Learning Micro-Planning Rules for Preventative Expressions*

Keith Vander Linden†
Information Technology Research Institute
University of Brighton
Brighton BN2 4AT, UK
email: knvl@itri.brighton.ac.uk

Barbara Di Eugenio
Computational Linguistics
Carnegie Mellon University
Pittsburgh, PA, 15213 USA
email: dieugeni@andrew.cmu.edu

Abstract

Building text planning resources by hand is time-consuming and difficult. Certainly, a number of planning architectures and their accompanying plan libraries have been implemented, but while the architectures themselves may be reused in a new domain, the library of plans typically cannot. One way to address this problem is to use machine learning techniques to automate the derivation of planning resources for new domains. In this paper, we apply this technique to build micro-planning rules for preventative expressions in instructional text.

1 Introduction

Building text planning resources by hand is time-consuming and difficult. Certainly, much work has been done in this regard; there are a number of freely available text planning architectures (e.g., Moore and Paris, 1993). It is frequently the case, however, that while the architecture itself can be reused in a new domain, the library of text plans developed for it cannot. In particular, micro-planning rules, those rules that specify the low-level grammatical details of expression, are highly sensitive to variations between sub-languages, and are therefore difficult to reuse.

When faced with a new domain in which to generate text, the typical scenario is to perform a corpus analysis on a representative collection of the text produced by human authors in that domain and to induce a set of micro-planning rules guiding the generation process in accordance with the results. Some fairly simple rules usually jump out of the analysis quickly, mostly based on the analyst's intuitions. For example, in written instructions, user actions are typically expressed as imperatives. Such observations, however, tend to be gross characterisations. More accurate micro-planning requires painstaking analysis. In this paper, for example, the micro-planner must distinguish between phrasing such as "Don't do *action-X*" and "Take care not to do *action-X*". Without analysis, it is far from clear how this decision can best be made.

Some form of automation would clearly be desirable. Unfortunately, corpus analysis techniques are not yet capable of automating the initial phases of the corpus study (nor will they be for the foreseeable future). There are, however, techniques for rule induction which are useful for the later stages of corpus analysis and for implementation.

In this paper, we focus on the use of such rule induction techniques in the context of the micro-planning of preventative expressions in instructional text. We define what we mean by a preventative expression, and go on to describe a corpus analysis in which we derive three features that predict the grammatical form of such expressions. We then use the C4.5 learning algorithm to construct a micro-planning sub-network appropriate for these expressions. We conclude with an implemented example in which the technical author is allowed to set the relevant features, and the system generates the appropriate expressions in English and in French.

*This work is partially supported by the Engineering and Physical Sciences Research Council (EPSRC) Grant J19221, by BC/DAAD ARC Project 293, and by the Commission of the European Union Grant LRE-62009.

†After September 1, Dr. Vander Linden's address will be Department of Mathematics and Computer Science, Calvin College, Grand Rapids, MI 49546, USA.

2 Preventative Expressions

Preventative expressions are used to warn the reader not to perform certain inappropriate or potentially dangerous actions. The reader may be told, for example, "Do not enter" or "Take care not to push too hard". Both of these examples involve negation ("do *not*" and "take care *not*"). Although this is not strictly necessary for preventative expressions (e.g., one might say "stay out" rather than "do not enter"), we will focus on the use of negative forms in this paper, using the following categorisation:[1]

- negative imperatives proper (termed *DONT* imperatives) — These are characterised by the negative auxiliary *do not* or *don't*, as in:

  (1) Your sheet vinyl floor may be vinyl asbestos, which is no longer on the market. *Don't sand it or tear it up* because this will put dangerous asbestos fibers into the air.

- *NEVER* imperatives — These are characterised by the use of the negative adverb *never*, as in:

  (2) Whatever you do, *never go to Vienna if you are on a diet.*

- other negative imperatives (termed *neg-TC* imperatives) — These include *take care* and *be careful* followed by a negative infinitival complement, as in the following examples:

  (3) To book the strip, fold the bottom third or more of the strip over the middle of the panel, pasted sides together, *taking care not to crease the wallpaper sharply at the fold.*

  (4) If your plans call for replacing the wood base molding with vinyl cove molding, *be careful not to damage the walls* as you remove the wood base.

3 Corpus Analysis

In terms of text generation, our interest is in finding mappings from features related to the *function* of these expressions, to those related to their grammatical *form*. Functional features include the semantic features of the message being expressed, the pragmatic features of the context of communication, and the features of the surrounding text being generated. In this section we will briefly discuss the nature of our corpus, and the function and form features that we have coded. We will conclude with a discussion of the inter-coder reliability. A more detailed discussion of this portion of the work is given elsewhere (Vander Linden and Di Eugenio, 1996).

3.1 Corpus

The corpus from which we take all our coded examples has been collected opportunistically off the internet and from other sources. It is 4.5 MB in size and is made entirely of written English instructional texts. As a collection, these texts are the result of a variety of authors working in a variety of contexts.

We broke the corpus texts into expressions using a simple sentence breaking algorithm and then collected the negative imperatives by probing for expressions that contain the grammatical forms we were interested in (i.e., expressions containing phrases such as *don't*, *never*, and *take care*). The grammatical forms we found, 1283 occurrences in all, constitute 2.7% of the expressions in the full corpus. The first line in Table 1, marked "Raw Grep", indicates the quantity of each type.

We then filtered the results. When the probe returned more than 100 examples for a grammatical form, we randomly selected around 100 of those returned, as shown in line 2 of Table 1 (labelled "Raw Sample"). We then removed those examples that, although they contained the desired lexical string, did not constitute negative imperatives (e.g., "If you *don't* like the colors of the file, ..., use Binder to change them."), as shown in line 3, labelled "Final Coding".

The final corpus sample is made up of 279 examples, all of which have been coded for the features to be discussed in the next two sections. Table 2 also shows the relative sizes of the various types of instructions in the corpus as well as the number of examples from this sample that came from each type.

[1] Horn (1989) gives a more complete categorisation of negative forms.

	DONT		NEVER	Neg-TC			
	don't	*do not*		*take care*	*make sure*	*be careful*	*be sure*
Raw Grep	417	385	108	21	229	52	71
Raw Sample	100	99	108	21	104	52	71
Final Coding	78	89	40	17	3	46	6
	167		40	72			

Table 1: Distribution of negative imperatives

Instruction type	*Corpus size*	*# of preventatives*
Recipes	1.7M	83
Do-it-yourself	1.26M	99
Di Eugenio's thesis[2]	336K	69
Software instructions	264K	0
Administrative forms	317K	9
Other	565K	19
Totals	4.5M	279

Table 2: Distribution of examples from sample

3.2 Form

Because of its syntactic nature, the form feature coding was very robust. The possible feature values were: **DONT** — for the *do not* and *don't* forms discussed above; **NEVER**, for imperatives containing *never*; and **neg-TC** — for *take care*, *make sure*, *be careful*, and *be sure* expressions with negative arguments. The two authors agreed on their coding of this feature in all cases.

3.3 Function Features

We will now briefly discuss three of the function features we have coded: INTENTIONALITY, AWARENESS, and SAFETY. We illustrate them in turn using α to refer to the prevented action and using "agent" to refer to the reader and executer of the instructions.

Intentionality: This feature encodes whether or not the writer believes that the agent will consciously adopt the intention of performing α:

CON is used to code situations where the agent intends to perform α. In this case, the agent must be aware that α is one of his or her possible alternatives.

UNC is used to code situations in which the agent doesn't realize that there is a choice involved (cf. Di Eugenio, 1993). It is used in two situations: when α is totally accidental, or the agent may not take into account a crucial feature of α.

Awareness: This feature captures whether or not the writer believes that the agent is aware that the consequences of α are bad:

AW is used when the agent is aware that α is bad. For example, the agent may be told "Be careful not to burn the garlic" when he or she is perfectly well aware that burning things when cooking them is bad.

UNAW is used when the agent is perceived to be unaware that α is bad.

Safety: This feature captures whether or not the author believes that the agent's safety is put at risk by performing α:

BADP is used when the agent's safety is put at risk by performing α.

NOT is used when it is not unsafe to perform α, but may, rather, be simply inconvenient.

[2]Note that we used a number of examples from Di Eugenio's thesis (1993) which were included as excerpts. In this table we include only an estimate of the full size of that portion of the corpus.

3.4 Inter-coder reliability

Each author independently coded each of the features for all the examples in the sample. The percentage agreement for each of the features is shown in the following table:

feature	*percent agreement*
form	100%
intentionality	74.9%
awareness	93.5%
safety	90.7%

As advocated by Carletta (1996), we have used the Kappa coefficient (Siegel and Castellan, 1988) as a measure of coder agreement. For nominal data, this statistic not only measures agreement, but also factors out chance agreement.

If $P(A)$ is the proportion of times the coders agree, and $P(E)$ is the proportion of times that coders are expected to agree by chance, K is computed as follows:

$$K = \frac{P(A) - P(E)}{1 - P(E)}$$

There are various ways of computing $P(E)$ according to Siegel and Castellan (1988); most researchers agree on the following formula, which we also adopted:

$$P(E) = \sum_{j=1}^{m} p_j^2$$

where m is the number of categories, and p_j is the proportion of objects assigned to category j.

The mere fact that K may have a value k greater than zero is not sufficient to draw any conclusion, however, as it must be established whether k is significantly different from zero. There are suggestions in the literature that allow us to draw general conclusions without these further computations. For example, Rietveld and van Hout (1993) suggest the correlation between K values and inter-coder reliability shown in the following table:

Kappa Value	*Reliability Level*
.00 – .20	slight
.21 – .40	fair
.41 – .60	moderate
.61 – .80	substantial
.81 – 1.00	almost perfect

For the form feature, the Kappa value is 1.0, indicating perfect agreement. The function features, which are more subjective in nature, engender more disagreement among coders, as shown by the K values in the following table:

feature	*K*
INTENTIONALITY	0.46
AWARENESS	0.76
SAFETY	0.71

According to this table, therefore, the AWARENESS and SAFETY features show "substantial" agreement and the INTENTIONALITY feature shows "moderate" agreement. We have coded other functional features as well, but they have either not proven as reliable as these, or are not as useful in text planning.

In addition, Siegel and Castellan (1988) point out that it is possible to check the significance of K when the number of objects is large; this involves computing the distribution of K itself. Under this approach, the three values above are significant at the .000005 level.

4 Automated Learning

The corpus analysis results in a set of examples coded with the values of the function and form features. This data can be used to find correlations between the two types of features, correlations, which, in text generation, are typically implemented as decision trees or rule sets mapping from function features to forms.

In this study, we used 179 coded examples as input to the learning algorithm. These are the examples on which the two authors agreed on their coding of all the features. The distribution of the grammatical forms in these examples is shown in the following table:

form	*frequency*
DONT	100
Neg-TC	57
NEVER	22

The learning algorithm used these examples to derive a decision tree which we then integrated into an existing micro-planner.

4.1 Data Mining

We have used Quinlan's C4.5 learning algorithm (1993) in this study; this algorithm can induce either decision trees or rules. To provide a more convenient learning environment, we have used Clementine (1995), a tool which allows rapid reconfiguration of various data manipulation facilities, including C4.5. Figure 1 shows the basic control stream we used for learning and testing decision trees. Data is input from the **split-output** file node on the left of the figure and is passed through filtering modules until it reaches the output modules on the right. The two **select** modules (pointed to by the main input node) select the examples reserved for the training set and the testing set respectively. The upper stream processes the training set and contains a **type** module which marks the main syntactic form (i.e., DONT, NEVER, or Neg-TC) as the variable to be predicted and the AWARENESS, SAFETY, and INTENTIONALITY features as the inputs. Its output is passed to the C4.5 node, labelled **mform**, which produces the decision tree. We then use two copies of the resulting decision tree, represented by the diamond shaped nodes marked with **mform**, to test the accuracy of the testing and the training sets.

One run of the system, for example, gave the following decision tree:

```
awareness = AW: NEG-TC
awareness = UNAW:
|    intention = CON: DONT
|    intention = UNC:
|    |    safety = BADP: NEVER
|    |    safety = NOT: DONT
```

This tree takes the three function features and predicts the DONT, NEVER, and Neg-TC forms. It confirms our intuitions that *never* imperatives are used when personal safety may be endangered (coded as safety="BADP"), and that Neg-TC forms are used when the reader is expected to be aware of the danger that may arise (cf. Vander Linden and Di Eugenio, 1996). It accurately predicts the grammatical form of 74.5% of the 161 training examples, and 83.3% of the 18 testing examples.

Because there are relatively few training examples in our coded corpus, we have also performed a 10-way cross-validation test.[3] None of the derived trees in this test were remarkably different from the one just shown, although they did order the INTENTIONALITY and AWARENESS features differently. The average accuracy of the learned decision trees on the testing sets was 75.4%.

Note that although this level of accuracy is better than 55.9%, the score achieved by simply selecting DONT in all cases, there is still more work to be done. The current features must be refined, and more features may be need to be added. We are currently experimenting with a number of possibilities. Note also that we have not distinguished between the various sub-forms of DONT and Neg-TC shown in Table 1; this will require yet more features.

Clementine can also "balance" the input to C4.5 by duplicating training examples with underrepresented feature values. We used this to increase the number of NEVER and Neg-TC examples to match the number of DONT examples. Ultimately, this reduced the accuracy of the learned trees to 68.0% in a cross-validation test. The resulting decision trees tended not to include all three features.

4.2 Integration

Because it is common for us to rebuild decision trees frequently during analysis, we implemented a routine which automatically converts the decision tree into the appropriate KPML-style system networks with their associated choosers, inquiries, and inquiry implementations (Bateman, 1995). This makes the network compatible with the DRAFTER micro-planner, a descendent of IMAGENE (Vander Linden and Martin, 1995). The conversion routine takes the following inputs:

- the applicable language(s) — C4.5 produces its decision trees based on examples from a particular language, and KPML is capable of being conditionalised for particular languages. Thus, we may perform separate corpus analyses of a particular phenomenon for various languages, and learn separate micro-planning trees;

[3] A cross-validation test is a test where C4.5 breaks the data into different combinations of training and testing sets, builds and tests decision trees for each, and averages the results (Clementine, 1995).

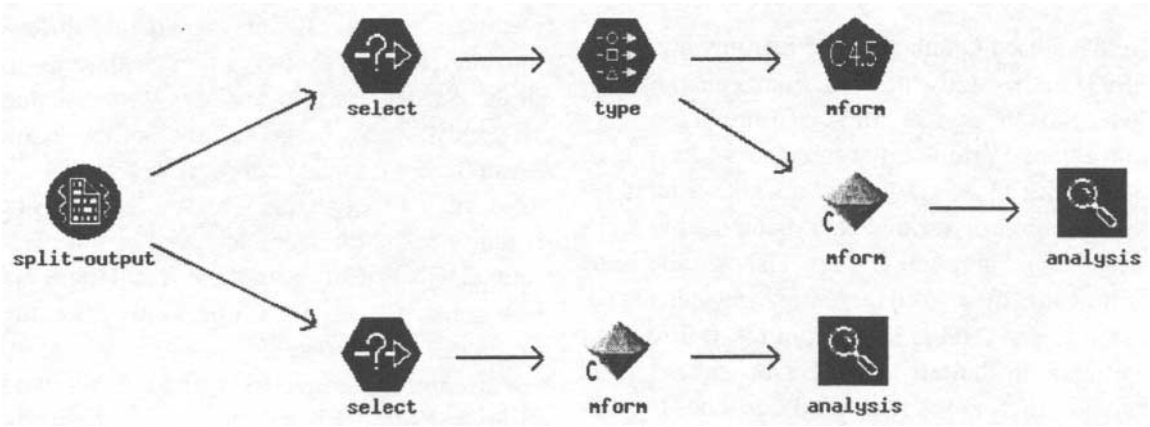

Figure 1: The Clementine learning environment

- the input feature(s) — The sub-network being built must fit into the overall categorisations of the full micro-planner, and thus we must specify the text functions that would trigger entry to the new sub-network;
- the decision tree itself;
- a feature-value function — To traverse the new sub-network, the KPML inquiries require a function that can determine the value of the features for each pass through the network;
- grammatical form specifications — The sub-network must eventually build sentence plan language (SPL) commands for input to KPML, and thus must be told the appropriate SPL terms to use to specify the required grammatical forms;
- an output file name.

For our example, the system sub-network shown in Figure 2 is produced based on the decision tree shown above.[4] It is important to note here that although the micro-planner is implemented as a systemic resource, the machine learning algorithm is no respecter of systemic linguistic theory. It simply builds decision trees. This gives rise to three distinctly non-systemic features of these learned networks:

1. The realisation statements are included only at the leaf nodes of the network. We have built no intelligent facility for decomposing the realisation statements and filtering common realisations up the tree.
2. The learning algorithm will freely reuse systems (i.e., features) as various points in the tree. This did not happen in Figure 2, but occasionally one of the features is independently used in different sub-trees of the network. We are forced, therefore, to index the system and feature names with integers to disambiguate.
3. There is no meta-functional distinction in the network, but rather, all the features, regardless of their semantic type, are included in the same tree.

The sub-network derived in this section was spliced into the existing micro-planning network for the full generation system. As mentioned above, this integration was done by manually specifying the desired input conditions for the sub-network when the micro-planning rules are built. For the preventative expression sub-network, this turned out to be a relatively simple matter. DRAFTER's model of procedural relations includes a *warning* relation which may be attached by the author where appropriate. The micro-planner, therefore, is able to identify those portions of the procedure which are to be expressed as warnings, and to enter the derived sub-network

[4] Only the systems are shown in the KPML dump given in Figure 2. The realisation statements, choosers, inquiries, and inquiry implementations are not shown.

Figure 2: The micro-planner system network derived from the decision tree

appropriately. This same process could be done with any of the other procedural relations (e.g., purpose, precondition). This assumes, however, the existence of a core set of micro-plans which perform the procedural categorisation properly; these were built by hand. We have only just begun to experiment with the possibility of building the entire network automatically from a more exhaustive corpus analysis.

5 A DRAFTER Example

Given the corpus analysis and the learned system networks discussed above, we will present an example of how preventative expressions can be delivered in DRAFTER, an implemented text generation application. DRAFTER is a instructional text authoring tool that allows technical authors to specify a procedural structure, and then uses that structure as input to a multilingual text generation facility (Paris and Vander Linden, 1996). The instructions are generated in English and in French.

To date, our domain of application has been manuals for software user interfaces, but because this domain does not commonly contain preventative expressions (see Table 2), we have extended DRAFTER's domain model to include coverage for do-it-yourself applications. Although this switch has entailed some additions to the domain model, DRAFTER's input and generation facilities remain as they were.

5.1 Input Specification

In DRAFTER, technical authors specify the content of instructions in a language independent manner using the DRAFTER specification tool. This tool allows the authors to specify both the propositional representations of the actions to be included, and the procedural relationships between those propositions. Figure 3 shows the DRAFTER interface after this has been done. We will use the procedure shown there as an example in this section, details on how to build it can be found elsewhere (Paris and Vander Linden, 1996).

The INTERFACE and ACTIONS panes on the left of figure 3 list all the objects and actions defined so far. These are all shown in terms of a pseudo-text which gives an indication, albeit ungrammatical, of the nature of the action. For example, the main goal, "repair device", represents the action of the reader repairing an arbitrary device. This node may be expressed in any number of different grammatical forms depending upon context.

The WORKSPACE pane shows the procedure, represented in an outline format. The main user goal of repairing the device is represented by the largest, enclosing box. Within this box, there is a single method, called "Repair Method" which details how the repair should be done. There are three sub-actions: consulting the manual, unplugging the device, and removing the cover. There is also a warning slot filled with the action "[reader] damage service cover". This indicates that the reader should avoid damaging the service cover.[5]

Neither the propositional nor the procedural information discussed so far specify the three features needed by the decision network derived in the previous section (i.e., intentionality, awareness, and safety). At this point, we see no straightforward way in which they could be determined automatically (see Ansari's discussion of this issue (1995)). We, therefore, rely on the author to set them manually. DRAFTER allows authors to set generation parameters on individual actions using a dialog box mechanism. Figure 4 shows a case in which the author has marked the following four features for the warning action "damage service cover":

[5] Actually, this could also be interpreted as an *ensurative* warning, meaning that the reader should make sure to damage the service cover (although this is clearly nonsensical in this case). We have not yet analysed such expressions and thus do not support them in DRAFTER.

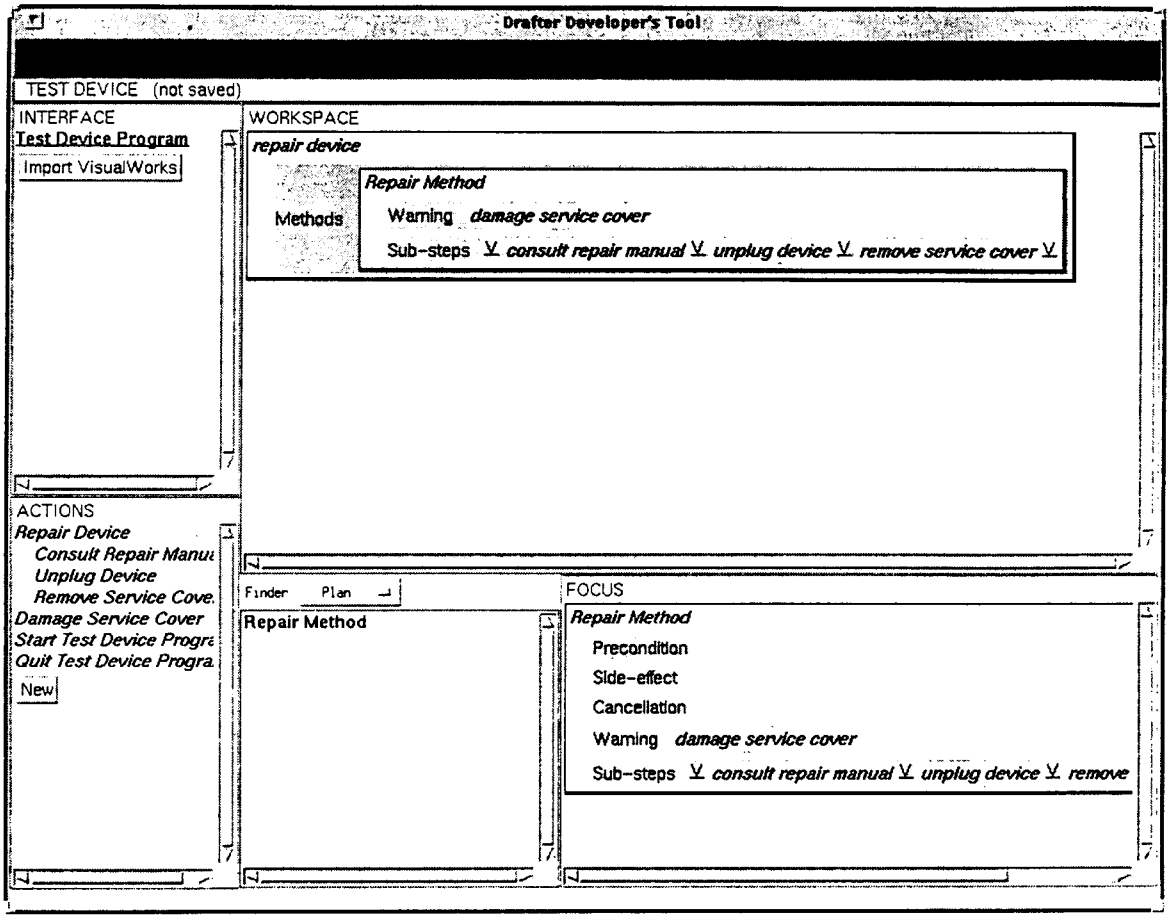

Figure 3: DRAFTER screen with the procedural structure for the example

- The action is to be prevented, rather than ensured;
- Performing the action would result in inconvenience, but not in personal danger;
- The user is likely to do the action accidentally, rather than consciously;
- The user is likely to be aware that performing the action would create problems;

5.2 Text Generation

Once the input procedure is specified, the author may initiate text generation from any node in the procedural hierarchy. When the technical author generates from the root goal node in Figure 3, for example, the following texts are produced:

English:

To repair the device
1. Consult the repair manual.
2. Unplug the device.
3. Remove the service cover.

Take care not to damage the service cover.

French:

Réparation du dispositif
1. Se reporter au manuel de réparation.
2. Débrancher le dispositif.
3. Enlever le couvercle de service.

Éviter d'endommager le couvercle de service.

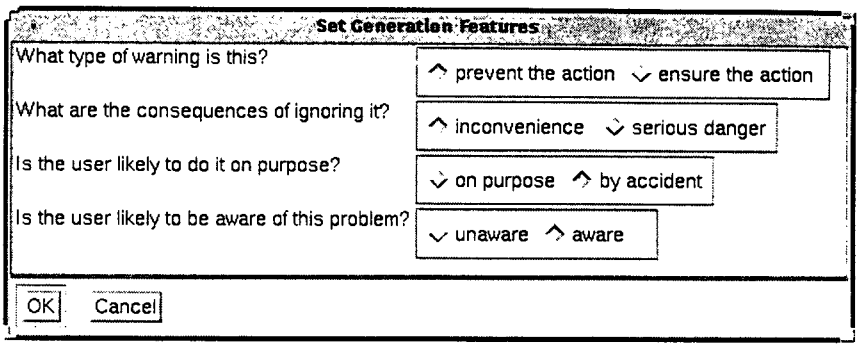

Figure 4: The DRAFTER dialog box for setting the local parameters

Note that the French version employs *éviter* (*avoid*) rather than the less common *prendre soin de ne pas* (*take care not*). This is possible because the French text is produced by a separate micro-planning sub-network. This sub-network was not based on a corpus study of French preventatives, but rather was implemented by taking the learned English decision tree, modifying it in accordance with the intuitions of a French speaker, and automatically constructing French systems from that modified decision tree. Clearly, a corpus study French of preventatives is still needed, but this does show DRAFTER's ability to make use of KPML's language conditionalised resources.

Were we to replace the warning with other sorts of warnings, the expression would also change according to the learned micro-planning network. If authors, for example, wish to prevent the reader from performing the action of dismantling the frame of the device, and they decide that the reader is unaware of this danger, that the action is consciously performed and not unsafe, DRAFTER produces the following text:

> Do not dismantle the frame.
> Ne pas démonter l'armature.

If authors wish to prevent the reader from disconnecting the ground connection, and they decide that the reader is unaware of this danger, that the action would be unconsciously performed, and that the consequences are indeed life-threatening, DRAFTER produces the following text:

> Never disconnect the ground.
> Ne jamais déconnecter la borne de terre.

6 Conclusion

In this paper we have discussed the use of machine learning techniques for the automatic construction of micro-planning sub-networks. We demonstrated this for the case of preventative expressions in instructional text.

We noted that because the automatic derivation of useful, well-defined features for corpus analysis is beyond the current state of the art, the painstaking process of corpus analysis must still be performed manually. As an example of how this can be done, we presented an analysis of English preventative expressions. We intend to continue this part of the work by addressing more preventative forms, addressing ensurative forms, and by extending the analysis to other languages.

Although the analysis cannot be fully automated, we noted that the derivation of decision networks from coded corpus examples can. This greatly simplifies the tasks of building and testing text planning resources for new domains. We intend to continue this part of the work by applying the technique to larger portions of the planning resources.

Acknowledgements

The authors wish to acknowledge valuable discussions with Tony Hartley, Xiaorong Huang, Adam Kilgarriff, Cécile Paris, Richard Power, and Donia Scott, as well as detailed comments from the anonymous reviewers.

References

Ansari, D. (1995). Deriving procedural and warning instructions from device and environment models. Master's thesis, Department of Computer Science, University of Toronto.

Bateman, J. A. (1995). KPML: The KOMET-Penman (Multilingual) Development Environment. Technical report, Institut für Integrierte Publikations- und Informationssysteme (IPSI), GMD, Darmstadt. Release 0.8.

Carletta, J. (1996). Assessing agreement on classification tasks: the kappa statistic. *Computational Lingustics*, 22(2). to appear.

Clementine (1995). *Clementine User Guide, Version 2.0.* Integral Solutions Limited.

Di Eugenio, B. (1993). A Study of Negation in Instructions. In *The Penn Review of Linguistics, Volume 17.*

Di Eugenio, B. (1993). *Understanding Natural Language Instructions: A Computational Approach to Purpose Clauses.* PhD thesis, University of Pennsylvania. also available as IRCS Report 93-52.

Horn, L. R. (1989). *A Natural History of Negation.* University of Chicago Press, Chicago.

Moore, J. D. and Paris, C. L. (1993). Planning text for advisory dialogues: Capturing intentional and rhetorical information. *Computational Linguistics*, 19(4):651–694.

Paris, C. and Vander Linden, K. (1996). Drafter: An interactive support tool for writing multilingual instructions. *IEEE Computer.* to appear.

Quinlan, J. R. (1993). *C4.5: Programs for Machine Learning.* Morgan Kaufmann.

Rietveld, T. and van Hout, R. (1993). *Statistical Techniques for the Study of Language and Language Behaviour.* Mouton de Gruyter.

Siegel, S. and Castellan, Jr., N. J. (1988). *Nonparametric statistics for the behavioral sciences.* McGraw Hill.

Vander Linden, K. and Di Eugenio, B. (1996). An empirical study of negative imperatives in natural language instructions. In *Proceedings of the 16th International Conference on Computational Linguistics,* August 5–9, Copenhagen, Denmark. To appear.

Vander Linden, K. and Martin, J. H. (1995). Expressing local rhetorical relations in instructional text: A case-study of the purpose relation. *Computational Linguistics*, 21(1):29–57.