zeijlstra. Negation in natural language: On the form and meaning of negative elements. *Language and Linguistics Compass*, 1(5):498–518, 2007.

This page intentionally left blank

B. Davis et al. (Eds.)

Subject Index

This page intentionally left blank

Controlled Natural Language
B. Davis et al. (Eds.)

Author Index

This page intentionally left blank