A Cloud-Based Editor for Multilingual Grammars

Thomas Hallgren **Ramona Enache** **Aarne Ranta**
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
`hallgren@chalmers.se` `ramona.enache@chalmers.se` `aarne@chalmers.se`

Abstract

Writing deep linguistic grammars has been considered a highly specialized skill, requiring the use of tools with steep learning curves and complex installation procedures. As the use of statistical methods has increased, new generations of computational linguists are getting less and less prepared for grammar writing tasks. In an aim to provide a better learning experience for grammar writers, we present a grammar engineering tool that resides in the cloud. It has been used in several tutorial courses and self-studies, and it allows absolute beginners to write their first grammars and parse examples in 10 minutes. The cloud-based grammar engineering tool is built on top of GF (Grammatical Framework), a grammar formalism that has an explicit tecto/phenogrammar distinction, is based on ideas from type theory and functional programming and comes equipped with a grammar library supporting 30 languages.

1 Introduction

Writing deep linguistic grammars has been considered a highly specialized skill. As the use of statistical methods has increased, new generations of computational linguists are getting less and less prepared for grammar writing tasks. A part of the problem is the steep learning curve in tools: systems like LKB (Copestake, 2002) and XLE (Xerox Linguistic Environment) are designed for professional linguists. Getting started with their use requires substantial training, and installing the tools requires large and unfamiliar software packages, in addition to a firm knowledge of operating system command-line tools.

GF (Ranta, 2004) is a more recent grammar formalism, born so to say in the middle of the statistical era. GF shares the ambition of the "classical" formalisms to enable deep linguistic descriptions, which it wants to support with some new ideas: type theory, functional programming, and an explicit tecto/phenogrammar distinction. However, GF was also meant to be a formalism for "ordinary" programmers without linguistic training. Thus the majority of the currently 30 languages included in the GF Resource Grammar Library (Ranta, 2009a) are in fact written by students and scholars in computer science, who find the GF style of programming familiar from other contexts, in particular compiler construction (Appel, 1998).

However, the GF approach has a "nerdy" flavour to it, in particular requiring coping with command line tools, text editors, and Haskell libraries. Some programmers are helped by the Eclipse plug-in (Camilleri, 2012), but installing both GF and Eclipse on a personal computer can be a daunting task for many.

The present paper describes an attempt to eliminate all trouble with software installation from linguistic grammar writing. We describe a grammar engineering tool that resides in the cloud and can be used in ordinary web browsers. The tool supports writing grammars in the cloud, compiling them to executable parsers and translation systems, and finally running and testing them in the cloud. Thus an entire grammar project can be written and used without installing any specific software. The project can also be published and shared, so that many users can work on the same grammars (although not simultaneously yet in the current version).

The cloud-based GF editor has been used on several tutorial courses and self-studies. It enables absolute beginners to write their first grammar and parse examples in 10 minutes. It scales up to most of the grammars described in the GF book (Ranta, 2011), although it has some limitations, in partic-

```
abstract Foods = {

flags startcat = Comment;

cat Comment; Kind; Item; Quality;

fun Pred : Item -> Quality -> Comment;

    This, That : Kind -> Item;

    Bread, Fish, Wine : Kind;

    Very : Quality -> Quality;
    Bad, Good : Quality;
    Cold, Warm : Quality;
}
```

Figure 1: Abstract syntax `Food.gf`.

```
concrete FoodsGer of Foods =

open SyntaxGer, LexiconGer in {

lincat Comment = Utt;
       Kind = CN;
       Item = NP;
       Quality = AP;

lin
  Pred item qual = mkUtt(mkCl item qual);
  This kind = mkNP this_Det kind;
  That kind = mkNP that_Det kind;
  Bread = mkCN bread_N;
  Fish = mkCN fish_N;
  Wine = mkCN wine_N;
  Very quality = mkAP very_AdA quality;
  Bad = mkAP bad_A;
  Good = mkAP good_A;
  Cold = mkAP cold_A;
  Warm = mkAP warm_A;
}
```

Figure 2: Concrete syntax `FoodGer.gf`.

ular a simplified module system, which makes it unpractical for larger tasks. But students who have got the first experience of grammar writing without the overhead of installation troubles are more likely to proceed to the full-scale systems when they feel the need for it.

1.1 Grammar development with GF

Writing a multilingual grammar in GF consists of writing (1) an abstract syntax that captures the meanings of interest and (2) a number of concrete syntaxes that map the meanings of the abstract syntax to concrete representations in the natural (or formal) languages relevant to the application.

Traditionally, GF grammars are created in a text editor. For example, to create a grammar for comments about food, the grammar author would create an abstract syntax file `Food.gf` (Figure 1) and perhaps a concrete syntax file `FoodsGer.gf` for German (Figure 2). But text editors know very little (if anything) about the syntax of GF grammars, and thus provide little guidance for novice GF users. Instead, to be able to write grammars like the one above, users have to rely on separately available documentation (manuals, tutorials and books) for guidance.

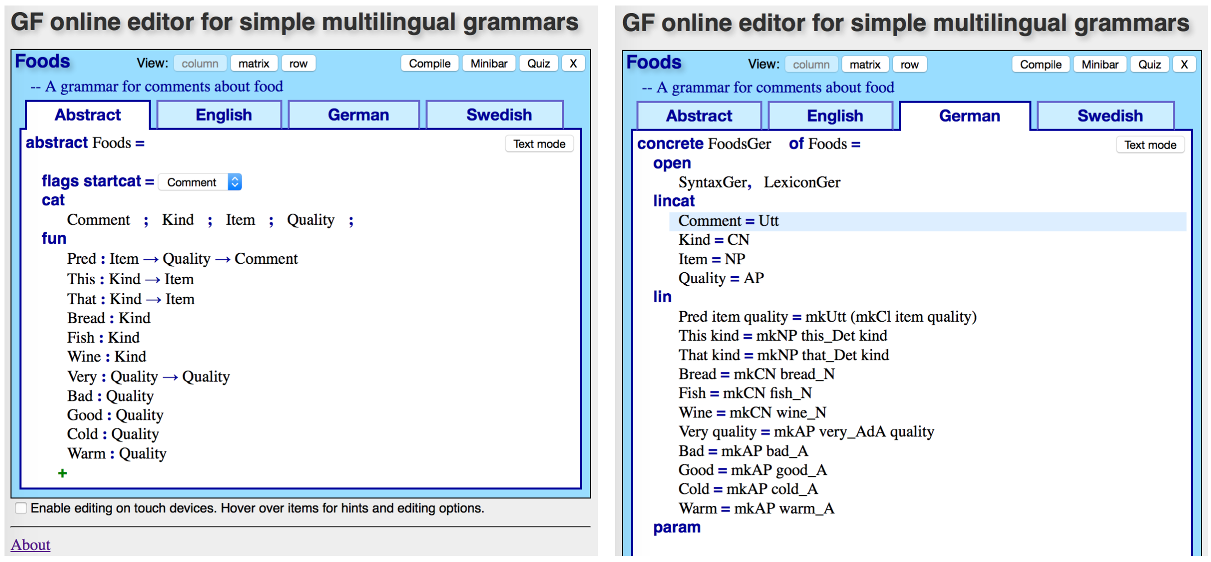
For testing, grammars can be loaded in the GF shell. For example, by loading `FoodsGer.gf`, the user can check that the German sentence *dieses Brot ist sehr gut* can be parsed and represented as `Pred (This Bread) (Very Good)` in the abstract syntax.

GF grammar source files can also be compiled to Portable Grammar Format files (Angelov et al., 2010) that can be used with the GF run-time library to include natural language processing in applications. The key operations provided by the run-time library are parsing, generation, and (by combining the former two) translation. The GF run-time library is also available as a web service, which can be used to create interactive natural language web applications (Bringert et al., 2009; Ranta et al., 2010). Examples can be seen in Figures 6 and 7.

While applications based on GF grammars could be made available online by using the GF web service, until now the grammars themselves had to be created with offline tools that the grammar developer had to download and install on his/her own computer. With the cloud-based editor presented here, the grammars can also be created online.

1.2 Outline

In section 2 we describe the cloud-based grammar editor introduced above. In section 3 we describe a new technique for *example-based grammar writing* that we are adding support for in the cloud-based editor. This makes it possible for a user with minimal knowledge of GF grammar construction to add new languages to a multilingual grammar by translating automatically generated examples in one of the existing languages to the new language. In section 4 we present some implementation details and in sections 5 and 6 we describe related and future work.


GF online editor for simple multilingual grammars
Foods
View: column matrix row
Compile Minibar Quiz X
Abstract English German Swedish
Text mode
Enable editing on touch devices. Hover over items for hints and editing options.
About

```
-- A grammar for comments about food
abstract Foods =

  flags startcat = Comment
  cat
     Comment ; Kind ; Item ; Quality ;
  fun
     Pred : Item → Quality → Comment
     This : Kind → Item
     That : Kind → Item
     Bread : Kind
     Fish : Kind
     Wine : Kind
     Very : Quality → Quality
     Bad : Quality
     Good : Quality
     Cold : Quality
     Warm : Quality
   +
```

```
-- A grammar for comments about food
concrete FoodsGer   of Foods =
  open
     SyntaxGer, LexiconGer
  lincat
     Comment = Utt
     Kind = CN
     Item = NP
     Quality = AP
  lin
     Pred item quality = mkUtt (mkCl item quality)
     This kind = mkNP this_Det kind
     That kind = mkNP that_Det kind
     Bread = mkCN bread_N
     Fish = mkCN fish_N
     Wine = mkCN wine_N
     Very quality = mkAP very_AdA quality
     Bad = mkAP bad_A
     Good = mkAP good_A
     Cold = mkAP cold_A
     Warm = mkAP warm_A
  param
```

Figure 3: GF online editor for simple multilingual grammars

2 The GF online grammar editor

As the name suggests, the *GF online editor for simple multilingual grammars* is available online (Hallgren, 2013), so all that is needed to use the editor is a device with a reasonably modern web browser. Even smartphones and tablets can be used. To help novice grammar authors, the editor provides some guidance, e.g. by showing a skeleton grammar file and hinting how the parts should be filled in. When a new part is added to the grammar, it is immediately checked for errors.

Figure 3 illustrates what the editor looks like. Editing operations are accessed by clicking on editing symbols embedded in the grammar display: **+**, **x** and **%** to add, delete and edit items. These are revealed when hovering over items. On touch devices, hovering is in some cases simulated by tapping, but there is also a button to ”Enable editing on touch devices” that reveals all editing symbols.

The current version of the editor supports a small but useful subset of the GF grammar notation. Grammars consist of one module for the abstract syntax, and a number of modules for concrete syntaxes. Proper error checking is done on the fly for abstract syntax, but not (yet) for concrete syntax.

Grammars can import modules from the *Resource Grammar Library* (Ranta, 2009a), freeing the grammar author from dealing directly with the linguistic complexities of natural languages, such as inflection and agreement.

2.1 Abstract syntax

The definition of an abstract syntax consists of

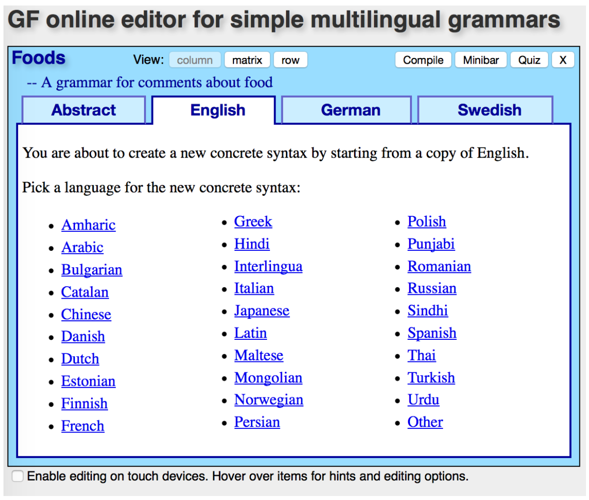
- a list of inherited abstract syntaxes,
- a list of *category names*, C_1 **;** ... **;** C_n,
- a list of *functions*, $Fun_i : C_{i_1} \rightarrow ... \rightarrow C_{i_n}$
- and the designation of a *start category*.

This is somewhat restricted compared to the full GF grammar formalism, e.g. dependent types are not supported.

Available editing operations include:

- Inherited abstract syntaxes can be added and removed.
- Categories can be added, removed and renamed. When renaming a category, occurrences of it in function types will be updated accordingly.
- Functions can be added, removed, renamed and edited. Concrete syntaxes are updated to reflect changes.
- Functions can be reordered using drag-and-drop.

The editor checks the abstract syntax for correctness as it is entered. Syntactically incorrect function definitions are rejected. Semantic errors such as duplicated definitions or references to undefined categories, are highlighted. This is enough to ensure that a grammar that is accepted by the editor will also be accepted by the GF grammar compiler.



-- A grammar for comments about food

You are about to create a new concrete syntax by starting from a copy of English.

Pick a language for the new concrete syntax:

- Amharic
- Arabic
- Bulgarian
- Catalan
- Chinese
- Danish
- Dutch
- Estonian
- Finnish
- French
- Greek
- Hindi
- Interlingua
- Italian
- Japanese
- Latin
- Maltese
- Mongolian
- Norwegian
- Persian
- Polish
- Punjabi
- Romanian
- Russian
- Sindhi
- Spanish
- Thai
- Turkish
- Urdu
- Other

Enable editing on touch devices. Hover over items for hints and editing options.

Figure 4: Adding a new concrete syntax

2.2 Concrete syntax

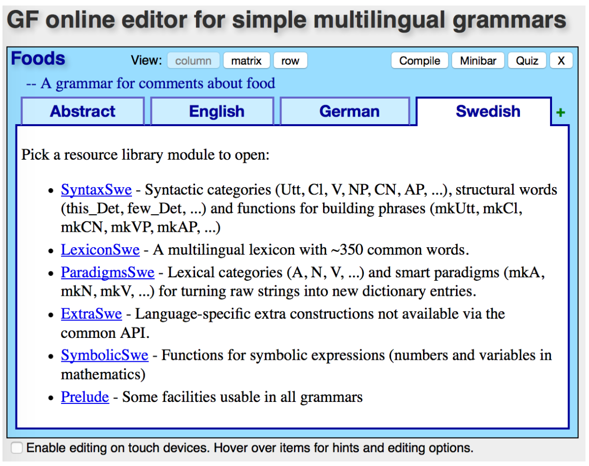
When adding a new concrete syntax to a grammar, the editor shows a list of supported natural languages and the user just picks one. See Figure 4. The name of the new module is filled in automatically based on naming conventions, e.g. `FoodsEng` if abstract syntax is called `Foods` and we are adding a translation to English. The body of the new concrete syntax can be created by copying and modifying an existing concrete syntax, or by starting with a skeleton based on the abstract syntax.

The key components of a concrete syntax are *linearization types* for the categories and *linearizations* for the functions in the abstract syntax. The editor automatically provides correct left-hand sides for these, since they are determined by the abstract syntax, while the right-hand sides can be edited freely.

The editor allows a concrete syntax to open some of the relevant Resource Grammar Library modules. A list of suitable library modules is shown, e.g., `SyntaxEng` and `LexiconEng` in a concrete syntax for English, so the user does not need to know their names by heart. See Figure 5.

The editor also supports definitions of *parameter types* and auxiliary *operations*, but usually it is enough to rely on the types and operations provided by the Resource Grammar Library.

The editor checks all user editable parts of the concrete syntax for syntactic correctness as they are entered. Duplicated definitions of parameter types or operations are highlighted. Checks for other semantic errors are delayed until the grammar is compiled.



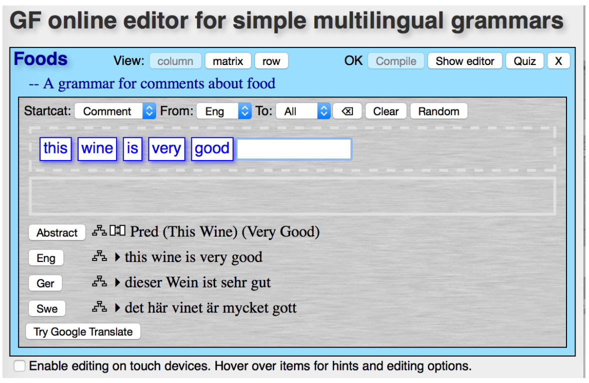
-- A grammar for comments about food

Pick a resource library module to open:

- SyntaxSwe - Syntactic categories (Utt, Cl, V, NP, CN, AP, ...), structural words (this_Det, few_Det, ...) and functions for building phrases (mkUtt, mkCl, mkCN, mkVP, mkAP, ...)
- LexiconSwe - A multilingual lexicon with ~350 common words.
- ParadigmsSwe - Lexical categories (A, N, V, ...) and smart paradigms (mkA, mkN, mkV, ...) for turning raw strings into new dictionary entries.
- ExtraSwe - Language-specific extra constructions not available via the common API.
- SymbolicSwe - Functions for symbolic expressions (numbers and variables in mathematics)
- Prelude - Some facilities usable in all grammars

Enable editing on touch devices. Hover over items for hints and editing options.

Figure 5: Opening modules from the Resource Grammar Library


GF online editor for simple multilingual grammars
Foods
View: column matrix row
OK Compile Show editor Quiz X
Startcat: Comment From: Eng To: All
Clear Random
this wine is very good
Try Google Translate

-- A grammar for comments about food

Abstract Pred (This Wine) (Very Good)
Eng this wine is very good
Ger dieser Wein ist sehr gut
Swe det här vinet är mycket gott

Enable editing on touch devices. Hover over items for hints and editing options.

Figure 6: Testing grammars in the Minibar

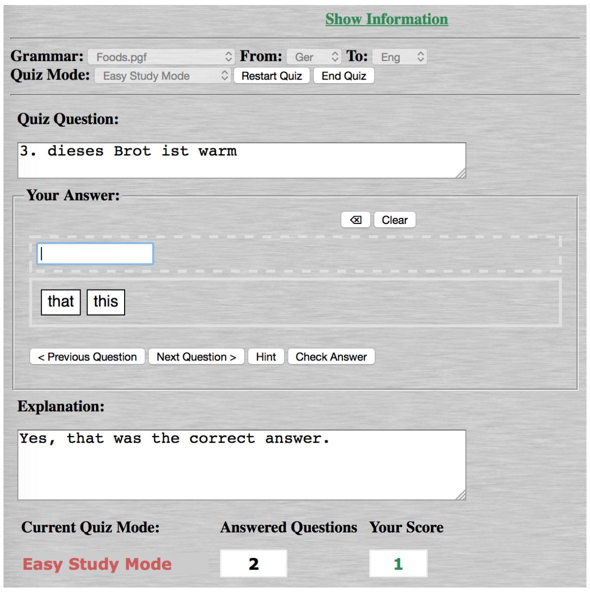
2.3 Compiling and testing grammars

When pressing the *Compile* button, the grammar is uploaded to the server and compiled with GF, and any errors not detected by the editor will be reported. Error-free grammars can be tested by clicking on the the *Minibar* button, which is a web-based translation tool, and the *Quiz* button, which is a web-based language training tool (Abolahrar, 2011). See Figures 6 and 7.

2.4 Grammars in the cloud

While grammars created in the editor are stored locally in the device by the browser, it is also possible to store grammars in the cloud. Each device is initially assigned to its own unique cloud and has its own set of grammars, but it is also possible to merge clouds and share a common set of grammars between multiple devices.

Users can also choose to “publish” a grammar. A copy of the grammar is then added to a list of



Quiz Question:

```
3. dieses Brot ist warm
```

Your Answer:

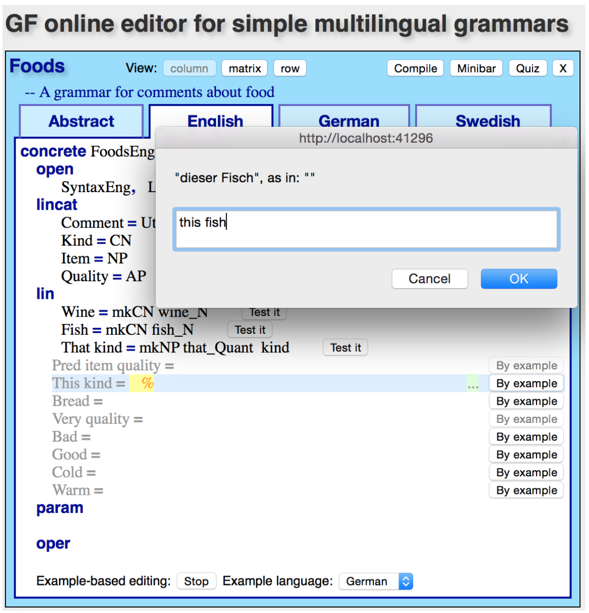
that this

Explanation:

```
Yes, that was the correct answer.
```

Current Quiz Mode:	Answered Questions	Your Score
Easy Study Mode	2	1

Figure 7: Testing grammars in the Translation Quiz



```
-- A grammar for comments about food
concrete FoodsEng
  open
      SyntaxEng,  L
  lincat
      Comment = Ut
      Kind = CN
      Item = NP
      Quality = AP
  lin
      Wine = mkCN wine_N
      Fish = mkCN fish_N
      That kind = mkNP that_Quant  kind
    Pred item quality =
    This kind = %
    Bread =
    Very quality =
    Bad =
    Good =
    Cold =
    Warm =
  param

  oper
```

"dieser Fisch", as in: ""

this fish

Figure 8: Example-based grammar construction

grammars visible to all users of the cloud-based grammar editor.

3 Example-based grammar writing

The example-based grammar writing mechanism is aimed at helping users who build concrete grammars using the resource grammar for the given language. The resource library provides over 300 functions for building grammatical constructs such as predication, complementation, etc (Ranta, 2009b). Using the resource library is advantageous on one hand, because it alleviates the difficulty of reimplementing language-specific features every time when writing a grammar for the language, but on the other hand it assumes a working knowledge of the resource library, which could lead to a larger overall effort. We aim at freeing users from this burden by making it possible for them to write function linearizations by giving example of their usage. In the current scenario, we assume that a large lexicon covering the words that could be used in the grammar is available already. We will use the resource grammar enhanced with the larger dictionary for parsing the examples from the user in order to infer the right linearization form.

Since the functions from the grammar could take arguments, in order to give an example for the usage of a certain function, we need to have one example for each of its arguments in order to get more precise information about the behavior of the function. For this reason, only the function for which all arguments can be found among the already implemented functions, are highlighted as available for the example-based method.

In order to clarify the usage of a certain function, its context is made explicit by embedding the function into a tree returning the start category, like in Figure 8 where ”this fish” is used to make phrases like ”this fish is delicious”. Since certain parts of the phrase are not relevant for the task, they are underspecified by using ”?” instead. In case that the grammar returns more than one parse tree, the results are ranked in the descending order of their probability (defined in the corresponding resource grammar or defined by the user), and the first tree from which the arguments can be abstracted is chosen as the linearization tree.

The technique has been used as an experimental way for developing a tourist phrasebook grammar in GF for 4 languages (Ranta et al., 2011), but no tool support was available at that time. The positive results obtained were a strong motivation to make the method available to end users as part of a GF grammar writing system.

The example-based grammar writing system is still work in progress and the basic prototype currently available will be further developed and improved. It is possible to use it already for 5 languages where a large dictionary is available in GF (English, Swedish, Finnish, Bulgarian, French).

4 Implementation

The implementation of the cloud-based editor consists of a client-side part and a server-side part.

The client-side part is written directly in JavaScript. In total, roughly 6350 lines of JavaScript (180KB) is loaded by the browser when opening the editor. This divides roughly into 2800 lines written specifically for the editor, 2600 lines of code for other components (the minibar) and 750 lines of supporting library code.

All server-side code is written in Haskell. This includes the GF grammar compiler, the PGF runtime library and the PGF web service (Bringert et al., 2009). We created a new GF cloud service API (Hallgren, 2014) to support the editor. It includes functionality for grammar syntax checking, grammar upload, grammar compilation and access to the GF shell. It's implemented as 500 lines of Haskell code.

To support example-based grammar writing, we added 200 lines of client-side JavaScript code and 680 lines of Haskell code in the server.

5 Related work

GF is a grammar formalism comparable in expressive power to HPSG (Pollard and Sag, 1994) and LFG (Bresnan, 1982), but different due to the distinction between the abstract and concrete dimension of a grammar, along with the possibility to share the abstract syntax which makes translation between any pair of languages possible. In the same way, the GF resource library could be compared to two other multilingual resources based on the above-mentioned formalisms: LinGO Grammar Matrix (Bender et al., 2002) for HPSG and ParGram (Butt et al., 2002) for LFG.

Since the task of developing a multilingual grammar within such a grammar formalism poses specific challenges, each system comes equipped with its own IDE/editor that aids the grammar development process. LinGO Grammar Matrix has a grammar-customization system (Bender et al., 2010) and ParGram has XLFG, a customized IDE (Clément, 2009). While XLFG allows storing and editing grammars, it is only available for English and French. The customization system from LinGO Grammar Matrix allows specifying linguistic features of certain grammar constructions in the shape of a questionnaire and it is mainly used for developing new language resources for the Matrix library. The further use of the resources is supported by a parser, sentence generator and facilities for profiling and regression testing (Oepen and Flickinger, 1998).

In addition to the cloud-based IDE, GF also has a desktop IDE, implemented as an Eclipse plugin (Camilleri, 2012).

6 Future work

The GF grammar editor described here already supports a useful subset of the GF grammar notation. Some of the guidance and error checking is done in the editor on the client side (resulting in well integrated user friendly error indications), some is delegated to the server (resulting in less user friendly error messages). We do not expect to create a full implementation of GF that runs in the web browser.

If we want to support the full-fledged GF grammar formalism, the easy way out would be to duplicate a typical desktop development environment in the browser, i.e. create an environment with a text editor and command line tools, or perhaps a more integrated Eclipse-like environment. But we would prefer to create something that is more interactive and provides more guidance for novice users, so we are thinking of an interactive development environment in the style of proof assistants based on type theory, such as Alfa (Hallgren, 2004) and Agda (Norell, 2007). However, being a batch compiler, GF does currently not provide an API that makes it easy to create this kind of development environment. If a GF server with an appropriate API becomes available, it should be possible to extend the editor to support a larger fragment of GF, to provide more user guidance, more user friendly error reporting, and in general make more of the functionality in the existing GF tools accessible directly from the online editor.

More work is needed on the grammar cloud storage service. In particular, it is currently not suitable for multiple users developing a grammar in collaboration. This could be done by interfacing with a online collaboration tool (like GitHub), or by allowing concurrent access to shared grammars and propagating edits to all users in real-time (like in Google Docs).

Combining the cloud-based grammar editor with other cloud-based tools opens up possibilities for new applications, such as a tourist phrasebook that can be extended by the user with a new topic of interest, or a language training tool (like the one in Figure 7) that instructors or students can customize for training or testing a particular vocabulary or particular grammatical forms. It should be possible to make this kind of customizations in minutes, without installing any software.

Future work on the example-based method includes combining it with traditional grammar writing and the possibility to develop more languages in parallel and use one as an example for the other. Moreover, since currently the method works for the case when the linearization type is a category from the resource library (noun phrase, sentence, etc.), one could also extend the algorithm in order to handle record types comprising more such syntactic categories. Last but not least, we aim at covering languages for which large dictionaries are not available by making the method robust to unknown words that could be later implemented by the user.

References

Elnaz Abolahrar. 2011. Multilingual Grammar-Based Language Training: Computational Methods and Tools. Master's thesis, Chalmers University of Technology.

Krasimir Angelov, Björn Bringert, and Aarne Ranta. 2010. PGF: A Portable Run-time Format for Type-theoretical Grammars. *Journal of Logic, Language and Information*, 19:201–228. 10.1007/s10849-009-9112-y.

Andrew W. Appel. 1998. *Modern Compiler Implementation in ML*. Cambridge University Press.

Emily M. Bender, Dan Flickinger, and Stephan Oepen. 2002. The grammar matrix: an open-source starter-kit for the rapid development of cross-linguistically consistent broad-coverage precision grammars. In *COLING-02 on Grammar engineering and evaluation*, pages 1–7, Morristown, NJ, USA. Association for Computational Linguistics.

Emily M. Bender, Scott Drellishak, Antske Fokkens, Laurie Poulson, and Safiyyah Saleem. 2010. Grammar Customization. *Research on Language & Computation*, 8(1):23–72. 10.1007/s11168-010-9070-1.

Joan Bresnan. 1982. *The Mental Representation of Grammatical Relations*. MIT Press.

Björn Bringert, Krasimir Angelov, and Aarne Ranta. 2009. Grammatical framework web service. In *Proceedings of the Demonstrations Session at EACL 2009*, pages 9–12, Athens, Greece, April. Association for Computational Linguistics.

Miriam Butt, Helge Dyvik, Tracy Holloway King, Hiroshi Masuichi, and Christian Rohrer. 2002. The Parallel Grammar project. In *COLING-02 on Grammar engineering and evaluation*, pages 1–7, Morristown, NJ, USA. Association for Computational Linguistics.

John J. Camilleri. 2012. An IDE for the Grammatical Framework. Gothenburg, Sweden, June.

Lionel Clément. 2009. XLFG5 Documentation. `https://signes.bordeaux.inria.fr/xlfg5/doc/en/`, October.

Ann Copestake. 2002. *Implementing typed feature structure grammars*, volume 110. CSLI publications Stanford.

Thomas Hallgren. 2004. Home Page of the Proof Editor Alfa. `www.cse.chalmers.se/~hallgren/Alfa/`.

Thomas Hallgren. 2013. GF online editor for simple multilingual grammars. `http://cloud.grammaticalframework.org/gfse/`.

Thomas Hallgren. 2014. GF Cloud Service API. `http://cloud.grammaticalframework.org/gf-cloud-api.html`.

Ulf Norell. 2007. *Towards a practical programming language based on dependent type theory*. Ph.D. thesis, Department of Computer Science and Engineering, Chalmers University of Technology, SE-412 96 Göteborg, Sweden, September.

Stephan Oepen and Daniel P. Flickinger. 1998. Towards Systematic Grammar Profiling Test Suite Technology Ten Years After. *Special Issue on Evaluation), 411*, 12:411–436.

Carl Pollard and Ivan A. Sag. 1994. *Head-Driven Phrase Structure Grammar*. University of Chicago Press.

Aarne Ranta, Krasimir Angelov, and Thomas Hallgren. 2010. Tools for multilingual grammar-based translation on the web. In *Proceedings of the ACL 2010 System Demonstrations*, pages 66–71, Uppsala, Sweden, July. Association for Computational Linguistics.

Aarne Ranta, Ramona Enache, and Grégoire Détrez. 2011. Controlled Language for Everyday Use: the MOLTO Phrasebook. *Proceeding of the 2nd Workshop on Controlled Natural Languages (CNL 2010)*.

Aarne Ranta. 2004. Grammatical Framework: A Type-Theoretical Grammar Formalism. *The Journal of Functional Programming*, 14(2):145–189.

Aarne Ranta. 2009a. The GF resource grammar library. *Linguistic Issues in Language Technology*, 2(2).

Aarne Ranta. 2009b. Grammars as Software Libraries. In Yves Bertot, Gérard Huet, Jean-Jacques. Lévy, and Gordon Plotkin, editors, *From Semantics to Computer Science. Essays in Honour of*

Gilles Kahn, pages 281–308. Cambridge University Press. `http://www.cse.chalmers.se/~aarne/articles/libraries-kahn.pdf`.

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