[From: *CLAW '96*, Universiteit Leuven]

Controlled Language Support for Perkins Approved Clear English (PACE)

Shona Douglas and Matthew Hurst
Language Technology Group
Human Communication Research Centre
University of Edinburgh
S.Douglas,Matthew.Hurst@edinburgh.ac.uk

Abstract

We present an account of our approach to developing Controlled Language specifications and Controlled Language checking software for customized applications. A structured approach combines document flow analysis, corpus analysis, and negotiation with relevant stakeholders. Modular reuse of previous Controlled Language specifications is a key ingredient, allowing relatively simple tailoring of specifications for different functional text types at a level beyond that of terminology. The approach is illustrated using our experiences applying it to Perkins Approved Clear English (PACE) to produce a computational checking system.

1 Introduction

In the Language Technology Group at Edinburgh University's Human Communication Research Center, we have been working with the Technical Publications department, of Perkins International Limited, based in Peterborough, England, on computational support for writing in the controlled language PACE (Perkins Approved Clear English). PACE was developed on the Caterpillar model, and consists primarily of a single wordlist plus ten very general rules of writing ([5]). In this paper, we present the general approach we are developing to the construction of controlled language checkers, and illustrate this from our current experience of building a checker for PACE.

2 Costs and Benefits of Controlled Language

Claimed advantages of Controlled Language (CL) in the document life-cycle include:

- Improved comprehensibility.
- Easier post-processing and reuse of documents, such as for translation.
- A measurable index of document quality.
- Uniformity of documents across an organisation.

However, various problems stand in the way of using Controlled Language to achieve these advantages.

- It is hard to author documents in a Controlled Language.

  Firstly, the need to conform to explicit rules of CLs disrupts highly automatized human sentence processing functions.

  Second, the form in which CLs are defined (e.g., wordlists, general construction rules, general stylistic rules), and the media in which they are often presented (textual manuals) make it hard for a writer to find out whether a particular sentence conforms, and how to alter it if it does not.

  Alteration is a more complex operation than simple replacement via word list lookup. For example, verb to noun changes required by the AECMA rules concerning part of speech entail complete restructuring of sentences in order to accommodate the alteration of word category: *number* cannot be used as a verb, as in

  > *Number the components.*

  and should be replaced by *give X numbers.* This sort of restructuring places a relatively high burden on writers.

- It is hard to evaluate conformance of a document to a Controlled Language.

  As for the previous problem, this is in large measure due to the form in which CLS are defined and presented.

  Partly this is an 'indexing' problem; given a sentence, it is hard to know what part of the definition of a Controlled Language to look up to assess the sentence's conformance. For instance, when the above example requires *number* not to be used as a verb, this seems to contradict the general stylistic rule of AECMA "Let the verb do the action", by introducing the weak verb *give.*

  However, it is not just a matter of finding or making consistent the rulings somewhere in the specification. No commonly used CL definition exhaustively defines the language it covers in the sense that a computational grammar does. Lexicon or wordlist-based specifications (as used by PACE) make verification easy, but don't address the problem of how to generate suitable alterations; moreover they even more clearly underspecify the actual requirements on acceptable text. Use of a check-list approach ([7]) may help, but is cumbersome.

- It is hard to evaluate a Controlled Language itself.

  Perkins report a cut in translation costs of between 50% and 70% with use of PACE ([5]); there have been similar claims for human comprehensibility. A CL like AECMA Simplified English (SE), which goes beyond wordlists, may produce many possible interactions between rules in difference contexts, but even so they clearly underspecify the complexity of the actual language that results from proper use of the CL and which should be taken into account in evaluating it. Wordlist based CLs are even less clearly related to the resulting language. There are dauntingly many variables in an adequately explicit specification of a Controlled Language.

  While it might be possible to evaluate the quality of a given document conforming to a specific CL *(pace* the preceding point about the difficulty of identifying such a document), this does not allow us to say anything about the effect of particular elements in the definition of the CL, some of which may impose burdens on the writer disproportionate to their benefits to the reader. Little empirical work has yet been carried out which supports this intuition (but see [3], this volume, for recent work). One of the difficulties is that, like most complex tasks, producing high quality text is a multiple constraint satisfaction problem ([9]), and individual rules if followed slavishly are likely to produce inferior results.

- It is hard to customize a Controlled Language to the specialised requirements of particular applications.

The AECMA SE standard, like some other CLs, includes provisions for the introduction of new terminology, in a way that makes changes of domain relatively modular in their effect on the CL. This is possible because terminology is an area in which writers and domain experts have good vocabularies and well-defined meta-concepts. Technical thesauri are relatively common and well-understood tools of many trades (e.g., [1]), and it is easy to define the circumstances in which terms should be used in referring to concepts and objects. However, changes in the genre or document type are much more problematic, because they affect aspects of language structure much less easy to pin down than referring expressions. Procedural manuals and maintenance manuals for the same artefacts, for example, may use the same terminology, but quite different constructions reflecting different functions of the text. People do not have good shared intuitions for *describing* text functions, or the way in which such functions are fulfilled by different constructions and formulations. The use of individual lexical stipulations to effectively define much of the allowable language, as in the PACE definition, means that functional units and related constructions are defined only implicitly, through their lexical components, so that changing the specification to reflect new functional requirements is next to impossible.

Our aim is to develop and test a framework for the design of Controlled Languages and Controlled Language Checkers (CL(C)) which will address some of these problems, and permit the maximum reuse of resources consistent with solutions tailored to specific needs. It is clear that computational tools, which 'index' off the text itself, offer great scope for improvement, but to create them, we need definitions of CL that are suitably expressed to be translated into appropriate computational form. So far, we have concentrated on procedures for developing modular specifications of Controlled Languages suitable for instantiating our software checker shell.

3 Overall methodology for CL(C) construction

In order to define a Controlled Language and associated checker for a new application, we go through the following procedure. The overall outputs of the procedure are

- a definition of a Controlled Language;
- an instantiated CL Checker.

This is treated as a requirements engineering process; we are developing a structured approach which can be thought of as an application-specific specialisation of the KADS[1] modelling approach ([6]), in particular in its use of a set of generic task models to be adapted for the particular CL(C) task in hand.

In the rest of this section, this procedure is outlined in the abstract, then in the subsequent sections each phase of the procedure will be illustrated with examples from our work with Perkins.

3.1 Organizational, Application, and Task Models

In KADS, the organizational model identifies functions, tasks, and bottlenecks in the target organization, while the application model specifies the function the introduced system will have and the problem it will solve; the task model specifies a decomposition of the function.

For CL(C), we distinguish two aspects of the target organization to analyse. The fundamental one is the identification of document processing tasks, and document flows and stores and their ownership. In particular, for a particular target document type, we seek to identify 'upstream'

[1] "[originally] "Knowledge Analysis and Documentation System". Later, other interpretations were given to this acronym... .Currently, most people use it as a proper noun." [6], p.xi

influences on document quality, and 'downstream' requirements for document quality. These include

- Downstream use of documents as input to automatic text processing systems such as machine translation (MT).
- Downstream use of documents for (possibly second-language) readers of varying types.
- Upstream document creation and reuse practices.
- Upstream editing and quality control practices, including any existing textual artefacts such as wordlists or controlled language definitions.

Secondly we want an account of what procedures are in place to modify or extend any quality control artefacts, and facilities that are available to discuss and negotiate the requirements for downstream quality.

Obtaining the process model is fairly standard systems analysis work, making use of appropriate combinations of a number of generic task model components such as: **write; reuse**; **edit**; **translate; monitor.**

The application model defines the problem the system is to solve and the function of the system in the new state of the organisation. Our application is relatively fixed; however, additional questions of organizational structure can be considered here, and the possible need for new practices to supplement the use of a computational checker. As far as the task model is concerned, it is more or less a first-level decomposition of the way in which the system will carry out its function, and for the moment, it boils down to whether we offer just a checking system or a checking system plus a text-function template authoring facility.

3.2 The Co-operation Model

In KADS, the Co-operation Model describes the interactions between the system and external agents, and any interaction between 'internal agents'. In it, we seek to identify where in the overall task system we intend to intervene, what kind of checker we intend to construct. Since we are designing for reuse of our existing checker shell where possible, rather than from scratch, there are some restrictions. However, the checker shell is implemented on a client-server model coupled with the use of SGML input and output, and this allows for great flexibility in dealing the range of possible interactions between external and internal agents, as well as providing a clean and portable system.

For user interfaces, we have abstract specifications of a range of typical interaction requirements, which can be mapped to specific implementations in, e.g., menu and dialogue systems of a word processor. These include the distinction between batch and real-time checker operation, batch and interactive error reporting, and the various types of integration with text processing systems for replacement of text which may be necessary. Building the co-operation model involves specifying which of these interaction styles will be appropriate.

Also specified in the co-operation model are the information requirements of the system users, who can be thought of as carrying out the role of editor, whether or not they also originate the text. This includes consideration of whether the system is to be used for training (in which case detailed examples may be appropriate, indexed off particular detected errors) or for swift checking by writers who are already familiar with its operation and for whom a brief reminder will suffice.

The design and implementation of a particular checking application must, then

- Establish the requirements of the user in terms of current practices, e.g., the need to integrate checker functionality with existing document production tools.
- Realise the user requirements through customization of current tools.

- Realise interaction between internal agents, i.e., retrieving and interpreting the marked up output from the server.

3.3 The Expertise Model

Our version of KADS expertise modelling should be thought of as a process of redescribing or reformulating the practices of the target organization in such a way that our checker shell can be instantiated to treat the required coverage and errors. We don't use the knowledge representation language of KADS, since linguistic notations are more suitable for the task.

The expertise model has four layers in KADS, of which we currently only explicitly consider the first two.[2]

3.3.1 The domain layer

The domain layer in KADS is described as static knowledge describing concepts, relations, and structures in the domain. In our domain, expertise is primarily linguistic, and the domain model has two main parts:

- the core controlled sublanguage(s);
- a taxonomy of particular errors to be checked for.

The model of the core controlled sublanguage is developed as a negotiated process between the CL(C) developer and client. First, a description of existing language practices is produced, based on analysis of existing proofed texts. The aim is to produce a description that covers existing document types and sublanguage(s), expressed in terms of a layered model.

- The top layer relates to functional units of text, such as instructions, warnings, descriptions, etc; we consider a text type to be made up of an appropriate mixture of these.
- The layer below relates to sentence-level constructions such as verb subcategorization frames.
- The layer below that relates to terminology and lexis.

We then analyse the relationships between this language description and and existing document quality control instruments such as wordlists, rules, and editing and monitoring practices.

The results are discussed with the client in conjunction with existing sets of rules for good writing practice, with the aim of identifying confusing or ambiguous usages, or unnecessary proliferation of forms for particular functional units. In a sense, it is a knowledge elicitation process, by means of which we make the bridge between the kind of representation of a CL that we require to build a computational checker, and the generation of examples of possible allowed and disallowed usages which are suitable for discussion with the users.

From this, using existing linguistic resources such as computational grammars and heuristics relating constructions to functional text units, we develop a computational model of the language suitable for instantiation in our checker shell. At this stage, downstream requirements such as the needs of MT systems are factored in to the definition, possibly in consultation with external organizations.

The taxonomy of errors is developed from unproofed client texts, where possible, and from the proofed texts after the definition of the core CL. In this way, constructions found in existing documents, but not chosen to be in the CL, are identified for treatment in terms of error rules in the inference layer along with previously identified errors found by comparing pre- and post-editing versions of documents.

These procedures are summarized in Figure 1.

[2] We ignore the task and strategic layers.

3.3.2 The inference layer

Inferences in this application are mappings between errors in the error taxonomy and the corresponding CL preferred forms. This forms a specification for the checker program. Since we treat this process as an instantiation of existing general checking facilities, developing this model means expressing the mappings in terms of the rule types in our checker shell:

- **adjacency**—character-level specifications of allowable combinations of, e.g., spaces, punctuation, and other characters.
- **illegal words**—identifying illegal words can be made sensitive to parse context, so that a word can be found an error only in certain parts of speech.
- **unification**—typically, errors of agreement that can be expressed as feature clashes in unification rules, which can be used to detect errors by relaxation and generate replacements by feature modification.
- **insertion** and **deletion** of constituents.
- **positive errors:** pattern based recognition and associated parse tree transformation to generate replacements.

(See [2] for a more detailed discussion of a similar error taxonomy.)

The error detection/repair system as implemented is capable of detecting the error types listed above, as well as generating various responses.

The simplest form of response to an error is marking the error and its context through location in the text. Above this is the use of canned text to describe the error. For example, the ability to detect agreement errors through the use of relaxed unification ([2]) is coupled with the use of some suitable phrase to provide a meaningful description. The highest level of inference is to provide a full set of possible repairs covering any errors found in a suitable unit of text, for now the sentence.

Generating repairs involves, at the system level, implementing suitable recovery strategies and at the CL level, providing the required information. This information can generally be found in the computational model adopted to describe the grammar and lexicon, together with relaxations over the CL definition ([4]). Other information may be required for purposes of indexing such as the *root* of each word being held in the lexicon. Repairing errors due to relaxed unification, for example, can be accomplished through the use of root based lexical look up and a replacement process which maintains consistency in the otherwise correct sentence. Grammatical errors which require transformation of the sentence structure are repaired through the use of pattern and target parse descriptions together with possible root based lexical alterations. For example, the description of a certain form of passive may be marked as a positive grammatical error. This description will be accompanied by the pattern parse and a target parse. These parse descriptions map the appropriate constituents of the 'ill-formed' sentence to the correct analysis thereby generating the active form of the sentence (modulo necessary feature modifications).

To sum up, we identify three layers of responses to potential errors:

1. Location.
2. Canned responses, e.g. "Agreement violation"
3. A set of possible replacements.

Each level provides an increasing amount of information for the user. The apparatus required for generating these responses includes:

- Controlled Language definition in the form acceptable to the error checking engine, i.e., PATR-like grammar rules and a lexicon.

- Additional information describing relaxations of this model.
- Specification of repair information plus additional indexing information, e.g., specifying illegal words and their legal replacements.
- Mechanisms for generating consistent and correct repairs, e.g., parse tree transformations.

4 Creating a PACE Checker

In this section we describe some of our experiences so far in applying the methodology to the creation of a checker for the Perkins Engines CL PACE.

4.1 Organizational Model

The basic structure of the document production process with which we are concerned is shown in Figure 2. Site 1 is the Perkins main technical publications department in Peterborough. Site 2 is a relatively new addition to the organisation, at a factory in Shrewsbury, which has its own documentation department for its own products. Both sites are supposed to base their document quality control on the PACE definition, which consists of a single wordlist and a short list of general writing rules. Xerox Language Technology Centre in Slough provides translation services for both these document paths.

Our most detailed information relates to the Peterborough office, where PACE originated. The documentation department there is closely knit, and the use of the PACE wordlist is supplemented during and after training by the close monitoring of output by one skilled and experienced supervisor.

Documents at Site 1 are generally produced by heavy reuse of existing documents, since most work is updating documentation for new models of substantially similar engines. The homogeneity of the task has meant that the PACE wordlist, unlike, for example, AECMA, has required no provision for modularity even at the terminology level.

It is clear that simple adherence to the PACE wordlist hugely underdetermines the desired (and obtained) documentation result, and that a large cultural transmission process is involved in its successful use at the Peterborough site. The ten general rules of writing are almost never consulted in practice.

Problems arise in the extension of PACE to Site 2 at Shrewsbury because the cultural transmission process is dependent on physical co-presence, and in its absence the wordlist itself (as shown in Figure 2, the only real point of transmission) is an inadequate embodiment of the required practices.

The documentation tasks and subject domains dealt with at Site 2 are also sufficiently different from those for which PACE was developed that the monolithic structure of the wordlist is proving difficult to work with (all modification and extension procedures go through Site 1). Additionally, further acquisitions and diversifications will shortly worsen the heterogeneity of the document types and subject domains that need to be accommodated under the umbrella of PACE.

Xerox Language Technology Centre provides translation services for Perkins manuals. Since Perkins pay only for new material to be translated, they have a large interest in making the material they submit to Xerox as uniform as possible. The requirements of the MT system have been cited as the major motivation for PACE. Negotiations to modify the single word basis of PACE to incorporate phrasal terms are currently in train, initiated by Xerox to improve the performance of their system.

4.2 Application and Task Model

The problem we hope to solve is the fact that the structure of PACE fails to capture the more construction based and functional aspects of the actual sublanguages successfully converged on at Site 1 and hence is not a suitable vehicle for transmitting this convergence to Site 2. Additionally, the requirements of diverging document types and subject domains suggest that a more modular specification would be useful.

We plan to do this by fitting a checker system into the same organizational position as the PACE wordlist in Figure 1 — that is, shared between Site 1 and Site 2. This will embody not only the existing explicit PACE specification, but a modular version of a more comprehensive, empirically derived, description of current approved practices, to support more easily modified and customized specifications for approved document types and subject domains.

As far as the task model is concerned, we currently aim to supply a checker, not a template tool.

4.3 The Co-operation Model

The technical publications department at Perkins has expressed a requirement for batch use of checking facilities, after writing is complete. Automatic or mediated replacement would be ideal, but the document handling and production system they use is a specialised DOS-based one, which would be difficult to integrate with the client-server architecture of our checker. However, it is intended that all Perkins sites will migrate to Windows software, although the choice of text processing tool has not yet been made. In order to permit piloting of the checker application, the current agreement is that our existing interface in Word for Windows will be used on a trial basis.

Figure 3 shows the Word interface. It operates on a 'user-led' interaction basis, whereby a whole document or document section is sent to be checked, and any errors are marked up in the text with hot-spots. The user can then browse the document, in the course of normal revision and editing, and initiate the kind of error dialogue seen in Figure 3 by double-clicking the hotspot.

This is to be extended to provide navigation via summary lists.

5 Expertise Model

5.1 Domain model

Existing Language Model The language model for existing language practices at Perkins is currently under development. For this we use SGML-based corpus analysis tools which allow automatic, semi-automatic, and fully human operations on SGML marked up texts, with interfaces in XEmacs. For instance, to produce information about verb subcategorization frames, we use a tagging interface which allows the user to insert appropriate tags for parts of syntactic categories and grammatical roles of verb arguments, and collate and count the results. The use of this tool is illustrated in Figure 4. The screen dump shows that verb number 41, *included,* is the current verb; once the menu item NP has been chosen, the text *Data and dimensions* will be marked up as an NP, related by its VerbID attribute to verb 41, and another menu offered from which the Role SUBJECT can be chosen.

Different parameterizations of this interactive tool are being used to

- mark and count functional units of text;
- mark and count verb subcategorization frames and relate them to the functional units in which they occur.

Automatic versions are being used to

- automatically mark up occurrences of words from the PACE wordlist, and words not in the PACE wordlist.
- tag for part of speech to help investigate restrictions on allowable PoS ambiguity.

Automatic functions are carried out using the NSL tools for simplified handling of SGML streams ([8]).

5.2 Inference model

Care must be taken in providing the relevant information to the system. For example, the prescription of full relatives and proscription of reduced relatives could be implemented in at least two ways.

1. Specification of the relative grammar rule plus the annotation providing the relaxation (i.e., deleted initial constituent).
2. Specification of a positive error rule with an appropriate transformation introducing the missing relative pronoun.

The decision over which implementation to use must be based on considerations of efficiency with respect to the parsing/recognition system. In this case 2 is more appropriate.

6 Conclusions

In this paper we have presented our current thoughts on a structured procedure for developing Controlled Language specifications and associated checkers, with the aims of allowing modular customization of computational checking. The first experiences of our current collaboration with Perkins Engines suggest that the procedure, and the capabilities of our checker shell, are reasonably sound as far as we have tested them. However, we have not yet produced a full analysis of the documents supplied to us by Perkins, nor more than a first general pass at grammar and error rules; preliminary introduction of a prototype system at Peterborough is intended to take place before the summer. We also have a preliminary agreement to consult with the Xerox Language Technology Centre on their downstream requirements for PACE. More detailed and empirical results will become available once these tasks have been completed.

References

[1] BSI. *Glossary of Building and Civil Engineering Terms.* Blackwell Scientific, 1993.

[2] Shona Douglas and Robert Dale. Towards Robust PATR. In *Proceedings of the 14th International Conference on Computational Linguistics,* Nantes, France, August 23-28 1992.

[3] Heather Holmback and Serena Shubert. Issues in conducting empirical evaluations of controlled language. In *Proceedings of the First International Workshop on Controlled Language Applications*, March 1996. Leuven, Belgium.

[4] Matthew Hurst. Parsing For Targeted Errors in Controlled Languages In *Proceeding of Recent Advances In Natural Language Processing*, Bulgaria, March 1995.

[5] Peter Pym. Perkins Engines and Publications. In *Proceedings of TECHNOLOGY & LANGUAGE IN EUROPE 2000: The IK Perspective,* 1993.

[6] G Schreiber, B Wielinga, and J Breuker, editors. *KADS: A Principled Approach to KBS Development*. Academic Press Limited, London, 1993.

[7] Dirk Schreurs and Geert Adriaens. Controlled English (CE): from COGRAM to ALCOGRAM. In Patrik O'Brian Holt and Noel Williams, editors, *Computers and Writing: State of the Art*, chapter 14, pages 206-221. Intellect, Oxford, 1992.

[8] Henry Thompson, Steve Finch and David McKelvie The Normalised SGML Library (NSL) HCRC Technical Report HCRC/TR-74, Human Communication Research Centre, University of Edinburgh, Edinburgh, Scotland, UK, 1995.

[9] Patricia Wright. Quality or Usability? Quality Writing Provokes Quality Reading. In Michael Steehouder, Carel Jansen Peiter van der Poort, and Ron Verheijn, editors, *Quality of technical documentation: Utrecht Studies in Language and Communication*. University of Utrecht, 1992.

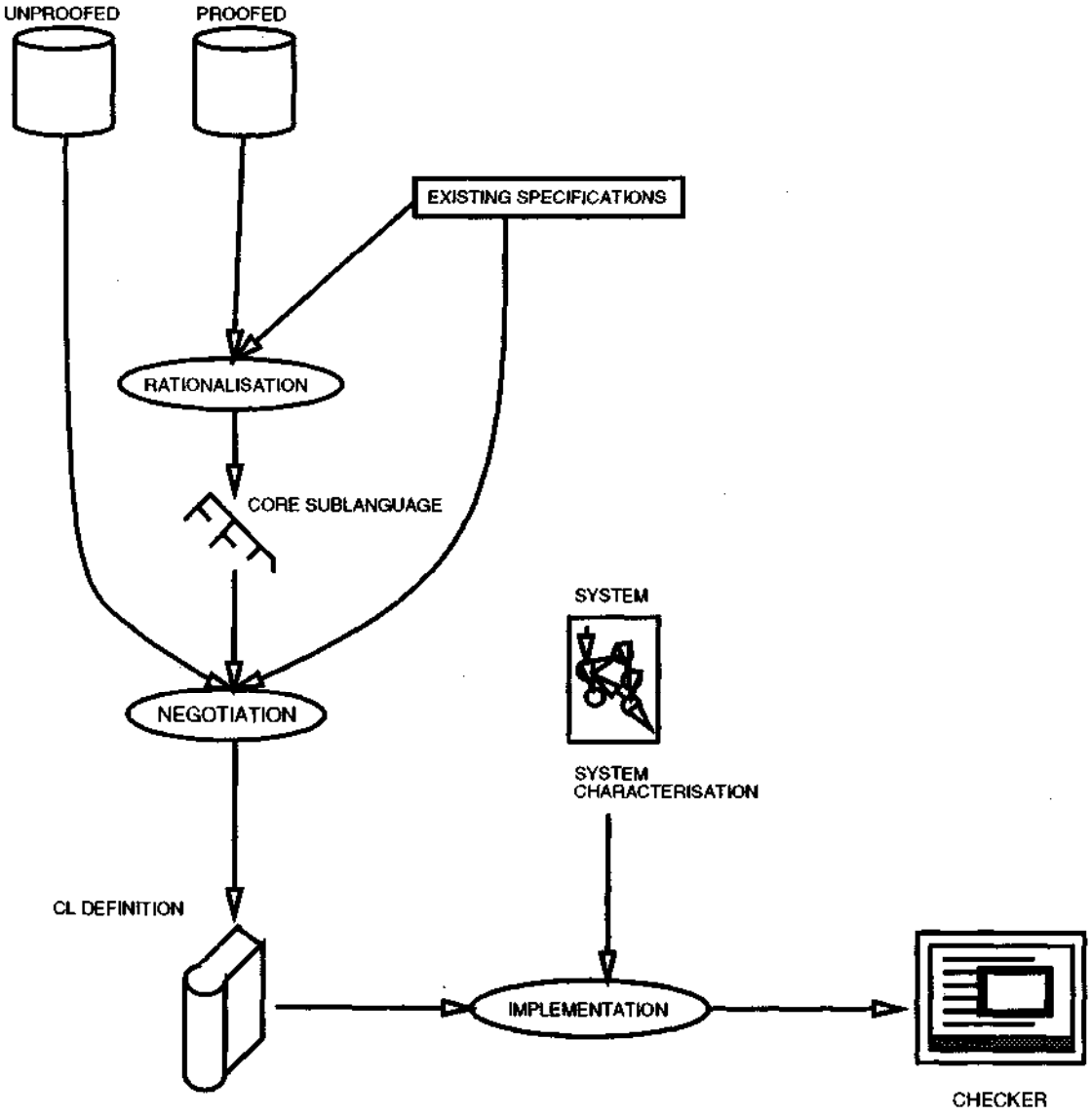

Figure 1: Expertise modelling process resulting in CL definition and instantiated checker

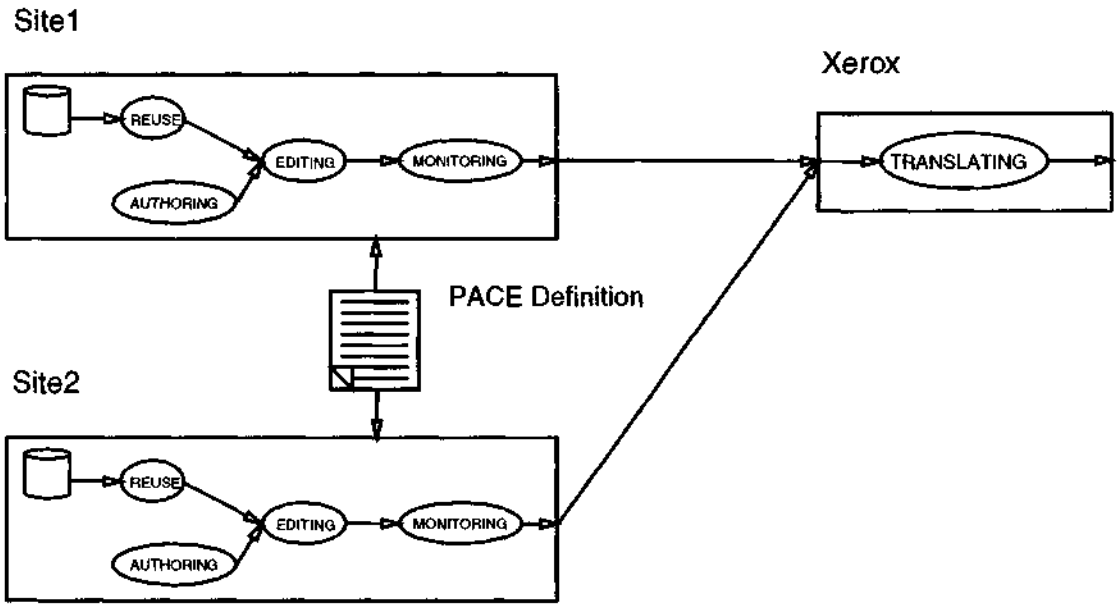

Figure 2: Document flows and organizational structure of the Perkins setup

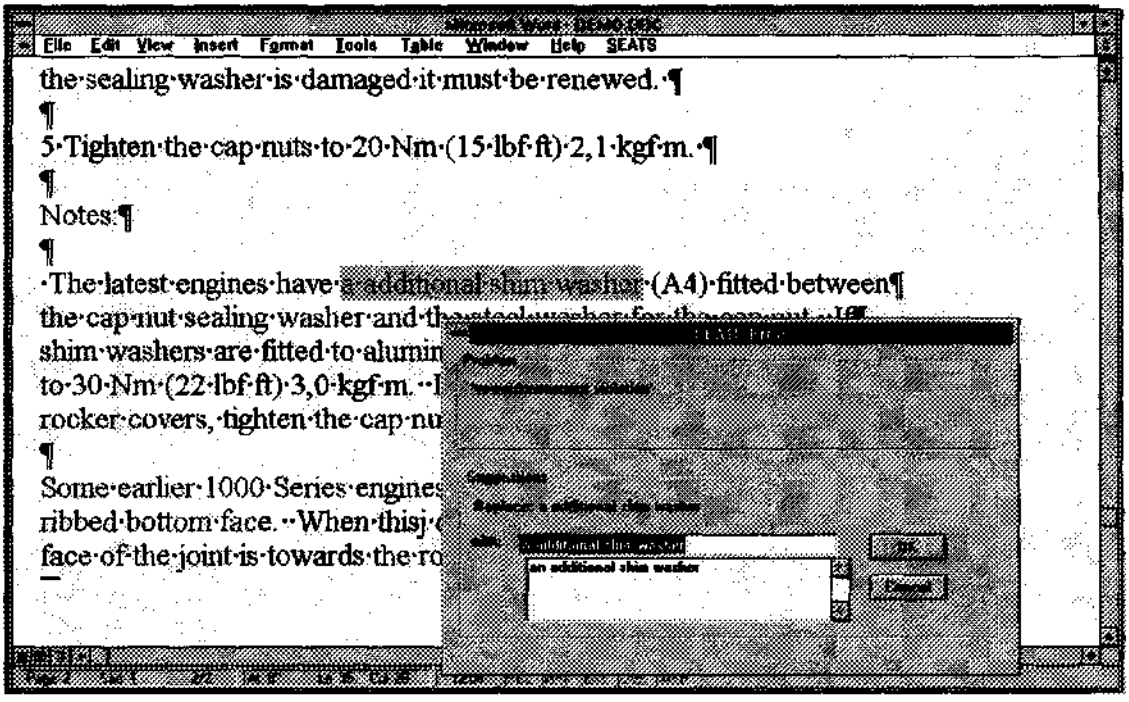

Figure 3: Screen dump of MS Word interface to checker

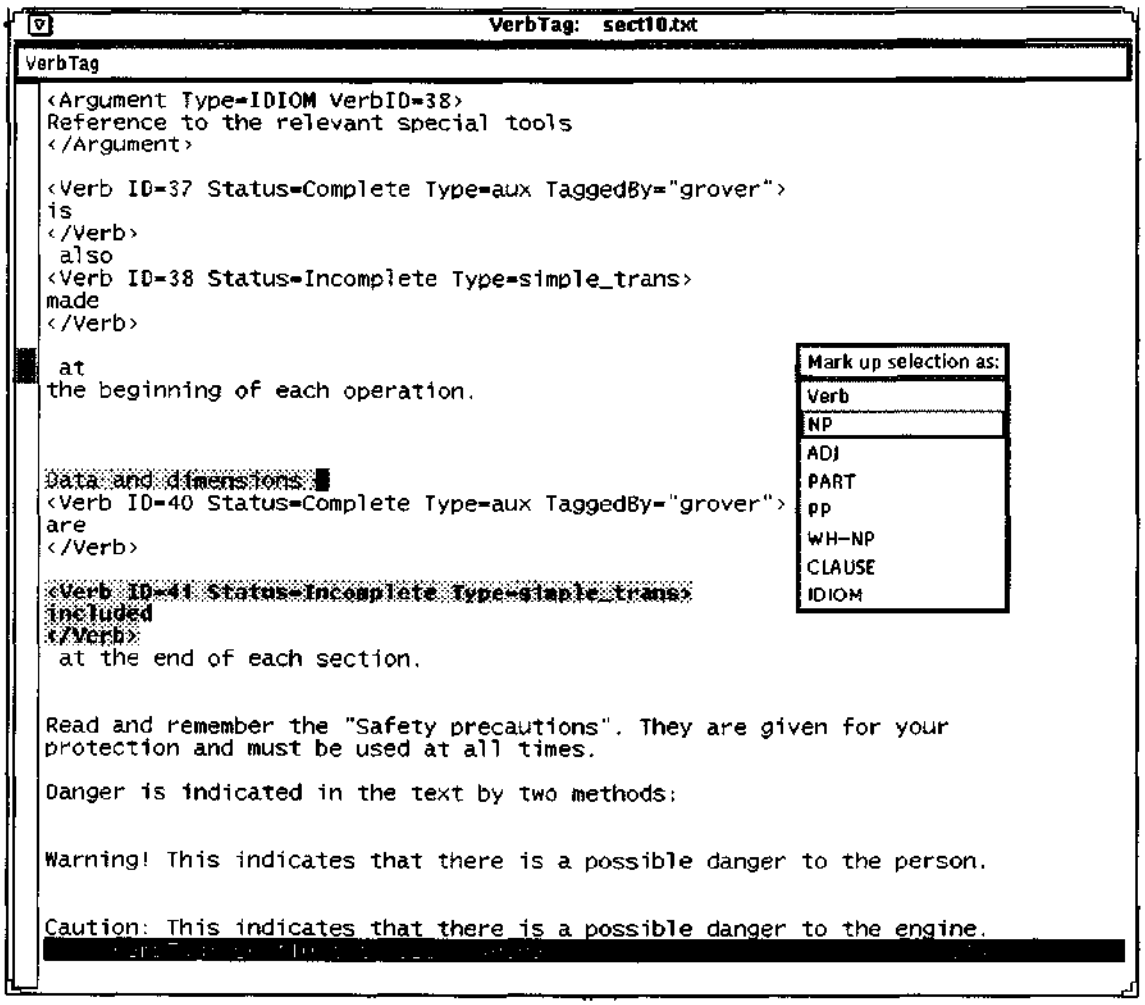

Figure 4: SGML markup of verb subcategorization frames