An Architecture for Data-to-Text Systems

Ehud Reiter
University of Aberdeen
Aberdeen, UK
ereiter@csd.abdn.ac.uk

Abstract

I present an architecture for data-to-text systems, that is NLG systems which produce texts from non-linguistic input data; this essentially extends the architecture of Reiter and Dale (2000) to systems whose input is raw data instead of AI knowledge bases. This architecture is being used in the BABYTALK project, and is based on experiences in several projects at Aberdeen; it also seems to be compatible with many data-to-text systems developed elsewhere. It consists of four stages which are organised in a pipeline: Signal Analysis, Data Interpretation, Document Planning, and Microplanning and Realisation.

1 Introduction

Data-to-text systems are Natural Language Generation (NLG) systems which generate texts from non-linguistic input data, such as sensor data and event logs. Such systems need to perform data analysis as well as linguistic processing. In this paper I present a 4-stage pipeline architecture which I believe is suitable for many data-to-text systems. It is being used in a new project at Aberdeen, BABYTALK, and is partially based on our experiences with other data-to-text systems developed at Aberdeen.

Very briefly, the four stages are

1. *Signal Analysis:* Analysing numerical and other input data, looking for patterns and trends.
2. *Data Interpretation:* Identifying more complex (and domain-specific) messages from the patterns and trends detected in Signal Analysis; also identifying causal and other relations between messages.
3. *Document Planning:* Deciding which of the above messages should be mentioned in the generated text, and creating a document and rhetorical structure around these messages.
4. *Microplanning and Realisation:* Creating an actual text which communicates the document plan.

In the rest of this paper, I describe the above stages. I then explain why I think this architecture is an appropriate one for data-to-text systems; look at how well it fits existing data-to-text systems; discuss intermediate representations between the stages; and briefly describe software resources which we are (slowly) developing to support the architecture.

2 Background

2.1 Data-to-text

Data-to-text systems generate texts from non-linguistic input data, which is typically numerical. For example, SUMTIME (Reiter et al., 2005) and FOG (Goldberg et al., 1994) generate textual weather forecasts from numerical weather prediction data; ANA (Kukich, 1983) generates textual stock market reports from numerical stock market data; and VITRA (Herzog and Wazinski, 1994) and DESCRIBER (Roy, 2002) generate natural descriptions of visual scenes. Some work has also been done on generating texts from lists of events; for example PLANDOC (McKeown et al., 1994) generates summaries based on trace files from a simulator, and Hallett and Scott (2005) describe a system which generates summaries of events in a medical record.

Perhaps the biggest difference between data-to-text systems and NLG systems whose input is a

knowledge base is that data-to-text systems must analyse and interpret their input data, as well as decide how to linguistically communicate it. While there is of course a substantial literature on data analysis, this primarily focuses on hypothesis testing and data mining; there are differences between this kind of data analysis and data analysis for the purposes of generating a textual summary (Sripada et al., 2003).

2.2 NLG Architectures

Perhaps the best-known architecture for general NLG systems is the three-stage pipeline model (Reiter and Dale, 2000), which divides NLG into document planning, microplanning, and realisation stages. This architecture focused on NLG systems whose input was a knowledge base. The architecture proposed here essentially extends the Reiter and Dale architecture by adding two new stages, Signal Analysis and Data Interpretation, which are placed before document planning in the pipeline; this extension allows the inputs to the system to be data instead of (or in addition to) knowledge.

The RAGS (Mellish et al., 2006) project conducted a survey of NLG systems, and concluded that they were architecturally quite diverse in terms of how they were modularised; it proposed a fairly abstract NLG architecture which described different representations which might be used in an NLG system, but did not commit to specific stages or modules.

The only proposal for an architecture which is specifically for data-to-text systems (which I am aware of) is Yu et al. (2004). The architecture given in that paper is quite detailed, and based on one system; the architecture presented here, in contrast, is higher-level and applies to many systems.

3 The Architecture

In this section I outline my proposed architecture, which is summarised in Figures 1 and 2. Note that some data-to-text systems only have some of the stages; Figure 2 explains when a stage is needed in a system.

I also explain how this architecture is being used in BABYTALK (Portet et al., 2007). BABYTALK is a new project at Aberdeen whose goal is to generate summaries of medical data about babies in a neonatal intensive care unit. BABYTALK systems have two types of inputs: (1) numerical sensor data about the baby, recording information such as heart rate, blood pressure and temperature; and (2) records of

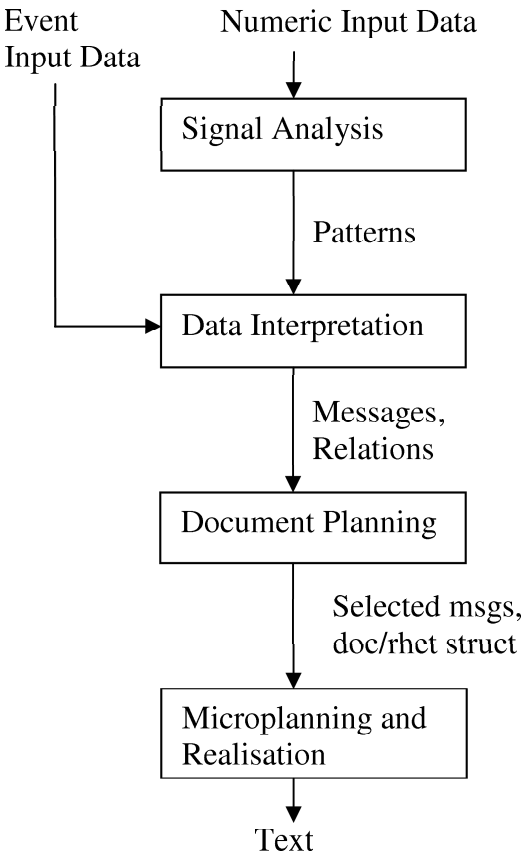

Figure 1: The Architecture

medical actions and observations, such as medication administered to the baby and blood test results. Different BABYTALK systems generate different kinds of texts from this input data; here I will focus on the BABYTALK BT45 system, which generates summaries of about 45 minutes worth of medical data which are intended to help doctors and nurses make appropriate treatment decisions (Law et al., 2005).

Figures 3 and 4 show examples of numeric and event inputs to BT45; Figure 5 shows the text that a doctor wrote to describe this data. This is an extract from a corpus text, not a generated text; it illustrates the type of text BT45 is trying to generate.

BABYTALK is a multi-person project, involving people with diverse backgrounds, ranging from signal analysis to computational linguistics. From this perspective, the architecture needs to not only be intellectually sensible, it also needs to modularise the system in a way which allows people to develop modules in their area of expertise without needing to become experts in all of the research areas involved in BABYTALK.

Stage	*input*	*output*	*needed if . . .*
Signal Analysis	numeric data	discrete patterns in data	there is numeric input data
Data Interpretation	basic patterns and events	higher-level messages, relations between messages	text communicates more than basic patterns
Document Planning	messages, relations	messages to be mentioned, document and rhetorical struct	only some messages are mentioned in text
Microplanning and Realisation	messages, structure	text	users want fluent texts

Figure 2: Stages in the Architecture

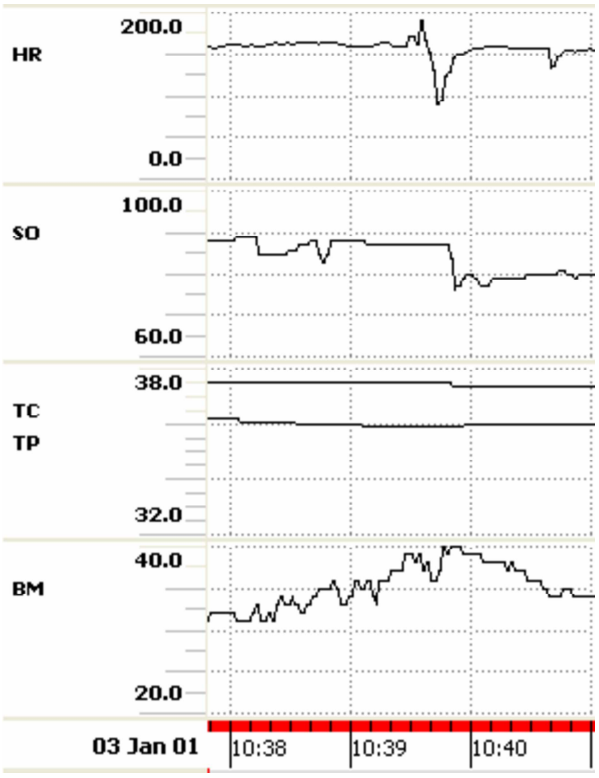

Figure 3: Example of BT45 numeric input data

time	*action*	parameters
10.38	FiO2 changed	new-value = 35%
10.39	morphine given	amt=50ug, route=IV
10.41	incubator opened	agent = doctor
10.51	baby intubated	agent = doctor

Figure 4: Example of BT45 event input data

3.1 Signal Analysis

The first stage in the architecture is to try to detect basic patterns in the numerical input data. Patterns are often organised into a taxonomy or ontology, although this is not required by the architecture. From a high-level perspective, the goal of signal analysis is to replace numerical data by a set of discrete patterns; this allows the remainder of the system to reason symbolically instead of numerically.

In preparation for re-intubation, a bolus of 50ug of morphine is given at 1039 when the FiO2 = 35%. There is a momentary bradycardia and then the mean BP increases to 40. The sats go down to 79 and take 2 mins to come back up. The toe/core temperature gap increases to 1.6 degrees.

Figure 5: Extract from BT45 corpus text

Signal analysis must also distinguish 'real' data from noise. For example, RR (Pianesi et al., 2007), which summarises the behaviour of participants in a meeting, must analyse audio signals and determine when a participant is talking (and hence contributing to the meeting), and when he or she is making non-communicative noises, such as coughing.

Input data which is already structured as discrete events, such as log files or records of medical actions, can bypass signal analysis. If all input data is of this form (as in PLANDOC (McKeown et al., 1994), for example), then the data-to-text system does not need to perform any signal analysis.

In my experience, signal analysis in data-to-text systems can usually be done with existing signal analysis algorithms. There is a substantial literature on signal analysis and pattern detection, including specialist journals such as *Pattern Recognition*. Quite sophisticated algorithms can be used in data-to-text systems; for example TREND (Boyd, 1998) used wavelet analysis. The basic challenge for the system developer is to understand the algorithms, the domain, and the way humans talk about the domain well enough to identify which algorithms are appropriate for a particular system.

BT45's signal analysis module uses three algorithms to detect patterns:

- Short-term changes in channels, such as spikes and steps, are detected using a simplified version of the algorithm proposed by Yu et al. (2007).
- Longer-term changes in the data, such as sensor values increasing or decreasing over time, are detected using linear segmentation (Keogh et al., 2001).
- *artefacts*, that is data which is corrupted or otherwise should be ignored (for example, because a sensor has fallen off the baby) are detected as described by Portet et al. (2007).

To take a concrete example, one pattern type in BT45 is SPIKE; this indicates that the values in a sensor channel have temporarily increased or decreased, but then reverted to their previous value. The SPIKE class has several parameters, including channel, direction, time, and extremeValue. One particular spike which is detected in the Figure 3 data has parameters channel=HR (Heart Rate), direction=Down, time=1039, extremeValue=90.

3.2 Data Interpretation

The second stage of the architecture analyses the patterns detected by signal analysis, and also any events which are directly specified in the input data, and infers more complex (and domain-specific) *messages* about the data set from these patterns and events. It may also infer relationships (such as causality) between patterns, events, and messages. From a high-level perspective, the goal of data interpretation is to map basic patterns and events into the messages and relationships that humans use when discussing this domain.

In some applications, such as marine weather forecasts (Reiter et al., 2005), humans only refer to basic patterns when discussing the domain, and do not refer to relationships. Systems in such domains do not need to perform data interpretation. But in other domains, humans also communicate about higher-level events or patterns. For example, SCUBATEXT (Sripada and Gao, 2007), which generates safety-oriented summaries of scuba dives from depth profiles and other dive computer data, looks for patterns that suggest potentially dangerous activities in a dive. For instance, one risky activity is a 'sawtooth' dive where the diver descends, ascends, and then descends again. SCUBATEXT creates sawtooth messages wherever it sees an increasing trend in depth, followed by a decreasing trend in depth, followed by another increasing trend; this is an example of data interpretation.

BT45 performs three basic activities in data interpretation:

- *Create messages:* This is similar to SCUBATEXT. For example, a BRADYCARDIA (heart rate is temporarily too low) message is created from downward spikes in heart rate which go below 100.
- *Decide how important events are:* BT45 assigns an importance to every message; this is needed by the document planner. For example, the importance of a BRADYCARDIA depends primarily on how long it lasts for (5 seconds is unimportant, 5 minutes is very important), and secondarily on how low heart rate goes.
- *Detect relationships between events:* BT45 looks for three kinds of relationships between messages: causality (for example, blood oxygen increases *because* the nurse increased oxygen levels in the ventilator); part of a procedure (for example, giving morphine is *part of* a reintubation procedure); and other (for example, an increase in blood oxygen is *associated with* a decrease in blood CO2).

Currently, data interpretation in BT45 is primarily done by production rules written in JESS[1], which are based on knowledge-acquisition activities conducted with experienced doctors. We may in the future use KBTA (Shahar, 1997) for some of this reasoning.

To take a concrete example in BT45, the SPIKE event mentioned at the end of the Signal Analysis section is interpreted in Data Interpretation as a BRADYCARDIA. It has moderate importance (15 on a scale of 0 to 100), and has an *associated with* link to a drop in blood oxygen saturation (SO) which happens at about the same time.

3.3 Document Planning

The third stage of the architecture decides which events to mention in the text, and also on the text's rhetorical and document (e.g., paragraph break) structure. This is the same as the Document Planning stage in the architecture of Reiter and Dale

[1] http://herzberg.ca.sandia.gov/jess/

(2000). From a high-level perspective, signal analysis and data interpretation can produce a large number of messages, patterns, and events (often hundreds or thousands), but texts usually are limited to only describing a small number of messages (the actual number depends on the genre, but often is between 5 and 25 events). The Document Planner must decide which messages are actually communicated in the text; this decision is based on the domain and genre. It must also try to communicate how the messages mentioned in the text relate to each other; this can partially be done using rhetorical and document structure.

In some data-to-text applications, such as pollen forecasts (Turner et al., 2006), all the input data is communicated to the user, so document planning is a trivial task. But this is unusual, usually some selection is needed. Indeed, Law et al. (2005), who found that doctors made better decisions from textual summaries than from graphical displays, speculate that this happened precisely because the texts only communicated the most relevant information.

Document planning is perhaps the least-understood aspect of data-text systems, and indeed of NLG in general; in fact Evans et al. (2002) argue that document planning should not be considered part of NLG. Of course document planning still needs to be done in data-to-text systems regardless of whether it is regarded as an 'NLG' task.

Yu et al. (2007) treat document planning in data-to-text systems as the task of finding 'interesting patterns', and use two mechanisms to do this: rules acquired from domain experts, and novelty (how often a pattern has been seen before). They use simple schemas to specify document structure.

Hallett and Scott (2005), who summarise medical events, use a different approach. The input to their document planner is a graph of events and relations between events. Their system works by partitioning this graph into inter-connected clusters of events; dropping small clusters (unless they contain information which domain knowledge says must be reported); and then mapping clusters onto a specific *report spine* for the target type of report, which specifies which events are central for this type of report. Non-spinal events are linked to spinal events using rhetorical relations which are based on event relations; they may be dropped if they are too far away from a spinal event in the graph.

Document planning in BT45 is currently done in a similar fashion to Hallett and Scott, except that many decisions take into account the importance of messages (calculated by Data Interpretation). BT45's document planner decides on paragraph boundaries, but not sentence boundaries (sentence boundaries are chosen by the microplanner/realiser, as part of aggregation).

For example, the BRADYCARDIA message mentioned in the last section is included in a cluster of messages which happen at about the same time (these essentially are the messages mentioned in the text of Figure 5). This cluster is overall the second-most important message cluster, so it is included in the text; and the BRADYCARDIA message is moderately important, so it is included in the text describing the cluster. The cluster is realised at the document level as a single paragraph (this decision is based on its size). A simple SEQUENCE rhetorical relation is used to relate the BRADYCARDIA message to other messages, because it only has generic *associated with* links to other messages in its cluster.

3.4 Microplanning and Realisation

The fourth stage of the architecture generates actual texts based on the content and structure chosen by Document Planning. It corresponds to the two stages of Microplanning and Realisation in Reiter and Dale (2000). From a high-level perspective, microplanning and realisation must decide how to actually express in language the concepts and structure selected by earlier stages.

It is possible to perform microplanning and realisation using simple templates. Whether this is acceptable depends on what is appropriate for users. For example, IGRAPH (Ferres et al., 2006) uses templates to generate (spoken) descriptions of graphs for visually-impaired users. IGRAPH's texts look clumsy and repetitive on paper, and probably would not be acceptable as textual summaries for people who can read. But visually impaired users listening to a speech synthesiser have different requirements, and in some cases may prefer simple repetitive texts.

Microplanning and realisation in data-to-text are fairly similar to microplanning and realisation in other NLG systems, so I will not describe them in detail here. There are a few differences in emphasis, though.

One difference is that most (although not all) data-to-text systems produce simple language from a syntactic perspective, because their users prefer this. Syntactic realisation in such systems is relatively straightforward.

Another difference is that data-to-text systems must deal with differences in how readers interpret words that communicate data. For example, Reiter et al. (2005) found considerable differences in how different readers and writers of weather forecasts interpreted time phrases such as *by late evening* and *later*; and Roy (2002) found inconsistencies in the way that different people mapped colour terms such as *pink* into numerical RGB colour specifications. There is no clear solution to this problem, it is a major open research issue.

Finally, data-to-text system may need to communicate uncertainty about about the reliability of the input data or the system's analysis. Again this is an open research issue. There is a substantial literature in linguistics and psychology on communicating uncertainty, but I am not aware of attempts to incorporate these findings into data-to-text systems.

The current version of BT45 lexicalises the BRADYCARDIA message mentioned above as *There is a bradycardia to 90*; ie, it uses a there-is sentence and mentions the extreme value but not the duration. The corpus text shown in Figure 5, in contrast, uses the phrase *There is a momentary bradycardia*. '*momentary*' is a qualitative description of the duration of the bradycardia; determining when it can be used is not easy, in part because different people use it in different ways.

4 Justification for Architecture

The architecture presented here divides the data-to-text generation process into 4 stages. Obviously we could form an architecture with more stages by splitting the stages presented here into smaller stages (for example, split Microplanning and Realisation into two separate stages); we could also form an architecture with fewer stages by combining stages (for example, combine Signal Analysis and Data Interpretation into one stage). The stages of the architecture described here are based on the following criteria.

Types of processing performed and knowledge required. Signal analysis uses numerical pattern-recognition algorithms and is to some degree domain independent; data interpretation uses symbolic reasoning and relies on domain knowledge; document planning uses symbolic reasoning and relies on domain communication knowledge (Kittredge et al., 1991); and microplanning and realisation are based on linguistic reasoning and are partially domain-independent. This consideration is especially important in projects such as BABYTALK which involve developers from diverse backgrounds (as mentioned above).

Intermediate representations. Modules need clear API's which will remain fairly stable even if a module is re-implemented using different techniques. One of the primary reasons for combining Microplanning and Realisation into one stage is that it is difficult to define a generic API between a Microplanner and a Realiser, because the inputs expected by a Realiser depend on the syntactic formalism it is based on.

Perhaps the hardest decision to justify is the separation of Data Interpretation and Document Planning, as these both involve domain-dependent symbolic AI reasoning, and they tend to either both be present or both be absent in individual systems. My main reason for separating these is that they emerge from two very different research communities; Data Interpretation has been studies by researchers in knowledge-based (expert) systems, while Document Planning has been studied by researchers in the NLG community.

5 Applicability

Perhaps not surprisingly, the architecture described here fits many data-to-text systems developed at Aberdeen by the author and his colleagues, including SUMTIME (Reiter et al., 2005) (generates marine weather forecasts for offshore oil rigs); pollen forecast generator (Turner et al., 2006); SUMTIME-TURBINE (Yu et al., 2007) (generates summaries of sensor data from a gas turbine); and SCUBATEXT (Sripada and Gao, 2007) (generates safety-oriented summaries of scuba dives). It also seems to fit many data-to-text systems developed elsewhere.

For example, ANA (Kukich, 1983), which generates stock market summaries, is described as having 4 modules in a pipeline: fact generator, message generator, discourse organiser, and text generator. These correspond to the signal analysis, data interpretation, document planning, and microplanning/realisation modules described here.

Another example is PLANDOC (McKeown et al., 1994), which generates summaries of what happened in a simulation. It is described as having 5 modules: Message Generator, Ontologiser, Content Planner, Lexicaliser, and Surface Generator. The Message Generator is essentially an interface to the simulation package; the Ontologiser performs data interpretation; the Content Planner does document

planning; and the Lexicaliser and Surface Generator perform microplanning and realisation (respectively). As mentioned above, PLANDOC does not need to perform signal analysis, because its input is already in the form of discrete events.

However, the architecture described here does not fit all data-to-text systems. For example DESCRIBER (Roy, 2002) takes a more integrated approach based on machine learning.

6 Representations

The above description of the architecture is of course very high-level. To make it more concrete, we need to specify intermediate representations (APIs) between the modules. As Mellish et al. (2006) point out, it is difficult to do this for NLG as a whole, because the field is very diverse. However, we believe this problem is more tractable (although still hard) in the more limited area of data-to-text.

In particular, we use the following intermediate representations in BT45, and believe these could be used in other systems as well:

- output of *Signal Analysis* is a set of patterns. Patterns are represented as objects in a Protégé[2] ontology, which includes types such as SPIKE and INCREASING TREND. Objects have parameters (feature values), such as the channel they occurred in and the time they occurred at.
- output of *Data Interpretation* is a set of messages which are again represented as objects in a Protégé ontology; all messages have an importance parameter. *Data Interpretation* also produces a set of relations between messages; these are represented as (RelationType, Message1, Message2) triples.
- output of *Document Planning* is a document plan. This is represented as a tree. Nodes in the tree can specify messages; they can also be annotated with document structure level (e.g., paragraph). Parent-child links can be annotated with rhetorical relations (which are finer-grained than the domain relations produced by Data Interpretation; e.g., VOLITIONAL-CAUSE instead of CAUSES). In RAGS (Mellish et al., 2006) terminology, a document plan is a tree which represents both document and rhetorical structure, and whose nodes can specify conceptual structures (ie, messages).
- output of *Microplanning and Realisation* is a text, which may include HTML mark-ups.

7 Software

An architecture should also be supported by software resources which help developers create systems based on the architecture. We are trying to create such resources for building data-to-text systems. In particular

- *Signal Analysis:* The TSNET (Hunter, 2006) system, which has been developed at Aberdeen over a number of years, includes many signal analysis algorithms for time series data. We are adapting TSNET so that it can be used to perform Signal Analysis in BABYTALK, and believe it could be used in other data-to-text systems as well.
- *Microplanning and Realisation:* We are developing a Java library called SIMPLENLG[3] to perform these tasks; again this is based on several previous projects. SIMPLENLG currently performs morphological processing, realisation of simple syntactic structures, and simple lexicalisation; we hope to add support for simple microplanning soon.

We hope over the next few years to expand the above, and in particular also provide software tools for Data Interpretation and Document Planning.

8 Conclusion

There is growing interest in data-to-text systems. Such systems are both practically useful (all fielded NLG systems that I am aware of are data-to-text systems) and scientifically interesting (in part because they help us study how language relates to the non-linguistic world). I have tried in this paper to outline the kinds of processing that data-to-text systems must perform, and show how this processing can fit into an architecture which is an extension of (not a replacement of) existing NLG architectures.

Acknowledgements

Many thanks to the reviewers and to my colleagues at Aberdeen for their comments on this paper. All opinions expressed here are of course mine alone. BABYTALK is supported by grant EP/D049520/1 from the UK Engineering and Physical Sciences Research Council (EPSRC).

[2] http://protege.stanford.edu

[3] http://www.csd.abdn.ac.uk/~ereiter/simplenlg/

References

Sarah Boyd. 1998. TREND: a system for generating intelligent descriptions of time-series data. In *Proceedings of the IEEE International Conference on Intelligent Processing Systems (ICIPS-1998)*.

Roger Evans, Paul Piwek, and Lynne Cahill. 2002. What is NLG? In *Proceedings of the Second International Conference on Natural Language Generation*, pages 144–151.

Leo Ferres, Avi Parush, Shelley Roberts, and Gitte Lindgaard. 2006. Helping people with visual impairments gain access to graphical information through natural language: The iGraph system. In *Proceedings of the 10th International Conference on Computers Helping People with Special Needs*.

Eli Goldberg, Norbert Driedger, and Richard Kittredge. 1994. Using natural-language processing to produce weather forecasts. *IEEE Expert*, 9(2):45–53.

Catalina Hallett and Donia Scott. 2005. Structural variation in generated health reports. In *Third International Workshop on Paraphrasing (IWP2005)*.

Gerd Herzog and Peter Wazinski. 1994. VIsual TRAnslator: Linking perceptions and natural language descriptions. *Artificial Intelligence Review*, 8(2-3):175–187.

Jim Hunter. 2006. TSNet a distributed architecture for time series analysis. In *Proceedings of IDAMAP 2006*, pages 85–92.

Eamonn Keogh, Selina Chu, David Hart, and Michael Pazzani. 2001. An online algorithm for segmenting time series. In *Proceedings of IEEE International Conference on Data Mining*, pages 289–296.

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

Karen Kukich. 1983. Design and implementation of a knowledge-based report generator. In *Proceedings of ACL-1983*, pages 145–150.

Anna Law, Yvonne Freer, Jim Hunter, Robert Logie, Neil McIntosh, and John Quinn. 2005. Generating textual summaries of graphical time series data to support medical decision making in the neonatal intensive care unit. *Journal of Clinical Monitoring and Computing*, 19:183–194.

Kathleen McKeown, Karen Kukich, and James Shaw. 1994. Practical issues in automatic document generation. In *Proceedings of ANLP-1994*, pages 7–14.

Chris Mellish, Donia Scott, Lynn Cahill, Daniel Paiva, Roger Evans, and Mike Reape. 2006. A reference architecture for natural language generation systems. *Natural Language Engineering*, 12:1–34.

Fabio Pianesi, Massimo Zancanaro, Elena Not, Chiara Leonardi, Vera Falcon, and Bruno Lepri. 2007. Multimodal support to group dynamics. *Personal and Ubiquitous Computing*, 11. In press.

François Portet, Ehud Reiter, Jim Hunter, and Somayajulu Sripada. 2007. Automatic generation of textual summaries from neonatal intensive care data. In *Proceedings of AIME 2007*. Forthcoming.

Ehud Reiter and Robert Dale. 2000. *Building Natural Language Generation Systems*. Cambridge University Press.

Ehud Reiter, Somayajulu Sripada, Jim Hunter, and Jin Yu. 2005. Choosing words in computer-generated weather forecasts. *Artificial Intelligence*, 167:137–169.

Deb Roy. 2002. Learning visually grounded words and syntax for a scene description task. *Computer Speech and Language*, 16:353–385.

Yuval Shahar. 1997. A framework for knowledge-based temporal abstraction. *Artificial Intelligence*, 90:79–133.

Somayajulu Sripada and Feng Gao. 2007. Summarizing dive computer data. In *Proceedings of the Workshop on Multimodal Output Generation (MOG-2007)*, pages 149–157.

Somayajulu Sripada, Ehud Reiter, Jim Hunter, and Jin Yu. 2003. Generating English summaries of time series data using the Gricean maxims. In *Proceedings of KDD-2003*, pages 187–196.

Ross Turner, Somayajulu Sripada, Ehud Reiter, and Ian Davy. 2006. Generating spatio-temporal descriptions in pollen forecasts. In *Proceedings of EACL-2006 Poster Session*, pages 163–166.

Jin Yu, Ehud Reiter, Jim Hunter, and Somayajulu Sripada. 2004. A new architecture for summarising time series data. In *Proceedings of INLG-04 Poster Session*, pages 47–50.

Jin Yu, Ehud Reiter, Jim Hunter, and Chris Mellish. 2007. Choosing the content of textual summaries of large time-series data sets. *Natural Language Engineering*, 13:25–49.