FI MU

Faculty of Informatics
Masaryk University Brno

PRESEMT Phrase Model Generator

by

Michalis Troullinos

FI MU Report Series **FIMU-RS-2013-3**

Copyright © 2013, FI MU **March 2013**

**Copyright © 2013, Faculty of Informatics, Masaryk University.
All rights reserved.**

**Reproduction of all or part of this work
is permitted for educational or research use
on condition that this copyright notice is
included in any copy.**

Publications in the FI MU Report Series are in general accessible via WWW:

`http://www.fi.muni.cz/reports/`

Further information can be obtained by contacting:

**Faculty of Informatics
Masaryk University
Botanicka 68a
602 00 Brno
Czech Republic**

PRESEMT Phrase Model Generator

Michalis Troullinos

November 29, 2013

1 Phrasing model generator (PMG)

1.1 Basic aspects & design

The Phrasing model generator uses the output of the Phrase aligner module to train a phrasing model for the SL. The output of the Phrase aligner module contains the segmentation into phrases of the SL side of the bilingual corpus. This model is then applied for segmenting an SL text being input to the PRESEMT system for translation. The aforementioned procedure is illustrated in figure 1.

The main method for extracting the phrasing model is statistical-based, since a substantial amount of research has already been invested in creating statistical language models in NLP tasks (e.g. [2]).

1.2 Design of Phasing Model Generator

The Phrasing model generator uses as input the PAM output and specifically the XML string representing the phrase-aligned SL side of the parallel corpus. The PMG output consists of the SL texts to be translated by PRESEMT, which are segmented into phrases compatible with the phrasing model used in the TL. The method for extracting the phrasing model is statistical and following comparative evaluations is based on the CRF (Conditional Random Fields) model [1].

The CRF model that is trained with the above process is being used by the PRESEMT machine translation system as a phrasal segmentation module for the SL. The purpose of CRF is to group together words that form a complete phrase as a pre-processing operation to the translation. In the translation process, the phrases created by the Phrasing model generator will be used to generate a segmentation of the input sentence and,

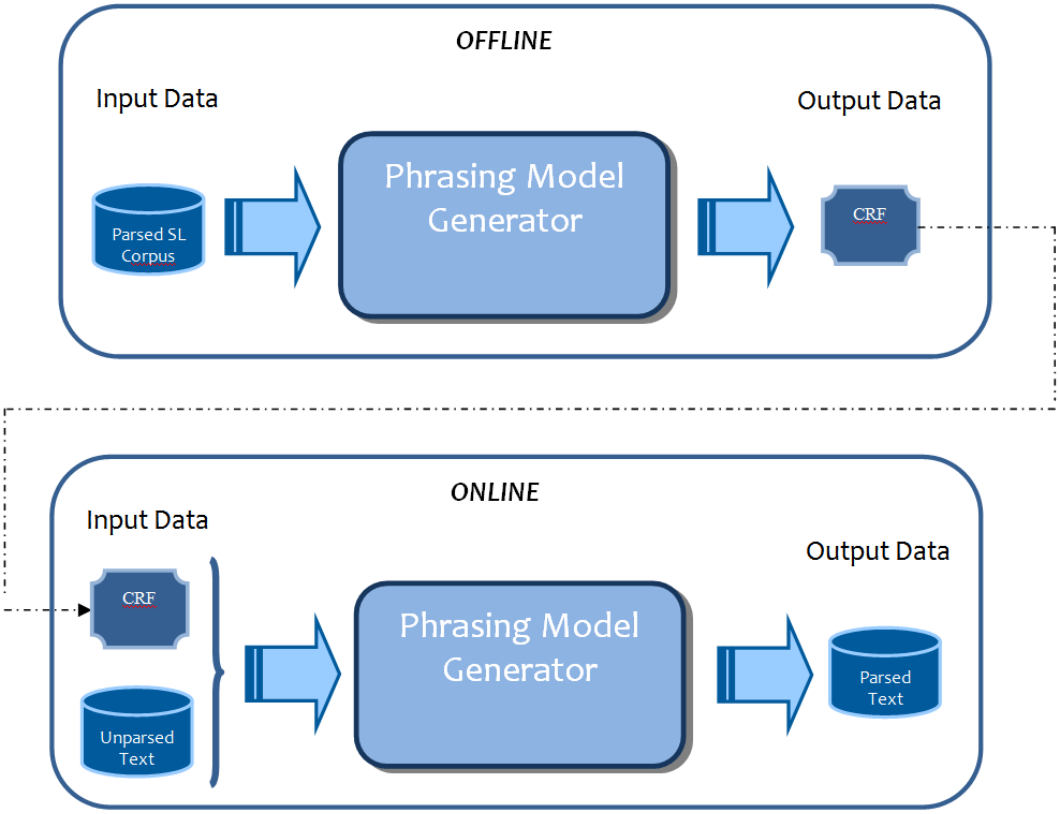

Figure 1: Overview of PMG

based on that, to search for appropriate translations. One main requirement for the methodology used to develop this module is to be language-independent, in a way that the proposed model can adapt to any language provided that a training set of acceptable quality is available, in the form of a parallel corpus of sentences coupled with a parser that splits TL-side sentences into phrases.

1.3 PMG Implementation

For implementation, the **MALLET** package was chosen as it is written in Java (which is the programming language favored in the PRESEMT project) and has more extensive support in comparison to other existing software implementations of CRF. Additionally, for parameter passing and object injection the Spring[1] framework was used. The Spring framework made executing experiments easier as for each set of experiments it only requires the creation of a set of xml configuration files (with no need to modify java code).

[1] http://www.springsource.org/

Finally the training method used for the CRF was the **CRFTrainerByLabelLikelihood** as provided by MALLET.

The regular expressions concept was selected for tag manipulation and transformation within PMG, as it provides a wide range of capabilities. In the current implementation only the PoS information per word is taken into account. In addition, in the case of PoS tags which have a case, this type of information is retained (e.g. Nominative, Accusative). This choice was made since much less training data would be required for training in order to elicit syntactic relationships involving only tag sets rather than words (given that the number of possible tags is much lower than the number of potential words/lemmas). As the number of sentences is limited to approximately 200 to simplify the creation of MT systems for new language pairs, the use of PoS tags confers the ability to train a realistic model even with this limited set, by extracting sufficient training data. Besides, by using only tags, the model is easier to adapt to an arbitrary new language.

Mapping

The CRF model provided within the MALLET open source toolkit was used in order to create a mapping between the sequence of tags that constitute a period and the sequence of attributes that describe the words' role inside each phrase. In the experiments reported here, the annotation of segments is summarised as follows:

- A three-column format was used for training data, where each word occupies one line, the first column containing the actual token, the second one the PoS tag and the third column the clause information.
- The segment information is expressed as a single-letter label, ("**B**", if this is the first token in the given segment, "**I**" otherwise), followed by the type of segment (e.g. B-NPAC indicates the first token of a segment, which is a Noun Phrase whose head is in Accusative). At a second level of abstraction, only the type of phrase was included, without a case feature (e.g. the segmentation information would be of the form B-NP or I-NP).
- Tokens not belonging to any specific segment are identified by a "**0**" entry in the third column. This is used when, for instance, a token is a punctuation mark.
- The end of a sentence is identified by an empty line.

As the type of phrase is carried over from the TL, the diversity of phrase types for a PMG-generated MT system for a given language pair is defined by the parser used in the TL.

1.4 Experimental setup and results

Experimental results

For the PMG experiments, subsets of the parallel corpora available for each language pair have been created, in order to have two independent resources for each language pair. The first set used for training the phrase model and the second set for evaluating the segmentation accuracy. The accuracy of the second set is compared against the golden segmentation and as a result the accuracy has been evaluated within the PRESEMT project only for the language pairs with available golden sets.

Additionally for the language pair Greek-English, an extra corpus that is termed “development corpus" has been used for evaluation. The segmentation accuracy is compared against the reference (golden) segmentation (which manually created), by using two distinct evaluator functions. Both of the functions are implemented in software, and evaluate the edit distance between the optimum segmentation and the CRF model’s segmentation. The first one is based on the edit distance calculated over the tokens while the second one is calculated over phrases. The evaluation of the development corpus allows the use of each sentence from the bilingual parallel corpus for training.

1.4.1 Performance factors

A number of modifications and extensions have been tested on the CRF model used to obtain the best possible performance, these including:

1. modifying the CRF **order** parameter which sets the time window based on which the model creates connections between the training sequence and the observed symbols,

2. adding a different feature counter functionality by creating a java class implementing the MALLET basic Pipe interface used for feature measurement, employing **regular expressions** to modify input sequences,

3. combining different input parameters such as tags and lemmas to represent **n-gram features** by replacing each plain symbol at a given time with a more complex combination of “previous", “current", and “next seen" symbols and

4. the merging of the lower frequency tags with the high frequency ones based on linguistic similarities (for tags with a very low accuracy, very few instances can be expected to occur, so an accurate CRF model could not be trained).

Order of CRF model – Regular Expression

In order to determine the best CRF order parameter and the regular expression have been used the default n-gram feature that replaces each plain symbol with the current one. Experiments with higher order than 2 have not been performed because models of such a high order cause out of memory errors due to the large model size. Table 1 illustrates that the best performance obtained by using the order-1 and a regular expression that grouping the PoS and the case.

Order	Regular Expression Modification	Evaluation based on Phrases	Evaluation based on Words
0	“Extract only PoS”	55.910	32.098
0	“Extract PoS and Case”	61.554	37.246
1	“Extract only PoS”	37.583	20.033
1	“Extract PoS and Case”	29.618	15.377
2	“Extract only PoS”	36.720	19.508
2	“Extract PoS and Case”	38.446	19.967

Table 1: Evaluation results for EL-EN in development corpus

N-Gram Features

Within the current implementation, the option of combining different input parameters such as tags and lemmas is also provided. This functionality represents n-gram features by replacing each plain symbol (corresponding to a single token) at a given time with a more complex combination of “previous", “current", and “next seen" symbols based on configured time slots. In order to determine the N-Gram and the corresponding regular expression that cause the optimum performance it is used the CRF of order 1 is used,

as determined from the previous step. Experiments with an order higher than 2 cannot be performed, because they cause out-of-memory errors. Table 2 illustrates that the best results are obtained for two distinct cases marked as bold.

N-Gram	Regular Expression Modification	Evaluation based on Phrases	Evaluation based on Words
[0]	"Extract only PoS"	37.583	20.033
[0]	"Extract PoS and Case"	29.618	15.377
[0,1]	"Extract only PoS"	38.997	20.590
[0,1]	"Extract PoS and Case"	38.977	20.590
[-1,0]	"Extract only PoS"	27.490	14.262
[-1,0]	"Extract PoS and Case"	30.080	15.672
[-1,0,1]	"Extract only PoS"	41.036	21.836
[-1,0,1]	"Extract PoS and Case"	46.414	24.712
[0,1,2]	"Extract only PoS"	51.992	2.869
[0,1,2]	"Extract PoS and Case"	59.363	33.508
[-2,-1,0]	"Extract only PoS"	37.450	19.508
[-2,-1,0]	"Extract PoS and Case"	42.961	23.115
[-2,-1,0,1,2]	"Extract only PoS"	69.854	39.115
[-2,-1,0,1,2]	"Extract PoS and Case"	80.810	46.459

Table 2: Evaluation results for EL-EN in development corpus

Merging of tags

The segmentation performance was evaluated for various experiments by merging low frequency tags with high frequency tags. The merging of tags was based on linguistic similarities. The main purpose of this modification was to further simplify the proposed model and thus attain a higher accuracy, since the total number of tags is decreased, effectively increasing the availability of training patterns. For instance, in the case of the Greek taggers, most foreign words (denoted as "Rg") can be seen to correspond to nouns. However, the frequency of "Rg" is approximately 20 over the 200 sentences. Thus, if all "Rg" tags are replaced by "No", the frequencies of the two tag categories are combined to provide a more comprehensive training set that leads to a more accurate model trained, in particular for the lower frequency "Rg" category. Table 3 illustrates that the best result are obtained for the merging of tag "DIG" with the tag "No[Nm]".

PMG performance for the various experiments was evaluated by using two distinct evaluators. The first one (the evaluator that is based on words) evaluates the number of words and phrase types that belong to the correct phrase against the number of the total words. The second one (the evaluator which is based on phrases) evaluates the number of correct phrases against the total number of phrases. In order to evaluate the PMG output it is first transformed to an appropriate format, which is then processed by the evaluators.

PC(Βρετανοί επιστήμονες) VC(θεωρούν) πως PC(ο χυμός ροδιού) VC(μπορεί) να PC(μας) VC(βοηθήσει) VC(να νικήσουμε) PC(το στρες) PC(στη δουλειά). (EL200-1)
PC(Οι αλλαγές) PC(στην παρούσα συμφωνία) VC(θα ισχύουν) PC(από τη στιγμή) PC(που) VC(θα ανακοινωθούν) PC(στον ιστότοπο). (EL200-2)
PC(Οι γνώσεις) PC(των παιδιών) VC(δεν μένουν) PC(περιορισμένες) PC(στα ίδια βιβλία) PC(επί χρόνια). (EL200-3)
PC(Ο λαός), PC(σ΄ αυτό το νοητικό και πραγματικό σχήμα) , VC(παίζει) PC(καταλυτικό ρόλο) . (EL200-4)

Figure 2: Optimal segmentation of the word evaluator

PC_Βρετανοί_επιστήμονες VC_θεωρούν πως PC_ο_χυμός_ροδιού VC_μπορεί να PC_μας VC_βοηθήσει VC_να_νικήσουμε PC_το_στρες PC_στη_δουλειά (EL200-1)
PC_Οι_αλλαγές PC_στην_παρούσα_συμφωνία VC_θα_ισχύουν PC_από_τη_στιγμή PC_που VC_θα_ανακοινωθούν PC_στον_ιστότοπο (EL200-2)
PC_Οι_γνώσεις PC_των_παιδιών VC_δεν_μένουν PC_περιορισμένες PC_στα_ίδια_βιβλία PC_επί_χρόνια (EL200-3)
PC_Ο_λαός, PC_σ΄_αυτό_το_νοητικό_και_πραγματικό_σχήμα,
VC_παίζει PC_καταλυτικό_ρόλο (EL200-4)

Figure 3: Optimal segmentation of the phrase evaluator

Phrasing Model Generator's Evaluation

Table 4 illustrates the segmentation accuracy for each language pair with available golden set by using the optimum configuration parameters as they obtained from the previous experiments. The segmentation accuracy is calculated by counting the number of phrases that have exactly the same segmentation as the golden segmentation to

N-Gram	Symbol Merging	Evaluation based on Phrases	Evaluation based on Words
[0]	Rg, DIG→No[xy], Distance<=4, Nm[x]→Aj[x], Pn[x]→no[x]	36.720	19.213
[0]	Rg→No[xy],DIG→Aj[xx],Vbmnpp→Aj[Ac], AsPpPa[x]→At[x] Distance<=4, Chooses Case from any PoS with case	36.122	19.311
[0]	Rg→No[xy], DIG→Aj[xy],Nm..Aj→Aj[xy], Nm..(!Aj)→No[xy], Vbmnpp[x]..Pv.. →Aj[x]	37.649	19.902
[0]	Rg→No[Ac]	28.287	14.951
[0]	DIG->No[Ac]	29.615	15.311
[0]	Nm[x]→Aj[x]	28.685	15.016
[0]	Pn[x]→No[x]	29.216	15.213
[0]	Rg→No[Nm]	28.752	15.115
[0]	DIG→No[Nm]	26.959	14.131
[0]	Rg→No[xy]	34.595	18.459
[0]	AsPpPa[x]->At[x]	27.025	14.230
[0]	Rg→No[xy], DIG→Aj[xy]	37.251	19.672
[-1,0]	DIG→No[Nm]	27.756	14.426
[-1,0]	AsPpPa[x]->At[x],	27.623	14.131
[-1,0]	DIG→No[Nm], AsPpPa[x]->At[x],	27.955	14.098

Table 3: Evaluation results with merging of tags for Greek-English

the total number of phrases. For all experiments, subsets of the parallel corpus are used in order to have two independent resources. More specifically, the first fifty sentences of each language pair are used as testing data, while the remaining sentences serve as training data.

LANGUAGE PAIR	TRAINING SET	TESTING SET	ACCURACY
Czech-German	51-228	1-50	81.36%
Czech-English	51-200	1-50	69.24%
German-English	51-164	1-50	84.47%
Greek-English	51-200	1-50	89.67%
Norwegian-German	51-197	1-50	84.47%

Table 4: Evaluation results for each language pair

2 References

[1] Lafferty, J., McCallum, A., & Pereira, F. 2001. Conditional Random Fields: Probabilistic Models for Segmenting and Labelling Sequence Data, Proceedings of ICML Conference, June 28-July 1, Williamstown, USA 282-289.

[2] Manning, C.D. & Schuetze, H. 1999. Foundations of Statistical natural Language Processing. MIT Press, Cambridge, Massachussetts.

[3] Wallach, H. M. 2004. Conditional Random Fields: An Introduction. CIS Technical Report, MS-CIS-04-21. 24 February 2004, University of Pennsylvania.

[4] Tambouratzis, G., Simistira, F., Sofianopoulos, S., Tsimboukakis, N. & Vassiliou, M. 2011. A resource-light phrase scheme for language-portable MT, Proceedings of the 15th International Conference of the European Association for Machine Translation, (eds. M. L. Forcada, H. Depraetere & V. Vandeghinste) 30-31 May 2011, Leuven, Belgium, pp. 185-192.

3 PMG's User Manual

Introduction

This is a mini guide on how to use the **Phrasing model generator (PMG)**. PMG supports two distinct operations. The first operation processes the output of Phrase Aligner Module to train a phrasing model for the SL of the specified language pair. The second operation makes use of the phrasing model established to parse any SL text input and split it into phrases in preparation for the translation process.

How to run PMG

Training a phrasing model (1st operation): PMG needs a number of arguments denoting the language pair for which a phrasing model is trained. If the required resources for the specific language pair are not available, the program terminates.

The command line arguments for invoking PMG have the following form:

```
pmg -train -lang [<srcLang>-<tgtLang>]
```

where

- `<srcLang>`: the source language denoted by the first two characters and
- `<tgtLang>`: the target language denoted by the first two characters.

Example: `pmg -train -lang DE-EN`

Training a phrasing model for the language pair German → English

Example: `pmg -train -lang EL-DE`

Training a phrasing model for the language pair Greek → German

The output of the PMG train operation is stored under the path `data/PMG/models/` and is named `model_language_pair.crf` (e.g. `model_DE-EN.crf` and `model_EL-DE.crf` respectively in the above examples).

Parsing a text input (2nd operation): PMG needs a number of arguments denoting the language pair, the text to be processed, the encoding and the name of the output file. If the required resources for the specific language pair are not available, the program terminates. The command line arguments for invoking PMG have the following form:

```
pmg -parse -lang [<srcLang>-<tgtLang>] -encoding [<encode>] \
    -input [<inputPathfile>] -output [<outputPathfile>]
```

where

- <srcLang>: the source language denoted by the first two characters,
- <tgtLang>: the target language denoted by the first two characters,
- <encode>: the encoding of the output file,
- <inputPathfile>: the path of the input file containing the unparsed SL text and
- <outputPathfile>: the path of the file with the output parsed text.

Example:

```
pmg -parse -lang DE-EN -encoding UTF-8 -input data/unparsedFile.xml \
    -output data/parsedFile.xml
```

This command results in the parsing of the text that is contained in the file “unparsedFile.xml" for the language pair German → English and finally storing the output (the parsed DE-side) into the file “parsedFile.xml", based on the phrasing of the EN side parser.

3.1 How to handle a language pair in PAM

Training a phrasing model (1st operation): PMG requires the following resources for each language pair: (a) the parsed sentences and (b) the configuration file. Each one of those resources is described below.

The parsed sentences: The parsed sentences that are created by the Phrase Aligner Module and are stored under the path /data/Corpora/<SL_LANG>-<TL_LANG>/parsed.xml. The parsed sentences are used only within the training mode for generating the phrasing model.

Example: The parsed sentences for the language pair German-English are stored under the path /data/Corpora/DE-EN/parsed.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<text>
  <sent id="1">
    <clause id="16" type="">
      <phrase id="12" type="PC">
        <word id="2" head="n" fhead="n" token="Innere" tag="ADJA.Pos.Nom.Sg.Fem" lemma="inner"/>
        <word id="3" head="n" fhead="n" token="und" tag="CONJ.Coord.-2" lemma="und"/>
        <word id="4" head="n" fhead="n" token="äußere" tag="ADJA.Pos.Nom.Sg.Fem" lemma="äußer"/>
```

```
      <word id="5" head="y" fhead="n" token="Sicherheit" tag="N.Reg.Nom.Sg.Fem" lemma="Sicherheit"/>
    </phrase>
    <phrase id="13" type="VC">
      <word id="6" head="n" fhead="y" token="sind" tag="VFIN.Sein.3.Pl.Pres.Ind" lemma="sein"/>
    </phrase>
    <phrase id="14" type="PC">
      <word id="7" head="n" fhead="n" token="zwei" tag="CARD" lemma="zwei"/>
      <word id="8" head="y" fhead="n" token="Seiten" tag="N.Reg.Nom.Pl.Fem" lemma="Seite"/>
    </phrase>
    <phrase id="15" type="PC">
      <word id="9" head="n" fhead="n" token="derselben" tag="PRO.Dem.Attr.-3.Gen.Sg.Fem"
          lemma="dieselbe"/>
      <word id="10" head="y" fhead="n" token="Medaille" tag="N.Reg.Gen.Sg.Fem" lemma="Medaille"/>
    </phrase>
      <word id="11" head="n" fhead="n" token="." tag="SYM.Pun.Sent" lemma="."/>
  </clause>
</sent>

<sent id="2">
  <clause id="24" type="">
    <phrase id="17" type="PC">
      <word id="2" head="n" fhead="n" token="Die" tag="ART.Def.Nom.Sg.Fem" lemma="der"/>
      <word id="3" head="y" fhead="n" token="EU" tag="N.Name.Nom.Sg.Fem" lemma="EU"/>
    </phrase>
    <phrase id="18" type="VC">
      <word id="4" head="n" fhead="y" token="muss" tag="VFIN.Mod.3.Sg.Pres.Ind" lemma="müssen"/>
    </phrase>
    <phrase id="20" type="PC">
      <word id="5" head="n" fhead="n" token="wirksame" tag="ADJA.Pos.Acc.Pl.Fem" lemma="wirksam"/>
      <word id="6" head="y" fhead="n" token="Maßnahmen" tag="N.Reg.Acc.Pl.Fem" lemma="Maßnahme"/>
    </phrase>
    <phrase id="19" type="VC">
      <word id="7" head="y" fhead="n" token="ergreifen" tag="VINF.Full.-2" lemma="ergreifen"/>
      <word id="8" head="n" fhead="n" token="," tag="SYM.Pun.Comma" lemma=","/>
      <word id="9" head="n" fhead="n" token="um" tag="CONJ.SubInf.-2" lemma="um"/>
    </phrase>
    <phrase id="22" type="PC">
      <word id="10" head="n" fhead="n" token="die" tag="ART.Def.Acc.Sg.Fem" lemma="der"/>
      <word id="11" head="y" fhead="n" token="Sicherheit" tag="N.Reg.Acc.Sg.Fem" lemma="Sicherheit"/>
    </phrase>
    <phrase id="23" type="PC">
      <word id="12" head="n" fhead="n" token="ihrer" tag="PRO.Poss.Attr.-3.Gen.Pl.Neut" lemma="ihr"/>
      <word id="13" head="y" fhead="n" token="Mitglieder" tag="N.Reg.Gen.Pl.Neut" lemma="Mitglied"/>
    </phrase>
    <phrase id="21" type="VC">
      <word id="14" head="n" fhead="n" token="zu" tag="PART.Zu" lemma="zu"/>
      <word id="15" head="y" fhead="n" token="gewährleisten" tag="VINF.Full.-2" lemma="gewährleisten"/>
    </phrase>
    <word id="16" head="n" fhead="n" token="." tag="SYM.Pun.Sent" lemma="."/>
  </clause>
</sent>
</text>
```

The configuration file: It contains the Spring framework' s parameters which defines (a) the regular expression, (b) the order of the CRF model, (c) the n-gram features and (d) the variation. In order to create a configuration file, the following steps need to be followed:

(1) Add the default header as illustrated in figure 4. The header is required for generating the declared java classes.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:util="http://www.springframework.org/schema/util"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
      http://www.springframework.org/schema/beans/spring-beans.xsd
      http://www.springframework.org/schema/util
      http://www.springframework.org/schema/util/spring-util.xsd">
```

Figure 4: The default header of Spring framework

(2) Add the default configuration parameters as illustrated in figure 5 for the class "pmg.AlgorithmBean" except for the parameters in bold typeface: the regular expression and the model order. The regular expression consists of two parts and their values are customized based on the SL and the modifications in terms of input tags. The following figure 5 illustrates a regular expression that modifies Greek tags by removing all parts except the PoS and the cases. The CRF order parameter sets the time window based on which the CRFmodel creates connections between the training sequence and the observed symbols. The following figure illustrates the use of a CRF model with order 1.

(3) Add the default configuration parameters as illustrated in figure 6 for the class "pmg.pipe.SentTransContext2FeatureVSequence"except the parameter "timeslots". This parameter combines different input parameters such as tags and lemmas to represent n-gram features by replacing each plain symbol at a given time with a more complex combination of "previous", "current", and "next seen" symbols. The following figure illustrates a n-gram that replaces each plain symbol with the "current" symbol.

(4) Add the default configuration parameters as illustrated in figure 7 for the class "cc.mallet.fst.CRF".

```
<bean id="algorithmBeanClass" class="ilsp.pmg.AlgorithmBean">
  <property name="featurePipe" ref="featurePipeClass"/>
  <property name="crf" ref="crfClass"/>
  <property name="trainer" ref="trainerClass"/>
  <property name="defaultStateName" value="myDefaultStateName"/>
  <property name="iterations" value="1000"/>
  <property name="regex"
    value="(^(at|aj|no|vbmnpp|aspppa).*(nm|ge|ac|da|vo)$)|
      (^(vb|abbr|date|ad|asppsp|cj|dig|pterm_p|rg).*$)|
      (^(nm|pn).*(..)(..)$)|(^(vb)(mnpp).*(xx)$)|(^(pt)(..)$)"/>
  <property name="replacement" value="$2[$3]$5$7[$8]$11$15"/>
  <property name="modelOrder">
    <array>
      <value>0</value>
      <value>1</value>
    </array >
  </property>
</bean>
```

Figure 5: The configuration for the class “pmg.AlgorithmBean“.

```
<bean id="featurePipeClass" class="ilsp.pmg.pipe.SentTransContext2FeatureVSequence">
  <property name="fieldDelimiter"><value>&#9;</value></property>
  <property name="expressions">
    <array>
      <bean class="ilsp.pmg.pipe.model.MultiGramExpression">
        <description>x[0,1]</description>
        <property name="endChar" value="not_found"/>
        <property name="endCharIndex" value="1"/>
        <property name="timeSlots">
          <list>
            <value>0</value>
          </list>
        </property>
      </bean>
    </array>
  </property>
</bean>
```

Figure 6: The configuration for the class “pmg.pipe.SentTransContext2FeatureVSequence"

```
<bean id="crfClass" class="cc.mallet.fst.CRF">
  <constructor-arg name="inputPipe">
    <ref local="featurePipeClass"/>
  </constructor-arg>
  <constructor-arg name="outputPipe">
    <null/>
  </constructor-arg>
</bean>
```

Figure 7: The configuration for the class “cc.mallet.fst.CRF"

(5) Add the default configuration parameters as illustrated in figure 8 for the class “cc.mallet.fst.CRFTrainerByLabelLikelihood".

```
<bean id="trainerClass" class="cc.mallet.fst.CRFTrainerByLabelLikelihood">
  <constructor-arg name="crf" type="cc.mallet.fst.CRF">
    <ref local="crfClass"/>
  </constructor-arg>
  <property name="gaussianPriorVariance" value="10.0"/>
</bean>
```

Figure 8: The configuration for the class “cc.mallet.fst. CRFTrainerByLabelLikelihood"

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:util="http://www.springframework.org/schema/util"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans.xsd
    http://www.springframework.org/schema/util
    http://www.springframework.org/schema/util/spring-util.xsd">
  <bean id="algorithmBeanClass" class="ilsp.pmg.AlgorithmBean">
    <property name="featurePipe" ref="featurePipeClass"/>
    <property name="crf" ref="crfClass"/>
    <property name="trainer" ref="trainerClass"/>
    <property name="defaultStateName" value="myDefaultStateName"/>
    <property name="iterations" value="1000"/>
    <property name="regex" value="(^(at|aj|no|vbmnpp|aspppa).*(nm|ge|ac|da|vo)$)|
      (^(vb|abbr|date|ad|asppsp|cj|dig|pterm_p|rg).*$)|
      (^(nm|pn).*(..)(..)$)|(^(vb)(mnpp).*(xx)$)|(^(pt)(..)$)"/>
    <property name="replacement" value="$2[$3]$5$7[$8]$11$15"/>
    <property name="modelOrder">
      <array>
        <value>0</value>
```

```
      <value>1</value>
    </array>
  </property>
</bean>

<bean id="featurePipeClass" class="ilsp.pmg.pipe.SentTransContext2FeatureVSequence">
  <property name="fieldDelimiter"><value>&#9;</value></property>
  <property name="expressions">
    <array>
      <bean class="ilsp.pmg.pipe.model.MultiGramExpression">
        <description>x[0,1]</description>
        <property name="endChar" value="not_found"/>
        <property name="endCharIndex" value="1"/>
        <property name="timeSlots">
          <list>
            <value>0</value>
          </list>
        </property>
      </bean>
    </array>
  </property>
</bean>

<bean id="crfClass" class="cc.mallet.fst.CRF">
  <constructor-arg name="inputPipe">
    <ref local="featurePipeClass"/>
  </constructor-arg>
  <constructor-arg name="outputPipe">
    <null/>
  </constructor-arg>
</bean>

<bean id="trainerClass" class="cc.mallet.fst.CRFTrainerByLabelLikelihood">
  <constructor-arg name="crf" type="cc.mallet.fst.CRF">
    <ref local="crfClass"/>
  </constructor-arg>
  <property name="gaussianPriorVariance" value="10.0"/>
</bean>
</beans>
```

The example above illustrates the configuration file for the language pair Greek-German.

The configuration file that is created with the above process must be stored under the path `/data/PMG/AppContext/` in an XML file. The XML file should be named as `appContext_<SL_LANG>-<TL_LANG>`.

Example: The configuration file for the language pair German-English is stored under the path `/data/PMG/AppContext/appContext_DE-EN.xml`

Parsing a text input (2nd operation): PMG requires the following resources for each language pair: (a) the configuration file that is created by the previous process and stored

under the filename /data/PMG/AppContext/appContext_<SL_LANG>-<TL_LANG>.xml and (b) the phrasing model that is created by the previous process and stored under the filename /data/PMG/Models/model_<SL_LANG>-<TL_LANG>.crf.