

# WYSIWYM – building user interfaces with natural language feedback

**Roger Evans and Richard Power**

Information Technology Research Institute

University of Brighton

Roger.Evans@itri.brighton.ac.uk

Richard.Power@itri.brighton.ac.uk

## 1 Introduction

WYSIWYM (‘What you see is what you meant’) is a user-interface technique which uses natural language generation (NLG) technology to provide feedback for user interactions. To date, the technology has been applied in a number of demonstrator applications, using customised, non-portable implementations. In this demonstration, we introduce a WYSIWYM *library package*, designed to be used as a modular component of a larger JAVA-based application. We show how the overall design of the package aims to support a range of possible applications using simple configuration options and JAVA subclassing, and illustrate the approach using examples ranging from the simplest proof-of-concept application to a complex web-delivered authoring tool for pharmaceutical leaflets.

## 2 WYSIWYM – a brief history

### 2.1 The WYSIWYM architecture

A long-standing research goal has been to enable subject matter experts (SMEs) to create or modify a knowledge base directly, without need for specialised training in knowledge representation languages; this is the main objective, for example, of the Rapid Knowledge Formation project (Clark et al., 2001). Several research groups have developed direct-manipulation interfaces in which the current state of the knowledge base, and the options for editing it, are displayed diagrammatically (Skuce and Lethbridge, 1995; Paley, 1996; Grosso et al., 2000). The basic idea of WYSIWYM is to pro-

vide a direct-manipulation interface in which the knowledge is displayed through a *feedback text*, generated automatically from the current knowledge base. Options for editing are shown through pop-up menus which open on spans of the feedback text. Unfilled slots in the knowledge base are shown through coloured text spans called *anchors*, the wording of the anchor indicating the constraint on the filler:

Swallow two tablets **in this way**

By clicking on the anchor **in this way**, the author obtains a list of options for filling the slot, also presented in natural language. When an option is selected, the system creates a filler of the specified type, and regenerates the feedback text:

Swallow two tablets with a glass of **some liquid**

The revised feedback text introduces a new anchor, through which the author can specify that the liquid in the glass is water. Thus by direct manipulation of the feedback text, our SME author has created a knowledge fragment equivalent to this feature structure:

```
swallow
  ACTOR patient
  ACTEE tablet
    NUMBER 2
  INSTRUMENT glass
    CONTENT water
```

However, the author need never see this structure nor understand the underlying knowledge formalism: all interaction with the knowledge base is mediated through the feedback text. Additional

advantages of this approach are (a) that the feedback text can be generated in multiple languages, thus allowing distributed editing by an international team and automatic localisation of the interface, and (b) that documentation of the knowledge base is automatic.

## 2.2 Previous WYSIWYM implementations

**Drafter** The earliest application of WYSIWYM developed from the DRAFTER project (Paris et al., 1995), which introduced the application of multilingual NLG for the production of instructions, using the domain of software manuals. This project introduced the technique of *symbolic authoring*, in which a SME creates a knowledge-base from which documentation is generated in multiple languages. The authoring interface was initially a conventional graphical one, but the subsequent DRAFTER-2 system (Power et al., 1998) showed that the knowledge could be more easily edited through direct manipulation of a text. In DRAFTER-3 (van Deemter and Power, 1998), the interface was extended to allow finer control over coreference, so that given the feedback text

Remove a tablet from the foil and swallow **something**

the author could copy ‘a tablet’, either with or without coreference, and paste it onto the anchor **something**, obtaining two possible outcomes:

Remove a tablet from the foil and swallow it  
[with coreference]

Remove a tablet from the foil and swallow another tablet [without coreference]

Of course, in either case, what was actually manipulated was a fragment of knowledge, not a string of characters, and the coreference distinction corresponded to whether the new reference was to the original fragment, or a copy of it.

**CLIME** In the CLIME project (Piwek et al., 2000), a WYSIWYM interface allowed the user to construct semantically complex queries for a legal inference system which answered questions about shipping regulations. This application differed significantly from the DRAFTER system in a number of ways: it was not a symbolic authoring system – the knowledge base created with WYSIWYM

was converted into a logical formula for submission to the inference engine; it was web-delivered – the user interface was written as a JAVA applet communicating with a server-side generation engine; and it added support for plurals and groups of objects to the WYSIWYM expressive repertoire (Piwek, 2000).

**PILLS** The PILLS project (Bouayad-Agha et al., 2002) demonstrated the application of WYSIWYM technology to produce documentation about pharmaceuticals in multiple languages. From a single ‘master model’ containing information about a particular product, the program could generate ingredient information in three document-types (patient information leaflets, summaries of product characteristics and European Pharmacopia entries) and three languages (English, French and German). The ontology and lexicons include concepts extracted automatically from a large medical database, the Unified Medical Language System (Schultz and Hahn, 2001). As with CLIME, the PILLS interface was web-delivered, with the output documents also being displayed via HTML frames.

## 3 The WYSIWYM library

In the systems just described, the WYSIWYM component was tightly integrated into the application as a whole. In order to facilitate wider and more flexible use of WYSIWYM technology, we have taken the CLIME/PILLS implementation and repackaged it as a JAVA-based library.

The general architecture of the WYSIWYM library is shown in figure 1. The library has two main components, a set of JAVA classes providing the user interface and API, and a PROLOG subsystem providing the knowledge-base management and NLG functionality. The application developer provides the domain information (ontology of the knowledge-base and linguistic information for supported languages), plus configuration and localisation information for any menu- and toolbars associated with the WYSIWYM panel.

There are two main aspects to configuration and deployment of a WYSIWYM application. First, the disposition of the PROLOG subsystem relative to JAVA is controlled by using different combinations

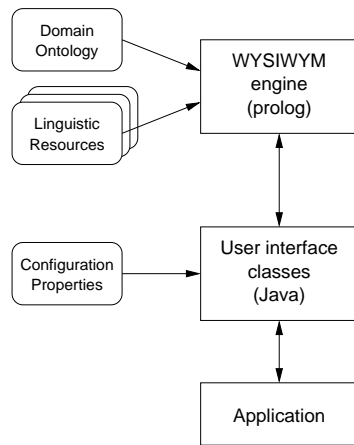


Figure 1: WYSIWYM library architecture

of the JAVA classes provided. The three principal dispositions are **internal** (PROLOG and JAVA running in a single executable), **external** (PROLOG and JAVA as separate executables on the same system) and **remote** (JAVA deployed as a web applet with PROLOG running on the server).

Second, the design of the menu- and tool-bars associated with a WYSIWYM panel, and callbacks to the main application functions, are controlled by simple property files, allowing a high degree of flexible control without any additional coding or recompilation.

In addition to these controls, the JAVA class structure is designed to allow more sophisticated extensions to the library, for example by subclassing the options menu class to provide a more advanced concept browser.

The basic application interface to the library is straightforward: an application creates an instance of a WYSIWYM interface to allow the user to create a knowledge base. When the user has completed the task, the knowledge base is returned to the application as an XML string, such as<sup>1</sup>:

```
<kb>
  <object id=o1 name=swallow>
    <role name=ACTOR target=o2/>
    <role name=ACTEE target=o3/>
    <role name=INSTRUMENT target=o4/>
  </object>
  <object id=o2 name=patient/>
  <object id=o3 name=tablet>
```

<sup>1</sup>Notice that this encoding does not rely on the XML structure to represent the underlying network structure, so that reentrancy and mutual dependence can easily be modelled.

```
<role name=NUMBER value=2/>
</object>
<object id=o4 name=glass>
  <role name=CONTENT target=o5>
</object>
<object id=o5 name=water/>
</kb>
```

## 4 Example applications

The following example applications are included in the library package.

### 4.1 A simple example

The simplest example of a WYSIWYM application just displays a WYSIWYM window, allows the user to construct a knowledge base within it, and then prints out the resulting knowledge base in XML format. This example serves primarily to show how the components of the library fit together, and how the configuration facilities can be used to control different aspects of the interface (components, menus, size, language etc.).

### 4.2 Database interface

This example extends the simple case, by interpreting the knowledge base returned as a specification of a database query to be executed. Thus WYSIWYM provides a natural language front-end for a database which is completely robust – the interface will not allow the user to construct an ill-formed query.

### 4.3 The PILLS demonstrator

A more complex application of the library is a recasting of the PILLS demonstrator, introduced above. Here the main WYSIWYM panel is deployed as a web applet in a browser, and additional functionality is provided for building WYSIWYM menus from file-based data (such as lists of ingredients), displaying generated documents in a companion browser frame, and loading and saving WYSIWYM knowledge bases.

## 5 Future plans

We are continuing to develop the idea of a direct-manipulation text-based interface in a number of directions. The main areas of development that may contribute to future version of the library are as follows.

**Coverage** Provision of a core grammar including an increased range of grammatical patterns, for example complex tenses, sentential complements, and relative clauses. This will enhance the system's ability to express data as text, in a way that is largely independent of the application-developer's interface (which would remain primarily concerned with creating an ontology and linking it to the basic linguistic resources).

**Organisation of options** Improvements to the menu-based option selection, such as ordering according to plausibility, or more advanced concept-browsing tools.

**Feedback text design** Since in WYSIWYM the system has control over the feedback text, usability depends on whether this text is well designed – an issue that requires empirical testing.

**Interface design** Use of drag-and-drop to speed up the operation of a coreferential copy-and-paste, and allowing the author to maintain a palette of icons for entities that are referenced often during the text (e.g., the patient and the product in a patient information leaflet).

### Implementation notes and availability

The WYSIWYM library currently runs under Java 1.4 (j2sdk 1.4.1) and SICStus prolog 3.9.1. Most of the basic prolog functionality is built into a SICStus delivery image, so a SICStus licence is not required for basic use. Ports to other prolog systems are also under consideration. The library will shortly be released on the ITRI website: <http://www.itri.brighton.ac.uk>

### Acknowledgements

The WYSIWYM library derives from code developed over a number of years by a number of people in addition to the authors. Principal among the programming credits are Chris Douce, Paul Piwek and Neil Tipper, and for grammar and domain development, Anja Belz, Nadjet Bouayad-Agha, Lynne Cahill and Paul Piwek. WYSIWYM has been developed in part through projects funded by the EPSRC (DRAFTER) and the EC (CLIME, PILLS).

### References

- 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.
- P. Clark, J. Thompson, K. Barker, B. Porter, V. Chaudhri, A. Rodriguez, J. Thomr, Y. Gil, and P. Hayes. 2001. Knowledge entry as the graphical assembly of components: The shaken system. In *Proceedings of the First International Conference on Knowledge Capture (K-CAP)*, Victoria, Canada.
- W. Grosso, H. Eriksson, R. Fergerson, J. Gennari, S. Tu, and M. Musen. 2000. Knowledge modeling at the millennium – the design and evolution of protege. In *Proceedings of the 12th International Workshop on Knowledge Acquisition, Modeling and Management (KAW'99)*, Banff, Canada.
- S. Paley. 1996. Generic knowledge-base editor user manual. Technical report, SRI International, California.
- Cécile Paris, Keith Vander Linden, Markus Fischer, Anthony Hartley, Lyn Pemberton, Richard Power, and Donia Scott. 1995. A support tool for writing multilingual instructions. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, pages 1398–1404, Montreal, Canada.
- P. Piwek, R. Evans, L. Cahill, and N. Tipper. 2000. Natural language generation in the MILE system. In *Proceedings of IMPACTS in NLG workshop*, pages 33–42, Schloss Dagstuhl, Germany.
- P. Piwek. 2000. A formal semantics for generating and editing plurals. In *Proceedings of COLING 2000*, pages 607–613, Saarbruecken, Germany.
- R. Power, D. Scott, and R. Evans. 1998. What you see is what you meant: direct knowledge editing with natural language feedback. In *Proceedings of the 13th Biennial European Conference on Artificial Intelligence*, pages 675–681, Brighton, UK.
- S. Schultz and U. Hahn. 2001. Medical knowledge re-engineering: Converting major portions of the umls into a terminological knowledge base. *International Journal of Medical Informatics*, 64/2-3:207–221.
- D. Suce and T. Lethbridge. 1995. CODE4: A unified system for managing conceptual knowledge. *International Journal of Human-Computer Studies*, 42:413–451.
- K. van Deemter and R. Power. 1998. Coreference in knowledge editing. In *Proceedings of the COLING-ACL workshop on the Computational Treatment of Nominals*, pages 56–60, Montreal, Canada.