An MT Learning Environment for Computational Linguistics Students

Svetlana Sheremetyeva

Department of Computational Linguistics
Copenhagen Business School,
Bernhard Bangs Alle 17 B,
DK-2000, Denmark
lanaconsult@mail.dk

Abstract

This paper discusses the issue of suitability of software used for the teaching of Machine Translation. It considers requirements to such software, and describes a set of tools that have initially been created as developer environment of an APTrans MT system but can easily be included in the learning environment for MT training. The tools are user-friendly and feature modularity and reusability.

Introduction

A current trend in the teaching of Machine Translation to bridge the gap between theory and practice to a large extent consists of using on-line MT demos and/or commercial systems or other tools developed in the course of MT research. No matter that it has become possible only recently when in addition to the text books, computers and MT software have become an inherent part of MT learning environment; quite an experience has already been accumulated in developing an MT training task pool. Example exercises based on existing MT software can be found in practically every paper on the teaching of MT. See, for example, recent articles such as Pérez-Ortiz and Forcada (2001), Somers (2001), Clavier and Poudat 2001), Kenny and Way (2001), Blanc (2001).

Still it is not that easy nowadays to provide for students to have a hands-on MT experience. The survey on Tools and Techniques for Machine Translation Teaching (Balkan et al., 1997) concludes that using a working MT system in the classroom involves a huge amount of effort both in terms of getting funds for purchasing commercial MT systems and in terms of using them to the most advantage of MT students which is not always possible due to the "locked" character of commercial products.

This paper attempts to contribute to the problem by raising the issue of suitability of MT software for an MT student (and instructor). In other words, we believe that the MT learning environment will benefit from a kind of training software that was specially selected or designed for an MT student as the main user. From this perspective MT training software should be treated as any computer application for special purposes and thus should meet quite a number of requirements.

A massive collection of papers on human-computer interaction - see, for example, Dix et al. (1998), Beyer and Holtzblatt (1998), Hackos and Redish (1998) - attempt to bring some structure to the often chaotic application design process. Topics include human limitations, usability principles, interface design, models of the user, task analysis, and practical methods of gathering data about users and tasks. It is stressed that there are no minor issues in the field of interface design and even such common topics as error messages, toolbars, tabbed dialogues, icons and responsiveness should be well thought out.

In what follows we first attempt task and user analysis for an MT training tool as a special computer application for a Computational Linguistics student. We then illustrate our considerations with a brief

description of an APTrans application (see next paragraph) focusing on its developer environment that might be used as such training tool. At the end we discuss MT training issues that could be covered with the APTrans tool kit.

APTrans is an experimental MT system for translating patent claims as described in Sheremetyeva and Nirenburg (1999) that is currently under development. The current version of APTrans is a 32-bit Windows application developed to run in a number of operating environments: Windows 95/98/2000/NT. The facts that APTrans draws on domain knowledge but does not impose a sublanguage approach to MT, and that both the system and developer tools run on a personal computer, offer a wide range of training applications.

Desiderata for MT Training Software

Task and User analysis. Practically everyone involved in the teaching of MT recognizes what is very clearly formulated in Somers (2001): the teaching of Machine Translation takes different perspectives depending of student profiles, such as Computer Science, Computational Linguistics, Translation or Foreign Language Learning. The first two groups are often put in one category, which may be due to the fact that "historically, MT was probably the first non-numerical use of computers proposed" (Somers, op.cit). From those early days both Computer Science and Computational Linguistics grew into related but well established fields of their own with the focus on different issues. For the Computer Science students the focus is on high level programming skills in the most advanced programming languages, such as C, C+, C++, etc. For the Computational Linguistics students the emphasis is on MT *linguistic problems* and solutions. That means that if we want to train students in solving MT related *linguistic* problems it would be reasonable not to augment the difficulty of their tasks with programming matters. As for the content of training that should be covered by an MT training tool (or a tool kit) ideally it should be possible to use it for both

- illustrative purposes to familiarize students with weaknesses of MT software, by presenting them (in a very clear way) the results of every processing step for linguistic error analysis and
- "participation" purposes to let a student interfere with MT processing at every possible stage (lexicon and grammar acquisition, disambiguation, etc.) to influence the output of MT so as to immediately see the results of his/her updates.

It should also be possible to save traces of students' training activity for further work or classroom discussion.

User constraints. An MT training tool should meet general user requirements of making the program as easy as possible to install and to use despite its diverse and complex functionality. We should take into consideration human limitations and desire to automate tedious tasks such as typing, revising texts, propagating changes, etc. It is desirable to visualize the results of every step of the training procedure in the most "human" way. All user-computer communication should be done in the most natural way, i.e. in a natural language.

APTrans Overview

A prototype of the APTrans application has been described in Sheremetyeva and Nirenburg (1999) so we shall not deal with specifications of the application but rather with the description of the application and its developer environment from the angle of using it as part of MT learning environment.

APTrans is an experimental interactive MT system for translating patent claims between any pair of European languages. The model has been initially developed for English and Russian as both SLs and TLs but is readily extensible to other languages, - the Danish language is now being added to the system. For better understanding all examples in this paper are in English.

APTrans consists of i) a partially interactive SL text analysis module, ii) an automated transfer module and iii) an automated TL text generation module. The analysis module, in turn, includes a fully automated tagger and phrase chunker, and a

submodule of interactive predicate/case-role dependency analysis of SL text.

The system takes an SL claim text as input. After automatic tagging and chunking the interactive analysis module guides the user through a sequence of SL analysis procedures, as a result of which the system produces a set of internal knowledge structures in the form of predicate-argument templates filled with SL strings:

text::={ template}{template}*

template::={predicate-class predicate ((case-role)(case-role}*)

case-role::= (rank status value)

value::= {word tag}*

where *predicate-class* is a label of an ontological concept, *predicate* is a string corresponding to a predicate from the system lexicon, *case-roles* are *ranked* according to the frequency of their co-occurrence with each predicate in the training corpus, *status* is a semantic status of a case-role, such as agent, theme, place, instrument, etc., and *value* is a string which fills a case-role. *Tag* is a label, which conveys both morphological information (such as POS, number and inflection type) and semantic information, an ontological concept, (such as object, process, substance, etc.). For example, the tag Nf means that a word is a noun in singular (N), means a process (f), and does not end in -*ing*. This tag will be assigned, for example, to such words as *activation* or *alignment*. At present we use 23 tags that are combinations of 1 to 4 features out of a set of 19 semantic, morphological and syntactic features for 14 parts of speech. For example, currently the feature structure of noun tags is as follows:

Tag
[POS
[Noun

[object[plural, singular]

process[-ing, other[plural, singular]]

substance [plural, singular]

other [plural, singular]]]]]

The number of semantic classes (concepts) and case-roles is domain based and is rather small but can be easily augmented. A set of content templates filled with SL tagged strings is further transferred into a set of content templates filled with TL tagged strings. The latter are input into the APTrans generator that creates a tree of the text plan and outputs a TL claim text.

At the current stage of development the APTrans knowledge includes analysis, generation and transfer grammar, flexible depth *tagging lexicons* for tagging case-role fillers as mentioned above, and *deep (information-rich) lexicons* of predicates. This lexicon is the main part of the APTrans static knowledge, covers the lexical, semantic and syntactic knowledge, and is the basis of knowledge representation. The user can customize these lexicons. The architecture of APTrans is integrated with the development environment.

Developer Tools

The developer environment of the APTrans is a set of interactive tools created for a linguist developer. The development process of APTrans validated the effectiveness the tools and their integration into the system. The linguist is free to experiment with different kinds (and sizes) of lexical and grammatical knowledge to improve the MT output without extra programming effort. The APTrans development environment consists of lexicon and grammar acquisition tools with control and interactive interfaces for updating linguistic knowledge and tracing complete or partial processing in APTrans Analyzer, Transfer Module and Generator. Each of these main steps of APTrans processing is achieved by several lexical and structural phases every one of which is customized by an individual developer tool. It is impossible to detail every component of the APTrans developer environment in one paper so we will limit our description to those that have already been tested in MT training and will only briefly mention the others including the main interface that can also be used as part of the development environment.

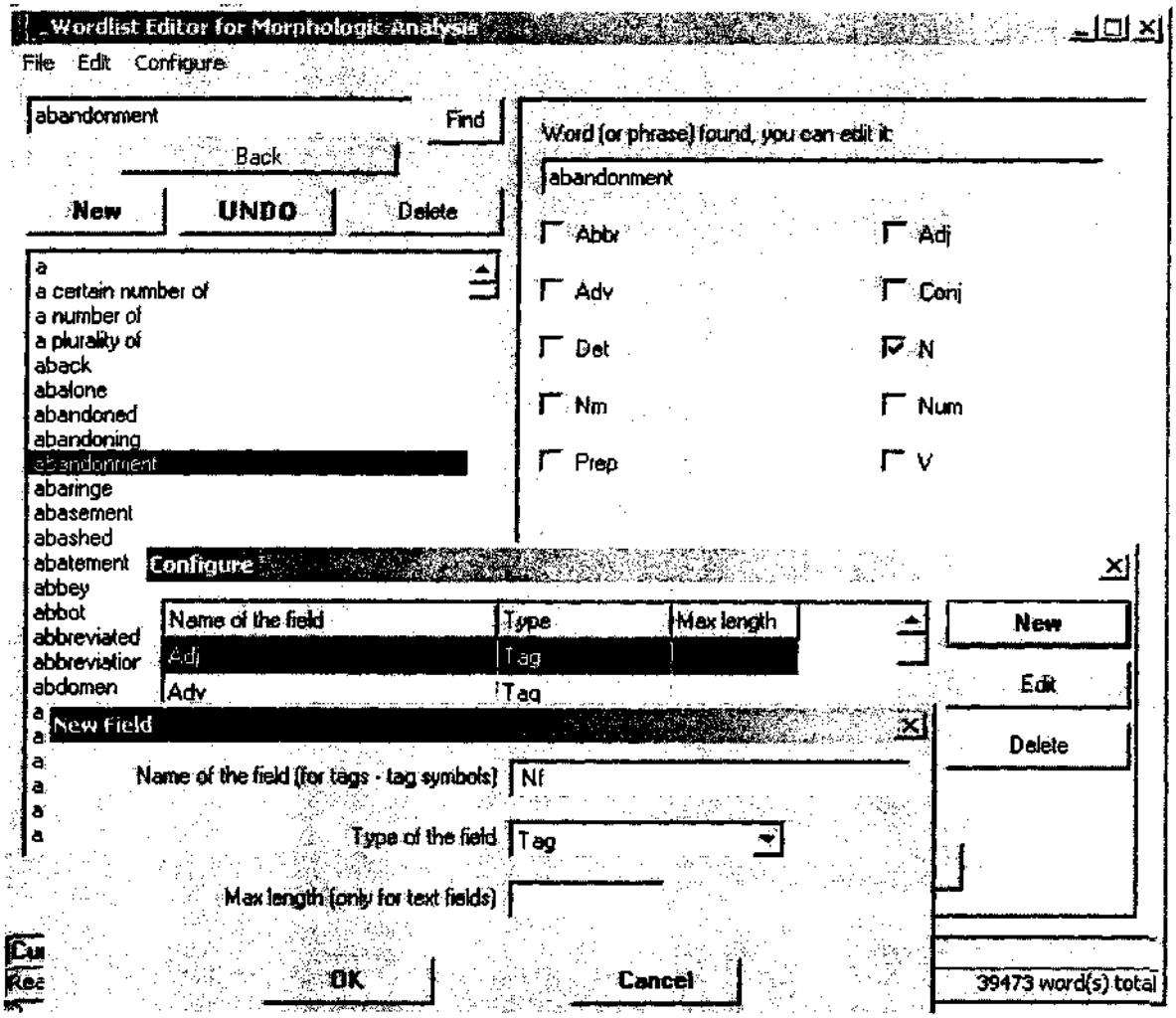

Figure 1. A screen shot of the interface of the tagging lexicon acquisition tool.

Tagging lexicon acquisition tool

Figure 1 shows a screen shot of the interface of the tagging lexicon acquisition tool when it is set for assigning only 10 shallow tags (just parts of speech). A new "deeper tag" Nf coding the features "noun", "singular" "no *-ing* ending", "process" is being added through a succession of pop-up windows. Selecting "Configure" in the main menu gives access to a window through which one can delete, edit or add new tags to the tool. For example, clicking on the "New" button in the first pop-up window gives access to the second window where one can type in any new tag. After clicking on the "OK" button the new tag will be displayed on the right pane of the interface next to its check box.

The basic principle for this tool is that the user can easily update both the list of words and tags making the latter larger or smaller in number and as "shallow" or "deep" as required. Any word in this lexicon can be easily "retagged".

The main menu in the right top corner of the interface has "File", "Edit" and "Configure" selections. The left pane shows an interactive "Find" window, the buttons "New", "Undo" and "Delete" (for updating a word list) and a scrollable list of lexical units, including multiword prepositions, adverbs, idiomatic phrases, etc. The right pane contains an interactive editing window and a number of tags next to check boxes. A checked box indicates a tag assigned to a highlighted word. Using selections under "File" in the main menu the user can import any lists of words either from external text files or from files in a special format where words were presorted according to their morphological and/or semantic features. In the latter case the tool assigns tags to the imported words automatically. The tool is

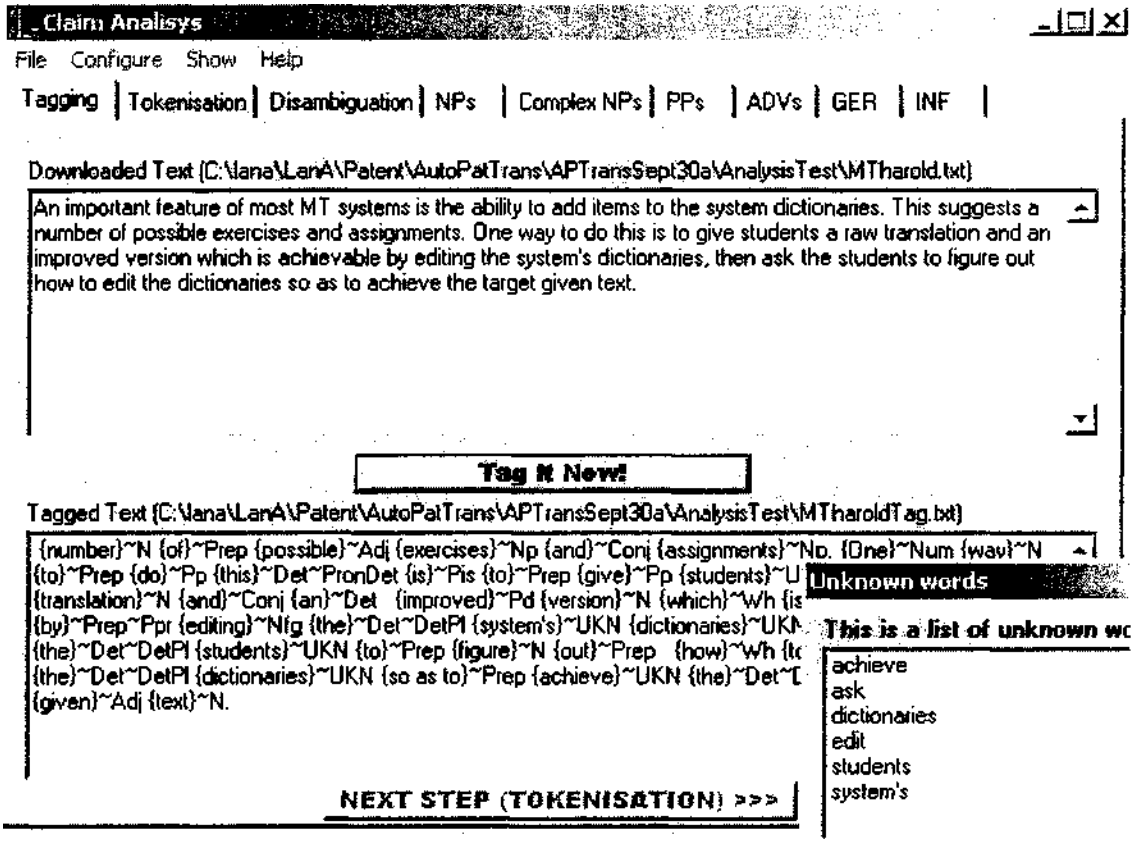

Figure 2. A screen shot of the interface of the analysis grammar acquisition tool with a trace of tagging. A list of new words is displayed in a pop-up window and can be automatically transferred to the tagging lexicon.

pipelined to the tagger, so as to automatically import words that were not recognized by the system after tagging a certain amount of text (see Figure 2). The coverage of the lexicon thus improves incrementally.

Depending upon the selection in the "Edit" menu the user can get either a full word list of the lexicon or sublists of words sorted by their suffixes, prefixes, or tags. It is also possible to get a list of untagged words and tag them through the tool interface. The user can also open a completely new (empty) lexicon to start lexicon acquisition from scratch. Any variant of tagging lexicon thus updated can be saved and re-opened for further work.

Analysis grammar acquisition tool

Figures 2 and 3 show screen shots of the developer interface for tracing and updating analysis process at different phases, - tagging, disambiguation and structural analysis of the input text. It contains the main menu "File", "Configure" "Show" and "Help", a set of bookmarks "Tagging", "Tokenization", "Disambiguation", and "Phrase Chunking", and a control screen divided into two windows with instruction buttons under each of them. The lower window of the screen traces the analysis phase corresponding to a certain bookmark, the upper window displaying the analysis trace of the previous phase, which makes it convenient to spot any errors. At the set up stage the upper window is empty and interactive. The user can either download a text from an external file or type it directly into the upper interactive window. With the help of the bookmarks or/and instruction buttons the user is guided through the various processing steps: tagging, tokenization, disambiguation, and chunking, the latter being a succession of processing steps itself starting with simple noun phrases followed by integrating them into complex

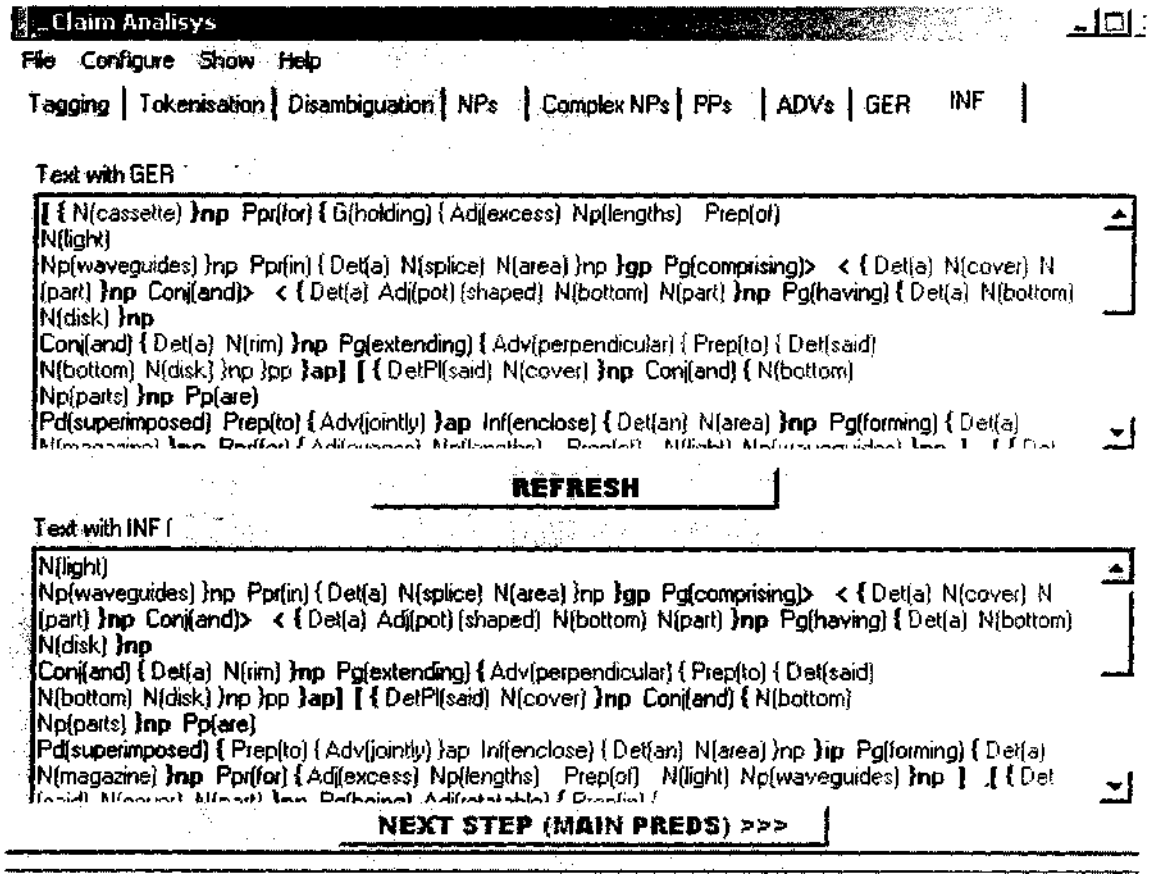

Figure 3. A screen shot of the interface of the analysis grammar acquisition tool with traces of chunking noun, prepositional, adverbial, gerundial and infinitival phrases.

noun phrases (those including prepositions and conjunctions), prepositional, adverbial, infinitival and gerundial phrases. Clicking on the selection "Save all" in the "File" menu saves traces of individual processing stages in different text files, the file names being automatically marked with suffixes corresponding to processing phases, for example, as follows:

Text, TextTag, textTok, TextNP, etc.,

which makes it very easy to compare different traces based on different rule sets when "brushing up" the grammar. What further makes the update procedure less tedious and time consuming is that processing traces are presented in a very illustrative form, e.g., in addition to brackets different colors are used for every type of phrase.

Different selections in the "Configure" menu of this tool open corresponding interactive compilers for writing or updating rules underlying certain analysis phases. Immediately after saving new rules an updated trace can be displayed in the lower part of the control screen by the "Refresh" button. Figure 4 shows the interface of the phrase-chunking compiler (all other compilers have similar functionalities). The right pane is a type-in area in which the user can write the rules in a very simple IF-THEN-ELSE-ENDIF formalism. The right pane is designed to support rule writing. This, on the one hand, makes the work less difficult and time consuming and, on the other hand, controls rule consistency and correctness of the formalism. The right pane

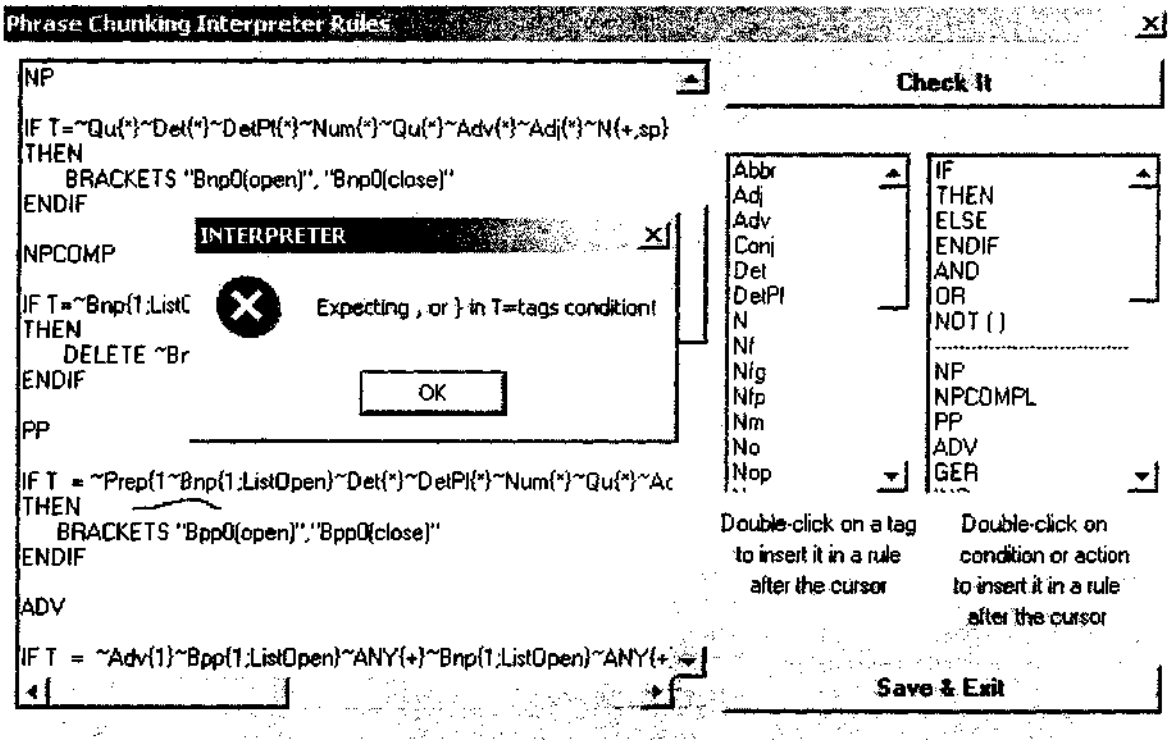

Figure 4. A screen shot of the compiler for acquisition phrase chunking rules.

of the compiler in Figure 4 contains two clickable menus, the menu of tags and the menu of expressions used in the rule formalism. It is enough to double-click on any of the selections in these menus to transfer it into the text of a rule. The tool would not let the user leave a compiler without the "Check it" button click. This button triggers a rule formalism check and in case of a mistake displays an error description message. After closing the message box on the "OK" button click the user will find the cursor right in the place where a correction should be made.

In spite of formalism simplicity the rules have a quite rich and flexible inventory of "right hand side" conditions (different for different compilers) that can provide for rather fine (vs. coarse) analysis. For example, in Disambiguator one can check a context within a five-string window with the tag in question in the middle. The context could be checked either in terms of tags and/or word strings and/or border marks from the tokenizer. It is also possible to check whether a context tag/word belongs to a certain list, etc. There is one more compiler that belongs to the Analyzer, the *Head* compiler, which acquires rules for determining agreement features between the predicate and its first case-role (the subject).

Predicate-argument grammar acquisition tool

As was mentioned above, the predicate lexicon is the most important part of the APTrans knowledge. Figure 5 shows the interface of the main tool for predicate acquisition. It is directly linked to the main application engine, which relies on linguistic knowledge contained in the lexicon. The interface allows for editing any of the lexicon fields, searching any word by its prefix or semantic class, and propagating changes from one field to another. For every language the interface has a built-in morphological generator that automatically generates all the predicate word forms, which are used in patent claims in that language. The interface has a standing menu of semantic classes and case-roles. The acquirer can customize the menu of semantic

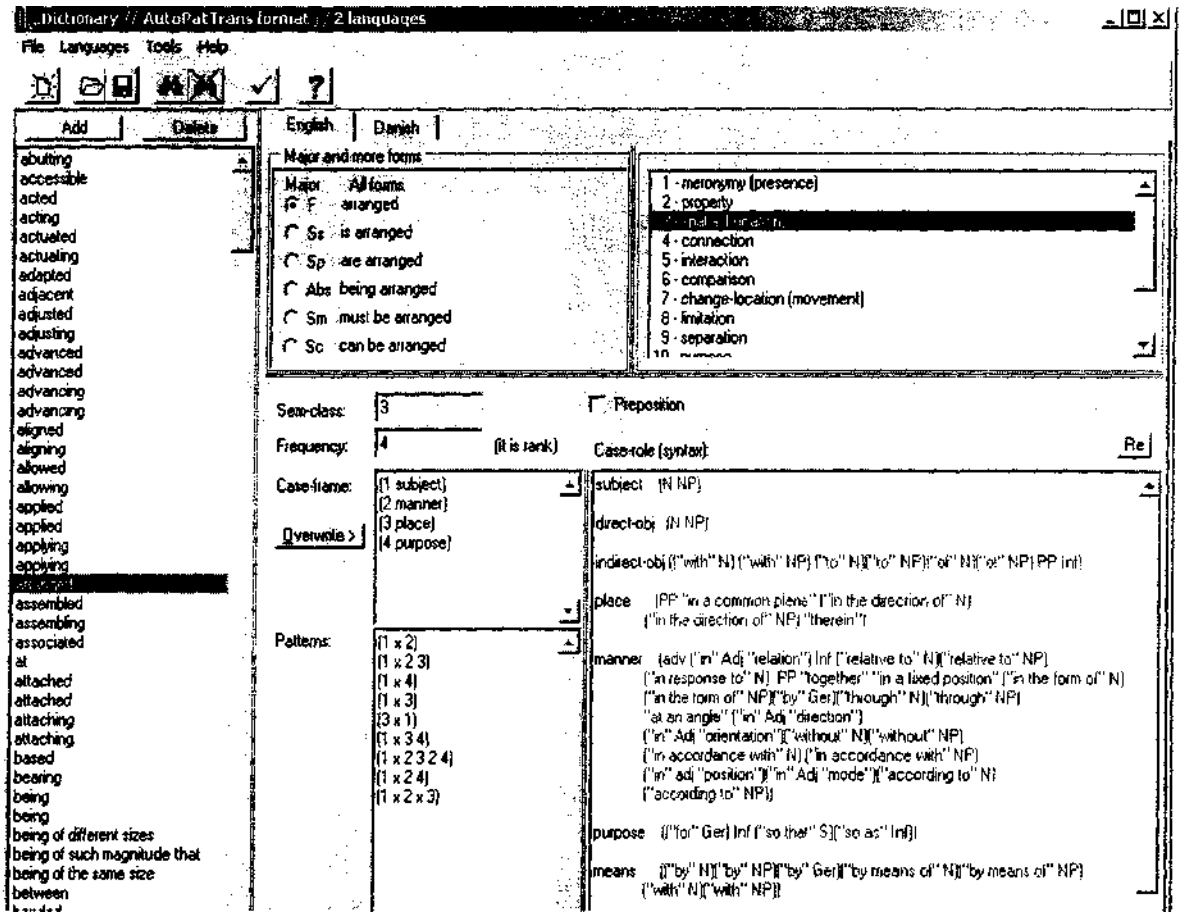

Figure 5. A screen shot of the interface of the predicate-argument grammar acquisition tool.

classes. Most of the fields of a new predicate entry are automatically filled with default fillers after a semantic class is acquired. The acquirer then checks them and edits if necessary. The interface is programmed so as to keep the acquirer "on the right road" by means of different hints and warning messages.

With this interface predicates can be either acquired one by one or automatically downloaded from the tagging lexicon acquisition tool.

The stage of determining predicate-argument structures that includes assigning a case-role status to every phrase chunked earlier (semantico-syntactic analysis) is currently done interactively through the *Main Application Interface* as described in Sheremetyeva and Nirenburg (1999) and draws heavily on the predicate lexicon. The traces of this phase of analysis can be controlled through the Main Application Interface.

Transfer compilers are still under development and we will skip them here, but generator compilers are reused from our previous application and can be mentioned.

The main tool for the MT generation trace is the *Generator Interface* that is used to control the correctness of the internal knowledge representation tree, linearization, grammaticalization, and final output text in a target language.

A number of compilers designed similarly to those of the analyzer acquire generation rules. *Linearizer* compiles the rules that specify the order of the words in the output TL text. *Grammaticalizer* is for writing cohesion rules and rules to assign morphological features to the TL lexemes, insert punctuation, and generate an MT output text.

APTrans in a Classroom

Actually, the description of tools given in the previous sections gives some hints about for what and how to use the APTrans application and its development environment for the teaching of Machine Translation to Computational Linguistics students. Extreme user-friendliness of the tool interfaces makes the software rather suitable for the classroom. First of all an instructor might show the application just as one more example of an MT system to familiarize students with the problems of MT software, linguistic error analysis, specificity of the sublanguage approach to MT, etc. Another way is to use APTrans developer tools to participate in building an MT system. One might either create a toy MT system or concentrate on solving particular MT problems. For example, based on the tagging lexicon acquisition tool and analysis compilers, exercises can be developed, using a specially designed test suit, to investigate the problem of coverage and knowledge necessary for disambiguation. A student can experiment with changing (inventing) tags to see whether a "deeper" description of lexical units gives better resolution. Compilers can also be used to teach students to write formal (programmable) grammar descriptions, etc.

Conclusions

We suggested using the APTrans application and its developer tools as part of Machine Translation learning environment. The tools described in the paper are mainly targeted to Computational Linguistics students and do not require programming skills so that students could concentrate on linguistic problems of MT. It looks like it is linguistic issues that now, with the advent of advanced computer technologies, are the main obstacle in developing high quality MT systems. The APTrans MT learning environment is extremely user-friendly and has the features of modularity and reusability. The tools can be used for different languages.

Acknowledgments. Thanks to Victor Raskin and Katrina Triezenberg for their contribution to the presentation of this paper.

References

Balkan, L., D. Arnold and L. Sadler (1997) Tools and Techniques for Machine Translation Teaching: A Survey. http://clwww.essex.ac.uk/group/projects/MTforTeaching/index_l.html

Beyer, H. and K. Holtzblatt (1998) *Contextual Design: Defining Customer-Centered Systems.* Morgan Kaufmann: San Francisco, CA.

Blanc, E. (2001) An Interactive Hypertextual Environment for MT Training. *MT Summit VIII Workshop on Teaching Machine Translation,* Santiago de Compostela, Spain, pp. 51-55.

Clavier, V. and C. Poudat (2001) Teaching Machine Translation in non Computer Science Subjects: Report of An Educational Experience within the University of Orleans, *MT Summit VIII Workshop on Teaching Machine Translation,* Santiago de Compostela, Spain, pp. 19-23.

Dix, A, J. Finlay, G. Abowd, and R. Beal (1998) *Human-Computer Interaction* (2nd Edition). Prentice Hall Europe: London.

Hackos, J. T. and J. C. Redish (1998) *User and Task Analysis for Interface Design.* Wiley: New York.

Kenny, D. and A. Way (2001) Teaching Machine Translation & Translation Technology: A Contrastive Study, *MT Summit VIII Workshop on Teaching Machine Translation,* Santiago de Compostela, Spain, pp. 13-17.

Pérez-Ortiz J. A. and M. L. Forcada (2001) Discovering MT strategies beyond Word-for-Word Translation: A laboratory assignment. *MT Summit VIII Workshop on Teaching Machine Translation,* Santiago de Compostela, Spain, pp. 57-60.

Sheremetyeva, S. and S. Nirenburg (1999) Interactive MT as support for non-native language authoring. *Proceedings of MT Summit VII,* Singapore, pp. 324-330.

Somers, H. (2001) Three perspectives on MT in the classroom. *MT Summit VIII Workshop on Teaching Machine Translation,* Santiago de Compostela, Spain, pp. 25-29.