In search of a reference architecture for NLG systems

Lynne Cahill† Christy Doran† Roger Evans† Chris Mellish‡ Daniel Paiva†
Mike Reape‡ Donia Scott† Neil Tipper†

†Information Technology
Research Institute
University of Brighton
Lewes Rd, Brighton
UK

‡Division of Informatics
& Human Communication Research Centre
University of Edinburgh
80 South Bridge, Edinburgh
UK

rags@itri.brighton.ac.uk
http:/www.itri.brighton.ac.uk/projects/rags

Abstract

This paper addresses the question of whether Reiter's 'Consensus NL Generation Architecture' [Rei94] really exists, and if so whether it is a suitable candidate for a 'reference architecture' for NLG systems. Our answer to the first question is a tentative yes, but we are less comfortable to accept the second. In pursuit of a better understanding, we develop an alternative style of architecture model, and while this also has its shortcomings, the insights gained lead us to propose a hybrid strategy for the development of a reference architecture.

1 Introduction

The primary goal of the RAGS[1] project is to develop a 'reference architecture' for applied natural language generation (NLG) systems. Our aim is to produce an architectural specification which reflects mainstream current practice and provides a framework for the development of new applications and new components within NLG systems. To achieve this, such an architecture has to be sufficiently conventional to be relevant to developers of existing systems, but also sufficiently generic and detailed to be useful as a resource for novel approaches.

Our starting point in this endeavour is the observation made by Reiter [Rei94] that a consensus was emerging among developers about the overall architecture of applied NLG systems. If the observation is correct, then such a consensus architecture would surely make a strong claim to satisfy the relevance requirement of our reference architecture. If in addition the architecture was, or could be filled out to be, sufficiently generic and detailed, then it would fulfill the requirements of our research programme.

This paper describes our progress to date in pursuing this line of argument. We describe our attempts to validate Reiter's claim against a larger sample of NLG systems (nineteen as opposed to Reiter's five) and conclude that it is broadly supported. We then consider the issue of genericity and detail to a first approximation and find the architecture less satisfactory, but find sufficient evidence to suggest that a more suitable refinement is possible. We conclude by sketching our current view on what this refined architecture might be like. A more detailed discussion of the analysis of systems on which these findings are based can be found in [Pai98].

Before proceeding, however, a few words about other architectural issues in NLG are in order. There are, of course, numerous published accounts of generation systems whose architecture differs slightly or radically from a simple pipeline. It is not our intention to undermine this research in any way, or to suggest that a pipeline has some special theoretical status for generation. The present exercise, and indeed the project as a whole, seeks to make more explicit the idea that there are striking commonalities among many applied NLG systems, in the hope that this will be a useful aid to further development. Our starting point is a simple pipeline, but our finishing point may well not be, and indeed is not, even in the early stages described below.

[1] RAGS (*Reference Architecture for Generation Systems*) is a research project supported by the EPSRC under grants GR/L77041 to Chris Mellish (Edinburgh) and GR/L77102 to Donia Scott (Brighton).

Figure 1: Reiter's pipeline architecture

2 Reiter's pipeline architecture

The architecture described by Reiter is given in figure 1. According to this model, an NLG system consists of three modules organised in a strict pipeline: content determination, followed by sentence planning, followed by surface realisation[2]. Here content determination takes some form of input and produces semantic and rhetorical structure; sentence planning maps such structures into linguistic ones, including processes such as referring expressions, clause and sentence grouping and choosing grammatical relations; surface realisation takes such abstract syntactic structures and produces actual text. For ease of reference, we shall call this architecture the 'standard pipeline architecture.'

One immediate observation about this architecture is worth noting at the outset. Notwithstanding the apparent structure shown in figure 1, the architecture is really defined by its data interfaces rather than its processing modules. Terms such as 'content determination', 'sentence planning' and 'surface realisation' are quite vague and applied differently, if at all, by different researchers. What seems to be more concrete in the architecture is the development of *data structures*: from input, through semantic/ rhetorical structure, then linguistic/ syntactic structure to text. It is this aspect which proves to be a more useful basis for analysing real systems.

3 Validating the pipeline architecture

3.1 The test set

The first requirement for our investigations was to establish a collection of systems to use as a standard test set. In order to achieve some degree of coherence in this set, we imposed the following selection criteria:

1. they had to be fully implemented;
2. they had to be complete, i.e., perform complete generation of text from non-textual input;
3. they had to take non-hand-crafted input.

[2]Reiter calls the latter *surface generation*, and also identifies Morphology and Formatting modules, but says very little about them.

Through a comprehensive (but not exhaustive) survey of the NLG literature, and also interactions with the developers of many of the systems, we established a set of nineteen systems meeting these criteria. These systems are described in [Pai98], and were chosen on the basis of availability of information rather than because we thought they had some special representative status. The nineteen systems[3] fall into two subgroups according to the ultimate source of their input: nine systems take input created by a user using some input interface (GIST, Drafter, Drafter2, Patent Claim Expert, Joyce, ModEx, Exclass, HealthDoc and Ghostwriter), and ten systems take input from a knowledge or database (Ana, PlanDoc, FoG, Gossip, LFS, Caption Generation System (CGS), PostGraphe, Komet, AlethGen and Proverb).

3.2 Validation

Two steps were required to validate the systems against the pipeline model. First, we had to analyse each system in terms of its own internal architecture. That is, we attempted to reconstruct the overall architecture of the systems, drawing on written descriptions and where possible filling the gaps by consulting the implementors. In some cases, there were good descriptions of a system's architecture, but in others it was quite difficult to sort out all the details, even at a high level.

The second step was to compare the resultant architecture with the standard pipeline, both in terms of structure (how close is the system's architecture to a simple pipeline?) and content (how well do the system's modules map onto the functionalities of Content Determination, Sentence Planning and Surface Realisation, or more accurately, to the phases of data processing identified by the standard pipeline).

The first conclusion we drew was that almost all the systems adopted a pipeline architecture. The most significant exception was Komet ([BT95, TB94]),

[3]See figures 2 and 3 for individual references to systems.

System	GIST [Con96, PC96]	Drafter [PVF+95]	Drafter2 [PS98, SPE98]	Pat Claim [SN96]	Joyce [KKR91, Ram]	ModEx [LRR97, LR97]	Exclass [CK94]	HealthDoc [DHW95, HDHP97]	Ghostwriter [MCM96, Cer96]
Content Determination	User interface Strategic planner	User interface Strategic planner	Authoring tool	Authoring module Text planner	Text planner	Text planner	Job desc builder	Authoring tool Selection of content	Text planner
Sentence Planning	Strategic planner	Micro planner	Generator	Realizer	Sentence planner	Sentence planner	Job desc builder	Sentence repair	Sentence planner
Surface Realisation	Tactical gen.	Tactical gen.	Generator	Realizer	Realizer	Realizer Formatter	Job desc gen.	Realization and formatting	Surface realization Post editing

Figure 2: Correspondence to standard pipeline – user input systems

System	Ana [Kuk88]	PlanDoc [KMS+93, MKS94]	FoG [GDK94, BCG+90]	Gossip [CI89, IKP91]	LFS [IKLP92, KP91]	CGS [MMCRng, MRM+95]	PostGraphe [FL96, GL96]	AlethGen [Coc96, CD94]	Proverb [HF96, HF97]
Content Determination	Fact gen. Message gen. Discourse organiser	Message gen. Ontologizer Content planner	Conceptualizer Planner	Planner	Planner Interlingual component	Text planner Ordering Aggregation	Planning module	Conceptual planner	Macroplanner
Sentence Planning		Lexicalizer	Interlingual component Deep-syntax	*Unnamed1* *Unnamed2*	Interlingual component Semantics Deep-syntax	Centering Referring Expression Lexical choice		Rhetorical planner NP/anaphora planner	Microplanner
Surface Realisation	Surface gen.	Surface ge.	Surface-syntax Morphology	Linguistic module	Deep-syntax Surface-syntax Morphology	Lexical choice FUF/SURGE	Realization module	Linguistic realization	Realizer

Figure 3: Correspondence to standard pipeline – data input systems

whose modular structure appears to be representational rather than procedural: processing develops different levels of analysis somewhat in parallel. There was considerable variation within the pipeline itself, with different numbers of explicit modules and no particular standardisation of terminology or module function. Nevertheless we were able to construct a reasonable mapping from each system's own modules to the standard pipeline modules. This mapping is shown in figures 2 and 3, where the modules of each system are divided into three groups corresponding to the three standard pipeline architecture modules[4].

From these tables we can identify a number of different phenomena occurring. Some systems, for example Drafter, are a perfect fit to the standard pipeline, in that all their modules fit within the three standard modules (although not necessarily one to one). Others, such as Drafter2, fit the standard pipeline, but are *less specific* than it – they have one module spanning two standard pipeline modules. Still other systems, such as GIST, require a *more specific* architecture – they have modules which appear to cut across standard pipeline modules. And finally a few systems, such as PostGraphe, appear to have gaps in the architecture – apparently missing functional components. Figure 4 summarises these different groups:

	no. of systems	*% of total*
Perfect fit	10	55
Less specific	3	17
More specific	3	17
Gaps	2	11
Total	**18**	**100**

Figure 4: Breakdown of system matches to standard pipeline

Drawing firm conclusions from these results is difficult, especially considering that in some cases judgements about architecture mappings were made on the basis of limited available knowledge. Nevertheless, they seem to lend reasonable support to Reiter's original claim, and certainly argue for the relevance of his architecture, or something quite close to it, to current systems.

[4]From this point we exclude Komet from consideration because we lack confidence in any pipeline-based analysis of its processing. Komet's architecture seems to us to be sufficiently different from the rest of the test set that more useful progress in the short term will be made by deferring consideration of it to a later date.

3.3 Genericity and detail

The standard pipeline architecture might be representative of current NLG systems, but as we discussed above, this on its own is not enough for our purposes. We also require the architecture to be generic and detailed. Unfortunately we have reservations about the standard pipeline on both counts, and in this section we discuss our difficulties and propose a way out of them.

Taking genericity first, we note that the pipeline model is essentially an emergent one; that is, it has arisen out of looking at actual NLG systems, seeing what they do and abstracting from it (roughly speaking). While this is a useful exercise, and the research underlying the systems it is developed from is clearly sound, it is not, to our mind, as unbiased as it might be. If one were to consider the problem of natural language generation from 'first principles' as a primarily linguistic task, it seems to us unlikely that one would necessarily think in terms of content determination, sentence planning and surface realisation as the basic building blocks. It is more likely that one would think in terms of more specific linguistic functions such as 'lexicalisation' 'referring expression construction' or 'aggregation'. A reference architecture expressed in such terms is likely to be more generally useful.

In addition, we have already seen evidence in our test sample that the standard pipeline is not detailed enough to support some existing systems. A finer level of detail in the pipeline might resolve this problem, if the correspondences between systems' own modules and standard ones still survive, but it is also possible, perhaps even likely, that at any finer level of detail the strict pipeline generalisation breaks down – more flexibility of ordering and interaction between modules is required.

4 A functional model

To address these issues we introduced an alternative model of NLG, which we intended to be more generic and also lower level. This model is based on the *linguistic* operations involved in generation, and is actually only half a model, since it says very little about processing strategy. Our idea was to start with a deliberately simple model, with the idea that if the approach were successful we would subsequently refine it. Thus our architecture consisted of seven functional modules, covering most of the main linguistic aspects of generation, but not aiming to be exhaustive, and with no ordering constraints imposed *a priori*. The functional modules we chose, together with a few notes about each, are as follows:

Lexicalisation (Lex)
We take lexicalisation to mean the choice of content words which will appear in the final output text. We differentiate between "lexicalisation" and "lexical choice". The first we take to refer to conversion of something typically conceptual to lexical items, while the second we use in a narrower sense to mean deciding between lexical alternatives. More detailed discussion of lexicalisation in the given systems is in [Cah98].

Aggregation (Agg)
Any process of putting together more than one piece of information, which at some other level are separate. More detailed discussion of aggregation in the given systems is in [Rea].

Rhetorical structuring (Rhet)
Determining the rhetorical relations between pieces of information. It involves concepts such as "elaboration", "contrast" etc., and determines how the pieces of information should be related by the text structure.

Referring expressions (Ref)
Deciding how to refer to concepts or entities. This relates to lexicalisation, but it may also be done at a higher level, determining, for instance, that a pronoun is appropriate in a particular case, without determining which pronoun is used.

Ordering (Ord)
The linear ordering of pieces of information may be determined at a fairly high level, but equally it is possible for ordering of sentences in the surface text to be determined at quite a late stage in the generation process.

Segmentation (Seg)
Segmentation involves the dividing up of information/text into sentences and paragraphs, and so is the complement of aggregation. We do not restrict our definition of segmentation to segmentation into actual text units, but consider also the segmentation into "sentence-sized chunks" of information.

Coherence (Coh)
Coherence is a cover term for phenomena such as centering, salience and theme processing. Such processing is actually relatively rare in our test set systems, so it seemed better not to attempt to match a more detailed model to so little data.

5 Validating the functional model

Having proposed such a functional model, our next task was to validate it against the test set. Actually we had two goals in mind when undertaking this task. The first was basic validation of the model, in the sense that we validated the pipeline model – do the functional modules occur and if so do they align with the systems' own module boundaries? (NB: no ordering issues arise since the functional model is unordered.)

The second goal was to validate the model against the pipeline model. By combining the mappings from the functional model to individual systems and the mappings given in figures 2 and 3 above from systems to the pipeline model, we attempted to induce a mapping from the functional model to the pipeline model. This second step abstracts away from the details of each individual system, using our original conclusion of the overall validity of the pipeline model to 'smooth' the data in our test set. An ideal result, from the point of view of our initial hypothesis, would be to find that the functional model embeds neatly into the pipeline model, so providing the extra genericity and detail we are looking for in our reference architecture.

Our first result from this analysis was that the functional model was also, broadly speaking, valid. That is, all the test systems undertook all the tasks in the functional model, with a few odd exceptions, and one notable generalisation, namely that almost all the systems taking input from a user apparently did no coherence processing. Our second result, however, is less favourable: the test set systems do not induce a reasonable mapping between the functional model and the pipeline model.

Figures 5 and 6 show mappings induced between the models by the individual systems. The module names are abbreviated (`CD` = Content Determination; `SP` = Sentence Planning; `SR` = Surface Realiser; the functional ones are as given above) and multiple modules indicate a function is spread across multiple pipeline modules. An empty cell indicates the system does not appear to undertake the function at all, `user` that the user undertakes the function, and mappings we are unsure about are placed in

	Lex	Agg	Rhet	Ref	Ord	Seg	Coh
GIST	CD/SR	CD/SP	CD	SP/SR	CD/SP	SP	
Drafter	CD/SR	CD/SP	CD	SP/SR	CD/SP	SP	
Drafter2	CD/SR	SR	(user)	CD/SP		CD/SP/SR	
Pat Claim	CD/SR	SP	SP	CD	SP		
Joyce	SP	SP	CD	SP	CD	CD	
ModEx		SP	CD	(user)	CD	CD	
Exclass	CD/(SR)	SR	(user)	(user)	(user)	CD	
HealthDoc	SP	SP	CD/SP	(user)	SP	(user)	(CD)
GhostWriter	(SP)	SP	CD	(SP)		CD	CD/(SP)

Figure 5: Mapping from functional to pipeline model – user input systems

	Lex	Agg	Rhet	Ref	Ord	Seg	Coh
Ana	SR	SR	CD	SR	CD	SR	
PlanDoc	SP/SR	CD/(SP)	(CD)	(CD)	(SR)	CD	CD
FoG	SP/SR	SP	CD	SR	CD	CD	
Gossip	SP/SR		(CD)	SR	CD	(CD)	CD/SP/SR
LFS	SP/SR	(CD)	CD	CD	CD	CD	CD/SP/SR
CGS	SR	CD	CD	SP	CD	(CD)	CD/SP
PostGraphe	CD	(CD)	CD	SR	CD	CD	
AlethGen	SP/SR	SP	SP	SP	SP	(CD)	
Proverb	SP	SP	CD	SP	CD/SP	CD	

Figure 6: Mapping from functional to pipeline model – data input systems

parentheses.

These individual system mappings do not provide good evidence for an overall ordering of the functional tasks. However, close inspection of the tables does reveal some interesting points:

1. lexicalisation of user input systems displays a clear split between content determination and surface realisation. This distinction correlates with the two kinds of lexicalisation we noted above: lexicalisation from concepts and lexical choice. If we split the function into two, the picture becomes clearer;

2. most functions do have a reasonable 'average location' in the pipeline (see figure 7), but some of these seem to be on pipeline module boundaries, rather than neatly within a model;

3. several of the functions, such as referring expressions and aggregation seem quite ubiquitous in the pipeline model – they can happen almost anywhere.

6 Conclusions

Our attempt to validate the functional model against the pipeline model did not really succeed, but it did highlight a number of important points. First, we have demonstrated more clearly that the pipeline model is not detailed enough: like the test systems themselves, our functional model cuts across its module boundaries. However, the pipeline aspect of the architecture remains moderately secure, at least insofar as the functional modules group into broadly ordered chunks, as long as some leeway is allowed for the more ubiquitous functions. But it is also clear that the functional model, although useful for this exercise, is itself incomplete, and needs further refinement.

But the overall conclusion is, we feel, positive. A reference architecture along the lines of Reiter's proposal may still be possible, but needs to be based on a more detailed functional model and have a more flexible attitude to the pipeline constraint.

6.1 Towards a Reference Architecture

The above considerations lead us to think that the most productive approach is to divide the problem into two parts. On the one hand, we specify the abstract functionality of a generation system without specific reference to processing. Our approach to this is to develop a **data model,** that is, to define the functional modules entirely in terms of the datatypes

Function	Occurrences			Average Location
	CD	**SP**	**SR**	
Rhetorical structuring	15	4	0	CD
Segmentation	14	4	2	CD
Ordering	12	7	1	CD
Coherence	7	5	2	CD/SP
Lexicalisation	6	9	3	CD/SP
Aggregation	6	11	4	SP
Referring expressions	5	9	6	SP
Lexical choice	1	4	14	SR

Figure 7: Average pipeline locations for functional modules

they manipulate and the operations they can perform on them. On top of such a model, we can create more specific **process models**, in terms of constraints on the order and level of instantiation of different types of data in the data model. We might then produce a 'rational reconstruction' of the pipeline model, but other process models would also be possible.

The intention is that someone implementing an operation in the abstract data model can guarantee that it will be appropriate in all instantiations of the architecture. Researchers are free to find interesting ways of scheduling these operations in a complete system. The more specific pipeline architectures exist as guidance and to facilitate the sharing of datasets that span multiple levels.

For the data model, we are considering a three level representation of information[5]: *conceptual/ semantic, rhetorical* and *syntactic*. For each one we identify two stages of representation: *abstract,* when a structure at the level is first proposed, and *concrete,* when a structure at the level is complete. The data model specifies the types of these representations and the operations one can perform either to make a representation more concrete or to propose abstract structure at a different level on the basis of structure already produced. The goal of the system is to produce concrete representations at all the levels, but the data model only weakly constrains how this is achieved, allowing a wide range of processing strategies.

For example, the lexicalisation module might receive as input (at least) a set of conceptual structures and return a set of lexicalised semantic structures. The input/output requirements of operations impose some limits on how these can be ordered, but the intention is that the data model should reflect fairly uncontentious minimal constraints. It should be possible to derive most existing architectures by further specialising the data model and imposing extra ordering constraints. For instance, many systems do not clearly distinguish conceptual from semantic information. For these systems one simply assumes additionally that conceptual structures and semantic structures are one and the same thing.

Such a data model defines a network of possible operations that take us from an initial input to a final output. The more specific pipeline architecture is created by notionally drawing cross-sections through this network for the two intermediate representation levels of the pipeline. These cross-sections are defined in terms of additional constraints on the data representations of the model, constraints which take the form of specifying that particular representations must, must not or may be present. Any compatible pair of cross-section definitions constrains the general architecture to be a pipeline. Some very prescriptive definitions will correspond to very conservative pipelines that correspond closely to one or more existing systems. Some very loose definitions will allow more flexible working, at the expense of making it harder to implement conformant processing components.

Our goal of an adequate reference architecture will be achieved if we can produce a data model sufficiently generic and detailed to be generally useful, and show that it is possible to define on top of it in a principled fashion one or more pipelines that are relevant to systems of the sort we have studied.

[5]Our first attempt at formalisation of these ideas can be found in [Pro99]).

References

[ANL94] *Proceedings of the Fourth Conference on Applied Natural Language Processing*, Stuttgart, Germany, 1994.

[ANL97] *Proceedings of the Fifth Conference on Applied Natural Language Processing*, Washington, DC, 1997.

[BCG+90] Laurent Bourbeau, Denis Carcagno, E. Goldberg, Richard Kittredge, and Alain Polguère. Bilingual generation of weather forecasts in an operational environment. In *Proceedings of the 13th International Conference on Computational Linguistics (COLING-90)*, volume 1, pages 90–92, Helsinki, 1990.

[BT95] J.A. Bateman and E. Teich. Selective Information Presentation in an Integrated Publication System: an Application of Genre-driven Text Generation. *Information Processing and Management*, 31(5):753–767, 1995.

[Cah98] Lynne Cahill. Lexicalisation in applied NLG systems. Technical Report ITRI-99-04, Information Technology Research Institute (ITRI), University of Brighton, 1998. Available at `http://www.itri.brighton.ac.uk/projects/rags`.

[CD94] J. Coch and R. David. Representing Knowledge for Planning Multisentential Text. In ANLP'94 [ANL94], pages 203–204.

[Cer96] F. Cerbah. A Study of Some Lexical Differences between French and English Instructions in a Multilingual Generation Framework. In INLG'96 [INL96], pages 131–140.

[CI89] D. Carcagno and L. Iordanskaja. Content Determination and Text Structuring in GOSSIP. In *Extended Abstracts of the 2nd European Natural Language Generation Workshop (ENLG'89)*, pages 15–21, University of Edinburgh, 1989.

[CK94] D. Caldwell and T. Korelsky. Bilingual Generation of Job Descriptions from Quasi-Conceptual Forms. In ANLP'94 [ANL94], pages 1–6.

[Coc96] J. Coch. Evaluating and Comparing Three Text-Production Techniques. In *Proceedings of the 16th International Conference on Computational Linguistics (COLING'96)*, pages 249–254, Copenhagen, Denmark, 1996.

[Con96] GIST Consortium. GIST - Generating InStructional Text. Technical Report LRE Project 062-09 Final Report, Information Technology Research Institute (ITRI), University of Brighton, 1996.

[DHW95] C. DiMarco, G.and L. Wanner Hirst, and J. Wilkinson. Healthdoc: Customizing patient information and health education by medical condition and personal characteristics. In *First International Workshop on Artificial Intelligence in Patient Education*, Glasgow, August 1995.

[FL96] M. Fasciano and G. Lapalme. Postgraphe: a system for the generation of statistical graphics and text. In INLG'96 [INL96], pages 51–60.

[GDK94] E. Goldberg, N. Driedger, and R. Kittredge. Using natural-language processing to produce weather forcasts. *IEEE Expert*, 9(2):45–53, 1994.

[GL96] M. Gagnon and G. Lapalme. From conceptual time to linguistic time. *Computational Linguistics*, 22(1):91–127, 1996.

[HDHP97] G. Hirst, C. DiMarco, E. Hovy, and K. Parsons. Authoring and generating health-education documents that are tailored to the needs of the individual patient. In *Proceedings of the 6th International Conference on User Modeling*, Italy, June 1997.

[HF96] X. Huang and A. Fiedler. Paraphrasing and aggregating argumentative text using text structure. In INLG'96 [INL96], pages 21–30.

[HF97] X. Huang and A. Fiedler. Proof verbalization as an application of nlg. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI'97)*, Nagoya, Japan, 1997.

[IKLP92] Lidija Iordanskaja, Richard Kittredge, Benoit Lavoie, and Alain Polguère. Generation of extended bilingual statistical report. In *Proceedings of the 15th International Conference on Computation Linguistics (COLING'92)*, pages 1019–1023, Nante, 1992.

[IKP91] L. Iordanskaja, R. Kittredge, and A. Polguére. Lexical selection and paraphrase in a meaning-text generation model. In Cécile L. Paris, William R. Swartout, and William C. Mann, editors, *Natural Language Generation in Artificial Intelligence and Computational Linguistics*, pages 292–312. Kluwer Academic Publishers, Boston, 1991.

[INL94] *Proceedings of the Seventh International Workshop on Natural Language Generation*, Kennebunkport, Maine, 1994.

[INL96] *Proceedings of the Eighth International Workshop on Natural Language Generation*, Herstmonceux, Sussex, UK, 1996.

[KKR91] Richard Kittredge, Tanya Korelsky, and Owen Rambow. On the need for domain communication knowledge. *Computational Intelligence*, 7(4):305–314, November 1991.

[KMS+93] K. Kukich, K. McKeown, J. Shaw, J. Robin, J. Lim, N. Morgan, and J. Phillips. User-needs analysis and design methodology for an automated documentation generator. In *Proceedings of the 4th Bellcore/BCC Symposium on User-Centered Design*, Piscataway, NJ, 1993.

[KP91] R. Kittredge and A. Polguére. Generating extended bilingual texts from application knowledge bases. In *Proceedings on Fundamental Research for the Future Generation of Natural Language Processing*, pages 147–160, Kyoto, 1991.

[Kuk88] Karen Kukich. Fluency in natural language reports. In *Natural Language Generation Systems*, pages 280–311. Springer-Verlag, New York, NY, 1988.

[LR97] B. Lavoie and O. Rambow. A fast and portable realizer for text generation systems. In ANLP'97 [ANL97], pages 265–68.

[LRR97] B. Lavoie, O. Rambow, and E. Reiter. Customizable descriptions of object-oriented models. In ANLP'97 [ANL97], pages 253–256.

[MCM96] B.P. Marchant, F. Cerbah, and C. Mellish. The GhostWriter Project: a demonstration of the use of AI techniques in the production of technical publications. In *Proceedings of Expert Systems, 16th Conference of the British Computer Society*, pages 9–25, Cambridge, 1996.

[MKS94] K. McKeown, K. Kukich, and J. Shaw. Practical issues in automatic documentation generation. In ANLP'94 [ANL94], pages 7–14.

[MMCRng] V. O. Mittal, J. D. Moore, G. Carenini, and S. Roth. Describing complex charts in natural language: A caption generation system. *Computation Linguistics*, Forthcoming.

[MRM+95] V. O. Mittal, S. Roth, J. D. Moore, J. Mattis, and G. Carenini. Generating explanatory captions for information graphics. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI'95)*, pages 1276–1283, Montreal, Canada, August 1995.

[Pai98] Daniel Paiva. A survey of applied natural language generation systems. Technical Report 98-03, Information Technology Research Institute (ITRI), University of Brighton, 1998. Available at `http://www.itri.brighton.ac.uk/techreports`.

[PC96] R. Power and N. Cavallotto. Multilingual generation of administrative forms. In INLG'96 [INL96], pages 17–19.

[Pro99] The RAGS Project. Towards a Reference Architecture for Natural Language Generation Systems. Technical Report ITRI-99-14, Information Technology Research Institute (ITRI), University of Brighton, 1999. Available at `http://www.itri.brighton.ac.uk/projects/rags`.

[PS98] 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*, pages 1053–1059, Montreal, Canada, 1998.

[PVF+95] Cécile Paris, Keith Vander Linden, Markus Fischer, Anthony Hartley, Lyn Pemberton, Richard Power, and Donia Scott. A support tool for writing multilingual instructions. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, pages 1398–1404, Montreal, Canada, 1995.

[Ram] Owen Rambow. Domain communication knowledge. pages 87–94.

[Rea] Mike Reape. Just what is aggregation anyway?

[Rei94] Ehud Reiter. Has a consensus NL generation architecture appeared and is it psycholinguistically plausible? In INLG'94 [INL94], pages 163–170.

[SN96] S. Sheremetyeva and S. Nirenburg. Knowledge elicitation for authoring patent claims. *IEEE Computer*, pages 57–63, July 1996.

[SPE98] D.R. Scott, R. Power, and R Evans. Generation as a solution to its own problem. In *Proceedings of the Ninth International Workshop on Natural Language Generation*, pages 256–265, Niagara-on-the-Lake, Ontario, 1998.

[TB94] E. Teich and J.A. Bateman. Towards the application of text generation in an integrated publication system. In INLG'94 [INL94].