AGILE

Automatic Generation of Instructions in Languages of Eastern Europe

---

Title ***Lexical and morphological resources for the final prototype***

Authors Hana Skoumalová
Ivana Kruijff-Korbayová
Jirka Hana
Michael Malkovsky
Mikhail Boldasov
Danail Dochev

Deliverable *LEXN-2-Bu, LEXN-2-Cz, LEXN-2-Ru*
Status *Final*
Availability *Public*
Date *July 1999*

INCO COPERNICUS PL961104

Abstract:

This report constitutes the second deliverable in a sequel within the Task 4.2 (Lexical and morphological resources for final prototype). It follows upon the deliverable LSPEC2 (Modelling Lexical Resources in KPML for Generating Instructions in Slavic Languages).

In the present report, we describe the final format of lexical entries we decided to use for the lexica of Bulgarian, Czech and Russian to be employed in the final prototype of our system. We also describe in detail the lexical features that we use for various classes of lexical entries.

The lexica for the three languages we have developed up to this point extend the lexica developed for the purpose of the initial demonstrator. The current coverage corresponds to the corpus of instructional texts assembled in the work package WP3, and reported on in the deliverables CORP-Bu, CORP-Cz, CORP-Ru and CORP.

The results reported in this deliverable, together with those reported in the LSPEC deliverable, serve as the input for Tasks 7.2 ad 7.3 (Implementation of generators for intermediate and final prototypes).

More information on AGILE is available on the project web page and from the project coordinators:

URL:	http://www.itri.brighton.ac.uk/projects/agile
email:	agile-coord@itri.bton.ac.uk
telephone:	+44-1273-642900
f	+44-1273-642908

Table of Content

Table of Figures

1. Introduction

This report constitutes the second deliverable in a sequel within the Task 4.2 (Lexical and morphological resources for final prototype). It follows upon the deliverable LSPEC2 (Modelling Lexical Resources in KPML for Generating Instructions in Slavic Languages), where we discussed the issues of modelling lexical resources for Slavic languages. In particular, it addresses how we can indicate what complementations a given lexical item can take and in what form(s) they may or need to be realized.

In the present report, we describe the final format of lexical entries we decided to use for the lexica of Bulgarian, Czech and Russian to be employed in the final prototype of our system. We also describe the lexical features that we use for various classes of lexical entries.

As we discussed in the preceding LSPEC2 deliverable, the main purposes of the lexical features are as follows:

- to determine the linking of a lexical entry to a concept in the domain model
- to satisfy constraints imposed by the grammar
- to capture the obligatory or optional complementations of the given lexical entry and if necessary, to restrict their grammatical realizations.

The lexica for the three languages we have developed up to this point extend the lexica developed for the purpose of the initial demonstrator. The coverage of the current lexica corresponds to the corpus of instructional texts from the AutoCAD manual, we assembled in work package WP3, and reported on in the deliverables CORP-Bu, CORP-Cz, CORP-Ru and CORP. We also added some other lexical entries, which we needed for testing various grammatical phenomena.

The results reported in this deliverable, together with those reported in deliverable LSPEC, serve as the input for Tasks 7.2 and 7.3 (Implementation of generators for intermediate and final prototypes).

The rest of this report is organized as follows: in the second part we describe the final format of a lexical entry in general and its linking to domain model concepts; in the third part we define lexical features needed by different word classes in all three languages; in the fourth part of the report we describe the external morphological modules for all languages and how they cooperate with KPML.

2. Format of a lexical entry in general

In this section we provide a general description of the format of lexical entries in the lexica for Bulgarian, Czech and Russian as used for the purposes of generation in AGILE. We explain the meaning of each part of a lexical entry and briefly summarize the reasons why particular parts are included, and how the lexical features specified there are exploited in the process of generation.

We concentrate mainly on the following issues:

- the linking of a lexical entry to a concept in the domain model
- the satisfying of constraints imposed by the grammar

- the encoding of the obligatory or optional complementations of the given lexical entry and the restrictions on their grammatical realizations when needed.

2.1 The linking of lexical entries to concepts in the domain model

In the domain model, concepts for CAD-CAM objects and processes are defined. These concepts can then be used in SPLs to describe words of the generated sentence. The concepts used in SPLs can either be more general concepts from the upper model or narrower concepts from the CAD-CAM model. In the former case, the slot `:LEX` with the desired lexical entry as its value must be present; in the latter case we can provide a mapping between CAD-CAM model concepts and corresponding lexical entries.

Example: two SPLs for generating the Czech sentence *Nakreslete oblouk. (Draw an arc.)*.

```
(EXAMPLE
   :NAME   TEXT2B-0
   :TARGETFORM   "Nakreslete oblouk"
   :LOGICALFORM
   (S / DISPOSITIVE-MATERIAL-ACTION
        :LEX NAKRESLIT
        :SPEECHACT IMPERATIVE
        (P / OBJECT
             :LEX OBLOUK
             :IDENTIFIABILITY-Q NOTIDENTIFIABLE))
)
(EXAMPLE
   :NAME   TEXT2B-0
   :TARGETFORM   "Nakreslete oblouk"
   :LOGICALFORM
   (S / DM::DRAW
        :SPEECHACT IMPERATIVE
        (P / DM::ARC
             :IDENTIFIABILITY-Q NOTIDENTIFIABLE))
)
```

In the first case, we used upper model concepts (`DISPOSITIVE-MATERIAL-ACTION` and `OBJECT`) and explicit choice of lexical entries (`:LEX NAKRESLIT` and `:LEX OBLOUK`), in the second case we used CAD-CAM model concepts `DRAW` and `ARC`.

One concept can be mapped onto several lexical entries which differ in certain features (e.g. perfective and imperfective verb that form together an aspectual pair). The choice is then constrained by features which are imposed by the grammar.

Example:

```
(annotate-concept DM::DRAW :lex-items (kreslit nakreslit))
```

The concept `DRAW` is mapped onto two lexical entries *kreslit* and *nakreslit*. The lexical entries look like this:

```
(LEXICAL-ITEM
     :NAME kreslit
     :SPELLING "kreslit"
     :SAMPLE-SENTENCE "Honza rád kreslí."
```

```
     :FEATURES (VERB IMPERFECTIVE DO-VERB EFFECTIVE-VERB
CREATION-VERB))

(LEXICAL-ITEM
     :NAME nakreslit
     :SPELLING "nakreslit"
     :SAMPLE-SENTENCE "Honza mi nakreslí lva."
     :FEATURES (VERB PERFECTIVE DO-VERB EFFECTIVE-VERB
CREATION-VERB))
```

If the grammar selects imperfective aspect, the first lexical entry is chosen; if the grammar selects perfective aspect, the second entry is chosen. This mechanism is used also for the choice between a verb and its nominalization, among other purposes.

Beside those simple concepts that can be mapped onto one lexical entry, there are also complex concepts, that represent a more sophisticated object in the CAD-CAM world. The concept DIALOGBOX can serve as an example. In some languages (like German), where word compounding is very productive, this concept can be mapped onto one lexical entry. But in other languages this concept should be mapped onto a nominal group consisting of an attribute (attribute noun in English, or adjective in Slavic languages) and a noun. Instead of a simple mapping function we need something that converts the DM concept to an SPL:

```
DM::DIALOGBOX      <—>  (P / OBJECT
                           :LEX OKNO
                           :PROPERTY-ASCRIPTION
                           (Q / QUALITY
                                :LEX DIALOGOVY2))
```

In the current version of the lexico-grammar we only use mapping onto single lexical entries. A function for converting more complex concepts will be created in work package WP7.

2.2 Constraints imposed by the grammar

The grammar network exploits three realization operations that interact with the lexicon. They are CLASSIFY, OUTCLASSIFY and LEXIFY. The choice of the lexical item is driven by three corresponding realization statements. In this section, we will discuss the first two operations, CLASSIFY and OUTCLASSIFY. They serve for setting up an association between a grammatical constituent and features drawn from the lexicon. For example, if a system of the grammar uses the realization statement (CLASSIFY FINITE DO-VERB), then the constituent FINITE can be lexically realized only by a word that has the feature DO-VERB among its features. The statement OUTCLASSIFY works in the opposite way — the feature must not be found among the lexical features.

If a lexical entry does not contain all necessary features, the entry is not chosen, even if the :LEX slot in the SPL is filled. On the other hand, if SPL contains neither :LEX slots nor DM concepts, the first lexical entry that satisfies the CLASSIFY and OUTCLASSIFY statements is chosen.

In the original English system, there were many features that make sense only in the English grammar. Such features as BAREINFINITIVECOMP do not make sense in Slavic languages. On the other hand, some features necessary for the Slavic languages are missing. The discussion about lexical features follows in the next subsection.

2.3 Special requirements for inflectional languages

In Systemic Functional Grammar [Halliday, 1985], the generation of sentences is driven by the semantics. The process of generation leads (usually) to a verb that is characterized by its semantic arguments. This list of semantic arguments roughly corresponds to θ-roles or inner participants. However, the presupposition, derived from English, that every argument can be realized just by one surface realization does not hold for highly inflectional languages like Czech or Russian. It is a well-known fact that different verbs require different surface realization for the same participants. For example the verbs *pověřit* and *oznámit* have three participants — Actor, Patient (Goal in SFG), and Addressee (Beneficiary). Their valency frames are, however different. The frame of the former (in the active voice) is <NP_{Nom}, NP_{Ins}, NP_{Acc}>, the frame of the latter is <NP_{Nom}, NP_{Acc}, NP_{Dat}>. As was proposed in a previous report [Kruijff et al., 1998], we add the information about the valency frame to the `:FEATURES` slot of the description of a lexical item.

The question was, whether it should be the semantic arguments, or rather the sentential complements, which are connected with the surface cases. We decided to bind the surface cases with the complements. We use the feature `TRANSITIVE` for verbs whose Actee (Goal or Medium) is realized as a nominal group in Accusative and this Actee is the direct complement of the verb (i.e. Actee becomes the subject of the passive sentence). The feature `INTRANSITIVE` is used for intransitive verbs, i.e. verbs with one participant. Other verbs, with Agent and Actee realized by a case other than Accusative, have the feature `NON-TRANSITIVE`. For describing complements that are not in Accusative we use features like `INS-OTHERCOMPLEMENT` for a nominal group in instrumental, or `NA-ACC-OTHERCOMPLEMENT` for a prepositional group with the preposition *na* and the nominal group in Accusative.

Other features that had to be added for the Slavic languages are features for verb aspect (`PERFECTIVE` and `IMPERFECTIVE` — these features are used also for deverbative nouns and adjectives), features for morphological cases (in Czech and Russian), and gender.

3. Definitions of lexical entries

In this section we provide examples of lexical entries for each lexical category and a discussion of the lexical features employed.

3.1 Bulgarian lexicon

The specification of the lexical entries for the Bulgarian lexicon follows the conventions for the AGILE Initial Demonstrator lexicons [Skoumalová et al.,1998]. It uses the slots common to the Czech and Russian lexicons: name, spelling, sample-sentence (optional), features, properties (optional), comments (optional), editor (optional):

- The slot `:NAME` contains an identifier of a lexical item (in Latin characters). This value is referred to by SPL `:LEX` keywords and `LEXIFY` statements in the grammar.
- The slot `:SPELLING` contains a string of Cyrillic letters presenting the base form of the lexical item.
- The slot `:SAMPLE-SENTENCE` contains an example — a Bulgarian phrase illustrating the possible use of the lexical item.

- The slot :FEATURES contains the lexico-grammatical features of the lexical entry used by the Bulgarian grammar.
- The slot :PROPERTIES contains morphological information; currently it is not used in the Bulgarian lexicon.
- The slot :COMMENTS currently is not used in the Bulgarian lexicon.
- The slot :EDITOR may contain the name of the person who included the item in the lexicon and the date of inclusion.

The Bulgarian lexical resources are organized in two lexicon files:

bulgarian_words.lexicon, containing the majority of word forms for the Intermediate Demonstrator;

Cad-Cam.lexicon containing special domain-specific terms as well as constant strings for object names, generally in English.

Examples of lexical entries in Bulgarian lexicons:

```
(LEXICAL-ITEM
    :NAME   IME
    :SPELLING   "име"
    :FEATURES   (OUTCLASSIFY-PROPERNOUN NOUN COMMON-NOUN
                   COUNTABLE))
(LEXICAL-ITEM
    :NAME   SAZDAVANE
    :SPELLING   "създаване"
    :FEATURES   (NONSUBSTITUTE COUNTABLE NOMINALIZATION-
                   NOUN COMMON-NOUN NOUN))
(LEXICAL-ITEM
    :NAME   IZBERA
    :SPELLING   "избера"
    :FEATURES   (DO-VERB EFFECTIVE-VERB DISPOSAL-VERB
                   PERFECTIVE-VERB VERB))
(LEXICAL-ITEM
    :NAME   ILI
    :SPELLING   "или"
    :SAMPLE-SENTENCE   "Бяло или червено - White or red?"
    :FEATURES   (SENTENCECONJUNCTION ADDITIVE CONJUNCT NOT-
                   PUNCTUATION NOT-SUBORDINATOR LINKER))
(LEXICAL-ITEM
   :NAME   DIALOGOV
   :SPELLING   "диалогов"
   :FEATURES  (NOT-PREDICATEONLY NONE-OF-SIZE-PROVENANCE-
                   MATERIAL-COLOUR-AGE INTRINSIC NONDEGREE
                   ADJECTIVE))
LEXICAL-ITEM
    :NAME   V
    :SPELLING   "в"
    :FEATURES (NOT-OBJECTNOTREQUIRED NOT-PPOBJECT LOCATION-
                   VERB PREPOSITION))
```

The Bulgarian lexicon contains the basic word forms, while the necessary derivatives are generated by the morphological module, according to the following list:

Verbs

Basic form

Derivatives, specified by:

- Aspect (imperfective, perfective)
- Tense (present, simple past, future)
- Number (singular, plural)
- Person (1^{st}, 2^{nd}, 3^{rd})
- Voice (active, passive)
- Mood (indicative, imperative)

Nouns

Base form

Derivatives, specified by:

- Gender (masculine, feminine, neuter)
- Number (singular, plural)
- Article (for masculine - full and short form)

Adjectives

Base form

Derivatives, specified by:

- Gender (masculine, feminine, neuter)
- Number (singular, plural)
- Article (for masculine - full and short form)

Numerals

Base form – ordinal type

Derivatives, specified by:

- Gender (masculine, feminine, neuter)
- Number (singular, plural)

Pronouns

Types considered:

- Personal
- Demonstrative
- Possessive

3.2 Czech lexicon

The Czech lexicon contains all words that are needed for the Intermediate Prototype. As for lexical features, we choose a minimalist approach — we use only features that were found as necessary for generating sentences in the Intermediate Prototype, and we plan to add new features as they will be needed by grammar systems.

The lexicon is divided into two files. One, called `agcis.lexicon`, is the main lexicon of Czech words. The other, called `cadcam-gui.lexicon`, is a lexicon of CAD-CAM labels that are used in the user interface. These labels can be translated to Czech or they can be left in English. As this lexicon is kept separately, it is easy to replace the English labels with Czech ones.

3.2.1 The structure of the lexical entries

- The slot `:NAME` contains the name of the entry in the form that is used in `:LEX` slots in SPLs and in `LEXIFY` command in the grammar. As this name should contain only ASCII characters, we decided to replace all accented characters by digraphs, where 2 represents an acute accent on previous character, 3 stands for caron and 4 for ring accent.
- The slot `:SPELLING` contains the base form of the word. In case of uninflected words, this is the form that occurs in the generated sentence. In case of inflected words, this form can also contain an index or a mark expected by the morphological module.
- The slot `:SAMPLE-SENTENCE` now contains an output from morphological analysis. This is a leftover of semi-automatic extraction of the lexicon from existing resources. In the next version of the lexicon, this slot will be filled with sample sentences.
- The slot `:FEATURES` contains the features that can be used by the grammar for constraining the choice of a lexical item. As mentioned above, we chose the minimalist approach of adding features to the lexicon. The current version of the lexicon uses the following features:
  - Noun: `COMMON-NOUN, PROPER-NAME, NOMINALIZATION-NOUN, COUNTABLE, PERFECTIVE, IMPERFECTIVE, M-ANIMATE, M-INANIMATE, FEMININE, NEUTER.`
  - Adjective: `PERFECTIVE, IMPERFECTIVE, PASSIVE, ACTIVE, NEGATIVE, ARABICPROP, ORDINAL.`
  - Pronoun: `PERSONAL, POSSESIVE, INTERROGATIVE, RELATIVE, DEMONSTRATIVE, POSSESIVE-RELATIVE, NEGATIVE, TOTAL, REFLEXIVE, FIRSTPERSON, SECONDPERSON, THIRDPERSON, NEUTER.`
  - Numeral: `ARABICPROP, CARDINAL, TIMES, ROMAN.`
  - Verb: `DO-VERB, EFFECTIVE-VERB, DISPOSAL-VERB, MIDDLE-VERB, PERFECTIVE, IMPERFECTIVE, NOT-PASSIVE, TRANSITIVE, INTRANSITIVE, NON-TRANSITIVE, NA-ACC-OTHERCOMPLEMENT, INS-OTHERCOMPLEMENT, V-LOC-OTHERCOMPLEMENT, INF-OTHERCOMPLEMENT, S-INS-OTHERCOMPLEMENT, LOCATION, DIRECTION, MANNER.`
  - Copula: `COPULA, VERB, MIDDLE-VERB, RELATIONAL-VERB, RELATIONAL-BE-VERB.`
  - Auxiliary: `AUXILIARY, VERB.`
  - Adverb: `INTERROGATIVE, RELATIVE, DEMONSTRATIVE, LOCATIVE, TEMPORAL, MANNER, VARIABLE, UNIQUE, EXISTENCIAL.`

- Conjunction: `COORDINATIVE, SUBORDINATIVE, AUXILIARY`.

Every word class can have the feature `UNINFLECTED`, which instructs the morphological module, that it should just return the value of the slot `:SPELLING`.

We treat the verb *být* separately from other verbs and we even treat differently the copula and the auxiliary. The reason is, that the copula and the auxiliary have different morphological and syntactic behaviour.

- The slot `:PROPERTIES` is used only by numerals. They use it for connecting the meaning of numbers expressed by digits and by words. We were considering using this slot for prepositions as well, namely for connecting the meaning with the required case of the following noun group, but we decided to put this information into the grammar, following the design of Penman grammar.

Examples of lexical entries:

```
(LEXICAL-ITEM
   :NAME veliky2
   :SPELLING "veliký"
   :SAMPLE-SENTENCE "veliký+Adj^DB+Noun+Fem+Sg+NomAcc"
   :FEATURES (ADJECTIVE)
)
(LEXICAL-ITEM
   :NAME dany2
   :SPELLING "daný"
   :SAMPLE-SENTENCE "daný+Adj+Masc+Anim+Sg+GenAcc"
   :FEATURES (ADJECTIVE PASSIVE PERFECTIVE)
)
```

The word *veliký* (*great, big*) is a "plain" adjective, while the word *daný* (*given*) is derived from a passive participle of a perfective verb *dát*.

```
(LEXICAL-ITEM
   :NAME kde
   :SPELLING "kde"
   :SAMPLE-SENTENCE "kde+Adv"
   :FEATURES (ADVERB UNINFLECTED INTERROGATIVE
              RELATIVE LOCATIVE)
)
(LEXICAL-ITEM
   :NAME kdekoliv
   :SPELLING "kdekoliv"
   :SAMPLE-SENTENCE "kdekoliv+Adv"
   :FEATURES (ADVERB UNINFLECTED LOCATIVE VARIABLE)
)
```

The word *kde* (*where*) is a relative or interrogative adverb, and the word *kdekoliv* (*anywhere*) is an indefinite adverb.

```
(LEXICAL-ITEM
   :NAME a
   :SPELLING "a"
   :SAMPLE-SENTENCE "a+Conj"
   :FEATURES (CONJ UNINFLECTED COORDINATIVE)
)
(LEXICAL-ITEM
   :NAME co-3
   :SPELLING "co"
```

```
    :SAMPLE-SENTENCE "od té doby, co"
    :FEATURES (CONJ UNINFLECTED SUBORDINATIVE)
)
 (LEXICAL-ITEM
    :NAME aby
    :SPELLING "aby"
    :SAMPLE-SENTENCE "aby+Conj^SBbýt+Aux+Cond+2P+Pl+Clit"
    :FEATURES (CONJ AUXILIARY SUBORDINATIVE)
)
```

The conjuctions *aby* (*so that*) and *kdyby* (*if*) are "conjugated", as they are in fact contracted with the auxiliary in Conditional.

```
(LEXICAL-ITEM
    :NAME barva
    :SPELLING "barva"
    :SAMPLE-SENTENCE "barva+Noun+Fem+Pl+NomAccVoc"
    :FEATURES (NOUN COMMON-NOUN COUNTABLE FEMININE)
)
(LEXICAL-ITEM
    :NAME nakresleni2
    :SPELLING "nakreslení"
    :SAMPLE-SENTENCE "nakreslit+Verb+Imp+2P+Pl"
    :FEATURES (NOUN COMMON-NOUN NOMINALIZATION-NOUN
         COUNTABLE NEUTER PERFECTIVE)
)
(LEXICAL-ITEM
    :NAME second
    :SPELLING "druhý-1"
    :SAMPLE-SENTENCE "druhý-1+Adj+Masc+Anim+Sg+GenAcc"
   :FEATURES   (ADJECTIVE ARABICPROP ORDINAL)
   :PROPERTIES   ((ARABICPROP 2))
)
```

We treat the ordinal numeral numerals as adjectives, as they behave morphologically and syntactically as adjectives. The information (needed for word order) that they are in fact numerals can be obtained from the feature `ORDINAL`.

```
(LEXICAL-ITEM
    :NAME jeden
    :SPELLING "jeden`1"
    :SAMPLE-SENTENCE "jeden+Num+Card+Fem+Sg+Ins"
    :FEATURES   (NUMERAL ARABICPROP CARDINAL)
   :PROPERTIES   ((ARABICPROP 1))
)
(LEXICAL-ITEM
    :NAME jednou
    :SPELLING "jednou"
    :SAMPLE-SENTENCE "jednou+Num+Card^DB+Adv+Times"
   :FEATURES   (NUMERAL ARABICPROP TIMES UNINFLECTED)
   :PROPERTIES   ((ARABICPROP 1))
)
(LEXICAL-ITEM
    :NAME z
    :SPELLING "z"
    :SAMPLE-SENTENCE "z+Prep+Gen"
    :FEATURES (PREPOSITION UNINFLECTED)
)
(LEXICAL-ITEM
```

```
    :NAME ja2
    :SPELLING "já"
    :SAMPLE-SENTENCE "já+Pron+Pers+1P+Sg+Nom"
    :FEATURES (PRONOUN PERSONAL FIRSTPERSON)
)
(LEXICAL-ITEM
    :NAME tvu4j
    :SPELLING "tvùj"
    :SAMPLE-SENTENCE "PS2SIS4-tvùj+Adj+Poss+In2P+InSg"
    :FEATURES (PRONOUN POSSESIVE SECONDPERSON)
)
(LEXICAL-ITEM
    :NAME co-1
    :SPELLING "co-1"
    :SAMPLE-SENTENCE "Co to dìláš?"
    :FEATURES (PRONOUN INTERROGATIVE RELATIVE NEUTER)
)
(LEXICAL-ITEM
    :NAME jen
    :SPELLING "jen"
    :SAMPLE-SENTENCE "jen-1+Prtcl"
    :FEATURES (PARTICLE UNINFLECTED)
)
(LEXICAL-ITEM
    :NAME copula
    :SPELLING "být"
    :SAMPLE-SENTENCE "být+Verb+FutInd+2P+Pl"
    :FEATURES (COPULA VERB MIDDLE-VERB RELATIONAL-VERB
         RELATIONAL-BE-VERB)
)
(LEXICAL-ITEM
    :NAME auxiliary
    :SPELLING "být"
    :SAMPLE-SENTENCE "být+Verb+PresInd+2P+Pl"
    :FEATURES (AUXILIARY)
)
(LEXICAL-ITEM
    :NAME aplikovat
    :SPELLING "aplikovat"
    :SAMPLE-SENTENCE "aplikovat+Verb+Inf"
    :FEATURES (VERB PERFECTIVE IMPERFECTIVE DO-VERB
         EFFECTIVE-VERB DISPOSAL-VERB TRANSITIVE)
)
```

3.3 Russian lexicon

3.3.1 The structure of the lexical entries

The complete specification of information to be included in the lexical entry in the Russian lexicon is concerned to the intermediate prototype. Basic classification features are extended to include additional information required for the prototype. Morphological information is based on the classification system for Russian morphology developed by Zaliznjak [Zaliznjak, 1977]. The structure of morphological information is described in [Kruijff et al., 1998].

A typical lexical entry in the Russian lexicon includes the following slots: name, spelling, sample-sentence (may be omitted), features, properties, comments (may be omitted).

An example:

```
(LEXICAL-ITEM
    :NAME   EKRAN
    :SPELLING   "ekran"
    :SAMPLE-SENTENCE   "Na ekrane pojavitsja okno"
    :FEATURES  (NONE-OF-THATCOMP-TYPIC NONSUBSTITUTE COUNTABLE
                NOT-NOMINALIZATION COMMON-NOUN NOUN MASC)
    :PROPERTIES   (("экран")  (м 1 а))
    :COMMENTS   "Window appears"
)
```

- The :NAME slot contains an identifier of a lexical item (in Latin form, as the fonts for displaying Cyrillic characters has not been provided). This value is referred by SPL :LEX keywords and LEXIFY statements in the grammar. Two special names are: &-LINKER (for a comma) and ELLIPSISZERO (for items without lexical realization, such as *a/the* articles of English in the Russian grammar).
- The :SPELLING slot contains a real string of letters presenting the lexical item.
- The :SAMPLE-SENTENCE slot contains an example of a Russian phrase in which this item can appear in.
- The :FEATURES slot contains features, which are used by the Russian grammar. These are both lexical/morphological features (NOUN, FEMIN, etc.) and syntactic and semantic ones (DISPOSAL-VERB, RELATIONAL-BE-VERB, etc.).
- The :PROPERTIES slot contains a string with morphological information: the list of stems and the morphological index of a lexical item. The list of stems provides data for selection between different types of stems in the case of stem alternation (“один” and “один”, for example). The morphological index consists of a specifying string (a kind of part-of-speech index), a numerical class of declension or conjugation, an alternation mark, an accentuation index and additional features (for irregular words). This structure is described in greater detail in [Kruijff et al., 1998].
- The :COMMENT slot contains an English phrase corresponding to the Russian one shown in the sample-sentence slot.

Now we have five CAD-CAM lexicons: NOUN, ADJECTIVE, VERBS, INDECL, NUMERATIVES. The short fragments of the Dictionaries are presented below:

Noun.lexicon

```
 (LEXICAL-ITEM
    :NAME   DUGA
    :SPELLING   "duga "
    :SAMPLE-SENTENCE   " Chtoby narisovatj poliliniju, sostojaschuju iz
otrezkov prjamykh i dug "
    :FEATURES (NONE-OF-THATCOMP-TYPIC NONSUBSTITUTE COUNTABLE
               NOT-NOMINALIZATION COMMON-NOUN NOUN FEMIN)
    :PROPERTIES   (("дуг")  (ж 3 d))
    :COMMENTS   " To draw a line and arc combination polyline "
)
(LEXICAL-ITEM
    :NAME   FON
    :SPELLING   "fon"
    :SAMPLE-SENTENCE   "chtoby pokazatj tsvet fona"
    :FEATURES  (NONE-OF-THATCOMP-TYPIC NONSUBSTITUTE COUNTABLE
                NOT-NOMINALIZATION COMMON-NOUN NOUN MASC)
```

```
    :PROPERTIES   (("фон")  (м 1 а))
    :COMMENTS   " to display a background color "
)
```

Adjective.lexicon

```
 (LEXICAL-ITEM
    :NAME   CHETVERTYJ
    :SPELLING   "chetvertyj"
    :SAMPLE-SENTENCE   "Ukazite chetvjertuju tochku muljtilinii"
    :FEATURES          (ARABICPROP ORDINAL ADJECTIVE)
    :PROPERTIES   (("четверт")  (п 1 а))
)
(LEXICAL-ITEM
    :NAME   DEVJATYJ
    :SPELLING   " devjatyj"
    :SAMPLE-SENTENCE   "Ukazite devjatuju tochku muljtilinii"
    :FEATURES          (ARABICPROP ORDINAL ADJECTIVE)
    :PROPERTIES   (("девят")  (п 1 а))
)
```

Verbs.lexicon

```
 (LEXICAL-ITEM
    :NAME DOBAVITJ
    :SPELLING   "DOBAVITJ"
    :FEATURES   (PERFECT DO-VERB EFFECTIVE-VERB DISPOSAL-VERB
DESTINATION-VERB)
    :PROPERTIES   (("добави" "добавл")  (св 4 А))
)
(LEXICAL-ITEM
    :NAME ISCHEZNUTJ
    :SPELLING   "ISCHEZNUTJ"
    :FEATURES   (PERFECT DO-VERB EFFECTIVE-VERB MOTION-VERB
OBJECTNOTPERMITTED EVENT)
    :PROPERTIES   (("исчезну" "исчезн")  (св 3 * А "6"))
)
 (LEXICAL-ITEM
    :NAME   JAVLJATJSJA
    :SPELLING   "JAVLJATJSJA"
    :SAMPLE-SENTENCE   "kotorye javljajutsja otdeljnymi objektami"
    :FEATURES (IMPERFECT MIDDLE-VERB NOT-NEGATIVE RELATIONAL-BE-VERB
            ASCRIPTIVE INTENSION-VERB RELATIONAL-VERB
            NONE-OF-BITRANSITIVE-INDIRECTOBJECT REFLEXIVE)
    :PROPERTIES   (("явля")  (нсв 1 А -ся))
    :COMMENTS   "that are separate objects"
)
```

Indecl.lexicon

```
 (LEXICAL-ITEM
     :NAME          &-LINKER
     :SPELLING      ","
     :FEATURES     (PUNCTUATION SENTENCECONJUNCTION ADDITIVE CONJUNCT
                    NOT-SUBORDINATOR LINKER)
)
 (LEXICAL-ITEM
    :NAME   CHTOBY
    :SPELLING   "chtoby"
    :SAMPLE-SENTENCE   "Chtoby sokhranitj risunok"
    :FEATURES               (CONJUNCTION)
    :PROPERTIES   (("чтобы")  (н))
    :COMMENTS   "To save a drawing"
)
 (LEXICAL-ITEM
```

```
    :NAME   DLJA
    :SPELLING   "dlja"
    :FEATURES          (PREPOSITION PURPOSE)
    :PROPERTIES   (("для")  (н))
    :COMMENTS   "for"
)
```

Numeratives.lexicon

```
 (LEXICAL-ITEM
    :NAME   ODIN-NUMBER
    :SPELLING   " odin"
    :SAMPLE-SENTENCE   "Zapustite komandu L odnim iz sleduyushchikh
sposobov"
    :FEATURES  (SUBSTITUTE COUNTABLE PRONOUN CARDINAL)
    :PROPERTIES   (("одн" "один")  (п-мс 3 а))
    :COMMENTS   "... using one of these methods"
)
(LEXICAL-ITEM
    :NAME   DVA
    :SPELLING   "dva"
    :SAMPLE-SENTENCE   ""
    :FEATURES               (COUNTABLE CARDINAL)
    :PROPERTIES   (("дв") (числ))
    :COMMENTS   "two"
)
```

4. Morphology modules

The three languages we are dealing with are all languages with a relatively rich morphology. In this section, we describe the final solution adopted for dealing with the morphological derivation and inflection. All three languages employ an external morphological module. For each language, we describe the functionality of the respective module and its integration within the KPML grammar.

4.1 Technical issues

One of the problems that we had to solve was the correct handling of non-ASCII characters in Harlequin Lisp and in the communication with external modules. In the initialization file it is necessary to set the default character type to `'SIMPLE-CHAR:`

```
(SET-DEFAULT-CHARACTER-ELEMENT-TYPE 'SIMPLE-CHAR)
```

This enables the correct handling of 8th bit. Further, in the definition of the foreign language function it is necessary to assign the variable `external-format` the value `'(win32:code-page :id 1250)` (for the Central European character set) or the value `'(win32:code-page :id 1251)` (for the Cyrillic character set).

When KPML is using the internal morphology, the file `properties.lisp` contains lines which instruct the system to use the systemic morphology. It also contains the name of a function providing mapping between grammatical and morphological features.:

```
(define-language-morphology-requirements
  :language :CZECH
  :systemicized T
  :generator-fn CZECH-GRAMMATICAL-FEATURES-TO-LEXICAL-FEATURE)
```

To disable the systemic morphology, the instructions read:

```
(define-language-morphology-requirements
```

```
:language :CZECH
:systemicized NIL)
```

4.2 Bulgarian morphology module

An external morphological module was chosen to handle the Bulgarian morphology for the needs of AGILE [Skoumalová et al.,1998]. It consists of the following program segments:

- `Libmorf.dll` — the morphological processor. This file can be placed in a directory like \Windows\System or \Windows\System32 (for NT) along with the other Windows dll-s.
- `Bg.sv` — a vocabulary database. It must be placed in a fixed directory - C:\lingv. This inconvenience is due to be remedied.
- `bulg_morph.lisp` (`bulg_morph.fsl`) — this file integrates the Libmorf.dll in the Lisp environment. It can be placed anywhere, but its path should be described in the end of the `properties.lisp` file located in the Resource directory.
- `realize_word` — this module defines KPML's `REALIZE-INFLECTIFY` method.

The morphological characteristics used and their internal coding required by the morphological processor are described in the tables below. The program `realize_word` implements the mapping between the lexico-grammatical characteristics and the internal codes of the word forms.

Noun word forms

Word form No	Features
1.	Singular (base form)
2.	Singular, articled
3.	Singular, full-articled
4.	Plural
5.	Plural, articled
6.	Countable form

Adjective word forms

Word form No	Features
1.	Masculine, singular
2.	Masculine plural
3.	Feminine, singular
4.	Neuter singular
5.	Masculine, singular, articled
6.	Masculine, sing., full-articled
7.	Masculine, plural, articled
8.	Feminine, singular, articled

9.	Neuter, singular, articled

Numeral word forms

Word form No	Features
1.	Masculine, sing. (base form)
2.	Plural
3.	Feminine, singular
4.	Neuter, singular
5.	Masculine, singular, articled
6.	Masculine, sing., full-articled
7.	Plural, articled
8.	Feminine, singular, articled
9.	Neuter, singular, articled
10.	Masculine personal form
11.	Masculine personal, articled
12.	Approximate form
13.	Approx. form, articled

Verb word forms

Word form No	Features
1.	Present, sing., 1p.
2.	Present, sing., 2p.
3.	Present, sing., 3p.
4.	Present, pl., 1p.
5.	Present, pl., 2p.
6.	Present, pl., 3p.
7.	Past perf., sing., 1p.
8.	Past perf., sing., 2 & 3p.
9.	Past perf., pl., 1p.
10.	Past perf., pl., 2p.
11.	Past perf., pl., 3p.
12.	Past imperf., sing., 1p.
13.	Past imperf., sing., 2 & 3p.
14.	Past imperf., pl., 1p.

15.	Past imperf., pl.,2p.
16.	Past imperf., pl.,3p.
17.	Imperative, singular
18.	Imperative., plural

Only the base forms of propositions, adverbs, pronouns are considered.

4.3 Czech morphology module

For the Czech morphology an external module was used. The module is written in C and it accesses a lexical database containing more than 70 million morphological forms. For the purposes of the Agile project, a smaller database was compiled that contains only words used in the generated texts.

As the morphological database contains derived words as single entries, we decided to adopt this approach in our lexicon as well. Thus aspectual pairs, deverbative nouns and deverbative adjectives are listed separately. The connection between these entries is ensured by the mapping function from the domain concepts to lexicon (see above).

The communication between KPML and the morphological database is controlled by a module `morphgen.lisp`, and two `.dll` libraries. A schema of their interaction is shown in the Figure 4-1.

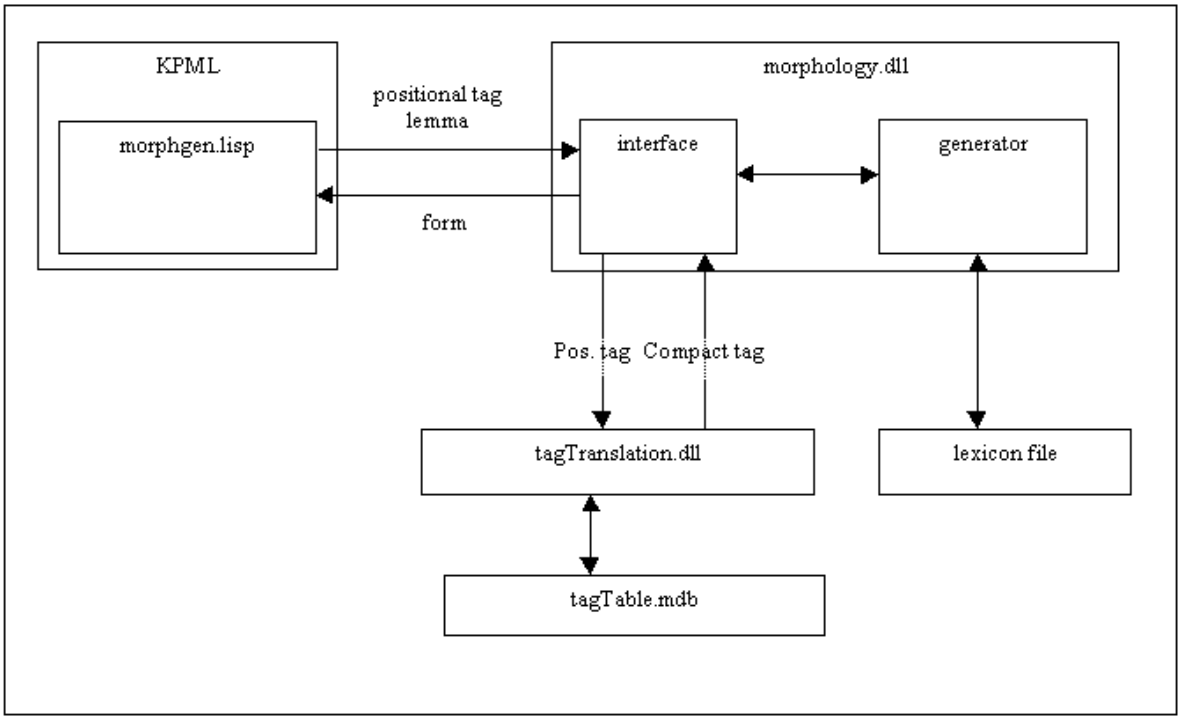

Figure 4-1 – Czech external morphology module

The `morphgen.lisp` module contains two parts:

- Mapping of the AGILE morphological features to the so-called positional tag. Positional tag is a string of 15 characters where each character corresponds to one morphological category, where the unused categories use are substituted by '-'. The function goes through the list of morphological features obtained from the KPML system and asks one

morphological category after another. If the value for the category is present in the list the corresponding position in the tag is set to that value, otherwise it is set to '-'.

- Interface to the C function: for this purpose the Harlequin `fli` library was used.

The module `morphology.dll` receives the request from the `morphgen.lisp` module consisting of a lemma (base form) and a positional tag. It asks the `tagTranslation.dll` module to translate the positional tag to the so-called compact tag and then sends lemma and compact tag to the generator, which returns the generated form. The generator module is written in C language.

Example of the request with the compact tag:

```
input:  abeceda NFS3A
output: abecedě
```

The compact tag contains the information about the word class (part of speech) and abbreviated grammatical categories relevant to that word class. In the example, the tag `NFS3A` stands for **n**oun, **f**eminine, **s**ingular, dative (**3**rd case in Czech terminology), **a**ffirmative form.

The module `tagTransl.dll` is responsible for translating tags from positional form to the compact one. There is an MS Access database containing a table with all the positional and compact tag pairs. This module is written in C++ because it uses the DAO library to access the MS Access database. The exported function uses the C calling conventions to be accessible from the module `morphology.dll`.

The reason for not using the compact tag directly is the fact that it is easier to translate the list of morphological features obtained from KPML to the positional tag and then use a translation table for transforming it to the compact tag.

The module `morphology.lisp` and header files of the `.dll` libraries are listed in the Appendix A.

4.4 Russian morphology module

4.4.1 Lexical features employed

Russian is a language with rich inflection. The grammatical categories that are given in the Table below are relevant for the Russian Morphological generation.

Gender	Masculine
	Feminine
	Neuter
Number	Singular
	Plural
Case	Nominative
	Genitive
	Dative
	Accusative

	Instrumental
	Prepositional
Gradation	Positive
	Comparative
	Superlative
Person	1st
	2nd
	3rd
Tense	Present
	Future
	Past
Form	Infinitive
	Imperative
	Adv-participle
	Active-participle
	Passive-participle

Part of this information is an attribute of the stem / stems (in the case of alternation), e.g. GENDER for Nouns. Another part is connected with the inflection: CASE for Nouns and Adjectives, PERSON for Verbs, etc.

4.4.2 The place of the morphological generator in the KPML system

The Russian morphological generator is an external component of the KPML system. Interaction is realized as a KPML interface to the generator. The morphological generator intervenes between Penman's internal code for producing morphological forms and the construction of the constituent tree containing the final generated strings. In the present model of the Russian morphological interface, full use of an object-oriented approach has been adopted.

4.4.3 Morphological classes in the morphological generator

The implementation of the morphological generator is based on a library of classes that describe grammatical features of Russian words. This pre-existing system is based on quite different principles than the generation grammar [Kruijff et al., 1998]. In the course of the project it has been ported to the Common Lisp Object System (CLOS) Common LISP.

The basic class of the morphological generator is `rusword`:

```
(defclass rusword ()
          ;;; The list of stems is
((stemlist :type list :initarg :stemlist  :accessor stemlist)
          ;;; The morphological index by Zaliznjak
   (morph-info :type list :initarg :morph-info  :accessor morph-
   info)
```

```
          ;;; A symbol which carries interpreted
          ;;; morphological features.
    (morph :accessor morph :initform (gentemp "morph"))
          ;;; The rest of slots is for current values.
   (stem :type string :initarg :stem :accessor stem)
   (flex :type string:initarg :flex :accessor flex)
   (wordform :type string :accessor wordform)))
```

Classes which are derived from the `rusword` include the class of declinable words and the class of verbs (including participial forms):

```
(defclass declinword (rusword)
  ((gender :initarg :gender :initform '(#\м) :accessor gender)
   (rnumber :initarg :rnumber :initform '(sing) :accessor
   rnumber)
   (rcase  :initarg :rcase :initform '(nom) :accessor rcase)
   (form  :initarg :form :initform 'simple :accessor form)))
```

Its subclasses are classes of noun, cardinal, pronoun and adjective (they have no additional slots).

Slots of the class of verbs define its characteristics:

```
(defclass verb (rusword)
 ((person :initarg :person :initform 1 :accessor person)
  (rnumber :initarg :rnumber :initform '(sing) :accessor rnumber)
  (tense :initarg :tense :initform 'present :accessor tense)
  (aspect :initarg :aspect :initform 'нсв :accessor aspect)
  (form :initarg :form :initform 'infinite :accessor form)
  (gender :initarg :gender :initform '(#\м) :accessor gender)
  (reflexive :initarg :reflexive :accessor reflexive)
  (adj :type adjective :initarg :adj :accessor adj)))
```

The last slot of verbs holds an adjective corresponding to a participial form of this verb.

4.4.4 Organization of the working morphological generator

During grammar traversal, certain choices result in the specification of morphological properties. For example, when a direct complement is inserted, it is supplied with the realization statement (`INFLECTIFY DIRECTCOMPLEMENT ACCUSATIVE`). Then, when morphology is not being handled by a multilingual grammatical description in the form of a system network, there is a call to the KPML method:

```
REALIZE-INFLECTIFY (chosen-word inflection-feature-list lg)
```

This has been specialized for Russian so that the features selected from the grammar are analyzed and mapped appropriately for calls to the external lexical form generation procedures described below. The lexical item for the chosen is retrieved by the KPML function:

```
(SETQ word-from-lexicon (fetch-lexicon-info Chosen-Word
   'instance)))
```

Morphological properties and the list of stems are retrieved from the lexical item, and if they are present an object carrying morphological information is created by a call to the morphological generator function `make-word`:

```
(when (and property-list stem)
        (setq morph-object (make-word stem property-list))
```

4.4.4.1 Initialization of morphological classes instances

The function `make-word` is designed for initialization of instances of the above mentioned classes. It analyzes the content of the string with a morphological index and creates an instance of a corresponding class:

```
(defun make-word (stemlist morph-string)
(case (car morph-string)
 ((м мо ж жо с со жо-мо) (make-noun stemlist morph-string))
 ((мест) (make-pronoun stemlist morph-string))
 ((п п-мс пм пж пс) (make-adjective stemlist morph-string))
 ((св нсв св-нсв) (make-verb stemlist morph-string))
 (otherwise (make-indeclinword stemlist morph-string))))
```

Non-latin characters are represented here by the standard Windows encoding Win-CP1251 and by Unicode in the application of the Harlequin Common LISP.

Later on, this method (using the function `make-word`) provides a mapping of grammatical features, selected by the grammar, into morphological properties handled by the morphological generator. This is done by appending the existing list of morphological features with the keywords followed by a corresponding morphological feature, for example:

```
(if (member 'imperative Grammatical-Feature-List)
   (SETQ lexical-feature-list (append '(:form imperative)
     lexical-feature-list)))
(if (intersection '(plural-form plural) Grammatical-Feature-List)
   (SETQ lexical -feature-list (append '(:rnumber (plur))
     lexical-feature-list)))
```

Then the morphological object together with the morphological features list are supplied as parameters to the generic function `generate-form`, which is specialized by the morphological object:

```
(SETQ spelling (apply 'generate-form (cons morph-object lexical-
   feature-list)))
```

4.4.4.2 Generation methods

The most important generic function for the generation of word forms is `generate-form`, which is specialized for declinable words and verbs. In the former case it receives keyword parameters for case, number and gender:

```
(defmethod generate-form ((r declinword) &key (rcase (rcase
  r)) (rnumber (rnumber r)) (gender (gender r)))
```

Keyword parameters supplied to `generate-form`, when specialized for verbs, include person, number, tense, reflexivity and form (the last belongs to the list of `'(finite infinite imperative active-participle passive-participle adv-participle passive)`. `adv-participle passive` is a special short form of a passive past participle that is used in building passive forms. For participial forms, case and gender may be specified:

```
(defmethod generate-form ((r verb) &key (person (person r))
(rnumber (rnumber r)) (form (form r)) (tense (tense r))
(reflexive (reflexive r)) (rcase '(nom)) (gender '(#\м)))
```

Several methods of the above-mentioned morphological classes provide specialized access which is relevant to generation of forms. They are called by `generate-form`, though they maybe irrelevant for external applications; in particular:

`Selectstem`	helps in selection of a stem from the list of possible stems according to morphological features;
`Setflex`	calculates a flex which conforms to current morphological features;
`Generate`	generates a word form which conforms to current morphological features;
`Generate-paradigm ((r noun))`	iterates a call to generate for number and case through the noun paradigm
`Generate-paradigm ((r adjective))`	iterates a call to generate for cases sequentially of masculine, feminine genders in singular number, then for cases of plural number
`Generate-paradigm ((r indeclinword))`	returns the first stem of the stemlist
`Generate-paradigm ((r cardinal))`	iterates a call to generate for case through the cardinal paradigm
`Generate-paradigm ((r verb))`	Iterates a call to generate all the forms of the verb paradigm (including infinite forms)

If fonts for displaying Cyrillic characters in KPML have not been provided, the string in Cyrillic characters which is output by morphological generator is converted to a string of their standard Latin equivalents by calling:

```
(if (not *cyrillic-display-loaded*)
   (SETQ spelling (transliterate spelling))))
...
(defconstant *cyrillic-characters* (coerce
   "абвгдежзийклмнопрстуфхцчшщъыьэюя" 'list))
(defconstant *latin-characters* '("a" "b" "v" "g" "d" "e" "zh"
   "z" "i" "j" "k" "l" "m" "n" "o" "p" "r" "s" "t" "u" "f" "kh"
   "ts" "ch" "sh" "sch" "~" "y" "j" "e" "ju" "ja"))
(defconstant *transliteration-pairs* (pairlis *cyrillic-
   characters* *latin-characters*))
(defun transliterate (cyr-string)
  (apply 'concatenate (append '(string) (mapcar #'(lambda (x) (or
   (cdr (assoc x *transliteration-pairs*)) (string x))) (coerce
   cyr-string 'list)))))
```

4.4.5 Examples of the generated output

```
(setq duga (make-word '("дуг")  '(ж 3 d)))
(generate-form duga :rcase '(acc) :rnumber '(sing))
=> "дугу"
(generate-paradigm duga)
"дуга"
"дуги"
```

```
"дуге"
"дугу"
"дугой"
"дуге"
"дуги"
"дуг"
"дугам"
"дуги"
"дугами"
"дугах"

(setq imja (make-word '("им" "имен")  '(с 8 а)))
(generate-form imja :rcase '(acc) :rnumber '(sing))
=> "имя"
(generate-form imja :rcase '(inst) :rnumber '(sing))
=> "именем"

(setq edinichnyj (make-word '("диалогов")  '(п 1 а)))
(generate-form granichny :rcase '(acc) :rnumber '(sing) :gender
   '(#\ж))
=> "диалоговую"

(setq narisovatj (make-word '("нарисова" "нарису")  '(св 2 а)))
(generate-form narisovatj :form 'infinite)
=> "нарисовать"
(generate-form narisovatj :form 'passive-participle :tense 'past
   :rcase '(gen) :rnumber '(sing) :gender '(#\м))
=> "нарисованного"
(generate-form narisovatj :form 'active-participle :tense 'past
   :rcase '(gen) :rnumber '(sing) :gender '(#\м))
=> "нарисовавшего"

(setq pojavljatsja (make-word '("появля")  '(нсв 1 b -ся)))
(generate-form pojavljatsja :form 'active-participle :tense
   'present :rcase '(inst) :rnumber '(sing) :gender '(#\ж))
=> "появляющейся"
(generate-form pojavljatsja :form 'finite :tense 'present :person
   3 :rnumber '(plur))
=> "появляются"

(setq odin (make-word '("одн" "один")  '(п-мс 3 * b)))
(generate-form odin :rcase '(inst) :rnumber '(sing) :gender
   '(#\м))
=> "одним"
(generate-form odin :rcase '(nom) :rnumber '(sing) :gender
   '(#\м))
=> "один"
```

5. Conclusions

The format of a lexical entry described in this report reflects our understanding of problems of interaction between the grammar and the lexicon at the half-way stage of the project. It may happen that during the future work we will have to add some new lexical features to those described here or to change usage of some existing features in the grammar. These changes will be described in reports of WP7 — Implementation of tactical generators.

The lexicons for all three languages cover the texts for both the Initial Demonstrator and the Intermediate Prototype. Entries for the Final Prototype will be added in the course of the next year.

The morphological modules for all three languages are completed and they are working.

References

Bateman, J. [1997] KPML Development Environment.

Halliday, M. A. K. [1985] *An introduction to Functional Grammar.* Arnold, London.

Kruijff, G.-J. M. et al. [1998] Modelling Lexical Resources in KPML for Generating Instructions in Slavic Languages. AGILE project, task WP4.2 deliverable, University of Brighton.

Skoumalová, H. et al. [1998] Lexical specifications and resources. Agile project, task WP4.1 deliverable, University of Brighton.

Zaliznjak, A. [1977]. *Grammatical Dictionary of the Russian Language,* Russkij Jazyk, Moscow. (in Russian).

Appendix A. Mapping DM concepts onto lexical items

cadcam-lex-bu.lisp

```
(in-package :KPML)

(annotate-concept DM::ADD :lex-items (PRIBAVIA PRIBAVIAM))
(annotate-concept DM::ANGLE :lex-items (UGUL))
(annotate-concept DM::ARC :lex-items (DUGA))
(annotate-concept DM::CONSTRAINT :lex-items (OGRANICHENIE))
(annotate-concept DM::CREATE :lex-items (SAZDAM SAZDAVAM SAZDAVANE))
(annotate-concept DM::DEFINE :lex-items (DEFINIRAM DEFINIRANE))
(annotate-concept DM::DESCRIPTION :lex-items (OPISANIE))
(annotate-concept DM::DISTANCE :lex-items (RAZSTOJANIE))
(annotate-concept DM::HATCH :lex-items (SHTRIHOVKA))
(annotate-concept DM::HEADING :lex-items (ZAGLAVIE))
(annotate-concept DM::LINE-SEGMENT :lex-items (OTSECHKA))
(annotate-concept DM::MODE :lex-items (REJIM ))
(annotate-concept DM::NOT-SEE :lex-items (SKRIVANE))
(annotate-concept DM::POINTS :lex-items (TOCHKI))
(annotate-concept DM::PROPERTIES :lex-items (SVOISTVA))
(annotate-concept DM::ROOT :lex-items (KOREN))
(annotate-concept DM::SEE :lex-items (VIDIA VIJDAM))
(annotate-concept DM::SPECIFICATION :lex-items (SPESIFIKATSIJA))
(annotate-concept DM::STYLE :lex-items (STIL))
(annotate-concept DM::VERTICES :lex-items (VURHOVE))
(annotate-concept DM::BUTTON :lex-items (KLAVISH))
(annotate-concept DM::CHOOSE :lex-items (IZBERA IZBIRAM))
(annotate-concept DM::CLICK :lex-items (SHTRAKNA SHTRAKVAM))
(annotate-concept DM::DRAW :lex-items (CHERTAJA CHERTAENE))
(annotate-concept DM::DRAWING :lex-items (CHERTEJ))
(annotate-concept DM::ENTER :lex-items (ZADAM ZADAVAM ZADAVANE))
(annotate-concept DM::MENU :lex-items (MENU))
(annotate-concept DM::SAVE :lex-items (ZAPISHA ZAPISVAM ZAPISVANE))
(annotate-concept DM::SOFTWARE-COMMAND :lex-items (KOMANDA))
(annotate-concept DM::START-TOOL :lex-items (STARTIRAM))
```

cadcam-lex-cz.lisp

```
(in-package :KPML)

(annotate-concept DM::ADD :lex-items (prbavia pribaviam))
(annotate-concept DM::ANGLE :lex-items (u2hel))
(annotate-concept DM::ARC :lex-items (oblouk))
(annotate-concept DM::CONSTRAINT :lex-items (omezeni2))
(annotate-concept DM::CREATE :lex-items (vytvor3it vytva2r3et vytvor3eni2
   vytva2r3eni2))
(annotate-concept DM::DEFINE :lex-items (definovat definova2ni2))
(annotate-concept DM::DESCRIPTION :lex-items (popis))
(annotate-concept DM::DISTANCE :lex-items (vzda2lenost))
(annotate-concept DM::HATCH :lex-items (str3i2s3ka))
(annotate-concept DM::HEADING :lex-items (nadpis))
(annotate-concept DM::LINE-SEGMENT :lex-items (u2sec3ka))
(annotate-concept DM::MODE :lex-items (rez3im))
(annotate-concept DM::NOT-SEE :lex-items (nevide3t))
(annotate-concept DM::POINTS :lex-items (body))
(annotate-concept DM::PROPERTIES :lex-items (vlastnosti))
(annotate-concept DM::ROOT :lex-items (kor3en))
(annotate-concept DM::SEE :lex-items (vide3t))
(annotate-concept DM::SPECIFICATION :lex-items (urc3eni2))
```

```
(annotate-concept DM::STYLE :lex-items (styl))
(annotate-concept DM::VERTICES :lex-items (vrcholy))
(annotate-concept DM::BUTTON :lex-items (tlac3i2tko))
(annotate-concept DM::CHOOSE :lex-items (vybrat vybi2rat vybra2ni2
   vybi2ra2ni2))
(annotate-concept DM::CLICK :lex-items (kliknout klikat))
(annotate-concept DM::DRAW :lex-items (kreslit nakreslit kresleni2
   nakresleni2))
(annotate-concept DM::DRAWING :lex-items (obra2zek))
(annotate-concept DM::ENTER :lex-items (zadat zada2vat zada2ni2
   zada2va2ni2))
(annotate-concept DM::MENU :lex-items (menu))
(annotate-concept DM::SAVE :lex-items (uloz3it ukla2dat uloz3eni2
   ukla2da2ni2))
(annotate-concept DM::SOFTWARE-COMMAND :lex-items (pr3i2kaz))
(annotate-concept DM::START-TOOL :lex-items (spustit spous3te3t
   spus3te3ni2 spous3te3ni2))
```

cadcam-lex-ru.lisp

```
(in-package :KPML)

(annotate-concept DM::ADD :lex-items (dobavitj dobavljatj dobavlenie))
(annotate-concept DM::ANGLE :lex-items (ugol))
(annotate-concept DM::ARC :lex-items (duga))
(annotate-concept DM::CONSTRAINT :lex-items (ogranichenie))
(annotate-concept DM::CREATE :lex-items (sozdatj sozdavatj sozdanie))
(annotate-concept DM::DEFINE :lex-items (opredelitj opredeljatj
   opredelenie))
(annotate-concept DM::DESCRIPTION :lex-items (opisanie))
(annotate-concept DM::DISTANCE :lex-items (passtojanie))
(annotate-concept DM::HATCH :lex-items (shtrihovka))
(annotate-concept DM::HEADING :lex-items (zagolovok))
(annotate-concept DM::LINE-SEGMENT :lex-items (otrezok))
(annotate-concept DM::MODE :lex-items (rezhim))
(annotate-concept DM::NOT-SEE :lex-items (nevide3t))
(annotate-concept DM::POINTS :lex-items (tochka))
(annotate-concept DM::PROPERTIES :lex-items (svojstvo))
(annotate-concept DM::ROOT :lex-items (korenj))
(annotate-concept DM::SEE :lex-items (videtj))
(annotate-concept DM::SPECIFICATION :lex-items (spetsifikatsia))
(annotate-concept DM::STYLE :lex-items (stylj))
(annotate-concept DM::VERTICES :lex-items (vershina))
(annotate-concept DM::BUTTON :lex-items (knopka))
(annotate-concept DM::CHOOSE :lex-items (vybratj vybiratj vybor))
(annotate-concept DM::CLICK :lex-items (nazhatj nazhimatj nazhatije))
(annotate-concept DM::DRAW :lex-items (risovatj narisovatj risovanije))
(annotate-concept DM::DRAWING :lex-items (risunok))
(annotate-concept DM::ENTER :lex-items (vvesti vvoditj vvod))
(annotate-concept DM::MENU :lex-items (menju))
(annotate-concept DM::SAVE :lex-items (sohranitj sohranjatj sohranenije))
(annotate-concept DM::SOFTWARE-COMMAND :lex-items (komanda))
(annotate-concept DM::START-TOOL :lex-items (zapustitj zapuskatj zapusk))
```

Appendix B. Code for accessing the Czech external module

Morphology.lisp

```
(in-package "COMMON-LISP-USER")

;;; This file contains interface between KPML and the external
;;; morphological module
;;; Before any inflection the function initMorphology has to be called
;;; The top function called by KPML is the KPML::realize-Inflectify
;;; function

;;; Following two lines introduce the module morhology.dll to the lisp
;;;
(fli::load-dll "c:/windows/system/morphology.dll")
(fli::register-module 'morphology)

;;; Top function of this module - called by KPML to realize inflection.
;;;
;;; @param chosen-word - lemma of the word that should be inflected
;;; @param Feature-List - morphological features obtained from the KPML
;;; @return generated inflected form
;;; @see #generateForm, #translateToTag
;;;
(defmethod KPML::realize-Inflectify (chosen-word Feature-List (lg (eql
:czech)))
   (let
      (
         ;;; Gets lexical features of chosen-word from the lexicon and
         ;;; appends them to Feature-List
         (features
            (remove-duplicates (append Feature-List (kpml::fetch-lexicon-
info chosen-word 'kpml::features))))
      )
      (if (member 'kpml::uninflected features)
         ;;; For uninflected words returns just the basic form
         (kpml::fetch-lexicon-info chosen-word 'kpml::spelling)
         ;;; For inflectable words calles the external module
         (generateForm
            (kpml::fetch-lexicon-info chosen-word 'kpml::spelling)
            (translateToTag features chosen-word)
         )

      )
   )
)

;;; Assigns lisp name generateFormC to the C function generateForm
;;; located in morphology.dll
;;;
;;; @see #generateForm
;;;
 (fli:define-foreign-function (generateFormC "generateForm")
      ((aLemma :pointer) (aTag :pointer))
   :result-type :pointer
   :documentation "performs inflection"
)

;;; Assigns lisp name initMorhologyC to the C function initMorphology
located
;;; in morphology.dll
```

```
;;;
;;; @see #initMorphology
;;;
(fli:define-foreign-function (initMorphologyC "init") ((aLexicon
:pointer))
   :result-type :pointer
   :documentation "initializes the morphological module"
)

;;; Initializes morphology module.
;;; The lexicon with supplied name is opened.  Performs mapping of lisp
;;; strings to C strings and back. Calls function initMorphologyC which
;;; is located in morphology.dll under name init.
;;;
;;; @param aLexicon - full file name of the lexicon. If the name of the
lexicon is a
;;;   empty string no lexicon is opened and the morhological module
returns just
;;;   a lemma concatenated with a tag
;;; @return dummy string (I don't know how connect void function to LISP)
;;; @see #initC
;;;
(defun initMorphology(aLexicon)
   (fli:with-foreign-string
      (pLexicon element-count byte-count external-format
win32::*multibyte-code-page-ef*)
      aLexicon
      (declare (ignore element-count byte-count))

      (fli::convert-from-foreign-string
         (initMorphologyC pLexicon)
         :external-format win32::*multibyte-code-page-ef*
      )
   )
)

;;; Generates surface form based on the supplied lemma and tag.
;;; Performs mapping of lisp strings to C strings and back. Important
;;; is the pararameter :external-format '(win32:code-page :id 1250)
;;; which ensures that Czech characters can be passed.
;;; Calls function generateFormC which is located in morphology.dll
;;; under name generateForm.
;;; e.g. (generateform "hrad" "NNFS6-----A----") -> "hradu"
;;;
;;; @param aLemma - lemma
;;; @param aTag - positional tag
;;; @return generated form
;;; @see #generateFormC, #realize-Inflectify
;;;
(defun generateForm (aLemma aTag)
   (fli:with-foreign-string
      (pTag element-count byte-count external-format '(win32:code-page
:id 1250))
      aTag
      (declare (ignore element-count byte-count))

      (fli:with-foreign-string
         (pLemma element-count byte-count :external-format '(win32:code-
page :id 1250))

         aLemma
         (declare (ignore element-count byte-count))
```

```
         (fli::convert-from-foreign-string
            (generateFormC pLemma pTag)
            :external-format '(win32:code-page :id 1250)
         )
      )
   )
)

;;; Gets specified property of specified lexeme from lexicon
;;;
(defun fetch-lexical-property (property lexeme)
   (second
      (assoc property (KPML::fetch-lexicon-info lexeme
'KPML::properties))
   )
)

;;; Translates list of characters to the string of that characters.
;;;
;;; @param tags - list of characters corresponding to a tag
;;; @return string representing a tag
;;;
(defun LIST-OF-TAGS-TO-STRING (tags)
   (format nil "~{~A~}" tags)
)

;;; Creates a positional tag from a feature-list and lexeme obtained from
KPML
;;;
(defun translateToTag (feature-list lexeme)
      (let ((tag nil))
      (Progn
      ;;; Sets each character of the positional tag
         (SETQ tag (append tag (selectPOS      Feature-List)) )
         (SETQ tag (append tag (selectSubPOS   Feature-List)) )
         (SETQ tag (append tag (selectGender      Feature-List)) )
         (SETQ tag (append tag (selectNumber   Feature-List)) )
         (SETQ tag (append tag (selectCase      Feature-List)) )
         (SETQ tag (append tag (selectPGender   Feature-List)) )
         (SETQ tag (append tag (selectPNumber   Feature-List)) )
         (SETQ tag (append tag (selectPerson      Feature-List)) )
         (SETQ tag (append tag (selectTense      Feature-List)) )
         (SETQ tag (append tag (selectGrade      Feature-List)) )
         (SETQ tag (append tag (selectNegation   Feature-List)) )
         (SETQ tag (append tag (selectVoice      Feature-List)) )
         (SETQ tag (append tag '(-) ) )
         (SETQ tag (append tag '(-) ) )
         (SETQ tag (append tag (selectVar      Feature-List)) )
         ;;; Transforms the created list to a string
         (list-of-tags-to-string tag)
      )
   )
)

;;; Returns the character expressing part of speech
;;;
(defun selectPOS (Feature-List)
   (cond
      ((member 'kpml::noun            Feature-List)   '(N))
      ((member 'kpml::adjective         Feature-List)   '(A))
      ((member 'kpml::pronoun         Feature-List)   '(P))
      ((member 'kpml::conjunction         Feature-List)   '(C))
      ((member 'kpml::verb            Feature-List)   '(V))
```

```
      ((member 'kpml::adverb         Feature-List)   '(D))
      ((member 'kpml::preposition         Feature-List)   '(R))
      ((member 'kpml::conjunction         Feature-List)   '(J))
      (T                  '(-))
   )
)

;;; Returns the character expressing subpart of speech
;;;
 (defun selectSubPOS (Feature-List)
   (cond
      ((member 'kpml::noun            Feature-List)   '(N))
      ((member 'kpml::pronoun-Personal      Feature-List)   '(P))
      ((member 'kpml::pronoun-Possesive      Feature-List)   '(S))
      (T                  '(-))
   )
)

;;; Returns the character expressing part of gender
;;;
 (defun selectGender (Feature-List)
   (cond
      ((member 'kpml::gender-f-form      Feature-List)   '(F))
      ((member 'kpml::gender-n-form      Feature-List)   '(N))
      ((member 'kpml::gender-m-form      Feature-List)   '(M))
      ((member 'kpml::gender-i-form      Feature-List)   '(I))
      ((member 'kpml::feminine         Feature-List)   '(F))
      ((member 'kpml::neuter         Feature-List)   '(N))
      ((member 'kpml::m-animate         Feature-List)   '(M))
      ((member 'kpml::m-inanimate         Feature-List)   '(I))
      (T                  '(-))
   )
)

;;; Returns the character expressing number
;;;
(defun selectNumber (Feature-List)
   (cond
      ((member 'kpml::number-sg-form      Feature-List)   '(S))
      ((member 'kpml::number-pl-form      Feature-List)   '(P))
      ((member 'kpml::singular-form      Feature-List)   '(S))
      ((member 'kpml::plural-form         Feature-List)   '(P))
      ((member 'kpml::sg-form         Feature-List)   '(S))
      ((member 'kpml::pl-form         Feature-List)   '(P))
      (T                  '(-))
   )
)

;;; Returns the character expressing case
;;;
(defun selectCase (Feature-List)
   (cond
      ((member 'kpml::case-nominative      Feature-List)   '(1))
      ((member 'kpml::case-genitive      Feature-List)   '(2))
      ((member 'kpml::case-dative         Feature-List)   '(3))
      ((member 'kpml::case-accusative      Feature-List)   '(4))
      ((member 'kpml::case-vocative      Feature-List)   '(5))
      ((member 'kpml::case-locative      Feature-List)   '(6))
      ((member 'kpml::case-instrumental      Feature-List)   '(7))
      ((member 'kpml::nominative         Feature-List)   '(1))
      ((member 'kpml::genitive         Feature-List)   '(2))
      ((member 'kpml::dative         Feature-List)   '(3))
      ((member 'kpml::accusative         Feature-List)   '(4))
```

```
      ((member 'kpml::vocative         Feature-List)   '(5))
      ((member 'kpml::locative         Feature-List)   '(6))
      ((member 'kpml::instrumental         Feature-List)   '(7))
      ((member 'kpml::nominative-case      Feature-List)   '(1))
      ((member 'kpml::genitive-case      Feature-List)   '(2))
      ((member 'kpml::dative-case         Feature-List)   '(3))
      ((member 'kpml::accusative-case      Feature-List)   '(4))
      ((member 'kpml::vocative-case      Feature-List)   '(5))
      ((member 'kpml::locative-case      Feature-List)   '(6))
      ((member 'kpml::instrumental-case      Feature-List)   '(7))
      (T                  '(-))
   )
)

;;; Returns the character expressing relative gender
;;;
(defun selectPGender (Feature-List)
   (cond
      ((member 'kpml::pGender-f-form      Feature-List)   '(F))
      ((member 'kpml::pGender-n-form      Feature-List)   '(N))
      ((member 'kpml::pGender-m-form      Feature-List)   '(M))
      ((member 'kpml::pGender-i-form      Feature-List)   '(I))
      ((member 'kpml::pFeminine         Feature-List)   '(F))
      ((member 'kpml::pNeuter         Feature-List)   '(N))
      ((member 'kpml::pM-animate         Feature-List)   '(M))
      ((member 'kpml::pM-inanimate      Feature-List)   '(I))
      (T                  '(-))
   )
)

;;; Returns the character expressing relative number
;;;
(defun selectPNumber (Feature-List)
   (cond
      ((member 'kpml::pNumber-sg-form      Feature-List)   '(S))
      ((member 'kpml::pNumber-pl-form      Feature-List)   '(P))
      ((member 'kpml::pSingular-form      Feature-List)   '(S))
      ((member 'kpml::pPlural-form         Feature-List)   '(P))
      ((member 'kpml::pSg-form         Feature-List)   '(S))
      ((member 'kpml::pPl-form         Feature-List)   '(P))
      (T                  '(-))
   )
)

;;; Returns the character expressing person
;;;
(defun selectPerson (Feature-List)
   (cond
      ((member 'kpml::Person-First-Form      Feature-List)   '(1))
      ((member 'kpml::Person-Second-Form   Feature-List)   '(2))
      ((member 'kpml::Person-Third-Form      Feature-List)   '(3))
      (T                  '(-))
   )
)

;;; Returns the character expressing tense
;;;
(defun selectTense (Feature-List)
   (cond
      ((member 'kpml::tense-present-Form      Feature-List)   '(P))
      ((member 'kpml::tense-past-Form      Feature-List)   '(M))
      ((member 'kpml::tense-future-Form      Feature-List)   '(F))
      ((member 'kpml::present-Form      Feature-List)   '(P))
```

```
      ((member 'kpml::past-Form         Feature-List)   '(M))
      ((member 'kpml::future-Form         Feature-List)   '(F))
      (T                  '(-))
   )
)

;;; Returns the character expressing grade
;;;
(defun selectGrade (Feature-List)
   (cond
      ((member 'kpml::positive         Feature-List)   '(1))
      ((member 'kpml::comparative         Feature-List)   '(2))
      ((member 'kpml::superlative         Feature-List)   '(3))
      (T                  '(-))
   )
)

;;; Returns the character expressing negation
;;;
(defun selectNegation (Feature-List)
   (cond
      ((member 'kpml::negative         Feature-List)   '(N))
      (T                  '(A))
   )
)

;;; Returns the character expressing voice
;;;
(defun selectVoice (Feature-List)
   (cond
      ((member 'kpml::voice-active-Form      Feature-List)   '(A))
      ((member 'kpml::voice-passive-Form      Feature-List)   '(P))
      (T                  '(-))
   )
)

;;; Returns the character expressing variant (formal/ colloquial/ ..) -
not used for AGILE
;;;
(defun selectVar (Feature-List)
   (cond
      (T   '(-))
   )
)
```

morphologyApi.h

```
/** This file contains interface functions of the library morphology.dll.
  * Before generating the function init has to be called.
  * Inflection is done by the function generateForm.
  *
  * @author Jiri Hana (jiri.hana@mff.cuni.cz);
  *  UFAL MFF UK Prague, Czechia
  * @version 2.1
  * @date 1999-6-30
  */

// Macro to swich between importing and exporting of function
// automatically
// Functions serving as interface of this library have this macro in
// their declaration
// Compiler compiling this library has to have MORPHOLOGY_EXPORTS defined
```

```
#ifdef MORPHOLOGY_EXPORTS
      #define MORPHOLOGY_API __declspec(dllexport)
#else
      #define MORPHOLOGY_API __declspec(dllimport)
#endif

#define true 1;
#define false 0;


#ifdef __cplusplus
extern "C" {
#endif


/** Initializes all neccesary structures.
  * Specifies if lexicon should be used or if only test version
  * without lexicon (returning lemma concatenated with tag) is used
  *
  * @param aLexicon full name of the file with lexicon or empty string
  *         if the lexicon is not be used
  * @return "OK" if the initialization was successful
  * (I don't how to return bool to LISP)
  * @see init, mIsLexiconLoaded
  */
MORPHOLOGY_API char* init(const char* aLexicon);


/** Cleans all structures from memory (Should)
  * This function is not used because there is no way to disconect
  * the dll from lisp - therefore it is done automatically by the
  * system when lisp is closed
  */
MORPHOLOGY_API void finalize();

/** Generates a form based on supplied lemma and tag.
  * If the lexicon is not used it returns lemma concatenated with tag.
  *
  * @param aLemma lemma
  * @param aTag positional tag (the tag is translated to the compact
  * form by
  *         by a tagTransl.dll library)
  * @return inflected form or lemma concatenated with tag
  * @see init, mIsLexiconLoaded, generateFormImp, generateLemmaPlusTag
  */
MORPHOLOGY_API char* generateForm(unsigned char* aLemma, unsigned char*
aTag);

#ifdef __cplusplus
}
#endif
```

tagtranslApi.h

```
/** This file contains API of the libray tagTransl.dll.
  * The api uses C-language calling convention to be callable from C,
  * Lisp, etc.
  *
  * This library is used to translate tags from positional form
  * to the compact one or vice versa.
  * It uses MS Access database table to store "dictionary" of tags.
  * The MS DAO (ActiveX)
```

```
  * library is used to access the database
  *
  * @author Jiri Hana (jiri.hana@mff.cuni.cz);
  *  UFAL MFF UK Prague, Czechia
  * @version 1.1
  * @date 1999-6-30
  */

extern "C"
{

/** Translates a tag from positional form to the compact one
  *
  * @param aPosTag tag in positional form
  */
const char* CTagTransl_toCompact(char* aPosTag);


/** Translates a tag from compact form to the positional one
  *
  * @param aCompactTag tag in compact form
  */
const char* CTagTransl_toPos(char* aCompactTag);
};
```