Fast, Scalable and Reliable Generation of Controlled Natural Language

David Hardcastle
Faculty of Maths, Computing
and Technology
The Open University
Milton Keynes, UK
d.w.hardcastle@open.ac.uk

Richard Power
Faculty of Maths, Computing
and Technology
The Open University
Milton Keynes, UK
r.power@open.ac.uk

Abstract

In this paper we describe a natural language generation system which takes as its input a set of assertions encoded as a semantic graph and outputs a data structure connecting the semantic graph to a text which expresses those assertions, encoded as a TAG syntactic tree. The scope of the system is restricted to controlled natural language, and this allows the generator to work within a tightly restricted domain of locality. We can exploit this feature of the system to ensure fast and efficient generation, and also to make the generator reliable by providing a rapid algorithm which can exhaustively test at compile time the completeness of the linguistic resources with respect to the range of potential meanings. The system can be exported for deployment with a minimal build of the semantic and linguistic resources that is verified to ensure that no runtime errors will result from missing resources. The framework is targeted at using natural language generation technology to build semantic web applications where machine-readable information can be automatically expressed in natural language on demand.

1 Introduction

This paper describes a fast, reliable and scalable framework for developing applications supporting tactical generation – by which we mean applications which take as their input some semantic structure that has already been organised at a high level, and choose the syntactic structures and words required to express it. The framework takes as input a semantic graph representing a set of assertions in Description Logic (DL) (Baader et al., 2003) and transforms it into a tree which encodes the grammar rules, syntactic subcategorisations, orderings and lexical anchors required to construct a textual representation of the input data. The resulting text is *conceptually aligned*, by which we mean that each component of the text structure (such as words, clauses or sentences, for example) is linked back to the mediating structure from which the text was generated, and from there back to vertices and edges in the semantic graph received as input. The target context for the framework is the construction of semantic web (Berners-Lee et al., 2001) resources using Natural Language Generation (NLG) technology which extends the notion of semantic alignment developed in the WYSIWYM system (Power and Scott, 1998; Power et al., 2003). In this context the text is ephemeral and is generated on demand, while the document content is fully machine-readable, supporting tasks such as automated consistency checking, inferencing and semantic search/query. Since the text is fully linked to the underlying semantic representation it supports a rich user interface encompassing fast and reliable semantic search, inline syntax or anaphora highlighting, knowledge editing, and so on. Finally, the text could be generated in many different natural languages making the information content more widely available. We envisage the technology supporting a range of different use cases such as information feeds, technical instructions, medical orders or short, factual reports.

For such a system to be of practical value in an enterprise system the NLG component must sup-

port standard aspects of software engineering quality such as modularity, reliability, speed and scalability. The design of the framework relies on two key simplifying assumptions and these limit the range of information which can be represented and the fluency of the text used to express it. Specifically, the information is limited by the expressivity of DL – for example only limited quantification is possible – and the surface text is restricted to controlled natural language (Hartley and Paris, 2001). The upside of this trade-off is that the domain of locality is very restricted. This means that there is minimal search during generation and so the algorithm is fast and scalable. It also enables us to design the generator so that it is predictable and can therefore be statically tested for *completeness*, a notion which we define in Section 3.

Our aim in this paper is to show how the simplifying assumptions behind the design bring considerable engineering benefits. We discuss the theoretical background to our approach (Sections 2 and 3) and then present the implementation details, focusing on the features of the design that support speed and scalability (Section 4) and reliability (Section 5), followed by an overview of the architectural considerations (Section 6). Finally we present the results of tests evaluating the system's performance (Section 7).

2 Implementation Theory

The generation algorithm has its roots in the WYSIWYM system, which was originally developed as a way of defining the input for multilingual NLG in DRAFTER (Paris et al., 1995), one of a series of projects in the KPML/Penman tradition (Bateman et al., 1989). The system uses the standard semantic representation employed in DL and the Semantic Web: a Terminology Box, or *Tbox*, defining the concepts and their interrelations and an Assertion Box, or *Abox*, representing the information content that forms the input (Baader et al., 2003). An Abox is a set of assertions defining relations between instances of the types defined in the Tbox. It can be depicted by a connected graph (Figure 1) in which vertices represent entities and edges represent relations, and is represented in the input to the system by a set of RDF subject-predicate-argument triples (Lassila and Swick, 1998), with one-place predications assigning types and two-place predications asserting relationships. Assuming that the entities are being mentioned for the first time, we might express this Abox fragment in English by the sentence 'a woman lost her bag'[1]. This sentence can be aligned with the Abox by associating spans of the text with the entities expressed by the Abox, as follows:

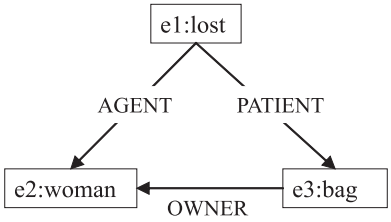

Figure 1: Sample Abox

Span	**Entity**	**Context**
a woman lost her bag	e_1	ROOT
a woman	e_2	AGENT
her bag	e_3	PATIENT
her	e_2	OWNER

Note that the same entity may be expressed in multiple contexts (denoted by the incoming arcs in the semantic graph). The relationships between the entities are represented by syntactic dependencies between the spans of the text. For instance, AGENT(e_1,e_2) is realised by the clause-subject relation between 'a woman lost her bag' and its subspan 'a woman'. This direct linking of semantic and syntactic dependencies has of course been noted many times, for instance in Meaning-Text Theory (Candito and Kahane, 1998).

The structure of the spans of text can be represented by a reconfiguration of the original Abox as an ordered tree, which we will henceforth call an *Atree*. Figure 2 shows an Atree that fits the example Abox. Note that since this is a tree, the vertex with two incoming edges (e_2) has to be repeated, and there are two spans referring to the woman.

[1]The system is able to generate a referring expression, 'her', for the second reference to the woman since it knows that the entity has already been mentioned in the text. This information is available because the Atree, see Figure 2, is an ordered structure.

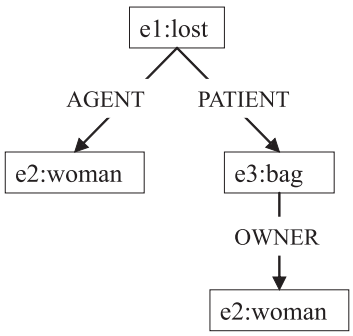

Figure 2: Sample Atree

This Atree is constructed using a set of bindings which map instances of concepts from the Tbox in a given context onto a subcategorisation frame, grammar rule call and list of lexical anchors. As each vertex of the Atree is constructed it is labelled with the grammar rule and lexical anchors and linked back to the vertex of the Abox which it expresses. Our current model uses the Tree Adjoining Grammar (TAG) formalism, see Joshi (1987), and the Atree acts as a stand-in *derivation tree* from which the derived syntactic tree can be computed. Each vertex of the derived tree is linked back to the vertex of the Atree from which it was generated, and so the output from the system is a composite data structure comprising the Tbox, Abox, Atree and derived tree with a chain of references connecting each span of the surface text via the Atree to the underlying semantic representation. A detailed exposition of the process through which the Atree and derivation tree are constructed is presented in a separate Technical Report (Hardcastle and Power, 2008).

2.1 Simplifying assumptions

The design of the generator relies on two simplifying assumptions. The key assumption for this paper is that the text should adhere strictly to a controlled language, so that a given local semantic configuration is always realised by the same linguistic pattern. The cost is that the text is likely to be repetitive and may be awkward at times; however the trade-off is that the domain of locality is tightly restricted and this yields important benefits in speed, scalability, reliability and verifiability that make the system suitable for deployment in an enterprise environment.

We also assume that the strategic element of the NLG process, comprising content selection and document structuring, has occurred prior to our system receiving its input. Our framework is focused specifically on tactical generation – rendering the semantic representation of the selected content as text.

3 Completeness

We can verify that the generator is *complete*, in the sense that we can guarantee that it will produce a derivation tree for any Abox valid under the Tbox. We present the details of the verification algorithm below, in Section 5. Note that we assume that the system is equipped with the requisite morphological and orthographic rules to realise the resulting derivation tree. We also note that we cannot verify that the generator is *consistent*, by which we mean that it should produce different texts for different Aboxes, nor that the syntactic frames and lexical anchors mapped to the concepts in the Tbox are appropriate. Checking the system for consistency remains an open research question.

4 Speed and Scalability

In many NLG systems the choice of syntactic structure and lexical arguments depends on a large number of interdependent variables. This means that the process of realizing the semantic input involves exploring a large search space, often with some backtracking. In contrast, the algorithm described in this paper is monotonic and involves minimal search. The system begins at the root of the Abox and uses a set of mappings to construct the Atree one node at a time. Because the same local semantic context is always expressed in the same way the choice of syntactic structure and lexical arguments can always be made on the basis of a single mapping. Over the course of this section we demonstrate this with a simple example using resources that were automatically inferred to construct the test harness described in Section 8, which could be used to construct the following simple sentence:

> The nurse reminded the doctor that the patient was allergic to aspirin.

The Abox representing this sentence is rooted in an instance of a Tbox concept representing an event

in which one person reminds another of a fact. Figure 3 shows the attributes defined for this Tbox concept, *938B*, namely an *actor*, *actee* and *target*. The range of *actor* and *actee* is any concept in the Tbox subsumed by the *person* concept, the range of *target* is any *fact*. There will therefore be three out-going arcs from the root Abox node, labelled with attributes *actor*, *actee* and *target*, pointing respectively to nodes representing the nurse, doctor and the fact about the patient's allergy described in the sample sentence above. Some of these nodes will themselves have out-going arcs expressing their own attributes, such as the subsidiary details of the fact about the allergy.

938B
actor(person)
actee(person)
target(fact)

Figure 3: A Sample Tbox Node

To realize the first node in the Abox the system searches for mappings for concept *938B*. The controlled language assumption allows the system to search with a restricted domain of locality, and so the only variables affecting the choice of frame will be: the Tbox concept represented by the Abox node to be realized (*938B* in this case), the syntactic context (there is none at this stage since we are processing the root node, so the system will default to *clause* context), the incoming arc (there is none at this stage so no constraint is applied), the out-going arcs (the three attributes specified), and whether or not the instance has already been expressed (in this case it has not)[2]. The search parameters are used to locate a mapping such as the one depicted in Figure 4 below.

```
<frame concept="938B"
       role="any"
       subcat="CatClause-33"
       bindings="SUB,D_OB,CL_COM">
  <gr key="TnxOVnx1s2">
       <anchor lemma="remind" pos="verb"/>
  </gr>
</frame>
```

Figure 4: A Sample Mapping

This mapping tells the system which subcategorisation frame to use, which grammar rule to associate with it, which lexical anchors to pass as arguments to the grammar rule and also how to order the subsidiary arguments of the subcategorisation frame (the *bindings* attribute in the *frame* element). The subcategorisation frame itself (shown in Figure 5) is highly reusable as it only defines a coarse-grained syntactic type and a list of arguments, each of which consists of a free text label (such as *SUB* indicating the subject syntactic dependency) and a coarse-grained syntactic constraint such as *clause*, *nominal* or *modifier*. In this example the first attribute

CatClause-33
type= CLAUSE
args= SUB/NOMINAL,
D_OB/NOMINAL,
CL_COM/CLAUSE

Figure 5: Sample Subcategorisation Frame

of the *938B* node, namely the *actor*, is mapped to the *SUB* (subject) argument, so it will become the first child of the Atree node representing the *remind* event. The *nominal* syntactic constraint will be carried forward as the syntactic context for the *nurse* node of the Abox, constraining the choice of mapping that the system can make to realise it. So, each mapping enforces an ordering on the out-going arcs of the Abox which is used to order the Atree and provides a syntactic context which is used to constrain

[2]The last of these variables is used to determine whether or not a referring expression (an anaphoric reference to an entity which has already been mentioned) is required. Because the Atree is ordered and is constructed in order, the system always knows whether an instance is being mentioned for the first time. We currently render subsequent mentions by pruning all of the out-going arcs from the Abox node, which also allows us to manage cycles in the semantic graph. Since the system knows which nodes in the semantic graph have already been mentioned it would also be possible to configure an external call to a GRE system (Dale, 1989) - an application which infers the content of a referring expression given the current semantic context.

the mapping search for each child. The process of locating and imposing mappings cascades through the Abox from the root with minimal search and no backtracking. If multiple mappings are defined for a given context the first one is always chosen. If no mapping is located then the system fails.

While the Atree is constructed, it is annotated with the grammar rules and lexical anchors listed in each mapping, allowing it to serve as a stand-in TAG derivation tree from which a derived syntactic tree can be constructed by instantiating and connecting the elementary trees specified by each grammar rule. Further details of this process are given in a Technical Report (Hardcastle and Power, 2008).

So while the controlled language assumption that we should always express the same local semantic context in the same way limits expressivity, it also limits algorithmic choice and prevents backtracking, which means that the system can generate rapidly and scale linearly. In the following section we show how we can prove at compile time that no Abox can be constructed which will result in a local semantic context not accounted for in the mappings.

5 Reliability

In a real-world context the Tbox will evolve as the underlying domain model is extended and enhanced. As a result, some of the mappings described above will become defunct and in some instances a required mapping will not be present. If the system encounters an Abox which requires a mapping that is not present it will not backtrack but will fail, making the system fragile. To address this problem we need to be able to run a static test in a short period of time to determine if any mappings are unused or missing.

Although the set of possible Abox graphs is an infinite set, the tight domain of locality means that there is a finite set of parameters which could be passed to the generator for any given Tbox. As described in the previous section the choice of mapping is based only on the following information: the concept being realised, the syntactic context, the number of attributes expressed by the concept, the attribute used to select it, and whether or not this Abox instance is being mentioned for the first time. Given a starting TBox node and syntactic context the system can crawl the TBox recursively using the subcategorisation frames returned from each parameter set to derive a new list of parameter sets to be tested. Each of these must be tested both as a first and as a subsequent mention. The result is an algorithm which proves the application's *completeness* (as defined in Section 3) with respect to a particular domain (represented by the Tbox); if the test succeeds then it guarantees that the mappings defined by the system can transform any Abox that is valid under the given domain into an Atree annotated with the information required to produce a derived syntactic tree.

As above, the proving algorithm starts with a root concept in the Tbox and an initial syntactic context and uses these as the starting parameter set to find the first mapping. Once a mapping is located it explores each of the attributes of the root concept using the syntactic context to which the attribute is bound by the mapping. Since there is no Abox it constructs a list of parameter sets to check using every concept in the range of the attribute.

For example, during the verification process the prover will encounter the mapping shown above in Figure 4 for the *remind* concept *938B* in a clausal context. The concept has three attributes: an *actor*, an *actee* and a *target*. The first of these has as its range all of the subconcepts of *person* defined in the Tbox, and this introduces a new sub-problem. The first attribute is bound to the *SUB* argument of the subcategorisation frame used in the mapping, in Figure 4, by the *bindings* element, and this argument of the subcategorisation frame imposes a nominal constraint. So the fact that concept *938B* might need to be expressed using this mapping means that any subconcept of *person* might need to be expressed in a nominal syntactic context with semantic role *actor*, and so the prover now checks each subconcept with these parameters. If none of the subconcepts of *person* define any attributes and a mapping is found for each then no new sub-problems are introduced and so this branch of the search bottoms out.

The prover then returns to *938B* and processes the *actee* and *target* attributes. The *target* attribute is bound to the *CL_COM* argument of the subcategorisation frame, and so the new sub-problem involves checking that every subconcept of *fact* can be expressed as a clause with semantic role *target*. In

the ontology the subconcepts of *fact* include *events*, each of which define a number of attributes, and so this sub-problem branches out into many new sub-problems before it bottoms out. One such *event* will be the concept *938B*, but since the mapping that we have already encountered (Figure 4) is encoded for any semantic role and the system has already processed it the prover can break out and does not fall into an infinite loop. This checking process continues recursively until the search space is exhausted, with each parameter set tested being cached to reduce the size of the search space.

6 Relaxing the Simplifying Constraints

The simplifying assumptions described in Section 2.1 deliver benefits in terms of performance and reliability; however, they limit the expressivity of the language and reduce the scope of what can be expressed. We can relax some of the constraints imposed by the simplifying assumptions and still have a performant and reliable system, although proving completeness becomes more complex and some localised exponential complexity is introduced into the generation algorithm. In this section we explore the ways in which relaxing the constraints to allow quantification or underspecification impact on the system.

The simplest scenario, which adheres to our simplifying constraints, is that each node in the Abox expresses exactly one of each of the attributes defined by the Tbox concept which it instantiates. So, using the *remind* example above, every instance of *remind* must express an *actor*, an *actee* and a *fact*. In practice the Tbox may allow an attribute not to be expressed, to be expressed many times or to be expressed but not specified. We handle the first case by allowing arguments in the subcategorisation frames to be marked as optional; for example, a verb frame may include an optional *adverb* slot. These optional arguments increase the number of tests that must be performed; if a frame has n optional slots then the system will need to perform 2^n checks to verify it, and will have to consider 2^n mapping combinations during generation. This introduces localised exponentiation into both the generation and the verification algorithm, although it will only lead to tractability problems if the number of optional slots on any single frame is too high, since the exponent is *only* applied to each frame and not across the whole search space.

Where an attribute may remain unspecified the system can be configured to respond in two different ways. First, unspecified attributes can be included in the text using the concept that represents the root of the range. For example, if an event occurs at a time which is not specified then the system can use the concept that represents the root of the range (e.g. *timePeriod* perhaps) and render it accordingly ("at some time"). Alternatively the system can prune all underspecified instances from the Abox before the Atree is generated. Attributes which may not be expressed (for either reason) must be flagged in the TBox so that the proving algorithm knows to match them to optional arguments in the subcategorisation frames. This is implemented with a flag on each attribute definition indicating whether its presence in the Abox is optional.

Relaxing the constraints also impacts on our ability to verify the grammar rules which are associated with each mapping. If we use TAG, then we can easily verify that the syntactic type of the root of the elementary tree defined by each mapping matches the syntactic type of the subcategorisation frame to which it is bound. However, if a mapping can be accessed via an optional slot in another subcategorisation frame, then it must be bound to an *auxiliary* tree, that is to an elementary tree which can be added to the derived tree through adjunction, since any derived tree with open substitution sites will be grammatically incomplete. For the system to support this behaviour each mapping must declare not just the concept which it realises but also the role (Tbox attribute) which it fulfils, so that both the prover can determine whether it may be left out, and this increases the combinatorial complexity of the algorithm.

7 Architecture

The design of the generator ensures that it can generate rapidly and that it can be verified at compile time. A further feature is that it is implemented with a component-based modular architecture. For NLP applications it is particularly important that individual components can be independently verified

and reused, because linguistic resources are time-consuming and expensive to build and curate. Furthermore, because the mappings from concepts to subcategorisation frames, grammar rules and lexical anchors are defined in a single file, the task of building and maintaining the mappings is easier to learn and easier to manage. It is also easier to bootstrap the mappings through resource mining, as we did ourselves in the construction of the test data set discussed in Section 8.

The framework manages the graph and tree structures and the transformations between them, and it defines the API for the domain and language specific resources that will be required by the application. It also defines the API of the linguistic resource manager, leaving it to the application layer to provide an appropriate implementer using dependency injection (Fowler, 2004). Rather than define a core 'interlingual' feature structure that attempts to capture all of the lexical features used by the grammar, the framework provides a genericised interface to the linguistic resource manager. This means that grammars for different natural languages can use different feature structures to define the lexical anchors used by the application and to support tasks that are the responsibility of the grammar, such as unification or morphological inflection. For example, all verbs in French should have a flag indicating whether *avoir* or *être* is used as a modal auxiliary for the *passé composé*, but this flag need not be present for other languages. The Tbox, the subcategorisation frames and the mappings between them are all defined as data sources and can be reused across applications as appropriate. Although they are not defined in code they can still be verified at compile time by the prover discussed in the previous section, and this allows the system to be flexible and modular without introducing the risk of runtime failures caused by faulty mapping data.

7.1 Export

A further feature of the system which arises from the proving algorithm is that it supports export behaviour. In an enterprise context we want to be able to reuse linguistic resource components, such as a lexicon, a grammar, a morphological generator and so on, across many different applications. These resources are large and complex and for a given application much of the data may not be required. Because the proving algorithm is able to compile a comprehensive list of the concepts, grammatical relations, subcategorisation frames and lexical anchors that will be required to realise any Abox, given a starting concept and syntactic context, the system can cut the Tbox, lexicon, grammar, subcategorisation frame store and related resources to export a build for deployment, while guaranteeing that the deployed application will never fail because of a missing resource. This is of particular value if we want to reuse large-scale, generic, curated resources for a small domain and deploy where bandwidth is an issue – for example where language generation is required in a client-heavy internet-based or mobile application.

8 Testing and Results

We unit-tested the mechanics of the framework, such as the graph and tree managers. We then built a proof-of-concept application with a small ontology representing the domain of patient treatment narratives and handcrafted the subcategorisation frames, lexical resources and TAG grammars for English, French, Spanish and Italian. We used this application to verify the independence of the framework, domain and linguistic resources and verified that we could develop linguistic resources offline and plug them into the application effectively. The application also served as a test harness to test the adaptibility of the framework to render the same semantic context in different syntactic structures depending on the target natural language. For example, we included the examination of a body part belonging to a person in the domain, and this was expressed through a Saxon genitive in English but a prepositional phrase (with the subsidiary NPs in the reverse order) in the other languages.

To test our assumptions about efficiency and scalability we inferred a larger Tbox, subcategorisation frames and mappings using a pre-existing data set of verb frames for English encoded using the COMLEX subcategorisation frame inventory (Grishman et al., 1994). The linguistic resources for the application comprised a generative TAG grammar based on X-TAG (Doran et al., 1994) which we wrote our-

selves, the CUV+ lexicon[3], and a pre-existing morphological generator for English (Hardcastle, 2007).

To test the performance of the generation process we used a set of randomly-generated Aboxes derived from the Tbox to produce texts of increasing size. For the purposes of testing we defined the size of an Abox as the total number of nodes and edges in the graph, which is the number of RDF triples required to represent it. Table 1 shows the size of the output text in sentences, the time taken to generate it in milliseconds, averaged over 5 runs, and the ratio of the time taken to the size of the output which shows linear scaling[4].

Size	Timing	Timing/Size
31	2	0.065
280	10	0.036
2,800	59	0.021
28,000	479	0.017

Table 1: The time, in milliseconds, taken to generate Aboxes of increasing size and the ratio of time taken to the size of the output.

To test the performance of the proving algorithm we ran the algorithm on a set of Tboxes of differing sizes. The smallest Tbox in Table 2 is the handcrafted proof-of-concept Tbox, the largest is the inferred Tbox described above, and the intermediate ones were pruned from the large, inferred Tbox at random cut points. The size of each Tbox is the total number of attribute-concept pairs which it defines. The table shows the time taken to run the prover from the root node of the Tbox with no starting syntactic context and the ratio of time taken to size, which shows linear scaling.

We tested the mechanics of the implementation of the prover through unit testing, and we tested the the design with a test suite of sample data. We performed white box tests by removing individual bindings from a set of mappings which we judged to be complete for the small handcrafted Tbox, and checked to ensure that each was highlighted by the prover. We performed black box tests by using a set of inferred mappings, judged by the prover to be complete, to generate from a large number of randomly structured Aboxes, drawn from our large inferred Tbox, and checked that the generation process never failed.

[3] A publicly available lexicon for English available from the Oxford Text Archive

[4] In fact scaling is slightly sub-linear for this test and the test of the proving algorithm. In both cases that is because of caching within the framework to improve performance.

Size	Timing	Timing/Size
125	10	0.08
86,766	432	0.005
2,054,020	8,217	0.004
9,267,444	21,526	0.002

Table 2: The time, in milliseconds, taken to prove reliability for Tboxes of increasing size and the ratio of time taken to size.

We chose not to undertake a formal evaluation over and above the unit and sampling tests, because the accuracy of the prover is a function of the restricted domain of locality imposed by the system and of the recursive algorithm which depends on it. Instead we show that the prover is accurate by describing the parameters that guide search in generation and explaining why they can be exhaustively tested (see Section 5).

9 Conclusion

In this paper we presented a tactical generator which exploits a simplifying assumption that the output text will be restricted to controlled natural language to enforce a restricted domain of locality on search in generation. As a result, the generation process is fast and scales linearly, and furthermore the system is reliable, since we are able to perform a compile-time check of the data sources which drive the assignment of syntactic subcategorisations to the expression of each node in the input semantic graph.

The generator is most appropriate for applications which need to present small chunks of structured data as text on demand and in high volume. For example, information feeds such as local weather forecasts, traffic information, and tourist information or technical information that must be both machine-readable (for example because it is safety critical and requires consistency checking) and also human-readable (for example for an operator to make use of it) such as machine operator instructions, business process/protocol descriptions and medical orders.

References

F. Baader, D. Calvanese, D. L. Mcguinness, D. Nardi, and P. F. Patel-Schneider, editors. 2003. *The Description Logic Handbook : Theory, Implementation and Applications*. Cambridge University Press.

J. Bateman, R. Kasper, J. Moore, and R. Whitney. 1989. A general organization of knowledge for natural language processing: The Penman Upper Model. Technical report, Information Sciences Institute, Marina del Rey, California.

T. Berners-Lee, J. Hendler, and O. Lassila. 2001. The Semantic Web. *Scientific American*, 284(5):34–43.

M. Candito and S. Kahane. 1998. Can the TAG derivation tree represent a semantic graph? An answer in the light of the Meaning-Text Theory. In *Proceedings of the Fourth Workshop on Tree-Adjoining Grammars and Related Frameworks*, Philadephia, USA.

R. Dale. 1989. Cooking up referring expressions. In *Proceedings of the 27th Annual Meeting of the Association for Computational Linguistics*, pages 68 – 75, Vancouver, Canada.

C. Doran, D. Egedia, B. Hockey, B. Srinivas, and M. Zaidel. 1994. XTAG system - a wide coverage grammar for English. In *Proceedings of the 15th International Conference on Computational Linguistics*, pages 922–928, Kyoto, Japan.

M. Fowler. 2004. Inversion of control containers and the dependency injection pattern `http://www.martinfowler.com/articles/injection.html`.

R. Grishman, C. McLeod, and A. Myers. 1994. Comlex Syntax: Building a Computational Lexicon. In *Proceedings of the The 15th International Conference on Computational Linguistics*, pages 268–272, Kyoto, Japan.

D. Hardcastle and R. Power. 2008. Generating Conceptually Aligned Texts. Technical Report 2008/06, The Open University, Milton Keynes, UK.

D. Hardcastle. 2007. Riddle posed by computer (6): The Computer Generation of Cryptic Crossword Clues. PhD thesis, University of London.

A. Hartley and C. Paris. 2001. Translation, controlled languages, generation. In E. Steiner and C. Yallop, editors, *Exploring Translation and Multilingual Text production*, pages 307–325. Mouton de Gruyter.

A. Joshi. 1987. The relevance of tree adjoining grammar to generation. In G. Kempen, editor, *Natural Language Generation: New Directions in Artificial Intelligence, Psychology, and Linguistics*. Kluwer.

O. Lassila and R. Swick. 1998. Resource Description Framework (RDF) model and syntax specification. W3C Working Draft WD-rdf-syntax-19981008.

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

R. Power and D. Scott. 1998. Multilingual authoring using feedback texts. In *Proceedings of the 17th International Conference on Computational Linguistics and 36th Annual Meeting of the Association for Computational Linguistics*, pages 1053–1059, Montreal, Canada.

R. Power, D. Scott, and N. Bouayad-Agha. 2003. Document structure. *Computational Linguistics*, 29(4):211–260.