GENERATION AS A SOLUTION TO ITS OWN PROBLEM

Donia Scott, Richard Power and Roger Evans
Information Technology Research Institute
University of Brighton
Brighton BN2 4GJ
email: {firstname.lastname}@itri.bton.ac.uk

Abstract

Natural language generation technology is now ripe for commercial exploitation, but one of the remaining bottlenecks is that of providing NLG systems with user-friendly interfaces for specifying the content of documents to be generated. We present here a new technique we have developed for providing such interfaces: WYSIWYM editing. WYSIWYM (What You See Is What You Meant) makes novel use of the system's generator to provide a natural language input device which requires no NL interpretation — only NL generation.

1 Introduction

It is generally agreed that the technology of natural language generation has evolved to a stage where it can feasibly be expected to be found in 'real world', applied systems. Indeed, within the last year there has been a specialist tutorial (Dale and Reiter, 1997) and a journal article (Reiter and Dale, 1997) aimed at guiding interested parties on how to build such systems; a textbook on this subject is also about to appear (Reiter and Dale, forthcoming). A problem that remains outstanding, however, is that of the input to NLG applications: where should we get it from and what should it look like (McDonald, 1993)?

A popular school of thought on this issue is echoed in the following quotation:

> **Rules of Thumb for NLG**
>
> NL generation only makes sense when
>
> 1) *The data to be communicated is already present in an existing database (knowledge base etc.)* (emphasis added)
> 2) The users need the information, and want it presented as text (e.g., instead of graphically)
> 3) The volume of documentation justifies the expense of building an NLG system.
>
> from Ehud Reiter, 'Using Natural Language Generation to Produce Real-World Documents'. Seminar presented at the ITRI, 1994

This first rule of thumb clearly does make good general sense, and a number of successful systems have followed it:

- ANA takes a set of temporally ordered stock quotes from the Dow Jones News and Information Retrieval System and produces a natural language report on the activities of the stock market over the period (Kukich, 1983).

- GOSSIP takes data from an audit trail of an operating system and produces a report (for a security officer) on user activity over the period (Carcagno and Iordanskaja, 1989).
- FOG takes data from a time series of weather depiction charts and produces a bilingual (French and English) weather report for the period (Goldberg et al, 1994).
- LFS takes statistical data from labour force surveys and from this produces a report on employment statistics over the given period (Iordanskaja et al, 1992).
- MODELEXPLAINER takes data from graphical object oriented data models and from this generates a textual description of the model (Lavoie et al, 1996).
- POSTGRAPHE takes tabular data (of the sort found in a typical spreadsheet) and generates a report integrating both graphics and text (Fasciano and Lapalme, 1996).
- PLANDOC takes the data from a simulation log file and from this produces a report of the explored simulation options (McKeown et al, 1994).
- ALETHGEN takes data from a customer database and produces a customised letter (in French) (Coch, 1996).
- A system developed by Johanna Moore and her colleagues at the University of Pittsburg takes the data from SAGE, a graphics presentation system (Roth et al, 1994), and produces an accompanying natural language caption (Mittal et al, in press).

However, it doesn't make the *only* sense. Some applications require the user to interact rather more closely with the semantic model that drives the generation process, and there is a small, but growing, number of systems that are able to provide this kind of interaction. They achieve this through a common solution: interfaces that allow the user to engage in *symbolic authoring* of the generated text.

- EXCLASS is an intelligent support tool for personnel officers writing (bilingual English and French) job descriptions. The user builds the job description by composing and editing conceptual representations; these representations are trees of concepts from a structured conceptual dictionary. Concepts are presented to the user through diagrammatic trees with natural language labels (Caldwell and Korelsky, 1994).
- DRAFTER-I is an authoring tool to support technical authors and software developers in writing (bilingual English and French) software manuals. The user directly builds the domain model (semantic knowledge base) describing the procedures for using a selected software application. As it is being constructed, the model is presented to the user through diagrams and fragments of text (Paris et al, 1995).
- GIST is an authoring tool to support forms designers. It generates (multilingual English, German, Italian) forms in the domain of social administration. The user's interaction with GIST is very similar to that with DRAFTER-I. (Power et al, 1995).
- A tool to support inventors in the authoring of patent claims allows the user to build a semantic model of the invented apparatus by selecting (via multiple-choice menu options) the apparatus parts, their functions and relations to each other (Sheremetyeva and Nirenburg, 1996).

These systems all comprise a natural language generator coupled to an interface that supports the manual creation of the generator's input (i.e., the authoring of the symbolic (conceptual) content of the output document).

In the remainder of this paper we describe a new type of symbolic authoring, WYSIWYM (What You See Is What You Meant) editing. A unique feature of WYSIWYM is that it uses the text generator not only to produce the output document but also to contribute to the editing of its own input specification.

2 WYSIWYM editing

WYSIWYM is a technique for creating and maintaining knowledge models (e.g., knowledge bases, databases, domain models) by direct knowledge editing with the benefit of dynamic natural language feedback. With WYSIWYM, each operation on the model results in a presentation in natural language of the current state of the model; this textual description also includes explicit information about what parts of the model remain incomplete (and, of those, which must be obligatorily completed). In this way, text generation technology is used as a means to present complex data entry tasks in a way which is both easy to understand and practical to work with.

The basic idea of WYSIWYM editing is that a special kind of natural language text is generated in order to present the current configuration. This text includes generic phrases called 'anchors' which mark attributes that have no value. The anchors serve as 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 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. The text associated with the fragment may or may not remain the same, depending on the context of the new location.

Although the user appears to be creating text, she is doing this only indirectly by creating knowledge; it is the system which creates the text. Whereas WYSIWYG editors (e.g., Microsoft Word, FRAMEMAKER and INTERLEAF) present the user with text as it will appear on the printed page, WYSIWYM editors present a text that reflects only what the user *meant* – not necessary what she will *get* as the final output.

3 Illustration of WYSIWYM editing

Our first application of WYSIWYM editing was in the context of the DRAFTER project, which developed a system to support the production of software documentation in English and French (e.g., Paris et al, 1995). 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-I, 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. Some authors found these diagrams hard to interpret and modify, so 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-II , in which the generator not only produces the final output texts, but also supports a WYSIWYM knowledge editor.

As a detailed example of WYSIWYM editing, we will describe a session in which a technical author uses DRAFTER-II 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 Calendar Manager's 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.

> **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. In the conventional setting, the technical author would be using a word processor to construct this information as text. With DRAFTER-II, the author will be operating at the symbolic (conceptual) level by interacting directly with 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.

We now follow stage by stage a session in which the procedure for scheduling an appointment was defined. Each stage is illustrated by the corresponding snapshot in Figure 2 (shown here on the last page)[1].

Stage 1 The process starts when the author selects the menu options `Input` modality and `New` file (not shown). In response, the system constructs a model of an empty procedure, i.e. one in which a goal and the methods of achieving it are undefined. From this model, the generator produces the single-sentence text shown in the figure. This text has special features related to its authoring function.

- The phrase "Do 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.
- The phrase "using 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.
- 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, and then the methods for achieving it.

[1] For of lack of space, we have had to cut parts of the snapshots and occassionally assume intermediate displays. This means that at times we refer to parts of the display not visible in the figure (e.g., the end of a long pop-up menu) or take the reader through a step that is not shown. We ask the reader's indulgence in this.

Stage 2 To begin the process of defining the goal (scheduling an appointment), the author clicks on the red anchor and obtains a pop-up menu listing the available action concepts. Near the bottom of this list (not shown) is the concept `schedule`. When this concept is selected, DRAFTER-II updates its model by filling the goal slot with a scheduling action, which includes a new slot for the event to be scheduled (as yet undefined).

Stage 3 From the updated model, a new text is generated and displayed. The green anchor remains the same, but the anchor "Do this action" is replaced by a phrase showing the information already defined (the scheduling action) in black, along with a new red anchor showing that the author must choose which kind of event is to be scheduled. 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`.

Stage 4 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 yield the text in snapshot 3, including the anchor "this event", so that the correct choice can be made. If the goal were completely wrong, the author could cut the span "Schedule the meeting", undoing both choices and returning to snapshot 1.

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 "using these methods" yields a single menu choice, which the author selects.

Stage 5 The inclusion of methods in the procedure requires a reorganization of the generated text.

- 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 an infinitival 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" can be opened 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 technique 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.

Stage 6 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. The transition from snapshot 5 to snapshot 6 could have occurred in several ways, since anchors can be developed in any order; one possibility is the following:

- The author develops the red anchor in the first step, defining the action of choosing the start time of the appointment.
- The author develops the optional anchor "Next step". This yields a text in which the second step reads "Do this action by using these methods", and a third step with the optional anchor "Next step" is added.
- The second and third steps are fully defined.
- The author remembers that a precondition is needed, and develops the anchor "perform this task" at the end of the second line in snapshot 5. The regenerated text shows the undefined goal and methods of the precondition, so that the sentence now reads " Before starting, do this action by using these methods".
- The goal of the precondition is fully defined, yielding the text shown in snapshot 6.

Stage 7 To add the last piece of information, the method for opening the Appointment Editor window, the author opens the anchor "using these methods" (snapshot 6). 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.

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 Appointment 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.

Stage 8 From snapshot 7, the author completes the model by developing the anchor "Do 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[2]. When a model is potentially complete, the author can switch the modality from `Input` 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.

Note that the output text in snapshot 8 is completely regenerated; it is not obtained merely by omitting the green anchors from the input text. 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 as in snapshot 7.

This output text is a computer-generated draft of part of the section of a user manual for the OpenWindows Calendar Manager that instructs users on how to schedule appointments.

[2]Note that the model was also potentially complete at snapshots 4 and 6.

4 Implementation

Figure 1 shows the basic architecture of a WYSIWYM editing system, including the following features:

- The only thing presented to the user is a text generated from the current domain model.
- The user can choose between input and output modality.
- The only way in which the user can edit the domain model is by selecting from pop-up menus on an input text.

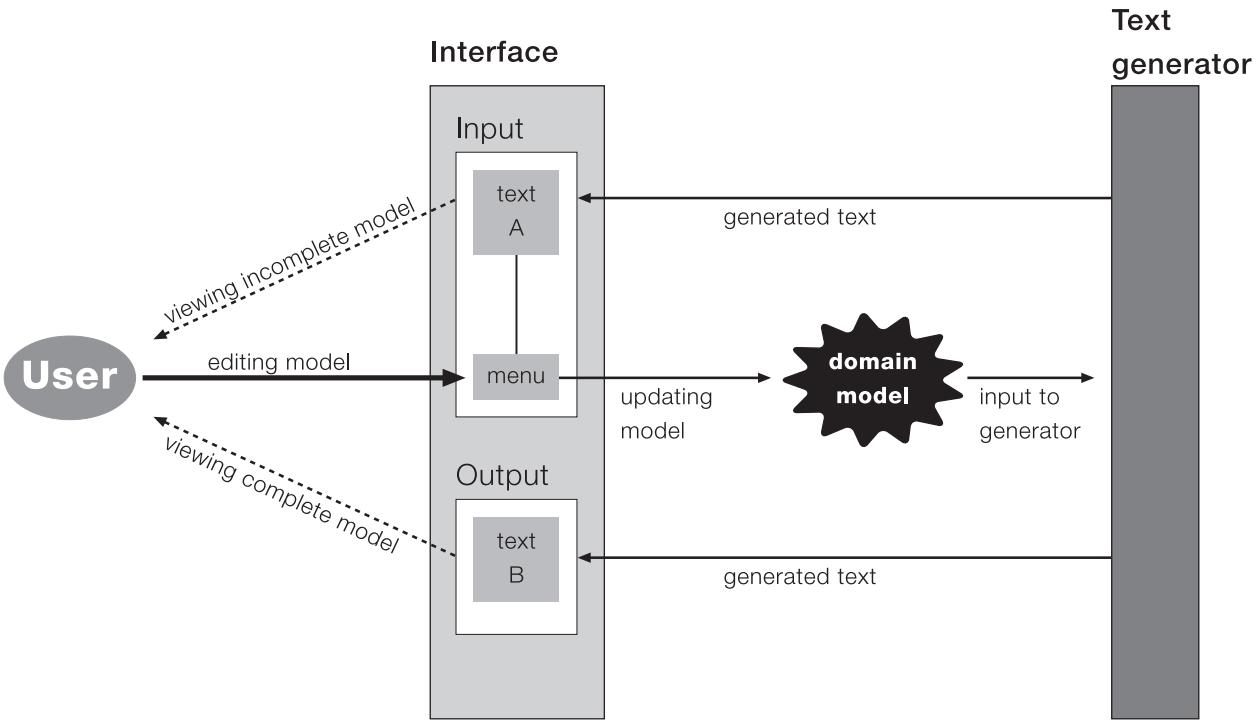

Figure 1: The WYSIWYM architecture

The text is completely regenerated every time the user changes the domain model or switches the modality. So far we have developed two experimental systems with this architecture. In DRAFTER-II, described above, the domain model and the generator are implemented in Prolog, while the interface is implemented in CLIM (CLIM 1994). In the other system, the Prolog generator produces HTML source files which can be read by a web browser. In both applications, texts several paragraphs long can be generated very quickly, so that when the model is changed the text seems to be updated instantaneously[3].

DRAFTER-II currently supports English, French and Italian. Switching from one language to another results in the immediate re-generation of the currently viewed text in the new chosen language. This takes no longer than the generation of a new text when the model is expanded during editing (as described in the steps of Figure 2).

[3]Very large domain models might require regeneration of just one part of the domain model at a time.

5 Conclusion

Natural language generation technology has till now been viewed exclusively as a method for producing textual *output* — answers to queries put to dialogue systems, report generation, software documentation, and the like. We have shown here that, with the use of WYSIWYM interfaces, the technology can be extended to include the *input* as well. WYSIWYM interfaces provide natural language input without the requirement of NL interpretation (i.e., no parsing takes place); instead, it makes use exclusively of NL generation. We believe that this method of input specification provides a new potential for natural language generation systems to be applied to a significantly wider range of 'real' applications than has previously been envisaged.

WYSIWYM editing is a new idea that requires practical testing. We have not yet carried out formal usability trials, nor investigated the design of input 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:

- WYSIWYM editors are extremely easy to use.
- 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-II, 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 (AECMA 1995).

References

AECMA *Simplified English: A Guide for the Preparation of Aircraft Maintenance Documentation in the International Aerospace Maintenance Language*, (1995) AECMA, Brussels.

Coch, J. (1996). Evaluating and comparing three text production techniques. *Proceedings of the Sixteenth International Conference on Computational Linguistics* (COLING-96), pp 249–254.

Caldwell, D. and Korelsky, T. (1994). Bilingual generation of job descriptions from quasi-conceptual forms. *Proceedings fo the Fourth Conference on Applied Natural Language Processing*, pp 1–6.

Carcagno, D. and Iordanskaja, L. (1989). Content Determination in Gossip. Extended Abstracts of the Second European Natural Language Generation Workshop, Edinburgh, pp 15–22.

CLIM *Common Lisp Interface Manager: User Guide* (1994). Franz Inc.

Dale, R. and Reiter, E. (1997). Building Applied Natural Language Generation Systems. Tutorial presented at the 1997 Applied Natural Language Processing Conference, Washington, DC.

Fasciano, M. and Lapalme, G. (1996). PostGraphe: a system for the generation of statistical graphics and text. *Proceedings of the Eighth International Workshop on Natural Language Generation*, Herstmonceux, Sussex, June 1996, pp 51–60

Goldberg, E., Driedger, N. and Kittredge, R. (1994). Using Natural-language Processing to Produce Weather Forcasts. *IEEE Expert*, 9 (2), pp 45–53.

Iordanskaja, L., Kim, M., Kittredge, R., Lavoie, B. and Polguere, A. (1992) Generation of Extended Bilingual Statistical Reports. *Proceedings of COLING-92*, Nantes, August 1992, pp 1019–1023

Kukich, K. (1983). Design and Implementation of a Knowledge-Based Report Generator. *Proceedings of the 21st Annual Meeting of the Association for Computational Linguistics*, Cambridge, Massachusetts, pp 145–150.

Lavoie, B., Rambow, O. and Reiter, E. (1996) The ModelExplainer. Demonstration presented at the Eighth International Workshop on Natural Language Generation, Herstmonceux, Sussex, June 1996.

McKeown, K., Kukich, K., and Shaw J. (1994). Practical Issues in Automatic Documentation Generation. *Proceedings of the Applied Natural Language Processing Conference*, Stuttgart, Germany, pp 7–14.

McDonald, D. (1993). Issues in the Choice of a Source for Natural Language Generation. *Computational Linguistics*, 19 (1), pp 191–197.

Mittal, V., Moore, J.D, Carenini, G., and Roth, S. (in press) Describing Complex Charts in Natural Language: A Caption Generation System *Computational Linguistics. Special Issue on Natural Language Generation*.

Paris, C., Vander Linden, K., Fischer, M., Hartley, A., Pemberton, L., Power, R., and Scott, D. (1995). A Support Tool for Writing Multilingual Instructions. *Proceedings of the Fourteenth International Joint Conference in Artificial Intelligence* (IJCAI-95), pp 1398–1404.

Power, R., Cavallotto, N., and Pemberton, L. (1995) The GIST Specification Tool, LRE Project 062-09, Deliverable PR-3b. Available at http://ecate.itc.it:1024/projects/gist/gist-bibliography.html

Reiter, E. and Dale, R. (1997). Building Applied Natutal Language Generation Systems. *Natural Language Engineering*, 3 (1): 57–87

Reiter, E. and Dale, R. (forthcoming). *Building Applied Natural Language Generation Systems*. Cambridge University Press.

Roth, S.F., Kolojejchick, J., Mattis, J. and Goldstein, J. (1994) Interactive graphic design using automatic presentation knowledge. *Proceedings of CHI'94: Human factors in Computing Systems*, Boston.

Sheremetyeva, S., Nirenburg, S. and Nirenburg, I. (1996) Generating Patent Claims from Interactive Input. *Proceedings of the Eighth International Workshop on Natural Language Generation*, Herstmonceux, Sussex, June 1996, pp 61–70.

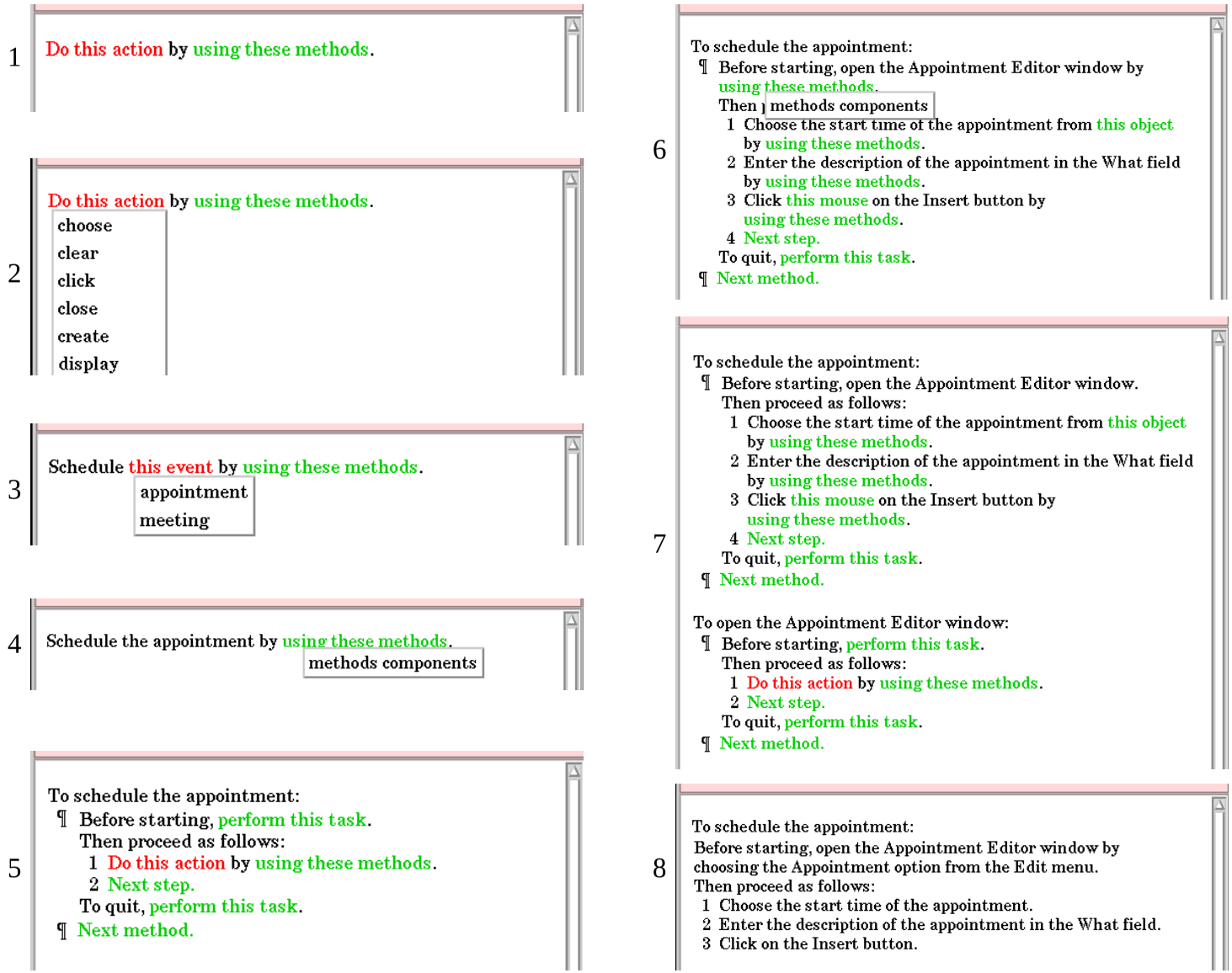

Figure 2: Stages in knowledge editing with WYSIWYM