A Reference Architecture for Natural Language Generation Systems

Natural Language Engineering **12** (1): 1–34. © 2006 Cambridge University Press
doi:10.1017/S1351324906004104 Printed in the United Kingdom

*A Reference Architecture for Natural Language Generation Systems**

CHRIS MELLISH
Department of Computing Science, University of Aberdeen, Aberdeen, UK

DONIA SCOTT
Centre for Research in Computing, The Open University, Milton Keynes, UK

LYNNE CAHILL, DANIEL PAIVA
University of Sussex, UK

ROGER EVANS
School of Maths and Computing, University of Brighton, Brighton, UK

MIKE REAPE
School of Informatics, University of Edinburgh, Edinburgh, UK
e-mail: {rags@open.ac.uk}

(*Received 25 October 2004; revised 6 May 2004*)

Abstract

We present the RAGS (Reference Architecture for Generation Systems) framework: a specification of an abstract Natural Language Generation (NLG) system architecture to support sharing, re-use, comparison and evaluation of NLG technologies. We argue that the evidence from a survey of actual NLG systems calls for a different emphasis in a reference proposal from that seen in similar initiatives in information extraction and multimedia interfaces. We introduce the framework itself, in particular the two-level data model that allows us to support the complex data requirements of NLG systems in a flexible and coherent fashion, and describe our efforts to validate the framework through a range of implementations.

1 Motivation

This paper describes an attempt to specify a reference architecture for natural language generation systems. The field of Natural Language Generation (NLG) is characterised by a variety of theoretical approaches, which has the effect of fragmenting the community and reducing the bandwidth of communication. The technology is mature enough for commercial applications to be close, and yet it is relatively rare for NLG software and data to be reused between different researchers.

* This is a revised and updated version of the paper “A Reference Architecture for Generation Systems” which appeared (in error) in *Natural Language Engineering* 10(3/4) the Special Issue on Software Architectures for Language Engineering. This version should be cited in preference to the earlier one.

The RAGS (Reference Architecture for Generation Systems) project set out to exploit the implicit agreement that *does* exist in the field and focus discussion on the genuine points of theoretical disagreement with the aid of a "standard" vocabulary and reference point. Thus RAGS aimed to make it easier to:

- create reusable data resources (e.g., representative inputs to algorithms, corpora of generated texts together with their underlying representations);
- communicate data between NLG program modules written at different times and with different assumptions, and hence allow reuse of such modules;
- allow modules (or at least their inputs and outputs) to be defined in a relatively formal way, so that their scope and limitations may be better understood.

In this way, applications-oriented systems development should be facilitated, standard interfaces and datasets can emerge, ideas and software can be more easily reused, researchers can realistically specialise in particular areas and comparative evaluation of systems and components may become possible.

For natural language *understanding*, there has been considerable progress in the development of "architectures" and "infrastructures" to support application development, standardisation and evaluation. For instance, the GATE architecture (Cunningham, Wilks and Gaizauskas 1996; Bontcheva, Tablan, Maynard and Cunningham 2004) provides a suite of usable software modules, facilities to interface new modules whose input and output fit the underlying model and services to aid corpus management, statistical analysis and visualisation of results. RAGS attempts to start the process for natural language *generation* by defining an "architecture" for generation systems. We see below that there are interesting reasons why this has turned out to be rather different from GATE. However, we do not believe that the current state of the art puts anyone in the position to say what the "right" NLG architecture is: there is simply too little knowledge of the NLG process and too much variation in the goals of creating NLG systems. Premature standardisation would be damaging to the field and stifle theoretical diversity. And yet RAGS does define an architecture. We resolve this apparent contradiction in the following ways:

- We target specifically application-oriented end-to-end generation systems, primarily because systems like these must address all aspects of the generation process. Additionally, a study by Reiter (1994), discussed further below, suggested that there is some commonality to be found in such systems. We make no claims about other types of system, although it seems likely that many aspects of the discussion here would be relevant to a wider range of systems.
- We define the architecture in a particularly flexible way (as described in section 3). This means that people can "buy into" RAGS to varying extents. Naturally the prospect of sharing is increased as one buys more, but it is not an "all or nothing" decision (see section 5).
- We use the term *reference architecture* to refer to RAGS in order to emphasise the fact that it is not a proposed standard. The existence of a considered, well-specified reference point may be valuable not only to designers of new

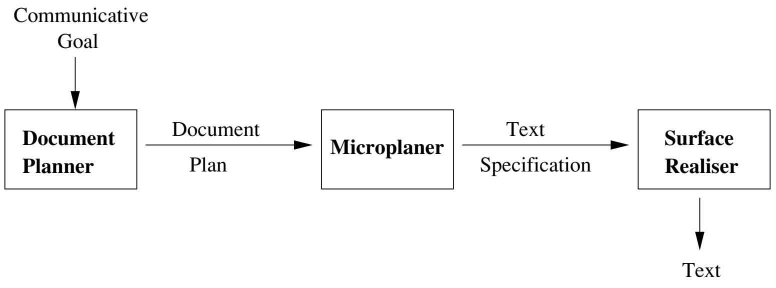


Fig. 1. Reiter and Dale's "consensus" architecture.

systems, but also as a point of comparison for existing systems, and even as a focus of discussion of its own deficiencies.

In the next section we discuss existing ideas about architectures for NLG and how our proposed framework compares with similar initiatives in other areas. We then proceed to discuss the key features of the framework itself, followed by some of the concrete implementations that have been developed within it. We summarise by discussing how one can use RAGS as it is now. Finally, we discuss the key contributions that this work has made, look at some of the areas it does not yet address and indicate possible future directions for the RAGS initiative. Further details of the RAGS specification, and the software that we distribute, can be obtained from our web site at `http://mcs.open.ac.uk/rags`.

2 Generic architectures for NLG

2.1 Reiter and Dale's "consensus" architecture

Many different overall architectures have been proposed for NLG systems (De Smedt, Horacek and Zock 1996), reflecting the range of different applications and the different motivations for constructing systems. Hope that there could be a single, well-specified architecture that would be adequate for many systems was raised when Reiter (1994) suggested that many end-to-end applications-oriented NLG systems followed the same architectural model: a simple three-stage pipeline which could therefore be viewed as a *de facto* standard or "consensus" architecture. Reiter and Dale (2000) updated and added further detail to this proposal in their NLG textbook. The overall structure of their model is shown in Figure 1.

The first major task for the RAGS project, undertaken in 1998, was a comprehensive survey of applied NLG systems in order to validate this "consensus" architecture proposal[1]. We identified 19 systems[2] which met the criteria of being end-to-end

[1] This survey drew on Reiter's original (Reiter 1994) formulation of the model. The later (Reiter and Dale 2000) formulation uses slightly different terminology, which we also use here, but for our purposes is otherwise not significantly different.

[2] The systems surveyed were: **Alethgen** (Coch and David 1994); **Ana** (Kukich 1988); **Caption Generation System** (Mittal, Moore, Carenini and Roth 1998); **Drafter** (Paris, Vander Linden, Fischer, Hartley, Pemberton, Power and Scott 1995); **Drafter2** (Scott, Power and Evans 1998); **Exclass** (Caldwell and Korelsky 1994); **FoG** (Goldberg, Driedger and Kittregde 1994); **GhostWriter** (Marchant, Cerbah and Mellish 1996); **GIST** (Power and Cavallotto

application-oriented NLG systems, and sought to correlate their published descriptions with the proposed consensus model. Initial conclusions were encouraging: although these systems had anything from two to seven named modules, virtually all the systems could be viewed as a pipeline architecture in which all of the modules corresponded to one, part of one or more than one of the three "consensus" modules (the principal exception to this model was Komet—for more details, see Cahill and Reape (1998)). In other words, no system cut across the notion of *Document Plan* and *Text specification* as the principal stepping stones of the generation process. This was apparent strong confirmation of Reiter's original insight.

However, the "consensus" model is actually more constraining than it may seem at first sight. The intermediate data structures (*Document Plan* and *Text Specification*) are not clearly formalised in Reiter and Dale (2000), but they are exemplified in some detail, and effectively prescribed to be complete and of specific types (although some variation is allowed in the contents of a *Text Specification*). This means that there is almost no flexibility over the assignment of lower level generation tasks (e.g., rhetorical structuring, lexical choice) to modules—if this is proposed as a "consensus" architecture, then one would expect to see all systems undertaking each of these tasks within the same module. The second step of the RAGS survey attempted to validate this aspect of the model also, a more difficult task because not all systems are described in sufficient detail to make clear judgements.

The low level generation tasks considered for this analysis were:

Lexicalisation: The choice of content words to appear in the final output text.
Aggregation: The combination of several structures (e.g., sentences) into a single, more complex, structure.
Rhetorical structuring: The determination of rhetorical relations and their scope.
Referring expression generation: The planning of the material to be incorporated within referring expressions (including the pronominalisation decision).
Ordering: The choice of linear ordering of the elements of the text.
Segmentation: The dividing up of information into sentences and paragraphs.
Centering/salience/theme: The tracking of salience-related properties of the text within a discourse.

This set does not constitute an exhaustive breakdown of NLG into components, but represents the tasks most commonly identified as independent functions within NLG systems. In order to validate the "consensus model" at this level, we assigned each of these tasks to the module of the consensus model in which it occurred. Figure 2 is a graphical representation of the results of this exercise[3]. For each

1996); **Gossip** (Kittredge and Polguére 1991); **HealthDoc** (Hirst, DiMarco, Hovy and Parsons 1997); **Joyce** (Rambow 1990); **Komet** (Teich and Bateman 1994); **LFS** (Iordanskaja, Kittredge, Lavoie and Polguère 1992); **ModelExplainer** (Lavoie, Rambow and Reiter 1996); **Patent Claim Expert** (Sheremetyeva, Nirenburg and Niren burg 1996); **PlanDoc** (McKeown, Kukich and Shaw 1994); **PostGraphe** (Fasciano and Lapalme 1996); **Proverb** (Huang 1994). Full details and discussion of the survey and results obtained can be found in Paiva (1998) and Cahill and Reape (1999).

[3] This figure is a re-presentation of the two tables in Cahill and Reape (1998). Where a task was performed by user intervention, not performed at all or where its placement was unclear, we have omitted it; however we have included placements marked as uncertain in the original tables.

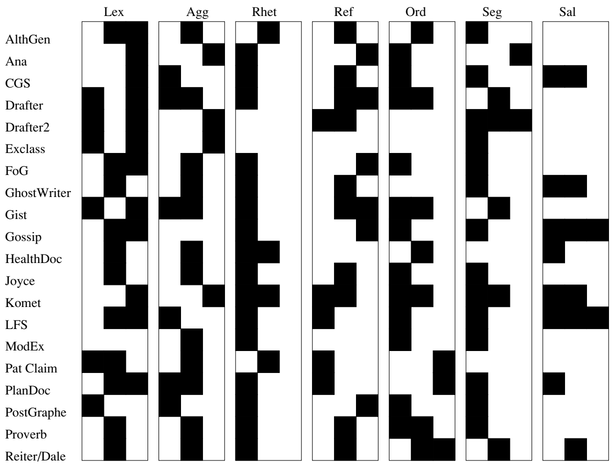


Fig. 2. Timing of tasks in systems surveyed.

system and task, there are three positions that can be shaded, indicating that the task is (partly or completely) performed within the corresponding module (with the ordering *Document Planner*, *Microplanner* and *Surface Realiser*). For instance, in GhostWriter, aggregation is performed in the *Microplanner* and salience is considered in both the *Document Planner* and the *Microplanner*. For reference, we have included the Reiter and Dale model as the last line of the table, even although it was not specified in detail at the time of the survey.

For the model to be validated at this level of granularity, we would expect most of the systems to follow the same pattern as this last line. It is quite clear from the diagram that they do not. There is good agreement that *Rhetorical Structuring* occurs in the *Document Planner*, but almost as strong **disagreement** that *Segmentation* occurs in the *Microplanner*—most systems do it in the *Document Planner*. Most tasks seem capable of appearing in *any* module, or across two modules, and *Lexicalisation* seems to occur quite often in two *disjoint* phases.

Although such an exercise is always only an approximation, the message of Figure 2 is not in the particular details but in the significant amount of diversity that is indicated. This diversity suggests that while the Reiter and Dale model may have captured some high level intuitions about the organisation of NLG systems, it is too constrained to act as the basis for a formally defined functional architecture reflecting the full range of applied NLG systems.

In summary, our conclusions about the validity of the consensus model were:

1. Many of the systems surveyed claimed or appeared to follow the three-stage model, however, it was not possible to assign detailed functional descriptions to the three modules that were compatible with all, or even most, systems.
2. Many of the systems had functional submodules in common (such as ‘referring expression generation’), but the order of execution of those submodules, and their assignment to the three stages did not form a single overall consistent pattern.
3. There was no simple definition of the data interfaces between the proposed consensus modules, or even the lower level functional modules, in the systems

surveyed—most systems manipulated data at several linguistic levels in fairly complicated ways, and 'snapshots' of the data at the putative interfaces did not reveal any strong patterns of what was and what was not instantiated in different systems.

Thus, while their proposed "consensus" model represents an important observation about the field, Reiter and Dale's particular position about the definition and timing of modules, as well as their assumptions about how the types of data come together in the interfaces, although examples of good NLG system design, are too restrictive for the general case. To be useful, the RAGS architecture would need to take a more flexible position on these issues.

2.2 Requirements for the RAGS *architecture*

On reflection, perhaps the conclusions of our survey were not so surprising. Fundamentally, the NLG task can be thought of as a simultaneous constraint satisfaction problem involving knowledge at several linguistic levels (De Smedt *et al.* 1996). Researchers proposing novel architectures and search methods for NLG have maintained that efficient generation of optimal texts cannot be achieved by making choices in a predetermined order (for instance, Danlos (1984), Wanner and Hovy (1996), Mellish, Knott, Oberlander and O'Donnell (1998)). This argues against there being an optimal pipeline or layered approach as found in much current NL understanding work. Greater complexity in data interaction and more variation in order of processing might be expected, making the task of developing a reference architecture more challenging.

Nevertheless the survey did reveal enough commonality to justify a search for common principles, as long as the requirement for a precise architecture specification was replaced by the pursuit of a more general framework for developing NLG systems. Such a framework could still provide a basis for interaction between researchers who agree on basic assumptions about their system architectures and data structures—it would not force a particular view (as a full reference architecture would have), but it would provide a common language for expressing and exploring an agreed perspective. The resulting RAGS framework relaxes the 'architectural' requirement to a point where it is sufficiently inclusive of actual systems to be relevant, yet still sufficiently restrictive to be useful. We achieved this by characterising at a quite abstract level the data types, functional modules and protocols for manipulating and communicating data that most modular NLG systems seem to embody. Thus the main components of the RAGS proposal are:

- a high-level specification of the key (linguistic) data types that NLG systems manipulate internally. This uses abstract type definitions to give a formal characterisation independent of any particular implementation strategy;
- a low-level reference implementation specifying the details of a data model flexible enough to support NLG systems. This implementation (the "Objects and Arrows Model") formally characterises the set of legal data representations as a set of typed directed graphs;

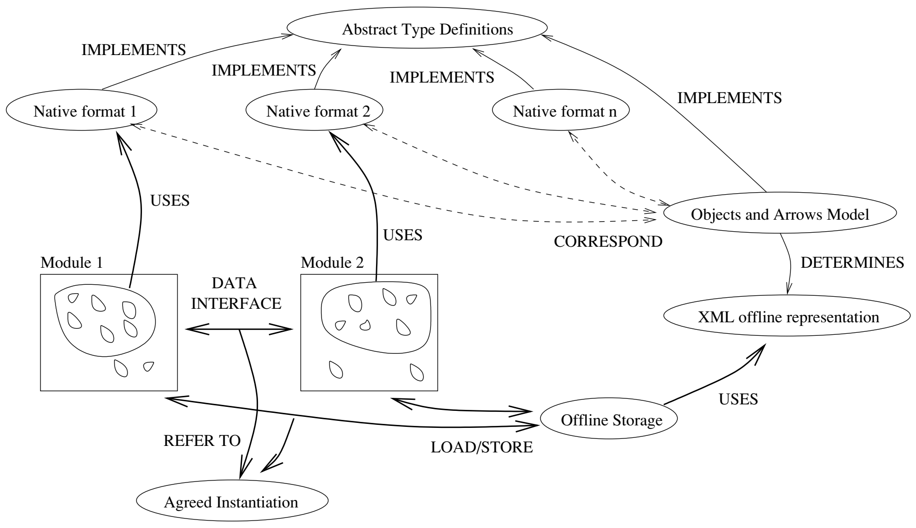


Fig. 3. Structure of the RAGS model.

- a precise XML specification for the data types, providing a standard ‘off-line’ representation for storage and communication of data between components;
- a generic view of how processing modules can interact and combine to make a complete NLG system, using data formats “native” to their particular programming languages which are faithful to the high- and low-level models and exploiting agreed instantiations of the high-level data types;
- several sample implementations to show how the development of a range of concrete architectures can be achieved.

Figure 3 summarises the structure of the RAGS model and how it is intended to be used. Different processing modules manipulate RAGS data using formats native to their particular programming languages (they can, of course, manipulate other kinds of data, but RAGS says nothing about how that might be communicated between them). These formats provide implementations of the abstract type definitions and are also in correspondance with the “objects and arrows” model, through which they inherit a standard XML serialisation. Two communicating modules need to agree on an instantiation of the abstract data types and can then define their interfaces and exchange data either in an agreed native format or via the XML offline representation.

The decoupling of the abstract from the more concrete aspects of the reference architecture results in a ‘framework’ for developing NLG systems which allows many possible concrete architectures—to construct an actual NLG architecture, additional decisions about process and data flow have to be taken. This is one of the ways that the RAGS architecture is unusually flexible.

2.3 Comparison with other generic architectures

The resulting model is similar in some ways to the GATE architecture for information extraction systems (Cunningham *et al.* 1996; Bontcheva *et al.* 2004). Both provide assistance in building an application out of modular components provided from elsewhere, and are quite uncommitted to actual processing strategies employed (although both include suggested functional submodules a system might typically include). There are two key areas of difference, however:

1. In GATE, data communication is achieved through incremental annotation of the texts being analysed, so the implicit data types are strongly grounded in textual elements; since of course NLG does not start off with a text, RAGS needs a more abstract NLG-oriented repertoire of data types and representations, with less dependence on a single common reference point. RAGS thus entails taking a stronger theoretical stance, attempting to make a substantive statement but not alienate alternative approaches.
2. GATE is a concrete generic implementation platform, with specific libraries and interfaces for application development, concentrating on high-quality support for low-level operations. RAGS is a framework for specifying implementations (plus some sample implementations), more committed functionally, but less committed implementationally.

Both RAGS and GATE support the building of modular systems and the comparative evaluation of system components, but their scope is different. RAGS supports the *specification* of a component task and the standardised representation of desired inputs and outputs. GATE does this too (with a simpler model of data), but also provides much more direct implementation support.

GATE has been very successful. In a recent user survey[4], around half of those who replied were using the system actively in research, applications development or teaching, suggesting a continuing substantial and healthy user base. This success encourages us to hope that something similar will be possible for NLG. However, the success of GATE relies on an existing underlying data model, the TIPSTER architecture. Such a model does not yet exist for NLG, so a key contribution of RAGS is the attempt to specify one.

A second useful point of comparison is the 'standard reference model' for intelligent multimedia presentation systems (IMMPS) described by Bordegoni and his colleagues (Bordegoni *et al.* 1997). This specifies a high level but quite concrete architecture for an IMMPS, drawing on a widely agreed view of what kinds of components an IMMPS should have and how they interact. This results in a quite prescriptive proposal, reflecting the high degree of agreement over principles of good high level design in this area. In contrast, RAGS is not prescriptive at the architectural level, confining its attention to supporting flexible interaction among modules, rather than specifying which modules interact and how. The model proposed by Bordegoni and his colleagues is now a well-established reference for describing and implementing multimedia presentation systems (e.g., see McRoy and Ali (1999), Rutledge, Hardman, van Ossenbruggen and Bulterman (1999),

[4] Approximately 5000 people who have downloaded the current version of the system, response rate 9%, conducted in Autumn 2003—Hamish Cunningham, personal communication.

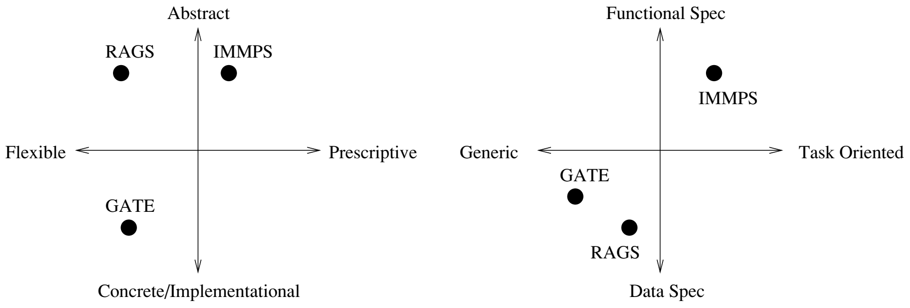


Fig. 4. Informal classification of reference architectures.

van Ossenbruggen, Geurts, Cornelissen, Hardman and Rutledge (2001) and Loeber, Aroyo and Hardman (2002)).

We conclude that all these reference proposals, while similar in intent and structure, have subtly different balances of emphasis. We can distinguish architectures according to a number of dimensions. An architecture can be *abstract* (supporting design and specification) or *concrete* (supporting actual implementation mechanisms). It can be *prescriptive* (narrowly specifying what is possible) or *flexible* (allowing a range of types of implementation). It can be oriented towards *functional specification* (specifying the modules and their interactions) or *data specification* (specifying the data exchanged between or held by modules). Finally, it can be *task-oriented* (focused on a narrow range of systems with a well-defined task) or *generic* (open-ended, in terms of what the systems are for). Figure 4 shows informally how RAGS, GATE and the IMMPS model relate to these dimensions.

3 The RAGS framework

The core of the RAGS framework is an account of the data that NLG systems manipulate: the RAGS 'data model'. This model is a two-level description of NLG data structures: the high level model divides NLG data into six basic types with different linguistic roles, and specifies how they relate to each other; the low level model specifies formally what data states can be communicated between NLG modules in the context of the evolving specification of a text. The low level model also provides the basis for uniform off-line representation and communication of data between modules.

A user of the RAGS framework can "buy into" the framework (and existing work using it) to varying degrees. They can, for instance, choose whether or not to:

- represent (some of) their data in ways compatible with the RAGS data model;
- use particular instantiations of the abstract types that are already in use;
- implement modules with functionality as specified in the set of example RAGS modules, or compatible with other existing work;
- organise their modules in standard ways that emerge from previous work, possibly using RAGS application components.

The basic RAGS framework focuses primarily on the first of these, providing a firm basis for the establishment of agreed approaches to the other aspects as the framework gets used.

3.1 High level data definitions

The RAGS framework supports six basic data types: Conceptual, Rhetorical, Document, Semantic, Syntactic and Quote. Of these, Syntactic and Quote are dependent on the target language and Conceptual will often be independent of it. For the other three, we take no position on language-dependence, although we suspect that only (parts of) the Semantic level could be language-independent.

NLG systems are built assuming many different theories of syntax, semantics, discourse, etc. It would be impossible for RAGS to produce a generally-acceptable complete definition of representations at any of these levels as there is so much theoretical diversity. Instead, RAGS defines the different levels of representation primarily in terms of *abstract type definitions* which specify the components out of which these representations are expected to be made. Here we use standard notation from set theory to characterise the essential structure of the data, independent of any particular implementation (a common approach in theoretical Computer Science, although not used extensively in Computational Linguistics). The notation used in our definitions is fully explained in the RAGS Reference Manual (Cahill, Evans, Mellish, Paiva, Reape and Scott 2001b), and briefly introduced here. Thus, for instance:

$$\begin{aligned} RhetRep &= RhetRel \times RhetRepSeq \\ RhetRepSeq &= (RhetLeaf \cup RhetRep)^{+} \end{aligned}$$

(from section 3.1.2, below) defines a Rhetorical Representation (*RhetRep*) as consisting of a *RhetRel* (rhetorical relation) and a *RhetRepSeq* (sequence of *RhetRep*s) for the children. The latter is itself defined in terms of a sequence of one or more elements, each of which is a *RhetLeaf* or a *RhetRep*.

The abstract type definitions "bottom out" in *primitive* types, which are indicated as follows:

$$RhetRel \in Primitives$$

RAGS has nothing further to say about the type *RhetRel*, which is the set of rhetorical relations. Researchers are invited to instantiate this set as they see fit according to any theoretical position they wish to adopt. That is, RAGS does not specify the *subtypes* of primitive types, apart from acknowledging that they may exist. Different implementations may make different assumptions about the subtypes of primitive types, although most effective reusability of code or data will be possible between researchers that do agree.

All the definitions of RAGS datatypes are thus *parameterised* via these primitive types—they define the overall format of a set of structures but do not stipulate the exact set of values that can appear at the leaves of these structures. Users of RAGS datasets or modules need to indicate how the primitive types are instantiated as well as how they are using the abstract types.

In addition to the abstract type definitions, we also give informal examples of the representations in notations which suggest actual implementation possibilities[5].

It is important to realise that in a sense there is nothing very novel in the following proposals. They are intentionally anchored in existing practice, and based on detailed study of existing theories. The significance of the RAGS model comes from the explicit articulation of the distinct levels of representation and their form. We do not believe that NLG systems are always explicit about the nature of their representations (e.g., whether something is semantic or syntactic), and problems arise if this is done badly (Mellish 2000).

3.1.1 Conceptual representations

Conceptual representations encapsulate references to data 'outside' the NLG system—typically information provided as the generator's input. These are not semantic representations: they may have no simple relation to linguistic decisions and they are assumed not to be created or altered by the NLG system.

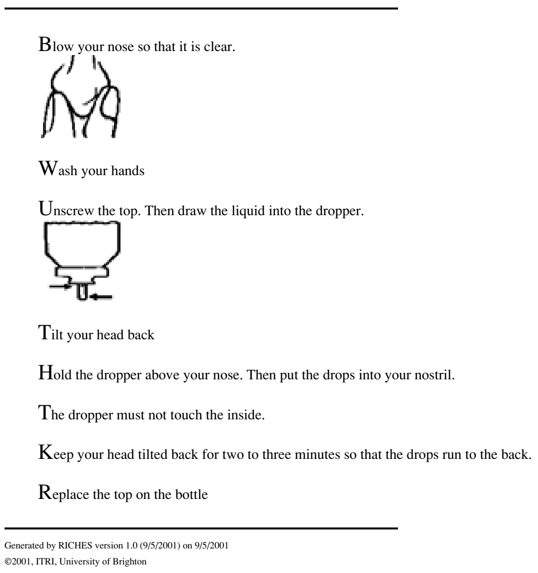
For conceptual representations, there is a single primitive type *KBId* ("knowledge base identifier"). RAGS specifies how an NLG system can interrogate such data to find out about the outside world, using functions such as:

$$\begin{aligned}
kb\_subsumed\_by &: \quad KBId \times SemPred \rightarrow Bool \cup \{\texttt{Unknown}\} \\
kb\_role\_values &: \quad SemRole \times KBId \rightarrow 2^{KBId} \cup \{\texttt{Unknown}\} \\
kb\_types &: \quad KBId \rightarrow 2^{SemPred} \\
kb\_roles\_with\_values &: \quad KBId \rightarrow 2^{SemRole}
\end{aligned}$$

where *SemPred* and *SemRole* are the predicates and roles used in semantic representations. The functions shown here are for telling whether a *KBId* is subsumed by a given semantic type, what the values of a given semantic role for a *KBId* are, what semantic types a *KBId* has, and what semantic roles it has values for.

Although conceptual entities may have no simple linguistic interpretation, they nevertheless have to be related to linguistic concepts in order that an NLG system can handle them. This is why the above functions are expressed in terms of *semantic* predicates, roles, etc., rather than a separate conceptual vocabulary. The RAGS API for *KBId*s is very close to the abstract interface between the PENMAN system (Mann 1983) and its knowledge base, and it could be implemented via an Upper Model (Bateman 1990). The specification of this API is also compatible with the view that an NLG system's interaction with an external knowledge source should be through a controlled interface that provides some protection from deficiencies and irregularities in the underlying data (Reiter and Robertson 1999). Reiter and Dale (2000) discuss the elements of a domain model in terms of entities, attributes, relationships and

[5] RAGS endorses no specific notations apart from the XML interchange format described in section 3.4.



Blow your nose so that it is clear.

Wash your hands

Unscrew the top. Then draw the liquid into the dropper.

Tilt your head back

Hold the dropper above your nose. Then put the drops into your nostril.

The dropper must not touch the inside.

Keep your head tilted back for two to three minutes so that the drops run to the back.

Replace the top on the bottle

Generated by RICHES version 1.0 (9/5/2001) on 9/5/2001
©2001, ITRI, University of Brighton

Fig. 5. Sample instructions from the RICHES system (Cahill, Carroll, Evans, Paiva, Power, Scott and van Deemter 2001a).

classes (in a hierarchical taxonomy). The above API permits easy querying of such a domain (assuming that relationships are reified). However, Reiter and Dale differ from RAGS in asserting that the predicates, roles etc. are actually part of the domain, rather than part of a language-oriented view of the domain.

In general, principles of *content determination* in NLG may be very domain-dependent, and so a framework like RAGS can say little about them. Some aspects of content determination are, however, involved in the implementation of *KBId*s.

3.1.2 Rhetorical representations

Rhetorical representations define how propositions (in RAGS, *SemRep*s—see section 3.1.4) within a text are related. For example, the first sentence of the text in Figure 5, "Blow your nose, so that it is clear", generated by the RICHES system (Cahill *et al.* 2001a), can be considered to consist of two propositions: *blow your nose* and *your nose is clear*, connected by a relation like `MOTIVATION`.

Following Scott and Souza (1990) and Power, Scott and Bouayad-Agha (2003), rhetorical representations differ from document representations (section 3.1.3) in that they define the underlying rhetorical relations between parts of a discourse, without specifying anything about how the relations may be realised. Whereas document representations define features such as linear ordering and page layout, rhetorical representations define what may be termed the *pragmatic* relations.

The type definitions for rhetorical representations are as follows:

$$\begin{aligned} RhetRep &= RhetRel \times RhetRepSeq \\ RhetRepSeq &= (RhetLeaf \cup RhetRep)^+ \\ RhetRel, RhetLeaf &\in Primitives \end{aligned}$$

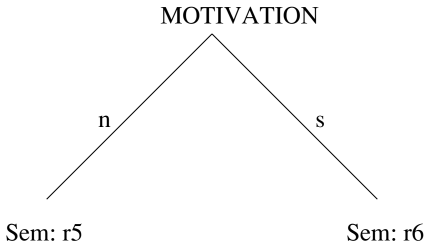


Fig. 6. Example rhetorical representation (informal).

Thus a rhetorical representation (*RhetRep*) is a tree whose internal nodes are labelled with relations (*RhetRel*), and whose leaves are atomic rhetorical nodes (*RhetLeaf*, which, as we will see in section 3.2, can, for instance, be thought of as pointers to conceptual or semantic content).

An example of a simple rhetorical representation is shown informally in Figure 6. Here there is a single *RhetRep* containing a sequence of two *RhetLeaf* s (associated with semantic content, which is not strictly part of the *RhetRep* itself). The *RhetRel* for the *RhetRep* has the subtype `MOTIVATION`, a theory-specific relation.

The significance of the positions of the subtrees is not specified for the RAGS representation in general, because it is theory-specific. Theory-specific definitions of relations therefore specify the role of their daughters. For example, an implementation of Rhetorical Structure Theory (Mann and Thompson 1988) would need to specify that, for example, the first daughter of the `MOTIVATION` relation is the "nucleus" and the second the "satellite"; were the relation instead `CONTRAST`, it would specify that the second was another "nucleus". Similarly, an implementation based on the discourse theory of Moser and Moore (1996) would need to specify that, for example, the first daughter of `MOTIVATION` is the "core" and the second the "contributor", but that the daughters of a `CAUSE` relation carried no such designation.

Following the prevailing computational theories of discourse structure (e.g., Grosz and Sidner (1986), Mann and Thompson (1988) and Hobbs (1985)) the RAGS architecture assumes that the rhetorical structure of a discourse is hierarchical. Our data-specification presupposes the use of trees with relations at the nodes and content at the leaves. The trees are defined in a very general way, with the subtypes of *RhetRel* open to definition in potentially different ways.

Reiter and Dale's *document plans* are equivalent to RAGS *RhetRep*s, except that the former also convey information about surface order. In RAGS, surface ordering is specified in the document structure, following the tenets of Document Structure Theory (Power *et al.*, 2003).

3.1.3 Document representations

Document representations encode information about the graphical presentation of a document, such as its organisation into abstract textual units (paragraph, orthographic sentence, etc.), their relative positions, and layout (indentation, bullet lists etc.). It is likely that a notion of document representation also makes sense for spoken language, although this remains to be established.

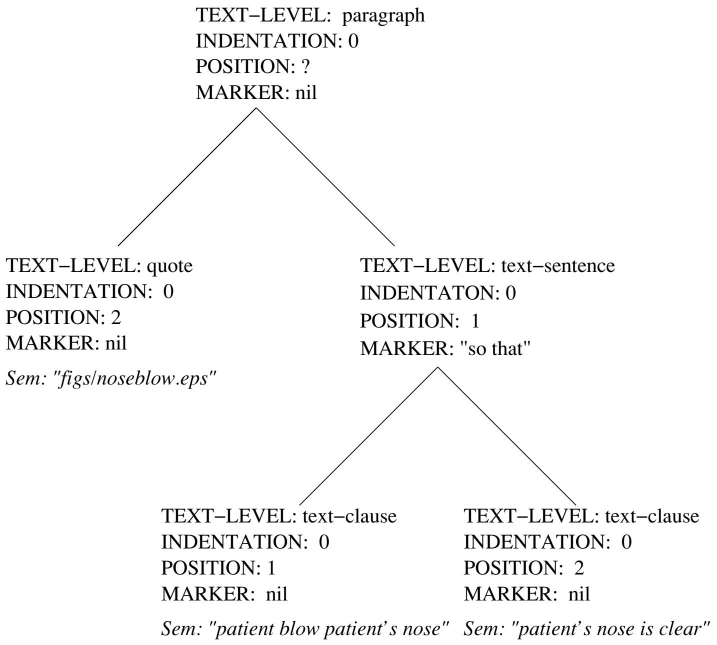


Fig. 7. Example document representation (informal).

The type definitions for document representations are as follows:

$$\begin{aligned} DocRep &= DocAttr \times DocRepSeq \\ DocRepSeq &= (DocLeaf \cup DocRep)^{++} \\ DocLeaf &= DocAttr^{1} \\ DocAttr &= (DocFeat \rightarrow DocAtom) \\ DocFeat, DocAtom, DocLeaf &\in Primitives \end{aligned}$$

A document representation (*DocRep*) can thus be thought of as a tree with no unary branches and feature structures at its nodes (both internal and leaf nodes). A feature structure (*DocAttr*) consists of features with atomic values. There are no constraints on the features or the sets of values which may be defined. In a way that parallels rhetorical and semantic representations, document leaf representations are often associated with syntactic representations (section 3.2).

An example simple document representation is shown in Figure 7, which represents the first paragraph of Figure 5. Here there is a single *DocRep* containing a sequence of two structures at the top level, representing the text and picture; the text is further decomposed into another sequence (mirroring the example *RhetRep* in Figure 6). Each of these has an associated *DocAttr* which provides a mapping from *DocFeat*s to *DocAtom*s. The *DocFeat*s used are `text-level`, `indentation`, `position` and `marker`, with appropriate values. Different theories might use a different repertoire here.

Following Power *et al.* (2003), RAGS distinguishes between *abstract* and *concrete* document structure. Abstract structural properties (e.g., paragraph, text-sentence, text-clause) may be realised concretely in different ways according to different conventions. For example, a paragraph may be expressed by a number of devices, including a new line with a tab, or two new lines (or some other vertical space) with no tab. Similarly, text-clauses may be realised with a semi-colon, a full-stop (if it were text-sentence final), or with no distinguishing punctuation (if it were an item in a bulleted list). None of these choices affect wording, as could the choice of a different abstract document category. We thus consider the specification of paragraph, text-sentence and text-clause to be part of abstract *DocRep* and the specification of the style of paragraph break to be part of concrete *DocRep*. We do not cover the concrete level of representation in RAGS.

There is relatively little previous work that addresses these issues in the context of language generation, although Reiter and Dale's Text Specifications can be loosely thought of as document structures with various kinds of leaves. Our proposals for this level are largely based on our own experiences of representing document structure in the generation process. The theoretical foundations of this level of representation are described in detail in Power *et al.* (2003).

3.1.4 Semantic representations

Semantic representations specify information about the meaning of individual propositions. By "semantic information" we mean propositional content, or "ideational meaning" in the terminology of systemic grammar (e.g., see Halliday (1994)).

Semantic representations are representations of linguistic meaning structured in terms of *lexical semantic predicates* and semantic (i.e., thematic) *roles*. They contrast with syntactic representations, which are structured in terms of lexical *heads* and *grammatical functions* or *relations*. Lexical semantic predicates are intended to be sufficiently abstracted from surface forms that the mapping from configurations of them to actual lexical items of a language could be non-trivial. Indeed, it might be possible to create semantic representations that are language-independent, although we do not take a position on this.

The type definitions for semantic representations are as follows:

$$\begin{aligned}
SemRep &= DR \times SemType \times SemAttr \\
ScopedSemRep &= DR \times SemType \times SemAttr \times Scoping \\
SemType &= 2^{SemPred} - \phi \\
SemAttr &= SemRole \rightarrow \\
&\qquad (SemRep \cup ScopedSemRep \cup DR \cup SemConstant) \\
Scoping &= ScopeConstr^{*} \\
ScopeConstr &= ScopeRel \times DR \times DR \\
DR, SemConstant &\in Primitives \\
SemPred, ScopeRel &\in Primitives \\
SemRole &\in Primitives
\end{aligned}$$

```
(h1 / sneeze
:actor (h2 / person :quant (h3 / every))
:time past
:polarity (h4 / negative)
)
```

Fig. 8. Example semantic representation (informal).

A semantic representation (*SemRep*) thus consists of a DR (discourse referent), a non-empty set of predicates (interpreted as a conjunction) and values for semantic roles (*SemRole*)[6]. Semantic roles can be filled by discourse referents, constants or other semantic representations (scoped or unscoped). Constants (*SemConstant*) logically correspond to names of individuals whereas *DR*s are logical variables. That leaves only *ScopedSemRep*s to explain and describe. (Complex) semantic representations (at every level) can include scoping information or not. The extra *Scoping* component of a scoped semantic representation is intended to express constraints on how subparts of that representation relate to others in terms of scope. Quantifier scope can be arbitrarily specified or left unspecified altogether in which case the fourth component of the *ScopedSemRep* is empty.

An example simple semantic representation of '*everybody didn't sneeze*' is shown informally in an SPL-like notation in Figure 8. The notation here shows a *SemRep* as a list consisting of the *DR*, a fixed separator "/", the *SemPred*s and then alternating *SemRoles* and their values. Here the main *SemRep* has a *DR* `"h1"` and a single predicate (RAGS allows a set) *sneeze*. Its *SemAttr* is a mapping from two semantic roles (*SemRole*) to values. The value associated with the *SemRole actor* has its own non-trivial *SemAttr* with a value for the *SemRole quant*. In the example, the relative scope of the `every` and `negative` is left unspecified. One possible *ScopedSemRep* would combine the above with something like:

```
{h3 < h4}
```

indicating that the `negative` is intended to outscope the `every` (it is not the case that every person sneezed). Here the *ScopeRel* `<` indicates inclusion of scope. The representation of the quantifiers as complete *SemRep*s above (unlike the `time` information) enables such constraints to be stated.

Approaches to "flat" semantic representations for NLG (e.g., Kay (1996), Copestake, Flickinger, Sag and Pollard (1999) and Stone, Doran, Webber, Bleam and Palmer (2001)) would probably choose to represent the above content as a *set* of *SemRep*s. An MRS[7] representation of the above (without scope being specified) might, for instance look like:

$$< h0, \{h4 : not(X), h3 : every(x, h2, Y), h2 : person(x),\\ h1 : sneeze(x), h5 : past(h1)\}, \{\} >$$

[6] We intend "discourse referent" here in the sense of Discourse Representation Theory (Kamp and Reyle 1993), that is, as a variable which denotes an individual (entity) or a set, group or collection of individuals (entities). Such variables are used in a range of semantic formalisms, e.g., SPL, the Sentence Planning Language (Kasper 1989). Theories that do not use such variables can simply omit the *DR*s in RAGS *SemRep*s.

[7] Minimal Recursion Semantics, as introduced in Copestake *et al.* (1999).

In this case, each of the "elementary predications" can be thought of as a *SemRep*, with a *DR* (the "handle", indicated by the name before the ":"), a predicate and semantic roles given by *SemRoles* 1, 2 and 3. Here we have indicated the handles whose relation to others is ambiguous by `X` and `Y`. To indicate the scoping relations precisely or partially, these ambiguous handles could be replaced by specific other handles (e.g., replacing `X` and `Y` by `h3` and `h5` would give the same scoped reading as above). Alternatively, MRS allows explicit statement of constraints on scope relations (e.g., $h0 =_q h5$) in a way that corresponds directly to the RAGS approach.

The subtypes used for discourse referents, predicates and roles in such examples are, of course, theory-dependent. This means (as with any other uses of RAGS primitives) that two modules exchanging *SemRep*s need to have an agreement on (the relevant parts of) their respective semantic vocabularies, or at least a way of translating between them.

Semantic representations used in NLG tend to use complex term representations for proposition arguments instead of having explicit quantifiers (Creaney 1996), (akin to the "quasi logical forms" of Alshawi (Alshawi 1982) or the *wffs* discussed by Hobbs and Shieber (1987)). This permits, for instance, referring expression generation to produce a single complex formula suitable for substituting into a proposition. On the other hand, it is a weakness of such representations that quantifier scope cannot be indicated or respected. The RAGS semantic representations are based primarily on SPL, ESPL (Vander Linden 1994) and Zeevat's Indexed Language (InL) (Zeevat 1991). However, we allow quantifier scope (optionally) to be specified using ideas from Underspecification Discourse Representation Theory (UDRT) (Reyle 1995; Frank and Reyle 1995)[8].

Reiter and Dale use what RAGS would call *SemRep*s in two places. Firstly, the leaves of their Document Plans ("messages") are flat predicate-argument structures with links to conceptual objects (see section 3.3.2). However, in these, Reiter and Dale think of the predicates, roles etc. as elements of the domain, rather than true semantic representations. Secondly, the leaves of their Text Specifications can be skeletal propositions or meaning specifications, which are flat (incomplete) and nested (including representations for referring expressions) *SemRep*s respectively. If one wished to distinguish between two such uses in RAGS, this could be done by partitioning the sets of *SemPred*s etc. into those that are specific to the domain and those that are more general. One might, for instance, claim that the former are language independent but not the latter.

3.1.5 Syntactic representations

Syntactic representations are mainly for specifying the input to realisers which are based on some notion of *abstract* syntactic structure which does not encode surface constituency or word order.

[8] UDRT allows an unscoped representation to consist of a set of components, together with a partial order that determines how components may nest inside one another.

```
blow [class:verb mode:imperative]
(I you [class:pronoun]
 II nose [class:common-noun number:sing]
  (ATTR you [class:pronoun case:genitive]))
```

Fig. 9. Example syntactic representation (informal).

The type definitions for syntactic representations are as follows:

$$\begin{aligned} SynRep &= FVM \times (SynRep \cup Nil) \times SynArg \times Adj \\ FVM &= SynFeat \rightarrow (SynAtom \cup FVM) \\ SynArg &= SynFun \rightarrow SynRep \\ Adj &= SynRep^{*} \\ Nil, SynFun, SynFeat, SynAtom &\in Primitives \end{aligned}$$

A (non-atomic) *SynRep* is thus a tuple $\langle \text{Head}, \text{Spec}, \text{Args}, \text{Adjs} \rangle$ where Head is a feature-value matrix, Spec is the *SynRep* of the specifier of Head, Args is a function from grammatical functions to argument *SynReps* and Adjs is a tuple of adjunct *SynRep*s. It is a tuple and not a set to accomodate those theories in which order of adjunction or modification is significant[9].

The atomic unit of abstract syntactic representations is simply a *feature-value matrix*, *FVM*, of the usual kind in syntactic models (Cf. Shieber (Shieber 1986))[10]. We envisage that such FVMs might contain four main types of information: (a) morpholexical class (part of speech and other "syntactic" information usually described as *head features*, see Gazdar, Klein, Pullum and and Sag (1985), and Pollard and Sag (1987, 1994)), (b) morphological markup suitable for input to inflectional morphology (e.g., "third person", "plural"), (c) root, stem and orthography and (d) other information for controlling realisation (e.g., "active", "passive"). Although these FVMs basically fulfil the role of *lexemes* in Meaning Text Theory (MTT) (Mel'čuk 1988), we nonetheless recognise that practical systems will include extra realisation control information in these FVMs to choose amongst multiple possible realisations.

An example of a simple syntactic representation (for "blow your nose") is shown informally in Figure 9, using a syntax similar to that used for the DSyntS input to REALPRO (Lavoie and Rambow 1997). Here a *SynRep* is indicated by a sequence of two or three items. The first two items indicate the *FVM* by giving the value of a special *SynFeat* (perhaps named `root` or `lex`) followed by the other features and values. The final (optional) element of the sequence gives the values of the syntactic functions, adjuncts and specifiers (in any order, distinguished by their role

[9] All of the terms "head", "specifier", "argument" and "adjunct" are, of course, subject to theory-dependent interpretation. We believe that there is enough structure here to accommodate most modern syntactic theories. Many theories will decide not to use parts of this structure, for instance avoiding the use of specifiers or avoiding any distinction between adjuncts and arguments.

[10] Since it is a partial function from *SynFeat*s, the number of elements in a *FVM* is limited to the number of *SynFeats* in the particular instantiation being used.

names). Here the MTT relations `I` and `II` have been used for *SynFun*s (subject and object) and `ATTR` has been used for general adjuncts (*Adj*). In the example, there is a main *SynRep* ("blow your nose") and two subsidiary *SynRep*s corresponding to the subject (unexpressed "you") and object ("your nose"). The latter has a non-empty specifier (another *SynRep*) corresponding to a genitive pronoun. In this case, a specifier such as "your" has been treated as a kind of adjunct, although one could have distinguished it specially through the use of a different role name. Note that this example contains exactly the information in a RAGS *SynRep*, even although the organisation of this material is perhaps different from what one would first think of, given the abstract type definitions.

SynRep is based on the D-structure of GB (Chomsky 1981), the functional structure of LFG (Bresnan 1982), the ARG(UMENT)-ST(RUCTURE) of HPSG(4) (Manning and Sag 1999), Meteer's Text Structure (Meteer 1992) and the *D(eep) Synt(actic) R(epresentation)s* of MTT (Mel'čuk 1988). They correspond to the "abstract syntax" used in leaves of Reiter and Dale's Text Specifications.

3.1.6 Quote representations

Quote representations allow for the referencing of fixed pieces of output (speech, text, graphics) which are being considered for direct inclusion into the output of the NLG system. *Quote* only refers to canned material ready for the "output" stage of generation. Other kinds of canned material (e.g., canned syntactic or semantic representations) can be represented by the other levels without any changes—in general, we understand "canned" to mean simply that a complex representation has been retrieved as a single item rather than constructed from first-principles.

Since RAGS has nothing to say about the internal structure of a quote (and the NLG system will not in general consider it to have any), *Quote* is a Primitive type:

$$Quote \quad \in \quad Primitives$$

It could be argued that when an NLG system uses a piece of fixed output there is no need to represent this apart from actually producing this output. However, when output is assembled from a mixture of fixed and generated material, it is useful for a system to be able to represent where the fixed material is to appear. A template, for instance, can be viewed as a mixed representation with some fixed parts and some other parts that will be filled in "from first principles".

For the case of fixed *text*, *Quote*s correspond to the *orthographic strings* of Reiter and Dale. That is, they are outputs that will undergo no further processing.

3.2 Low level objects and arrows model

The high level data types characterise data structures which are plausible linguistic 'objects'. However, the flexible creation and manipulation of data by NLG systems places extra demands on what should be allowed (Mellish, Evans, Cahill, Doran, Paiva, Reape, Scott and Tipper 2000):

- structures may exist at different levels of specificity—an underspecified structure might still be 'complete' enough to be part of a module's results;
- implementations (particularly in syntactic representations) need to allow reentrancy—making a distinction between substructures that are type vs token identical;
- structures may be 'mixed'—that is a complex consisting of data of more than one type may be treated as a single 'object'. For instance, a number of linguistic theories make use of "signs" which combine information at several levels (semantic, syntactic, phonological).

To express a position on issues such as these, it is necessary for RAGS to specify more than the abstract type definitions. Otherwise attempts to share data may founder because of differences in assumptions at an implementation level.

The 'objects and arrows' model is a "reference implementation" of the abstract type definitions which defines in a formal way what possible data configurations are allowed[11]. It "implements" RAGS data as typed directed graphs (which need not be rooted), for instance as used in Computational Linguistics by systems such as TFS (Emele and Zajac 1990). The purpose of this mathematical characterisation is to allow a precise recursive definition of the legal information states (Cahill *et al.* 2001b; Mellish and Evans 2004). The importance of the objects and arrows model lies not in the raw material used but in its ability formally to define a low-level model of data that caters for the above NLG requirements. An implementation of the RAGS data model is faithful just in case its set of possible data configurations corresponds exactly to those of the objects and arrows model.

In the objects and arrows model, the typed nodes are called "objects" and the edges "arrows". Edges are in fact divided into two types: "local arrows" indicate the relationship between an object and one of its components and "non-local arrows" link together objects at different levels to make mixed representations. For local arrows, the label *el* is used to indicate components of sets and functions, and labels of the form *n-el* (*n* an integer) are used to indicate components of fixed-size tuples and sequences. For non-local arrows, there is an extendable repertoire of labels indicating possible relationships between the linked objects (e.g., *realised_by* indicates the relation between two objects representing the same linguistic entity but at different stages in the generation process).

Figure 10 shows how the objects and arrows model would treat the example rhetorical representation of Figure 6. In this example there are non-local arrows between the *RhetLeaf*s and the corresponding *SemRep*s, indicated by dotted lines with label *refers_to*. The *SemRep*s in this example are underspecified (they could mark places where another module is expected to add content, for instance).

Although this model can be interpreted as a direct specification of how to implement RAGS data (and we have produced implementations of this kind), its main role in the framework is to characterise the set of legal RAGS datasets. As

[11] The name 'objects and arrows' model is perhaps not optimal, as it refers to the syntax of the model, rather than its role in RAGS. However, we retain the name for compatibility with our other publications.

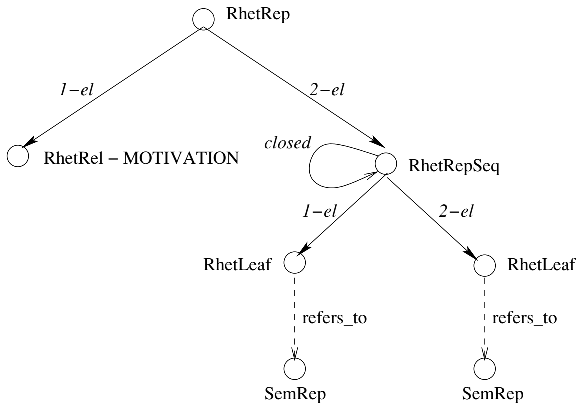


Fig. 10. “Objects and arrows” version of a rhetorical representation.

long as an implementation produces results that are in one-to-one correspondance with objects and arrows configurations (i.e., it preserves the exact set of distinctions that the objects and arrows model does) then those results are valid RAGS data (and can, for instance, be mapped in an information-preserving way into the RAGS XML format).

3.3 Examples of complex RAGS *data*

In this section we give an indication of how a variety of complex representations can be created by mixing the basic elements of the RAGS representations. For precision, we show these examples through graphs built according to the objects and arrows model. Real implementations may, of course, make use of arbitrary native formats that are in correspondance with the graphs.

3.3.1 Mixed structures

Mixed structures are data objects which include components of more than one high level data type, for example, semantic and syntactic representations. Such mixed representations are endemic in NLG systems because the more abstract levels provide the glue to hold the more concrete representations together, particularly above the sentence level. In the RAGS Objects and Arrows Model, ‘local’ structures (connected networks of local arrows) represent instances of a single high level data type and non-local arrows link them together into larger, mixed structures.

The most generic non-local arrow is `refers_to`, as used in Figure 10. It allows one local structure to include a semantically netural reference to another local structure. In figure 10 each *RhetLeaf* is associated with a (completely underspecified) *SemRep*, but no particular semantic role for the *SemRep* is implied, beyond being the *SemRep* associated with this particular *RhetLeaf*. A similar relationship holds in document structures, between *DocLeaf* objects and their associated *SynRep*s.

A less neutral non-local relationship is provided by the `realised_by` arrow. `realised_by` also links local structure together into mixed structures, but carries

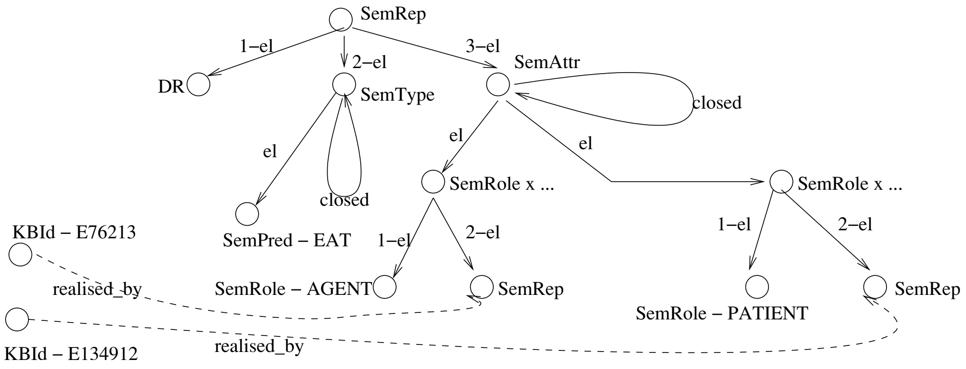


Fig. 11. Mixture of conceptual and semantic representations.

with it the implication that the object at the end of the arrow is a 'realisation' of the object at the beginning, that is that it is closer to the concrete output form. Typically in an NLG system one might expect to find *KBId* objects realised by *SemRep* objects which are realised by *SynRep* objects which may be realised by *Quote* objects. Examples of such links can be seen in Figures 11, 12 and 13.

3.3.2 Partial structures

During NLG processing it is often useful to refer to (high level) data objects which are not yet completely determined. In RAGS this is achieved through the use of underspecified high level representations. The mapping from high level data to objects and arrows defines the granularity of underspecification allowed: objects and arrows structures are always fully specified, but may represent only part of a high level structure, simply by leaving out parts of the complete network. This is similar in effect to underspecification of, say Prolog terms, although the granularity is slightly different—in RAGS it is possible to underspecify the number of arguments in a term, for example.

For example, during the top-down development of a semantic representation, it may be useful to create placeholders for semantic subcomponents so that their relationship with conceptual structure can be recorded, but before their full semantic specification has been decided. Figure 11 shows how this can be captured via a mixed, partial representation.

Here the top level of semantic representation, representing the predicate 'EAT' is in place, complete with a specification of its semantic roles, 'AGENT' and 'PATIENT'. The fillers for those roles, however, are completely underspecified *SemRep*s, included so that they can be the target of *realised_by* arrows for their corresponding *KBId*s—the conceptual agent and patient of the eating event.

In terms of Reiter and Dale (2000), such mixed conceptual/semantic representations correspond to the notion of a *message*, and when combined with the leaves of rhetorical representations, as shown in figure 10, the result corresponds to a *document plan*.

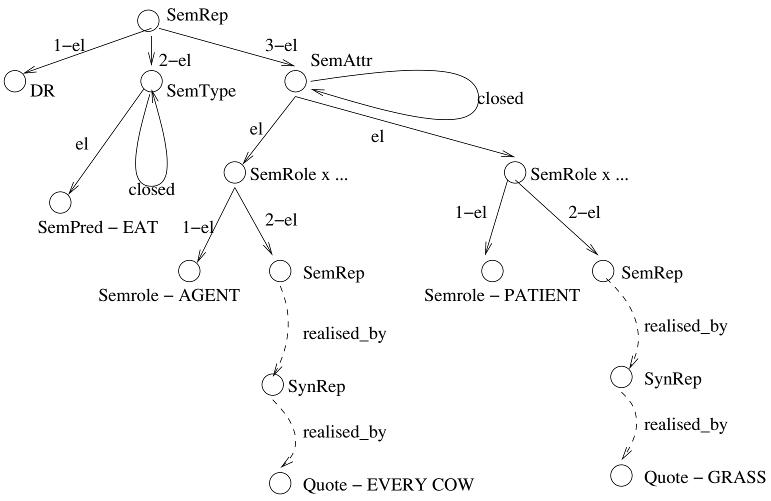


Fig. 12. Mixture of semantic and quote representations.

3.3.3 Quote representations

Figure 12 shows an example of the use of *Quote* representations. *Quote* objects allow the system to manipulate fragments of output directly, for example pictures, or actual orthographic words and word strings. The figure shows a semantic representation where two role fillers are associated with quoted literal strings to realise them. In this example, the semantic content of the role fillers is probably irrelevant (and not worth further determining if it is partial), as the existence of the *realised_by* arrows represents a strong suggestion to use fixed phrases instead of following a more first-principles approach through syntax.

Quoted material can be directly related to rhetorical structure (e.g., discourse marker expressions) and document structure (e.g., pictures, bullets) as well as syntactic and semantics representations. Such mixed representations are similar to proposals in, for instance, Reiter, Mellish and Levine (1992), Busemann and Horacek (1998) and Pianta and Tovena (1999).

3.3.4 Specifying the input to a sentence realiser

As a final example we illustrate how the RAGS framework is flexible enough to represent structures required by the commonly used FUF/SURGE realisation system (Elhadad and Robin 1992). Figure 13 shows a mixed semantic/syntactic representation akin to the input to FUF/SURGE for the sentence “this item is made from chrome”, and similar to the input for a number of other recent sentence realisers (e.g., YAG (Channarukul, McRoy and Ali 2000), HALOGEN (Langkilde-Geary 2002) and AMALGAM (Corston-OIiver, Gamon, Ringger and Moore 2002)).

Here the overall structure of a clause is described mainly at a semantic level, although some syntactic information is also provided. The semantic participants, however, are described only in terms of their syntactic properties—they are completely

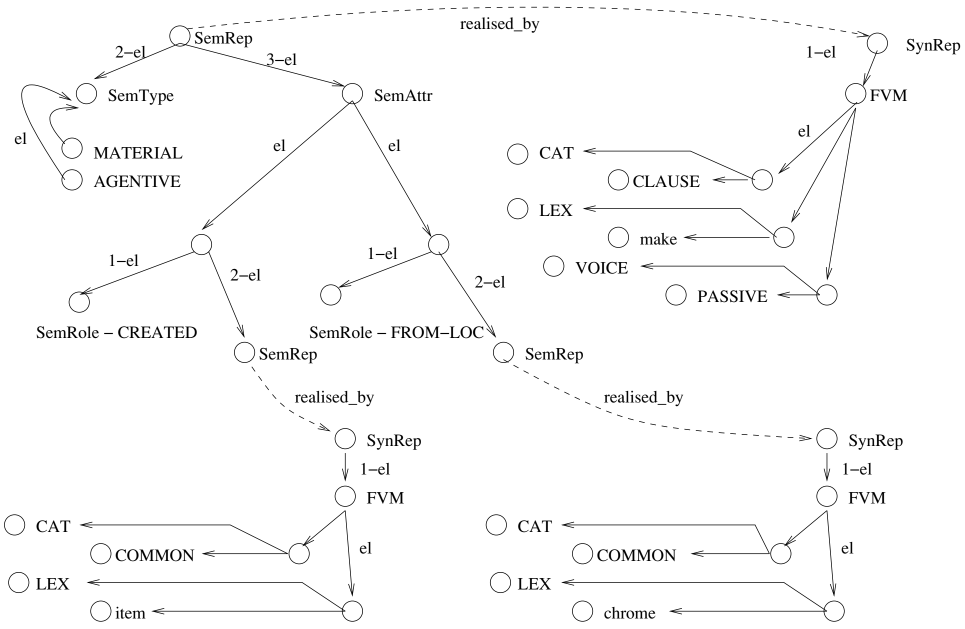


Fig. 13. FUF/SURGE-style mixed representation.

underspecified semantically (or more precisely, the FUF/SURGE engine has no interest in any semantic specification they may have). In an implementation, this might correspond to a single input structure such as:

```
((CAT CLAUSE)
(PROCESS
((TYPE MATERIAL) (EVENT-TYPE AGENTIVE)
(LEX "make") (VOICE PASSIVE)))
(PARTIC ((CREATED ((CAT COMMON) (LEX "item")))))
(CIRCUM
((FROM-LOC
((CAT COMMON) (LEX "chrome") (DEFINITE NO) (COUNTABLE NO))))))
```

3.4 Off-line data representation—the XML interface

An important part of any initiative to support sharing of data resources is a clear specification of how those resources should be stored off-line. RAGS uses a plain text representation using XML markup. As well as supporting easy storage and distribution of data on any platform, XML supports manual or automatic creation and editing of resources and provides a universal language for communication between RAGS modules via text streams where no more tightly coupled protocols exist.

The XML representation is described fully in Cahill *et al.* (2001b). The representation of a RAGS dataset echoes directly its objects and arrows form.

Table 1. *Possible modules for operations which frequently arise in NLG systems*

Module	Description	Primary input	Output
CON	Content selection		KBId
RHET	Rhetorical structuring	KBId	RhetRep
DOC	Document structuring	RhetRep	DocRep
AGG	Aggregation	*	*
LEX	First stage of lexicalisation	KBId	SemRep
TLC	True lexical choice	SemRep	SynRep
REG	Referring expression generation	KBId	SemRep/SynRep
REAL	Surface realisation	SynRep	Quote

3.5 Processing issues

The data model introduced above is the most significant and constraining part of the RAGS framework. This is because the RAGS survey revealed insufficient consensus on processing matters to draw strong conclusions. Nevertheless RAGS makes two contributions with respect to processing issues in NLG systems.

Firstly, it is our belief that smaller modules based on linguistic functions are more likely to be definable in an agreed way within the RAGS framework than larger structural components (e.g., text planning). Our survey identified a list of possible modules for operations which frequently arise in NLG systems, but following the analysis of these functions within actual systems, and the detailed development of the data model, the following table gives a refined list of functional modules that look promising for further definition and sharing/reusability between systems (see Table 1).

In this list, lexical choice has been divided into two stages (c.f. the discussion of lexicalisation in Figure 2), the first corresponding to an initial decision about (possibly language-independent) linguistic content and the second choosing a particular word/phrase of the target language, taking into account for instance stylistic features (see Cahill (1998) for further discussion of such distinctions).

The possible scheduling of modules is, of course, constrained by their respective inputs and outputs. Thus, for instance, "true lexical choice" cannot take place before *SemRep*s are available (possibly through the action of the first stage of lexicalisation or referring expression generation).

Secondly, in our own implementations we have actively explored possible processing/scheduling regimes for modules. Although RAGS is deliberately not prescriptive about how communication between modules should be achieved, several of our own implementations are constructed around an event-driven blackboard architecture, where modules communicate objects and arrows structures via shared blackboards, generating and responding to 'publication' events[12]. Such an architecture can be configured to produce almost any desired control strategy (including pipelining, parallel processing on multiple machines etc.) and has already been

[12] This idea is discussed in greater detail in the context of the RICHES system (Cahill *et al.* 2001a).

motivated for NLG work (Nirenburg, Lesser and Nyber 1989; Wanner and Hovy 1996). We have also explored interfacing modules where data is not shared but is transported directly between modules when it is required.

4 Implementations

Throughout the development of the framework, we also undertook more concrete implementation projects, to ensure that our ideas remained practically grounded and to demonstrate the applicability of the framework in both existing and new NLG application contexts. The later implementations will also serve as the basis for supporting ongoing development of new RAGS applications and resources.

4.1 CGS reconstruction

The first implementation project was a reconstruction of the Caption Generation System (CGS) (Mittal *et al.* 1998). This reconstruction is described in detail in Cahill *et al.* (2001a). Its main purpose was to demonstrate the applicability of the RAGS approach to a real, non-trivial and independently developed NLG system. The overall architecture of CGS, and the way it manipulated and developed data, was recast in RAGS terms and small-scale implementations of the modules (sufficient for just a few examples) were developed.

This experience showed that analysis of an existing NLG system in RAGS terms was possible and gave useful insights into the system's behaviour. It also demonstrated the value of all aspects of the data model: the division of data into linguistic layers, the manipulation of partial and mixed structures, and the use of a common low level framework to support inter-module communication and scheduling in a uniform fashion. None of the inter-module interfaces of CGS turned out to use a single level of representation from RAGS, but all could be modelled appropriately using the facilities to create partial and mixed structures.

4.2 ILEX and RAGSOCKS

Two further implementation projects were based on the ILEX system developed at Edinburgh (O'Donnell, Knott, Mellish and Oberlander 2001). The first project concentrated on formalising the major interface within a reimplementation, EXPRIMO, of the system, that between the text planner and the realiser. In EXPRIMO, it is possible to produce a file containing an XML version of the input to the realiser and then to read that file back into the realiser to accomplish realisation. The "RAGSification" of the interface was accomplished by writing two XSL mappings, one from the Exprimo XML format to RAGS XML format and other in the reverse direction.

The other ILEX-related project involved building a system broadly comparable to ILEX by putting together several new modules implemented in LISP, Prolog and Java, as well as an existing stochastic text planner and the FUF/Surge realisation system (Elhadad and Robin 1992). This implementation allowed us to experiment with scheduling modules with point-to-point communication using sockets and to develop general code for translating from native LISP, Java and Prolog data formats into RAGS XML (and vice versa). This support code is available as the "RAGSOCKS" package from our website. The resulting system allows interesting datasets to be created and provides examples of packaging existing modules for use in a RAGS-based system.

4.3 RICHES and OASYS

The final implementation activity involved the implementation of a completely new NLG system, RICHES (Cahill *et al.* 2001a), based on the RAGS framework. Our goals here were to develop an architecture unconstrained by a prior implementation and to further develop our ideas about, and support for, inter-module communication and scheduling. The modules of RICHES were in fact not all new—we re-used modules from the CGS implementation and the ICONOCLAST system (Bouayad-Agha, Scott and Power 2000; Power *et al.* 2003), as well as the LinGo sentence realiser (Carroll, Copestake, Flickinger and Poznanski 1999)—but their organisation was, and the application manager, OASYS, was a completely new implementation, extending the ideas used in the CGS system. In addition, RICHES included a medium selection module that selected and inserted pictures into a document as representations of semantic content.

This implementation demonstrated the RAGS framework's ability to support more complex, non-pipelined architectures in an effective combination of re-used and new modules. It also showed that it was possible to experiment with different control regimes between modules and provide diagnostic support during application development entirely within the scope of the framework. Finally it provided a number of independently useful modules, notably OASYS itself, media selection, lexical choice and referring expression generation modules, for use in future systems.

5 How to Use RAGS

Developing an NLG system using RAGS involves the following stages:

1. Deciding which of the six levels of representation will arise in the implementation and how they will be instantiated. It is to be hoped that various standard instantiations of the RAGS primitive types will become popular. Therefore it is worth trying to adopt an existing instantiation (e.g., a version of rhetorical representations that is based on the original RST work) or to propose a new one in a way that others can easily take up.
2. Deciding on the module structure of the system and the module inputs and outputs. RAGS provides a very good basis for defining module inputs and outputs precisely, although of course not all module interfaces can be defined in a RAGS-compatible way. Decisions could be influenced by the possibility of reusing an existing module if one specifies the system appropriately (possibly mapping between different but similar instantiations of a RAGS type).
3. Choosing a programming language or languages and an implementation of the selected RAGS types in a way that is faithful to the Objects and Arrows

Model, making use of one of the native formats that we provide code for, or devising a new format with (if needed) XML input/output.

4. Choosing an approach to scheduling and inter-module communication. This might use facilities within one programming language and process, or using XML and files/sockets. Again we provide some example code that could be adapted.
5. Implementing the modules in a way that makes explicit the RAGS-compatible interfaces (so that modules can potentially be separated for reuse and the data passing across the interfaces can be collected if needed).
6. Documenting and publishing the RAGS-compatible modules for reuse.

The ILEX reimplementation discussed in section 4.2 gives one example of how one might build a system using RAGS. In this case, the decision was made to reuse a Prolog version of the ILEX database ("content potential"), a Prolog stochastic text planner and FUF/Surge in the context of a system with most new code being written in LISP. In this case, none of these components had previously been given RAGS interfaces, and so the first step was to do this, for the different components independently.

Interfacing the database. The ILEX database has a simple format (three kinds of objects—entities, facts and relations), and it was straightforward to relate this to the API for *KBId*s.

Interfacing the text planner. The stochastic text planner takes as input a set of facts and relations that are to be organised into a text structure. In RAGS terms, the relevant information in a fact is the predicate (a partial *SemRep*) and the sequence of arguments providing essentially a Centering-theory CF list (a partial *SynRep*, with links back to the *KBId*s of the entities involved). The relations could be characterised in terms of partial *RhetRep*s with leaves linked to the *SemRep*s for the relevant facts. The result returned by the text planner is similar to the input, but with a single *RhetRep* linking all the facts together and a *DocRep* specifying the order of realisation of the elements of the *RhetRep* tree.

Interfacing FUF/SURGE. As indicated in section 3.3.4, the input to FUF/Surge can be regarded as a mixture of syntactic and semantic information. It was necessary to decide which of the information in FUF/Surge is syntactic and which semantic. For semantic information, a conversion between FUF/Surge feature-value pairs and RAGS *SemPred*s had to be designed (in fact, some of the presentations of SURGE already treated these as essentially unary predicates). The output of FUF/Surge was treated as a single *Quote*.

For each of the above modules, a "wrapper" was built using the RAGSOCKS software to handle input/output of RAGS structures (via XML), each module acting as a "server", receiving valid inputs and returning the appropriate outputs to whoever requested them. The wrappers also had to convert between the native formats supported by RAGSOCKS and the native formats that were already used in the programs (of course, if these modules had been written with RAGS in mind, they might have used more similar formats). The result was a set of modules that could potentially be reused in other applications compatible with RAGS. Of these, the text planner has been documented and is available from our web site; work is underway to release the RAGS interface to FUF/Surge as well. Finally, a main controlling program was written in LISP to call the other modules as required, building the appropriate RAGS structures for their inputs and decoding the outputs as necessary. The controlling module also carried out other tasks, for instance content determination. A further very simple referring expression generation module was built and interfaced, although the intention is at some time to replace this with an improved one based on existing work.

6 Conclusions and future directions

We believe that RAGS has made an important step towards helping the NLG research community to share both data and processing resources. The key contributions of this work are:

- a detailed analysis of the three-stage pipeline model and assessment of its suitability as a generic architecture;
- the development of high level data type specifications firmly based on current practice in NLG;
- the development of the two-level data model (high level specifications and objects and arrows representation) as an approach to defining and managing the complex data interactions found in NLG systems;
- the outline specification of a functional procesisng model for NLG systems;
- the specification of a standard offline data storage and exchange representation;
- the development of sample implementations of RAGS technology and complete implementations of RAGS systems.

However, RAGS is only a step towards better understanding what NLG systems do and supporting software and data reuse. Here are some areas which remain to be explored:

- RAGS only defines levels of representation corresponding roughly to descriptions of the linguistic output of NLG, at different levels of abstraction. An NLG system needs to keep track of representations other than these, for instance goals, user models and discourse models. We have neglected these so far because there seems less common agreement about what is required and what form it should take.
- The definitions so far have also implicitly concentrated on the problems of producing text output, rather than speech. It is to be hoped that document representations will be able to handle aspects of the suprasegmental structure of speech, although this remains to be investigated.
- RAGS has little to say about content determination, although the definition of the API for *KBId*s is at least a start towards specifying some aspects of it in a domain-independent way. We also do not deal with concrete syntax or concrete document structure.

- More fundamentally, where RAGS does describe data, the definitions stop at the "primitives", which have to be defined in a theory-dependent way. A major part of reusing data involves coming to an agreement on the primitives, and yet RAGS has nothing to say about this. It is to be hoped nevertheless that RAGS helps focus attention on the places where theoretical discussion is productive and away from notational differences that have no deep theoretical significance.
- Finally, the example software and data available from our web-site demonstrates the basic principles, but now needs to be extended into a library of reusable modules and support software for RAGS development.

Although the first stage of the RAGS project has officially ended, further work is continuing. The two initial directions in which we intend to make further progress are the publication of further resources and the development of the framework to cover speech generation. We are also exploring the application of the framework to other areas of NLP (such as understanding) and looking at its applicability to other tasks with complex data manipulation requirements.

7 Acknowledgements

We would like to acknowledge the financial support of the EPSRC (grant GR/L77102 to Scott, grant GR/L77041 to Mellish). We also acknowledge the contribution of former colleagues Christy Doran, Neil Tipper and Rodger Kibble, other colleagues at Brighton and Edinburgh, the invaluable contributions of our RAGS 'consultants' and the wider NLG community.

This paper has improved significantly as a result of the comments of the three anonymous referees, and we are grateful for their efforts.

References

Alshawi, H. (editor) (1982) *The Core Language Engine*. MIT Press.

Bateman, J. (1990) Upper modeling: Organizing knowledge for natural language processing. *Proceedings of the Fifth International Natural Language Generation Workshop (INLG'90)*, pp. 54–56. Dawson, PA.

Bontcheva, K., Tablan, V., Maynard, D. and Cunningham, H. (2004) Evolving GATE to meet new challenges in natural language engineeing. *Natural Language Engineering*, **10**(3/4): 349–373.

Bordegoni, M., Faconti, G., Feiner, S., Maybury, M., Rist, T., Ruggieri, S., Trahanias, P. and Wilson, M. (1997) A standard reference model for intelligent multimedia presentation systems . *Computer Standards and Interfaces*, **18**: 477–496.

Bouayad-Agha, N., Scott, D. and Power, R. (2000) Integrating content and style in documents: a case study of patient information leaflets. *Information Design Journal*, **9**(2): 161–176.

Bresnan, J. (editor) (1982) *The Mental Representation of Grammatical Relations*. MIT Press.

Busemann, S. and Horacek, H. (1998) A Flexible Shallow Approach to Text Generation. *Proceedings of the Ninth International Workshop on Natural Language Generation, (INLG'98)*, pp. 238–247. Niagara-on-the-Lake, Canada.

Cahill, L. (1998) Lexicalisation in applied NLG systems. Technical Report ITRI-99-04, ITRI, University of Brighton.

Cahill, L. and Reape, M. (1998) Component tasks in applied NLG systems. Technical Report ITRI-99-05, ITRI, University of Brighton.

Cahill, L., Carroll, J., Evans, R., Paiva, D., Power, R., Scott, D. and van Deemter, K. (2001a) From RAGS to RICHES: exploiting the potential of a flexible generation architecture. *Proceedings of 39th Annual Meeting of the Association for Computational Linguistics (ACL'01)*, pp. 98–105. Toulouse, France.

Cahill, L., Evans, R., Mellish, C., Paiva, D., Reape, M. and Scott, D. (2001b) The RAGS Reference Manual . Technical Report ITRI-01-08, ITRI, University of Brighton.

Caldwell, D. and Korelsky, T. (1994) Bilingual Generation of Job Descriptions from Quasi-Conceptual Forms. *Proceedings of the Fourth Conference on Applied Natural Language Processing (ANLP'94)*, pp. 1–6. Stuttgart, Germany.

Carroll, J., Copestake, A., Flickinger, D. and Poznanski, V. (1999) An efficient chart generator for (semi-)lexicalist grammars. *Proceedings of the 7th European Workshop on Natural Language Generation (EWNLG'99)*, pp. 86–95. Toulouse, France.

Channarukul, S., McRoy, S. and Ali, S. (2000) Enriching partially-specified representations for text realization using an attribute grammar. *Proceedings of the First International Natural Language General Conference (INLG'00)*, pp. 163–170. Mitzpe Ramon, Israel.

Chomsky, N. (1981) *Lectures on Government and Binding*. Foris, Dordrecht.

Coch, J. and David, R. (1994) Representing Knowledge for Planning Multisentential Text. *Proceedings of the Fourth Conference on Applied Natural Language Processing (ANLP'94)*, pp. 203–204.

Copestake, A., Flickinger, D., Sag, I. and Pollard, C. (1999) Minimal Recursion Semantics: An introduction. Internal Report, CSLI, Palo Alto.

Corston-Oliver, S., Gamon, M., Ringger, E. and Moore, R. (2002) An overview of AMALGAM: A machine-learned generation module. *Proceedings of the Second International Natural Language Generation Conference (INLG'02)*.

Creaney, N. (1996) An algorithm for generating quantifiers. *Proceedings of the Eighth International Workshop on Natural Language Generation (INLG'96)*, pp. 121–130. Herstmonceux, Sussex, UK.

Cunningham, H., Wilks, Y. and Gaizauskas, R. (1996) GATE – a general architecture for text engineering. *Proceedings of the 16th International Conference on Computational Linguistics (COLING'96)*, vol. 2, pp. 1057–1060.

Danlos, L. (1984) Conceptual and linguistic decisions in generation. *Proceedings of the 10th International Conference on Computation Linguistics (COLING'84)*, pp. 501–504. Stanford, CA.

De Smedt, K., Horacek, H. and Zock, M. (1996) Architectures for Natural Language Generation: Problems and Perspectives. *Trends in Natural Language Generation*, pp. 17–46. Springer Verlag.

Elhadad, M. and Robin, J. (1992) Controlling content realization with functional unification grammars. In: D. R. R. Dale, E. Hovy and O. Stock, editors, *Aspects of Natural Language Generation*, number 587 in *Lecture Notes in Artificial Intelligence*, pp. 89–104. Springer Verlag.

Emele, M. and Zajac, R. (1990) Typed unification grammars. *Proceedings of the 13th International Conference on Computational Linguistics (COLING'90)*, pp. 291–298.

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 (INLG'96)*, pp. 51–60. Herstmonceux, Sussex, UK.

Frank, A. and Reyle, U. (1995) Principle-based semantics for HPSG. *Proceedings of the 6th Meeting of the Association for Computational Linguistics, European Chapter (EACL'95)*, pp. 9–16. Dublin.

Gazdar, G., Klein, E., Pullum, G. and Sag, I. (1985) *Generalized Phrase Structure Grammar*. Harvard University Press.

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

Grosz, B and Sidner, C. (1986) Attention, intentions and the structure of discourse. *Computational Linguistics*, **12**(3): 175–204.

Halliday, M. (1994) *An Introduction to Functional Grammar*. Arnold, London.

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

Hobbs, J. and Shieber, S. (1987) An algorithm for generating quantifier scopings. *Computational Linguistics*, **3**(1): 47–63.

Hobbs, J. R. (1985) On the coherence and structure of discourse. Technical Report CSLI-85-37, CSLI, Palo Alto, CA.

Huang, X. (1994) Planning argumentative texts. *Proceedings of the 15th International Conference on Computational Linguistics (COLING'94)*, pp. 329–333. Kyoto, Japan.

Iordanskaja, L., Kittredge, R., Lavoie, B. and Polguère, A. (1992) Generation of extended bilingual statistical report. *Proceedings of the 15th International Conference on Computation Linguistics (COLING'92)*, pp. 1019–1023. Nantes, France.

Kamp, H. and Reyle, U. (1993) *From Discourse to Logic: Introduction to Modeltheoretic Semantics of Natural Language, Formal Logic and Discourse Representation Theory*. Kluwer.

Kasper, R. T. (1989) A flexible interface for linking applications to PENMAN's sentence generator. *Proceedings of the DARPA Workshop on Speech and Natural Language*, Philadelphia.

Kay, M. (1996) Chart generation. *Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics (ACL'96)*, pp. 200–204.

Kittredge, R. and Polguére, A. (1991) Generating extended bilingual texts from application knowledge bases. *Proceedings on Fundamental Research for the Future Generation of Natural Language Processing*, pp. 147–160. Kyoto, Japan.

Kukich, K. (1988) Fluency in natural language reports. In: D. McDonald and L. Bolc, editors, *Natural Language Generation Systems*, pp. 280–311. Springer Verlag.

Langkilde-Geary, I. (2002) An empirical verification of coverage and correctness for a general-purpose sentence generator. *Proceedings of the Second International Natural Language Generation Conference (INLG'02)*, pp. 17–24. New York.

Lavoie, B. and Rambow, O. (1997) A fast and portable realizer for text generation systems. *Proceedings of the Fifth Conference on Applied Natural Language Processing (ANLP'97)*, pp. 265–68. Washington, DC.

Lavoie, B., Rambow, O. and Reiter, E. (1996) The ModelExplainer. *Proceedings of the Eighth International Workshop on Natural Language Generation (INLG'96)*, pp. 9–12. Herstmonceux, Sussex, UK.

Loeber, S., Aroyo, L. and Hardman, L. (2002) An explicit model for tailor-made ecommerce web presentations. *Proceedings of the Workshop on Recommendation and Personalization in eCommerce – 2nd International Conference on Adaptive Hypermedia and Adaptive Web Based Systems*, pp. 108–115. Malaga, Spain.

Mann, W. and Thompson, S. (1988) Rhetorical Structure Theory: Toward a functional theory of text organization. *Text*, **8**(3): 243–281.

Mann, W. (1983) An overview of the Penman text generation system. *Proceedings of the Third National Conference on Artificial Intelligence (AAAI'83)*, pp. 261–265. Washington, DC.

Manning, C. and Sag, I (1999) Dissociations between ARG-ST and grammatical relations. In: G. Webelhuth, J. Koenig and A. Kathol, editors, *Lexical and Constructional Aspects of Linguistic Explanation*, pp. 63–78. CSLI, Stanford.

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

McKeown, K., Kukich, K. and Shaw, J. (1994) Practical issues in automatic documentation generation. *Proceedings of the Fourth Conference on Applied Natural Language Processing (ANLP'94)*, pp. 7–14. Stuttgart, Germany.

McRoy, S. and Ali, S. (1999) A practical, declarative theory of dialogue. *Proceedings of the IJCAI-99 Workshop on Knowledge and Reasoning in Practical Dialogue Systems*, pp. 97–103.

Mellish, C. (2000) Understanding shortcuts in natural language systems. *Workshop on Impacts in Natural Language Generation*, pp. 43–50. Schloss Dagstuhl, Germany.

Mellish, C. and Evans, R. (2004) Implementation architectures for natural language generation. *Natural Language Engineering*, **10**(3/4): 261–282.

Mellish, C., Knott, A., Oberlander, J. and O'Donnell, M. (1998) Experiments using stochastic search for text planning. *Proceedings of the Ninth International Workshop on Natural Language Generation (INLG'98)*, pp. 98–107. Niagara-on-the-Lake, Ontario.

Mellish, C., Evans, R., Cahill, L., Doran, C., Paiva, D., Reape, M., Scott, D. and Tipper, N. (2000) A representation for complex and evolving data dependencies in generation. *Proceedings of the Language Technology Joint Conference, ANLP-NAACL2000*, Seattle, WA.

Mel'čuk, I. (1988) *Dependency Syntax: Theory and Practice*. SUNY Series in Linguistics. State University of New York Press, Albany, NY.

Meteer, M. (1992) *Expressibility and the Problem of Efficient Text Planning*. Pinter.

Mittal, V. O., Moore, J. D., Carenini, G. and Roth, S. (1998) Describing complex charts in natural language: A caption generation system. *Computational Linguistics*, **24**(3): 431–468.

Moser, M. and Moore, J. D. (1996) Toward a synthesis of two accounts of discourse structure. *Computational Linguistics*, **22**(3): 409–420.

Nirenburg, S., Lesser, V. and Nyberg, E. (1989) Controlling a language generation planner. *Proceedings of the 11th International Joint Conference on Artificial Intelligence*, pp. 1524–1530.

O'Donnell, M., Knott, A., Mellish, C. and Oberlander, J. (2001) *ILEX*: The architecture of a dynamic hypertext generation system. *Natural Language Engineering*, **7**: 225–250.

Paiva, D. (1998) A survey of applied natural language generation systems. Technical Report ITRI-98-03, Information Technology Research Institute (ITRI), University of Brighton.

Paris, C., Vander Linden, K., Fischer, M., Hartley, A., Pemberton, L., Power, R. and Scott, P. (1995) A support tool for writing multilingual instructions. *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, pp. 1398–1404. Montreal, Canada.

Pianta, E. and Tovena, L. (1999) Mixing representation levels: The hyrbid approach to automatic text generation. *Proceedings of the AISB'99 Workshop on Reference Architectures and Data Standards*, University of Edinburgh.

Pollard, C. and Sag, I. (1987) *Information-Based Syntax and Semantics. Vol 1: Fundamentals.* CSLI, Stanford.

Pollard, C. and Sag, I. (1994) *Head-Driven Phrase Structure Grammar*. University of Chicago Press, Chicago.

Power, R. and Cavallotto, N. (1996) Multilingual generation of administrative forms. *Proceedings of the Eighth International Workshop on Natural Language Generation (INLG'96)*, pp. 17–19. Herstmonceux, Sussex, UK.

Power, R., Scott, D. and Bouayad-Agha, N. (2003) Document Structure. *Computational Linguistics*, **29**(2): 211–260.

Rambow, O. (1999) Domain communication knowledge. *Proceedings of the Fifth International Natural Language Generation Workshop*, pp. 87–94. Dawson, PA.

Reiter, E. (1994) Has a consensus NL generation architecture appeared and is it psycholinguistically plausible? *Proceedings of the Seventh International Natural Language Generation Workshop (INLG'94)*, pp. 163–170. Kennebunkport, Maine.

Reiter, E. and Dale, R. (2000) *Building Natural Language Generation Systems.* Cambridge University Press.

Reiter, E. and Robertson, R. (1999) The architecture of the STOP system. *Proceedings of the Workshop on Reference Architectures for Natural Language Generation*, Edinburgh, Scotland.

Reiter, E., Mellish, C. and Levine, J. (1992) Automatic generation of on-line documentation in the IDAS project. *Proceedings of the Third ACL Conference on Applied Natural Language Processing*, pp. 64–71. Trento, Italy.

Reyle, U. (1995) Reasoning with ambiguities. *Proceedings of the 6th Meeting of the Association for Computational Linguistics, European Chapter (EACL'95)*, pp. 1–8.

Rutledge, L., Hardman, L., van Ossenbruggen, J. and Bulterman, D. (1999) Mix'n'match: Exchangeable modules of hypermedia style. *Proceedings of the 10th ACM conference on Hypertext and Hypermedia*, pp. 179–188. Darmstadt, Germany.

Scott, D. and Sieckenius de Souza, C. (1990) Getting the message across in RST-based text generation. In: R. Dale, C. Mellish and M. Zock, editors, *Current Research in Natural Language Generation*, Cognitive Science Series, pp. 47–73. Academic Press.

Scott, D., Power, R. and Evans, R. (1998) Generation as a Solution to its own Problem. *Proceedings of the Ninth International Workshop on Natural Language Generation, (INLG'98)*, pp. 256–265. Niagara-on-the-Lake, Canada.

Sheremetyeva, S., Nirenburg, S. and Nirenburg, I. (1996) Generating patent claims from interactive input. *Proceedings of the Eighth International Workshop on Natural Language Generation (INLG'96)*, pp. 51–60. Herstmonceux, Sussex, UK.

Shieber, S. M. (1986) *An introduction to unification-based approaches to grammar*. CSLI.

Stone, M., Doran, C., Webber, B., Bleam, T. and Palmer, M. (2001) Microplanning with communicative intentions: The spud system. Technical Report TR-65, Rutgers University Center for Cognitive Science, New Jersey.

Teich, E. and Bateman, J. (1994) Towards the application of text generation in an integrated publication system. *Proceedings of the Seventh International Natural Language Generation Workshop (INLG'94)*, pp. 153–162. Kennebunkport, Maine.

van Ossenbruggen, J., Geurts, J., Cornelissen, F., Hardman, L. and Rutledge, L. (2001) Towards second and third generation web-based multimedia. *Proceedings of WWW-2001*, pp. 479–488. Hong Kong.

Vander Linden, K. (1994) GIST – Specification of the Extended Sentence Planning Language. Technical Report LRE Project 062-09 Deliverable TST-0, Information Technology Research Institute (ITRI), University of Brighton.

L. Wanner and E. Hovy. The HealthDoc Sentence Planner. *Proceedings of the Eighth International Workshop on Natural Language Generation (INLG'96)*, pp. 1–10. Herstmonceux, Sussex, UK.

Zeevat, H. (1991) *Aspects of Discourse Semantics and Unification Grammar*. Unpublished PhD thesis, University of Amsterdam.