Controlled Language for Everyday Use: the MOLTO Phrasebook

Aarne Ranta, Ramona Enache, Grégoire Détrez

Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract. Controlled languages are usually targeted for technical domains and designed to be unambiguous. This paper presents a controlled language whose domain is touristic phrases, aimed to be usable by anyone without prior training. Despite its informal nature, the language of phrases has a firm notion of semantics, defining the correctness of translations. However, this semantics is formulated in terms of context and situation rather than by logical formulas. Moreover, the language is often ambiguous, and the translation may depend on resolving the ambiguity. This paper shows how to formalize a semantics for tourist phrases and implement it in 15 languages, how to deal with the ambiguities, and how to make the system available for layman users on the web and on mobile phones. While a useful application as such, the Phrasebook also paves the way for an extended notion of controlled language, and the techniques are aimed to be general enough to support many such extensions.

Keywords: controlled language, Grammatical Framework, multilingual grammar, tourist phrasebook, mobile translation application

1 Introduction

Controlled languages are typically designed for use on technical domains. Their users are experts such as aircraft engineers [1], medical doctors [2], and topographers [3]. The language is typically a natural-language image of a formal system, such as predicate logic in [4] or OWL (Web Ontology Language) [5] in [6]. The purpose of these controlled languages is to support knowledge representation, reasoning, and mechanical checking of correctness; the main point of using a natural language fragment rather than a formalism is to have a notation that is readable without special training. When there is no underlying formalism, as in [1], the purpose is to eliminate the ambiguity, vagueness, and unclarity of uncontrolled natural language.

However, the notion of controlled language can be given a wider interpretation: it can be just *any* fragment of natural language specified with a formal set of rules. Actually it can be seen as the technological counterpart of Wittgenstein's philosophical notion of **language games** [7], which are systems of rules specifying how language is used for performing different tasks. In the tradition represented by Wittgenstein and his followers, language games are the very essence of language: they should *not* be seen as mere fragments of an underlying total system, but as the building blocks that actually constitute the thing called language. Very little can be rigorously said about natural

language as a whole, whereas these limited fragments are units that (at least in many cases) permit a formal description and—consequently—a computer implementation.

When we start looking at language from the language game point of view, we suddenly begin to see "formal systems" everywhere. One of the most basic ones is the social game of greetings and politeness phrases. For instance, when I ask for something, I attach the word *please*. When you hand it over to me, you say *here we are*, to which I should say *thank you*, and you can conclude by replying *you're welcome*. These four phrases get their precise meanings in the context of this game. Actually, each of them could be used in some other context and mean something different. This is seen clearly when we look at their translations. Here is a simple dialogue in three languages:

English	**Swedish**	**German**
A beer please.	*En öl tack.*	*Ein Bier bitte.*
Here we are.	*Var så god.*	*Bitte.*
Thank you.	*Tack.*	*Danke.*
You're welcome.	*Var så god.*	*Bitte.*

English makes most distinctions here, by using a different phrase for each of the four moves of the game. Swedish uses *tack* for both asking and thanking. German uses different phrases for these two, but the word *bitte* (literally, "I request") is otherwise used for everything! Nevertheless, there's no problem in translating Swedish and German phrases to English, *as soon as we know what move they express in the language game*.

Of course, it is just a coincidence that English has unambiguous phrases for all language game moves here. English, and all other languages, are full of ambiguities, if we look only at the syntax without context. This is not just a feature of everyday language but even of mathematics, as convincingly shown in [8]. But the ambiguities are almost always easily resolved by looking at the context of use.

An ambiguity specific to English is generated from the word *you*. It has two translations in Swedish (the familiar singular *du*, the plural or formal singular *ni*), three in German (the familiar singular *du*, the familiar plural *ihr*, and the formal *Sie*), and up to eight in languages like Spanish (singular/plural, familiar/formal, masculine/feminine). For instance, the English phrase *are you German* has eight translations in Spanish. The translation is determined by the context of use—basically, by the addressee.

The "language game" of social phrases is not only a philosophical experiment, but also a lucrative business. Phrasebooks like Berlitz and Lonely Planet are still sold in millions of copies, although electronic phrasebooks running on mobile phones are taking more and more of the market share. A typical electronic phrasebook is just a digital version of the printed book: a collection of phrases that can be looked up either by typing search strings or by browsing in hierarchic menus. A particularly smart example is the Chinese iPhone application YoChina[1], which puts each phrase into a context and also shows a set of responses from which the interlocutor can choose.

Even the most sophisticated commercial phrasebooks are still just collections of **canned phrases**: fixed strings, which, even though there might be thousands of them, don't cover all possible combinations of the concepts involved. A different approach can

[1] http://www.yocoy.com

be taken by using **machine translation**; thus Google Translate [2] is available as a mobile phone application that actually translates each individual phrase separately. While this is the most natural and powerful approach to the problem, it still has open issues. The first issue is quality: even though Google Translate often does a good job, it can just as easily produce something totally wrong, and this can lead to embarrassing situations if used in a social context of communication (mostly resolved by a good laugh, of course). In particular, Google Translate is based on a generic, statistical language model which cannot make distinctions like the ones needed for the different uses of German *bitte*. The second issue with Google Translate as used by a traveller far away from home is the cost of mobile data transfer. It may just be too expensive to use the service.

In this paper, we will introduce a controlled language translator approach to tourist phrasebooks. We will show a **formal semantic model**, which unambiguously specifies an infinite class of phrases. Then we will show how the semantic model is translated to phrases in 15 natural languages. The translations are reversible, which means that the phrasebook can both generate natural language from the formal semantics and interpret it in the formal semantics. The combination of generation and interpretation is translation; our phrasebook is able to translate equally well with all of the 14*15=210 language pairs. The translator runs as an off-line application on Android mobile phones and can be downloaded free of charge from Android Market[3]. The phrasebook is also available as a web application[4].

Figure 1 (left) shows the web interface to the phrasebook. The user has constructed the English sentence *how far is the Russian restaurant*, which the system has translated to the other 14 languages. The construction is carried out by a **predictive parser** [9], which predicts the set of possible next words at each point. The input can be made by typing text (in the white slot on the right) or by clicking at one of the rectangles showing a word. The possible continuations here are *?* (to terminate the phrase), *by* (as in *by tram*), and *from* (as in *from the hotel*). As soon as there is enough input to translate, the translations are shown. Figure 1 (right) shows the Android mobile application. For size reasons, the application shows only one target language at a time. As a bonus, it has speech synthesizer output for some languages.

Touristic phrases are a rich domain, and one could easily spend a lifetime on building, refining, and extending an electronic phrasebook. What we want to show in this paper is a technology that gives maximal support to this work. The technology is based on GF (Grammatical Framework, [10]), which is a grammar formalism designed for supporting multilingual grammars of controlled languages. In addition to a programming language, GF provides RGL (Resource Grammar Library, [11]), which encapsulates the low-level linguistic knowledge of morphology and syntax that is needed when building high-quality translation systems.

In addition to the grammar engineering tools, GF has a set of tools supporting run-time applications. These include libraries for web servers and clients [12] and, most importantly for the current purpose, a Java-based run-time system for Android phones.

---

[2] http://translate.google.com

[3] https://market.android.com/details?id=org.grammaticalframework.android.apps.phrasedroid

[4] http://www.grammaticalframework.org/demos/phrasebook/

Fig. 1. The MOLTO Phrasebook as a web application (left) and as an Android mobile application (right).

Thus, at the same time as the phrasebook is a practical help for tourists, it is a showcase for a powerful general technology. This technology is being developed in the European MOLTO project [5]). In addition to using GF, MOLTO explores ways to use statistical translation models to help the construction and improve the coverage of grammar-based systems. The MOLTO Phrasebook is a first experiment of this: some of the languages involved were implemented by programmers not knowing the language at all, but using a statistical model to bootstrap the grammar and a native-speaker informant to evaluate it. This was developed into a general method that will be usable for any further project of building multilingual controlled language systems.

The structure of the paper is as follows: Section 2 specifies the coverage of the MOLTO phrasebook by giving an overview of its semantic model. Section 3 shows examples of how the different languages are implemented by using GF and RGL. Section 4 shows how ambiguities are displayed to users by means of disambiguation grammars. Section 5 introduces the method of example-based grammar writing using statistical models and human informants. Section 6 explains the Java run-time system of GF and the architecture of the mobile Android application. Section 7 presents some results from evaluation, and Section 8 concludes.

2 The Semantic Model

In GF, a semantic model is called an **abstract syntax**. It is defined by giving a set of **categories** (keyword `cat`) and a set of **functions** (keyword `fun`), which together define the notion of **well-typed trees**. For instance, the phrases in the beer-ordering dialogue above can be given the following abstract syntax:

```
cat
  Phrase ; Item
fun
  GivePlease    : Item -> Phrase
  HereWeAre     : Phrase
  ThankYou      : Phrase
  YouAreWelcome : Phrase
  ABeer         : Item
```

The model could be made more precise by specifying that these phrases must appear in a certain order to constitute a valid dialogue. But for the purposes of a phrasebook, it is enough to specify uniquely each type of phrase by giving it a function name. All functions in this simple model are actually **constants**, i.e. they take no arguments—except `GivePlease`, which takes an `Item` as its argument.

The linguistic realizations of the semantic model are specified by a **concrete syntax**, which tells how trees formed in abstract syntax are **linearized** into strings in different languages. We will return to the details of linearization in Section 3; just to give an example, the following linearization rules (`lin`) could be given for German:

[5] Multilingual On-Line Translation, http://www.molto-project.eu

```
lin
  GivePlease item = item ++ "bitte"
  HereWeAre       = "bitte"
  ThankYou        = "Danke"
  YouAreWelcome   = "bitte"
  ABeer           = "ein Bier"
```

All linearization rules in GF can be also used for **parsing**, that is, converting strings to trees. This tiny example clearly shows that parsing can be **ambiguous**, that is, return more than one tree. The everyday counterpart of parsing ambiguity is shown by the situation where someone asks: "What is *bitte* in English?" The correct answer is that it depends on context: it may mean *please* or *here we are* or *you are welcome*.

In the full MOLTO Phrasebook, none of the 15 languages is unambiguous. What we need is an abstract syntax that formalizes all possible distinctions, so that each abstract syntax tree has a unique linearization in every language. Now, capturing all relevant distinctions in 15 languages might sound like a hopeless task, but in fact the semantic model scaled up quite well when the grammar was extended language by language. After a careful initial design (with awareness of what typically happens in languages), almost no changes were needed in the abstract syntax when new languages were added.

Printed phrasebooks have canned, static phrases, whereas a digital grammar-based phrasebook has rules for forming phrases from smaller expressions. The MOLTO Phrasebook has 42 categories and 290 functions. Of the functions, 130 take arguments and 160 are constants, which means that they are either lexical items or canned phrases. What is a lexical item in one language can be a multiword phrase in another language, as shown for instance by *bitte* vs. *here we are*. The number of phrases is infinite because of recursion, but on the reasonable level of tree depth 3, the Phrasebook has 484,938 abstract syntax trees of phrases.

The full code of the phrasebook, with some documentation, can be found on-line[6]. We will here show a sample of the coverage, and then focus on a few interesting problems created by some of the constructions. Table 1 gives some of the categories, and Table 2 some of the combination functions.

For a detailed sample, let us focus on the category `Action`, and the ways of asking persons for information about themselves and what they do. The complete `Phrase` corresponding to the question

Are you Swedish?

has the tree (in GF's LISP-like notation)

```
PQuestion (QProp (PropAction
  (ACitizen YouFamMale (CitiNat Swedish))))
```

This tree is formed by the functions

```
PQuestion  : Question -> Phrase
QProp      : Proposition -> Question
```

[6] http://www.grammaticalframework.org/examples/phrasebook/doc-phrasebook.html

category	explanation	example
Phrase	complete phrase, unit of translation	*Where are you?*
Greeting	idiomatic greeting	*hello*
Sentence	declarative sentence	*I am in the bar*
Question	question, either yes/no or wh	*where are you*
Proposition	can be used as sentence or question	*this pizza is good*
Object	object of wanting, ordering, etc	*two pizzas and a beer*
Item	a single entity	*this pizza*
Kind	a type of an item	*pizza*
Quality	qualification of an item	*very good*
Place	location	*the bar*
PlaceKind	type of location	*bar*
Person	agent wanting or doing something	*you*
Action	proposition about a Person	*you are here*
Nationality	complex of language, property, country	*Swedish, Sweden*
Language	language (can be without nationality)	*Flemish*
Citizenship	property (can be without language)	*Belgian*
Country	country (can be without language)	*Belgium*
Currency	currency	*Swedish crown*
Number	number expression in words	*two hundred and five*
Price	price (number + currency)	*sixty-five dollars*

Table 1. Some of the 42 categories of the Phrasebook.

```
PropAction : Action -> Proposition
ACitizen   : Person -> Citizenship -> Action
YouFamMale : Person
CitiNat    : Nationality -> Citizenship
Swedish    : Nationality
```

But thinking in terms of reliable translations, there are many more trees, resulting from the semantic ambiguity of English *you*. Of these, the Phrasebook deals with dimensions of gender and politeness; plural *you* is not covered by the current version (mostly because it is not so frequently needed). Thus *you* corresponds to four constants of type `Person`,

```
YouFamMale, YouFamFemale, YouPolMale, YouPolFemale
```

Varying this constant in the above tree gives four French linearizations:

YouFamMale: *Est-ce que tu es suédois ?*
YouFamFemale: *Est-ce que tu es suédoise ?*
YouPolMale: *Est-ce que vous êtes suédois ?*
YouPolFemale: *Est-ce que vous êtes suédoise ?*

Although German also has gender, it makes no difference in this example. Thus we obtain

arguments	**value**	**examples**
Number, Kind	Object	*five pizzas*
Quality, Kind	Kind	*Italian pizza*
Kind	Item	*this pizza*, *the pizzas*
PlaceKind	Place	*the bar*, *a bar*
Proposition	Sentence	*the bar is open*, *the bar isn't open*
Proposition	Question	*is the bar open*
Action	Proposition	*I speak Polish*
Person, Object	Action	*you have beer*, *you have no beer*
Person, Citizenship	Action	*you are German*
Person, Place	Action	*you are in the bar*
Person, Sentence	Action	*you know that I am in the bar*
Person, Person	Action	*you know my wife*
Person, Question	Action	*you know how far the bar is*
Person, Number	Action	*I am seventy years old*
Person, Number	Action	*I have six children*
Person, Name	Action	*my name is Bond*
Person	Action	*I am hungry*
Person, Item	Action	*I like this pizza*
Person, Country	Action	*I live in Sweden*
Person, Language	Action	*I speak Polish*
Person, Currency	Action	*I have Swedish crowns*
Person, Object	Action	*I want two apples*
Person, Place	Action	*I want to go to the hospital*
Person	Question	*how old are you*
Item	Question	*how much does the pizza cost*
Item, Price	Proposition	*the pizza costs five euros*
Place	Proposition	*the museum is open*
Place, Date	Proposition	*the museum is open today*
Place, Day	Proposition	*the museum is open on Mondays*
Place, Date	Greeting	*see you in the bar on Monday*
Person	Person	*my wife, your husband*
Number, Currency	Proposition	*five euros*
Place	Question	*how far is the zoo*
Place, Place	Question	*how far is the centre from the hotel*
Transport, Place	Question	*which bus goes to the hotel*

Table 2. Some of the 130 combination rules of the Phrasebook.

YouFamMale, YouFamFemale: *Bist du schwedisch?*
YouPolMale, YouPolFemale: *Sind Sie schwedisch?*

One challenge in the Phrasebook is to communicate the ambiguities to the end user: when she types in *Are you Swedish?*, she should get a list of the alternatives in the desired target language, with explanations that enable her to decide which alternative to choose in her situation of use. We will return to this question in Section 4.

As `Action` is a subcategory of `Proposition`, it can be used for both questions and assertions, both positive and negated. Thus the functions involved in the question can be reused for sentences like *I am not Swedish*, which has two French translations. In general, the design of the abstract syntax follows two principles, which can be explained via geometrical metaphors:

- **Convexity**: for any two phrases contained, also all phrases "between" them (i.e. combining their concepts in different ways) are contained. This principle guarantees that the users can easily learn what to expect from the phrasebook, and their expectations will be fulfilled.
- **Orthogonality**: phrases are built from the least number of independent components.

While convexity is a great help for the user of the phrasebook, orthogonality helps the developer by giving her the minimum of concepts to implement for each language. A user who knows that the Phrasebook contains the property *Swedish* and the country *France* will, by convexity, expect it also to contain the property *French* and the country *Sweden*. The category `Nationality` is used to guarantee this, as it collects triples of language, property, and country. These triples can often be formed by a systematic word formation mechanism (e.g. *Swedish, Swedish, Sweden*), which helps the developer.

As a downside, abstract concepts like `Nationality` may be more complex to implement than more specific concepts like `Language`, `Citizenship`, and `Country`. [7] Often there is no regular word formation mechanism, and there are countries and languages that do not fit into the "national state" concept. For instance, the languages spoken in Belgium are Flemish and French. Thus in the Phrasebook, *Belgium* is a country without a lexically associated language, whereas *Flemish* is a language without a lexically associated country.

The set of combination rules in the Phrasebook is quite useful as it is, but the set of lexical items is still small and a little random. Therefore an obvious next step in developing the Phrasebook is to add words for drinks, food, nationalities, places, and so on. Keeping all this in synchrony for 15 simultaneous languages is not trivial.

3 Concrete Syntax

Constant phrases, such as *thank you* and *please*, are easy to define for all languages. Combination rules are more tricky: even in the small fragment covered by the Phrasebook, linguistic problems such as inflection, agreement, and word order arise, and require expertise in the grammar of each of the target languages. Fortunately for the

[7] The terminological choice between "Nationality" and "Citizenship" is of course arbitrary, and only an implementation detail not visible to the end user.

Phrasebook, this expertise was readily available in the GF Resource Grammar Library (RGL). This is of course not just a lucky coincidence—it is more proper to say that the Phrasebook was built as a showcase of GF in general and of the RGL in particular.

A concrete syntax has two components. One is linearization rules (`lin`) as shown above, telling how abstract trees are mapped into strings. The other one is the **linearization types** of categories (`lincat`). These types are linguistic categories such as sentences, noun phrases, and adjectives. In the `lin` rule examples so far, only one linearization type was used: the type `Str` of strings. But this is usually not enough. For instance, to account for all combinations of a German noun, we need the type

```
{s : Number => Case => Str ; g : Gender}
```

that is, a record with a string depending on number and case (the component `s`), and a gender (component `g`). In other languages, nouns can have other linearization types, and the features number, case and gender can get other values than in German. But in RGL, all this complexity is defined internally, and the user only needs to know that the type `CN` covers common nouns in all RGL languages.[8]

To give a sample of linearization types used in the Phrasebook, let us consider the categories needed in the example *Are you Swedish?*:

category	**linearization type**	**explanation**
Phrase	`Text`	text
Question	`QS`	question
Proposition	`Cl`	clause
Person	`NP`	noun phrase
Action	`Cl`	clause
Citizenship	`A`	adjective
Nationality	`{l : NP ; p : A ; c : NP}`	NP, adjective, NP

All these types are standard linguistic categories of RGL, except the one of `Nationality`, which uses a record consisting of a language noun phrase `l`, a property adjective `p`, and a country noun phrase `C`. This record is, so to say, the linguistic representation of the complex concept of a nationality, thus representing a **lexical family**.

The linearization rules are specified by RGL functions, most of which have the name `mk`*C* for the value category *C*. Thus we have

```
PQuestion q = mkText q
QProp p = mkQS (mkQCl p)
PropAction a = a
ACitizen p c = mkCl p c
YouFamMale = youSg_Pron
CitiNat n = n.p
Swedish = mkNationality "Sweden" "Swedish"
```

The last rule uses the operation `mkNationality`, which takes a string for a noun and for an adjective, to form the country name from the noun (*Sweden*) whereas both the

[8] See http://grammaticalframework.org/lib/doc/synopsis.html for RGL categories and functions.

property and the language use the adjective (*Swedish*). This is the only English-specific rule in this set. Other languages have different ways of defining this lexical family. Finnish, for instance, uses the country name as the language name, just spelled with a small initial.

Another example of a lexical family is types of locations. They are defined

```
PlaceKind = {name : CN ; at : Prep ; to : Prep}
```

Thus places have associated prepositions, used for expressing location and direction. For instance, in English we have *in the bar*, *at the station* for the location and *to* expressing the direction for both. In Finnish, prepositions are expressed by cases, so that "bar" uses so-called internal cases (*baarissa* "in the bar" inessive, *baariin* "to the bar" illative) whereas "station" uses external cases (*asemalla* "at the station" adessive, *asemalle* "to the station" allative). Sometimes even more fine-grained distinctions are needed; for instance, in Swedish "to the toilet" is expressed as *på toaletten* in phrases relating to the function ("I want to go to the toilet"), whereas phrases expressing pure direction say *till toaletten*.

The prepositions are thus stored in the record as lexical properties of the places; they are idiomatic in each language and highly unpredictable. GF provides ways to express them on a reasonably high level, so that just the minimal information need be given in the lexicon: thus in Finnish, we just need the noun and an identifier `ssa` or `lla` which is conventionally used for indicating the type of local case:

```
Bar = mkPlace (mkN "baari") ssa
Station = mkPlace (mkN "asema") lla
```

To determine this little piece of information—the proper case or preposition for each location—is linguistic knowledge that turned out to be possessed only by native speakers, who made several corrections to the initial grammars.

As the RGL has a common API for the syntax functions of the 18 languages included, combination rules in application grammars can in principle be expressed by code that is common to all languages. This is technically implemented by the use of **functors** ([10], chapter 5), and it is the technique used, for instance, in the GF implementation of Attempto Controlled English [13]. The use of a functor means that the languages use the same syntactic structures to express the meanings. For instance, all languages in the Phrasebook use an equivalent of *you know that I am in the bar* to express this proposition. However, the Phrasebook domain is particularly rich of idioms that the languages express by different syntactic means. This was a challenge we expected, and one of the reasons why the Phrasebook was an interesting case study for multilingual translation in the first place. Thus, of the 130 combination rules, only 96 (74%) are implemented by a functor; usually the percentage is close to 100.

Some typical examples of non-functorial expressions are the following:

- *I am fifty years old*: French *j'ai cinquante ans* ("I have fifty years").
- *my name is Bond*: German *ich heisse Bond* ("I have-name Bond"), French *je m'appelle Bond* ("I call myself Bond").
- *I am hungry*: French *j'ai faim* ("I have hunger"), Finnish *minun on nälkä* ("of-me is hunger").

– *I like this pizza*: Italian *questa pizza mi piace* (“this pizza pleases me”).
– *I am married*: Finnish *olen naimisissa* (“I am in-marriage”, with a special adverbial).
– *how old are you*: French *combien d’ans as-tu* (“how many years do you have”).
– *how far is the station*: French *à quelle distance est la gare* (“at what distance is the station”), Italian *quanto dista la stazione* (“how much does the station distance”, with a special verb).

Most of these variations are clustered in systematic ways, so that for instance all Romance languages use the same structure and all Germanic languages (except perhaps English) another structure. The construct *how* with an adjective or adverb does not exist in Romance languages, and is hence not even a part of the RGL API.

4 Ambiguity and Disambiguation

The abstract syntax is by definition unambiguous. Therefore the main way in which a grammar developer can analyse the ambiguity of a string is by inspecting the abstract syntax trees. But this device is of course not appropriate for a tourist phrasebook: it would be awkward and often useless to show the syntax trees to the user.

Fortunately, the technique of multilingual grammars provides a straightforward, declarative way to display ambiguities: one can write for each language a special concrete syntax, which is like the original grammar except that it eliminates its ambiguities by using alternative (although less idiomatic and often longer) expressions—a **disambiguation grammar**. For example, the original English grammar linearizes each of the four abstract variants of *you* as just *you*, but the disambiguation grammar attaches an explanation in parentheses: *you (familiar,male)*, *you(polite,female)*, etc. This idea is inspired by the notion of **feedback texts** of the WYSIWYM system [14].

The implementation of a disambiguation grammar can be written on top of the base grammar by using **restricted inheritance**: it inherits everything from the base grammar except those rules that need disambiguation. Those rules can then be replaced by other rules. The following module is a complete code for a disambiguation grammar for the phrasebook dealing with the four *you*’s. The unambiguous variants are formed from *you* by attaching an adverbial to it.

```
concrete DisambPhrasebookEng of Phrasebook = PhrasebookEng -
  [YouFamMale, YouFamFemale, YouPolMale, YouPolFemale]
  ** open SyntaxEng, ParadigmsEng in {
lin
  YouFamMale   = mkNP you_NP (mkAdv "(familiar,male)") ;
  YouFamFemale = mkNP you_NP (mkAdv "(familiar,female)") ;
  YouPolMale   = mkNP you_NP (mkAdv "(polite,male)") ;
  YouPolFemale = mkNP you_NP (mkAdv "(polite,female)") ;
}
```

In the full Phrasebook, the number of ambiguous constructs is between 10 and 20 for each language. An ambiguity shared by all languages is the notion of the currency

crown, as used for the currency of different Scandinavian countries. In the normal usage, one says *crown* rather than e.g. *Danish crown*, if it is clear from the context that one is speaking about Danish crowns. The implementation of this does not use the disambiguation grammar, because both expressions make sense in the base grammar as well. Thus the base grammar defines crowns by using the **variants** construct of GF (expressed by |):

```
DanishCrown =
  mkCN (mkA "Danish") (mkN "crown") | mkCN (mkN "crown")
SwedishCrown =
  mkCN (mkA "Swedish") (mkN "crown") | mkCN (mkN "crown")
```

and similarly in all languages.

Since the abstract syntax encodes all interpretations that are relevant in any of the languages, it can lead to **spurious ambiguities** when applied to any particular language pairs. For instance, the familiar *you* is *sinä* and the polite *you* is *Te* in Finnish, without the gender distinction involved anywhere in the sentence. Hence, when translating from English to Finnish, only two alternatives should be displayed. The same thing may happen in Italian, where the masculine and feminine forms of some adjectives are the same. Thus *are you Swedish* has only two translations (*sei/è svedese*) even though *are you Italian* has four (*sei/è italiano/italiana*).

In the Phrasebook, the user should of course not see spurious ambiguities but only relevant ones. This is guaranteed by the following modification of GF's translation algorithm, which otherwise shows as many translation strings as there are parse results. Each translation is equipped by the set of those disambiguating expressions that give rise to it. The translation algorithm is as follows:

1. parse the source sentence to obtain trees $t_1, \ldots, t_n$
2. for each target language L_i:
   (a) for each tree t_j: linearize t_j in L_i
   (b) group trees with the same linearization s_k into the pair $< s_k, \{t \mid t^* = s_k\} >$
   (c) return each s_k together with the linearizations of the associated trees in the disambiguation grammar of the target language

Here is an example of the algorithm at work:

English input:
- *Are you Swedish?*

French output:
- *Est-ce que tu es suédois ?* (Are you (Familiar,Male) Swedish?)
- *Est-ce que tu es suédoise ?* (Are you (Familiar,Female) Swedish?)
- *Est-ce que vous êtes suédois ?* (Are you (Polite,Male) Swedish?)
- *Est-ce que vous êtes suédoise ?* (Are you (Polite,Female) Swedish?)

Italian output:
- *Sei svedese?* (Are you (Familiar,Male) Swedish? / Are you (Familiar,Female) Swedish?)
- *È svedese?* (Are you (Polite,Male) Swedish? / Are you (Polite,Female) Swedish?)

As a further optimization, the algorithm could compress the alternatives (Familiar,Male) and (Familiar,Female) to just (Familiar). This would be helped by a disambiguation grammar that has more structure than just the unanalysed strings in parentheses. One could also achieve this by some hand-written code in the phrasebook application; however, this would be against the purpose of developing the Phrasebook as a show-case for a general technology.

5 Example-Based Grammar Writing

In previous projects, the typical author of a GF concrete syntax is fluent in the target language and has GF skills which are directly proportional to the complexity of the abstract syntax to implement. However, when dealing with 15 languages and a reasonably rich semantic interlingua, the task of finding such people is a difficult one. When adding the time constraints yielded by the MOLTO deadlines and the time needed to improve a native speaker's GF skills or a GF programmer's knowledge of a language that she had little to no skill in before, the task seemed to be a mission impossible. This was the case for German, Dutch, Danish, Norwegian and Polish. As a solution to this, we devised the example-based grammar learning system, that is meant to automate a significant part of the grammar writing process and ease grammar development. The two main usages of the system are, first, to reduce the amount of GF programming necessary in developing a concrete grammar, and, secondly and more importantly, to make the extraction of certain features of a language automatic for grammar development.

In the last years, the GF community has constantly increased and so has the number of languages in the resource library and the number of application grammars using them. The writer of a concrete application grammar is typically different from the writer of the resource grammar for the same language, has less GF skills and is most likely unaware of the almost 300 constructors that the resource grammars implement for building various syntactical constructions [11]. In order to hide this detail, an API is provided so that the domain grammar writer only needs to know the GF categories and look up how they can be built from each other.

For example, the sentence *my name is John* is parsed to the following abstract syntax tree:

```
PredVP (DetCN (DetQuant (PossPron i_Pron) NumSg)
  (UseN name_N)) (UseComp (CompNP (UsePN john_PN)))
```

If we use the API constructors, the abstract syntax tree is simpler and more intuitive, as the structure is flatter and each function has an easily memorable name:

```
mkCl (mkNP (mkQuant i_Pron) name_N) (mkNP john_PN)
```

The example-based grammar learning system aims to make one step more in this direction and reduce the need for using even the API functions. The key idea is based on parsing, followed by compilation to API. It provides considerable benefits, especially for idiomatic grammars such as the Phrasebook, where the abstract syntax trees are considerably different.

For example, when asking for a person's name in English the question *what is her name* has the syntax trees shown above. On the other hand, in French the question would be translated to *je m'appelle John* (literally, "I call myself John"), which is parsed to:

```
PredVP (UsePron i_Pron)
  (ComplSlash (SlashV2 appeler_V2) (UsePN john_PN))
```

and corresponds to the following API abstract tree:

```
mkCl i_NP appeler_V2 (mkNP john_PN)
```

By replacing `i_NP` and `john_PN` with variables, this tree can be used as the linearization of a two-place predicate:

```
lin HasName x y = mkCl x appeler_V2 (mkNP y)
```

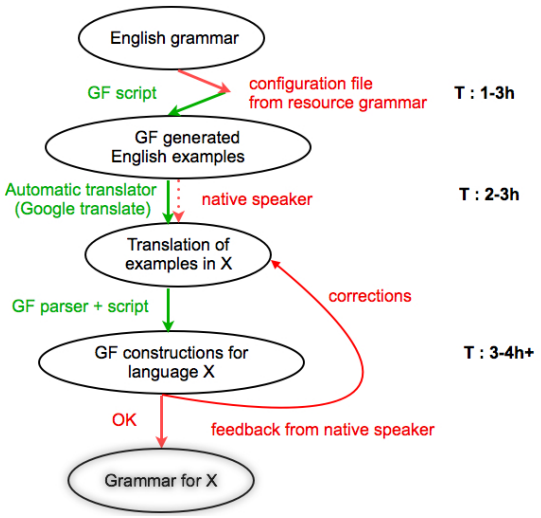

Fig. 2. The example-based grammar learning schema

Figure 2 shows the algorithm for example-based grammar writing. It shows the construction steps of the concrete syntax of the Phrasebook grammar for the language *X*, where the developer has basic or no skills in the language. In our experiment *X* was one of Danish, Dutch, German, Norwegian, and Polish. The arrows represent the main steps of the process, whereas the circles represent the initial and final results after each step of the process. For every step, the estimated time is given. This is variable and greatly influenced by the features of the target language and the semantic complexity of the phrases and would only hold for the Phrasebook grammar.

Initial resources:

- English Phrasebook
- resource grammar for X
- script for generating the inflection forms of words and the corresponding linearizations of the lexical entries from the Phrasebook in the language X. For example, in the case of the nationalities, since we are interested in the names of countries, languages and citizenship of people and places, we would generate constructions like "I am English. I come from England. I speak English. I go to an English restaurant" and from the results of the translation we will infer the right form of each feature. In English, in most cases there is an ambiguity between the name of the language and the citizenship of people and places, but in other languages all three could have completely different forms. This is why it is important to make the context clear in the examples, so that the translation will be more likely to succeed. The correct design of the test of examples, is language dependent and assumes analysis of the resource grammar, also. For example, in some languages we need only the singular and the plural form of a noun in order to build its GF representation, whereas in other languages such as German, in the worst case we would need 6 forms which need to be rendered properly from the examples.
- script for generating random test cases that cover all the constructions from the grammar. It is based on the current state of the abstract syntax and it generates for each abstract function some random parameters and shows the linearization of the construction in both English and language *X*, along with the abstract syntax tree that was generated.

Example-based concrete grammar learning algorithm:

- Step 1: **Analysis of the target grammar and lexicon acquisition**
  The first step assumes an analysis of the resource grammar and extracts the information needed by the functions that build new lexical entries. A model is built so that the proper forms of the word can be rendered, and additional information, such as gender, can be inferred. The script applies these rules to each entry that we want to translate into the target language, and one obtains a set of constructions.
- Step 2: **Generation of examples in the target language**
  The generated constructions are given to an external translator tool (Google translate) or to a native speaker for translation. One needs the configuration file even if the translator is human, because formal knowledge of grammar is not assumed.
- Step 3: **Parsing and decoding the examples with GF**
  The translations into the target language are further more processed in order to build the linearizations of the categories first, decoding the information received. Furthermore, having the words in the lexicon, one can parse the translations of functions with the GF parser and generalize from that.
- Step 4: **Evaluation and correction of the resulting grammar**
  The resulting grammar is tested with the aid of the testing script that generates constructions covering all the functions and categories from the grammar, along with some other constructions that proved to be problematic in some language. A

native speaker evaluates the results and if corrections are needed, the algorithm runs again with the new examples. The examples validated by the native informant are kept for regression testing of the future results. The algorithm is repeated as long as corrections are needed.

It is worth noting that the time needed for preparing the configuration files for a grammar will not be repeated, since the files are available for future usage. The time for the second step can be saved if automatic tools, like Google translate are used. This is only possible in languages with large corpora available. Good results were obtained for German and Dutch with Google translate, but for languages like Polish, which are both complex and lack enough resources, the results are discouraging. If the statistical oracle works well, the only step where the presence of a human translator is needed is the evaluation and feedback step. An average of 4 hours per round and 2 rounds were needed for the languages for which we performed the experiment. The final results are comparable to a grammar developed by a native speaker GF programmer.

However, one can already remark that the success of this method also depends highly on the lexicon acquisition, which we perform in the first step. What is more is that the lexicon is language-dependent, and is not alignable. Also, without previous knowledge of all the languages, one cannot foresee what words we would need to use, and since they are not used in all languages, it wouldn't make sense to have them all in a multilingual aligned lexicon. For the moment, this task was solved by either guessing the correct part-of-speech based on a similar concrete grammar already developed(for example Danish and Norwegian were bootstrapped from Swedish) or by having the lexicon built and POS-tagged with the aid of native informants.

Among the 5 languages considered, a concrete Phrasebook grammar was successfully built for Danish, Dutch, German and Norwegian, whereas for Polish, it was not possible to get through the first and most difficult step—target grammar analysis and lexicon acquisition, because of the complex morphology of the language and the lack of available resources. In the end the concrete grammar was developed by the writer of the resource grammar.

The experiment involved 7 programmers with basic or advanced GF skills that wrote 10 resource grammars, whereas for the 4 languages mentioned before, the example-based algorithm was used. The approximate development total time is 1 person month for the whole Phrasebook, or 1.5 days per language on the average.

Based on this case study, we roughly estimated the effort used in constructing the necessary sources for each new language and compiled Table 3.

Explanation of the scores

- Grammarian's language skills:
  - – : no skills
  - # : basic skills(general knowledge of the grammar)
  - ## : medium skills(fluent)
  - ### : advanced skills(native speaker)
- Grammarian's GF skills
  - — : no skills

Language	Fluency	GF skills	Inf. dev.	Inf. testing	Ext. tools	RGL edits	Effort
Bulgarian	###	###	-	-	-	#	##
Catalan	###	###	-	-	-	#	#
Danish	-	###	+	+	+	##	##
Dutch	-	###	+	+	+	#	##
English	##	###	-	+	-	-	#
Finnish	###	###	-	-	-	#	##
French	##	###	-	+	-	#	#
German	#	###	+	+	+	##	###
Italian	###	#	-	-	-	##	##
Norwegian	#	###	+	+	+	#	##
Polish	###	###	+	+	+	#	##
Romanian	###	###	-	-	+	###	###
Spanish	##	#	-	-	-	-	##
Swedish	##	###	-	+	-	-	##

Table 3. Development effort for the Phrasebook.

  - # : basic skills(simple GF exercises)
  - ## : medium skills(more comprehensive GF exercises)
  - ### : advanced skills(resource grammar writer/substantial contributor)
- Informant needed for development/Informant needed for testing
  - —: no
  - + : yes
- Changes on the resource grammars
  - —: no changes
  - # : 1-3 minor changes
  - ## : 4-10 minor changes, 1-3 medium changes
  - ### : $>$10 changes of any kind
- Overall effort
  - # : less than 8h/person
  - ## : 8-24h/person
  - ### : $>$24h/person

This experiment is significant because it is a showcase for the ongoing work on example-based concrete grammar learning technology which will make GF grammar writing easier in terms of adding more languages and developing larger grammars, but also because it represents an analysis on the possible interaction of GF with other available translation tools, which will ease the work of both beginners and advanced users of the technology.

6 The Mobile Application

If one wants to build a tool for a controlled language for everyday usage, it seems logical for this tool to be as unobtrusive as possible. Moreover, since our language is

targeting tourists, we have to take into account a particular setting where people, when going on vacation, may not have access to a computer and access to Internet can be very limited due to low coverage or prohibitive costs. This are the criteria that we tried to meet when building PhraseDroid, an application that works offline, on smartphone devices and with a simple user interface. Moreover, we wanted to do this by creating a technology that is as general as possible, and in fact applies to any multilingual GF grammar.

PhraseDroid is an Android application, that can be used on handheld devices running the Android operating system. Figure 1 shows a screen shot of the application in its current state. As you can see, the application is using the same "magnet interface" as the web application. This permits the user to compose a sentence while staying in the coverage of the grammar. Moreover, this kind of interaction works well on devices with touch screens because the magnets are large enough to be able to be selected with fingers.

What is more is that the Android platform provides a high-quality speech synthesis for several of the languages covered by the grammar, which can be plugged into our application. This gives clear benefits compared to a traditional (paper) phrasebook.

As mentioned in Section 1, there are more and more phrasebook applications developed for smartphones nowadays. They can be divided in two main categories:

1. The finite phrasebook. Those are usually made of a list of sentences translated in one, or more, foreign languages. Those phrasebooks are lacking from the point of view of expressivity since it isn't possible to change a sentence as needed, even if a very similar sentence is covered by the phrasebook.
2. The application providing machine translation through an on-line service. The Google Translate application (and the various applications that are just front-ends to it) is the best example in this category. This kind of applications can obviously be used while traveling, but they require the possibility to connect to the Internet, which is not guaranteed when one is travelling abroad due to technological or economical reasons. In addition, unlike in our application, the translation engine is not tailored toward tourist usage but is generic, which can lead so sub-optimal translation in many cases.

In contrast, our application, once installed, works off-line and features a grammar design specifically for tourist translations. And since the user inputs the sentence to be translated herself, it allows a great deal of variation and fine-tuned translation for a given situation.

The application is based on a Java interpreter for GF's binary grammar format, called PGF [15]. Therefore the application is very modular: adapting the application to a different controlled language requires little more than dropping a new pgf file in the right folder. This means that one can in no time create a translation application for another controlled language given that a GF grammar for this language has been written.

Thus an important part of the process of creating the application was to write a library in Java that provides the functions needed in the application. Its usage is not limited to Android phones, but it can also be plugged in into any Java program, whether on a desktop computer, or a web browser plug-in. In the current state, the Java library

supports (predictive) parsing, linearization, and random generation of well-typed trees. This is less than the features available in the full GF interpreter, written in Haskell, but it is sufficient for machine translation and lots of other uses.

7 Evaluation

7.1 Translation Quality

This is the first criterion of evaluation. It was first assessed by the systematic use of native speaker testers, and later by comments collected from more random users of the web demo and the Android application. The goal has been what might be called *perfect quality*, in terms of meaning-preservation, grammaticality, idiomaticity, and fluency. Hence all errors found in earlier versions were corrected immediately. With the first "official version" (the one also running on Android), few direct errors have been found, but there are some inadequacies that appear in reports:

- Some sentences permitted by the abstract syntax are semantically anomalous, e.g. *is there an airplane to the toilet*. This could be fixed by using a more strict type system; however, we consider this to be less important as long as the translations are correct.
- The choice of prepositions is not always fine-grained enough; for instance the distinction between *gå till toiletten* and *gå på toaletten* (both "go to the toilet") in Swedish is not handled (cf. Section 4).
- The usage of nationality adjectives for persons is not always optimal, but nouns should be introduced in the lexical family. Thus *I am a Finn* would be better than *I am Finnish*, with corresponding variations in many languages.

7.2 Coverage

There is no end of conceivable extensions if we want to cover everything that a tourist might want to say. The syntactic combination rules are sufficient for many situations, but they should definitely be extended with more vocabulary. For instance,

- drinks, food, currencies, countries
- time expressions like *half past eight*
- free-string input for names of places and persons

7.3 Engineering Effort

One of the main goals of the MOLTO project is to improve the productivity of GF-based translation systems "by an order of magnitude". This means that the development time of translation systems should be shortened to 10% of the original. The development time for the Phrasebook was two working days per language on average. If this is the baseline to be compared with at the end of the project (in 2013), then a new language should be possible to add in a couple of hours. Some of this improvement can be possible to reach by a better use of example-based grammar writing.

However, some parts of the grammar may be inherently difficult, due to idiomatic structures. Another way to interpret the productivity improvement would then be in

terms of the concepts covered. If the first Phrasebook built in two days covers hundreds of concepts, a realistic goal could be to cover thousands of concepts in the same time. To this end, methods of automatic lexicon extraction are being developed, with ontologies, terminologies, and statistical translation models as sources.

7.4 Usability

The size of the run-time PGF grammar is 500 kB. It runs smoothly on both web applications and mobile phones. For mobiles, a substantial optimization effort was needed, but it was made on the level of GF and will therefore benefit all future applications.

The web application provides the input method of typing strings, which the mobile doesn't have. This will certainly become an issue when the Phrasebook is extended to contain thousands of concepts. It will also become an issue how to navigate in the large space of words to find exactly the phrase one wants to use. A hierarchical approach similar to syntax editors [16] will probably be introduced as a useful complement to string-based input.

The mobile application has some usability issues reported by users, which will have to be addressed in future releases.

8 Conclusion

We have explained a controlled language approach to a multilingual tourist phrasebook, covering 15 languages. While intending to build a useful application for travellers, we have also seen it as an experiment to extend the notion of controlled language and scale it up in various respects:

- extending the notion of semantics from logic to "language games" (Section 2)
- porting a controlled language from one language to many (Section 3)
- coping with ambiguity, rather than banning it (Section 4)
- making it easier to implement controlled languages, in terms of both effort and skill (Section 5)
- building applications for laymen rather than specialists, and making them run on light devices (Section 6)

Our conclusion is that there is a lot of potential in controlled languages to become more useful in everyday life, the multilinguality aspect being at least as interesting for laymen as the traditional reasoning aspect is.

Acknowledgements The MOLTO Phrasebook is a collaborative project. In addition to the authors of this paper, it has involved Krasimir Angelov, Olga Caprotti, Thomas Hallgren, Inari Listenmaa, Jordi Saludes, Adam Slaski, and Shafqat Virk as programmers and Richard Bubel, Rise Eilert, Karin Keijzer, Michał Pałka, Willard Rafnsson, and Nick Smallbone as testers and informants. The research leading to these results has received funding from the European Union's Seventh Framework Programme (FP7/2007-2013) under grant agreement n:o FP7-ICT-247914.

References

1. The Boeing Company: Boeing Simplified English Checker. http://www.boeing.com/assocproducts/sechecker/ (2001)
2. Shiffman, R.N., Michel, G., Krauthammer, M., Fuchs, N.E., Kaljurand, K., Kuhn, T.: Writing clinical practice guidelines in controlled natural language. In: Proceedings of the 2009 conference on Controlled natural language. CNL'09, Berlin, Heidelberg, Springer-Verlag (2010) 265–280
3. Hart, G., Johnson, M., Dolbear, C.: Rabbit: Developing a control natural language for authoring ontologies. In: ESWC. (2008) 348–360
4. Fuchs, N.E., Kaljurand, K., Kuhn, T.: Attempto Controlled English for Knowledge Representation. In Baroglio, C., Bonatti, P.A., Małuszyński, J., Marchiori, M., Polleres, A., Schaffert, S., eds.: Reasoning Web, Fourth International Summer School 2008. Number 5224 in LNCS, Springer (2008) 104–124
5. Dean, M., Schreiber, G.: OWL Web Ontology Language Reference (2004) http://www.w3.org/TR/owl-ref/.
6. Gruzitis, N., Barzdins, G.: Towards a More Natural Multilingual Controlled Language Interface to OWL. In: 9th International Conference on Computational Semantics (IWCS). (2011) 335–339 http://www.aclweb.org/anthology/W/W11/W11-0138.pdf.
7. Wittgenstein, L.: Philosophical Investigations. Basil Blackwell, Oxford (1953)
8. Ganesalingam, M.: The Language of Mathematics. PhD thesis, Department of Computer Science, University of Cambridge (2010) http://people.pwf.cam.ac.uk/mg262/.
9. Angelov, K.: Incremental Parsing with Parallel Multiple Context-Free Grammars. In: EACL'09, Athens. (2009)
10. Ranta, A.: Grammatical Framework: Programming with Multilingual Grammars. CSLI Publications, Stanford (2011) ISBN-10: 1-57586-626-9 (Paper), 1-57586-627-7 (Cloth).
11. Ranta, A.: The GF Resource Grammar Library. Linguistics in Language Technology **2** (2009) http://elanguage.net/journals/index.php/lilt/article/viewFile/214/158.
12. Bringert, B., Angelov, K., Ranta, A.: Grammatical Framework Web Service. In: System demo, Proceedings of EACL'09, Athens. (2009)
13. Ranta, A., Angelov, K.: Implementing Controlled Languages in GF. In: Proceedings of CNL-2009, Athens. Volume 5972 of LNCS. (2010) 82–101
14. Power, R., Scott, D.: Multilingual authoring using feedback texts. In: COLING-ACL 98, Montreal, Canada (1998)
15. Angelov, K., Caprotti, O., Enache, R., Hallgren, T., Listenmaa, I., Ranta, A., Saludes, J., Slaski, A.: D10.2 molto web service, first version. (D10.2) (06/2010 2010)
16. Khegai, J., Nordström, B., Ranta, A.: Multilingual Syntax Editing in GF. In Gelbukh, A., ed.: Intelligent Text Processing and Computational Linguistics (CICLing-2003), Mexico City, February 2003. Volume 2588 of LNCS., Springer-Verlag (2003) 453–464 http://www.cs.chalmers.se/~aarne/articles/mexico.ps.gz.