AGILE

Automatic Generation of Instructions in Languages of Eastern Europe

---

Title ***Generation of simple text structures for instructions in Bulgarian, Czech and Russian***

Authors John Bateman
Tony Hartley
Ivana Kruijff-Korbayová
Danail Dochev
Nevenna Gromova
Jiøí Hana
Sergei Sharoff
Elena Sokolova

Deliverable *TEXS1-Bu, TEXS1-Cz, TEXS1-Ru, TEXM1*

Status *Final*

Availability *Public*

Date *June 1998*

INCO COPERNICUS PL961104

Abstract:

This report presents the results of the first task in the Agile project work package concerned with text structuring. It comprises the specifications of the simple text structures in Bulgarian, Czech and Russian, and an overview of the implementation of the Initial Demonstrator based on these specifications. The texts considered for generation in this stage are simplified versions of routine passages occurring frequently the user guide of the AutoCAD system, namely descriptions of step-by-step procedures for performing a task.

In the application scenario of the system developed in Agile, the user provides a definition of a text content in the form of an A-box. The text structurer yields a set of SPLs for the sentences to be generated to convey this content, and then the tactical generators for the three languages take the SPLs as input, and generate the corresponding sentences.

We decided to employ text templates corresponding to particular text styles as the guidance in text planning. A text template consists of pre-redefined text structure elements, which are mapped to layout rules, A-box configurations and syntactic realizations. We specify the text structure elements and the mappings for English and for the three languages, Bulgarian, Czech and Russian, for the texts of the Initial Demonstrator.

The current text structuring is simplified in a number of respects. We are dealing with A-box configurations of restricted complexity and with a restricted range of syntactic constructions. We have also not yet tackled the determination of the identifiability feature and Theme-Rheme structure.

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
f	+44-1273-642908

Table of Content

Table of Figures

1. Introduction

This report comprises the deliverables TEXS1-Bu, TEXS1-Cz, TEXS1-Ru which are concerned with the specification of the simple text structures in Bulgarian, Czech and Russian, respectively, generated by the Initial Demonstrator. This report also provides an overview of the deliverable TEXM1 which consists of the corresponding software module implemented according to the specifications.

The work described in this deliverable has used as input the work done within WP3 concerned with corpus analysis.

The development of the system for text generation in the AGILE project follows up on the experience gained in the development of the DRAFTER system for English, and also adopts its essential architecture, as specified in [Agile project 1997, Pemberton et al. 1996]. The system is intended for use in a scenario which involves a user, i.e. a technical writer or a domain expert producing a CAD-CAM software user guide or manual, on the basis of his/her knowledge of the software product, esp. its functionality and user interface. The user defines the content of the texts through a user interface tailored to the CAD-CAM domain, and then the system generates the corresponding texts in Bulgarian, Czech and Russian automatically.

Taking into account the current potential of the natural language generation technology in general, one cannot at present aim to generate texts with wide-ranging conceptual content or advanced stylistic features. Rather, the texts considered for generation by the system developed in AGILE first of all correspond to routine passages, which occur frequently in the user guides and manuals, have a quite simple semantic content and have a regular style, terminology and structure [Power et al. 1994, Power and Scott 1997]. As expected on the basis of the experience in DRAFTER, the preliminary text analysis of the user guide of the AutoCAD system performed within the WP3 of AGILE revealed that these criteria are satisfied by descriptions of step-by-step procedures for performing a task. A corpus of such texts has been assembled and studied, as described in the deliverable "Corpus analysis and preparation" of WP3. On the basis of this corpus, a set of simple text structures has been identified. These were chosen to be generated for the purpose of the Initial Demonstrator.

The definition of a text content by the user in the system developed in AGILE is formulated as a so-called A-box (see the deliverable "Domain modelling" of WP2). The tactical generators for the three languages take SPLs as input, and generate the corresponding sentences (see the deliverable "Grammatical Resource implementation for Bulgarian, Czech and Russian" of WP7).

The task of the text structuring module which is described in this deliverable, is thus to take the user's definitions of the text content as an input, and to yield a set of SPLs which determine how this content is to be conveyed in a text in a given style. This task involves macro-planning of the text, the splitting up of the content into chunks that can be expressed by sentences and also determination of the particular ways of expressing the content, depending on the surrounding context. These are all complex issues, which we have to handle in an incremental fashion in the successive stages of the project.

Given the text structures of the Initial Demonstrator, the strategy chosen for the sake of their generation can be quite simple. However, in order to be extendible in the next stages of the project, is has to be sufficiently general and flexible. We decided to employ text templates corresponding to particular text styles as the guidance in text planning. Each text

template has pre-redefined components which are filled by content from particular parts of the user's definition of the text content, and for which the typical ways of expressing the content in a given language are specified for the template. Furthermore, one can associate layout rules with each component of the text template, so that the desired formatting and type-setting of the generated text can be obtained when the text is printed or displayed as the output of the system.

The rest of this deliverable is organized as follows. Before presenting the text structuring rules for the Initial Demonstrator for the three languages, we discuss the underlying issues of text structuring in the Initial Demonstrator in more detail. We first demonstrate the text structures considered in the Initial Demonstrator in Section 0 using an English version of the Initial Demonstrator texts. Then we describe briefly the correlations between the text structures and the layout/format of the texts in Section 0. In Section 0, we describe the configurations of the A-boxes as the text content definitions we are dealing with in this phase of the AGILE project.

Then we turn to the specifications of the Initial Demonstrator text structures. The common core of specifications at a higher level of text structuring that is shared by the three languages is provided in Section 0. We define text structure elements (Section 0), their mapping to layout rules (Section 0), and their mapping to A-box configurations (Section 0).

The language specific syntactic realization rules for Bulgarian, Czech and Russian are provided in Section 0., respectively.

The specifications provided in Section 0. have been implemented within the Initial Demonstrator (Section 0). In Section 0, we describe the approach taken in the implementation, and in Section 0, we illustrate how the implementation works.

We conclude this report by a brief summary, accompanied by a discussion of the limitations of the current text structuring module, and a sketch of the improvements that we want to make in the next stage of the project (Section 0).

1.1 Overview of the Texts

We show below the English equivalent of the amended versions of the Initial Demonstrator texts, and describe briefly and informally the text structures encountered in them.

1.1.1 Text 1 (adapted from AutoCAD p. 45)

To draw a polyline

- Start the PLINE command by choosing Polyline from the Polyline flyout on the Draw toolbar.
- Specify the first point of the polyline.
- Specify the endpoint of the polyline.
- End the polyline by pressing Return.

1.1.2 Text 2a (adapted from AutoCAD pp. 49-50)

To draw a multiline

- Start the MLINE command by choosing Multiline from the Polyline flyout on the Draw toolbar.

- Select a style by entering **st** at the prompt.
- Display the style list by entering **?**
- Justify the multiline.
    - Enter **j**.
    - Choose a justification.
- Change the scale of the multiline.
    - Enter **s**.
    - Enter a scale.
- Specify the multiline.
    - Specify the first point of the multiline.
    - Specify the second point of the multiline.
    - Specify the third point of the multiline.
- End the multiline by pressing Return.

1.1.3 Text 2b (adapted from AutoCAD pp. 49-50)

To draw a multiline

- Start the MLINE command by choosing Multiline from the Polyline flyout on the Draw toolbar.
- Enter **st** at the prompt.
- Select a style.
    - Enter **?**.
    - The AutoCAD Text Window appears.
    - Enter the name of the style.
- Change the justification of the multiline.
    - Enter .
    - Enter a justification from top, zero and bottom.
- Change the scale of the multiline.
    - Enter **s**.
    - Enter a number.
- Close the AutoCAD Text Window.
- Specify the first point of the multiline.
- Specify the second point of the multiline.
- Specify the third point of the multiline.
- End the multiline by pressing Return.

Each text consists of a header line (heading), and of a list of steps. A list item can also be followed by an embedded sub-list. There is no more than this one level of embedding in the Initial Demonstrator texts. In our current version of the texts, the lists at both levels are bullet lists. In the original version from the manual, the list at the first level is numbered. The text structures are the same, bulletting versus numbering is an issue of layout.

The realization of the content of the heading varies for different languages. Whereas in English the heading takes the form of a nonfinite dependent purpose clause (*to draw a line*), in Bulgarian and Czech the form is a nominalization (Bu. *chertane na polilinia*, Cz. *kreslení èáry*). In Russian, both forms are acceptable, either the infinitival dependent purpose clause like in English (*chtoby narisovatj liniju*) or the nominalization like in Bulgarian and Czech (*risovanie linij*).

The realization of the step-by-step procedures in the itemized lists is the same as in English in all three languages, and it takes the form of a non-dependent imperative clause (En. start the PLINE command, Bu. *startirajte komandata MLINE*, Cz. *spus□te pøíkaz PLINE*, Ru. *zapustite komandu PLINE*).

The descriptions sometimes also state how the procedure is to be accomplished. There are two cases:

a) If there is only one step involved, in English it is realized by a non-finite dependent means clause (*by choosing Polyline*). In Bulgarian, it is realized by a finite dependent means clause with implicit agent (*kato izberete Polyline*), in Czech by a nominalization in the instrumental case (*vybráním Polyline*), and in Russian by a nonfinite dependent means clause using an adverbial participle (*vybrav punkt Polyline*).

b) If there are two or more steps involved, their descriptions are provided in an embedded list, where each step is expressed in the same way as the steps in the top level list, i.e. by a non-dependent imperative clause (in all three languages).

Last but not least, the Initial Demonstrator texts contain a description of a side-effect of a step, which is in English realized by a finite present tense clause simplex (*The AUTOCAD Text Window appears*). The same form of expression is used also in the other languages (Bu. *Tekstovijat prozorec na Autocad se pojavjava*, Cz. *Objeví se okno AUTOCAD Text Window*, Ru. *Na ehkrane poyavitsya okno AutoCAD Text Window*).

1.2 Layout Elements

As is apparent from the sample English text above, the heading of a text as well as each list element appear on separate lines in the output. An embedded list is further indented. The heading is typeset in bold face. In order to be able to define layout rules flexibly, we define a text template for the simple style used in the Initial Demonstrator, and link the elements of his template to the elements of text structure as specified above. Layout/formatting can thus be defined independently for the elements of the text template. If we specify other text structure elements in later stages, and link them to the elements of the text template, the layout will hold for them too. Or, we can define different layout rules without having to modify the text structure elements. In this way, our specification of the various aspects of text structure can be kept separately form each other, and defined in a modular way.

So, the text template for the Initial Demonstrator texts consists of the following elements: *heading, ordered list consisting of list items, and a description list*.

1.3 A-box Configurations

The domain model used in AGILE is described in the deliverable "Domain modelling" of WP2. We used the T-box descriptions of the DRAFTER-2 project as the basis for the T-box descriptions in AGILE. For the purpose of the Initial Demonstrator, the coverage of the "end-to-end" generation system has been restricted by restricting the configurations admissible in the A-box, i.e. by restricting the possibilities for the text content definitions.

Our current description for the Initial Demonstrator does not have the recursivity of the Drafter-2 rules; they support generation to the depth of sub-substeps only. Side-effects are defined on Procedures rather than on Methods. The current definition of Method is redundant, since it has only a single slot (Precondition and Interrupt are omitted). Top-procedure is, clearly, a configuration that can be derived from the structure of the A-box.

The currently admissible A-box configurations are defined as follows:

Top-procedure

A top-procedure has two slots:

- GOAL (obligatory, filled by an action)
- TOP-METHOD (obligatory)

Top-method

A top-method has one slot:

- STEPS (obligatory, filled by a list of procedures)

Procedure

A procedure has two slots:

Either PROCEDURE1 or PROCEDURE2 (obligatory)

SIDE-EFFECT (optional, filled by an action)

Procedure1

A procedure1 has one slot:

- GOAL (obligatory, filled by an action)

Procedure2

A procedure2 has two slots:

- GOAL (obligatory, filled by an action)
- METHOD (obligatory)

Method

A method has one slot:

- Either SINGLESTEPMETHOD (obligatory, filled by an action) or MULTIPLESTEPMETHOD (obligatory, filled by a list of procedure1s)

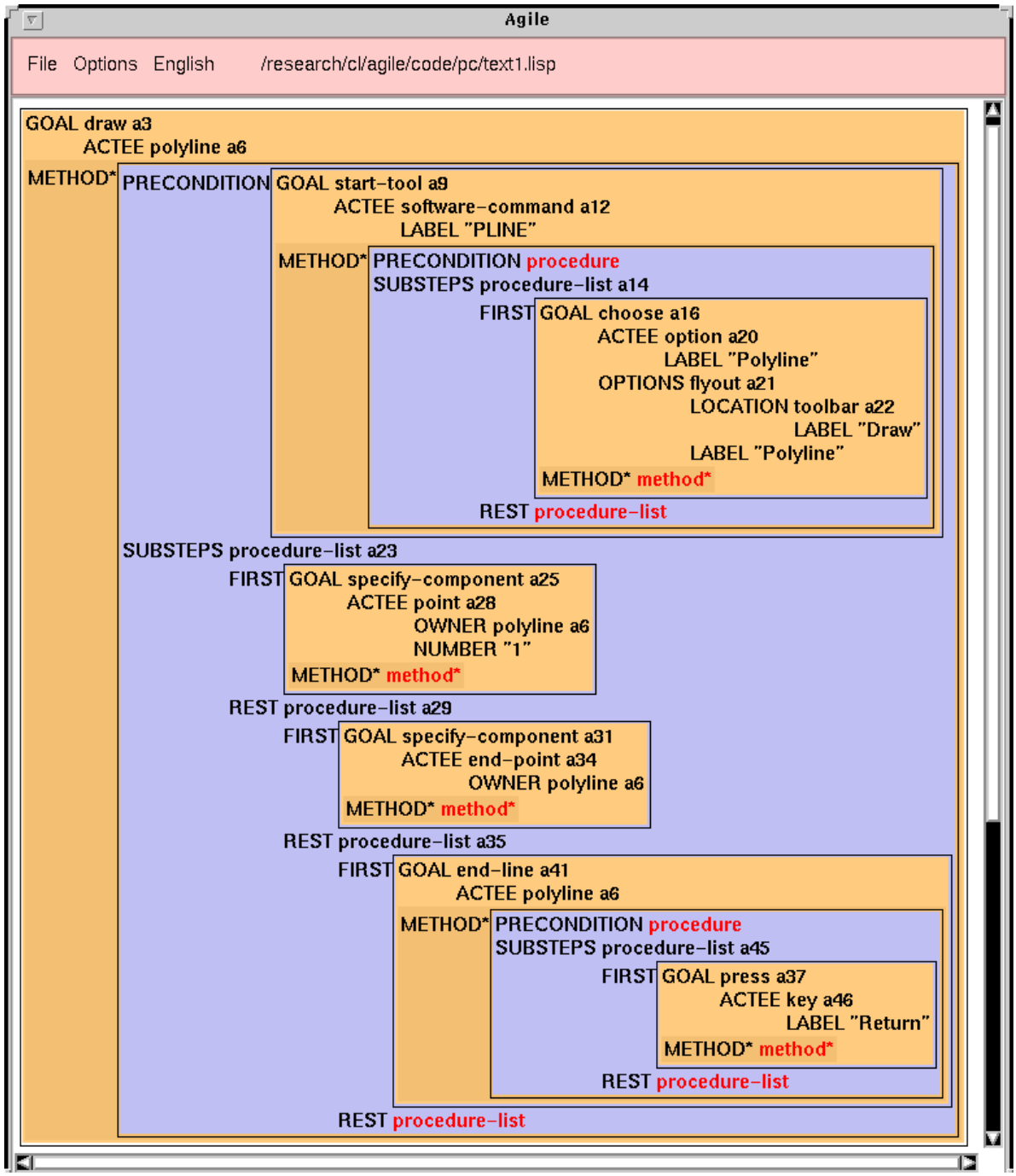
List

A list has two slots:

- FIRST (obligatory, filled by a procedure)
- REST (optional, filled by a list)

The admissible A-box configurations set the simplified semantics for the purpose of generating the amended Initial Demonstrator texts. As an illustration of an A-box configuration, Figure 1 shows the A-box for Text 1 of the Initial Demonstrator.

Figure 1: The A-box configuration for Text 1 of the Initial Demonstrator



In order to be able to define the rules for text structuring of a content defined in an A-box flexibly, we define a mapping between the T-box elements admissible in an Initial Demonstrator A-box, and the elements of text structure specified above. This mapping, which specifies the distribution of the contents of the A-box, reflects the text structuring appertaining to the Initial Demonstrator. In later stages of the project, we can define other mappings, which will correspond to more complex text structuring strategies and may require more complex planning procedures on the basis of a given A-box. The text structuring is thus defined in a modular way also with respect to the variety of T-box elements.

2. Specifications for Text Structuring: Common Core (TEXS1-Bu,Cz,Ru)

For the purposes of demonstrating "end-to-end" text generation for the three languages within WP5, we use amended versions of the target texts which were used for the stand-alone tactical generators described in the deliverable “Grammatical Resource implementation for Bulgarian, Czech and Russian” of WP7. The texts used in WP5 have been simplified and standardised. Syntactically, they are less varied (e.g. they do not contain purpose clauses). Both the syntactic realizations and the formatting stand in a direct relation to a simplified underlying semantics.

2.1 Text-Structure Elements

Let us now summarize the descriptions of the text structures used in the amended versions of the Initial Demonstrator texts using the reader-oriented terminology common in technical authoring guides.

Task-document

A task-document has two slots:

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

Task-instructions

Task-instructions has one slot:

- SUBTASKS (obligatory, filled by a list of tasks)

Task

A task has two slots:

- Either SIMPLETASK or COMPLEXTASK (obligatory)
- FEEDBACK (optional)

Complextask

A complextask has one slot:

- Either TASKANDSINGLEACTION or TASKANDMULTIPLEACTIONS (filled by a list of simpletasks)

2.2 Text Structure -- Layout Mappings

We link the template elements to the text structure elements specified above by the following mapping rules. The layout is given by HTML tags with unspecified (i.e. default) values.

- TASK-TITLE <--> HEADING level N
- TASK-INSTRUCTIONS <--> ORDERED-LIST
- SIMPLETASK <--> ORDERED-LIST-ITEM
- TASKANDSINGLEACTION <--> ORDERED-LIST-ITEM
- Head of TASKANDMULTIPLEACTIONS <--> ORDERED-LIST-ITEM
- Tail of TASKANDMULTIPLEACTIONS <--> DESCRIPTION-LIST
- FEEDBACK <--> DESCRIPTION-LIST

2.3 Text Structure -- T-box Mappings

For the sake of the Initial Demonstrator, we link the admissible T-box elements to the text structure elements specified above by the following mapping rules.

- TASK-TITLE <--> Goal of TOP-PROCEDURE
- TASK-INSTRUCTIONS <--> TOP-METHOD
- SUBTASKS <--> STEPS
- TASK <--> PROCEDURE
- SIMPLETASK <--> PROCEDURE1
- TASKANDSINGLEACTION <--> PROCEDURE2 with SINGLESTEPMETHOD
- TASKANDMULTIPLEACTIONS <--> PROCEDURE2 with MULTIPLESTEPMETHOD
- FEEDBACK <--> SIDE-EFFECT

To conclude this section, we provide an example of the part of the text plan which is common to English and all the three languages, for the Initial Demonstrator text 1. It consists of the choice of the text layout elements, the corresponding text structure elements and the distribution of the content of the A-box.

Text layout element	**Text structure element**	**A-box element**
Heading	Task title	Top-procedure goal
Ordered list	Task instructions	Top method
Ordered list item	Task-and-single-action	Procedure 2 with singlestepmethod
Ordered list item	Simple-task	Procedure 1

Ordered list item	Simple-task	Procedure 1
Ordered list item	Task-and-single-action	Procedure 2 with singlestepmethod

The language-specific syntactic realization rules corresponding to the text structure elements are described in the following section.

3. Specifications for Text Structuring per Language (TEXS1-Bu,Cz,Ru)

The admissible A-box configurations as specified above, set the simplified semantics for the purpose of generating the amended Initial Demonstrator texts. In this section, we provide rules for the syntactic realizations corresponding to the text structure elements specified above, in the form of text structure to grammar mappings. The syntactic realizations stand in a direct relation to this simplified underlying semantics. We first provide the rules for English, for the sake of comparison, and then we provide the rules for the three languages, i.e. Bulgarian, Czech and Russian. The latter are also accompanied by sample SPLs. Either we show an SPL for an entire sentence, in which the bold-face parts are those which reflect the rule under consideration. Or we show an SPL fragment reflecting the rule.

3.1 Syntactic Realizations in English

Rule 1: task-title

The task-title is expressed by a non-finite dependent purpose clause, agent unspecified, etc.

Examples: `To draw a polyline', `To draw a multiline'

Rule 2: simpletask

A simpletask is expressed by a non-dependent imperative clause, etc.

Examples: `Specify the first point of the polyline', `Enter **j**', `Choose a justification'

Rule 3: taskandsingleaction

A taskandsingleaction is expressed by a clause complex: the dominant clause (expressing the GOAL) is a simpletask, and the dependent clause (expressing the METHOD) a non-finite means clause, agent unspecified, etc. The dominant clause precedes the dependent clause.

Examples: `Start the PLINE command by choosing Polyline from the Polyline flyout', `End the polyline by pressing Return'

Rule 4: feedback

A feedback is expressed by a finite, present-tense clause simplex.

Example: `The AUTOCAD Text Window appears'

3.2 Syntactic Realizations in Bulgarian (TEXS1-Bu)

Rule 1: task-title

The task-title is expressed by a nominalized clause.

Examples: `Chertane na polilinia', `Chertane na polilinia s pravi segmenti', `Zadavane vida na multilinia', `Chertane na opisani mnogougulnitsi', `Triene na linii'

SPL example:

```
(EXAMPLE
     :NAME   DB-TEXT2b-0
     :GENERATEDFORM   "Chertane na polilinia"
     :TARGETFORM   "Chertane na polilinia"
     :LOGICALFORM
     (S / DISPOSITIVE-MATERIAL-ACTION
          :LEX CHERTANE
          :EXIST-SPEECH-ACT-Q NOSPEECHACT
          :ACTEE
     (P / OBJECT :LEX POLILINIA-NONDET
          :IDENTIFIABILITY-Q NOTIDENTIFIABLE))
     :SET-NAME   DB-TEXT2b)
```

Rule 2: simpletask

A simpletask is expressed by a non-dependent imperative clause, etc.

Examples: `Startirajte komandata MLINE', `Zadaite kraina tochka', `Vavedete **a**', `Izberete element', `Opredelete centara na mnogougulnika'

SPL example:

```
(EXAMPLE
     :NAME   DB-TEXT2-1
     :GENERATEDFORM   "Startiraite komandata MLINE, kato
izberete Multiline ot plavashtoto menu na funktsionalnija red
s ime Draw."
     :TARGETFORM   "Startiraite komandata MLINE, kato
izberete Multiline ot plavashtoto menu na funktsionalnija red
s ime Draw."
     :LOGICALFORM (REL /(RST-MEANS MANNER)
     :DOMAIN
     (S / DIRECTED-ACTION
          :LEX STARTIRAM
          :SPEECHACT IMPERATIVE
          :ACTEE (C1 / OBJECT
               :LEX SOFTWARE-COMMAND
               :IDENTIFIABILITY-Q IDENTIFIABLE
               :CLASS-ASCRIPTION
                    (C2 / SOFTWARE-COMMAND :NAME MLINE)))
     :RANGE
     (C / DIRECTED-ACTION
          :LEX IZBIRAM
          :ACTEE (P / OBJECT
               :NAME MULTILINE)
               :SOURCE (C1 / OBJECT
```

```
                    :property-ascription (q1 / quality
                    :lex plavashto)
               :LEX menu :IDENTIFIABILITY-Q IDENTIFIABLE
               :CLASS-ASCRIPTION (C2 / SOFTWARE-COMMAND
                    :NAME MULTILINE)
                    :SPATIAL-LOCATING
                    (C3 / ONE-OR-TWO-D-LOCATION
                    :property-ascription
                    (q / quality
                    :lex funktsionalen)
                    :LEX Red
                    :IDENTIFIABILITY-Q  IDENTIFIABLE
                    :CLASS-ASCRIPTION
                    (C2 / SOFTWARE-COMMAND :NAME DRAW)))))
    :SET-NAME   DB-TEXT2)
```

Rule 3: taskandsingleaction

A taskandsingleaction is expressed by a clause complex: the dominant clause (expressing the GOAL) is a simpletask, and the dependent clause (expressing the METHOD) a non-finite means clause, agent specified (and implicit), etc. The dominant clause precedes the dependent clause.

Examples: `kato izberete Multiline', `kato izpolzvate edin ot tezi metodi'

SPL example

```
(EXAMPLE
     :NAME   DB-TEXT2-1
     :GENERATEDFORM   "Startiraite komandata MLINE, kato
izberete Multiline ot plavashtoto menu na funktsionalnija red
s ime Draw."
     :TARGETFORM   "Startiraite komandata MLINE, kato
izberete Multiline ot plavashtoto menu na funktsionalnija red
s ime Draw."
     :LOGICALFORM  (REL /
     (RST-MEANS MANNER)
     :DOMAIN
     (S / DIRECTED-ACTION
          :LEX STARTIRAM
          :SPEECHACT IMPERATIVE
          :ACTEE (C1 / OBJECT
               :LEX SOFTWARE-COMMAND
               :IDENTIFIABILITY-Q IDENTIFIABLE
               :CLASS-ASCRIPTION (C2 / SOFTWARE-COMMAND
                    :NAME MLINE)))
     :RANGE
     (C / DIRECTED-ACTION :LEX IZBIRAM :ACTEE
     (P / OBJECT :NAME MULTILINE)
          :SOURCE
          (C1 / OBJECT
          :property-ascription (q1 / quality :lex plavashto)
          :LEX menu :IDENTIFIABILITY-Q IDENTIFIABLE
          :CLASS-ASCRIPTION
          (C2 / SOFTWARE-COMMAND :NAME MULTILINE)
          :SPATIAL-LOCATING
```

```
          (C3 / ONE-OR-TWO-D-LOCATION
          :property-ascription (q / quality
               :lex funktsionalen)
          :LEX Red
          :IDENTIFIABILITY-Q  IDENTIFIABLE
          :CLASS-ASCRIPTION
          (C2 / SOFTWARE-COMMAND :NAME DRAW)))))
     :SET-NAME   DB-TEXT2)
```

Rule 4: feedback

A feedback is expressed by a finite, present-tense clause simplex.

Example: ` Tekstovijat prozorec na Autocad se pojavjava'

Example SPL

```
(EXAMPLE
     :name DB-Text2b-2-b
     :targetform "Tekstovijat prosorec na AutoCAD se
pojavjava na ekrana."
     :logicalform
     (C / NONDIRECTED-ACTION
          :LEX POJAVJAVA-SE
          :ACTOR (C1 / OBJECT
          :property-ascription (q1 / quality
          :lex tekstov-fulldet)
               :LEX prozorec
                    :IDENTIFIABILITY-Q IDENTIFIABLE
                    :part-of  (C3 / object
                         :LEX AutoCAD ))
     :SPATIAL-LOCATING
          (P / ONE-OR-TWO-D-LOCATION
          :LEX EKRAN
          :IDENTIFIABILITY-Q IDENTIFIABLE))
     :SET-NAME   DB-TEXT2b)
```

To summarize, the text planning rules for Bulgarian reveal two differences with respect to the text planning rules for English:

- The top-goal procedure is expressed by a nominalization. The inquiry responsible for the nominalization is:
  EXIST-SPEECH-ACT-Q NOSPEECHACT
- The singlestepmethod is expressed by a finite dependent means clause. The inquiry responsible for the finite clause is:
  TEMPORAL-DEFINITENESS-SPECIFICATION-Q SPECIFIED

3.3 Syntactic Realizations in Czech (TEXS1-Cz)

Rule 1: task-title

The task-title is expressed by a nominalization.

Examples: `Kreslení køivky', ` Kreslení køivky s rovnými segmenty', `Zadání stylu multièáry', ``Kreslení opsaného mnohoúhelníku'

SPL example

```
(EXAMPLE
     :NAME   D0-TEXT1-0-Cz
     :GENERATEDFORM   "Kreslení èáry "
     :TARGETFORM   "Kreslení èáry "
     :LOGICALFORM
          (S / DISPOSITIVE-MATERIAL-ACTION
               :EXIST-SPEECH-ACT-Q NOSPEECHACT
               :ACTEE
                    (P / OBJECT
                    :LEX ÈÁRY
     :LEX KRESLENÍ)
     :SET-NAME   D0-TEXT1)
```

Rule 2: simpletask

A simpletask is expressed by a non-dependent imperative clause, etc.

Examples: ` Spus□te pøíkaz PLINE', `Urèete první bod', `Zadejte **a**', `Vyberte zarovnání'

SPL example

```
(S / DIRECTED-ACTION
     :LEX SPUSTIT
     :SPEECHACT IMPERATIVE
     :ACTEE
          (C1 / OBJECT
               :LEX PØÍKAZ
               :LABEL-ASCRIPTION
          (C2 / SOFTWARE-COMMAND :NAME PLINE)))
```

Rule 3: taskandsingleaction

A taskandsingleaction is expressed by a clause complex: the dominant clause (expressing the GOAL) is a simpletask, and the dependent clause (expressing the METHOD) a non-finite means clause, agent unspecified, usinga nominalization. The dominant clause precedes the dependent clause.

Examples: ` vybráním Polyline', `stisknutím Enter'

SPL example

```
    (REL /
    (RST-MEANS MANNER)
    :DOMAIN
    :RANGE
          (C / DIRECTED-ACTION
          :EXIST-SPEECH-ACT-Q NOSPEECHACT
          :LEX VYBRÁNÍ
          :ACTEE (P / OBJECT :NAME POLYLINE)))
```

Rule 4: feedback

A feedback is expressed by a finite, present-tense clause simplex.

Example: `Objeví se okno AutoCAD Text Window'

Example SPL

```
(EXAMPLE
     :name D0-Text2b-2b-Cz
     :set-name D0-text2b
     :LOGICALFORM
          (s / motion-process
          :LEX objevit-se
          :tense present
          :actor (c1 / OBJECT :LEX okno
               :label-ascription (j / template
                    :pattern "AutoCAD Text Window"))))
```

To summarize, the text planning rules for Czech reveal two differences with respect to the text planning rules for English:

- The top-goal procedure is expressed by a nominalization. The inquiry responsible for the nominalization is:
  EXIST-SPEECH-ACT-Q NOSPEECHACT
- The singlestepmethod is expressed also using nominalization.

3.4 Syntactic Realizations in Russian (TEXS1-Ru)

Rule 1: task-title

The task-title is expressed by either a nominalized clause or an infinitival clause.

Examples: `Chtoby narisovatj polililiju'

SPL example:

```
(EXAMPLE
     :NAME   D0-TEXT1-0-RU
     :GENERATEDFORM   "Chtoby narisovatj polililiju"
     :TARGETFORM   "Chtoby narisovatj polililiju"
     :LOGICALFORM
          (S / DISPOSITIVE-MATERIAL-ACTION
          :LEX NARISOVATJ :PROPOSAL-Q PROPOSAL
          :REPORT-Q REPORT
          :ENTIRENESS-Q PARTIAL
          :ACTEE (D / OBJECT :LEX POLILINIYA))
     :SET-NAME   D0-TEXT1)
```

Example: "Risovanie polilinij"

SPL example

```
(EXAMPLE
     :NAME   D0-TEXT1-0-RU-ALT
     :GENERATEDFORM   "Risovanie polilinij"
     :TARGETFORM   "Risovanie polilinij"
     :LOGICALFORM
          (S / DISPOSITIVE-MATERIAL-ACTION
               :EXIST-SPEECH-ACT-Q NOSPEECHACT
               :ACTEE
                    (P / OBJECT
```

```
                    :LEX POLILINIYA
                    :MULTIPLICITY-Q MULTIPLE)
      :LEX RISOVANIE)
      :SET-NAME   D0-TEXT1)
```

Rule 2: simpletask

A simpletask is expressed by a non-dependent imperative clause, etc.

Examples: ` Zapustite komandu PLINE'

SPL example

```
(S / DIRECTED-ACTION
      :LEX ZAPUSTITJ
      :SPEECHACT IMPERATIVE
      :ACTEE
            (C1 / OBJECT
                  :LEX KOMANDA
                  :LABEL-ASCRIPTION
            (C2 / SOFTWARE-COMMAND :NAME PLINE)))
```

Rule 3: taskandsingleaction

A taskandsingleaction is expressed by a clause complex: the dominant clause (expressing the GOAL) is a simpletask, and the dependent clause (expressing the METHOD) a non-finite means clause, agent unspecified, using adverbial participle (deyeprichastiye). The dominant clause precedes the dependent clause.

Examples: ` vybrav punkt Polyline'

SPL example

```
      (REL /
      (RST-MEANS MANNER)
      :DOMAIN
      :RANGE
            (C / DIRECTED-ACTION
            :LEX VYBRATJ
            :ACTEE (P / OBJECT
                  :LEX PUNKT
                  :LABEL-ASCRIPTION
                        (P2 / OBJECT :NAME POLYLINE)))
```

Rule 4: feedback

A feedback is expressed by a finite, present-tense clause simplex.

Example: ` Na ehkrane poyavitsya okno AutoCAD Text Window'

Example SPL

```
(EXAMPLE
      :targetform " "
      :name D0-Text2b-2b-RU
      :set-name D0-text2b
      :LOGICALFORM
            (s / motion-process
```

```
:LEX pojavitjsja
:tense present
:theme db
:actor (c1 / OBJECT
     :LEX okno
     :label-ascription (j / template
          :pattern "AutoCAD Text Window"))
:spatial-locating (db / one-or-two-d-location
          :LEX ekran)))
```

It is beyond the capacity of the text structurer for the Initial Demonstrator to handle the fact that the verb *pojavitjsja* in Russian requires a location, and that in order to obtain a contextually appropriate sentence, the location has to be thematized.

To summarize, the text planning rules for Russian reveal two differences with respect to the text planning rules for English:

- The top-goal procedure can also be expressed by a nominalization. The inquiry responsible for the nominalization is:
  EXIST-SPEECH-ACT-Q NOSPEECHACT
- The singlestepmethod is expressed by a finite dependent means clause using an adverbial participle.

4. Implementation of Text Structuring for the Initial Demonstrator (TEXM1)

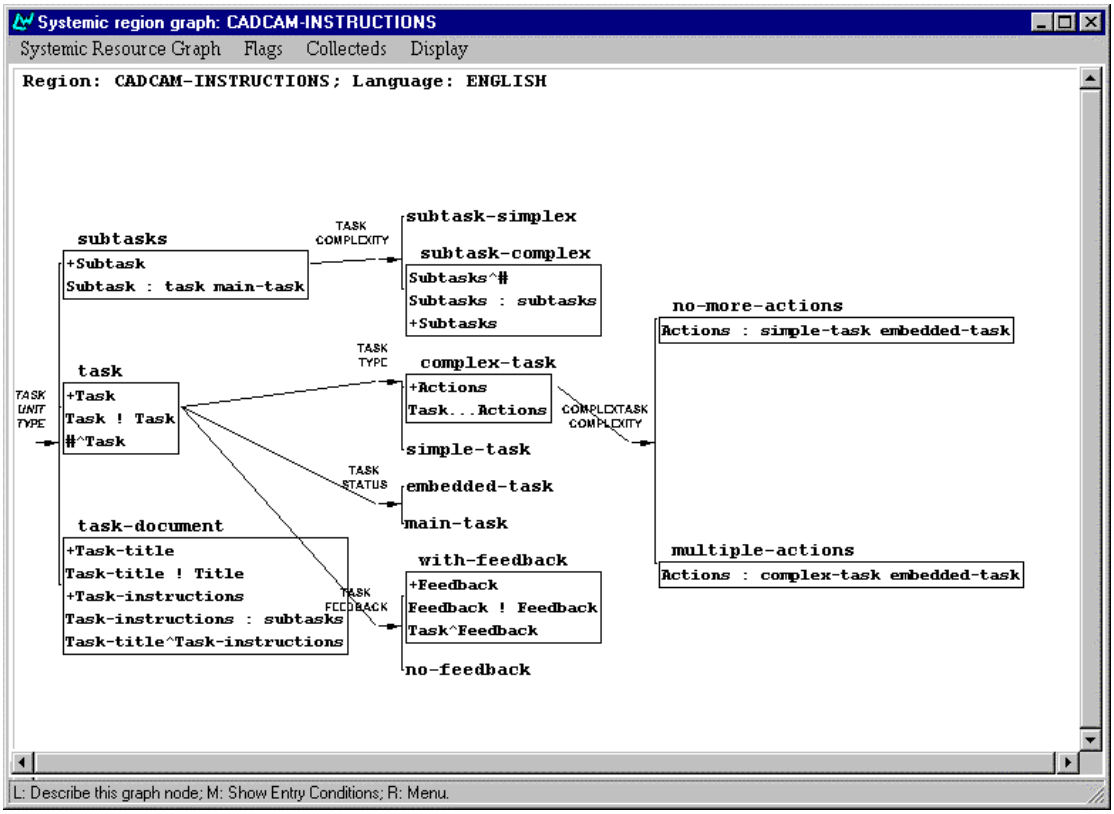
In this section, we describe the current implementation of text structuring, which follows the specifications provided in the preceding section. As we have shown above, the strategy of text structuring is the same for all the three languages, and the language-specific differences only come about at the stage of assigning syntactic realization statements to text structure elements. Since tactical generators are not our main concern in this deliverable (but see the deliverable "Grammatical Resource implementation for Bulgarian, Czech and Russian" of WP7), we use English examples throughout this section, for the sake of easier legibility.

The approach we have chosen uses KPML to build text structures rather than using an external module for doing so. In this way, we do not need an interface between the text structurer and KPML, and information can be easily shared throughout the process of generation, ``end-to-end''. The text structuring networks are presented in the next section.

4.1 Systemic Networks for Text Structuring

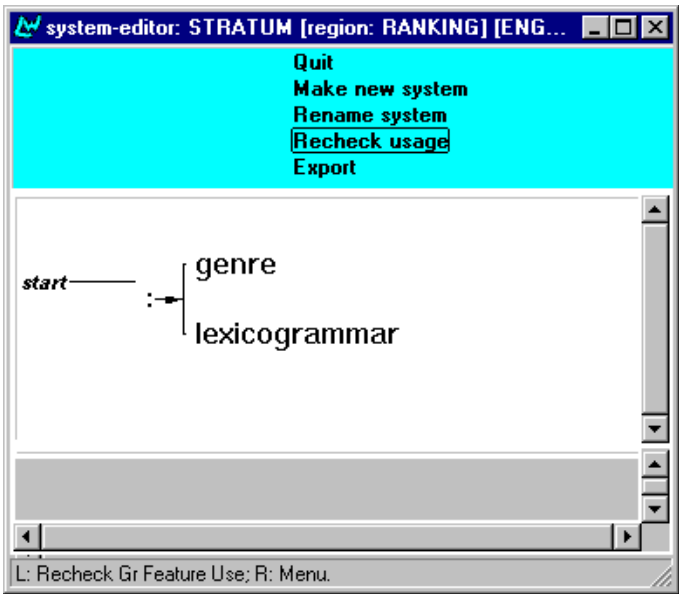
The current implementation assumes that a version of some linguistic resources have been loaded into KPML (see the deliverable "Grammatical resource implementation for Bulgarian, Czech and Russian" of WP7 for references) and the CLOS knowledge representation code in its version for the Initial Demonstrator (see the deliverable "Domain Modelling" of WP2) is present. Then, one loads in the resources called CADCAM-TEXTSTRUCTURE-0. This defines an additional level of linguistic resources for the level of `genre', or text structure, which enables the building up of text structure just like the grammar builds up grammatical structure. The network for the text structure given for the AGILE Initial Demonstrator texts is as shown in Figure 2.

Figure 2: Systemic network for the Initial Demonstrator text structures.



In addition, the first systems of the linguistic resources are changed so that the first choice is one of linguistic stratum, not grammatical rank (see Figure 3).

Figure 3: The first choice in the systems of the linguistic resources is one of linguistic stratum.

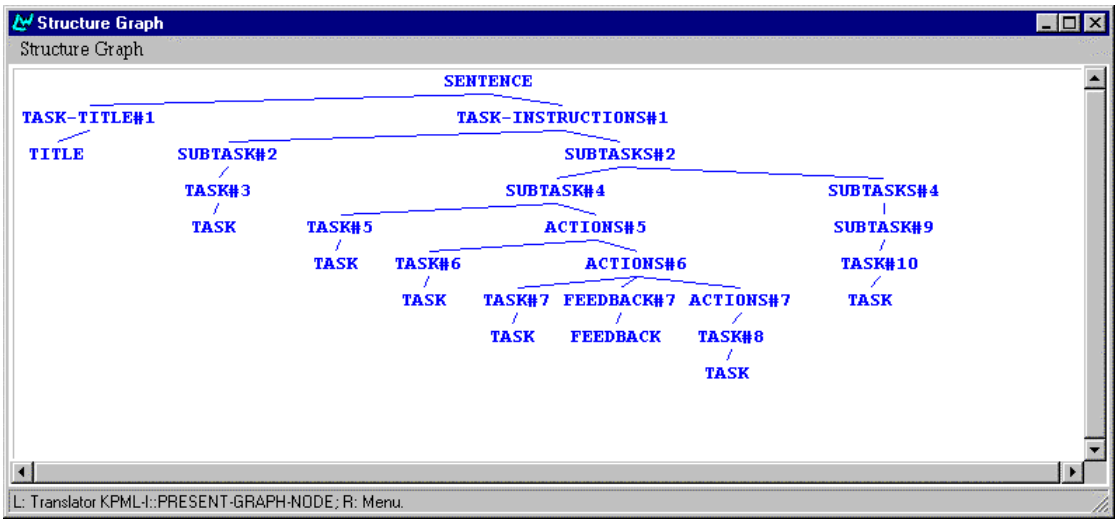


This enables the generation of text structures rather than clause structures in KPML by preselecting the feature `genre', thus:

(say '(um / um-thing) :preselections '(genre))

Then, a traversal through the network is made, asking what feature choice to make. This produces the text structures of our Initial Demonstrator. For the sake of an illustration, Figure 4 shows a text plan which closely corresponds to the text plan needed for Text 2 of the Initial Demonstrator. The leaves of the text structure are lexified with their names (e.g., `Task') in this example, in order to make it clearer what has happened and also to stop the generation process going on to try and generate clauses (which is what would otherwise happen, as descried in the deliverable "Grammatical Resource implementation for Bulgarian, Czech and Russian" of WP7).

Figure 4: Sample text structure generated in KPML



The string which would be produced in this way would be, however, quite unreadable, because sentences would just follow one after another, e.g.,

TITLE TASK TASK TASK TASK FEEDBACK TASK TASK

Therefore, we draw in the layout rules specified above, and add the corresponding HTML mark-up. Since the markup only need rely on the text structure, it can be given directly as punctuation in the KPML sense. The following rules are sufficient for the mappings between text elements and format/layout as given in the specification:

```
(define-pre-punctuation
  :language :ENGLISH
  :punctuation-rules '((task-document TASK-TITLE        "<h1>")
                       (task-document TASK-INSTRUCTIONS  "<ul>")
                       (subtasks      SUBTASK            "<li>")
                       (main-task     ACTIONS            "<ul>")
                       (embedded-task TASK               "<li>")
                       (task          FEEDBACK           "<li>")
```

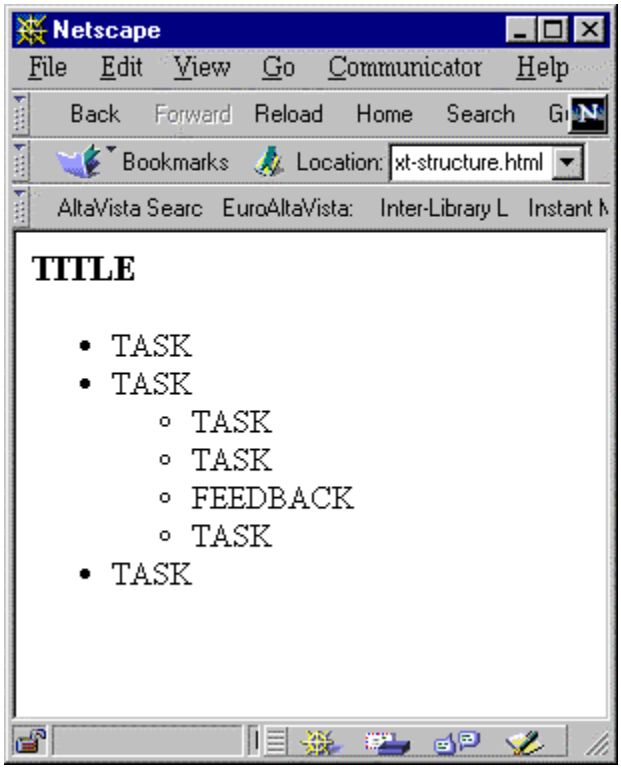
```
))


(define-post-punctuation :language :ENGLISH
 :punctuation-rules '((task-document TASK-TITLE       "</h1> ")
                      (task-document TASK-INSTRUCTIONS "</ul>")
                      (subtasks      SUBTASK           "</li>")
                      (main-task     ACTIONS           "</ul>")
                      (embedded-task TASK              "</li>")
                      (task          FEEDBACK          "</li>")
                      ))
```

Loading these rules in for the languages we are generating, produces the following:

"<h1> TITLE </h1>   TASK   TASK <ul> TASK   TASK   FEEDBACK   TASK   TASK "

We can send this output to a browser, which can interpret the HTML markup and display an accordingly formatted text (see Figure 5).

Figure 5: Sketch of text structure with the appropriate layout viewed in a browser



Now, two further steps need to be taken:

1. making the generation of the text structure follow the information given in the A-box
2. associating the elements of the text structure with appropriate fragments from the A-box
3. making those elements be realized by appropriate clause structures constrained in the way the text specification states.

The first step is carried out by defining appropriate choosers and inquiries that inspect the A-box and decide according to the configuration found there; the second step is also handled here, by using ID-inquiries to associate particular chunks of the A-box with particular text elements. And the latter step is performed by placing particular constraints on the realization of the text elements in the system network for the text type and converting the A-box chunks to ideational SPLs following the mappings given by the relation between domain model and upper model.

All of these steps are included in the resources CADCAM-TEXTSTRUCTURE-1. They can be loaded with a command such as:

```
(load-linguistic-resources :cadcam-textstructure-1 :root-directory
"C:\\Projects\\Agile\\Textstructurer\\" :clear nil)
```

(The :clear nil"` is to make sure that no grammatical resources are lost.)

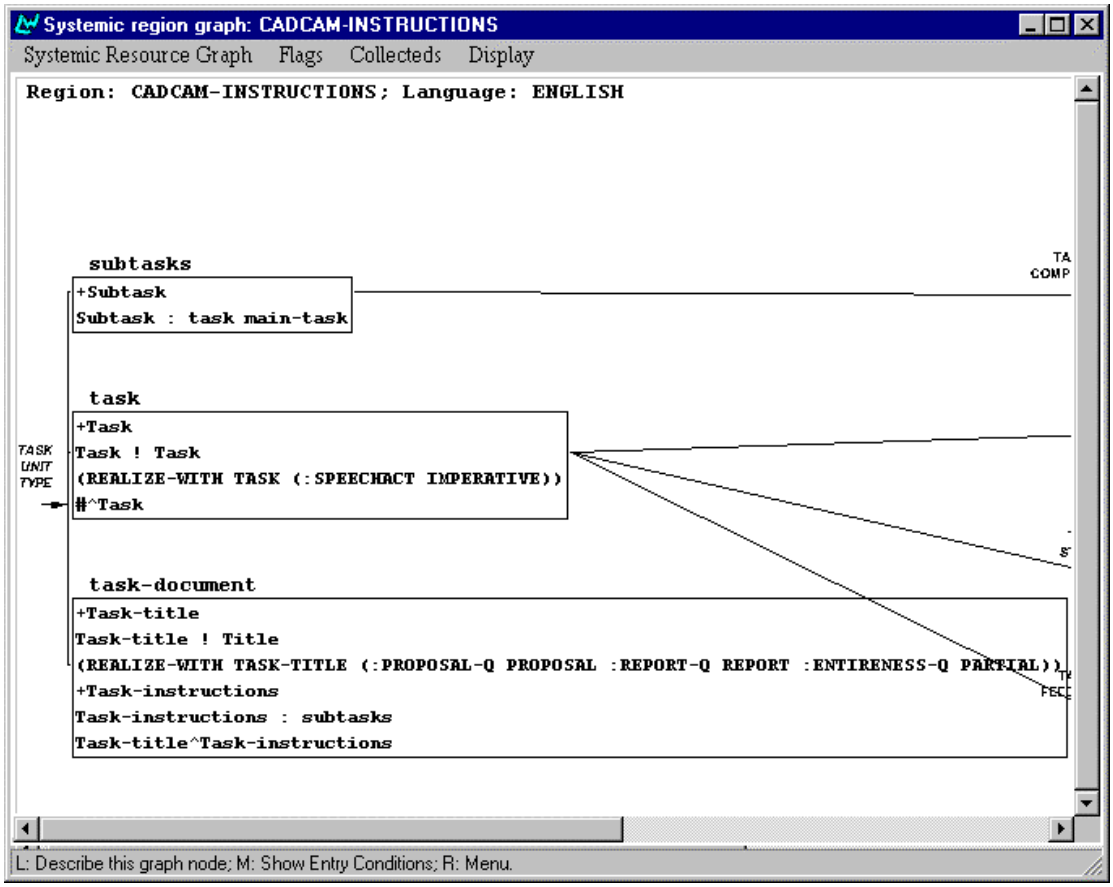
Then one also needs to load a few simple inquiry implementations and a function that starts generation of appropriately. These are in the file: \textstructurer.lisp.

Now, looking at the system network, or rather at particular systems, such as in Figure 6, we can see that there are some additional realization statements (in capital letters in the picture). These have the effect of placing particular realizational constraints on the text elements. For example, we can see that the title of the document we are generating (the Task-Title) should be realized with the additional inquiries and their responses l inquiries and their responses:

(:proposal-q proposal :report-q report :entireness-q partial)

present. These are just the inquiries necessary to make the grammar produce a purposive nonfinite clause such as: to make a Polyline. ., as is appropriate for the headers (in English or in Russian). Since these realization statements are just like any other, they can be conditionalized appropriately for each language where there is a difference, e.g., to obtain the nominalization forms that are more appropriate for Bulgarian and Czech, etc.

Figure 6: Placing realization constraints on the text elements.

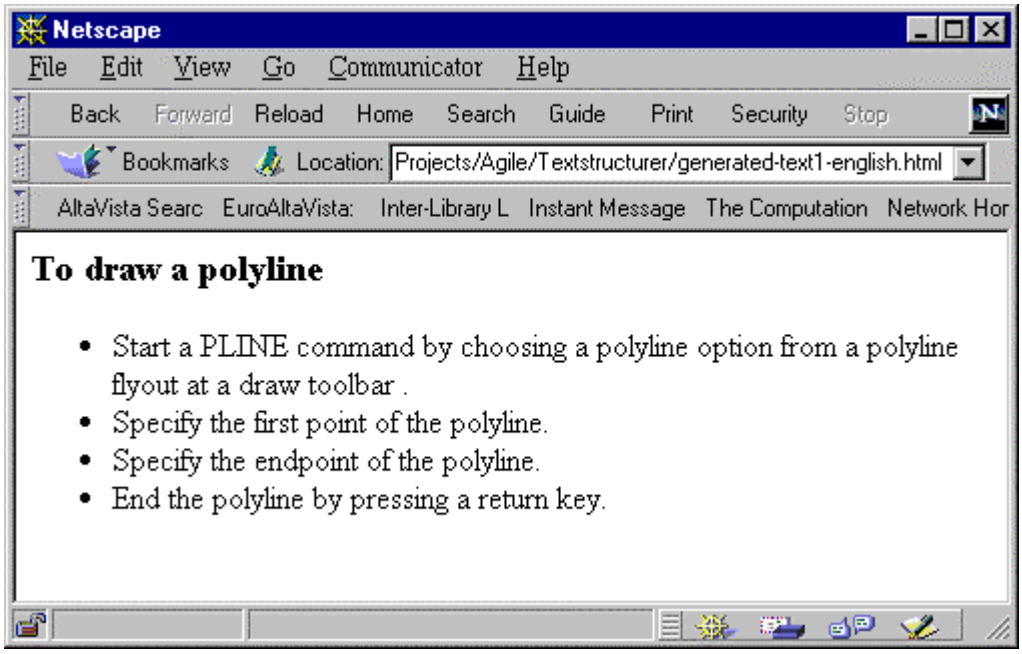


4.2 Sample Text Structure Generation

Generation of a text now works with the function make-text-0. This simply starts the grammar off and points the inquiries to look at the value of dm::*abox-root* rather than an SPL. Subsequent inquiries then decompose the A-box structure successively, associating the necessary components with the text elements. These text elements are then themselves realized by converting their A-box segments into SPL straightforwardly and adding in the additional constraints that are obtained from the network via the REALIZE-WITH realization statement.

For example, if we load the A-box definitions for Text 1 of the Initial Demonstrator, and start the generation function for the text, we receive the result displayed in Figure 7 (the output of KPML has been passed directly to a browser for ease of viewing in HTML):

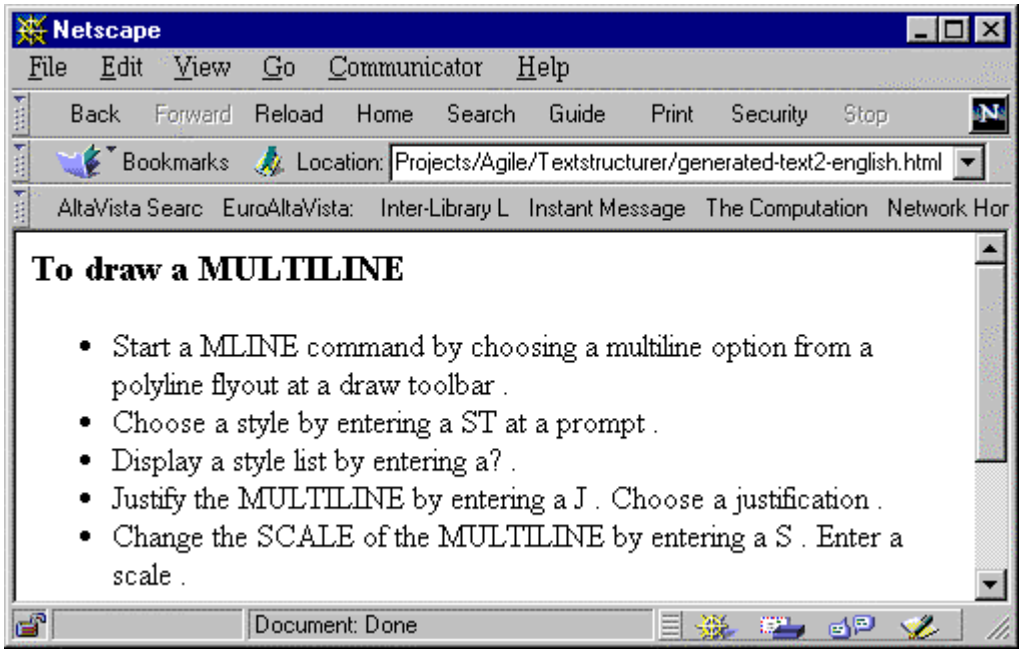
Figure 7: English version of Text 1 with automatically generated text structure.



As is apparent in this example, there are still several rough edges to this text which will be addressed as the work on text structuring proceeds. For example, the control of definiteness will need further work.

Loading the A-box definitions for Text 2a and running the generation yields the result displayed in Figure 8: English version of Text 2 with automatically generated text structure.:

Figure 8: English version of Text 2 with automatically generated text structure.



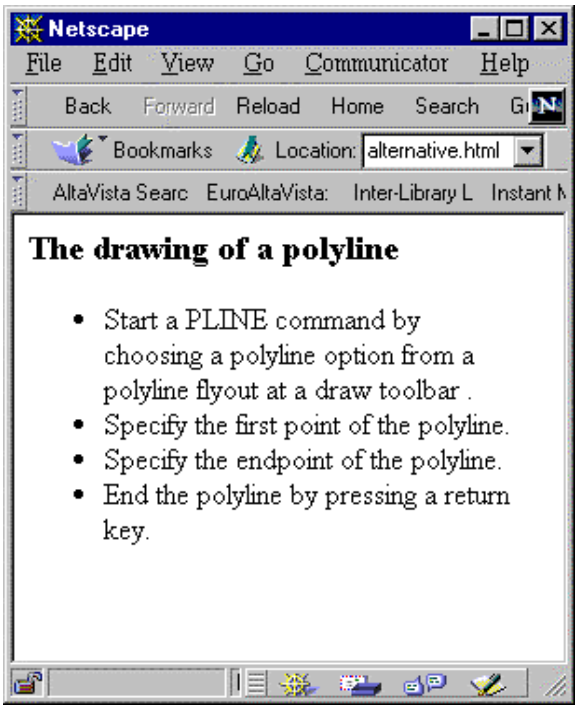
There are some reconfigurations required to get exactly the form of the target texts, which are simpler in grammatical structure than these shown here. This is to be achieved by breaking down the A-box into smaller units when it is being decomposed for the text structure: at present the finer level of structure within complex tasks is not being looked for by the text structure resource.

Finally, to actually show the spirit of the multilingual variation, we can change the realization statement imposed on, for example, the Task-Title shown above to that as given in the above specifications for one option for Russian and the preferred option for Bulgarian and Czech, i.e., a nominalization, with the following rule:

realization :exist-speech-act-q nospeechact

Now, if we try regenerating, from the A-box definitions for the Initial Demonstrator Text 1, the heading will be a nominalization, as illustrated in Figure 9.

Figure 9: English version of Text 1 with automatically generated nominalization in the heading.



This demonstrates that a substantial range of the variation that we have observed between the languages of the project in their selected linguistic forms for the instructional texts can be simply modelled by conditionalization in the constraints that apply to particular text structure elements.

5. Conclusions

We have presented the results of the Task 5.1 of the work package 5 concerned with text structuring in the AGILE project. The goal of this task was to provide text structuring specifications and implementation of a text structuring module for the texts of the Initial

Demonstrator in the Bulgarian, Czech and Russian language. We have specified the text structure elements, their mapping to layout rules, their mapping to A-box configurations and their mapping to syntactic realizations. We have also accomplished the essential implementation of text the structuring module as part of an end-to-end text generator, which takes user definitions of text content in the form of an A-box, and outputs texts in a given style and format which express this content.

The text structuring for the Initial Demonstrator of course has certain limitations which had been decided beforehand, in order to make the task feasible to complete in the allotted time.

First of all, we have been working with A-box configurations of restricted complexity. We have not been exploiting the full set of concepts and their combinations as their are defined in the AGILE T-box, and we been working with A-boxes with a restricted depth. These restrictions enabled us to formulate quite simple text structuring rules as the first attempt within the project.

Another limitation of the text structuring for the Initial Demonstrator lies in the invariability of the generated text structures. We have not developed strategies for choosing between alternative ways of expressing a given content in correlation to the complexity of the content to be expressed and the preceding context. We have also only employed one text style and one set of layout rules.

A further limitation concerns the use of various forms of referring expressions with respect to the status of the entities they refer to in a model of the context, taking into account for instance whether an entity has already been referred to and whether other entities have been referred to by expressions in a competing form. We currently use uniform expressions to refer to entities instantiating a given concept. Moreover, since we have not been working on the issues of identifiability, the text structurer currently cannot determine which noun groups should be definite, or what should be thematized in a generated sentence.

These are issues that are to be tackled in the next stages of the project. It of course cannot be expected that we shall solve all of them to full extend, but we will at least try to make improvements along these lines for the sake of the intermediate prototype.

Among the issues handle in the next stage, we shall especially concentrate on (i) more flexible generation of alternative forms, (ii) the text planning necessary to decide whether a given content should be expressed by a sequence of simplex clauses or by a complex clause, (iii) the text planning and context modelling underlying the choice of various abbreviated forms of referring expressions, and (iv) the text planning and modelling underlying the determination of Theme-Rheme structures, and in particular word order.

References

[Agile project 1997] Agile project. Automatic generation of instructions in languages of Eastern Europe. Technical report, Commission of the European Community, Brussels, DG XIII. Technical Annex.

[Power et al. 1994] Richard Power, Lyn Pemberton, Anthony Hartley and Louise Gorman (1994). User Requirements Analysis. DRAFTER Project deliverable WP2 IED4/1/5827, IITRI, University of Brighton, UK.

[Pemberton et al. 1996] Lyn Pemberton, Louise Gorman, Anthony Hartley and Richard Power (1996). Computer Support for Producing Software Documentation: Some Possible Futures. In: T. van der Geest and M. Sharples eds.), *The New Writing Environment: Writers at Work in a World of Technology*. Springer Verlag.

[Power and Scott 1997] Richard Power and Donia Scott, 1997. NLG tools to support technical authors and translators. A manuscript submitted to the ANLP'97 conference.