Enabling resource sharing in language generation: an abstract reference architecture

L Cahill[†], C Doran[†]*, R Evans[†], R Kibble[†], C Mellish[‡], D Paiva[†], M Reape[‡], D Scott[†], N Tipper[†]

[†]Information Technology Research Institute, University of Brighton, Lewes Rd, Brighton, UK
[‡]Division of Informatics, University of Edinburgh, 80 South Bridge, Edinburgh, UK
rags@itri.brighton.ac.uk
http:/www.itri.brighton.ac.uk/projects/rags

Abstract

The RAGS project aims to develop a reference architecture for natural language generation, to facilitate modular development of NLG systams as well as evaluation of components, systems and algorithms. This paper gives an overview of the proposed framework, describing an abstract data model with five levels of representation: Conceptual, Semantic, Rhetorical, Document and Syntactic. We report on a re-implementation of an existing system using the RAGS data model.

1. The RAGS enterprise

The primary goal of the RAGS project (Cahill et al., 1999a) is to develop a 'reference architecture' for applied natural language generation (NLG) systems. The aim is to produce an architectural specification which reflects mainstream current practice and provides a framework for the development of new applications and new components within NLG systems. The architecture is also intended to facilitate evaluation of NLG components, algorithms and systems. To achieve these goals, such an architecture has to be sufficiently conventional to be relevant to developers of existing systems, but also sufficiently generic and detailed to be useful as a resource for novel approaches.

One of the distinctive properties of natural language generation when compared with other language engineering applications is that it has to take seriously the full range of linguistic representation, from concepts through to morphology, or even phonetics. Any processing system is only as sophisticated as its input allows, so while a natural language understanding system might be judged by its syntactic prowess – even if its attention to semantics, pragmatics and underlying conceptual analysis is minimal – a generation system is only as good as its deepest linguistic representations. This has particular implications for *evaluation* of NLG systems: it is hard to think of evaluation exercises along the lines of the MUC tasks (Hirschman and Chinchor, 1997) for information extraction, for instance, where the inputs consist of naturally-occurring text and the outputs are precisely specified by the evaluators. There is no general agreement on how *inputs* for NLG systems should be specified, nor on how output texts can be evaluated.

2. Pipelines and beyond

Generation systems, especially end-to-end, applied generation systems, have, unsurprisingly, much in common. Reiter (Reiter, 1994) proposed an analysis of such systems in terms of a simple three-stage pipeline:

Content Determination - deciding the content of a message, and organising the component propositions into a text tree;

Sentence Planning - aggregating propositions into clausal units and choosing lexical items corresponding to concepts in the knowledge base, including referring expressions (RE);

Linguistic realisation which takes care of surface details such as agreement, orthography etc.

This was presented as a "consensus" model, based on a survey of 5 existing systems and partly motivated on engineering grounds in that a strict pipeline requires a minimal number of interfaces. More recently the RAGS project attempted to repeat and extend the analysis, surveying a set of 19 applied systems (Paiva, 1998; Cahill and Reape, 1998). A component tasks analysis was undertaken examining where in each system each of a set of core linguistic *tasks* was situated. That set was: lexicalisation, aggregation, rhetorical structuring, referring expression generation, ordering, segmentation and coherence[1]. It was found that, while most systems did implement a pipeline architecture, they did not implement the same pipeline – different functionalities occurred in different places and in different orders in different systems. For instance, different systems situate the lexicalisation task in Content Determination, Sentence Planning or Realisation. Or one functional task may be distributed among different modules: in the CGS system (Mittal et al., 1998), the task of generating referring expressions is distributed among the Text Planner, Lexical Choice and the Referring Expression module itself. To accomodate the results of this survey, we find it more appropriate to specify the architecture in terms of the above set of functions and a number of distinct levels of representation, which between them support the range of implementations observed, including pipelines as well as other more complex control regimes.

* Now at the MITRE Corporation, Bedford, MA, USA, cdoran@mitre.org.

[1]More details about the assumed definitions of these tasks can be found in (Cahill and Reape, 1998; Cahill et al., 1999b).

The central pillar of our proposed architecture is a data model, in the form of a set of declarative representations for the various levels of linguistic representation in generation. That is, the functional modules are defined entirely in terms of the datatypes they manipulate and the operations they can perform on them. On top of such a model, more specific process models can be created, in terms of constraints on the order and level of instantiation of different types of data in the data model. One might then produce a 'rational reconstruction' of the pipeline model, but other process models would be equally possible. We argue that by providing a common notation for the representations, it will be possible for generation systems to ultimately share both datasets and processing modules. Although it is not our intention to impose standards on anyone, we expect that the widespread adoption of (at least) an agreed set of representations will benefit the whole generation community by permitting sharing of resources as well as easier evaluation of both whole systems and modules within systems.

We assume five levels of representation: Conceptual, Semantic, Rhetorical, Document and Syntactic, each of which may have *abstract* (provisional) and *concrete* (finalised) variants. These representations are intended to be theory-neutral and various different formalisms may be used, some of which are mentioned below.

I. Conceptual Structure is defined only indirectly through an API via which a knowledge base (providing the content from which generation takes place) can be viewed as if it were defined in a simple KL-ONE (Brachman and Schmolze, 1985) like system.

II. Semantic Structure may be represented using a formalism such DRT (Kamp and Reyle, 1993) or SPL (Kasper, 1989), and is the level at which inferences may be defined. It may be useful to distinguish an **Abstract Semantic** level, which might include pointers to conceptual entities as arguments of semantic predicates in place of terms with semantic content.

III. Rhetorical: a rhetorical structure tree with relations in a formalism such as RST (Mann and Thompson, 1987) at the nodes and pointers to semantic representations at the leaves. **Abstract Rhetorical Structure** may differ from the final structure in various ways: the relations may be selected from a more restricted, generic, set, or it may be a disjunction of semantically equivalent structure.

IV. Document Structure in its concrete form will contain all formatting information needed to print or display a document, coded in some formatting language such as HTML or LaTeX. The RAGS scheme only considers **Abstract Document Structure**, where text-level (Nunberg, 1990), layout and position features are specified without commitment to any particular formatting system.

V. Syntactic Structure may be described in a variety of formalisms, such as HPSG, LFG etc. RAGS is only concerned with **Abstract SyntacticStructure**, a high level syntactic representation (largely) independent of particular syntactic theories, covering for instance head-complement relations. We believe that most current reusable realisers take as their input some combination of semantic and abstract syntactic representations.

Note that Abstract Conceptual, Concrete Syntactic and Concrete Document levels of representation are considered outside the remit of the current enterprise. Note also that this model is intended to be permissive rather than prescriptive: there is no implication that the structures should be built in the order listed above, that one structure should be complete before another one can be initiated, or even that all levels should be instantiated in any particular system. To the contrary, we envisage that systems will make use of mixed and partial representations.

Given the above set of representations, we have defined the syntactic and semantic requirements of the formalisms, XML DTDs to aid the representation of the formalisms in an easily sharable format and example implementations of the representations. We anticipate being able to define the functionality of existing systems in terms of these representations. For example, a lexicalisation module might receive as input (at least) a set of abstract semantic structures and return a set of lexicalised abstract syntactic structures. The input/output requirements of operations impose some limits on how these can be ordered, but the intention is that the data model should reflect fairly uncontentious minimal constraints. It should be possible to derive most existing architectures by further specialising the data model and imposing extra ordering constraints. For instance, many systems do not clearly distinguish conceptual from semantic information. For these systems one simply assumes additionally that conceptual structures and semantic structures are one and the same thing. Such a data model allows the definition of a network of possible operations that take us from an initial input to a final output.

3. Inter-module communication

Systems using the RAGS model will communicate with one another by sending RAGS representations, possibly involving mixed and partial structures, to one another. The most straightforward way that two modules can communicate is if they are written in the same programming language and they agree on how RAGS representations are implemented in that programming language. In that case, the two modules can operate in the same address space and one can simply pass pointers to the other. If the two modules are implemented in different languages, or if they are running on different machines, then it is possible to communicate by sending representations in a textual form via some communication medium. The RAGS project is producing code code that supports sending data in XML using sockets, together with code for parsing and generating XML from standard programming language representations.

Alternatively, one may wish to have a central server with which all modules communicate. Clients written in the same programming language and running on the same machine as the server can then communicate using programming language specific mechanisms. Otherwise each client module could have a two-way communication channel with the server, implementing a particular query/response interface. Here we briefly describe a particularly general model of inter-module communication based around modules communicating with a single centralised repository of data called the *whiteboard* (Calder et al., 1999; Mellish et al., 2000). A whiteboard is a *cumulative typed relational blackboard*:

- **typed and relational:** because it is based on using a representation scheme consisting of
  - a set of objects, organised into types: objects may be tree structures at one of the five levels of representation, sub-trees or individual nodes;
  - a set of binary relations, organised into types: relations include `realises`, linking a more abstract structure to a less abstract version, `revises` (see below) and `coreference`;
  - a set of arrows, each indicating that a relation holds between one object and another object.
- **a blackboard:** a control architecture and data store shared between processing modules; typically, modules add/change/remove objects in the data store, examine its contents, and/or ask to be notified of changes;
- **cumulative:** unlike standard blackboards, once data is added, it can't be changed or removed. So a structure is built incrementally by making successive copies of it (or of constituents of it) linked by `revises` links.

The use of a central server is just one option for inter-module communication and is not an essential part of the RAGS architecture. The RAGS project has implemented a "whiteboard" server as part of the case study reported in Section 5.

4. Examples of data structures

In this section we show examples of the types of data structures referred to above. We concentrate on the resources needed to generate Rhetorical Representations, (RhetReps), and for purposes of exposition we disregard the distinction between abstract and concrete variants[2].

```
<?xml version="1.0"?>
<!-- RhetRep DTD -->

 <!ELEMENT RhetRel EMPTY>
 <!ATTLIST RhetRel name CDATA #REQUIRED>

 <!ELEMENT RhetRep
   (SemRep | (RhetRep RhetRep+))>
 <!ATTLIST RhetRep relation CDATA #REQUIRED>

 <!ELEMENT SemRep ANY>
```

The above is a complete XML document specifying a DTD called RhetRep. The first line identifies the document as an XML document. **RhetRep** defines the element **RhetRel** for *declaring* rhetorical relations. It is an empty element with a required name attribute. The main body defines **RhetRep** as containing either a Semantic Representation (**SemRep**) or at least two **RhetReps**.

[2]Note that the DTD presented does not handle multiple or mixed or partial representations and that the representation of semantics is over- simplified for expository purposes.

The example shown below represents a sample of text from a museum domain (example (1)) using this DTD. Note here the DOCTYPE specification references the above DTD assumed to be stored in the file "RhetRep.dtd".

(1) Jessie M. King, who lived and worked in London, designed this Arts and Crafts brooch. She also designed the necklace we saw in Case 14.

```
<?xml version="1.0"?>
<!DOCTYPE RhetRep SYSTEM "RhetRep.dtd">

<RhetRel name="background"/>

<RhetRep relation="background">
 <RhetRep relation="background">
  <RhetRep>
   <SemRep prop="designer(j-9999,King01)"/>
  </RhetRep>
  <RhetRep>
   <SemRep prop="workplace(King01,London)"/>
  </RhetRep>
 </RhetRep>
 <RhetRep>
  <SemRep prop="designer(j-8888,King01)"/>
 </RhetRep>
</RhetRep>
```

The example representations shown above might be automatically transformed from XML into a simple Prolog clause form, resulting in structures such as:

```
RhetRep:
    background(
        background(
            designer(j_9999, king01),
            workplace(king01, london)
        ),
        designer(j_8888, king01)
    ).
```

5. A case study: Caption Generation System

As a proof of concept, we have chosen an existing applied NLG system and reimplemented it according to the RAGS data model and whiteboard. (The reimplementation is reported more fully in (Mellish et al., 2000).) This reimplementation involved two stages: a reinterpretation of the system in terms of the data model and a reimplementation of the system using a whiteboard implementation in which the aim was to attempt to reuse the existing code wherever possible. The system chosen for this exercise was the Caption Generation System (CGS) (Mittal et al., 1998). CGS is a system developed at the University of Pittsburgh, which takes input from a graphics presentation system and generates captions for the graphics. A specimen of generated text is shown below:

(2) These two charts present information about house sales from data-set ts-1740. In the two charts, the y-axis indicates the houses. In the first chart, the left edge of the bar shows the house's selling price whereas the right edge shows the asking price.

The system consists of the following modules organised in a pipeline: Text Planner, Ordering, Aggregation, Centering, Referring Expression and Lexical Choice. The text itself is produced by the FUF/SURGE realiser (Elhadad and Robin, 1992). The CGS system was chosen because of its apparently relatively simple, modular, pipelined architecture. Despite this apparent simplicity however, it turned out that the build up of levels of information does not directly correspond with module boundaries, as illustrated in Fig. 1.

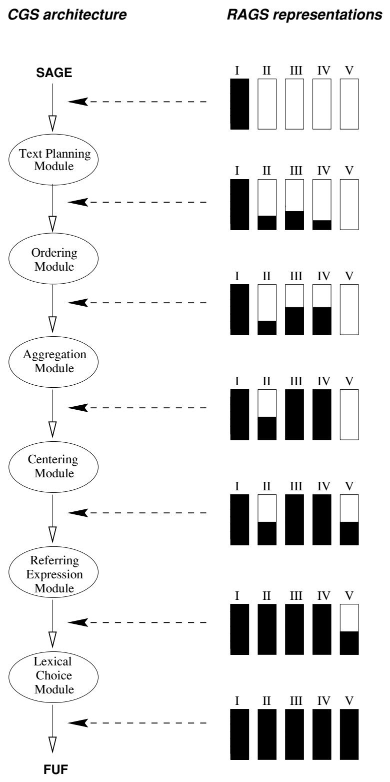

Figure 1: A RAGS view of the CGS system. I = conceptual, II = semantic, III = rhetorical, IV = document, V = syntactic.

For instance, some semantic (II) information is produced by the Text Planning module, and more work is done on this by Aggregation, but the semantic level of representation is not complete and final until the Referring Expression module has run. Also, for instance, at the point where the Ordering module has run, there are partially finished versions of three different types of representation. It is clear from this that the interfaces between the modules are more complex than could be accounted for by just referring to the individual levels of representation of RAGS. The ability to express combinations of structures and partial structures was fundamental to the reimplementation of CGS. The core modules of the CGS system were successfully reimplemented while adhering to the RAGS data model described above.

6. Conclusion

The CGS reimplementation was accomplished without any need to introduce new levels of representation or change the way the existing levels were defined, though it was necessary to make extensive use of partial and mixed representations.

It is envisaged that the architecture described above will permit much wider sharing of both modules (i.e., code that performs specific linguistic tasks) and resources (e.g., input data, lexicons, grammars etc.) within the NLG community. Although the survey of applied systems demonstrated that different systems tend to split the linguistic tasks differently, as well as scheduling them differently, the nature of the RAGS architecture allows interfaces to individual modules to be defined in generic terms. This will allow a researcher attempting to develop, for instance, an aggregation module to define the required input and output data structures in such a way that it can be slotted into any system that provides the necessary input data structures at virtually any stage of its processing, provided that the output data structures are compatible with the data structures required by subsequent modules of the system. It may appear that these requirements are nevertheless too restrictive, but it is easy to envisage a situation where an existing module could be enclosed in a "wrapper" that converts the actual input and output structures for that module into the structures required by the rest of the system. The RAGS datatype defintions allow this level of specification.

The next stage in the exploitation of the RAGS architecture is to test resource sharing more thoroughly by developing two end-to-end systems which will reuse selected modules from the CGS system, together with modules from other existing applied systems and newly developed modules. It will then be possible to test the capability of interchanging modules and datasets between the three systems.

Acknowledgements

This work is supported by the UK EPSRC under grant references GR/L77041 (Edinburgh) and GR/L77102 (Brighton).

7. References

Brachman, R. and J. Schmolze, 1985. An overview of the KL-ONE knowledge representation system. *Cognitive Science*, 9:171–216.

Cahill, Lynne, Christy Doran, Roger Evans, Chris Mellish, Daniel Paiva, Mike Reape, Donia Scott, and Neil Tipper, 1999a. Towards a Reference Architecture for Natural Language Generation Systems. Technical Report ITRI-99-14, Information Technology Research Institute (ITRI), University of Brighton. Available at http://www.itri.brighton.ac.uk/projects/rags.

Cahill, Lynne, Christy Doran, Roger Evans, Chris Mellish, Daniel Paiva, Mike Reape, Donia Scott, and Neil Tipper, 1999b. In Search of a Reference Architecture for NLG Systems. In *Proceedings of the 7th European Workshop on Natural Language Generation*. Toulouse.

Cahill, Lynne and Mike Reape, 1998. Component tasks in applied NLG systems. Technical Report ITRI-99-05, ITRI, University of Brighton. Obtainable at http://www.itri.brighton.ac.uk/projects/rags/.

Calder, Jo, Roger Evans, Chris Mellish, and Mike Reape, 1999. "Free choice" and templates: how to get both at the same time. In *"May I speak freely?" Between templates and free choice in natural language generation*, number D-99-01. Saarbrücken, pages 19–24.

Elhadad, M. and J. Robin, 1992. Controlling content realization with functional unification grammars. In *Aspects of Automated Natural Language Generation*, Lecture Notes in Artificial Intelligence, 587. Berlin: Springer-Verlag, pages 89–104.

Hirschman, Lynette and Nancy Chinchor (eds.), 1997. *MUC-7 Proceedings*. Science Applications International Corporation. See www.muc.saic.com.

Kamp, H. and U. Reyle, 1993. *From discourse to logic: Introduction to model theoretic semantics of natural language, formal logic and discourse representation theory*. Dordrecht; London: Kluwer.

Kasper, Robert T., 1989. A flexible interface for linking applications to PENMAN's sentence generator. In *Proceedings of the DARPA Workshop on Speech and Natural Language*. Philadelphia. Available from USC/Information Sciences Institute, Marina del Rey, CA.

Mann, William C. and Sandra A. Thompson, 1987. Rhetorical structure theory: A theory of text organization. Technical Report RS-87-190, USC Information Sciences Institute, Marina Del Rey, CA. Also appears in Livia Polanyi, editor, The Structure of Discourse, Ablex, Norwood, NJ, 1987.

Mellish, Chris, Roger Evans, Lynne Cahill, Christy Doran, Roger Evans, Daniel Paiva, Mike Reape, Donia Scott, and Neil Tipper, 2000. A Representation for Complex and Evolving Data Dependencies in Generation. In *Proceedings of ANLP-NAACL2000*. Seattle.

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

Nunberg, Geoffrey, 1990. *The Linguistics of Punctuation*. CSLI Lecture Notes, No. 18. Stanford: Center for the Study of Language and Information.

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

Reiter, Ehud, 1994. Has a consensus NL generation architecture appeared and is it psycholinguistically plausible? In *Proceedings of the Seventh International Workshop on Natural Language Generation*. Kennebunkport, Maine.