Are Reference Architectures Integration Tools or Descriptive Aids?

Ehud Reiter
Dept of Computing Science, University of Aberdeen, Aberdeen AB24 3UE, UK
ereiter@csd.abdn.ac.uk

Abstract

Reference architectures are most commonly thought of as tools which enable modules developed at different institutions to be easily integrated. However, reference architectures can also be used for other purposes. In particular, they can make it easier to describe the architecture of systems, by allowing developers to just describe how their architecture differs from the reference one. This is one of the goals of the architecture described in a forthcoming book on NLG (Reiter and Dale, 1999).

1 Introduction

What is the purpose of a reference architecture? In particular, is it supposed to be a integration tool which makes it easier to combine modules developed in different institutions, in an analogous fashion to the way SQL and ODBC make it easy to integrate database clients and servers? Or is it supposed to be a reference point for describing systems, in an analogous fashion to the way certain research papers become reference points for other research papers (e.g., *our algorithm is similar to Shieber et al. (1990) except for...*)? Or is it perhaps supposed to serve some other purpose(s)?

In this paper I discuss the integration and descriptive goals in general terms, and the constraints that they impose on a reference architecture. I suggest that the integration goal, which is probably currently the most prominent one in the community, may be difficult to achieve in its full-fledged form at this point in time; for this reason it may be worth putting more emphasis on the descriptive goal. I then give an overview of a reference architecture described in detail in a forthcoming book on NLG (Reiter and Dale, 1999), which is partially intended to serve as a descriptive aid.

2 Types of Standards

A reference architecture is of course a kind of standard. The British Standards Institution (BSI) defines several kinds of standards (BSI, 1997), the most relevant of which for NLP are VOCABULARY, SPECIFICATION, and METHOD. A vocabulary is a glossary, that is a set of textual definitions of domain terms. A specification states functionality by defining the interface of an object; in software terms, the API (Application Programming Interface) of the modules being standardised. It should also include a means for verifying compliance to the standard; for software, this often consists of a test set. A method specifies exactly how an action should be performed; in an NLP context, for example, a method could specify how the accuracy of a parser is evaluated.

As far as I am aware, there has been relatively little discussion in the NLP community about standards for evaluation methods, which is a pity, as I think this is an important area for standardisation. There has been little published on standard vocabularies as such, but there have been proposals to standardise modularisations which have something of this feel. For example, Bordegoni et al. (1997) define a standard decomposition of intelligent multimedia presentation systems into modules, but specify the function of modules with text glosses, not by input and output specifications; arguably this is closer to a standard vocabulary than a standard specification.

Most of the interest in NLP standards seems to revolve around what the BSI calls 'specifications', that is a definition of functionality which includes a specification of the interface; the BSI would also expect such a standard to include a means of verifying conformance to the standard. For software standards, this typically means specifying both an API and a test set which a module developer can use to determine whether his/her module meets the standard.

I do not know if there are any existing NLP standards which the BSI would regard as a true specification. Perhaps the closest thing to such a standard which I personally have encountered is the TIPSTER standard (which is used by GATE), as this includes both an API and guidelines on checking conformance to the API. I believe the original TIPSTER plan was to base the conformance-checking process on a formal test suite but in fact this was not done and instead the checking process was based on "manual comparison of the system's design with the design docu-

ment"[1]. It would be interesting to see if the BSI or similar organisations elsewhere (such as ANSI in the US) regarded this as an acceptable verification procedure for a standard.

Being very speculative (I was not in any way personally involved in TIPSTER), I wonder if the lack of a formal test suite was due to rapid changes in technology and user's expectations and requirements. It is much easier and cheaper to change an API than it is to change a test suite that verifies the API; and hence it may not be worth constructing a formal test suite for a fast-changing and research-dominated field such as NLP.

It also is worth noting that TIPSTER was intended to be a standard not for NLP in general but rather for a specific type of NLP application (information extraction) as used by a specific group of user organisations (US government agencies). It would be more difficult to establish a standard for a general area of NLP unconstrained by application type and user group.

Of course, the fact that an API without a formal test suite may not be considered to be a full standard by the BSI does not mean it is not useful. It depends on what the purpose of the standard is; for some purposes, just an API or perhaps even part of an API could indeed be useful.

3 Possible Purposes

3.1 Integration

If we look at existing reference architectures in natural-language processing as a whole, perhaps the most commonly stated goal is enabling modules and resources developed at different institutions to be easily integrated; this is emphasised in both GATE[2] and RAGS[3], for example.

This is surely a worthwhile goal, especially for applied system-building projects. After all, standardised APIs and languages for databases enable applications to easily connect to a variety of different databases; standard protocols and APIs make the Internet and Web possible; and indeed outside of the computing world, language itself is dependent on a set of standards with regard to spelling, grammar, and so forth.

Such standards are not easy to create, however. In the database world, SQL may be a standard, but every vendor adds their own extensions to the basic SQL defined in ISO 9075, partially in order to 'lock' customers into one particular database, and therefore decrease interchangeability. The standard protocols behind the Internet, such as IP and SMTP, took decades to be accepted throughout the world, despite the overwhelming need for standards in networks, since the whole point of networks is to enable as many people as possible to communicate. And standards in language took centuries to emerge, and were in many cases backed up by financial pressure (for example, telling people that they couldn't get a job unless they spoke real English) or even physical coercion (for example, caning students who spoke in dialects). Standardisation is neither easy not quick, except in a situation where a dominant entity can dictate a standard to the rest of the world; and even this doesn't always work. For example, IBM tried to get the computing world to accept the EBCDIC character set in the 1960s and 1970s, when it was the dominant IT company; but this attempt was not successful, and in the long term ASCII and its descendants (such as the ISO 8859 family) were accepted as standards instead.

Another point is that if a goal of an architecture is true 'plug-and-play module interchangeability' (as used to be stated in GATE's Overview Web page, although the most recent version of this page no longer mentions this), then it probably must be a specification as defined by the BSI. That is, it must include a detailed API and also a mechanism (such as a comprehensive test suite) for verifying conformance to the standard. As mentioned above, I am not personally aware of any existing NLP reference architecture which is a standard by this criteria. Furthermore, the architecture which comes closest (TIPSTER) is intended to apply to a specific type of application as used by a specific group of user organisations; it is not intended to be a general standard for all NLP for all types of users.

A less ambitious integration aim is to enable groups at different institutions to develop modules which can be integrated with a specific core module or system. For instance, the various APIs associated with Microsoft Windows 98 allow third-party developers to create applications which integrate with Windows and use its facilities. In such cases, it may be sufficient just to specify an API (and not a test suite). After all, in this context developers do not need to guarantee that their module will integrate with *any* other standard-conforming module, they just need to check that their module integrates with a specific core module whose detailed behaviour can be determined by direct experimentation. GATE perhaps falls into this category; from my limited acquaintance with the system it seems to be more a set of modules with well-documented API's for adding functionality than a standard in the BSI sense, that is a set of criteria which a standard-conforming entity is expected to satisfy.

Given these facts and the fact that NLP is primarily a research area which is still rapidly evolving and changing, it may be premature to try to develop standards which include sufficient details and conformance-testing procedures to make 'plug and play' a reality, at least for NLP as a whole. Perhaps it is possible to establish such standards for particular types of applications as used by particular types of users; but the TIPSTER experience is not encouraging here, especially considering the fact that TIPSTER was backed by organisations with considerable amounts of both money and influence.

However, it may well be appropriate to develop core

[1] http://www.nist.gov/itl/div894/894.02/related_projects/tipster/arch.htm

[2] http://www.dcs.shef.ac.uk/research/groups/nlp/gate

[3] http://www.itri.brighton.ac.uk/projects/rags

modules to which other modules can be attached, and to document these core modules and their APIs well enough so that it is reasonably straightforward for outside developers to develop compatible modules. It remains an open question whether it is possible to develop generic core modules which can be used by many types of NLP applications, or whether core modules are application-specific. For example, I believe that currently most GATE users are in the information extraction area, but perhaps this will change in the future.

3.2 Descriptive Aid

Integration is not the only possible goal of a reference architecture. There are many other goals as well, which are less ambitious but may be easier to achieve. In particular, a reference architecture may help people describe systems, by giving them a reference point to base their descriptions on.

At least in the Natural Language Generation (NLG) community, it is relatively unusual for researchers and developers to describe in detailed terms the architecture of the systems they build; this has certainly been my experience, and it also has been the experience of others who have tried to survey NLG systems. This is no doubt largely because such descriptions require a considerable amount of space, certainly more than is available in a conference paper, and often more than is available in a journal paper, given that the focus of the journal paper is often on a topic other than architecture. This is unfortunate, because it is difficult to compare systems in a detailed way without knowing architectural details such as the functionality of each system module, what input data structures it uses, and what output data structures it produces.

A reference architecture could make it easier to describe functionalities and data structures in several ways. First of all, it could define a standard vocabulary (using BSI terminology) and perhaps a standard modularisation, as done for intelligent multimedia presentation systems by Bordegoni et al. (1997). For example, in the NLG community the terms *sentence planner* and *microplanner* basically mean the same thing, and the continued used of both terms confuses newcomers to the field but achieves little else; a standard vocabulary might help resolve this. Also, sometimes different researchers use the same term to mean different things; for example in NLG the term *aggregation* is used by some people to cover purely syntactic phenomena and others to cover conceptual/semantic phenomena as well as syntactic ones. Again, a standard vocabulary or glossary could help ensure that terms mean the same thing in papers written by different authors.

A more ambitious goal would be to provide a base architecture for system descriptions. This might enable paper writers to quickly communicate details about their system by describing just how it differs from the base architecture. For example, *the output of my microplanner is SPL as defined in reference-architecture XXX, except that I've added the following features ...*, or *my document planner has the same functionality as defined in reference-architecture XXX, except that it also ...*. Of course, this only works if the architecture of the system being described is fairly similar to the reference architecture.

Another possibility, although this is perhaps controversial, is that a reference architecture could help researchers compare results by reducing 'accidental' differences between systems. Someone interested in, say, generating referring expressions needs to make assumptions about what the rest of the NLG system is going to be doing; and if different referring-expression researchers make different assumptions, it may be difficult to compare their results. If the different assumptions are the result of different beliefs about how NLG systems should be built, fair enough. However, if they are the result of different random choices between equally plausible alternatives, then perhaps a reference architecture could help reduce such differences, by giving researchers some guidance as to the 'reference' way of doing things that are not their primary research focus.

What criteria should a reference architecture satisfy in order to help people describe their systems in detail? I cannot answer this question on my own, it needs to be discussed by a wider audience (such as the attendees at the workshop), but some criteria that come to my mind are:

1. It must agree as much as possible with the way current systems are built; otherwise it is not useful as a comparison point. If there is a lot of diversity, it may be best to define several reference architectures, perhaps targeted at different applications and user communities; although on the other hand multiple reference architecture may be less useful in reducing accidental differences between systems.

2. It must be described in some detail; otherwise it is not useful as a tool for helping people describe details of their systems. In particular, it probably should go beyond vocabulary and text glosses, and give some descriptions of APIs and data structures; but these do not have to be as detailed as descriptions intended to support integration.

3. It must be well-explained in an accessible source; otherwise it will not be adopted.

4 A Reference Architecture Intended to be a Descriptive Aid

Reiter and Dale (1999) present a reference architecture (although it is not called this), which I will in this paper refer to as the RDNLG architecture. It is primarily intended to be an explanatory and descriptive aid. In other words, it is intended to achieve two goals:

- help newcomers to NLG understand how the different pieces of NLG fit together; and
- serve as a descriptive aid, as described above.

I will not discuss the first goal in detail, but basically any description of NLG which is intended to be accessible to people new to the field (as RDNLG is) must not only describe individual NLG tasks, but must also provide an architectural framework which shows readers how the individual tasks interact. Criteria (1)-(3) above are probably important to achieving this goal; another important criteria for this goal is that the architecture should be simple and easy to understand.

Space prohibits a detailed description of RDNLG in this paper; interested readers should read Reiter and Dale (1999). Very briefly, though, the architecture divides the NLG process into a three-stage pipeline:

Document planning: deciding on the information content and rhetorical structure of a document.

Microplanning: deciding what linguistic resources (lexemes, referring expressions, syntactic constructs) should be used to communicate the information and structure; and deciding how content is divided up in terms of sentences and paragraphs.

Realisation: generating actual sentences that use the above-mentioned linguistic resources in a grammatically correct way.

The architecture gives textual descriptions of the tasks described by each module, and also specifies a set of data structures which are passed between modules.

From the perspective of standards and reference architectures, the textual descriptions of the pipeline, modules, and tasks probably are best considered to be a type of reference vocabulary. The key component of RDNLG from the system description perspective is probably the data structure descriptions, which are not detailed enough to allow integration, but may perhaps be useful as base points in some system descriptions. Four inter-module data structures are defined: DOCUMENT PLAN, MESSAGE, TEXT SPECIFICATION, and PHRASE SPECIFICATION. A document plan is the output of the document planning module (and hence the input of the microplanning module). It is a tree, whose leaf nodes are messages; these specify actual conceptual or semantic content. In other words, messages specify content, and the rest of the document plan indicates how messages are structured in the document. A text specification is the output of microplanning (and hence the input to realisation). In an analogous manner to a document plan, it is a tree, whose leaf nodes are phrase specifications, which specify the structure of actual phrases. Typically a phrase specification defines a single sentence, but it may define a sub-sentential phrase where such an entity is an orthographic unit, such as a section heading or document title.

Figure 1 (taken from Reiter and Dale (1999)) describes the data structures used for document plans in RDNLG; it is in a generic object-oriented notation loosely based on Java. An example of an RDNLG document plan is given in Figure 2; this is in an attribute-value notation, and does not show the details of the message data structures which are the leaf nodes of the document plan. A graphical version of this document plan is shown in Figure 3. One possible text that might be produced from this example document plan is

> The month was slightly warmer than average with almost exactly the average rainfall, but rainfall for the year is still very depleted. Heavy rain fell on the 27th and 28th.

The RDNLG document plan data structure is not innovative, but it is not intended to be innovative; it is intended to be as similar as possible to the data structures used in existing NLG applications, which by definition means it must be similar to existing practice. It is also very simple, and would probably need to be extended when used to describe a real system. For this reason it is of very limited use as an API to support system integration; but we hope that it will be useful as an aid to system description. For example, the document plans used in the STOP system (Reiter et al., 1999) can be described as follows:

> STOP's document plans are similar to the ones defined in RDNLG, except that several additional types of DPs are defined to support document entities such as sections, subsections, and itemised lists. Also, DPs can be annotated with an importance attribute, which is used by a revision module to determine what to delete if the document is too large. Discourse relations are represented by cue phrases, not abstract RST-like relations.

Thus, a meaningful description of the document plans used in STOP can be communicated in just a few sentences, by referring to the RDNLG architecture.

Of course, the fact that the RDNLG architecture can be used to describe systems built by myself or Robert Dale is not surprising; the real test is whether other people can use the architecture in this way. Since Reiter and Dale (1999) has not yet been formally published, we do not yet know if RDNLG will be so used by other researchers.

In terms of the criteria described in Section 3.2:

1. We believe RDNLG is consistent at least in general terms with the way many, perhaps most, applied NLG systems are built (Reiter, 1994; Paiva, 1998). For the purposes of this paper I'll use Paiva's definition of 'applied', namely that the input comes from an external source and is not hand-crafted by the NLG researchers. Our architecture is less consistent with 'non-applied' NLG systems, but this is largely because there is much more variation in the architecture of these systems.

```
class DocumentPlan {
    children: Constituents;
}

class Constituents { // Constituents of a document-plan
}

class NSConstituents extends Constituents{
    // A nucleus and its rhetorical satellites
    nucleus: DocumentPlan or Message;
    satellites: List of SatelliteSpec;
}

class SatelliteSpec {
    // A satellite and its rhetorical relation to the nucleus
    relation: DiscourseRelation;
    satellite: DocumentPlan or Message;
}

class ConstituentSet extends Constituents{
    // A set of constituents, without a nucleus
    relation: DiscourseRelation; // Must be a non-nucleus relation
    constituents: List of DocumentPlan or Message;
}

class DPDocument extends DocumentPlan { // DP for complete document
    title: Message or PhraseSpec;
}

class DPParagraph extends DocumentPlan { // DP for paragraph
    // No extra information needed
}
```

Figure 1: The RDNLG document plan representation

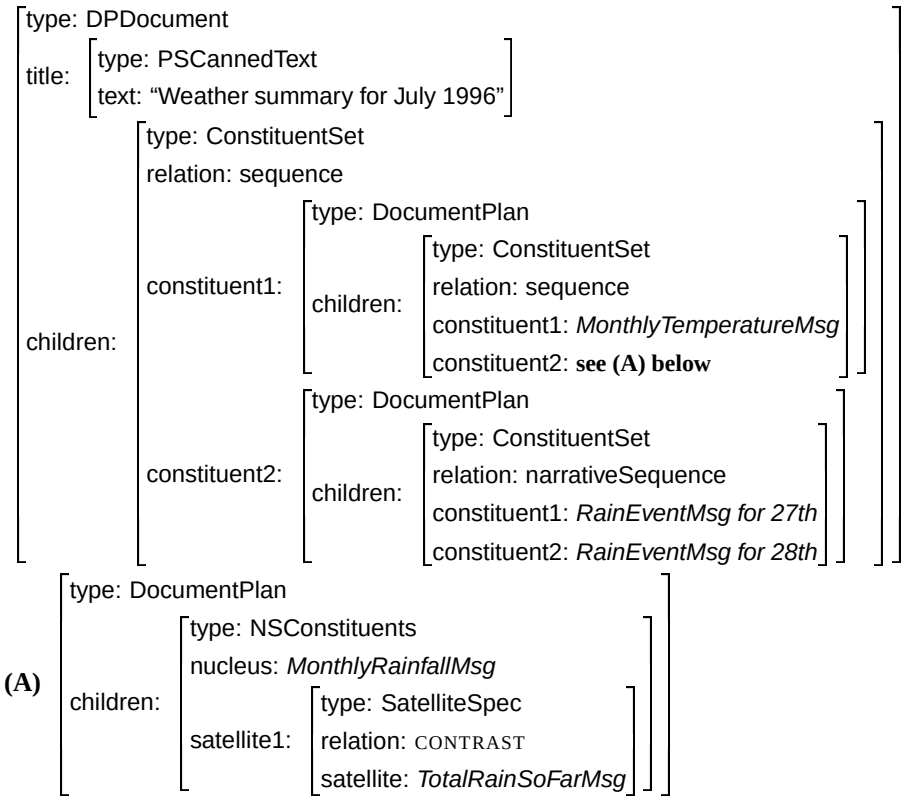

Figure 2: An example RDNLG document plan

2. The textual glosses in RDNLG are probably as detailed as such things can meaningfully be. As can be seen from the above examples, the data structure definitions are fairly simple, but they are not vacuous. RDNLG data structures are, however, certainly much simpler than the data structures used in many real systems. This is partially to keep RDNLG simple and easy to understand for newcomers to the field; perhaps an architecture intended purely to support descriptions should specify more complex data structures.

3. We hope Reiter and Dale (1999) is well-written and will become a standard reference work which is widely available. On the other hand, arguably the best way to publish a proposed standard in 1999 is as a Web document (instantaneously available world-wide without charge), instead of as a book.

In short, RDNLG is probably not ideal as a reference architecture to support system descriptions, largely because it is also supposed to the satisfy the additional goal of explaining how NLG systems are built to newcomers to the field. Nevertheless, we hope that despite its flaws it will still prove useful to at least some people in writing system descriptions.

One issue mentioned in Section 3.2 was variation; is it better to have one reference architecture or several? RDNLG is one architecture, but it does specify four possible variations for the phrase specification data structure: orthographic strings, canned text, abstract syntactic structures, and lexicalised case frames. Combinations of these are also allowed. The reason for having this variation is that there are considerable differences in how applied NLG systems represent phrase specifications, and we decided that it was best to try to describe several points in the spectrum of possibilities, instead of just picking one point (as we did with the other data structures). Of course, the converse side of this decision is that it makes RDNLG less useful in terms of reducing accidental inconsistencies, so this is a trade-off.

We present no other alternatives as part of the RDNLG architecture itself (although some alternatives are discussed in the book as part of an Other Approaches section). This again is partially motivated by the desire to keep the architecture simple and understandable for newcomers; perhaps an architecture intended solely as a descriptive aid should have more alternatives.

5 Conclusion

There are many possible objectives of a 'reference architecture', and what criteria a 'good' reference architecture should satisfy depends largely on what its objectives are. Perhaps the most discussed objective to date is defining standard APIs to enable modules developed

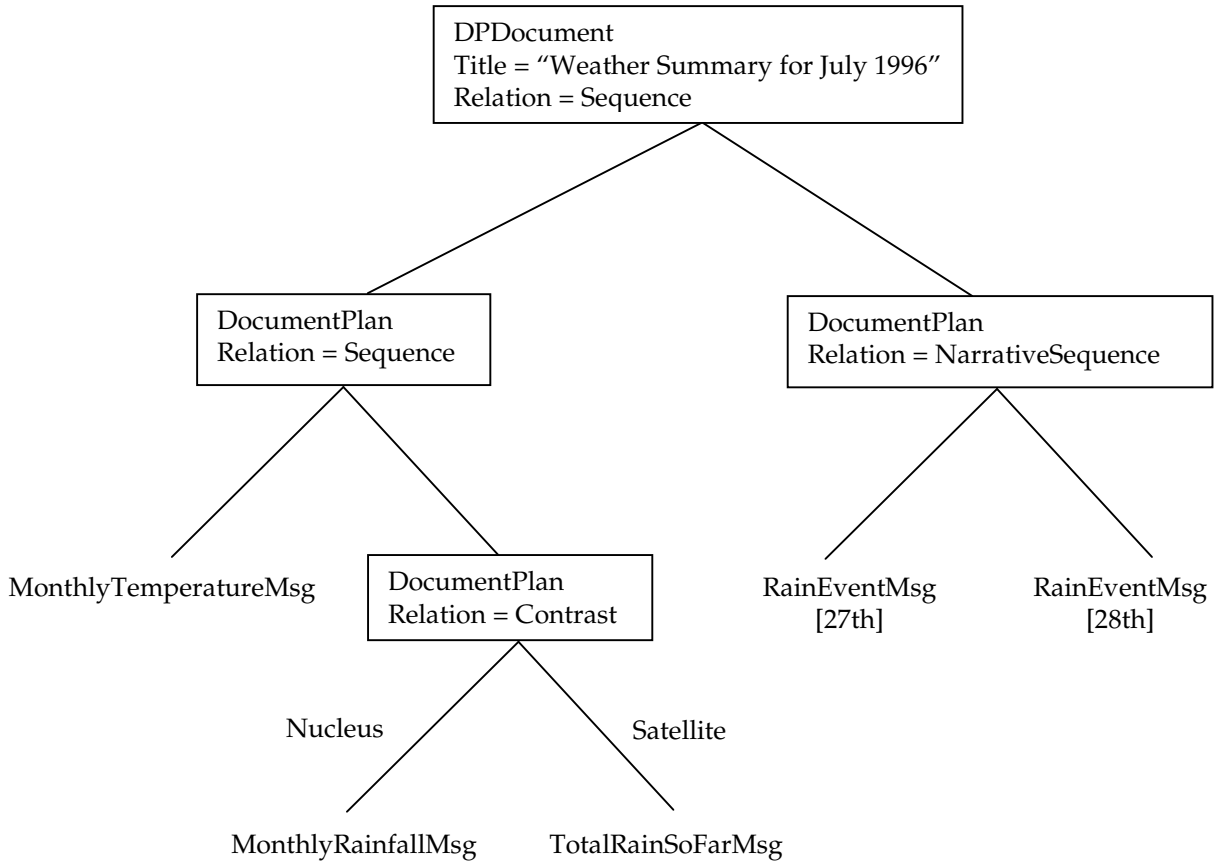

Figure 3: A graphical depiction of an RDNLG document plan

at different institutions to be easily connected; but this type of standard is difficult to establish, especially in a field such as NLP which is rapidly evolving and dominated by researchers. In the long term, I hope standard APIs supported by conformance-checking test suites are established for NLP, but in the immediate future, I think that perhaps the field is not yet ready for such standards; although it certainly would be very valuable to have standard core modules with well-documented APIs, as developed by GATE and similar projects elsewhere.

What is perhaps possible in the shorter term is reference architectures which are intended to help people describe systems, and to make comparing systems easier by reducing 'accidental' inconsistencies. These are some of the goals behind the architecture described in a new book on NLG (Reiter and Dale, 1999). No doubt this architecture does not achieve these goals as well as it might; but even if the implementation is flawed, I believe its goals are correct, and I hope to see further attempts at reference architectures which are intended to serve as descriptive aids.

Acknowledgements

Obviously much of this paper is based on joint work with Robert Dale, but I emphasise that all opinions expressed are purely my own.

References

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

BSI. BSI 0: A standard for standards. Technical report, British Standards Institution, 1997. Available at http://www.bsi.org.uk/bsi/standards/bs0.html.

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

Ehud Reiter. Has a consensus NL Generation architecture appeared, and is it psycholinguistically plausible? In *Proceedings of the Seventh International Workshop on Natural Language Generation (INLGW-1994)*, pages 163–170, 1994.

Ehud Reiter and Robert Dale. *Building Natural Language Generation Systems*. Cambridge University Press, 1999. In press.

Ehud Reiter, Roma Robertson, and Liesl Osman. Types of knowledge required to personalise smoking cessation letters. In *Proceedings of AIMDM-1999*, 1999. Forthcoming.

Stuart Shieber, Gertjan van Noord, Fernando Pereira, and Robert Moore. Semantic-head-driven generation. *Computational Linguistics*, 16:30–42, 1990.