AGILE

Automatic Generation of Instructions in Languages of Eastern Europe

---

Title ***The Text Structuring Module for the Intermediate Prototype***

Authors Geert-Jan M. Kruijff
Ivana Kruijff-Korbayová
John Bateman

Deliverable *TEXM2*

Status *Final*

Availability *Public*

Date *July 1999*

INCO COPERNICUS PL961104

Abstract:

In this report we discuss the implementation of a *Text Structuring Module* for the intermediate prototype. The discussion here elaborates on the specification of the text structures for the intermediate prototype as we discussed them in (AGILE 5.2), and relies on the grammar implementations discussed in (AGILE 7.2) for various aspects of realisation.

Our discussion will focus on the implementation of the progression from Goal (A-box) to Sentence Plans. The Text Structuring Module we have developed includes a *Text Planner* component that creates a text plan for realising a given A-box, and a *Sentence Planner* component that creates sentence plans (AGILE 5.1) for individual sentences on the basis of the text plan. The A-box that the Text Planner takes as its input comes from the user interface (AGILE 1.1). The SPLs provided by the Sentence Planner serve as input to a language-specific tactical generator (AGILE 7.2).

More information on AGILE is available on the project web page and from the project coordinators:

URL:	http://www.itri.brighton.ac.uk/projects/agile
email:	agile-coord@itri.bton.ac.uk
telephone:	+44-1273-642900
fax:	+44-1273-642908

Contents

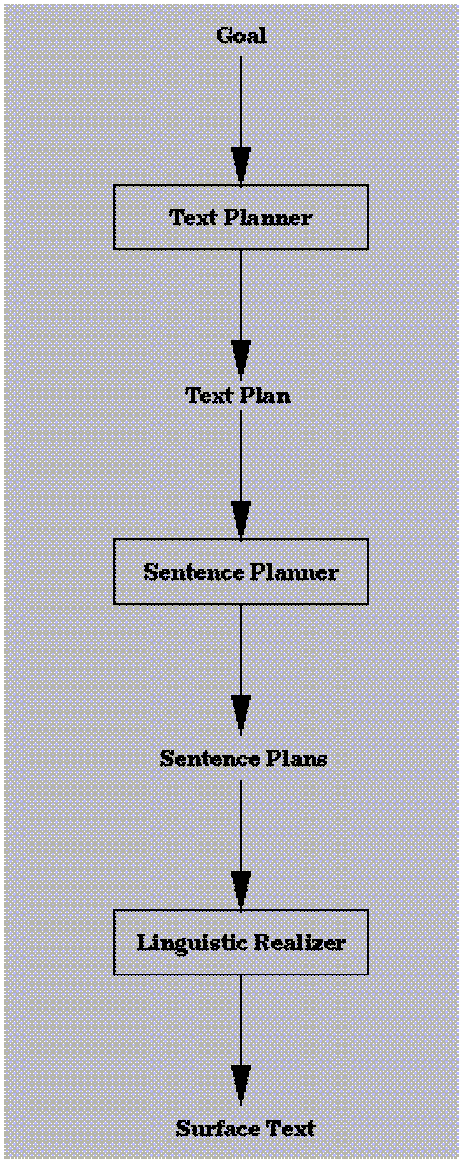
1. Introduction 1
   1.1 Overview 2
2. From content to text plan 3
   2.1 Text templates and text structure elements 3
   2.2 Configurational concepts for the intermediate prototype 4
   2.3 A text plan and its building blocks 6
   2.4 Mapping configurational concepts to text structure elements 6
   2.5 Text templates and general realisation constraints 7
   2.6 Adding discourse relations 9
   2.7 Implementation 10
      2.7.1 The CADCAM-INSTRUCTIONS region 10
      2.7.2 The *Text Planner* function 13
   2.8 Elaborated example 13
      2.8.1 The A-box 13
      2.8.2 The text plan 16
      2.8.3 Realisation and layout 17
3. From text plan to sentence plans 18
   3.1 Aggregation and sentence boundaries 18
   3.2 Determining coherent sentential information structure 19
   3.3 Implementation of the sentence planner 19
4. End-to-end generation in the intermediate prototype 20
   4.1 The A-boxes and their (target) texts 20
   4.2 Generated text plans and SPLs 22
      4.2.1 Text plan and SPLs for test-text 1 22
      4.2.2 Text plan and SPLs for test-text 2 23
5. Future research 25

List of Figures

Figure 1 - HTML-style specification of text structure element layout..... 9
Figure 2 - Text Structure Elements and RST relations..... 10
Figure 3 - The first choice is one of linguistic stratum..... 11
Figure 4 - CADCAM-INSTRUCTIONS region..... 12
Figure 5 - A-box for example text extracted from IMD-Text1..... 16
Figure 6 - From A-box to text structure..... 17
Figure 7 - Example of text realisation..... 17
Figure 8 - Text plan for test text 1..... 22
Figure 9 - Text plan for test text 2..... 24

1. Introduction

In the AGILE project we aim at *end-to-end generation*: i.e. given a specification of the content the user would like to see expressed, the system is to process this specification and produce a sequence of sentences that together make up a coherent text realising that particular content. Following Reiter and Dale (1997), we can conceive of this progression from content specification to actual text as a three-stage process (see Figure 1).



For the case of AGILE, each of the stages and their respective input/output have the following meaning.[1]

Our *Goal* is the content the user wants to be realised. Making use of the authoring interface that is described by Power in (AGILE 2.1), the user can specify this content interactively with the system. Formally, the content specification takes the form of a so-called *A-box* (short for *Assertion-box*). An A-box specifies content by instantiating domain-specific concepts like actions and events. Within an A-box these actions and events are placed together (*configured*) into domain-related structured contexts such as procedures and methods.

The *Text Planner* takes an A-box as input. The task of the Text Planner is to create a *text plan*. A text plan organises the content specified by the A-box into a structure that relates parts of the A-box to one another in a hierarchical fashion. The text plan may also specify discourse relations that hold between parts, or groups of parts, of the A-box. These discourse relations indicate how the different text fragments can be related rhetorically.

It is convenient to think of the text plan as a tree in which the leaves refer to individuated parts of the A-box. The *Sentence Planner* faces the task of creating partial sentential representations for these individuated parts, and combining (*aggregating*) these representations into sentence plans. For AGILE, this comes down to first of all producing ideational Sentence Planning Language (SPL) code for each of the individuated parts. Thereupon we may need to aggregate different fragments of SPL code into one sentence plan for them to be realised by a single sentence.

---

[1] The setting of AGILE is more restricted than the general one described by Reiter and Dale. Notably, the goal in AGILE already provides a complete description of the content that is to be realised. This is different from the more general setting Reiter and Dale consider, since in their case the goal is an abstract formulation for which the system needs to determine a concrete content that would "achieve" that goal. Content determination as such is part of the Text Planner, together with discourse planning. For AGILE we can restrict ourselves to discourse planning when developing a Text Planner.

Besides generating the ideational SPL code and creating basic ideational sentence plans from that, the Sentence Planner also needs to ensure that the sentence plans include constraints on syntactic realisation (to conform to the chosen layout) and decide on the information structure of each sentence. The approach we take to Sentence Planning in AGILE differs from the general architecture described by Reiter and Dale as follows. Reiter and Dale distinguish three tasks for a Sentence Planner: s*entence aggregation*, *lexicalisation*, and *referring expression generation*. Where needed we will perform aggregation as well (including imposing constraints on syntactic realisation). In the current phase, we leave lexicalisation entirely to the grammar (AGILE 4.2). Finally, the task of referring expression generation is in a sense integrated into the more general task of determining contextually appropriate information structure. In AGILE, we let the Sentence Planner determine an information structure for a sentence such that the sentence would be coherent in the given context. In the surface form of the sentence the information structure then becomes manifest by word order and referring expressions – but generating word order and referring expressions is, in AGILE, a concern of a language-specific tactical generator.

Subsequently, the sentence plans that have been composed by the Sentence Planner are provided as input to a *Linguistic Realizer* - in AGILE, one of the tactical generators for Bulgarian, Czech, or Russian. The output of the tactical generator is a sequence of sentences, each generated according to its individual sentence plan, and together making up a coherent text.

1.1 Overview

It may be clear from the discussion above that text planning plays a crucial role in end-to-end generation. In this report we discuss the implementation of a *Text Structuring Module* for the intermediate prototype. The discussion here elaborates on the specification of the text structures for the intermediate prototype as we discussed them in (AGILE 5.2), and relies on the grammar implementations discussed in (AGILE 7.2) for various aspects of realisation.

Our discussion will focus on the implementation of the progress from Goal (A-box) to Sentence Plans. The Text Structuring Module we have developed thus includes a Text Planner component as well as a Sentence Planner component.

We begin in section 2 with showing how we get from an A-box, specifying a content to be expressed, to a Text Plan. There we propose a small set of *text structure elements*, which are the elements from which the kind of task-oriented text considered in AGILE is built up. We then provide a mapping between these text structure elements and the concepts used in an A-box to configure content (*configurational concepts*). In this way we obtain a close connection between how content can be defined in an A-box and how that content is to be arranged in a text expressing that content. Consequently, by observing the configuration of content in an A-box we can compose in parallel a text plan out of the text structure elements that the configurational concepts map to.

However, the text structure elements only describe *what* is to be realised. In order to be able to control *how* the content is to be realised, we introduce *text templates*. A text template defines a *style* in terms of how specific text structure elements (like a title) should be linguistically realised. We discussed at length the different style preferences for Bulgarian, Czech and Russian in (AGILE 5.2).

After we have thus obtained a Text Plan, we move on to sentence planning. In section 2.8 we discuss the concrete tasks of the Sentence Planner component in more detail, and we show how these tasks are achieved in the implementation.

We end the document with two brief examples showing how the Text Structuring Module works in practice. We start out with two complex A-boxes, and provide the generated Text Plans and SPLs for these A-boxes.

The full implementation of the Text Structuring Module, like other AGILE code, is not made publicly available.

2. From content to text plan

In this section we discuss how we specify the structure of a text to be produced by the Text Structuring Module. We begin by making a distinction between *text structure elements*, being the elements from which a task-oriented text is built up, and *text templates*, which condition the way text structure elements are to be linguistically realised (Section 2.1). Subsequently, we focus on the relation between domain model concepts on the one hand, and text structure elements on the other hand. We are specifically interested in those concepts that are used to configure the content specified in an A-box (*configurational concepts*), like the PROCEDURE and METHOD* concepts. This is because we want to have a close connection between how content can be defined in an A-box and how that content is to be spelled out in a text using a particular organisation. That we can ensure that the content to be expressed is indeed reflected by the text (Section 2.4).

2.1 Text templates and text structure elements

In the introduction to the (AGILE 5.2) report we already alluded to the idea that texts may vary in their distribution and lexico-grammatical realisation of content. The corpus investigations we carried out for the languages under study in AGILE provided empirical support for the view that the observable alternations can be classified into two main groups, as follows:

- *Choice of grammatical means*: whether lexico-grammatical choices are predetermined in certain ways by the style of text.
- *Distribution of content and lexical realisation*: whether information is realised explicitly in the most straightforward way with respect to the A-box, so that there is only a minimal inference load required to interpret a piece of text, or whether information is left more or less implicit. And, what the realisation is of the explicitly conveyed information.

Here we propose two notions that should enable us to capture these differences in a principled fashion. These two notions are the notion of *text structure element* and the notion of *text template*.

A text structure element is a predefined component that is to be filled by one or more particular parts of the user's definition of the content to be spelled out by the text. Using the reader-oriented terminology common in technical authoring guides, we will thus distinguish text structure elements like TASK-INSTRUCTIONS, SUBTASKS, etc. (for a full listing of the text structure elements we use for the intermediate prototype see Section 2.3).

Orthogonal to the notion of text structure element is the notion of text template. Whereas text structure elements capture *what* needs to be realised, the idea of a text template

captures *how* that content needs to be realised. Thus, a template defines a *style* in which content is to be realised.

There are choices, though, that do concern realisation but which are relatively independent of the style in which a text is to be generated. We will call these *general realisation constraints.* A good example of such a choice is whether side effects, like the appearance of a menu, should be explicitly mentioned or whether they can be left more or less implicit. Thus:

(1) ***Explicit side effects:***

(a) *Click in the text pane*

(b) *The* File *menu appears.*

(c) *Select* Save

(2) ***Implicit side effects:***

(a) *Click in the text pane*

(b) *Select* Save *in the* File *menu.*

In the next two sections we provide a formal definition of the configurational concepts for the intermediate prototype, and the text structure elements distinguished in AGILE.

2.2 Configurational concepts for the intermediate prototype

The AGILE domain model, as previously described in (AGILE 2.1), has undergone various changes in order to

1. provide a more uniform, recursive scheme for modelling text structure (in comparison to the previous proposal), and to
2. cover the text structures as found in the corpora studied in the context of this deliverable (see the earlier section).

We now employ just four concepts to define the admissible A-box configurations, instead of the seven concepts used earlier. These configurations provide the structured contexts in which actions and events take place. The *configurational concepts* we use are PROCEDURE, METHOD*, PROCEDURE-LIST and METHOD-LIST. They are defined as follows[2]:

PROCEDURE

A procedure has three slots:

- GOAL (obligatory, to be filled by a USER-ACTION)
- METHODS (optional, to be filled by a METHOD-LIST)
- SIDE-EFFECT (optional, to be filled by a USER-EVENT)

[2] The asterisk on METHOD* prevents a clash with the LISP symbol Method.

METHOD

A method has three slots:

- CONSTRAINT (optional, to be filled by an OPERATING-SYSTEM)
- PRECONDITION (optional, to be filled by a PROCEDURE)
- SUBSTEPS (obligatory, to be filled by a PROCEDURE-LIST)

METHOD-LIST

A Method-List is essentially a list of METHOD*'s:

- FIRST (obligatory, of type METHOD*)
- REST (optional, of type METHOD-LIST)

PROCEDURE-LIST

A Procedure-List is essentially a list of PROCEDURE's:

- FIRST (obligatory, of type PROCEDURE)
- REST (optional, of type PROCEDURE-LIST)

In comparison to the previous set of concepts, which were modelled on the basis of the DRAFTER-2 project, we no longer have a distinction between PROCEDURE and COMPLEX-PROCEDURE, or METHOD and COMPLEX-METHOD (see also (AGILE 5.1)). This simplifies the T-box at the cost of complicating the A-box, in the sense that more entities are sometimes needed in order to model an instruction.

For instance, a simple goal-subgoal relationship like 'Choose Save to save the drawing' requires the following A-box structure:

```
procedure
GOAL 'save the drawing'
METHODS method-list
        FIRST method
              SUBSTEPS procedure-list
                       FIRST procedure
                             GOAL 'choose Save'
```

The new attribute CONSTRAINT on a method enables us to make a distinction between methods applicable under Windows and methods applicable under DOS or UNIX, as in the following example from IMD- Text 1 (AGILE 5.2):

(3) First open the Multiline Styles dialog box using one of these methods:
Windows: From the Object Properties toolbar or the Data menu, choose Multiline Style.
DOS and UNIX: From the Data menu, choose Multiline style.

2.3 A text plan and its building blocks

In this section we define the essential building blocks from which we compose a text plan: the *text structure elements*. These text structure elements provide a more fine-grained way of organizing the content that needs to be realised. Or, put differently, they determine the distribution of content in an abstract sense. Their definition is inspired by the reader-oriented terminology one often finds in technical authoring guides.

Our definition of text structure elements differs slightly from the definitions used in the previous deliverable of Work Package 5 (AGILE 5.1). The reason for this difference is that the underlying domain model has changed, as we noted in the previous section.

TASK-DOCUMENT

A task-document has two slots:

- TASK-TITLE (obligatory)
- TASK-INSTRUCTIONS (obligatory)

where "TASK-INSTRUCTIONS" means at least one INSTRUCTION.

INSTRUCTION

An instruction has three slots:

- TASKS (obligatory)
- CONSTRAINT (optional)
- PRECONDITION (optional)

where "TASKS" means at least one TASK.

TASK

A task has two slots:

- INSTRUCTIONS (obligatory)
- SIDE-EFFECT (optional)

In comparison to the definitions in (AGILE 5.1) the above set proves to be simpler, which is due to the recursive nature of the T-Box concepts defined in the (revised) domain model for the intermediate prototype.

2.4 Mapping configurational concepts to text structure elements

This mapping will enable us to provide a flexible definition of rules for text structuring of a content defined by an A-box. For the intermediate prototype we employ the following mapping between T-Box Concepts and Text Structure Elements.

- TASK-TITLE ↔ GOAL of topmost PROCEDURE
- TASK-INSTRUCTIONS ↔ METHODS of PROCEDURE
- SIDE-EFFECT ↔ SIDE-EFFECT of a PROCEDURE
- TASK ↔ a PROCEDURE's GOAL
- INSTRUCTIONS ↔ a PROCEDURE's METHODS
- INSTRUCTION ↔ METHOD
- CONSTRAINT ↔ CONSTRAINT of a METHOD
- PRECONDITION ↔ PRECONDITION of a METHOD
- TASKS ↔ a METHOD's SUBSTEPS

By way of an example, let us consider again the A-Box for *Choose* Save *to save the drawing*:

```
procedure
  GOAL 'save the drawing'
  METHODS method-list
          FIRST method
                  SUBSTEPS procedure-list
                           FIRST procedure
                               GOAL 'choose Save'
```

This A-Box would translate into the following arrangement of text structure elements, given the mapping we provided above:

TASK-TITLE "Save the drawing"
TASK-INSTRUCTIONS
INSTRUCTION
TASKS
TASK "Choose Save"

2.5 Text templates and general realisation constraints

At this point, let us recapitulate our discussion so far. Above we saw that text structure elements provide a more fine-grained organisation of what content needs to be realised. More precisely, they determine distribution of content in an abstract sense. Furthermore, in earlier sections we already alluded to the viewpoint that we can use text templates to prescribe the choice of grammatical means in the realisation of content, thus determining the style of the text. In order to do so templates may require the introduction of additional discourse markers to make text structure explicit (for example, as in **informative** parts of text, see (AGILE5.2)). Finally, independent of a particular text template there may be choices influencing the level of explicitness in informing the reader –for example, on observable side effects, as we saw earlier.

In the corpus investigation in (AGILE 5.2) we indicated that among the features attended to in the investigation were the following:

- Expressing or hiding the intentional agent of an action (agency, voice)
- Ways of expressing the readership – whether the reader should be addressed explicitly or not (person, number, voice)
- Mode of realising instructions (mood, voice)
- The complexity of linguistic expressions (Group and clause complexity, rank shifting)

These aspects were dealt with in a more precise way to specify the variation of text style for the intermediate prototype in (AGILE 5.2). We discussed two different styles in which instructions can usually be realised, being the *personal* or the *non-personal* style. More specifically, we distinguished an instruction's heading and its style of realisation from the instruction's steps and their realisation in a particular style. We also explained how there seems to be an interdependency between the style in which the instruction steps are generated and the style that seems appropriate for the heading.

The purpose of a text template is to relate these choices of style to the actual elements from which our text plans are composed – the text structure elements. In other words, a text template should specify the following aspects:

- **Realisation of instruction heading** – constraints that are to be imposed on the TASK-TITLE of a TASK-DOCUMENT. As we already pointed out in (AGILE 5.2), for the purpose of the intermediate prototype we will restrict ourselves to realising headings as nominal groups.
- **Realisation of instruction steps** – constraints that are to be imposed on the TASK-INSTRUCTIONS of a TASK-DOCUMENT. These constraints regard *Style*:
  - *Personal*: Realise TASK-INSTRUCTIONS, meaning TASKS and INSTRUCTIONS,
    - Using declarative mood (possibly including dropped subjects), or in imperative mood.
    - Using a polite form of addressing (second person plural) or a familiar way of addressing the reader.
  - *Non-Personal*: Realise TASK-INSTRUCTIONS, thus TASKS and INSTRUCTIONS, in indicative mood in passive voice or using an infinitive.

In (AGILE 5.2) we already specified the statements to be put into an SPL in order to reflect the various aspects of a particular style. Using these statements, a text template can impose constraints on the realisation of a text structure element by means of KPML's REALIZE-WITH statement. For example, if we want to see the TASK-TITLE realised as a nominal group, then we can use

```
(REALIZE-WITH TASK-TITLE
(:EXIST-SPEECH-ACT-Q NOSPEECHACT))
```

with ":EXIST-SPEECH-ACT-Q NOSPEECHACT" being the SPL component responsible for the realisation of the TASK-TITLE as a nominal group (cf. Section 3, AGILE 5.2).

In addition to constraints on lexico-grammatical realisation, we can also specify in a template the means to *layout* individual text structure elements. KPML provides us with the

ANNOTATE function, which can be used to specify in an SPL that the constituent(s) realising a particular piece of the SPL's content should be annotated with a particular set of tags. Following the practice of the previous Work Package 5 deliverable (AGILE 5.1) we can use HTML tags to specify a particular layout, which can be viewed when an annotated set of sentences is loaded into for example a web browser.

For example, the following mapping between text structure elements and HTML tags could be used in a text template delineating a procedural style of instructional text. We give the layout as HTML tags with unspecified (i.e. default) values.

Text Structure Element	Layout	HTML Tags
TASK-TITLE	Heading level *N*	<H*N*> … </H*N*>
(TASK-) INSTRUCTIONS	Ordered list	<OL> … </OL>
INSTRUCTION	(Ordered) list item	<LI>
CONSTRAINT	Description list item	<DL> <DD>… </DL>
PRECONDITION	(Ordered) list item	<LI>
SIDE-EFFECT	Description list item	< DL> <DD>… </DL>
TASK	Text	

Figure 1 - HTML-style specification of text structure element layout

Finally, besides the constraints on realisation imposed by a text template we may have general constraints on realisation – as we often noted already, in the case of the intermediate prototype an example of such a general constraint is the explicit or implicit realisation of a side effect.

2.6 Adding discourse relations

Text structure elements (TSEs) and text templates provide the principal means to form a text plan and impose language-dependent constraints on the style in which the text is to be generated. They are, however, not enough. TSEs in and by themselves provide a lot of structure, but we need some additional means to be able to generate a more fluent discourse. Discourse relations offer this opportunity.

We add discourse relations to the text plan in the following way. There are several recurring patterns of partial TSE structures that can be identified as particular relations from Rhetorical Structure Theory (RST, see Mann & Thompson, 1987). RST relations are part and parcel of the Penman Upper Model (Bateman et al, 1990; Hovy, 1993) and they can therefore be employed in a straightforward manner within AGILE. The patterns and relations we currently distinguish are the following ones:

A-box configurations	Corresponding TSEs	RST relation
A procedure's *Goal* and *Methods*	A *Task* and its *Instructions*	***Manner***: domain: *Task*, range: *Instructions*
A method's *Precondition* and *Substeps*	An *Instruction* and its *Tasks*	***Purpose****:* domain: *Instruction*, range: *Tasks*

Figure 2 - Text Structure Elements and RST relations

An important consequence of *having* discourse relations in the text plan is that the Sentence Planner can then derive coherent sequences of information structures for the clauses that are being related. Thereby we base ourselves on work described for example (Kruijff and Schaake, 1997). In the next deliverable (TEXM3) we will explore these issues in more detail.

2.7 Implementation

Following the approach that was taken in the TEXM1 deliverable (AGILE 5.1), we have implemented a region CADCAM-INSTRUCTIONS to establish a text plan for a given A-box, and a LISP function that prompts the traversal of this region upon receiving as input an A-box. The region defines an additional level of linguistic resources for the level of 'genre', and it enables the building up of text structure in a way that is similar to the way the grammar builds up grammatical structure. It was already noted in (AGILE 5.1) that such a region can be easily "placed on top of" a tactical generator. This makes it easy to integrate it not only with the lexico-grammars that are being developed for AGILE but - conceivably- also with other grammars developed using the KPML environment. Below we first describe the region (§ 2.7.1), and then the function (§ 2.7.2).

2.7.1 The CADCAM-INSTRUCTIONS region

The region, called CADCAM-INSTRUCTIONS, builds on the resources developed for the initial demonstrator. This region is to be loaded after language-specific resources (i.e. one or more grammars) have been loaded, together with a revised RANKING region that establishes that the first choice is one of linguistic stratum, not grammatical rank. Figure 3 shows the RANKING region, where there is first a choice between *genre* (text) and *lexicogrammar* (sentences).

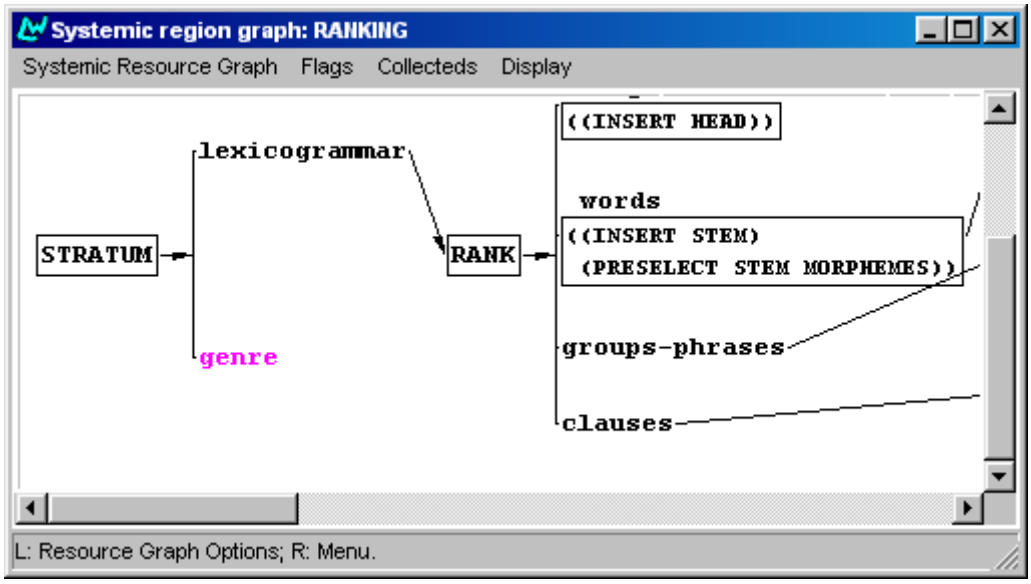
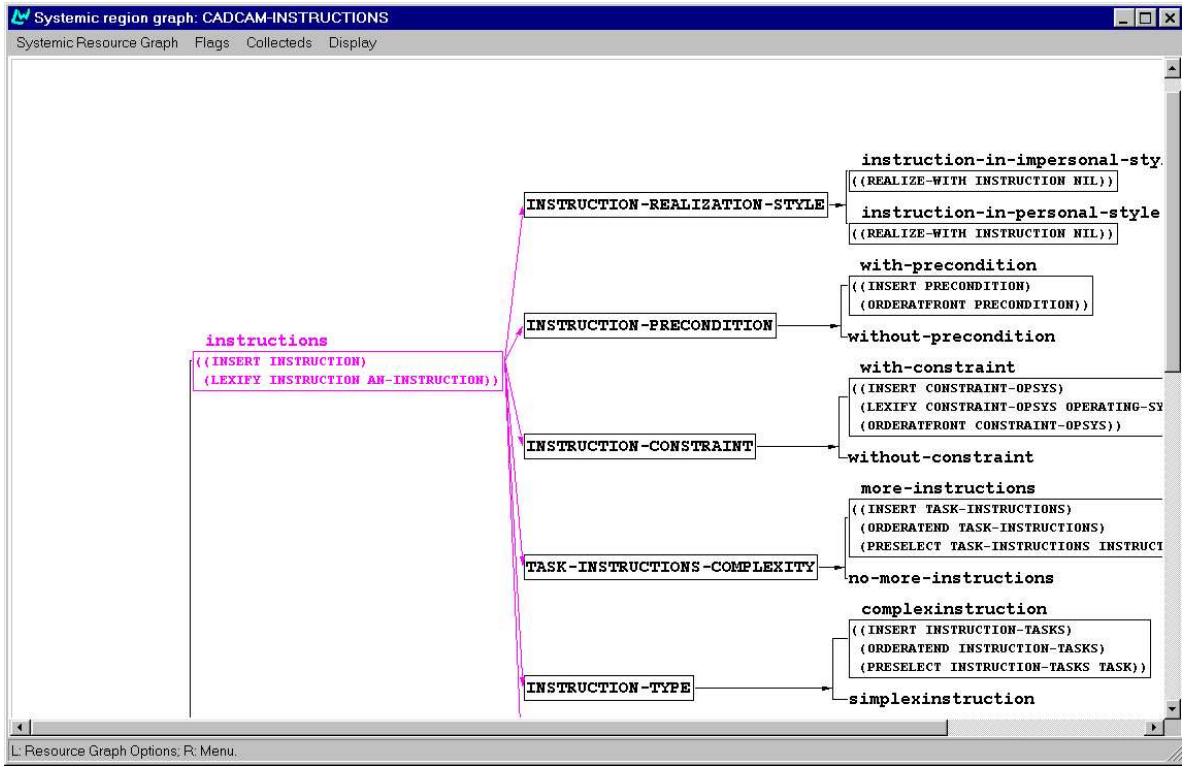


Figure 3 - The first choice is one of linguistic stratum

For the CADCAM-INSTRUCTIONS region we follow out the viewpoint that text templates and text structure elements are essentially orthogonal ideas. The region therefore consists of two interacting parts. One part deals with interpreting the A-box in terms of text structure elements. The systems of this part are placed directly under the TASK UNIT TYPE system. By traversing the network that these systems make up, we obtain a text structure that conforms to the definitions we provided above (§ 2.4), and which provides a text plan for the content defined in the A-box.

The other part of the region deals with the text templates and general conditions on realisation that we also discussed earlier. The aim of this part is to impose constraints on the realisation of the text structure elements that are being introduced by traversing the other part of the region, thus enforcing a particular *style* of text. In (AGILE 1.1) we describe the user interface through which the user can determine various style parameters. These parameters relate to whether a text should be generated in personal or impersonal style, and whether side effects should be realised or not.

The part of the CADCAM-INSTRUCTIONS region dealing with style is depicted in Figure 4, together with the systems making up the other part of the region. It shows the systems responsible for imposing constraints to bring about the style chosen by the user, and systems dealing with language-dependent variety that is independent from the choices the user made. An example of such a system is the one imposing constraints on the realisation of a TASK-TITLE. Titles in the Czech and Bulgarian versions of the target texts use a different syntactic construction than the ones found in the Russian targets – see the discussion in (AGILE 5.2).



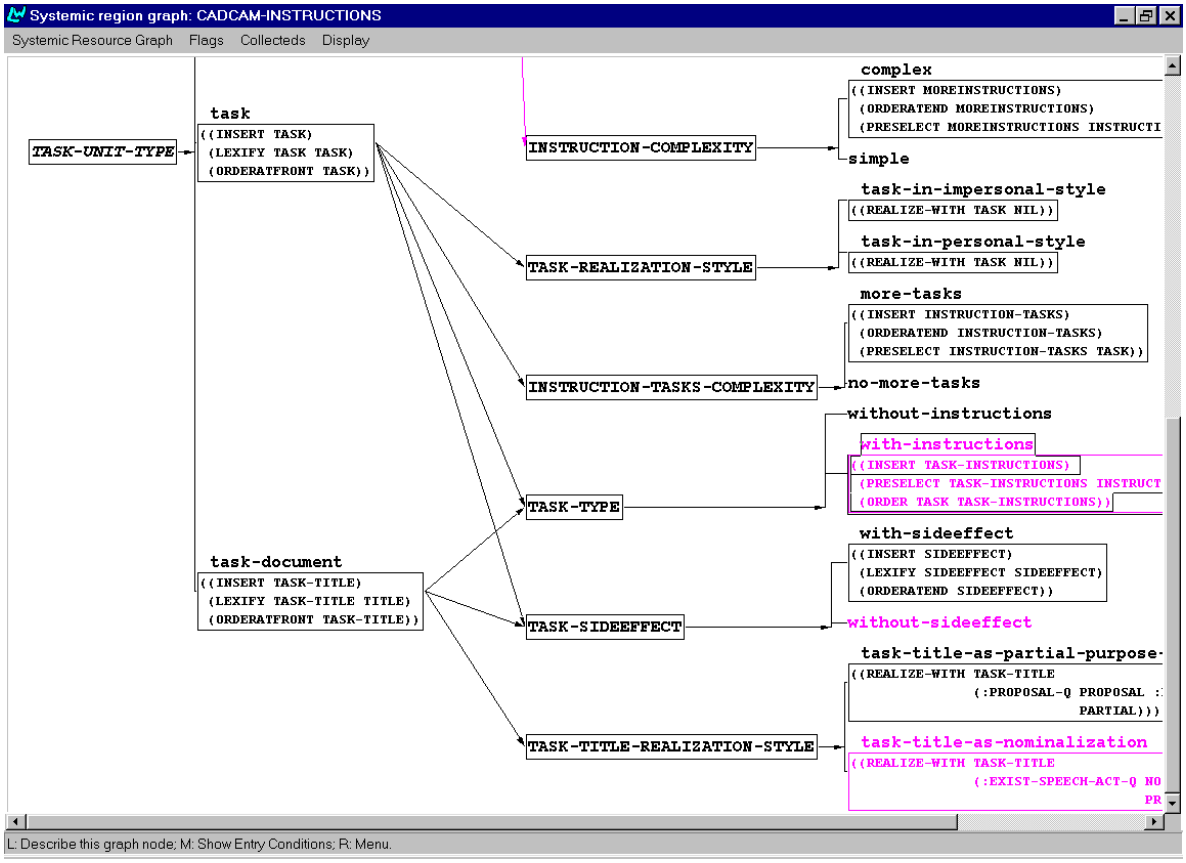


Figure 4 - CADCAM-INSTRUCTIONS region

Besides the CADCAM-INSTRUCTIONS region, the text planning also relies on a function called *text planner*, which we describe in the next section.

2.7.2 The *Text Planner* function

The driving force behind the text planner (and the Text Structuring Module in general) is the `text-planner` function, which has been implemented in LISP. This function can be called with an A-box and a data structure containing information on style parameters.

Once invoked, the function starts the traversal through the CADCAM-INSTRUCTIONS region to generate a text plan, following the content as specified by the A-box. This is done by a number of choosers and inquiries (defined by the region) that inspect the A-box and decide according to the configuration found there. Another important aspect of the traversal through the region, besides constructing a text plan, is the association of the elements of the text structure with appropriate fragments from the A-box. As we already indicated earlier, these parts of the A-box can be conceived of as making up the leaves of the text plan. By using so-called ID inquiries we can easily establish these associations.

Once the function has obtained a text plan, including associations to different parts of the A-box, the `text-planner` function calls the *sentence planner*. The sentence planner translates the text plan into a sequence of SPLs that specify individual sentences realising (parts of) the content given by the A-box, in the particular style and language as selected by the user. We describe the sentence planner in more detail in section 3.3 below.

2.8 Elaborated example

We close this section on text planning with a more elaborate example of how we generate a text plan, given a content specified by an A-box. For this purpose, we use a shortened version of the A-box of the first text for the intermediate prototype. This A-box actually corresponds to the texts we used in (AGILE 5.2) to illustrate possible style alternations in the context of the intermediate prototype.

Below we commence by providing the promised A-box. Our primary aim with the A-box is to provide a picture of a more complex configuration of content, using the concepts we discussed earlier. To that end, in the A-box we give below we do omit some of the concepts that are irrelevant to our discussion (notably, some of the concepts that are used to set up list-like structures). Subsequently, given this complex configuration, we proceed by providing the corresponding text plan in terms of text structure elements, following the mapping of Section 2.4. Once we have this text plan, we provide the realisations corresponding to the structure and following a particular style. We end the example by showing the text in a possible layout, using the mapping to HTML tags as in Figure 1.

2.8.1 The A-box

The A-box we given below in Figure 5 is a shortened version of the A-box for Text 1 for the intermediate prototype, in two respects. Firstly, its content corresponds to the text used in the first text for the intermediate prototype. Secondly, not all the concepts that can be found in the original A-box are represented here. As for the latter, whenever we omit some concepts we indicate their expunging by means of "…". With regard to typesetting, we display configurational concepts using *italics*, and content concepts using **boldface**.

```
PROCEDURE
  GOAL:
    CREATE
    ACTEE:
    … STYLE
      OWNER:
        MULTILINE
    ACTOR:
    … USER
  METHODS:                  %% This is the way to specify lists of methods,
  METHOD-LIST               %% as we already defined earlier. In the A-box
  FIRST:                    %% hereafter we will omit the extra concepts
    METHOD*                 %% needed to define lists (METHOD-LIST, FIRST,
      PRECONDITION:         %% REST) in order to shorten the A-box.
      PROCEDURE
        GOAL:
          OPEN-SCREEN-OBJECT
          ACTEE:
            GUI-ACTEE
            RANGE:
              DIALOGUE-BOX
            LABEL:
              Multiline Styles
          ACTOR:
          … USER
        METHODS:            %% Eventually, the following part of the A-box
        … METHOD*           %% will be realised as
            CONSTRAINT: %% „Windows: From the Object Properties toolbar
               OPERATING-SYSTEM   %%or Data menu, choose Multl. Style"
                 LABEL:       %%Observe the coordination involving
                   Windows    %%a disjunction- content wise, there are 2
            SUBSTEPS:         %% methods given for the Windows case. The
            … PROCEDURE       %% A-box part defines two METHODs, both of
              GOAL:              %% which have as CONSTRAINT the
                                 %% OPERATING-SYSTEM
                 CHOOSE       %% Windows. To the left of this comment is
                   ACTEE:        %% the first method.
                     DISPLAYED-ACTEE
                     RANGE:
                       OPTION
                     LABEL:
                       Multiline Style
                   OPTIONS:
                     GUI-ACTEE
                   RANGE:
                     TOOLBAR
                   LABEL:
                     Object Properties
                   ACTOR:
                   … USER
        … METHOD*           %% And here is the second part for Windows.
            CONSTRAINT:
              OPERATING-SYSTEM
                LABEL:
                  Windows
              SUBSTEPS:
              … PROCEDURE
                  GOAL:
                    CHOOSE
                      ACTEE:
```

```
                              DISPLAYED-ACTEE
                              RANGE:
                                 OPTION
                              LABEL:
                                 Multiline Style
                           OPTIONS:
                              GUI-ACTEE
                              RANGE:
                                 MENU
                              LABEL:
                                 Data
                           ACTOR:
                        … USER

            METHOD*
               CONSTRAINT:
                  OPERATING-SYSTEM
                  LABEL:
                     DOS and UNIX
               SUBSTEPS:
               … PROCEDURE
                  GOAL:
                     CHOOSE
                     ACTEE:
                        DISPLAYED-ACTEE
                        RANGE:
                           OPTION
                         LABEL:
                           Multiline Style
                     OPTIONS:
                        GUI-ACTEE
                     RANGE:
                        MENU
                        LABEL:
                           Data
                     ACTOR:
                        … USER
SUBSTEPS:
… PROCEDURE
     GOAL:
        ADD-TO
        ACTEE:
        … ELEMENTS
        RECIPIENT:
        … STYLE
        ACTOR:
        … USER
     METHODS:
     METHOD*
       SUBSTEPS:
       …  PROCEDURE
          GOAL:
             CHOOSE
             ACTEE:
                DISPLAYED-ACTEE
                RANGE:
                   OPTION
                LABEL:
                   Element Properties
```

```
                OPTIONS:
                   GUI-ACTEE
                ACTOR:
                … USER
   … PROCEDURE
        GOAL:
           ENTER
           ACTEE:
            … OFFSET
                OWNER:
                   STYLE-ELEMENT
           LOCATION:
              GUI-ACTEE
              RANGE:
                 DIALOGUE-BOX
              LABEL:
                 Element Properties
           ACTOR:
            … USER
```

Figure 5 - A-box for example text extracted from IMD-Text1

2.8.2 The text plan

Consider again the A-box above. Using the mapping we provided in Section 2.4 we can create a text plan, composed of text structure elements, as follows, see Figure 6. (Since the mapping only concerns the concepts used for configuring content in the A-box, we omit from the A-box below all concepts that are irrelevant to configuration.)

```
PROCEDURE
  GOAL:                                  → TASK-TITLE
    CREATE
  METHODS:                               → TASK-INSTRUCTIONS
  METHOD-LIST
  FIRST:
    METHOD*                              → INSTRUCTION
      PRECONDITION:                      → PRECONDITION
      PROCEDURE
        GOAL:                            → TASK
          OPEN-SCREEN-OBJECT
        METHODS:                         → INSTRUCTIONS
        … METHOD*                        → INSTRUCTION
          CONSTRAINT:                    → CONSTRAINT
                OPERATING-SYSTEM
            SUBSTEPS:
            … PROCEDURE
                GOAL:                    → TASK
                  CHOOSE
        … METHOD*                        → INSTRUCTION
            CONSTRAINT:                  → CONSTRAINT
                OPERATING-SYSTEM
            SUBSTEPS:
            … PROCEDURE
                GOAL:                    → TASK
                  CHOOSE
        … METHOD*                        → INSTRUCTION
            CONSTRAINT:                  → CONSTRAINT
```

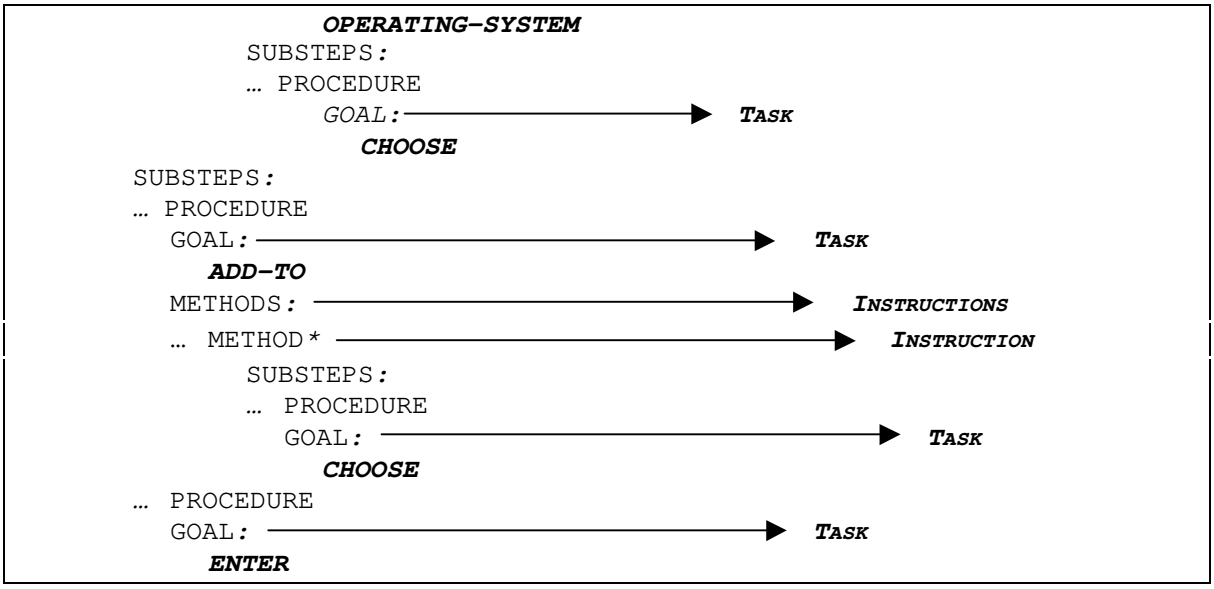


Figure 6 - From A-box to text structure

2.8.3 Realisation and layout

An example of the realisation that could result from the text structuring process discussed above is shown below for English and Czech.

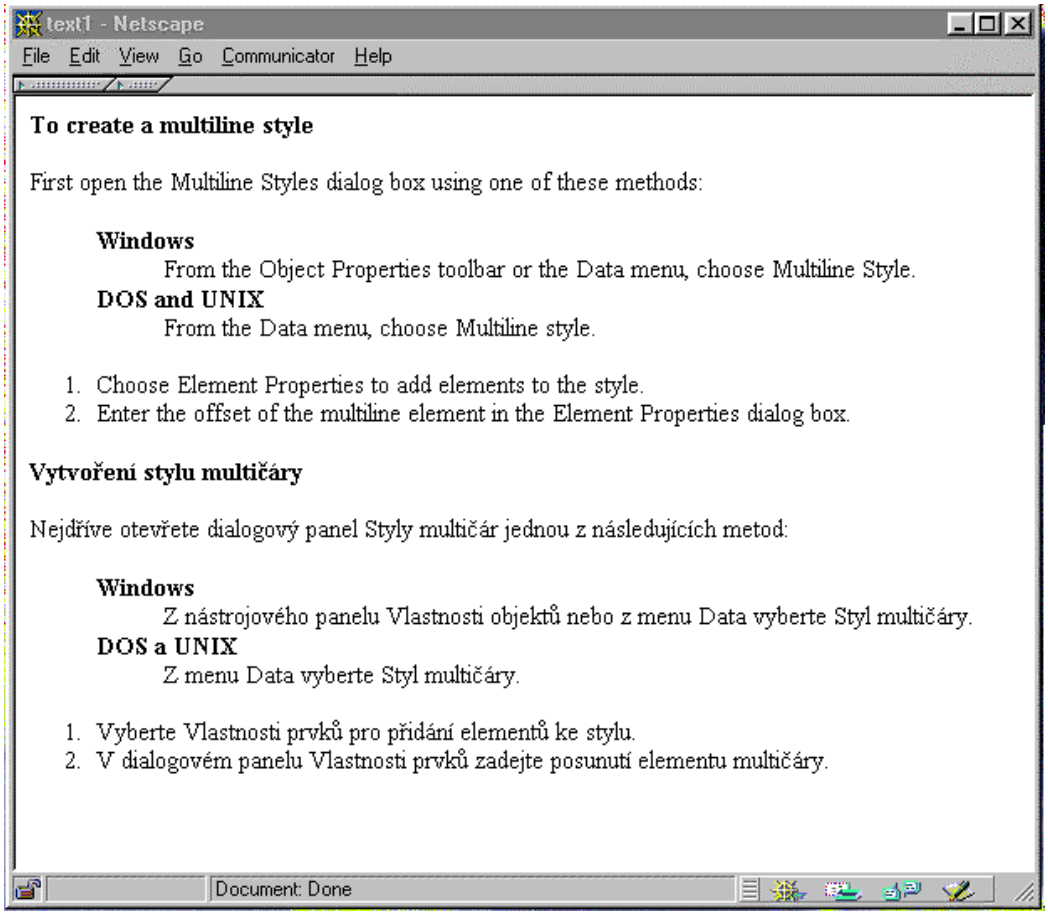


Figure 7 - Example of text realisation

3. From text plan to sentence plans

After we have obtained a text plan that includes constraints on how its parts are to be realised, and associations between its text structure elements and the parts of the A-box they correspond to, we need to take one further step before we can invoke the tactical generator. The aim of this step is to create SPLs, using the associations to parts of the A-box and the realisation constraints. It is the task of the Sentence Planner to fulfil this aim.

To that end, for each of the given parts of the A-box we commence by setting up ideational SPL code that realises the content as expressed by that part of the A-box. The code is ideational in the sense that it expresses only the content and nothing more. Therefore, once we have ideational SPL code, we need to render the constraints imposed by the text structure (in particular, by the text template) into terms of SPL code, and combine the outcome with the ideational part. The resulting SPL then in principle realises the content in a way as prescribed by the text template (and the general realisation conditions). The final outcome of this step is a set of SPLs that specify the realisation of the A-box in terms of the generated text structure. In order to obtain the text, then, we run this set of SPLs through a tactical generator.

There are several issues pertaining to generating sentence plans. We address two important ones in the next sections: *aggregation* (§3.1), and *information structure* (§3.2).

3.1 Aggregation and sentence boundaries

Aggregation is the task of combining two or more fragments of SPL code into a larger piece of SPL code. Every fragment specifies the plan for realising an individuated part of the A-box, possibly in a pre-determined style. From the viewpoint of realisation, each of these SPL fragments essentially specify *clauses*. In other words, when we speak of combining SPLs, we are essentially deciding about the complexity of the sentences that are to realised the content specified by the SPL code.

In a sense the problem of aggregation has a complementary problem, and that is the problem of deciding where *sentence boundaries* should be.[3] For example, if we determine that sentence boundaries should be between Tasks and Instructions, a new sentence is started whenever either a Task or an Instruction needs to be realised. In terms of SPL code that means the following. Text structure elements follow the way content is configured hierarchically in the A-box. Therefore, when we realised parts of this content by recursively descending through the A-box, inducing a sentence boundary means abandoning the recursive descent and beginning anew, starting at the point where we left off. Consequently, all the fragments of SPL code constructed up to the point where the recursion was halted will be combined (aggregated) into a single SPL.

For the purpose of the intermediate prototype we establish the following criteria for inducing a halt (resulting eventually in realising a sentence boundary). We stop recursively descending the A-box whenever we encounter:

- A PROCEDURE (which has as slots a Goal, Methods, and a Side-Effect), or

[3] The problems of aggregation and sentence boundaries are complementary in the sense that if you resolve one, the other one is settled as well.

- A METHOD* (which has as slots a Constraint, a Precondition, and Substeps)

To organise the SPLs that are being combined for realising an individual PROCEDURE or METHOD, we employ the discourse relations as indicated earlier (§2.6) in the manner described there (particularly, see Figure 2)

3.2 Determining coherent sentential information structure

In section 2.6 we already discussed how we can add discourse relations to a text plan, based on the (logical) relations that can be conceived to exist between various text structure elements. Discourse relations are needed if, in the end, we want to generate a smooth, *coherent* text.

To obtain coherence, we need to be able to generate sentences that reflect a proper information structure. As we have already discussed at length in (AGILE 6.2), Slavonic languages reflect information structure by means of word order. How we can generate a particular word order given an information structure to be realised is discussed in (AGILE 7.2). The issue we have to address in the context of the Text Structuring Module is thus *how* we could obtain information structures.

In (Kruijff & Schaake, 1997) it is shown how one can perceive of RST-style discourse relations (like the ones in section 2.6) as having a corresponding *chain* of information structures[4]. That is to say, the satellite and the nucleus of the RST discourse relation each are realised using a particular information structure, and are coherently related by virtue of how their individual topics and/or foci can be connected. This idea finds its roots in the work by Daneš, see also (Korbayová & Kruijff, 1996).

For the purposes of the Text Structuring Module, we can make use of this idea as follows. Once we have determined what discourse relation should hold between two text structure elements, we can require each of these text structure elements to be realised with the appropriate information structures. During sentence planning, these constraints on realisation are transferred to the sentence plans specifying the sentences realising these text structure elements. The grammar will then be able to generate the proper word order.

We defer full treatment of these issues until TEXM3. We conclude here by noting that, due to the close connections between the treatment of information structure in Systemic Functional Grammar and as in the work by Daneš, the approach outlined above seems a natural extension of KPML and the Text Structuring Module as described in this deliverable.

3.3 Implementation of the sentence planner

Once a text plan has been constructed from the A-box (see for example Section 4.2 below), the Sentence Planner processes the *leaves* of the text plan in order to create the SPL code for the semantics identified by those leaves.

The first step we take in order to create sentence plans is to translate the semantics into purely ideational SPL code – i.e. code that just specifies the realisation of a particular piece of semantics. As we already pointed out above, sentence boundaries are determined following the configuration of the content as well. This means that the chunks of ideational SPL code we are generating in this step do already correspond to sentences.

---

[4] In (Kruijff & Schaake, 1997) the term “sequence” is used rather than “chain”. The latter term is borrowed here from (Korbayová & Kruijff, 1996).

Because the sentence plans generated up to this point only specify *what* should be realised, but not *how* that content should be realised, we need to go one step further. During text planning we have already determined various aspects of style, following the specifications in (AGILE 5.1). In the text plan these aspects of style are indicated using KPML's REALIZE-WITH statements. After the sentence planner has determined the ideational sentence plans, we now continue by transferring the constraints on stylistic realisation to the sentence plans. As a result, the sentence plans not only show *what* should be realised, but also *how* that content should be presented.

In the next section we present two examples showing how one obtains a set of sentence plans from a text plan (based on an A-box).

4. End-to-end generation in the intermediate prototype

In this section we provide two short examples showing of generation in the intermediate prototype, from the A-box down to SPLs. We have created two A-boxes underlying texts that are shortened versions of target text 1 for the intermediate prototype (cf. (AGILE 5.2) for the full versions of all the target texts). These two A-boxes have been constructed with the Text Structuring Module in mind. Although their corresponding texts do not show the full range of linguistic phenomena as covered by the tactical generators, they do explore the potential variety of text structures.

4.1 The A-boxes and their (target) texts

Below we first give the A-boxes, then their target texts.

A-box 1

```
PROCEDURE a1
GOAL:
  CREATE a2
  ACTEE:
    CONFIGURED-ACTEE a3
    RANGE:
      STYLE a4
      OWNER:
        MULTILINE a5
  ACTOR:
    USER-ACTOR a6
    RANGE:
      USER a7
METHODS:
  METHOD-LIST a8
  FIRST:
    METHOD* a9
    CONSTRAINT:
      OPERATING-SYSTEM a10
      LABEL:
        Unix
    PRECONDITION:
      PROCEDURE a11
      GOAL:
        OPEN-SCREEN-OBJECT a12
        ACTEE:
          DISPLAYED-ACTEE a13
          RANGE:
            DIALOGUE-BOX a14
            LABEL:
              X
        ACTOR:
          USER-ACTOR a15
          RANGE:
            USER a7
    SUBSTEPS:
      PROCEDURE-LIST a16
```

A-box 2

```
PROCEDURE a1
GOAL:
  ADD-TO a2
  ACTEE:
    CONFIGURED-ACTEE a3
    RANGE:
      ELEMENTS a4
  RECIPIENT:
    CONFIGURED-ACTEE a5
    RANGE:
      STYLE a6
      OWNER:
        MULTILINE a7
  ACTOR:
    USER-ACTOR a8
    RANGE:
      USER a9
METHODS:
  METHOD-LIST a10
  FIRST:
    METHOD* a11
    SUBSTEPS:
      PROCEDURE-LIST a12
      FIRST:
        PROCEDURE a13
        GOAL:
          CHOOSE a14
          ACTEE:
            DISPLAYED-ACTEE a15
            RANGE:
              OPTION a16
              LABEL:
                Element Properties
          OPTIONS:
            GUI-ACTEE a17
          ACTOR:
            USER-ACTOR a18
```

```
      FIRST:
        PROCEDURE a17
        GOAL:
          CHOOSE a18
          ACTEE:
            DISPLAYED-ACTEE a19
            RANGE:
              OPTION a20
              LABEL:
                Element Properties
          OPTIONS:
            GUI-ACTEE a21
          ACTOR:
            USER-ACTOR a22
            RANGE:
              USER a7
      REST:
        PROCEDURE-LIST a23
        FIRST:
          PROCEDURE a24
          GOAL:
            ADD-TO a25
            ACTEE:
              CONFIGURED-ACTEE a26
              RANGE:
                ELEMENTS a27
            RECIPIENT:
              CONFIGURED-ACTEE a28
              RANGE:
                STYLE a4
                OWNER:
                  MULTILINE a5
            ACTOR:
              USER-ACTOR a29
              RANGE:
                USER a7
```

```
              RANGE:
                USER a9
            SIDE-EFFECT:
              SEE a19
              SENSER:
                SEE-SENSER a20
                RANGE:
                  USER a9
              PHENOMENON:
                SEE-PHENOMENON a21
                RANGE:
                  DIALOGUE-BOX a22
                  LABEL:
                    Element Properties
          REST:
            PROCEDURE-LIST a23
            FIRST:
              PROCEDURE a24
              GOAL:
                ENTER a25
                ACTEE:
                  CONFIGURED-ACTEE a26
                  RANGE:
                    OFFSET a27
                    OWNER:
                      STYLE-ELEMENT a28
                      OWNER:
                        MULTILINE a7
                LOCATION:
                  GUI-ACTEE a29
                ACTOR:
                  USER-ACTOR a30
                  RANGE:
                    USER a9
              REST:
                PROCEDURE-LIST a31
                FIRST:
                  PROCEDURE a32
                  GOAL:
                    CHOOSE a33
                    ACTEE:
                      DISPLAYED-ACTEE a34
                      RANGE:
                        OPTION a35
                        LABEL:
                          Add
                    OPTIONS:
                      GUI-ACTEE a36
                    ACTOR:
                      USER-ACTOR a37
                      RANGE:
                        USER a9
                  SIDE-EFFECT:
                    SEE a38
                    SENSER:
                      SEE-SENSER a39
                      RANGE:
                        USER a9
                    PHENOMENON:
                      SEE-PHENOMENON a40
                      RANGE:
                        DIALOGUE-BOX a41
                        LABEL:
                          Select Color
```

The target text for the first A-box is as follows:

To create a multiline style

UNIX OS: Open Dialog Box X

1. Choose Element Properties
2. Add elements to style

The target text for the second A-box is as follows:

To add elements to a style

1. Choose Element Properties. The Element Properties dialog box appears.
2. Enter the offset of the multilane element
3. Select Add. The Select Color dialog box appears

4.2 Generated text plans and SPLs

4.2.1 Text plan and SPLs for test-text 1

The text plan for text 1 is as follows:

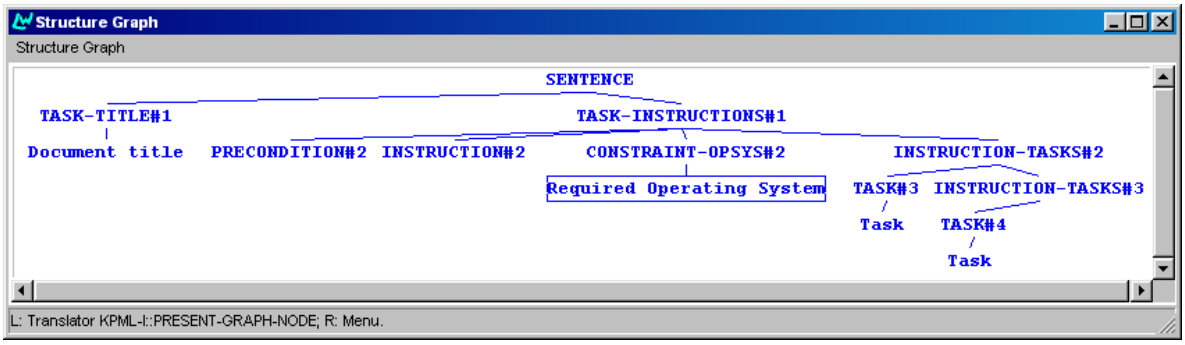


Figure 8 - Text plan for test text 1

The SPLs are as follows:

```
((|a2| / DM::CREATE :EXIST-SPEECH-ACT-Q NOSPEECHACT :PROCESSUAL-Q
      PROCESSUAL :ACTEE
      (|a4| / DM::STYLE :PART-OF
            (|a5| / DM::MULTILINE))
      :ACTOR
      (HEARER / DM::USER :IDENTIFIABILITY-Q DM::IDENTIFIABLE))
 (|a12| / DM::OPEN-SCREEN-OBJECT :ACTEE
       (|a14| / DM::DIALOGUE-BOX :CLASS-ASCRIPTION
              (G5580 / DM::OBJECT :NAME X))
       :ACTOR
       (HEARER / DM::USER :IDENTIFIABILITY-Q DM::IDENTIFIABLE))
 (|a9| / DM::METHOD* :SOURCE
```

```
        (|a10| / DM::OPERATING-SYSTEM :CLASS-ASCRIPTION
               (G5581 / DM::OBJECT :NAME UNIX))
        :PRECONDITION
        (|a12| / DM::OPEN-SCREEN-OBJECT :ACTEE
               (|a14| / DM::DIALOGUE-BOX :CLASS-ASCRIPTION
                      (G5582 / DM::OBJECT :NAME X))
               :ACTOR
               (HEARER / DM::USER :IDENTIFIABILITY-Q DM::IDENTIFIABLE))
        :SUBSTEPS
        (|a18| / DM::CHOOSE :ACTEE
               (|a20| / DM::OPTION :CLASS-ASCRIPTION
                      (G5583 / DM::OBJECT :NAME ELEMENT\ PROPERTIES))
               :ACTOR
               (HEARER / DM::USER :IDENTIFIABILITY-Q DM::IDENTIFIABLE)))
(|a25| / DM::ADD-TO :ACTEE
       (|a27| / DM::ELEMENTS)
       :RECIPIENT
       (|a4| / DM::STYLE :PART-OF
             (|a5| / DM::MULTILINE))
       :ACTOR
       (HEARER / DM::USER :IDENTIFIABILITY-Q DM::IDENTIFIABLE))
(|a10| / DM::OPERATING-SYSTEM :CLASS-ASCRIPTION
       (G5584 / DM::OBJECT :NAME UNIX))
(|a18| / DM::CHOOSE :ACTEE
       (|a20| / DM::OPTION :CLASS-ASCRIPTION
              (G5585 / DM::OBJECT :NAME ELEMENT\ PROPERTIES))
       :ACTOR
       (HEARER / DM::USER :IDENTIFIABILITY-Q DM::IDENTIFIABLE))
(|a25| / DM::ADD-TO :ACTEE
       (|a27| / DM::ELEMENTS)
       :RECIPIENT
       (|a4| / DM::STYLE :PART-OF
             (|a5| / DM::MULTILINE))
       :ACTOR
       (HEARER / DM::USER :IDENTIFIABILITY-Q DM::IDENTIFIABLE)))
```

4.2.2 Text plan and SPLs for test-text 2

The text plan for text 2 is as follows:

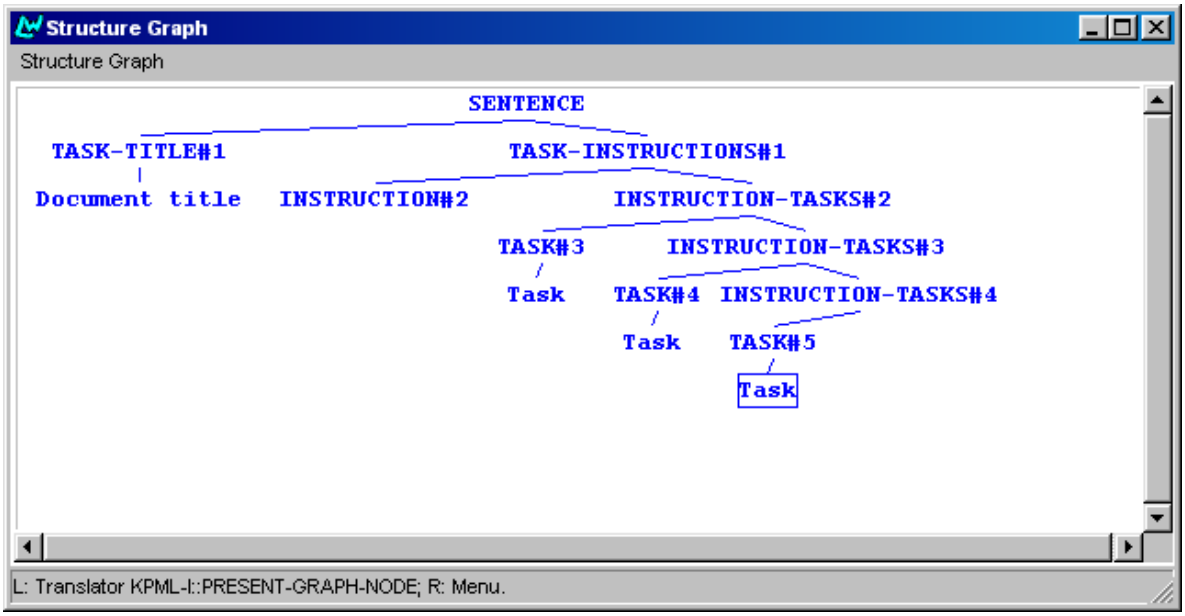


Figure 9 - Text plan for test text 2

The SPLs that the sentence planner generates, on the basis of the text plan for text 2, are the following.

```
((|a2| / DM::ADD-TO :EXIST-SPEECH-ACT-Q NOSPEECHACT :PROCESSUAL-Q
      PROCESSUAL :ACTEE
      (|a4| / DM::ELEMENTS)
      :RECIPIENT
      (|a6| / DM::STYLE :PART-OF
            (|a7| / DM::MULTILINE))
      :ACTOR
      (HEARER / DM::USER :IDENTIFIABILITY-Q DM::IDENTIFIABLE))
 (|a11| / DM::METHOD* :SUBSTEPS
       (|a14| / DM::CHOOSE :ACTEE
              (|a16| / DM::OPTION :CLASS-ASCRIPTION
                     (G5552 / DM::OBJECT :NAME ELEMENT\ PROPERTIES))
              :ACTOR
              (HEARER / DM::USER :IDENTIFIABILITY-Q DM::IDENTIFIABLE)))
 (|a25| / DM::ENTER :ACTEE
       (|a27| / DM::OFFSET :PART-OF
              (|a28| / DM::STYLE-ELEMENT :PART-OF
                     (|a7| / DM::MULTILINE)))
       :ACTOR
       (HEARER / DM::USER :IDENTIFIABILITY-Q DM::IDENTIFIABLE))
 (|a33| / DM::CHOOSE :ACTEE
       (|a35| / DM::OPTION :CLASS-ASCRIPTION
              (G5553 / DM::OBJECT :NAME ADD))
       :ACTOR
       (HEARER / DM::USER :IDENTIFIABILITY-Q DM::IDENTIFIABLE))
 (|a38| / DM::SEE :SENSER
       (HEARER / DM::USER :IDENTIFIABILITY-Q DM::IDENTIFIABLE)
       :PHENOMENON
```

```
        (|a41| / DM::DIALOGUE-BOX :CLASS-ASCRIPTION
               (G5554 / DM::OBJECT :NAME SELECT\ COLOR)))
(|a19| / DM::SEE :SENSER
       (HEARER / DM::USER :IDENTIFIABILITY-Q DM::IDENTIFIABLE)
       :PHENOMENON
       (|a22| / DM::DIALOGUE-BOX :CLASS-ASCRIPTION
              (G5555 / DM::OBJECT :NAME ELEMENT\ PROPERTIES)))
(|a14| / DM::CHOOSE :ACTEE
       (|a16| / DM::OPTION :CLASS-ASCRIPTION
              (G5556 / DM::OBJECT :NAME ELEMENT\ PROPERTIES))
       :ACTOR
       (HEARER / DM::USER :IDENTIFIABILITY-Q DM::IDENTIFIABLE))
(|a19| / DM::SEE :SENSER
       (HEARER / DM::USER :IDENTIFIABILITY-Q DM::IDENTIFIABLE)
       :PHENOMENON
       (|a22| / DM::DIALOGUE-BOX :CLASS-ASCRIPTION
              (G5557 / DM::OBJECT :NAME ELEMENT\ PROPERTIES)))
(|a25| / DM::ENTER :ACTEE
       (|a27| / DM::OFFSET :PART-OF
              (|a28| / DM::STYLE-ELEMENT :PART-OF
                     (|a7| / DM::MULTILINE)))
       :ACTOR
       (HEARER / DM::USER :IDENTIFIABILITY-Q DM::IDENTIFIABLE))
(|a33| / DM::CHOOSE :ACTEE
       (|a35| / DM::OPTION :CLASS-ASCRIPTION
              (G5558 / DM::OBJECT :NAME ADD))
       :ACTOR
       (HEARER / DM::USER :IDENTIFIABILITY-Q DM::IDENTIFIABLE))
(|a38| / DM::SEE :SENSER
       (HEARER / DM::USER :IDENTIFIABILITY-Q DM::IDENTIFIABLE)
       :PHENOMENON
       (|a41| / DM::DIALOGUE-BOX :CLASS-ASCRIPTION
              (G5559 / DM::OBJECT :NAME SELECT\ COLOR))))
```

5. Future research

Recapitulating, in this deliverable we have described the AGILE Text Structuring Module. This module takes as input an A-box specifying the content of a text. It outputs an ordered set of sentence plans that, once provided to a tactical generator, are realised as a sequence of sentences together making up the text realising the content given by the A-box. Internally, the Text Structuring Module consists of a Text Planner and a Sentence Planner. The Text Planner constructs a text plan, based on the way content is configured in the A-box. The text plan is constructed from text structure elements that each correspond to individual parts of content of the A-box. To reflect choices of style, text structure elements may be required to be realised in a particular way. The Sentence Planner takes the text plan as input, and creates a set of sentence plans. Each sentence plan specifies what content needs to be realised, and how that should be done.

Naturally, the approach we take, using text structure elements and text templates, is but one possible approach in a spectrum of many. As observed in (Reiter and Dale, 1997), the task of planning a coherent text can be performed using for example on of the following approaches.

- *Discourse relations forming a hierarchical structure*: Probably the most commonly used set of discourse relations is that suggested by Mann and Thompson's *Rhetorical Structure Theory* (1987). The Penman Upper Model also contains discourse relations modelled after RST – see (Bateman et al, 1990), and (Hovy, 1993) for related discussion.
- *Planning techniques*: Planning techniques as such is a highly developed area within Artificial Intelligence, and it is conceivable that one day such techniques could be used within natural language generation. However, as both Hovy (1993) and Reiter and Dale (1997) note, we still have a very limited understanding of how discourse can be structured, and when particular relations should be used. This fact, coupled to the fact that AI planning techniques are computationally expensive, means that the planning-based approach is not widely used in NLG.
- *Schema-based approaches*: Schemas are predefined "patterns" that can be used to build a text from by filling in the slots specified by the schema with particular pieces of content defined by the A-box. Whenever a domain is well-defined and limited, it is possible to construct a limited set of schemas that a text planner can use. See also (Hovy, 1993) for a general discussion.

Compared to the listed approaches, our approach is closely related to the *discourse relations* one. Leaving the planning-based approach out of the equation for the reasons given above, we could also have opted for a schema-based approach. However, we agree with Moore and Paris (1993) that schemas prove to be insufficient if the domain, even when it is limited and well-defined, allows for a very flexible definition of its goals (i.e. our A-boxes). Schemas are simply too rigid in such a case – and given the flexibility allowed in configuring content in an A-box, we find that text structure elements provide a more gentle approach.

Furthermore, by combining the ideas of text structure elements and text templates, we obtain the possibility to generate a variety of texts in alternative styles. This goes back to a point made in the conclusions at the end of (AGILE 5.1). There, on page 25, we mentioned that we would concentrate (among other things) on a model enabling a more flexible generation of alternative forms and on those aspects of text planning necessary to decide whether a given content should be expressed by a sequence of simplex clauses or by a complex clause. It should be clear that the first point is conveniently captured by the idea of text templates as we defined it above. The second point is covered by the (template-independent) general constraints on realisation.

Finally, what should be clear from putting the Text Structuring Module on top of a tactical generator is that it will be very easy to share information between the high-level task of strategic generation (text planning, sentence planning) and the actual tactical generation of individual sentences. The reason is that both tasks are performed within the KPML environment. The so-called generation gap can thus be minimised since the systems in the CADCAM-INSTRUCTIONS region are in this case able to exert control over the way the grammar will realise a sentence.

Among the topics for further research are the realisation of particular information structures based on discourse relations that can be discerned between different parts of a text plan (see section 3.2). Closely related to the issue of information structure is the generation of referring expressions.

Appendix A. Running the Text Structuring Module

In this document we describe the various components that make up the Text Structuring Module for the intermediate prototype:

- The CADCAM-INSTRUCTIONS region, and the revised RANKING region (§2 and particularly §2.7)
- The text planner function (§ 2.7)
- The sentence planner (§ 3, and in particular §3.3)

In order to run the Text Structuring Module in conjunction with a tactical generator, the following steps need to be taken:

1. Load the KPML environment.
2. Load the linguistic resources for the tactical generator.
3. Load the linguistic resources for the Text Structuring Module. For example, if these resources are located in a directory "`tsm`" that is located under "`c:\agile\kpml2_0\resources`" then you load the resources by issuing the following command:

   ```
   (kpml::load-linguistic-resources :tsm :root-directory
   "c:\\agile\\kpml2_0\\resources" :clear nil)
   ```

   The "`:clear nil`" statement ensures that no linguistic resources that were loaded prior to the text structuring module, will be lost.
4. Load the domain model by loading `agile.lisp`, which is located in the directory where the domain model files are. (Note: It is important that the latest version of the domain model is used. This version is dated February 1999, and can be downloaded from ITRI's AGILE website. See the instructions there for how to install the domain model.)
5. Load the text planner by loading `tsm2_2.lisp`.
6. Load the sentence planner by loading `splize.lisp`.

After having loaded these files, in the sequence as stated, the Text Structuring Module is ready. The next steps that need to be taken to generate a particular text are the following:

1. Load the A-box specifying the content that the text should realised.
2. Start the Text Structuring Module by calling `(kpml::text-planner ())` which will start the text planner. The text planner uses (in this case) the currently loaded A-box, and will call the sentence planner after it has created a text plan.
3. Call a tactical generator (for a specific language, say Czech) with the SPLs that the Text Structuring Module has produced:

   ```
   (kpml::say `(<SPLs>) :language Czech :preselects `(genre))
   ```

   The tactical generator should now generate the sentences as specified by the SPLs, in the order as the SPLs are given. The sequence of generated sentences makes up the text that realizeds the content given by the A-box we started with.

References

[AGILE 2.1] R. Power, *Preliminary model of the CAD/CAM domain.* AGILE deliverable 2.1, June 1998

[AGILE 5.1] J. Bateman, A. Hartley, I. Kruijff- Korbayová, D. Dochev, N. Gromova, J. Hana, S. Sharoff, L. Sokolova, *Generation of simple text structures in Bulgarian, Czech and Russian*. AGILE deliverable 5.1, June 1998. (Deliverable comprises TEXS1-Cz, TEXS1-Bu, TEXS1-Ru, TEXM1.)

[AGILE 5.2] I. Kruijff-Korbayová, G.J.M. Kruijff, J. Bateman, D. Dochev, N. Gromova, A. Hartley, E. Teich, S. Sharoff, L. Sokolova, and K. Staykova, *Specification of elaborated text structures*. AGILE deliverable 5.2, April 1999. (Deliverable comprises TEXS2-Cz, TEXS2-Bu, TEXS2-Ru)

[AGILE 6.2] E. Adonova, J. Bateman, N. Gromova, A. Hartley, G.J.M. Kruijff, I. Kruijff-Korbayová, S. Sharoff, H. Skoumalová, L. Sokolova, K. Staykova, and E. Teich, *Formal specification of extended grammar models*. AGILE deliverable 6.2, March 1999 (Deliverable comprises LSPEC2-Cz, LSPEC2-Ru, LSPEC2-Bu)

[Grosz and Sidner, 1986] Barbara J. Grosz and Candace L. Sidner, "Attention, intention, and the structure of discourse." *Computation Linguistics,* **12** (3): 175-204.

[Hovy, 1993] Eduard H. Hovy, "Automated Discourse Generation Using Discourse Structure Relations". *Artificial Intelligence* **63**: 341-385, 1993. Reprinted in Barbara J. Grosz and Fernando C.N. Pereira (eds.), *Natural Language Processing*, The MIT Press: Cambridge MS, 1994

[Korbayová and Kruijff, 1996] I. Korbayová and G.J.M. Kruijff, "Identification of topic-focus chains". In S.Botley, Glass J., T.McEnery, and A.Wilson (eds), *Approaches to Discourse Anaphora: DAARC96.* No. 8, Technical papers, University Centre for Computer Corpus Research on Language, University of Lancaster, 1996, 165—179.

[Kruijff and Schaake, 1997] G.J.M. Kruijff and J. Schaake, "Discerning Relevant Information in Discourses Using TFA". In R. Mitkov and N. Nicolov (eds.), *Recent Advances in Natural Language Processing*. John Benjamins: Amsterdam, Philadelphia, 1997, 259-270.

[Mann and Thompson, 1987] William Mann and Susan Thompson, "Rhetorical structure theory".

[Moore and Paris, 1993] Johanna D. Moore and Cécile L. Paris, "Planning text for advisory dialogues: Capturing intentional and rhetorical information." *Computational Linguistics*, **19** (4): 651-694, 1993

[Moore and Pollack, 1992] Johanna D. Moore and Martha E. Pollack, "A Problem for RST: The Need for Multi-Level Discourse Analysis". *Computational Linguistics*, **18** (4): 537-544, 1992

[Reiter and Dale, 1997] Ehud Reiter and Robert Dale, "Building Applied Natural Language Generation Systems". *Natural Language Engineering*, **1** (1), 1997

[Reiter, Mellish and Levine, 1995] Ehud Reiter, Chris Mellish and John Levine, "Automatic Generation of Technical Documentation". *Applied Artificial Intelligence*, **9**, 1995