# Dealing with Dependencies between Content Planning and Surface Realisation in a Pipeline Generation Architecture

**Kalina Bontcheva** and **Yorick Wilks**
Department of Computer Science, University of Sheffield, 211 Portobello St., Sheffield S1 4DP, UK
{kalina,yorick}@dcs.shef.ac.uk

## Abstract

The majority of existing language generation systems have a pipeline architecture which offers efficient sequential execution of modules, but does not allow decisions about text content to be revised in later stages. However, as exemplified in this paper, in some cases choosing appropriate content can depend on text length and formatting, which in a pipeline architecture are determined after content planning is completed. Unlike pipelines, interleaved and revision-based architectures can deal with such dependencies but tend to be more expensive computationally. Since our system needs to generate acceptable hypertext explanations reliably and quickly, the pipeline architecture was modified instead to allow additional content to be requested in later stages of the generation process if necessary.

## 1 Introduction

The astonishing explosion of the World Wide Web is leading to a growing need to personalise user experience in cyberspace (e.g., myYahoo, Amazon's book recommendations). Since research in Natural Language Generation (NLG) has already investigated ways to tailor automatically-generated texts to user goals and characteristics (e.g., [Paris, 1993; Zuckerman and McConachy, 1995]), these methods could be used to generate *dynamic hypertext*[1] which takes into account the interaction context and user preferences and characteristics.

Since users expect real-time interaction, efficient and robust applied NLG techniques are typically used for hypertext generation. For instance, ILEX [Knott *et al.*, 1996] uses a combination of canned stories and templates; EXEMPLARS [White, 1998] is rule-based; and PEBA [Milosavljevic *et al.*, 1996] uses text schemas [McKeown, 1986] and a phrasal lexicon. Also for efficiency reasons, dynamic hypertext generation systems have pipeline architectures where modules are executed sequentially and no module later in the architecture can request information from an earlier module. For example, in such an architecture it is not possible to take into account text formatting and length (which are determined towards the end) when choosing the text content (which happens in the beginning).

The goal of our dynamic hypertext generation system, HYLITE+, is to produce encyclopaedia-style explanations of domain terminology (see Figure 3 below). The corpus analysis of online encyclopaedia [Bontcheva, 2001] and previous empirical studies (e.g., [Reinking and Schreiner, 1985]) have shown the positive effect of additional information – e.g., definition of key vocabulary, less-technical content, supply of background information and illustrations – on the subjects' reading comprehension and reading behaviour. On the other hand, hypertext usability studies [Nielsen, 2000] have shown that people read 25% slower on the screen, so hypertext needs to be concise with formatting, that facilitates skimming. Our empirical studies have shown [Bontcheva, 2001] that users prefer different additional information depending on the chosen formatting and desired explanation length.

This paper discusses several ways to provide additional information about unknown terms in generated encyclopedic entity descriptions. When such information is needed, the most appropriate clarification needs to be chosen depending on formatting, user knowledge and constraints (e.g., concise versus detailed pages). Each alternative requires different text content to be selected at the start of the generation process but the choice of alternative can only happen after the content and formatting for the main description have already been determined. Therefore, the original pipeline architecture was extended to allow limited module feedback. In the resulting *recursive pipeline architecture* additional content can be requested in later stages of the generation process (e.g., during surface realisation).

In effect, the content planner first produces the text plan for the concise hypertext which contains only facts about the explained concept (e.g., personal computer). Then during surface realisation, after formatting has been decided, the most suitable adaptivity alternative is chosen. Often this leads to the posting of a new communicative goal, which results in expanding the basic text with extra information.

The paper is structured as follows. Section 2 describes briefly HYLITE+ – the dynamic hypertext generation system in the context of which the *recursive pipeline* architecture (Section 3) was developed. Section 4 exemplifies the use

[1] In *dynamic hypertext* page content and links are created on demand and are often adapted to the user and the previous interaction.

of the architecture for generating additional information that clarifies terms unknown to the user. The approach is also put in the context of previous work on interleaved, opportunistic, and revision-based language generation (Section 5). Finally the paper concludes with a discussion of some known limitations and future work.

## 2 HYLITE+ in a Nutshell

HYLITE+ generates encyclopedic explanations of terms in the chemical and computer domains. The user interacts with the system in an ordinary Web browser (e.g., Netscape, Internet Explorer) by specifying a term she wants to look up. The system generates a hypertext explanation where further information can be obtained by following links or specifying another query. Similar to all Web applications (see [Nielsen, 2000]), HYLITE+ needs to *(i)* respond in *real-time*, i.e., avoid algorithms with associated high computational cost; and *(ii)* be *robust*, i.e., always produce a response. Consequently the system uses some efficient and well-established applied NLG techniques such as text schemas and a phrasal lexicon (see [Reiter and Dale, 2000]).

Similar to other applied systems (see [Reiter, 1994]), HYLITE+ was initially implemented as a single-pass pipeline system, i.e., the generation modules were executed sequentially. The system input specifies the concept to be explained and the system parameters chosen by the user (e.g., concise versus detailed descriptions). The output is the generated hypertext explanation in HTML format, which is viewed by the user in a conventional browser (see Figure 3 below).

The system consists of several modules organised in two main stages: *(i)* content organisation, which includes *content selection*, *text organisation* and *semantic aggregation*; and *(ii) surface realisation* modules [Bontcheva, 2001]. As shown in Figure 1, the high-level modules use a discourse history and an agent model, ViewGen, [Ballim and Wilks, 1991] which contains both the system domain knowledge and user beliefs, stored in nested environments. The surface realisation modules use language-specific resources such as lexicons, morphology, and grammars.

The text planner uses high-level discourse patterns similar to text schemas [McKeown, 1986] which have been derived from analysing entries in encyclopedia and terminological dictionaries [Bontcheva, 2001]. For instance, entities (i.e., concepts inheriting from ENTITY in the hierarchy) are defined by their *supertype(s) or type definition*, *characteristics*, *functions*, *constituents* and *examples*. If the entity has several synonymous terms, the query one is used throughout the explanation and the rest are given in brackets when the entity is first introduced.

## 3 Adding Recursion to the Pipeline Architecture

One way of improving the user understanding of the generated hypertext is to clarify important unknown terms that occur in definitions, descriptions of parts and subtypes (for further detail see [Bontcheva, 2001]). However, different types of such information – definitions, familiar superconcepts, separate paragraphs, or just links – are preferred at different times, mainly depending on page length, formatting, and user preference for concise versus detailed explanations. In a sequentially organised pipeline architecture some of this information can be generated only if the necessary additional facts have already been extracted during content selection and incorporated in the text plan.

For instance, the decision whether it is appropriate to use a parenthetical definition or a paragraph-length description depends on their length, the chosen hypertext formatting, and the target length for the generated page[2]. However, formatting and length information are determined during surface realisation, not content planning, so HYLITE+ either needs to approximate the text length from the number of facts to be realised, or it has to extract clarifying information for each unknown concept in advance, or the architecture needs to be modified to allow such information to be requested at a later stage.

As shown by [Reiter, 2000], approximating the text length during content planning is possible but suffers from two problems. The first one is that the result is not exact, so when formatting is added later the text might not fit into the page any more. For example, if the surface realiser decides to use a bullet list when enumerating five items, instead of a sentence with conjunctions, the resulting text length on the screen will increase in comparison to the estimated one. Consequently, if there are constraints to generate text that fits on certain number of pages, the approximation method is problematic. The second, less-significant problem with this method is that it could be to maintain and update because each change to the realiser needs to be reflected in the approximation algorithm. Finally, the approximation method in systems that use a phrasal lexicon might not work very well, unless length information is encoded explicitly for each lexical entry.

Selecting all potentially useful clarifying information in advance is going to be more computationally expensive than adding it on demand, particularly as the text size grows. Still, because the recursion in clarifications is limited, the text size will grow linearly, not exponentially, so this approach is feasible, even if not optimal. The problem here is that unless length approximation is also implemented, the content planner still cannot commit to a particular strategy (e.g., a parenthetical definition or a longer paragraph after main explanation) because it needs to know, among other things, the overall text length and the length of the definition text itself.[3]

Another alternative is to implement a text revision mechanism to analyse the content and structure of the generated hypertext and choose between alternative ways of including relevant information. However, despite the gains in fluency and coherence, revision-based approaches tend to suffer from computational problems due to the large number of alternatives that need to be explored (e.g., [Robin and McKeown, 1996]).

[2] [Bontcheva, 2001] discusses the results of an empirical study of user acceptance of hypertext adaptivity techniques and the influence of length, formatting, and user characteristics on this choice.

[3] The user study showed that definitions longer than 10-15 words should not be provided in parenthesis because they disturb the text flow.

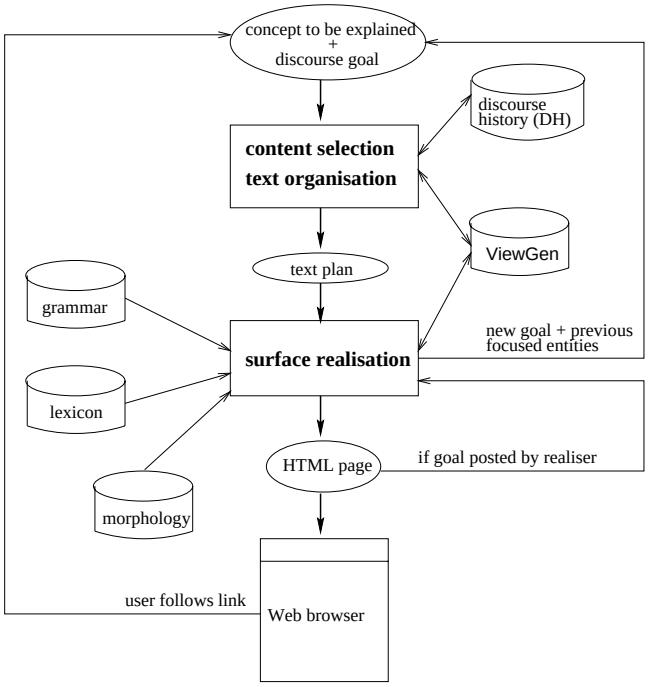


Figure 1: The HYLITE+ recursive pipeline architecture

Also, since HYLITE+ was built on the basis of an existing pipeline-based generation system, our goal was to find a solution that would both provide the needed module feedback functionality and work in combination with the existing sequentially-designed modules. In other words, we needed a modification of the pipeline, that will allow the surface realiser to invoke the generator with new goals, so additional information is extracted and included only when appropriate.

Consequently, the existing pipeline architecture was modified to allow later stages to request additional information to be included (see Figure 1). A special monitoring module was added to detect opportunities for adapting the generated hypertext. The monitor uses the complete text plan, received from the content planning stage, together with the propositions realised to this point to estimate the length of the main explanation. Based on this information, user preferences, and the chosen document formatting, the monitor decides whether to post additional high-priority goals on the generator's agenda (e.g., `define(microprocessor)`: a goal to define a concept, `example(microprocessor)`: give an example). The generator keeps the goals on its agenda (which is a priority queue) and realises them one after another, so no two goals can be active simultaneously.

When high-priority goals are posted to the generator, surface realisation is temporarily suspended while the text for the new goal is generated. This effectively opens a new discourse segment [Grosz and Sidner, 1986], which is dominated by the main explanation. All previously mentioned entities become part of the new focus space, so the generator can refer to them if necessary. The monitor also specifies some system parameters which influence the form of the newly generated text. For example, term definitions in brackets need to be noun phrases, instead of full sentences (see example in Figure 2), so the system configuration is changed from the default preference for sentences. When the text corresponding to the new goal is generated, the monitoring module evaluates the result and, if suitable, integrates it with the main text. Finally, text realisation is resumed until a new interruption occurs or the entire text plan has been verbalised.

If the newly generated text is found to be unsuitable (e.g., the definition is too long and will disturb the flow of the main text), the monitoring module tries another adaptivity technique if such is available (e.g., provide a known superconcept instead). Otherwise it leaves the text content as originally planned and the unknown concept is realised with its corresponding term with a hypertext link to a separate page.

In effect, the content planner first produces the text plan for the concise, unadapted hypertext which contains only facts about the explained concept (e.g., personal computer). Then during surface realisation, when formatting has been decided, the most suitable adaptivity alternative is chosen. In some cases this leads to the posting of a new communicative goal to the generator, which results in expanding the basic text with extra information.

Below we will show how the recursive pipeline is used for providing clarifying information about subtypes, parts, and unknown concepts in term definitions. The present experience shows that the efficiency of the pipeline architecture and the schemas is retained while the generator's scope and flexibility is improved.

## 4 Clarifying Unknown Terms

The content planner uses information from the user model to detect and annotate unknown concepts [Bontcheva and Wilks, 1999]. During surface realisation hypertext links, leading to separate explanation pages, are added for these concepts. In addition, some clarifying information is considered for unknown concepts in definitions, parts and subtypes.

The system mockup experiments [Bontcheva, 2001] showed that users prefer two types of concise parenthetical information: brief term definitions and a familiar superconcept. For longer texts, paragraph-long clarifying information can be provided in a separate section, containing more detail than just the definition/superconcept. Here we will only provide examples of using the recursive pipeline architecture to generate additional term definitions and paragraph-long explanations. Further details and a thorough discussion of the other adaptivity techniques are available in [Bontcheva, 2001].

### 4.1 Generating the definitions

Let us assume a user who has looked up computer programs and then followed a link to personal computer; the user has not specified preferences for types of clarifying information, so definitions can be provided where appropriate. Following this request, the content planner passes the following facts for realisation (the fact listing all parts is truncated here[4]):

[4]The facts are encoded as conceptual graphs, a type of semantic network, where concepts are written in square brackets and relations in round ones. Extra information (e.g., whether a concept is familiar to the user) can be associated with concepts and graphs as feature

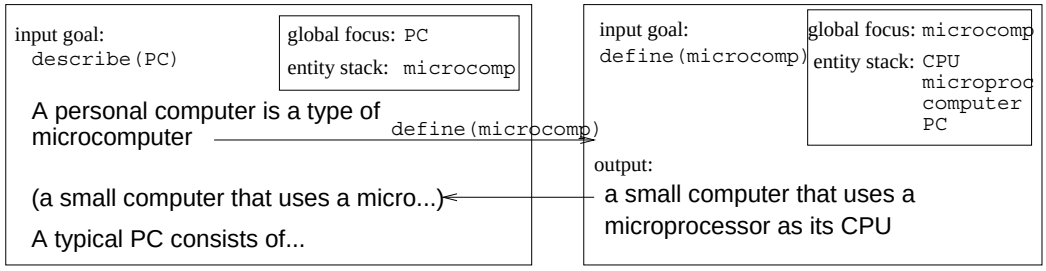


Figure 2: An example of clarifying information added during surface realisation

[PC] <– (ISA) <– [MICRO_COMP *fs(um_state:unknown)*].

[PC] <– (PART_OF) <– [CPU *fs(um_state:unknown)*]
– (PART_OF) <– [MEMORY *fs(um_state:unknown)*]
– (PART_OF) <– [HDD *fs(um_state:unknown)*]
– (PART_OF) <– [DISPLAY]...

First the realiser determines the document formatting; in this case a bullet list is chosen to enumerate all parts. Then it starts generating text for the first graph. Because the introduced supertype is unknown, but important for the understanding of the text, the realisation monitor decides to provide some extra material about it. The action corresponding to generating a new definition is `define(micro_comp)`, which is passed as a parameter to the recursively called generator. The generation parameters are also set to very short texts with syntactic preference for noun phrases. The resulting text – "a small computer that uses a microprocessor as its central processing unit" – is evaluated for length and provided in brackets (see Figures 2 and 3).

Similarly for all unknown parts of the PC, the realisation monitor calls the generator recursively and obtains their definitions. Since the parts are described in a bullet list, it is more appropriate to provide their definitions with dashes, instead of brackets (see Figure 3).

The realisation monitor has a set of rules which examine the type of proposition (e.g., `definition`, `part_of`, `attribute`) and the formatting of the main text, in order to determine the best formatting for the new material. At present, new material which is to be integrated in unformatted text is put in brackets. Since such parentheses disturb the flow of the main text, they are only provided once per sentence, usually for the most important concept in the proposition (e.g., the supertype). Other important unknown concepts in the same sentence are supplemented with a familiar supertype.

When a concept has been defined in brackets, the generated hypertext link also has a tag saying *Further information* to indicate that more information is available and can be reached by following the link (see Figure 3). If the knowledge base contains only the definition and no other information, then no link is provided because all relevant material has already been included in the current page.

structures.

## 4.2 Generating Paragraph-Length Clarifying Descriptions

So far we have discussed the generation of concise hypertext, where the main `describe(entity)` goal is realised in a schema-based, sequential fashion and brief additional definitions of unknown terms are included by posting additional high-priority goals. However, our empirical studies showed that in some cases users can prefer longer texts.

The corpus study showed that long encyclopedic articles often provide detailed descriptions of subtypes and/or object parts. Therefore, one way to adapt longer generated hypertext is to provide paragraph-length descriptions of the unknown subtypes/parts, instead of including just their definitions or familiar supertypes. The experiments showed that such pages are preferred when users need more detailed information on the topic (e.g., for a school essay); in this case, the additional material is best organised in a separate section.

HYLITE+ generates such longer additional material in separate sections based on new communicative goals posted on its agenda by the monitor (see Figure 1). For example, if the main explanation contains propositions about several unknown object parts (e.g., computer parts) and more detailed texts are preferred by the user, a new `describe_all(part_of, computer)` goal is posted on the agenda. In this case, the goal is not marked as a high-priority one, so surface realisation is not interrupted. After the main explanation is completed, the generator fetches the new goal and starts a new section in the document describing all computer parts.

The `describe_all(Rel, X)` goal is decomposed in the following way:

```
forall C where Rel(C, X)
                describe(C)
```

i.e., describe all concepts `C` for which the relation `Rel` holds with X, e.g. describe all parts of the computer. Apart from the new goal, the generator is also passed parameters that specify to generate the explanations as paragraphs, not whole pages.

In this way the generated document is regarded as an ordered set of goals and the text for each one of them is generated separately. This separation approach has the benefit of breaking down the text organisation problem into smaller parts which *(i)* can be more efficient to compute, but also *(ii)* can use different text organisation approaches for the different parts, e.g., schemas for the more 'rigid' parts and planning otherwise.

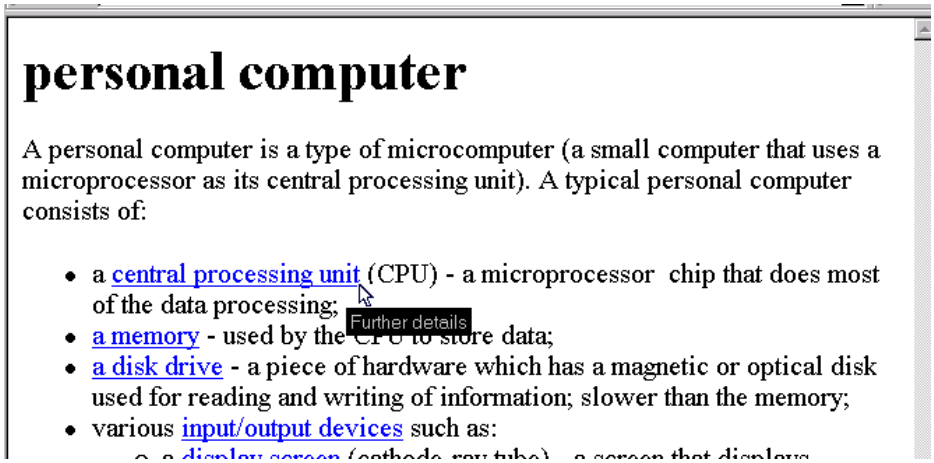


Figure 3: Example of clarifying definitions integrated in the main text

## 5 Related Work

The way in which the recursive pipeline architecture handles the dependencies between content planning and surface realisation bears some similarity with *interleaved language generation*. For example, [Rubinoff, 1992] describes an approach where the planner obtains feedback from the linguistic component in the form of a set of possible ways of expressing each plan element; the most suitable one is then added to the utterance. Some interleaved systems (e.g., [Finkler and Neumann, 1989; De Smedt, 1990]) are also *incremental*, i.e., their planner and realiser operate in parallel on different segments of an utterance. The most similar interleaved system is PAULINE [Hovy, 1988] where the planner produces only a partial text plan and makes the remaining commitments only when the realisation component needs them. In this way unexpected syntactic opportunities can be exploited and long-term goals (e.g., conciseness, politeness) taken into account during planning as well as realisation.

The main difference between the recursive pipeline approach and interleaved systems is that the latter tend to use feedback for every choice that depends on information from another module. In HYLITE+ the main text is always produced in a sequential fashion; interleaved planning and realisation only occur if the user preferences allow the use of adaptivity techniques that lead to new goals. Although potentially less flexible, the recursive approach enables the use of efficient techniques (schemas, sequential execution) to compute the main text reliably and quickly.

Another approach to the content adaptivity problem was explored in the ILEX system [Mellish *et al.*, 1998], which uses *opportunistic* planning to tailor the hypertext descriptions of museum exhibits. The planning mechanism is based on a structure called *text potential* – a graph of facts connected with thematic and rhetorical relations. Facts are chosen depending on their connection to the focus of the page (the described entity), interest and importance weights. This approach, however, seems difficult to apply to generated encyclopaedic-style explanations because it requires the creation of all links and weights in the text potential. Assigning values for importance and interest are particularly difficult because they both depend on the user goal (which is unknown to the system) and can change from one session to another (e.g., in-depth reading for a school essay versus looking up an unknown term).

The pipeline-based STOP system [Reiter, 2000] operates under very strict text size constraints which proved difficult to estimate at the content planning stage. In order to improve the generation results, Reiter experimented with a multiple-solution pipeline[5] (MSP) and revision-based variants of the system. The results showed that multiple-solution pipelines (for 4 solutions) and revision-based NLG are best suited for the task. The processing time per document is naturally higher – linearly increasing with the number of generated alternatives for the multiple-solution approach and the revision-based one is even slower than the MSP. The recursive pipeline architecture discussed here might not always produce the best texts under such tight constraints because the length of the yet unrealised propositions can only be approximated.

## 6 Conclusion and Future Work

This paper presented the *recursive pipeline architecture* developed as part of the dynamic hypertext generation system HYLITE+. The approach allows additional content to be requested in later stages of the generation process and integrated in the main explanation. The architecture also allows new goals to be executed after the main body of the explanation has been generated.

Although interleaved and revision-based approaches offer more flexibility, the advantages of the recursive pipeline is that it can be added to an already implemented sequential system with relatively minor modifications to the code. In addition, with our approach the main text is always computed in a sequential fashion, based on efficient applied NLG techniques (e.g., schemas, phrasal lexicon). Interleaved planning

[5] Several different document plans are produced and after surface realisation a choice module selects the one that provides most content within the given size limit.

and realisation only occur if the user preferences allow the use of adaptivity techniques that lead to new goals.

In the case when the additional text is generated during surface realisation, the monitor determines the overall text length based on the length of the already realised facts plus an estimated length for all unrealised ones. Therefore the result is only an approximation of the final length, which is sufficient when, as in hypertext, size constraints are important but not very strict.

The results from a small-scale formative evaluation have confirmed that the majority of users prefer the adapted texts to the neutral version where no additional information is provided for the unknown terms. More exhaustive subject-based evaluation is currently being undertaken and the goal is to gain further insight into the users' acceptance of hypertext adaptivity.

Performance-based evaluation of the recursive pipeline versus the baseline system (a version that does not provide additional information, just generates the main explanation) revealed that the new goals only add between 5% and 20% to the overall execution time – e.g., 5-10% for additional information which is less than 33% of the baseline explanation; 20% for additional information which is 50-60% of the baseline one. With both systems the overall processing time for a half page of hypertext is less than 1.5 seconds – a response time which allows users to stay focused on the interaction without any need for extra feedback [Nielsen, 2000].

In the future we plan to experiment with using different content organisation strategies to generate different parts of the text. For example, following the ideas of [Paris, 1993] different kinds of knowledge can be provided depending on the user's level of expertise. Also, in some cases schemas can be replaced with text planning based on rhetorical relations, e.g., [Moore, 1995; Power, 2000]. In this way the more 'rigid' parts of the text can be generated efficiently with schemas, while more powerful but computationally expensive planning techniques are used only when necessary.

**Acknowledgements**

We wish to thank Hamish Cunningham and the anonymous reviewers for their helpful comments and suggestions.

## References

[Ballim and Wilks, 1991] A. Ballim and Y. Wilks. *Artificial Believers*. Lawrence Erlbaum Associates, Hillsdale, New Jersey, 1991.

[Bontcheva and Wilks, 1999] Kalina Bontcheva and Yorick Wilks. Generation of adaptive (hyper)text explanations with an agent model. In *Proceedings of the European Workshop on Natural Language Generation (EWNLG'99)*, Toulouse, France, May 1999.

[Bontcheva, 2001] Kalina Bontcheva. *Generating Adaptive Hypertext Explanations with a Nested Agent Model*. PhD thesis, University of Sheffield, 2001. Forthcoming.

[De Smedt, 1990] Koenraad De Smedt. IPF: An incremental parallel formulator. In Robert Dale, Chris Mellish, and Michael Zock, editors, *Current Research in Natural Language Generation*, pages 167–192. Academic Press, New York, 1990.

[Finkler and Neumann, 1989] Wolfgang Finkler and Günter Neumann. POPEL-HOW: A distributed parallel model for incremental natural language production with feedback. In *Proceedings of the 11th International Joint Conference on Artificial Intelligence (IJCAI-89)*, volume 2, pages 1518–1523, Detroit, MI, August 20-25, 1989.

[Grosz and Sidner, 1986] Barbara J. Grosz and Candace L. Sidner. Attention, intentions and the structure of discourse. *Computational Linguistics Journal*, 12(3):175–204, 1986.

[Hovy, 1988] Eduard H. Hovy. *Generating natural language under pragmatic constraints*. Lawrence Erlbaum, Hillsdale, New Jersey, 1988.

[Knott *et al.*, 1996] Alistair Knott, Chris Mellish, Jon Oberlander, and Mick O'Donnell. Sources of flexibility in dynamic hypertext generation. In *Proceedings of the 8th International Workshop on Natural Language Generation (INLG'96)*, 1996.

[McKeown, 1986] Kathleen R McKeown. Discourse strategies for generating natural-language text. In B. L. Webber B. Grosz, K. S. Jones, editor, *Readings in Natural Language Processing*. Morgan Kaufmann Publishers, 1986.

[Mellish *et al.*, 1998] Chris Mellish, Mick O'Donnell, Jon Oberlander, and Alistair Knott. An architecture for opportunistic text generation. In *Proceedings of the International Natural Language Generation Workshop IWNLG'98*, 1998.

[Milosavljevic *et al.*, 1996] Maria Milosavljevic, Adrian Tulloch, and Robert Dale. Text Generation in a Dynamic Hypertext Environment. In *Proc. of 19th Australian Computer Science Conference*, Melbourne, 1996.

[Moore, 1995] Johanna D. Moore. *Participating in Explanatory Dialogues*. MIT Press, Cambridge, MA, 1995.

[Nielsen, 2000] Jakob Nielsen. *Designing Web Usability: The Practice of Simplicity*. New Riders Publishing, 2000.

[Paris, 1993] Cécile L. Paris. *User modelling in text generation*. Francis Pinter Publishers, London, 1993.

[Power, 2000] Richard Power. Planning by constraint satisfaction. In *Proceedings of COLING'2000*, 2000.

[Reinking and Schreiner, 1985] David Reinking and Robert Schreiner. The effects of computer-mediated text on measures of reading comprehension and reading behaviour. *Reading Research Quarterly*, Fall:536—551, 1985.

[Reiter and Dale, 2000] Ehud Reiter and Robert Dale. *Building Natural Language Generation Systems*. Cambridge University Press, Cambridge, U.K., 2000.

[Reiter, 1994] Ehud Reiter. Has a Consensus NL Generation Architecture Appeared, and is it Psycholinguistically Plausible? In *Proceedings of 7th Int. Workshop on NL Generation (INLG-94)*, pages 163–170, Kennebunkport, Maine, USA, 1994.

[Reiter, 2000] Ehud Reiter. Pipelines and size constraints. *Computational Linguistics*, 26:251–259, 2000.

[Robin and McKeown, 1996] Jacques Robin and Kathy McKeown. Empirically designing and evaluating a new

revision-based model for summary generation. *Artificial Intelligence*, 85(1-2), 1996.

[Rubinoff, 1992] Robert Rubinoff. Integrating text planning and linguistic choice. In *Aspects of Automated Natural Language Generation*, Lecture Notes in Artificial Intelligence, 587, pages 45–56. Springer Verlag, Berlin, April 1992.

[White, 1998] Michael White. Designing dynamic hypertext. In *2nd Workshop on Adaptive Hypertext and Hypermedia*, June 1998. Held in conjunction with Hypertext'98, Pittsburgh, USA.

[Zuckerman and McConachy, 1995] Ingrid Zuckerman and Richard McConachy. Generating explanations across several user models: Maximizing belief while avoiding boredom and overload. In *Proceedings of 5th European Workshop on Natural Language Generation (EWNLG-95)*, 1995.