WYSIWYM with wider coverage

Richard Power and Roger Evans
Information Technology Research Institute
University of Brighton
Lewes Road
Brighton BN2 4AT, UK
Firstname.Lastname@itri.bton.ac.uk

Abstract
We describe an extension of the WYSIWYM technology for knowledge editing through natural language feedback. Previous applications have addressed relatively simple tasks requiring a very limited range of nominal and clause patterns. We show that by adding a further editing operation called reconfiguration, the technology can achieve a far wider coverage more in line with other general-purpose generators. The extension will be included in a Java-based library package for producing WYSIWYM applications.

1 Introduction
WYSIWYM (What You See Is What You Meant) is a user-interface technology through which a domain expert can formally encode knowledge by structured editing of an automatically generated feedback text (Power and Scott, 1998). The technology has hitherto addressed two practical contexts: the automatic production of multilingual technical documentation, and the formulation of queries to a database or expert system. In the first case, WYSIWYM editing encodes the desired content of the document in an interlingua, from which versions can be generated in multiple languages; in the second case, it yields a query encoded in a formal query language such as SQL. The benefit is the same in either context: since editing is mediated through a presentation in natural language, there is no need for the user to be acquainted with the formal details of knowledge representation or query languages.

Elsewhere (Evans and Power, 2003) we have described a library package for developing WYSIWYM applications. This package was a consolidation of work carried out in a series of early applications (Power and Scott, 1998; Piewek et al., 2000; Bouayad-Agha et al., 2002), requiring a very restricted linguistic coverage, especially as regards the range of clausal and nominal patterns. We present here an extension to this library which allows a coverage more in line with general-purpose generators like FUF/SURGE (Elhadad and Robin, 1992), KPML/PENMAN (Bateman, 1996) and REAL-Pro (Lavoie and Rambow, 1997). The extension is based on two new ideas: first, a change to the underlying semantic model, replacing atomic entity types with feature structures; secondly, a corresponding change in the user interface, which now offers an extra editing operation (called reconfiguration) through which complex entity types may be modified. The purpose of this paper (and the accompanying demonstration) is to describe these novelties.

2 Editing with simple types

Figure 1: A-box with simple types

In early WYSIWYM applications, the editing process served to build an A-box like that shown in figure 1, comprising a set of entities (represented by rectangles), each entity having a simple type (represented by labels within rectangles) and a set of relationships (represented by labelled arcs). The graph in this figure is rooted in a take entity, denoting a taking event, the participants being a patient entity (the taker) and an aspirin entity (the takee). The intended meaning of the graph is expressed by the English sentence ‘the patient takes an aspirin’.

The construction of the graph through WYSIWYM editing proceeds as follows. The starting point is an empty A-box, which consists only in a constraint on the root entity — for in-
stance, the requirement that it should be some kind of event. This unpromising A-box is supplied as input to a natural language generator with two special features: (a) it can generate texts from an A-box in any state of completion (even empty); (b) it can generate menus opening on anchors within the text, in addition to the text itself. The resulting feedback text is presented to the user through a special interface in which some spans are mouse-sensitive anchors, marking points where a new entity may be added to the A-box. Anchors are normally shown through a colour code; here we will employ square brackets:

[Some event].

When the user mouse-clicks on an anchor, a menu pops up listing all entity types allowed in the relevant context — in this case, all event types.

| arrive | breathe | ... |
|--------|---------|-----|
| ...    | take    | ... |

After the user chooses one of these options, such as ‘take’, a new entity of the specified type is created, and added to the A-box at the current location (in this case, the root of the graph). Assuming the ontology decrees that a take event has two participants, a person and an object, the new A-box will include two anchors allowing these entities to be defined:

[Some person] takes [some object].

Opening the anchor ‘some person’ will yield a list of options including ‘patient’; opening ‘some object’ will yield options including ‘an aspirin’; in this way two more entities can be introduced, so obtaining the complete graph in figure 1.

### 3 Limitations in coverage

For some applications, the above procedure works well, but it allows far too few variations to cope with real documents or queries of normal linguistic complexity. A single choice of event type (‘take’) is assumed by default to imply just one out of the thousands of possible clause patterns that could be obtained by varying mood, tense, polarity, modality, etc., or by adding adverbial modifiers:

| FORCE | does the patient take an aspirin? |
|-------|----------------------------------|

| TIME  | patient took an aspirin |
|-------|-------------------------|
| POLARITY | patient will take an aspirin |
| MODALITY | patient does not take an aspirin |
| MODIFIER | patient may take an aspirin |
| MODIFIER | patient must take an aspirin |
| MODIFIER | patient might take an aspirin |
| MODIFIER | patient should take an aspirin |

By combining just the above features, we obtain over 300 combinations; these would multiply further if we included the semantic features controlling perfective, progressive, voice, and wh-questions. Such a large set of options challenges the feasibility of WYSIWYM, or indeed any other approach to knowledge editing by domain experts.

### 4 Editing with complex types

Our favoured (indeed, only) proposal for embracing these variations is based on an analogy with a drawing tool. In WYSIWYM, choosing take from a menu of event types introduces an event entity, implicitly defaulted to present time, positive polarity, and so forth. In a drawing tool, choosing the rectangle icon from a palette of shapes introduces a rectangle entity, implicitly defaulted to a certain size, colour, and border (to name just three features). Having introduced a rectangle entity, however, the user can reconfigure it by changing these features one at a time. Why should an equivalent operation not be provided for the semantic features underlying a clause?

![Figure 2: A-box with complex types](image-url)
To add this extra editing operation we must replace the simple entity types employed in early WYSIWYM systems by complex types, as illustrated in figure 2 (to simplify, just a few of the possible features are shown). To reconfigure an entity, the user selects the corresponding span in the feedback text (all such spans will be mouse-sensitive), and chooses from a menu of options, each corresponding to a change in just one feature.

With this potentially huge increase in the number of editing operations for a given feedback text, the idea of precomputing all possible menus and popping one up on demand becomes less attractive, both computationally and to the user. Instead, when the user selects a span of text, the menu of reconfigurations for that span is computed on the fly, and displayed in a static menu pane adjacent to the main text pane, which can be browsed and searched - see figure 3. At every stage during the interaction, the user sees a feedback text (right pane), with one span highlighted through a colour code, and a list of options for reconfiguring the currently selected unit (left pane). If the selected unit happens to be an anchor (square brackets), the operation will be one of choosing an initial entity type rather than reconfiguring an existing one, but the appearance of the interface will be the same. The user can continue the interaction in two ways: either by choosing an option from the menu pane, or by selecting a different current unit by mouse-clicking within the feedback text pane.

To illustrate, we will suppose that the current A-box is as depicted in figure 2, and that the ‘patient’ entity is currently selected. Highlighting the selected span in bold face rather than a colour code, the feedback text and the menu of reconfiguration options might be as follows:

| IDENTIFIABILITY | A patient | MULTIPLICITY | The patients |
|----------------|-----------|--------------|-------------|

The benefits of this approach were to provide effective support for document authoring, the underlying model offers additional functionality in other knowledge creation scenarios as well. The examples in this paper use the complex types of the knowledge objects to represent linguistic variation, but might just
as easily represent other kinds of semantic detail, for example in an object-oriented program specification scenario.

6 Conclusion

In this paper we have described an extension to our earlier WYSIWYM approach which supports more sophisticated interactions with the underlying knowledge base, allowing a far wider range of linguistic expressions to be constructed. This makes the system more suitable for real authoring tasks, particularly in controlled language or multilingual contexts, while also enhancing its potential for constructing and editing other kinds of complex knowledge.

The system has been implemented as an extension to our WYSIWYM library (Evans and Power, 2003), using a wide-coverage grammar based on the subcategorisation frames found in the XTAG (Doran et al., 1994) categories, and deployed in the domain of medical informatics. The demonstration requires a PC with Java and Sicstus Prolog.

References

John A. Bateman. 1996. KPML: The KOMET-Penman (Multilingual) Development Environment. Technical report, Institut für Integrierte Publikations- und Informationssysteme (IPSI), GMD, Darmstadt, March. Release 0.9.

Nadjet Bouayad-Agha, Richard Power, Donia Scott, and Anja Belz. 2002. PILLS: Multilingual generation of medical information documents with overlapping content. In Proceedings of the Third International Conference on Language Resources and Evaluation (LREC 2002), pages 2111–2114, Las Palmas.

Christy Doran, Dania Egedi, Beth Ann Hockey, B. Srinivas, and Martin Zaidel. 1994. XTAG system - a wide coverage grammar for english. In Proceedings of the 15th International Conference on Computational Linguistics (COLING 94), pages 922–928, Kyoto, Japan.

Michael Elhadad and Jacques Robin. 1992. Controlling content realization with functional unification grammars. In Aspects of Automated Natural Language Generation, pages 89–104. Springer Verlag.

Roger Evans and Richard Power. 2003. Wysiwyg: Building user interfaces with natural language feedback. In Research notes and demonstration papers at EACL-03, pages 203–206, Budapest, Hungary.

B. Lavoie and O. Rambow. 1997. RealPro: A fast, portable sentence realizer. In Proceedings of the Conference on Applied Natural Language Processing (ANLP’97), Washington, DC.

Paul Piwek, Roger Evans, Lynne Cahill, and Neil Tipper. 2000. Natural language generation in the mile system. In Proceedings of the IMPACTS in NLG Workshop, pages 33–42, Schloss Dagstuhl, Germany.

R. Power and D. Scott. 1998. 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, pages 1053–1059, Montreal, Canada.