Published in: The proceedings of the NLDB'95, First International Workshop on the Applications of Natural Language to Data Bases, pp 135-149, Versailles, France, June 28-29, 1995.

Aggregation, Formal Specification and Natural Language Generation

Hercules Dalianis

Department of Computer and Systems Sciences
The Royal Institute of Technology and
Stockholm University
Electrum 230, S-164 40 Kista
Sweden
email: hercules@dsv.su.se

Abstract

In this paper we show how to use the so-called aggregation technique to remove redundancies in the fact base of a formal specification. Redundancies are always present in formal languages to make them explicit and non-ambiguous The objective is to generate non-redundant natural language text. The aggregation rules preserves the meaning. The formal specification is expressed in the Delphi language, which is a sort of conceptuel modelling language. A prototype of the natural language aggregation generation system is implemented in Prolog.

Résumé

Dans cet article nous montrons comment utiliser la technique dite d'agrégation pour extraire la redondance de la base de fait d'une spécification formelle. Des redondances sont toujours présentes dans les langages formels afin de les rendre explicites et non ambigus. L'objective est de générer un texte en langage naturel non redondant. Les règles d'agrégation préservent le sens. La spécifcation formelle est exprimée dans le langage Delphi, qui est une sorte de langage de modélisation conceptuelle. Un prototype du système de génération de langage naturel par agrégation est implémenté en Prolog.

1. Introduction

Natural language generation for validation of formal specifications is a well-known concept in the formal specification community, [Swart82,Rolla92,Dalia92], however before generating natural language (NL) from a formal specification it is necessary to carry out a number of tasks on the formal specification.

First is content selection carried out, i.e. *what to say* , from the abundant knowledge base, then planning and organization of the information content. The text plan decides the order in which the sentences should be generated to make the text coherent, then follows the sentence planning and finally the surface generation where it has to be decided *how to say it,* i.e. the realization of the syntactic structures and lexical choices, but though all this is carried out certain information may be redundant or duplicated. The nature of formal languages is that they need to be very explicit while humans try to organize their facts conceptually and much information is implicit.

However there is a technique called aggregation in natural language generation developed in [Dalia93] and [Dalia95b] which can be used for removing redundancies in the fact bases of formal specifications. The aggregation process takes place before the surface generation. The aggregation is part of the sentence planning.

Aggregation is the process of removing redundant information in a text without losing any information. People do aggregation all the time to make natural language expressions shorter, non-redundant and easy read. Aggregation rules, which apply before realization to semantic objects, cluster common objects together and makes the text easier to read and consequently easier to understand.

The following example illustrates the effect of aggregation.

Before aggregation:

Leo has a powerbook. Chrysanne is a professor. Hercules is a student.
Richard is a student. Hercules has a powerbook. Richard has a book.
Chrysanne has a pen. Leo is a student.

After aggregation =>

Hercules, Leo and Richard are students.
Hercules and Leo have powerbooks
Richard has a book.
Chrysanne is a professor and has a pen.

The aggregated natural language text looks nicer and is easier to read. This aggregation is just one of several possibilities to aggregate the text.

2. Previous research and systems.

There are not many requirements engineering tools which uses natural language, examples are: AMADEUS, [Black87] and ALECSI, [Cauv91]. These utilizes a combination of graphics and single sentence parsing and generation. An other tool is VINST VIsual and Natural language Specification Tool [Engst91, Preif92, Dalia95a]. VINST is a multi-modal specification and validation tool, specifically for the functionality of telecom services. The users of this tool are intended to be customers and salesmen of telephone systems. The specification is carried out with a Visual Language and a restricted Natural Language (NL).

In this paper we are going to use the aggregation technique for making the output from a natural language generator more pleasant to read. The domain is the Delphi formal specification language in the Delphi Tool, [Ridle94]. The Delphi Tool is a requirements engineering tool and contains graphical editors and an environment of Interpreters and Theorem provers. The intended users of the Delphi Tool are system developers.

The Delphi language has mainly been used for specifying the functionality of telecom services. The Delphi language contains a Conceptual Model for the static descriptions and First Order Predicate Logic for the dynamic descriptions. The Conceptual Model contains entities, attributes, states and relations.

3. The aggregation rules

In linguistics, the results of aggregation is called ellipsis [Quirk72]; the term originates from the Greek word *ellipsis*, meaning missing or omission.

In [Quirk72] the strict sense of ellipsis is defined as words being elided in a text only if they are recoverable. The motivation of ellipsis is to reduce redundancy and avoid repetition. Also included in [Quirk72] is a careful study of ellipsis; e.g., combined and segregatory coordination, what we call predicate and direct object grouping, as well as, ellipsis of subject and auxiliaries, what we call subject and predicate grouping in [Dalia93] and [Dalia95b], other aggregation rules are predicate grouping, subject grouping, clause, and also various semantic aggregation rules.

Aggregation is basically carried out by a set of grouping rules, which make syntactic operations on the text. Aggregation is when the aggregated information can be reconstructed. Aggregation (at least non semantic aggregation) should always be carried out, since no information content is lost. The only reason for not carrying out aggregation is stylistic.

Here are the abbreviations of the categories of a clause

S = Subject	Do = Direct object
P = Predicate	Pc = Predicative Subject Complement
Conn = Connectives i.e. And, Or, Xor.....	Xor = exclusive or

E.g. A normal English clause has the following order S P Do.

Here follows a more exact definition of predicate and direct object grouping (PDO-grouping) and subject and predicate grouping (SP-grouping).

3.1. Predicate and direct object grouping

Definition Predicate and Direct Object grouping, (PDO-grouping)

S_1**P**Do Conn S_2**P**Do Conn ...S_n **P**Do => S_1 Conn S_2 Conn .. S_n**P**Do

An example before aggregation

John is a student

Mary is a student

After aggregation with Predicate and Direct Object grouping =>

John and Mary *are students*

Where the grouped phrase is: *are students* .

3.2. Subject and predicate grouping

Definition Subject and Predicate grouping, (SP-grouping)

SP(Do_1 Xor Pc_1) Conn **SP**(Do_1 Xor Pc_2 Conn) ...**SP** (Do_n Xor Pc_n)

=> **SP**(Do_1 Xor Pc_1) Conn (Do_2 Xor Pc_2) Conn .. (Do_n Xor Pc_n)

An example before aggregation

John is a boy

John is tall

After aggregation with Subject and Predicate grouping =>

John is a boy and tall

Where the grouped phrase is: *John is* .

Subject and predicate grouping works well also with negation

John did not drink a coke and eat a sandwich

John did neither drink a coke nor eat a sandwich

3.3. Grouping rules

The grouping rules, PDO grouping and SP grouping can be generalized to a set of mechanical reducing rules, i.e. if there is a redundant object one must then reduce, but when there is too many objects one may consider either a tabular form or some stylistic rules.

4. The solution

Here follows a suggestion on how to use the aggregation technique to make the generated natural language text non-redundant and easy to read. In our proposal the generated text comes from a specification expressed in the Delphi language.

The prototype is implemented in AAIS-Prolog on a Macintosh, see figure 1, below we explain the function of the natural language generation system.

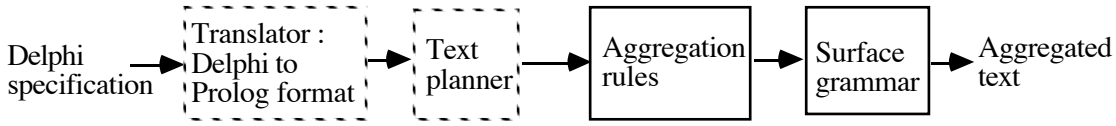

Figure 1. Overview of the natural language generation system

The dotted boxes are not implemented yet. We can pretend that the Delphi textual form is translated to a Prolog readable format and the text planner decides what should be included in the text and creates a text plan. A text plan contains a set of clauses. (See below). The content of the text plan is dependant on what the user asks for.

The aggregation module receives a text plan, sorts the input elements, applies two aggregation rules (SP and PDO grouping) and one pronominalization rule on the sorted text plans and then the surface generator finally generates the NL text. The surface grammar is a DCG-grammar, [Perei80,Clock84] (which still is little rough).

The representation used for the clauses in a text plan is a sort of conceptual modeling representation with entities, relations, states and attributes and a syntax close to Prolog syntax. The Delphi language has an internal language used in the Delphi Tool called Delphi textual form. The idea is that it will be easy to implement the translator from Delphi textual form to the Prolog format.

The syntax of the text plan representation follows below

Text plan	::== {p_1 & p_2 & ....p_n}
p	::== f(Tense,Predicate,Arg,Arg).
Tense	::== pres | fut | past
Predicate	::== isa | state | poss | work_action | walk_action |...
Arg	::== john | mary | subscriber | phone number busy | idle | ...

To create a text plan from the fact base of the Delphi textual form is rather straight forward, this has been made previously with other formal specifications languages, [Black87, Dalia95a]. If the fact base is very large, one may constraint it by generating only what has been changed between two executions in the simulator or by what the user asks for.

Below, in figure 2, follows an example in the telecom domain describing the functionality of telecom services. The static part of the example in the telecom domain is expressed in a Conceptual Model in the Delphi language. Static parts of specifications are expressed in Delphi Conceptual models.

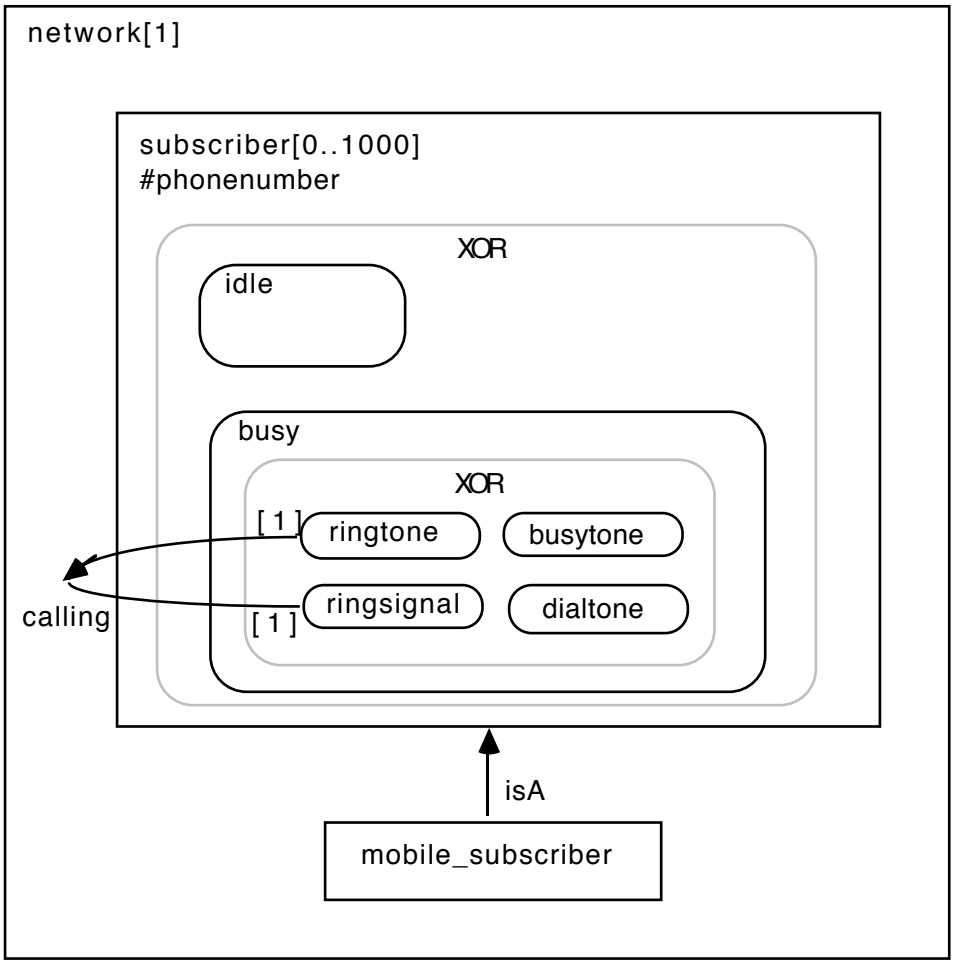

Figure 2. The Delphi conceptual model, which is the static part of the specification.

The above Delphi (static) conceptual model can be paraphrased manually to Natural Language and the reading is:

Subscribers are part of a network. Mobile subscribers are subscribers. Subscribers can either be in the state idle or busy. The state busy can either be in the substates ringtone, ringsignal, busytone or dialtone. When one subscriber is calling an other subscriber then the first subscriber has ringtone and the other subscriber has ringsignal.

The dynamic part of the example in the telecom domain is represented in a first order predicate logic language, see example 1 below. Dynamic parts of specifications in Delphi are expressed in a first order predicate logic language.

```
BEHAVIOUR
   RULES
         WHEN offHook(x) is DETECTED
               IF subscriber(x) AND
                  idle(x) CONCLUDE
                        dialTone(x);
               IF subscriber(x)  AND
                  subscriber(y)  AND
                  calling(y,x) CONCLUDE
                        inspeech(x,y);
         END;
```

Example 1.

If parts of the conceptual model, in figure 2 above, is instantiated one obtains a fact base, see example 2 below, which can be used by the simulator for executing the specification. The fact base is expressed in Delphi textual form.

Instantianted model (Fact base):

```
INSTANTIATION example
   OF SPECIFICATION network
         ENTITIES
         John:subscriber
         phonenumber = 100;phonenumber = 101
          IS
               STATES
               idle;
         Mary:subscriber
         phonenumber = 200
          IS
               STATES
               idle;
         Tom:subscriber
         phonenumber = 300
         IS
               STATES
               idle;
         END;
```

Example 2.

The above fact base in Delphi textual form is at present manually translated to the Prolog style representation which is given to the text planner and then to the aggregation rules module (sentence planner) and finally to the surface generator.

4.1. Ordering

One part of the text planning is to order the input propositions. The text plan contains clauses which have the format:

f(T,Pred,Arg1,Arg2)

The clauses can be sorted according to the keys Pred, Arg1, and Arg2. The keys are given various priority or weight. The sorting is made on the importance of the different predicates according to [Dalia93] as well as to [Patta92]. In [Dalia93] the natural ordering was made in texts over input propositions generated by test persons, in a study of total 15 test subjects.

isa > state > poss > alphabetical order

The weights which has been used in this paper are (1,2,3) and (2,1,3) given to the keys Pred, Arg1, Arg2 respectively. (The used orders are still under investigation in [Dalia95b]). The natural language generation prototype has the above sorting mechanism and a fixed order of applying the aggregation rules to SP grouping and PDO grouping .

4.2. No aggregation

Below, in example 3, we see an NL-paraphrase from the text planner with ordering (1,2,3) but without aggregation (no sentence planning) i.e. pure surface generation.

```
?- paraphrase(f(pres,isa,john,subscriber) &                    (text plan)
              f(pres,state,john,idle) &
              f(pres,poss,john,f(pres,attr,phonenumber,100)) &
              f(pres,poss,john,f(pres,attr,phonenumber,101)) &
              f(pres,isa,mary,subscriber) &
              f(pres,state,mary,idle) &
              f(pres,poss,mary,f(pres,attr,phonenumber,200)) &
              f(pres,isa,tom,subscriber) &
              f(pres,state,tom,idle) &
              f(pres,poss,tom,f(pres,attr,phonenumber,300)) ).

john is a subscriber
  and mary is a subscriber
  and tom is a subscriber
  and john is idle
  and mary is idle
  and tom is idle
```

```
and john has a phonenumber 100
and john has a phonenumber 101
and mary has a phonenumber 200
and tom has a phonenumber 300.
  yes
```

Example 3.

We observe that the above generated NL-text is redundant. Therefore we now activate the aggregation module

4.3. Aggregation

Below, in example 4, we have a NL-paraphrase from the text plan with aggregation (SP- and PDO-grouping rules) we show also the sentence plan i.e. the output from the aggregation module (sentence planner) which is given to the surface generator.

```
?- subject_pred.          (activates SP-grouping)
   yes
?- predicate_do.          (activates PDO-grouping)
   yes
?- pronoun.               (activates pronominalization)
   yes
?- paraphrase(f(pres,isa,john,subscriber) &          (text plan)
                  f(pres,state,john,idle) &
                  f(pres,poss,john,f(pres,attr,phonenumber,100)) &
                  f(pres,poss,john,f(pres,attr,phonenumber,101)) &
                  f(pres,isa,mary,subscriber) &
                  f(pres,state,mary,idle) &
                  f(pres,poss,mary,f(pres,attr,phonenumber,200)) &
                  f(pres,isa,tom,subscriber) &
                  f(pres,state,tom,idle) &
                  f(pres,poss,tom,f(pres,attr,phonenumber,300)) ).
*Spy: | |(4:5) Call: surface(f(pres, isa, john & mary & tom,subscriber) &
                   f(pres, state, pro(john & mary & tom),idle) &
                    f(pres, poss, john,                  (sentence plan)
                   f(pres, attr, phonenumber, 100 & 101)) &
                    f(pres, poss, mary,
                     f(pres, attr, phonenumber, 200)) &
                    f(pres, poss, tom,
                     f(pres, attr, phonenumber, 300)),
                   _2): n
```

```
john
  and mary
  and tom  are  subscribers
  and they  are  idle
  and john  has  phonenumbers  100
  and 101
  and mary  has  a  phonenumber  200
  and tom  has  a  phonenumber  300.
    yes
```

Example 4.

Previously, in example 4, we have seen the aggregated and non-redundant NL-text, which is rather easy to read. Then we execute an event in the Delphi simulator.This is carried out by giving the *event(offhook(john))* to the Delphi simulator which together with the dynamic rules changes the fact base in example 2 above to the one below in example 5. (John becomes busy (he receives dialtone))

```
Instantianted model (Fact base):
   INSTANTIATION example
      OF SPECIFICATION network
            ENTITIES
            John:subscriber
            phonenumber = 100;phonenumber = 101
             IS
                  STATES                        state change in fact base
                  busy;              <-         from idle -> to busy
            Mary:subscriber
            phonenumber = 200
             IS
                  STATES
                  idle;
            Tom:subscriber
            phonenumber = 300
            IS
                  STATES
                  idle;
            END;
```

Example 5.

The above fact base, see example 5, which is expressed in Delphi textual form is manually translated to the Prolog style format and given to the current NLG-system, see example 6 below.

Below we see NL-paraphrase with aggregation

```
?- paraphrase(f(pres,isa,john,subscriber) &                    (text plan)
              f(pres,state,john,busy) &
              f(pres,poss,john,f(pres,attr,phonenumber,100)) &
              f(pres,poss,john,f(pres,attr,phonenumber,101)) &
              f(pres,isa,mary,subscriber) &
              f(pres,state,mary,idle) &
              f(pres,poss,mary,f(pres,attr,phonenumber,200)) &
              f(pres,isa,tom,subscriber) &
              f(pres,state,tom,idle) &
              f(pres,poss,tom,f(pres,attr,phonenumber,300)) ).
*Spy: | |(4:5) Call: surface(f(pres, isa, john & mary & tom, subscriber) &
                            f(pres, state, john, busy) &
                            f(pres, state, mary & tom, idle) &
                             f(pres, poss, john,                   (sentence plan)
                            f(pres, attr, phonenumber, 100 & 101)) &
                             f(pres, poss, mary,
                            f(pres, attr, phonenumber, 200)) &
                             f(pres, poss, tom,
                               f(pres, attr, phonenumber, 300)),
                            _2): n
john
  and mary
  and tom  are  subscribers
  and john  is  busy
  and mary
  and tom  are  idle
  and john  has  phonenumbers  100
  and 101
  and mary  has  a  phonenumber  200
  and tom  has  a  phonenumber  300.
    yes
```

Example 6.

In example 6, above, we observe that the changing of the fact base blocks the pronominalization. The subscriber *John* becomes *busy* and this blocks him from being grouped together with *Mary and Tom.*

In example 7, who follows, we have changed the order of the input propositions from having the weight in sorting from (1,2,3) to (2,1,3) and this gives a completely different text. The PDO-grouping rule becomes also blocked since the SP-grouping rule has already consumed all the input for PDO-grouping. *John*, *Mary* and *Tom* have all their entities, states and phonenumbers respectively attached.

```
?- paraphrase(f(pres,isa,john,subscriber) &
                      f(pres,state,john,busy) &
                      f(pres,poss,john,f(pres,attr,phonenumber,100)) &
                      f(pres,poss,john,f(pres,attr,phonenumber,101)) &
                      f(pres,isa,mary,subscriber) &
                      f(pres,state,mary,idle) &
                      f(pres,poss,mary,f(pres,attr,phonenumber,200)) &
                      f(pres,isa,tom,subscriber) &
                      f(pres,state,tom,idle) &
                      f(pres,poss,tom,f(pres,attr,phonenumber,300)) ).
*Spy: | |(4:5) Call: surface(f(pres, isa_state, john,
                        subscriber & busy) &
                       f(pres, poss, pro(john),
                        f(pres, attr, phonenumber, 100 & 101)) &
                       f(pres, isa_state, mary,
                        subscriber & idle) &
                        f(pres, poss, pro(mary),
                         f(pres, attr, phonenumber, 200)) &
                       f(pres, isa_state, tom,
                        subscriber & idle) &
                        f(pres, poss, pro(tom),
                         f(pres, attr, phonenumber, 300)),
                       _2):
john is a subscriber
  and busy
  and he has phonenumbers 100
  and 101
  and mary is a subscriber
  and idle
  and she has a phonenumber 200
  and tom is a subscriber
  and idle
  and he has a phonenumber 300.
    yes
?-
```

Example 7.

We see that the aggregated text above, in example 7, gives more direct information about each object than the two previous aggregated ones, examples 4 and 6 respectively, which gave more overview information.

5. Conclusions

We have in this paper showed how to use the aggregation technique for removing redundancies in fact bases of formal specifications. This technique makes the generated natural language text easy to read and understand since the text is organized in a conceptual pleasant way.

Future objectives is to use the aggregation technique to remove redundancies from the Delphi conceptual model and the dynamic rules. Some work on using aggregation rules on dynamic rules has been carried out in [Dalia95a]. Other very important issues are the ordering of the input propositions contra the order of applying the aggregation rules. Which combinations of input propositions blocks aggregation and which order should the aggregation rules have on application. This is an subject currently under investigation in [Dalia95b]. Future studies could also be the use of cue words in e.g. examples 4 and 6, which could be further aggregated with use of cue words (e.g. *respectively*)

Before aggregation

Mary has a phonenumber 200

Tom has a phonenumber 300.

After aggregation and use of a cue word =>

Mary and Tom have phonenumbers 200 and 300 respectively

6. Acknowledgements

Many thanks to Ed Hovy at Information Sciences Institute/USC for inspiring me to write a completely new aggregation module and many thanks also to Guy Ridley, Ellemtel Telecommunication Systems Laboratory, for introducing me to the Delphi language and the telecom domain.

7. References

Black87 W.J.Black: Acquisition of Conceptual Data Models from Natural Language Descriptions: In The Proceedings of The Third Conference of the European Chapter of Computational Linguistics, Copenhagen, Denmark 1987.

Cauv91 C.Cauvet et al: ALECSI: An expert system for requirements engineering, in Proceedings of Computer Aided Information System Engineering, CAISE-91, Eds. R. Andersen et al,Trondheim , 1991.

Clock84 W.F. Clocksin & C.S. Mellish: Programming in Prolog, Springer Verlag 1984.

Dalia92 H.Dalianis: A method for validating a conceptual model by natural language discourse generation, CAISE-92 Int. Conf. on Advanced Information Systems Engineering, Loucopoulos P. (Ed.), Springer Verlag Lecture Notes in Computer Science, no 593, pp. 425-444, 1992.

Dalia93 H. Dalianis & Eduard Hovy: Aggregation in Natural Language Generation, EWNLG-93, Proceedings of the 4th European Workshop on Natural Language Generation, Pisa, Italy 1993. also in Trends in Natural Language Generation: an Artificial Intelligence Perspective, Springer Verlag Lecture Notes in Computer Science (fortcoming 1995).

Dalia95a H.Dalianis: Aggregation in the NL-generator of the VIsual and Natural language Specification Tool, in the proceedings of
The Seventh International Conference of the European Chapter of the Association for Computational Linguistics, (EACL-95), Student Session, pp 286-290, Dublin, Ireland, Marsh 27-31, 1995.

Dalia95b H.Dalianis: The Effects of Ordering on Aggregation, in Preparation, 1995.

Engst91 M.Engstedt: A flexible Specification Language using Natural Language and Graphics, Master's Thesis Report, The Centre of Cognive Science, University of Edinburgh, Sept 1991.

Patta92 T. Pattabhiraman: Aspects of Salience in Natural Language Generation, Ph.D thesis, Simon Fraser University, British Columbia, Canada, 1992

Perei80 F.C.N Pereira & D.H.D. Warren: Definite Clause Grammars for Language Analysis - A Survey of the Formalism and a Comparison with Augmented Transition Networks. J. of Artificial Intelligence 13, pp 231-278.1980.

Preif92 S. Preifelt & M. Engstedt: Resultat från VINST projektet, (In Swedish, Results from the VINST project), Ellemtel Telecommunication Systems Laboratory, Älvsjö, Sweden, 1992.

Quirk72 R. Quirk et al: A grammar of contemporary English, *Longman Group Limited*, 1972.

Ridle94 G.Ridley: Formal Methods for Requirement Specification - A Practical Approach using the EUA-Delphi Technology, Ellemtel Utvecklings AB, 1994.

Rolla92 C.Rolland & C.Proix: A Natural Language approach for Requirements Engineering, CAISE-92 Int. Conf. on Advanced Information Systems Engineering, (Ed.) P. Loucopoulos, Springer Verlag Lecture Notes in Computer Science, no 593, pp. 257 - 277, 1992.

Swart82 B.Swartout: GIST English Generator: In Proceedings of AAAI-92, American Association of Artifical Intelligence, Carnegie-Mellon University and University of Pittsburgh, Pittsburgh, Pennylvania, 1982.