SimpleNLG for German

Marcel Bollmann
bollmann@linguistics.rub.de
Department of Linguistics
Ruhr-Universität Bochum, Germany

May 27, 2013

Contents

1 What is SimpleNLG for German?

SimpleNLG for German[1] is an adaption of the original SimpleNLG by Robert Dale and Ehud Reiter.[2] It is a Java library—*not* a stand-alone program!—that can be used in software projects to generate grammatically correct sentences in German. For details, please refer to the website of the original SimpleNLG; it provides a great tutorial about the basic ideas and concepts behind the framework.

Unfortunately, this adaption of SimpleNLG for German has never been quite finished. There will be bugs. Also, please note that this version of SimpleNLG for German is based on version 3.8 of SimpleNLG, which has since been replaced by the 4.x versions with a different architecture.

If you are interested in continuing or contributing to the development of this software, please don't hesitate to contact me (`bollmann@linguistics.rub.de`).

If you would like to cite SimpleNLG for German in your own work, please refer to the following publication:

> Bollmann, Marcel (2011). Adapting SimpleNLG to German. In *Proceedings of the 13th European Workshop on Natural Language Generation (ENLG)* (pp. 133–138). Nancy, France.

1.1 About this document

This guide gives a brief overview on how to use SimpleNLG for German. At its current stage, it is better seen as a collection of examples rather than a comprehensive guide. Knowledge of Java is assumed, and familiarity with the original SimpleNLG package is definitely recommended. Also, I recommend using a Java IDE such as Eclipse[3] which provides auto-completion of function and variable names, as this document does not contain exhaustive documentation of all possible options at the moment.

Further examples on using SimpleNLG for German can be found in the—unfortunately rather unorganized—collection of JUnit test found in the appropriately named RandomTests.java file.

2 Basic usage

This section explains how to create simple phrases and combine them into sentences.

[1] At the moment, the software package is available from `http://www.linguistics.rub.de/~bollmann/`.
[2] The original SimpleNLG is available from `http://code.google.com/p/simplenlg`.
[3] `http://www.eclipse.org/`

2.1 Getting started

Before you can use SimpleNLG for German in your project, some basic objects need to be instantiated: a lexicon, which provides all necessary information about lexical units like grammatical category, gender, etc.; a factory object, which aids the creation of phrases by looking up words in the lexicon and combining them in the right manner; and a realiser, which produces the final output. An example initialisation is shown in Listing 1.

```
Lexicon lex = new DBLexicon(new XMLAccessor("path/to/the/lexicon.xml"));
Realiser r = new Realiser(lex);
NLGFactory factory = new NLGFactory(lex);
```

Listing 1: Setting up SimpleNLG for German

All following examples will assume that variables have been instantiated as shown above.

Important: At the moment, only an XML accessor for lexicon files is implemented. While the original SimpleNLG framework also provides a database accessor, it has not been adapted yet. Unfortunately, this has serious implications for runtime performance, as the XML accessor is slow. For now, I recommend loading the whole lexicon to memory first (which can be done by calling `lex.loadData();`), which might take several seconds for large lexicon files, but is still considerably than looking up single lexicon entries on-the-fly.

When the system has been set up as in Listing 1, basic sentences can then be generated in an easy and intuitive way: helper functions of the phrase factory are used to create phrases, which are then combined to form a sentence. Listing 2 is a simple example.

```
NPPhraseSpec frau = factory.createNounPhrase("die", "Frau");
NPPhraseSpec mann = factory.createNounPhrase("der", "Mann");
SPhraseSpec s1 = factory.createSentence(frau, "sehen", mann);

System.out.println( r.realise(s1) );
```

Listing 2: A basic sentence

Assuming that the lexicon file contains definitions for *Frau*, *Mann*, and *sehen*, this should print the sentence *"Die Frau sieht den Mann."* to the Java console. In all following examples, the last step (calling the realiser) will not be explicitly given as it is the same for each phrase/sentence/text object.

2.2 Noun phrases

To instantiate a noun phrase, simply give the lexical item as a string argument:

(1) `factory.createNounPhrase("Frau");`

If the given string is found in the lexicon, the noun phrase will be inflected automatically if necessary.

2.2.1 Specifiers and gender

To instantiate a noun phrase with a specifier, give the specifier either as a string or as the respective Java object (e.g., a previously instantiated pronoun):

(2) `factory.createNounPhrase("die", "Frau");`

(3) `factory.createNounPhrase("eine", "Frau");`

Article-noun combinations can alternatively be given as a single string, too:

(4) `factory.createNounPhrase("die Frau");`

The specifier can also be changed or added at a later stage:

(5)
```
NPPhraseSpec frau = factory.createNounPhrase("Frau");
frau.setSpecifier("die");
```

Note that the gender of the noun is **not** affected by the form of the article (*der, die, das*) in the string! The gender is always retrieved from the lexicon, or set to neuter if the lexicon does not contain the word (this latter behavior could conceivably be changed in future versions).

Some nouns can be used with several genders, e.g., *der/die Beamte*. In this case, gender should be set explicitly:

(6)
```
NPPhraseSpec beamte = factory.createNounPhrase("die Beamte");
beamte.setGender(Gender.FEMININE);
```

2.2.2 Compounds

In German, compounds are written as one word. They don't need to be explicitly added to the lexicon, though—you can use a vertical bar (`|`) to separate the last compound element from the rest instead. The following examples will use the lexicon entries for *Stadt* and *Kapitän* to correctly inflect the compound nouns built from them:

(7) `factory.createNounPhrase("die", "Heimat|stadt");`

(8) `factory.createNounPhrase("ein", "Donaudampfschifffahrts|kapitän");`

2.2.3 Genitive complements

A noun phrase added as a complement to another noun phrase will result in a genitive construction. Setting it as a specifier instead produces a "Saxon genitive" construction. Both forms can be combined, if desired.

```
NPPhraseSpec frau = factory.createNounPhrase("die", "Frau");
NPPhraseSpec mann = factory.createNounPhrase("der", "Mann");
frau.addComplement(mann); // produces "die Frau des Mannes"

frau.setComplement("");
frau.setSpecifier(mann);  // produces "des Mannes Frau"
```

Listing 3: Genitive complements of noun phrases

2.3 Prepositional phrases

The factory method for creating prepositional phrases requires a preposition and a noun phrase, which—again—can be given both as strings or objects. Prepositions are hard-coded along with the cases they require. The following examples all work to produce the phrase *wegen des Mannes*:

(9) PPPhraseSpec pp = factory.createPrepositionalPhrase("wegen", "der Mann");

(10) PPPhraseSpec pp = factory.createPrepositionalPhrase("wegen", "der", "Mann");

(11) NPPhraseSpec mann = factory.createNounPhrase("der Mann");
PPPhraseSpec pp = factory.createPrepositionalPhrase("wegen", mann);

Some prepositions, like *wegen*, can also be used as postpositions. A boolean value can be given as the third argument to signal this. The following example produces *des Mannes wegen*:

(12) PPPhraseSpec pp = factory.createPrepositionalPhrase("wegen", "der Mann", true);

Quite a few prepositions in German can take both the accusative and the dative case, depending on whether the intended meaning is directional (accusative) or positional (dative). SimpleNLG for German always defaults to the accusative case; if the dative case is required, the case can be explicitly given as a third argument.

```
// produces "auf den Berg":
pp = factory.createPrepositionalPhrase("auf", "der Berg");
// produces "auf dem Berg":
pp = factory.createPrepositionalPhrase("auf", "der Berg", Case.DATIVE);
```

Listing 4: Setting case for prepositions

2.3.1 Contraction of preposition and specifier

Contraction of articles and prepositions is supported. SimpleNLG for German makes a distinction between obligatory contractions and optional ones. Table 1 lists all contractions that are considered obligatory; these will be automatically performed by default. This behavior can be manually overridden by setting the respective property of the PPPhraseSpec object:

(13) pp.setContraction(false);

All other contractions (e.g., *hinter + dem → hinterm*) are disabled by default, but can be generated by calling setContraction(true). If a contraction is impossible, setting this option does not affect the output at all. At the moment, there is no option to set this parameter globally for all prepositional phrases.

an + dem → am
bei + dem → beim
in + dem → im
von + dem → vom
zu + dem → zum
an + das → ans
in + das → ins
zu + der → zur

Table 1: Obligatory contractions of article and preposition

```
// produces "im Haus":
pp = factory.createPrepositionalPhrase("in", "das Haus", Case.DATIVE);
// produces "in dem Haus":
pp.setContraction(false);

// produces "hinter dem Haus":
pp = factory.createPrepositionalPhrase("hinter", "das Haus", Case.DATIVE);
// produces "hinterm Haus":
pp.setContraction(true);

// produces "wegen des Mannes":
pp = factory.createPrepositionalPhrase("wegen", "der Mann");
// produces "wegen des Mannes" -- no contraction possible:
pp.setContraction(true);
```

Listing 5: Examples for contractions of article and preposition

2.4 Adjective phrases

Adjectives only require their base form to instantiate. They should be added to a noun phrase as modifiers, as Listing 6 shows.

```
NPPhraseSpec mann = factory.createNounPhrase("der", "Mann");
AdjPhraseSpec schoen = factory.createAdjectivePhrase("schön");
AdjPhraseSpec reich  = factory.createAdjectivePhrase("reich");
```

```
mann.addModifier(schoen); // produces "der schöne Mann"
mann.addModifier(reich);  // produces "der schöne, reiche Mann"
```

Listing 6: Using adjectives

Comparative and superlative forms are created by calling setDegree. The following examples produce *schöner* and *am schönsten*, respectively:

(14) schoen.setDegree(AdjectiveDegree.COMPARATIVE);

(15) schoen.setDegree(AdjectiveDegree.SUPERLATIVE);

Adjective phrases can be given complements to create a comparison, as Listing 7 shows.

```
// produces "schön wie der Mann":
schoen.addComplement(mann);
// produces "schöner als der Mann":
schoen.setDegree(AdjectiveDegree.COMPARATIVE);
```

Listing 7: Adjective phrases with complements

2.5 Sentences

A basic sentence requires a subject and a verb. An object can also be supplied to the factory method, if required. Indirect objects have to be added separately. Listing 8 shows a few examples for basic sentence creation.

```
// produces "Der Mann isst.":
SPhraseSpec s = factory.createSentence("der Mann", "essen");

// produces "Der Mann isst einen Apfel.":
s = factory.createSentence("der Mann", "essen", "ein Apfel");
// produces "Der Mann isst einen Apfel.":
s = factory.createSentence("der Mann", "essen");
s.addComplement(factory.createNounPhrase("ein Apfel"));

// produces "Die Frau gibt dem Mann einen Apfel.":
s = factory.createSentence("die Frau", "geben", "ein Apfel");
s.addIndirectObject(factory.createNounPhrase("der Mann"));
```

Listing 8: Basic sentence building

Direct objects are added with addComplement and are realised in the accusative case. Indirect objects are added with addIndirectObject and are always rendered in the dative case. In the rare case of genitive objects, it is useful to call addComplements with an extra argument indicating the case:

(16)
```
s = factory.createSentence("die Frau", "gedenken");
s.addComplement(DiscourseFunction.GENITIVE_OBJECT, mann);
// "Die Frau gedenkt des Mannes."
```

Other sentence constituents, e.g., adverbs, prepositional phrases, etc., should be added via addModifier:

(17)
```
s = factory.createSentence("die Frau", "gehen");
pp = factory.createPrepositionalPhrase("auf", "der Berg");
s.addModifier(pp);
// "Die Frau geht auf den Berg."
```

Finally, a sentence does not actually require an explicit subject. If no subject is given, the expletive *es* is used automatically:

(18)
```
s = factory.createSentence("sein");
s.addModifier("schön");
// "Es ist schön."
```

2.5.1 Separable verbs

Many German verbs have prefixes which are separated from the base verb in inflected forms, for example:

(19) *ausgehen → ich gehe aus*

These verbs are handled similarly to compound nouns: the prefix has to be separated from the base verb by a vertical bar (|). Only the base verb is required to be in the lexicon. Therefore, if the lexicon contains an entry for *gehen*, the verbs *ausgehen, hinaufgehen, hereingehen,* etc. can all be realised correctly.

```
// produces "Die Frau geht aus.":
SPhraseSpec s = factory.createSentence("die Frau", "aus|gehen");
// produces "Die Frau will ausgehen.":
s.addModal("wollen");
```

Listing 9: Separable verbs

Note that the vertical bar always has to be explicitly specified for this to work; one reason for this is that some verbs—e.g., *umgehen*—are ambiguous in this regard, depending on their meaning:

(20) *Wie geht er damit um?* (separable)

(21) *Wie umgehen wir das Problem?* (non-separable)

2.5.2 Tense and mood

Most grammatical features of a sentence can be set with a simple function call. Tense is set with setTense, while the perfect is controlled with the boolean switch setPerfect.

```
SPhraseSpec s = factory.createSentence("die Frau", "gehen");

// produces "Die Frau ging.":
s.setTense(Tense.PAST);
// produces "Die Frau war gegangen.":
s.setPerfect(true);
// produces "Die Frau wird gehen.":
s.setTense(Tense.FUTURE);
s.setPerfect(false);
```

Listing 10: Tense options for a sentence

For the use of perfect tense in combination with modal verbs, please also see Sec. 4.2.

Setting mood works similarly using setForm. For subjunctive mood, using Form.SUBJUNCTIVE produces the *Konjunktiv I* when combined with present or future tense, and *Konjunktiv II* in the past tense. The option Form.SUBJUNCTIVE_II can be used to enforce *Konjunktiv II* in the future tense. Listing 11 exemplifies this behaviour.

```
SPhraseSpec s = factory.createSentence("der Mann", "sehen");

// produces "Der Mann sehe.":
s.setForm(Form.SUBJUNCTIVE);
// produces "Der Mann sähe.":
s.setTense(Tense.PAST);
// produces "Der Mann werde sehen.":
s.setTense(Tense.FUTURE);
// produces "Der Mann würde sehen.":
s.setForm(Form.SUBJUNCTIVE_II);
```

Listing 11: Subjunctive mood

Imperative sentences can be created similarly by setting form to Form.IMPERATIVE; the imperative verb form takes number into account. However, as German also commonly uses a honorific imperative form (*Sehen Sie!* vs. the simple *Sieh!*), the helper function setImperativeForm can be used instead which takes a number argument and a boolean indicating the use of the honorific form (cf. Listing 12).

```
SPhraseSpec s = factory.createSentence("sehen");

// produces "Sieh!":
s.setForm(Form.IMPERATIVE);
```

```
// produces "Seht!":
s.setImperativeForm(NumberAgr.PLURAL, false);
// produces "Sehen Sie!":
s.setImperativeForm(NumberAgr.PLURAL, true);
```

Listing 12: Subjunctive mood

2.5.3 Passive

Passive constructions are created with the boolean switch setPassive. By default, the subject of the active sentence is realised as a complement in the passive sentence, as Listing 13 demonstrates:

```
// produces "Die Frau gibt dem Mann einen Apfel.":
s = factory.createSentence("die Frau", "geben", "ein Apfel");
s.addIndirectObject(factory.createNounPhrase("der Mann"));

// produces "Ein Apfel wird dem Mann von der Frau gegeben.":
s.setPassive(true);
```

Listing 13: A passive construction

Realisation of the passive complement (i.e., *von der Frau* in the example above) can be turned off:

(22) s.setPassiveComplementRealisation(false);

Alternatively, the passive complement can also be realised at a different position within the sentence by calling setPassiveComplementPosition. Please refer to the section about constituent ordering (Sec. 3) for further details.

3 Ordering of constituents

German allows for a relatively flexible ordering of sentence constituents. For instance, in a main clause, all non-verb constituents can be realised at the beginning:

(23) Die Frau gab dem Mann gestern ein Buch.

(24) Dem Mann gab die Frau gestern ein Buch.

(25) Ein Buch gab gestern die Frau dem Mann.

(26) Gestern gab die Frau dem Mann ein Buch.

To enable the realisation of all these variants (and more, as the list isn't exhaustive), SimpleNLG for German uses a two-level system based on **complement order** and **modifier position**.

Complement order defines the order of sentence complements in terms of subject (S), direct object (O), and indirect object (I).[4] For example, sentences (23) and (26) would be classified as SIO (subject before indirect object before direct object), while (25) is classified as OSI. The ordering of complements can be set at sentence level with the setWordOrder function:

(27) `s.setWordOrder(WordOrder.OSI);`

The default word order is always SIO unless it is explicitly overridden via setWordOrder.

Modifiers, on the other hand, can be given a position value when they are added to a sentence which determines their exact placement. Position is usually specified relative to one of the complements: e.g., before the subject, or after the indirect object. To realise Example (23), the adverb *gestern* could be added in one of the following ways:

(28) `s.addModifier(Position.POST_INDIRECT_OBJECT, "gestern");`

(29) `s.addModifier(Position.PRE_OBJECT, "gestern");`

This way, modifier placement is independent of complement ordering. If a modifier is added to a sentence and the order of complements is later changed (via setWordOrder), the modifier is also moved to fulfill its placement specification.

Alternatively, two "absolute" positions for modifiers exist: Position.FRONT causes the modifier to be realised as the first element of a sentence regardless of complement order, while Position.DEFAULT (which is also used if no position is explicitly given) causes it to be realised last after all other complements. As an example, the following two statements could both be used to realise Example (26):

(30) `s.addModifier(Position.PRE_SUBJECT, "gestern");`

(31) `s.addModifier(Position.FRONT, "gestern");`

Positions can be thought of as "slots" which are filled with the appropriate constituents at the time of realisation. Elements in the FRONT position, if any, are always realised first. Assuming the default word order of SIO, elements in PRE_SUBJECT position will be realised next, followed by the subject, followed by any elements in POST_SUBJECT position. Afterwards, the same is done for the indirect object and direct object, and finally, any elements in DEFAULT position are realised.

Modifiers which share the same position attribute are realised in the order in which they were added to the sentence.

[4] In the rare case of genitive objects, these are classified as direct objects for this purpose.

Passive complements (cf. Sec. 2.5.3) are positioned the same way as modifiers; however, as they are added automatically by the system (rather than manually by the user), a separate function must be called to specify their position:

(32) `s.setPassiveComplementPosition(Position.FRONT);`

While the system described here can realise most constituent orderings found in German, a notable exception is verb cluster fronting (as in *Gesehen hatte er mich nicht.*), which is currently not supported by SimpleNLG for German.

4 Additional features

4.1 Pronouns

There are two distinct ways to generate pronouns: calling a function with the required type of pronoun and agreement features; or creating a reference to a noun phrase.

The first option is restricted to simple personal, possessive, or reflexive pronouns. Personal pronouns can be created with helper functions of the NLGFactory class:

(33)
```
NPPhraseSpec pro = factory.createPersonalPronounPhrase(Person.FIRST, NumberAgr.SINGULAR);
SPhraseSpec s = factory.createSentence(pro, "singen"); // "Ich singe."
```

The above example could actually be realised even simpler:

(34) `SPhraseSpec s = factory.createSentence("ich", "singen"); // "Ich singe."`

Pronouns supplied to factory functions as strings will be recognised as such. However, many pronouns in German can be ambiguous (e.g., *sie*). In these cases, the detailed specification of person/number (and possibly gender) is necessary.

Possessive and reflexive pronouns are typically instantiated with functions of the Constants class:

(35) `Constants.getPossessivePronoun(Person.SECOND, NumberAgr.SINGULAR); //"dein"`

(36) `Constants.getReflexivePronoun(Person.THIRD); //"sich"`

The returned pronouns can then be used, e.g., as specifiers of a noun phrase:

(37)
```
pro = Constants.getPossessivePronoun(Person.FIRST, NumberAgr.SINGULAR); //"mein"
factory.createNounPhrase(pro, "Frau"); // "meine Frau"
```

Further pronouns can be accessed directly from strings with the getPronoun method:

(38) `Constants.getPronoun("derjenige");`

(39) `Constants.getPronoun("jemand");`

(40) `Constants.getPronoun("mancher");`

Alternatively, they can be directly cast to a noun phrase via factory methods:

(41) `factory.createPronounPhrase("derjenige");`

(42) `factory.createPronounPhrase("jemand");`

(43) `factory.createPronounPhrase("mancher");`

The difference between these two variants is that `getPronoun` returns a lexical item, while `createPronounPhrase` uses that lexical item to create a noun phrase. The latter variant can be used if the pronoun will be used as a standalone constituent (e.g., *Ich sehe denjenigen.*), while the former variant must be used if the pronoun is embedded in another NP as specifier (e.g., *Ich sehe denjenigen Mann.*).

While inflectional variants of the pronouns are recognized (i.e., *derjenige* and *diejenige* should both return the same object), the returned object will always default to masculine singular form. If this is not desired, agreement features must be changed manually with `setGender` and `setNumber`. A more elegant way is to create references to already-instantiated NPs, described below.

4.1.1 Referring to other noun phrases

The `NLGFactory` class provides a method for creating a noun phrase that is simply a reference to another noun phrase. The new noun phrase object will copy all agreement features of the one that is referred, and will default to a pronominal realisation.

(44)
```
np1 = factory.createNounPhrase("der", "Mann"); //"der Mann"
np2 = factory.createReferentialNounPhrase(np1); //"er"
```

References are dynamic; if the referenced object changes, the referential NP will reflect that change. In the example above, `np2` is created as a reference to `np1` (*der Mann*) and would be realised as the pronoun *er*. However, if `np1` is later set to plural form (resulting in *die Männer*), `np2` would be realised as the plural *sie*.

Referential NPs are not restricted to the personal pronoun, but can use any pronoun that is given as a second parameter:

(45) `np2 = factory.createReferentialNounPhrase(np1, "jener"); // "jener"`

(46)
```
np2 = factory.createReferentialNounPhrase(np1, "derjenige");
pp1 = factory.createPrepositionalPhrase("für", np2); // "für denjenigen"
```

4.2 Modal verbs

Adding a modal verb to a sentence is achieved by calling addModal, giving the modal verb as a string. Recognized modal verbs are *dürfen, können, mögen, müssen, sollen,* and *wollen*. Multiple modal verbs can be added to a sentence by calling the function more than once. To override the modal instead, use setModal. Calling clearModals removes all modal verbs from a sentence. Listing 14 gives an example.

```
SPhraseSpec s = factory.createSentence("der Mann", "sehen");

// produces "Der Mann kann sehen.":
s.addModal("können");
// produces "Der Mann muss sehen können.":
s.addModal("müssen");
// produces "Der Mann will sehen.":
s.setModal("wollen");
// produces "Der Mann sieht.":
s.clearModals();
```

Listing 14: Modal verbs

If a sentence contains modal verbs, any options concerning tense (including the perfect setting) are applied to the finite verb of the sentence. For example, assuming the sentence *Der Mann kann sehen*, setting the tense to Tense.PAST will result in *Der Mann konnte sehen*.

However, it is possible in German for the perfect to be applied separately to the main verb and/or the modal verb. To illustrate this, consider the following two sentences:

(47) *Sie hat es tun können.*

(48) *Sie kann es getan haben.*

Example (47) is the result of simply setting the perfective aspect (cf. Sec. 2.5.2) for the sentence *Sie kann es tun*:

(49) `s.setPerfect(true); // "Sie hat es tun können."`

To achieve the result in Example (48), an additional switch is introduced which sets the perfective aspect separately for the main verb:

(50) `s.setMainVerbPerfect(true); // "Sie kann es getan haben."`

A combination of both options is theoretically possible, but probably rarely desired.

4.3 Coordinate clauses

Support for coordination is very limited at this moment. In principle, all types of phrases can be coordinated by calling their coordinate function:

(51)
```
NPPhraseSpec mann = factory.createNounPhrase("der Mann");
NPPhraseSpec frau = factory.createNounPhrase("die Frau");
NPPhraseSpec coord = mann.coordinate(frau); //"der Mann und die Frau"
```

The coordinating element, *und* in the example above, can be changed be calling setConjunction. However, this requires an explicit cast to a coordinating phrase specification:

(52)
```
CoordinateNPPhraseSpec coord = (CoordinateNPPhraseSpec) mann.coordinate(frau);
coord.setConjunction("oder"); // "der Mann oder die Frau"
```

Coordinating more than two phrases is also possible. Note that calling coordinate with several arguments produces different results than calling it several times in a row:

(53)
```
NPPhraseSpec kind = factory.createNounPhrase("das Kind");
NPPhraseSpec coord = mann.coordinate(frau, kind);
// produces "der Mann, die Frau und das Kind"
```

(54)
```
NPPhraseSpec coord = mann.coordinate(frau).coordinate(kind);
// produces "der Mann und die Frau und das Kind"
```

The procedure is analogous for other phrase types. It should work fine as long as the phrases to be coordinated are relatively simple in structure. Coordination of structurally complex phrases was not really tested so far and might produce strange or even ungrammatical results.

Note that it is also possible to coordinate sentences this way. This is not only useful for chaining sentences with *und*, but also the required way to combine sentences with coordinating conjunctions such as *aber* or *denn*. Assuming two sentences *s1 = Der Mann geht schlafen* and *s2 = Er ist müde*, a coordination could be constructed in the following way:

(55)
```
CoordinateSPhraseSpec coord = (CoordinateSPhraseSpec) s1.coordinate(s2);
coord.setConjunction("denn");
coord.setEnforceComma(true);
// produces "Der Mann geht schlafen, denn er ist  müde."
```

The call to setEnforceComma is necessary because at this time, a coordination of only two phrases never introduces a comma, which is required with this type of coordinating construction, though.

4.4 Subordinate clauses

Although many variants are already supported, grammatical coverage of subordinate constructions is likely to be incomplete so far. In particular, infinitive subordinate clauses involving *zu + infinitive* cannot be generated so far.

4.4.1 Finite subordinate clauses

Finite subordinate clauses can be added to a sentence by calling addSubordinate with the desired complementiser and clause:

(56)
```
s1 = factory.createSentence("der Mann", "sehen");
s2 = factory.createSentence("regnen");
s1.addSubordinate("dass", s2); // "Der Mann sieht, dass es regnet."
```

Optionally, a position argument (cf. Sec. 3) can be given to specify placement of the subordinate within the sentence:

(57)
```
s1.addSubordinate(Position.FRONT, "dass", s2); // "Dass es regnet, sieht der Mann."
```

With the exception of specifying the position argument, subordinate clauses can also be attached to elements other than sentences:

(58)
```
np1 = factory.createNounPhrase("die Tatsache");
np1.addSubordinate("dass", s2); // "die Tatsache, dass es regnet"
```

For coordinating conjunctions such as *denn*, which do not trigger verb-final word order in the embedded clause (e.g., compare *dass er ihn sieht* and *denn er sieht ihn*), the coordination mechanism must be used instead (cf. Sec. 4.3).

4.4.2 Infinitive complement clauses

There is basic support for using sentences as infinitive complements:

(59)
```
s1.addComplement(s2); //"Der Mann sieht es regnen."
```

However, other constructions involving *zu + infinitive* (e.g., *Er kam, um zu helfen* or *Ich empfehle dir, zu bleiben*) cannot be realised at the moment.

4.4.3 Relative clauses

Relative clauses are constructed by attaching a sentence object to a noun phrase via the addRelativeClause function. The second argument of the function specifies the syntactic role which the relative pronoun should take in the embedded sentence.

(60)
```
np = factory.createNounPhrase("der Mann");
s = factory.createSentence("die Frau", "sehen");
np.addRelativeClause(s, DiscourseFunction.OBJECT);
// produces "der Mann, den die Frau sieht"
```

In the above example, a relative clause it added to the NP *der Mann*. The argument DiscourseFunction.OBJECT specifies that *der Mann* should take the role of the direct object in the embedded relative clause *die Frau sieht*. Similar examples can be constructed for other roles:

(61)
```
np = factory.createNounPhrase("der Mann");
s = factory.createSentence("sehen");
np.addRelativeClause(s, DiscourseFunction.SUBJECT);
// produces "der Mann, der sieht"
```

If the embedded sentence already contains elements in the position which should be filled by the relative pronoun, they are deleted. Therefore, if Example (60) had specified DiscourseFunction.SUBJECT instead, the result would have been the same as in Example (61), as the newly introduced relative pronoun overwrites the already existing subject *die Frau*.

Relative pronouns can also be embedded in prepositional phrases. This can be achieved by giving a preposition object as the second argument to addRelativeClause, e.g.:[5]

(62)
```
np = factory.createNounPhrase("das Haus");
s = factory.createSentence("der Mann", "gehen");
np.addRelativeClause(s, new Preposition("in", Case.ACCUSATIVE));
// produces "das Haus, in das der Mann geht"
```

An optional third argument can be used to replace the relative pronoun with a different one, typically *welcher*:

(63)
```
np = factory.createNounPhrase("der Mann");
s = factory.createSentence("die Frau", "sehen");
np.addRelativeClause(s, DiscourseFunction.OBJECT, Constants.PRO_WELCHER);
// produces "der Mann, welchen die Frau sieht"
```

Finally, the relative pronoun can also take the form of a possessive pronoun within a noun phrase in the relative clause:

[5] Note that at the moment, the preposition object has to be instantiated manually for this to work.

(64)
```
np1 = factory.createNounPhrase("der Mann");
np2 = factory.createNounPhrase("Frau");
sub = factory.createSentence("schwanger sein");
np1.addAttributiveRelativeClause(sub, np2);
// produces "der Mann, dessen Frau schwanger ist"
```

Note that in the above example, the noun phrase *Frau* which should embed the relative pronoun has to be specified separately from the rest of the relative clause (here, *schwanger sein*). Support for this kind of construction is still limited; in particular, the NP constructed this way (here, *dessen Frau*) will always take the role of the subject in the embedded clause at this moment.

Extraposition of relative clauses In principle, it is possible to create extraposed relative clauses; however, the procedure to do this a little bit hacky at the moment. The following lines give an example:

(65)
```
mann = factory.createNounPhrase("der Mann");
s1 = factory.createSentence(mann, "schön sein");
s2 = factory.createSentence("die Frau", "sehen");
s2.setComplementiser(new NPPhraseSpec(mann, Constants.PRO_REL), DiscourseFunction.OBJECT);
s1.addAttributivePostmodifier(s2);
// "Der Mann ist schön, den die Frau sieht ."
```

What happens here is that the relative clause is basically added at the sentence level (via addAttributivePostmodifier, which is not typically used in this German version of SimpleNLG), and the relative pronoun is constructed and inserted manually.

More convenient functions to achieve the same effect should be provided in future versions of this software.

4.5 Questions

Declarative sentences can be turned into questions by calling setInterrogative. One way to use this function is via the enum class InterrogativeType, which is the standard way to generate choice (or "yes/no") questions:

(66)
```
s = factory.createSentence("regnen");
s. setInterrogative (InterrogativeType.JA_NEIN); //"Regnet es?"
```

Further constants are available for *warum, was, wer, wie,* and *wo* questions, e.g.:

(67)
```
s. setInterrogative (InterrogativeType.WARUM); //"Warum regnet es?"
```

(68)
```
s. setInterrogative (InterrogativeType.WO);    // "Wo regnet es?"
```

Alternatively, the function can also be given a string argument:

(69)
```
s. setInterrogative ("weshalb"); // "Weshalb regnet es?"
```

For the interrogative *wer*, it is often useful to additionally supply a discourse function, which causes the interrogative pronoun to inflect for the proper case:

(70)
```
s = factory.createSentence("der Mann", "sehen");
s. setInterrogative (InterrogativeType.WER, DiscourseFunction.OBJECT); //"Wen sieht der Mann?"
```

Complex noun phrases can be supplied as interrogatives, which can be useful if the actual interrogative pronoun must be embedded:

(71)
```
frau = factory.createNounPhrase("Frau");
frau. setSpecifier (Constants.getPronoun("welcher"));
s. setInterrogative (frau, DiscourseFunction.OBJECT); //"Welche Frau sieht der Mann?"
```

Finally, if the interrogative sentence needs verb-final word order, the function setEchoInterrogative can be used instead:

(72)
```
s = factory.createSentence("regnen");
s.addModifier("wohl");
s.setEchoInterrogative("ob"); // "Ob es wohl regnet?"
```

Note that the word order within an interrogative sentence can be controlled the same way as for declarative sentences (cf. Sec. 3). However, the interrogative element is always assigned the front position. There is no elegant way to change this behaviour at the moment.[6]

4.6 Negation

There is no "intelligent" support for negation at this time. In principle, all types of phrases can be negated by calling setNegated(true). However, all this does is insert the word *nicht* at a fixed position, which may be sufficient in some cases, but often is inadequate: e.g., the system will currently generate **es ist schön nicht* instead of *es ist nicht schön*, and **nicht ein Mann* instead of the usually desired *kein Mann*.

Predicting the correct (or most natural) way of negating a phrase or placing the particle *nicht* in a sentence is not trivial in German. It is recommended to always negate a sentence by manually inserting the particle *nicht* at a desired position (cf. Sec. 3) within that sentence.

[6] In principle, the interrogative element can be instantiated manually and then inserted in the sentence at the desired position, but this is rather hacky.

4.7 Canned text in verb phrases

This section describes a useful trick when dealing with idiomatic expressions or other verb phrases containing some kind of canned text.

Similarly to prefixes, the placement of further words given before the verb (and separated from it with a space) is also determined dynamically based on whether the verb is inflected or not. This feature can be used to specify complex verb phrases, e.g., *Fahrrad fahren*, *Gassi gehen*, or even *auf die Nerven gehen*. Everything before the verb is treated as canned text, i.e., it is only copied verbatim and never looked up in the lexicon, changed for inflection, etc., but it is always placed in a grammatically correct position in the sentence.

```
SPhraseSpec s = factory.createSentence("die Frau", "auf die Nerven gehen");
// produces "Die Frau geht dem Mann auf die Nerven.":
s.addIndirectObject(factory.createNounPhrase("der Mann"));
// produces "Die Frau will dem Mann auf die Nerven gehen.":
s.addModal("wollen");
```

Listing 15: Conveniently setting complex verb phrases

Listing 16 shows an alternative method of creating the same sentence as in Listing 15. There is usually more than one way to build any given sentence: for idiomatic constructions or fixed strings, the first method is simply more convenient and requires less code, while the method in Listing 16 might be more appropriate if more control over each constituent is required.

```
// produces "Die Frau geht dem Mann auf die Nerven.":
SPhraseSpec s = factory.createSentence("die Frau", "gehen");
NPPhraseSpec nerv = factory.createNounPhrase("der Nerv");
nerv.setPlural(true);
PPPhraseSpec pp = factory.createPrepositionalPhrase("auf", nerv);
s.addIndirectObject(factory.createNounPhrase("der Mann"));
s.addModifier(pp);
```

Listing 16: Alternative way to build the same sentence

5 Questions?

I am aware that this guide is neither comprehensive nor ideally structured. I will try to make further improvements both to this guide and to the software itself if time allows. Meanwhile, if you do have any questions about the usage of this package, do not hesitate to contact me (`bollmann@linguistics.rub.de`). As long as I don't get flooded with e-mails, I will do my best to answer every question regarding the software.