What You See Is What You Meant: direct knowledge editing with natural language feedback

Richard Power and **Donia Scott** and **Roger Evans**[1]

Abstract. Many kinds of knowledge-based system would be easier to develop and maintain if domain experts (as opposed to knowledge engineers) were in a position to define and edit the knowledge. From the viewpoint of domain experts, the best medium for defining the knowledge would be a text in natural language; however, natural language input cannot be decoded reliably unless written in controlled languages, which are difficult for domain experts to learn and use. WYSIWYM editing is an alternative solution in which the texts employed to view and edit the knowledge are generated not by the user but by the system. The user can add knowledge by clicking on 'anchors' in the text and choosing from a list of semantic alternatives; each choice directly updates the knowledge base, from which a new text is then generated.

1 KNOWLEDGE EDITING

Many applications require editing of information expressed in a knowledge representation formalism. Expert systems are an obvious example; others are systems for generating documents [4, 8] and for encoding design specifications [5]. With currently available support tools, knowledge editing has to be performed by knowledge engineers who are familiar with the representation formalism; the knowledge cannot be directly inspected and modified by domain experts or other interested parties. Much recent research has sought to simplify knowledge editing, e.g. by graphical browsers, or by input in controlled languages.

We propose here a new knowledge-editing method called 'WYSIWYM editing'. WYSIWYM editing allows a domain expert to edit a knowledge base reliably by interacting with a *feedback text*, generated by the system, which presents both the knowledge already defined and the options for extending it. Knowledge is added by menu-based choices which directly extend the knowledge base; the result is immediately displayed to the author by means of an automatically generated natural language document: thus 'What You See Is What You Meant'.

1.1 Graphical tools

Most knowledge bases are coded in object-based formalisms, with objects classified by a conceptual hierarchy so that attributes can be inherited, as in such languages as LOOM [6]. Encoding the knowledge is often a programming task, but some knowledge-editing tools provide graphical browsers in which relations among objects are shown by network diagrams (examples are the Generic Knowledge-Base Editor [7] and the CODE4 Knowledge Management System [13]).

[1] Information Technology Research Institute, University of Brighton, Lewes Road, Brighton BN2 4GJ, UK, Firstname.Lastname@itri.bton.ac.uk

ECAI 98. *13th European Conference on Artificial Intelligence*
Edited by Henri Prade
Published in 1998 by John Wiley & Sons, Ltd.

Such graphical tools have the advantage of allowing what we will call 'direct knowledge editing', as opposed to 'text editing'. When you edit a network diagram, the basic operations have a direct semantic interpretation: for instance, you create an object belonging to one of the permitted categories; or you extend a labelled arc from one object to another, thus encoding one of the permitted relationships between the two objects. By contrast, when you encode the knowledge by writing a program in a text editor, the basic operations of inserting and deleting characters have no direct semantic interpretation: to extract the knowledge, the text must be parsed and interpreted, and even experienced programmers are likely to make syntax errors.

Although graphical editing tools make life easier for the knowledge engineer, they fail in the objective of making knowledge editing accessible to domain experts. Network diagrams correspond too directly to the underlying knowledge formalism: they cannot be understood and developed without training in such technical notions as 'object', 'attribute' and 'value restriction'. Empirical studies have reported high error rates by domain experts using graphical object-oriented modelling tools [3], and a clear advantage of text over graphics for understanding nested conditional structures [9].

1.2 Natural language input

Some researchers have tried an alternative solution in which the author defines the knowledge base by writing a natural language text, which the system automatically interprets and encodes in the knowledge representation formalism. This method is obviously attractive, since it would allow domain experts to specify a knowledge base in the most natural way; the problem is whether it is feasible. Unfortunately, information extraction from free text is unreliable: for the foreseeable future, natural language input can be used only if the author adheres to a controlled language such as Attempto Controlled English [2] or Computer Processable English [11]. Many benefits of natural language input are therefore lost. The domain expert can define a knowledge base only after training in the controlled language; and even after training, the author may have to try several formulations before finding one that the system will accept.

Natural language input implies text editing rather than direct knowledge editing: the author defines a character sequence that must be parsed and interpreted to extract the knowledge. An interesting intermediate technique, which might be called 'syntactic editing', is used in the NLMenu system [14]: sentences are composed not by typing in characters, but by choosing at each stage from a list of the continuations allowed by the grammar. In this way, the system guarantees that every input sentence can be parsed and interpreted; the disadvantage is that the author is constrained to focus on syntactic development of the current sentence rather than on semantic development of the knowledge.

1.3 A new solution

We have seen that direct knowledge editing is preferable to text editing, since it avoids any need for automatic parsing and interpretation; on the other hand, presentation in natural language is preferable to presentation in a network diagram (or in a programming language), especially if we want the editing tool to be accessible to domain experts as well as to knowledge engineers. WYSIWYM editing attempts to have your cake and eat it, by a technique that combines direct knowledge editing with presentation in natural language text. The method of presenting an object-oriented model through a text has been used before (see [12] for a review); what is novel about WYSIWYM is that the author can actually build and edit the model by interacting with the feedback text.

Before explaining the technique in detail, we will make some preliminary comments in order to identify the niche that WYSIWYM editing occupies in the space of possible editors.

2 TYPES OF EDITOR

At first sight WYSIWYM editing looks like word processing, because the edited material is presented as a formatted document. To understand the difference, we need to look more closely at the nature of word processing.

When you edit a document with a word processor, you aim to record various kinds of information. At a basic level, you define a sequence of characters. These characters make up words and sentences, which in turn express concepts and ideas. The word processor may also allow you to influence the graphical appearance of the document, e.g. by changing the font or the size of the characters; if it supports WYSIWYG, you will be able to see on the screen the appearance of the document when it is printed on paper.

Following the usual distinctions in linguistics, we can identify four levels of information, each level having its own characteristic features:

1. Graphical level (the two-dimensional bitmap)
2. Graphemic level (the character sequence)
3. Syntactic level (the words and sentences)
4. Semantic level (the meaning)

The visual display in a word processor shows all these levels. However, the editing operations mainly concern the graphemic level: the insertion or deletion of characters. This is the only level at which the user exerts direct control. The graphical level can be influenced by altering fonts, page sizes, etc., but the user cannot directly draw the characters. The syntactic and semantic levels can be influenced only through the mediation of the graphemic level.

To summarize this situation, we can distinguish three kinds of features.

1. **Directly controlled features** These are linked to the basic editing operations. For instance, in a word processor the character sequence is controlled directly by positioning the cursor and hitting keys.
2. **Interpreted features** These are not directly controlled, but may be recovered by interpreting lower-level features. In a word processor, syntactic and semantic features can be derived by interpreting the character sequence. This raises the issue of who (or what) recovers the interpreted features. A program cannot utilize syntactic or semantic features encoded in a character sequence unless it has the ability to parse and understand the language. Thus the semantic features encoded in a document might be useful only to human observers.
3. **Presentational features** These are introduced by the program in order to display the other features to the user. For instance, a text editor that only saves the character sequence must add graphical features in order to print the document or to display it on the screen.

We can now express the similarities and differences between text editing and WYSIWYM editing. What they have in common is that the editor displays graphical, graphemic, syntactic and semantic features; this is why they look alike. The difference is that a word processor allows direct control over graphemic features (the character sequence), while WYSIWYM editing allows direct control over semantic features (the knowledge). In WYSIWYM editing, the graphical, graphemic and syntactic features are all presentational. The user may influence them (e.g. by choosing French in preference to English), but cannot control them directly (e.g. by typing in specific characters).

Under this scheme, we can imagine four kinds of natural language editor, according to the level that is directly controlled by the user.

1. **Handwriting Editor** The user directly controls the graphical level by writing the document by hand, using a pen-based input device. In order to utilize information at other levels (e.g. to print a typed version of the document) the editor would have to perform character recognition.
2. **Text Editor** The user directly controls the graphemic level by typing in characters at the cursor. This brings a cost and a benefit: the user loses control over the detailed shaping of the characters, but reliably encodes a character sequence that the editor can utilize, e.g. by sending it through email, or varying the font sizes or column widths.
3. **NL-Menu** The user controls the syntactic level, by choosing from a list of words or phrases that are permissible extensions of the current sentence. In this way, the user loses direct control over the character sequence, but reliably encodes a linguistic structure from the sub-language that the editor can parse, so that the program can reliably derive the meaning.
4. WYSIWYM **editing** The user controls the semantic (or knowledge) level, by choosing conceptual extensions of the current meaning from pop-up menus. All other features become presentational, so that the user has no direct control over wording: switching from English to French becomes a matter of presentational convenience, akin to switching from small to large font. The program performs no parsing or interpretation of any kind, since the highest level (the knowledge) is directly controlled by the user and hence there are no interpreted features.

3 EDITING MEANING

At all levels, editing consists of modifying a current configuration either by inserting material at a given location, or by deleting material previously inserted. The initial, minimal configuration must have at least one location; as material is added, new locations become available. In a text editor, the initial configuration is an empty string with a single location (and hence a single cursor position); if you type the letter 'A' there are now two locations, before and after the letter; add 'B' and there are three locations, and so forth.

In a semantic network, the configuration comprises objects of various types (the labelled nodes in a network diagram); depending on its type, an object may have various attributes (the labelled arcs in a diagram); the value of an attribute is another object (the node to which the arc points). The basic editing operation is that of adding a new object, of a specified type, as the value of an attribute of an existing object. A location can be thought of as an attribute that currently has no value. As an initial minimal configuration, we need at least one fixed attribute that cannot be deleted. The new object added as the value of this root attribute will have attributes of its own, so that further locations become available.

The basic idea of WYSIWYM editing is that a special kind of natural language text is generated in order to present the current configuration of a semantic network. This 'feedback text' includes generic phrases called 'anchors' which mark attributes that have no value. The anchors represent the locations where new objects may be added. By opening a pop-up menu on an anchor, you obtain a list of short phrases describing the types of objects that are permissible values of the attribute; if you select one of the options, a new object of the specified type is added to the semantic network. A new feedback text is then generated to present the modified configuration, including the attributes of the new object.

As more information is added about a new object, it will be represented by longer spans of text, comprising whole sentences, or perhaps even several paragraphs. These spans of text are also mouse-sensitive, so that the associated semantic material can be cut or copied. The cutting operation removes the network fragment that was previously the value of an attribute, and stores it in a buffer, where it remains available for pasting into another suitable location. When the text is regenerated, an anchor will show that the attribute now has no value, and the span of text that previously represented this value will no longer appear.

4 ILLUSTRATION OF WYSIWYM EDITING

Our first application of WYSIWYM editing was in the context of the DRAFTER project [8], which developed a system to support the production of software documentation in English and French. The system includes a knowledge editor, with which a technical author can define the procedures for using common software applications such as word processors and diary managers; in this way the author builds the 'domain model' from which a text generator produces instructions, in English and French, that describe these procedures. The eventual aim of such systems is to support the technical authors who produce tutorial guides and user manuals for software applications, by automatically generating routine procedural passages in many languages.

In DRAFTER-1, the first version of the system, knowledge editing was performed through a graphical interface in which objects in the knowledge base were presented through nested boxes with brief linguistic labels. Since authors found these diagrams hard to interpret and modify, we decided to explore the new idea of presenting the growing domain model through a natural language text, thus exploiting the multilingual generator to support knowledge editing. The result was a completely re-engineered system, DRAFTER-2, in which the generator not only produces the final output texts, but also supports a WYSIWYM knowledge editor (see [12, 10] for more details of the architecture).

As an example of WYSIWYM editing, we will describe a session in which a technical author uses DRAFTER-2 in order to define the knowledge underlying a brief passage of software documentation. We will suppose that the author is producing a tutorial guide to the OpenWindows Calendar Manager, and is currently working on a section that explains how to schedule an appointment.

The procedure for scheduling an appointment requires various data to be entered in a dialogue box called the 'Appointment Editor' window. With some simplifications, it could be expressed by the following text, which we quote here in order to clarify the author's task.

(1) To schedule an appointment
Before starting, open the Appointment Editor window by choosing the Appointment option from the Edit menu.
Then proceed as follows:
1 Choose the start time of the appointment.
2 Enter the description of the appointment in the What field.
3 Click on the Insert button.

The author's aim is to introduce this information into the domain model. If this is done successfully, the system will be able to generate the above text (or an equivalent one) in English, French, and any other supported language.

Assuming that the model is to be built from scratch, the author begins by selecting the option 'New' from the main menu. Since the system is specialized for defining procedural models, a 'new' model comprises a single procedure object for which the attributes — i.e. the goal and the methods for achieving it — are undefined. From this model, the generator produces a single-sentence feedback text:

(2) Do **this action** by using *these methods*.

The feedback text has special features related to its authoring function.

- The phrase **this action** is coloured red, indicating that it marks a mandatory choice, a location where information *must* be added. In this case, the red phrase represents the undefined goal of the procedure. (Since colour is unavailable, we reproduce red phrases in bold face in this paper.)
- The phrase *these methods* is coloured green, indicating that it marks an optional choice, a location where information *may* be added. In this case, the green phrase represents the undefined methods for achieving the goal. (We reproduce green phrases in italics.)
- Both coloured phrases are mouse-sensitive. By clicking on either phrase, the author opens a pop-up menu from which a concept may be selected.

We refer to the mouse-sensitive coloured phrases as 'anchors', by analogy with the links in a hypertext.

Anchors can be developed in any order: we will assume in this illustration that the author decides to define the goal of the procedure first, followed by the methods for achieving it.

To begin the process of defining the goal (scheduling an appointment), the author clicks on the red anchor. In response, the system displays a pop-up menu listing the available action concepts

choose
click
close
.....
save
schedule
start
type

from which the author should select 'schedule'. DRAFTER-2 now updates its model by filling the goal attribute with a scheduling action, which includes a new attribute for the event to be scheduled (as yet undefined). From the updated model, a completely new feedback text is generated, incorporating the information just defined (the scheduling action) along with a new red anchor indicating that the author must choose which kind of event is to be scheduled.

(3) Schedule **this event** by using *these methods*.

Although the anchors can be developed in any order, it would be most logical for the author to continue defining the goal by clicking on **this event** and choosing the appropriate concept, in this case ‘appointment’.

(4) Schedule the appointment by using *these methods*.

The goal is now completely specificed, since it is shown by a phrase with no red anchors. If an error has been made (e.g. choosing ‘meeting’ instead of ‘appointment’), the author can undo the mistaken choice by opening a pop-up menu on the relevant span of text (in this case, ‘the meeting’) and selecting the option ‘Cut’. This will bring back text 3 so that the correct choice can be made from the anchor **this event**. If the goal were completely wrong, the author could cut the span ‘Schedule the meeting’, undoing both choices and returning to text 2.

When satisfied that the goal is correctly defined, the author proceeds to specify the method (or methods) by which appointments can be scheduled. Opening the anchor *these methods* yields a single option, labelled ‘methods’: in effect, the only choice here is whether to develop this optional anchor at all. Since a method has several attributes, all of which must be expressed through anchors, the feedback text grows suddenly more complicated.

(5) To schedule the appointment

- Before starting, follow *this procedure*.
  Then proceed as follows.
  1 Do **this action** by using *these methods*.
  2 *Next step*.
  To quit, follow *this procedure*.
- *Next method*.

Since the material will no longer fit into a single sentence, the generator chooses a different pattern in which the methods are presented in bulleted paragraphs, introduced by a TO-phrase that presents the goal. The rephrasing of the goal is instructive: it shows that the author has been choosing the meaning, not the words. It is the generator, not the author, that decides how the procedure should be worded; as a result, the wording may change (without any intervention from the author) as the editing of the knowledge proceeds.

The model provides for more than one method, since sometimes a goal can be achieved in several ways. Since the author has decided that there will be at least one method, the components of the first method (so far undefined) are shown by suitable anchors; the optional anchor *Next method* (at the bottom) can be developed if the author wishes to define further methods.

A method comprises a precondition (optional), a sequence of steps (obligatory), and an interrupt procedure (optional). Each step is a procedure, because in addition to a goal it may have methods of its own. The precondition is a task that should be performed before the steps; the interrupt procedure provides a way of abandoning the method if you have second thoughts.

Since there must be at least one step, the first step is shown by a sentence with anchors for goal and methods. Further steps can be added by opening the optional anchor *Next step*. The same pattern is thus used for a sequence of methods and a sequence of steps, except that the former is presented by a bulleted list and the latter by an enumerated list.

Since the basic mechanism should now be clear, we now jump to a later stage in which most of the information has been defined; the only missing piece is the method for opening the Appointment Editor window.

(6) To schedule the appointment

- Before starting, open the Appointment Editor window by using *these methods*.
  Then proceed as follows.
  1 Choose the start time of the appointment from *this object* by using *these methods*.
  2 Enter the description of the appointment in the What field by using *these methods*.
  3 Click on the Insert button by using *these methods*.
  4 *Next step*.
  To quit, follow *this procedure*.
- *Next method*.

To add the last piece of information, the method for opening the Appointment Editor window, the author develops the anchor *these methods* (third line). This poses a problem for the generator, since as we have seen the material for an expanded method will not fit into a single sentence. The problem is solved by deferring the procedure for opening the Appointment Editor window to a separate paragraph.

(7) To schedule the appointment

- Before starting, open the Appointment Editor window.
  Then proceed as follows.
  1 Choose the start time of the appointment from *this object* by using *these methods*.
  2 Enter the description of the appointment in the What field by using *these methods*.
  3 Click on the Insert button by using *these methods*.
  4 *Next step*.
  To quit, follow *this procedure*.
- *Next method*.

To open the Appointment Editor window

- Before starting, follow *this procedure*.
  Then proceed as follows.
  1 Do **this action** by using *these methods*.
  2 *Next step*.
  To quit, follow *this procedure*.
- *Next method*.

As a result of this reorganization of the text, the action of opening the window has to be expressed twice: in the first paragraph, it serves as the precondition in the procedure for scheduling an appointment; in the last paragraph, it serves as the goal of a sub-procedure. Of course this does not mean that there are now two actions. The author might decide to cut one of the phrases ‘the Apointment Editor window’ in order to replace ‘window’ by another concept, e.g. ‘dialogue-box’, thus redefining the action as one of opening the Appointment Editor dialogue-box; the effect on the text would be that both the sentences expressing this action would change. This reinforces the point that the author is editing meaning, not text.

To complete the model, the author should develop the red anchor **this action**, which is eventually replaced by the phrase ‘Choose the Appointment option from the Edit menu’. At this point the model is potentially complete, since it contains no red anchors. (Note that the model was also potentially complete for texts 4 and 6.) When a model is potentially complete, the author can switch the modality from 'Feedback' to 'Output' in order to obtain a text which simply presents the knowledge base, without indicating the locations at which further information may be added. The generator will now produce text 1, which was used at the start of this section to indicate the desired content.

Note that this output text has been completely regenerated; it was not obtained merely by omitting the green anchors from text 7. In particular, since the method for opening the Appointment Editor window can now be expressed by a phrase, there is no need to defer it to a separate paragraph.

5 SIGNIFICANCE

WYSIWYM editing is a new idea that requires practical testing. We have not yet carried out formal usability trials, nor investigated the design of feedback texts (e.g. how best to word the anchors), nor confirmed that adequate response times could be obtained for full-scale applications. However, if satisfactory large-scale implementations prove feasible, the method brings many potential benefits.

- A document in natural language (possibly accompanied by diagrams) is the most flexible existing medium for presenting information. We cannot be sure that all meanings can be expressed clearly in network diagrams or other specialized presentations; we can be sure they can be expressed in a document.
- Domain experts understand natural language much better than they understand network diagrams.
- Authors require no training in a controlled language or any other presentational convention. Apart from the expense of initial training, this means that there is no problem of having to relearn the conventions when a knowledge base is re-examined after a delay of months or years.
- Since the knowledge base is presented through a document in natural language, it becomes immediately accessible to anyone peripherally concerned with the project (e.g. management, public relations, domain experts from related projects). Documentation of the knowledge base, often a tedious and time-consuming task, becomes automatic.
- The model can be viewed and edited in any natural language that is supported by the generator; further languages can be added as needed. When supported by a multilingual natural language generation system, as in DRAFTER-2, WYSIWYM editing obviates the need for traditional language localisation of the human-computer interface. New linguistic styles can also be added (e.g. a terminology suitable for novices rather than experts).
- As a result, WYSIWYM editing is ideal for facilitating knowledge sharing and transfer within a multilingual project. Speakers of several different languages could collectively edit the same knowledge base, each user viewing and modifying the knowledge in his/her own language.
- Since the knowledge base is presented as a document, large knowledge bases can be navigated by the methods familiar from books and from complex electronic documents (e.g. contents page, index, hypertext links), obviating any need for special training in navigation.
- For systems in which information must be retrieved from the knowledge base by complex queries, WYSIWYM editing can be used in order to formulate queries as well as to edit the knowledge base.
- For systems which generate technical documentation, WYSIWYM editing ensures that the output texts will conform to desired standards of terminology and style. For instance, the generation rules could be tailored to meet the constraints of controlled languages such as AECMA [1].

ACKNOWLEDGEMENTS

We thank all members of the DRAFTER team who contributed to our ideas on knowledge editing, especially Cécile Paris and Keith Vander Linden. We also thank Kees van Deemter for useful comments on an earlier draft.

REFERENCES

[1] AECMA. AECMA Simplified English: A guide for the preparation of aircraft maintenance documentation in the International Aerospace Maintenance Language. AECMA, Brussels, 1995.

[2] Norbert Fuchs and Rolf Schwitter, 'Attempto controlled english (ace)', in *Proceedings of the first international workshop on controlled language applications*, Katholieke Universiteit Leuven, Belgium, (1996).

[3] Y. Kim. Effects of conceptual data modelling formalisms on user validation and analyst modelling of information requirements. PhD thesis, University of Minnesota, 1990.

[4] Richard I. Kittredge and Alain Polguère, 'Generating extended bilingual texts from application knowledge bases', in *International Workshop on Fundamental Research for the Future Generation of Natural Language Processing, Kyoto, Japan*, pp. 147–160, (1991).

[5] B. Macais and S. Pulman, 'A method for controlling the production of specifications in natural language', *The Computer Journal*, **38**(4), (1995).

[6] Robert MacGregor and Raymond Bates, 'The LOOM knowledge representation language', in *Proceedings of the Knowledge-Based Systems Workshop*, St. Louis, April 21–23, (1987).

[7] S. Paley, 'Generic knowledge-base editor user manual', Technical report, SRI International, California, (1996).

[8] Cécile Paris, Keith Vander Linden, Markus Fischer, Anthony Hartley, Lyn Pemberton, Richard Power, and Donia Scott, 'A support tool for writing multilingual instructions', in *IJCAI-95*, pp. 1398–1404, (1995).

[9] M. Petre, 'Why looking isn't always seeing: readership skills and graphical programming', *Communications of the ACM*, **38**(6), 33–42, (1995).

[10] R. Power and D. Scott, 'Multilingual authoring using feedback texts', in *Proceedings of the 17th International Conference on Computational Linguistics and 36th Annual Meeting of the Association for Computational Linguistics*, Montreal, Canada, (1998).

[11] Stephen Pulman, 'Controlled language for knowledge representation', in *Proceedings of the first international workshop on controlled language applications*, Katholieke Universiteit Leuven, Belgium, (1996).

[12] D. Scott, R. Power, and R. Evans, 'Generation as a solution to its own problem', in *Proceedings of the 9th International Workshop on Natural Language Generation*, Niagara-on-the-Lake, Canada, (1998).

[13] D. Skuce and T. Lethbridge, 'CODE4: A unified system for managing conceptual knowledge', *International Journal of Human-Computer Studies*, **42**, 413–451, (1995).

[14] Harry Tennant, 'The commercial application of natural language interfaces', in *COLING-86*, (1986).