AGILE

Automatic Generation of Instructions in Languages of Eastern Europe

---

Title ***Flexible text structuring for the final prototype (documentation to the software)***

Authors Geert-Jan M. Kruijff
Ivana Kruijff-Korbayová
Serge Sharoff
Ivan Hadjiiliev
Lena Sokolova
Michael Boldasov

Deliverable *TEXM3-CU, TEXM3-Ru, TEXM3-Bg*

Status *Final*

Availability *Public*

Date *Friday, February 02, 2001*

INCO COPERNICUS PL961104

Abstract:

The aim of the WP 5.3 task has been to develop a framework that enables the integration of different approaches to discourse structuring and provides a more general perspective on strategies for text generation beyond the domain of application. In this deliverable we describe the Final Prototype Text Structuring Module that achieves that aim, illustrating its capabilities on the various text types and styles as they were defined in the TEXS3 deliverable.

The combined deliverable covers

- TEXM3-CU (Full instructional texts, sentence planning),
- TEXM3-RRIAI (summaries, functional descriptions), and
- TEXM3-BAS (tables of contents, overviews).

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

Table of Content

Table of Figures

1. Introduction

The aim of the WP5.3 task has been to develop a framework that would enable one to integrate different approaches to discourse structuring, by providing a more general perspective on strategies for text generation that would be applicable beyond the domain of application considered in AGILE. In this deliverable we describe the Final Prototype Text Structuring Module, which achieves that aim.

The Text Structuring Module (TSM) for the final prototype is based on the modules developed at earlier stages in the project [TEXM1, TEXM2]. Architectonically, the organisation of the individual components making up the TSM has remained the same. Thus, as we described already in detail in [TEXM2] and [TEXS3] (cf. also Kruijff & Kruijff [1999, 2000]), the TSM consists essentially of a *text planner* and a *sentence planner* arranged sequentially as in the general reference architecture proposed by Reiter & Dale in [2000]. The task of the text planner is to generate a particular type of text in a specific style (as determined by the user), given an A-box. The output of the text planner is a text plan, which serves as input to the sentence planner. The sentence planner interprets the text plan, and generates a sequence of sentence plans that specify how the content, structured by the text plan, should be realised as sentences.

The organisation of the TSM into a separate text planner and sentence planner has been the same across all the prototypes developed within WP5. What makes the final prototype TSM different from the previous prototypes is the *division of labour* between the text planner and the sentence planner.

Up to [TEXM2], it was the case that important decisions regarding textual coherence, like aggregation, were actually made by the *sentence planner* instead of the text planner. Together with the fact that the sentence planner is a fairly intricate implementation, this division of labour did not allow for much flexibility in extending the planning of aggregation or any related textual phenomenon.

In [TEXS3] we addressed this problem by substantially revising the tasks of the text planner and the sentence planner. Essentially, the sentence planner was no longer to make any decisions regarding textual phenomena. Instead, it should simply interpret the text plan in the sense of translating a configuration of text plan elements (as given in the text plan) into a specification detailing how the content associated with these elements should be realised as a sentence. Naturally, this meant that the text planner's task became more complex. Because now the text planner should plan e.g. discourse-level aggregation using RST-relations by specifying such relations explicitly in the text plan.

The fact that the major part of the text planner had been implemented as a region in KPML [TEXM2] provided a good basis for extending the text planner in the intended direction. Namely, most of the work that would have to be done would be to add additional systems to the already existing region, each system covering the planning of a textual phenomenon. The gain would be substantial. Not only would we obtain more perspicuity in the control over introduction of textual phenomena, but most importantly, this proposal would also give rise to a *flexible approach* to text- and sentence planning. Making use of the KPML paradigm, we would now be able to write *text grammars* in the same spirit as we had been developing lexicogrammars before.

In the [TEXS3] deliverable we described a variety of text types and text styles that would illustrate the capabilities of such a revised TSM. Part and parcel of the descriptions were

formal specifications of systems that should enable the text planner to construct the relevant text plans.

The current deliverable provides documentation to the software developed for the Final Prototype TSM (FP TSM): how we have implemented the systems corresponding to these formal specifications, and how the sentence planner has been restructured.

More specifically, we begin in chapter 2 with describing the sentence planner. The reason for starting our discussion with the sentence planner rather than with the regions for doing text planning is that this makes it possible to briefly describe with each text planning region how the sentence planner has been extended to interpret the type of text plans generated by that region, if needed. Subsequently, we describe the regions that plan the different text types as specified in [TEXS3].

We begin in chapter 3 with describing the CADCAM-INSTRUCTIONS region. Because the principal systems making up this region have already been described in great detail in deliverable [TEXM2], we focus here only on the additions to the region. These additions regard the planning of discourse aggregation (RST-relations), syntactic aggregation (coordination involving conjunction, disjunction), sequence marking, and a new text style. We will also describe how we plan topic/focus structure, i.e. contextual boundness. This introduces the integration of a third strategy for discourse planning into the framework we have been developing here, namely one based on the Praguian framework of Functional Generative Description [Sgall et al, 1986]. As we described already in [Kruijff-Korbayová *et al* 1999], and in more detail in [Kruijff-Korbayová *et al* 2000], we can employ the Praguian approach to our advantage when dealing with contextually appropriate word order. The integration with Halliday's Systemic Functional Linguistics leads to interesting possibilities in dealing with issues regarding constraints on *syntactically* appropriate word order (e.g. ordering of clitics).

In chapter 4 we describe the SUMMARIES region. According to the description given in [TEXS3], this region specifies text planning required for presentation of a sequence of basic steps the user should perform to accomplish the topmost goal of an A-box.

In chapter 5 we describe the CADCAM-FUNCTIONAL-DESCRIPTIONS region. This region specifies text planning required for describing the functionality of the application being documented by AGILE. Functional descriptions link actions with buttons, keys and software commands to their function in the application. In real manuals they typically correspond to ready-reference sections.

In chapter 6 we describe the generation of Table of Contents as done within the TSM. The CADCAM-TOCS region aims at planning and producing tables of contents. As it was described in the [TEXS3] deliverable, in order to work properly, the region relies on a complex A-box, which is gathered in advance (see [TEXS3]).

Chapter 7 covers the implementation of the generation of overviews. This is done in the CADCAM-OVERVIEW region. The chapter covers all the additional features implemented in this region, so that the generation of OVERVIEW text style is as described in [TEXS3]. One of the new features is syntactic aggregation (CONJUNCTION), which is described in the chapter. Just like the TOCS region, this region also relies on the complex A-Box, described in the deliverable (see [TEXS3]).

We conclude the deliverable with a recapitulation of what we have achieved with the final prototype TSM - both in terms of goals determined for the project as such, and in relation to the *state of the art* in the area of Natural Language Generation.

This is a combined deliverable. It covers

- TEXM3-CU (Full instructional texts, sentence planning),
- TEXM3-RRIAI (summaries, functional descriptions), and
- TEXM3-BAS (tables of contents, overviews).

2. The TSM sentence planner

2.1 The task of the sentence planner

The task of the sentence planner is, simply put, to generate a sequence of sentence plans for a given text plan. The text plan structures the content of an A-box in such a way, that the content can be realised as a *text*. The sentence planner interprets the text plan to create plans for sentences, each realising pieces of the A-box's content as identified by text plan elements. By generating a *sequence* of sentence plans, and having a lexicogrammar generate each sentence plan *as it comes in that order*, we obtain the entire text that realises the content as the user specified it with the A-box.

A lengthier discussion of the tasks of a sentence planner in the context of a general reference architecture for NLG systems was provided in [TEXM2]. Here we will rather concentrate on the improvements that have been made to the sentence planner, and how these improvements enable a principled and straightforward way of extending its coverage of text plan configurations.

2.2 The previous implementations of the sentence planner

Previous implementations of the sentence planner could be characterized as follows:

- Sentence plans are to be specified in the **Sentence Planning Language** - i.e. the planner needs to create a sequence of SPLs.
- The text planner individuates content and distributes it over text plan elements making up the text plan, but the sentence planner should decide whether to realise the content identified by a text plan element as a single sentence, or to group it together with pieces of content identified by one or more elements and realise that composite content as a complex sentence.

In the context of the initial and intermediate prototype the latter point concerned primarily the generation of sentence plans for clause complexes specified using RST-relations. However, the disadvantage of this approach was that the decisions for aggregation were hard-wired into the sentence planner. Thus, there was no possibility to gain control over the criteria for constructing an aggregation using the standard tools available in KPML (i.e. systems, choosers, and inquiries) nor was there any easy way in which the sentence planner's coverage of aggregation phenomena could be extended.

Both these difficulties (control, extendibility) were addressed in TEXS3, where we specified a new division of labour between the text planner and the sentence planner.

2.3 The final prototype TSM implementation of the sentence planner

The essence of the new division of labor is to let the text planner to do the entire planning of distribution of content, and phenomena like discourse aggregation (involving RST-relations) or syntactic aggregation (coordination involving conjunction or disjunction). To illustrate the increased complexity in text plans, consider for example the two plans given in Figure 1 below.

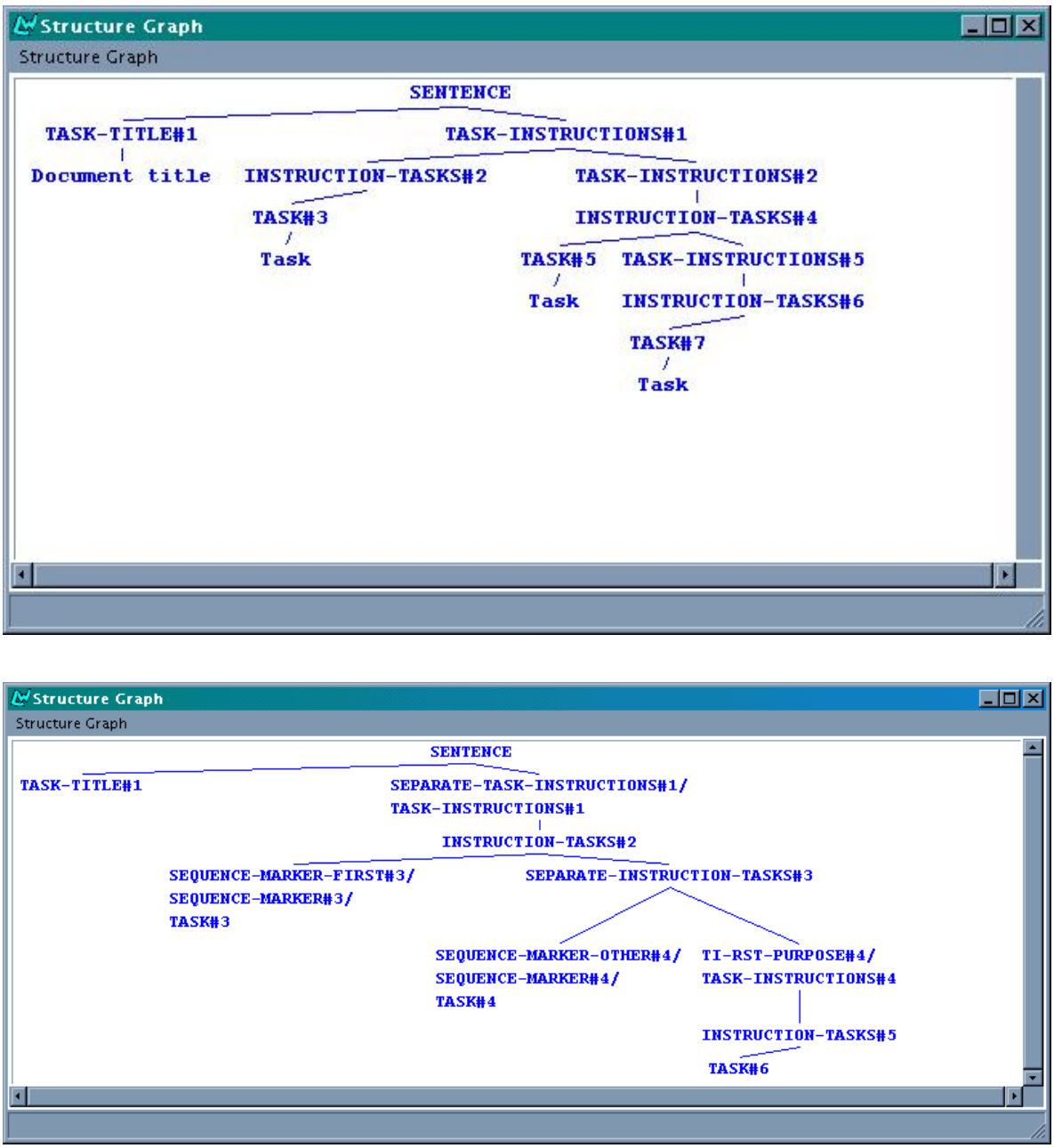


Figure 1 - IMD TSM text plan vs Final Prototype TSM

The textplan at the top in Figure 1 is a textplan produced by the TSM for the intermediate demonstrator, whereas the bottom textplan is one produced by the TSM for the final prototype. Both textplans were generated from the same A-box. However, whereas the IMD TSM textplan includes fairly little information about what to do with the content identified by the leaves, essentially just following the structure of the A-box, the FP TSM textplan illustrates to some extent the wealth of information that is now being included. For the text being planned here, the TSM for example decided that the last step and its method (Tasks #4 and #6) should be aggregated using an RST-Purpose relation, and that there should be explicit linguistic marking of sequentiality. In the next chapter we will discuss the text planning for full procedural instructions in much more detail, showing the full power of the FP TSM.

Besides a text plan the text planner also builds a (rudimentary) discourse model. This discourse model is used by the sentence planner to determine whether a piece of content is retrievable from the preceding context and should therefore be considered as contextually bound (and hence, be topicalised), or whether it is not. The importance of discerning the retrievability of content is that it has an effect on contextually appropriate word ordering: Words realising contextually bound content should be ordered before words realising "new" content [Sgall *et al*, 1986]. We discuss our model of word order in more detail in [Kruijff-Korbayova *et al*, 1999; Kruijff-Korbayova *et al*, 2000] and in [SPEC3]. In this section we will concentrate on how the sentence planner is used to interpret planned relations between content, as made explicit in the text plan.

Particularly, in the remainder of this section we briefly discuss the basic approach to interpreting text plans, and the two basic classes of relations that are distinguished by the text planner (on the basis of the AGILE domain model) - relations between PROCEDUREs/Tasks, and those between METHODs/Instructions. Naturally, the discussion here does not entirely exhaust the capabilities of the sentence planner: notably, see the [TEXM2] deliverable for more details on how we deal with complex lexical semantics in the sentence planner.

2.3.1 Basic approach to interpreting text plans

The basic approach in which the sentence planner interprets a text plan is as follows. The text plan itself is a tree, with leaves that are associated to pieces of content. The sentence planner traverses the tree, and creates for each leaf an SPL plan that realises the content associated with that leaf. The recursion has been formulated such that, when descending, SPL plans are generated that are collected later when the recursion backtracks to the root of the text plan.

Naturally, when the sentence planner traverses the tree, it has to check whether the node it is currently considering is a terminal node (i.e. a leaf), or not. Only when it is a leaf, it generates an SPL plan; otherwise it continues descending. The nonterminal nodes are interesting as well, though. As we discussed already at length in [TEXS3], and illustrated in Figure 1 above, nonterminal nodes can be used to indicate *relations* like RST-relations for discourse aggregation or syntactic relations like coordination involving conjunction or disjunction. For example, consider again the node with the TI-RST-PURPOSE label in Figure 1, which indicates that the bits of content associated to Tasks #4 and #6 should be realised by a complex sentence relating the content by means of an RST-Purpose relation.

In the final prototype implementation of the sentence planner, whenever a nonterminal node is encountered, the sentence planner retrieves the (possibly complex) *label* of the node and checks whether it is used by the text planner to indicate a relation. Whenever the sentence planner encounters a nonterminal node indicating the need to relate different pieces of content, the sentence planner inserts a *trace*. The trace simply keeps track of the fact that the two pieces of content associated to the leaves under it in the text plan, need to be realised by a complex sentence plan involving a particular relation.

Subsequently, when the sentence planner gathers all the SPL plans to arrange them into a sequence of sentence plans, it checks for traces. If a trace is encountered, the two relevant SPL plans are combined using the indicated relation, and it is the composition that gets added to the sequence rather than the two individual SPL plans.

Thus, unlike in its previous incarnations, the sentence planner no longer makes any decisions whether or not to aggregate content. It is strictly guided by what the text plan prescribes.

Finally, to allow for maximal generality, the sentence planner is -naturally- capable of handling theoretically infinite recursive depth of relations like coordinations (RST-relations cannot be recursively infinite). The sentence planner leaves it up to the text planner to constrain recursive depth of relations in a manner that is justifiable in terms of textual coherence.

2.3.2 Aggregation of *Tasks*

The AGILE domain model [MODL2] distinguishes four types of concepts that are used to configure the content in an A-box: PROCEDURE, METHOD*, and lists of these - PROCEDURE-LIST and METHOD-LIST. A PROCEDURE-LIST describes a sequence of instructions or "goals", whereas a METHOD-LIST specifies alternative ways of obtaining a (higher) goal.. As we already specified in various previous WP5 deliverables (e.g. in [TEXM3]), we map these configurational concepts to text plan elements that make up a text plan. Thus, the structure of the text plan closely follows the way content is configured in the A-box.

The text plan element that corresponds to a PROCEDURE is a Task. Each Task is identified with the content of a PROCEDURE's Goal slot. The sentence planner is capable of interpreting the following types of aggregation of Tasks with other Tasks:

- Conjunction of Tasks that correspond to PROCEDUREs in one and the same PROCEDURE-LIST. The sentence planner imposes no constraints whatsoever on the *depth* of the conjunction - as far as the sentence planner is concerned, a list to an arbitrary recursive depth can be realised. In other words, we put the responsibility for planning comprehensible conjunctions with the text planner.
- Aggregation of two Tasks using an RST-relation like PURPOSE or MEANS/MANNER. In this case, one Task (the RST nucleus) has a METHODS field, whose corresponding node in the textplan governs the second Task (being the RST satellite).
- Aggregation of several Tasks using RST-SEQUENCE. This case is similar to the case of conjunction mentioned above, but now explicit sequence markers are included. Again, any arbitrary recursive depth can be dealt with.

The sentence planner is also capable of creating sentence plans for the "aggregation" of a Constraint with a Task (or any of the above aggregations). We already discussed the following possibilities in the [TEXS3] deliverable with regard to incorporating a Constraint into running text:

- Incorporating a Constraint as a LOCATION, like "In Windows, do …"
- Relating a Constraint and the succeeding text using an RST-MEANS MANNER rhetorical relation, as in "Using Windows, …."
- Relating a Constraint and the succeeding text using an RST-LOGICAL-CONDITION relation, as in "When using Windows, …."

Finally, sequencing is dealt with in a straightforward way. To enable language-specific realisation of types of sequence markers, we make use of REALIZE-WITH statements in the CADCAM-INSTRUCTIONS region to impose the appropriate realisation constraints. The only action that the sentence planner has to take, is to ensure that these constraints get included in an SPL. We do this by gathering all the REALISE-WITH constraints that have been associated to a particular function shown as a label on a node (whereby there may be *more than one* label on a node, and hence multiple sets of realisation constraints). Subsequently, we combine these constraints with the SPL code that specifies how the ideational part (i.e. the content) should be realised. The resulting SPL thus includes both a content specification and all the constraints specifying genre-specific realisation.

2.3.3 Aggregation of *Instructions*

Instructions are the text plan elements corresponding to METHODs in the A-box, modeling alternative ways of obtaining a PROCEDURE's Goal. As we proposed in [TEXS3], it is therefore natural to plan these alternatives as a coordination involving disjunction, provided certain conditions are met regarding the complexity of the METHODs involved.

The sentence planner is capable of interpreting any disjunction the textplan may include (again, without any restrictions on recursive depth). A slight modification of the A-box used to generate the textplans shown in Figure 1 produces a textplan that includes disjunction:

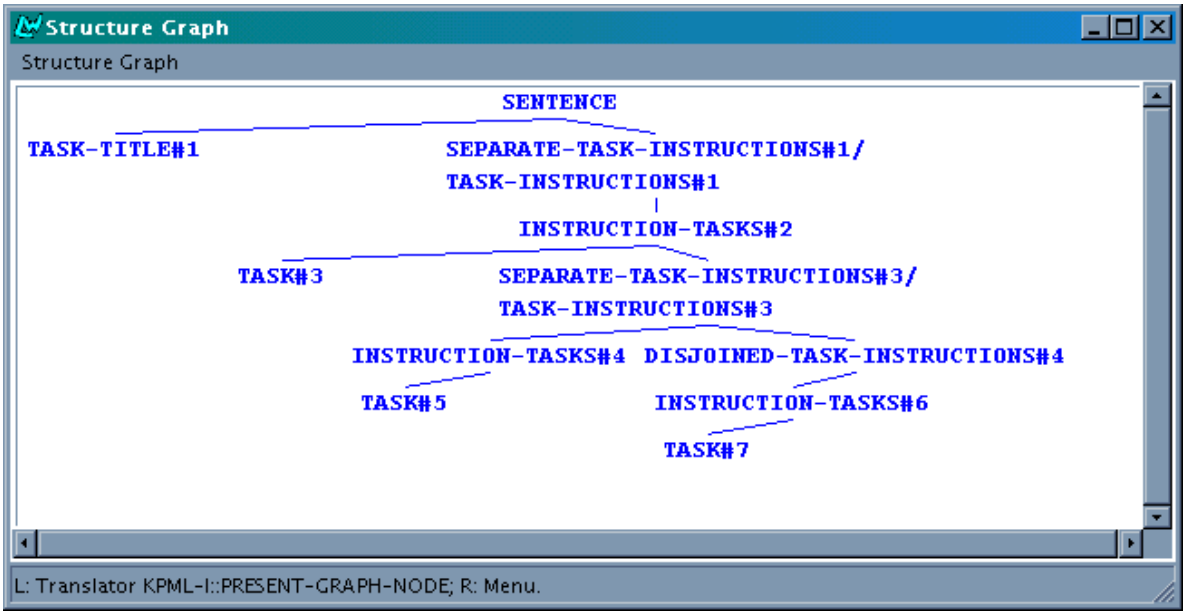


Figure 2 - A FP TSM text plan showing disjunction of instructions

Note how the two textplans generated by the FP TSM differ. In the textplan in Figure 1 a conjunction was planned of elements that were all part of a subtree with as its root an INSTRUCTION-TASKS node. Conversely, in the textplan in Figure 2 the disjunction (indicated by DISJOINED-TASK-INSTRUCTIONS) appears as a node in a subtree headed by TASK-INSTRUCTIONS - i.e. it holds -essentially- between METHODs of one and the same METHOD-LIST (to which TASK-INSTRUCTIONS corresponds, see [TEXM2]).

For more details on how conjunction and disjunction are planned by the text planner, see Section 3.2 below.

2.4 Extendibility of the sentence planner

There are two ingredients to making a sentence plan based on a relation indicated in a text plan:

1. Checking whether a nonterminal bears a label indicating that such a relation should be used in a sentence plan to realise the content associated with the two dominated leaves;
2. Building the sentence plan involving the indicated relation (possibly to some recursive depth > 2) once a trace is encountered.

Whenever the sentence planner is to be extended to cover a new relation, the above two ingredients should be provided. The checking of nonterminal labels, the insertion of relevant traces, and the subsequent processing of traces all happen in a single function in the sentence planner. Furthermore, the already available functions for building sentence plans involving a particular relation can be used as templates for constructing similar functions. Altogether, this makes the sentence planner extendible in a straightforward and principled way. The focus on planning a text thus gets shifted to the phase of *text* planning, where we have the intuitive tools of KPML at our hands to write "text" (or "discourse") grammars. In the next chapter, we describe an extensive example of a text grammar for planning full procedural instructions - the CADCAM-INSTRUCTIONS region.

3. The CADCAM-INSTRUCTIONS region

3.1 Overview

In this chapter we describe the CADCAM-INSTRUCTIONS region. This region is responsible for generating text plans for full procedural instructions. We described already in [TEXM2] how the region creates text plans from A-boxes of arbitrary recursive depth, and how various styles of full instructional texts can be planned. Therefore, we restrict ourselves here to describing the additions to the region, as they have been specified in the [TEXS3] deliverable:

- planning of syntactic aggregation, i.e. coordination involving conjunction or disjunction;
- planning of discourse aggregation, i.e. RST-relations;
- planning of sequence markers;
- planning of different ways of incorporating constraints into running text, if possible in the given context;
- planning of information structure using an elementary discourse model;
- planning of an additional (personal) style.

With each addition we describe the new systems, and their underlying choosers and inquiries. All the nodes bearing information about how to realise that particular node (for example, with a sequence marker), or how to relate various nodes (for example, using coordination or an RST-relation), are interpreted by the sentence planner.

3.2 Planning of syntactic aggregation

3.2.1 Description of the phenomena

Coordinations involving conjunction and disjunction are two kinds of syntactic aggregation that follow quite naturally from two configurational concepts in the AGILE Domain Model. Namely, a PROCEDURE has a METHODS field (of type METHOD-LIST) that lists the alternative ways of obtaining the PROCEDURE's GOAL. This usually can be translated into disjunction, particularly when we are aiming at generating running text:

To save a document
Select the *Save* command from the *File* menu
Click on the *Save* icon in the toolbar

To save a document: Select the *Save* command from the *File* menu or click on the *Save* icon in the toolbar.

Similarly, a METHOD that specifies one way to achieve a PROCEDURE's GOAL, has a SUBSTEPS field (of type PROCEDURE-LIST) that gives the PROCEDUREs or "steps" that have to be taken. In other words, the SUBSTEPS field specifies a sequence, which we can conceive of as a form of conjunction.

To save a document under a different name

1. Select the *Save As* command from the *File* menu.
2. Enter the file name in the *File name* box.
3. Click the OK button.

To save a document under a different name, select the *Save* command from the *File* menu, enter the file name in the *File name* box, and click the OK button.

Neither a conjunction nor a disjunction should be planned under *all* circumstances. We need to have restrictions on their planning for the resulting text to appear natural. To that end, we suggest here to introduce a conjunction into the text plan when we have two TASKs that are *simplex*, i.e. which do not specify side-effects nor METHODs. When we construct the part of a text plan corresponding to a METHOD's SUBSTEPS, then above each step (a TASK) we have a node that is either labelled "CONJOINED-INSTRUCTION-TASKS" to indicate conjunction or "SEPARATE-INSTRUCTION-TASKS" to signify that the TASK under that node should not be conjoined to the preceding TASK(s).

We can take a similar approach for disjunction. As we pointed out above, the possibility for disjunction arises when a PROCEDURE specifies several METHODs as members of its METHODS' METHOD-LIST. Each of these METHODs present an alternative way of obtaining the PROCEDURE's GOAL.

By its definition in the Domain Model a METHOD must have a non-empty SUBSTEPS field, and may specify a CONSTRAINT and/or a PRECONDITION. A CONSTRAINT is of type OPERATING-SYSTEM, like "Windows" in the examples above. A PRECONDITION is a PROCEDURE that should be executed before the METHOD's SUBSTEPS can be carried out.

Again, we propose a simplicity measure. We should only insert a disjunction (realised later as an "or") between METHODs if each METHOD is simplex, i.e. it does not specify a CONSTRAINT nor a PRECONDITION. However, a simplicity measure in the case of disjunction is a bit more complex than it was for conjunction. The uncomplicated situation is that the *entire* list consists of METHODs none of which specify a CONSTRAINT or PRECONDITION. In that case we can straightforwardly allow for disjunction.

However, the situation becomes more complicated in case no METHODs in the list specify a PRECONDITION but all METHODs *do* specify a CONSTRAINT. In that case we should keep separated METHODs that have different CONSTRAINTs, but we can put a disjunction between METHODs that follow upon one another and which have identical CONSTRAINTs. For example, the simple text below could be specified using one PROCEDURE and three METHODs. Two of these METHODs would specify alternative ways of saving a document under a different name when using Windows, whereas the third METHOD would specify the step needed to achieve that same GOAL under Dos.

To save a document under a different name:

Windows: Select *Save As …* from the *File* menu or click the *Save As...* icon on the toolbar

Dos: Select *Save As...* from the *File* menu.

3.2.2 The specifications for conjunction and disjunction - in brief

We briefly repeat here the formal specifications as we provided them in [TEXS3]. Formally, we specified conjunctions as follows.

To begin with, we are dealing here with TASKS and their planning as members of an INSTRUCTION-TASKS list. The systems in the CADCAM-INSTRUCTIONS region that are responsible for that planning are given in Figure 3.

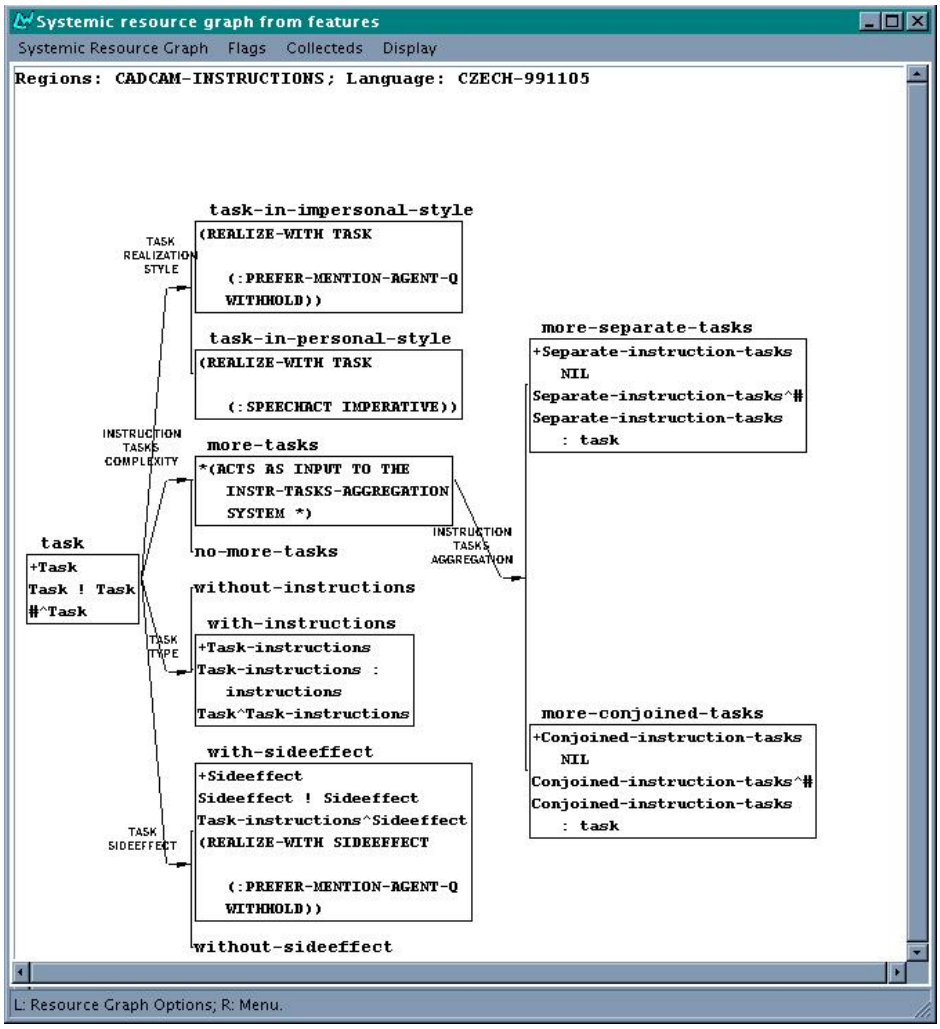


Figure 3: Systems dealing with Tasks

Instead of immediately introducing an INSTRUCTION-TASKS node in the system INSTRUCTION-TASK-COMPLEXITY we can decide to introduce only the abstract feature *More-Tasks* and let that feature be an entry condition to another system INSTRUCTION-TASKS-AGGREGATION where we decide about conjoinability.

The decision for introducing a node SEPARATE-INSTRUCTION-TASKS or CONJOINED-INSTRUCTIONS-TASKS depends whether the TASK that we will be introducing under this node will be simplex or not, as we discussed above.

System:

```
More-tasks →
(more-conjoined-tasks
      insert CONJOINED-INSTRUCTION-TASKS
      preselect CONJOINED-INSTRUCTION-TASKS TASK
      orderatend CONJOINED-INSTRUCTION-TASKS
)
(more-separated-tasks
      insert SEPARATE-INSTRUCTION-TASKS
      preselect SEPARATE-INSTRUCTION-TASKS TASK
      orderatend SEPARATE-INSTRUCTION-TASKS
)
```

Chooser:

```
Choose more-conjoined-tasks if conjoinable
else choose more-separated-tasks
```

The chooser itself relays the question about the conjoinability of the TASK to be inserted to an inquiry that looks in the A-box whether the PROCEDURE, to which the TASK is to correspond, specifies a SIDE-EFFECT and/or METHODS. If the PROCEDURE has neither a SIDE-EFFECT nor METHODS then we consider the PROCEDURE (and the TASK) simplex:

Inquiry:

```
Consider root to be the PROCEDURE to be considered next
if (get-abox-slot dm::side-effect)
      or (get-abox-slot dm::METHODs)
then return NIL   ;; not simplex
else return TRUE  ;; simplex, no side-effect nor METHODs
```

We provide the following formal specification for deciding to include disjunction into the text plan. To begin with, what we are dealing with METHODs (and their SUBSTEPS) that are specified as members of a METHOD-LIST. In text planning a METHOD corresponds to an INSTRUCTION, and the METHOD-LIST being the value of the PROCEDURE's METHODS field corresponds to TASK-INSTRUCTIONS in a way similar to the INSTRUCTION-TASKS construction above. The relevant systems are shown in Figure 5.

Again, instead of immediately introducing a TASK-INSTRUCTIONS node in the system TASK-INSTRUCTIONS-COMPLEXITY we introduce only the abstract feature *More-Instructions* and let that feature be an entry condition to an other system TASK-INSTRUCTIONS-AGGREGATION where we decide about disjoinability.

System:

```
More-Instructions →
(more-disjoined-instructions
      insert DISJOINED-TASK-INSTRUCTIONS
      preselect DISJOINED-TASK-INSTRUCTIONS INSTRUCTIONS
      orderatend DISJOINED-TASK-INSTRUCTIONS
)
(more-separate-instructions
      insert SEPARATE-TASK-INSTRUCTIONS
      preselect SEPARATE-TASK-INSTRUCTIONS INSTRUCTIONS
      orderatend SEPARATE-TASK-INSTRUCTIONS
)
```

Chooser:

```
Choose more-disjoined-instructions if disjoinable
else choose more-separated-instructions
```

Inquiry:

```
;; if we are at the beginning of a new METHOD-LIST then
;; current-constraint must be reset to nil.

Consider the root to be the METHOD to be considered next
if the entire list contains METHODS that are simplex
then
      return TRUE ;; can insert disjunction
else
      if (equal (get-abox-slot DM::constraint)
            (current-constraint))
      then  return TRUE
      else  current-constraint = (get-abox-slot DM::constraint)
            return NIL
```

Based on these formal specifications, we implemented planning of conjunction and disjunction in the following way.

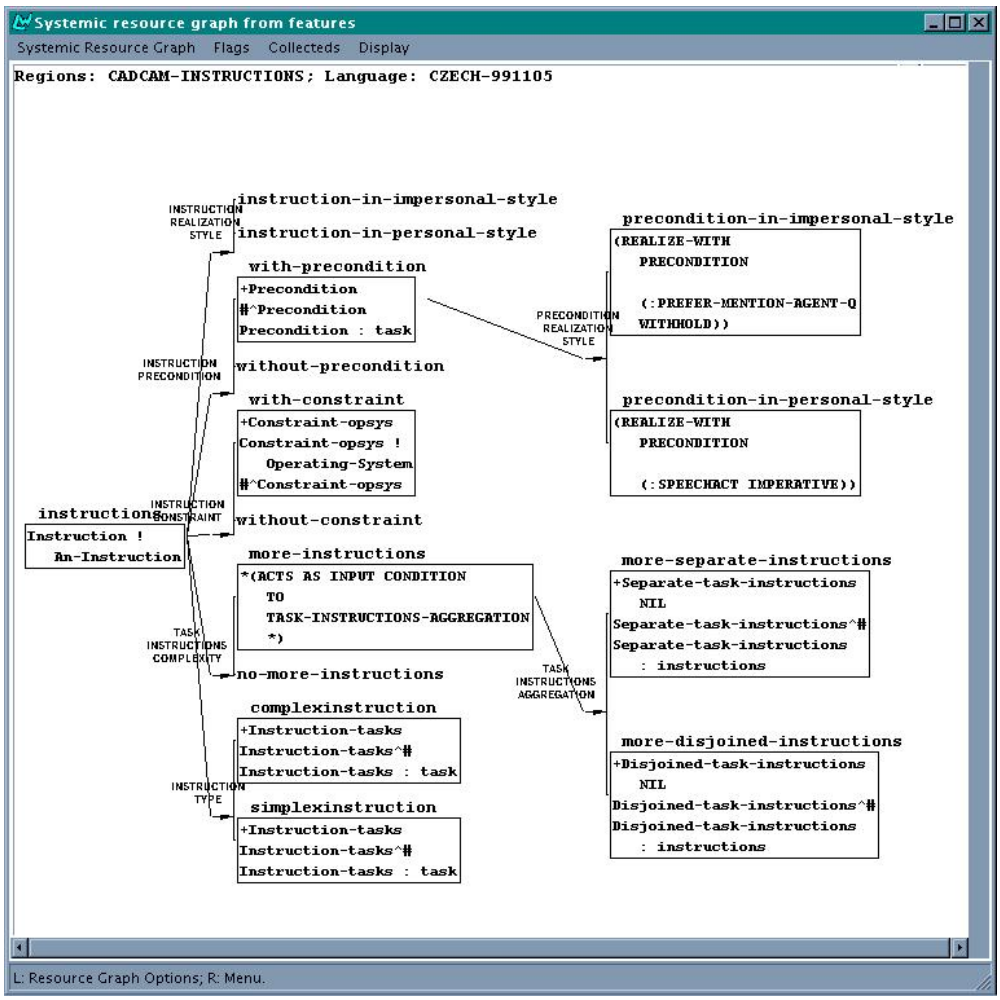


Figure 4: Systemic region focusing on INSTRUCTION

3.2.3 Specification of explicit sequence marking of conjoined tasks

This is a rudimentary attempt to interleave aggregation and explicit marking of sequencing using sequence discourse markers in addition to a coordination connective. For a discussion of the use of linguistic markers as means of sequence realisation see Section 3.5.

For conjoined tasks, explicit sequence markers are used when there are more than two conjoined Tasks. No explicit sequence markers are used for less than two.

```
System: AGGREGATED-INSTRUCTION-TASKS-SEQUENCE
    More-Conjoined-Tasks →
          [Conjoined-Tasks-With-SeqM]
          [Conjoined-Tasks-Without-SeqM]

Chooser: AGGREGATED-INSTRUCTION-TASKS-SEQUENCE-CHOOSER
    If there are more than two SUBSTEPS,
    Then choose Conjoined-Tasks-With-SeqM
    Else choose Conjoined-Tasks-Without-SeqM
```

The difference between conjunction with and without explicit markers is reflected by the sentence planner as follows:

- For a conjunction with an explicit sequence discourse marker the sentence planner introduces RST-Sequence into the SPL.
- For a conjunction without an explicit sequence discourse marker the sentence planner introduces Conjunction in to the SPL.

3.2.4 Implementation of planning conjunctions and disjunctions

The system responsible for planning a conjunction between Tasks is the INSTRUCTION-TASKS-AGGREGATION system, displayed in Figure 5.

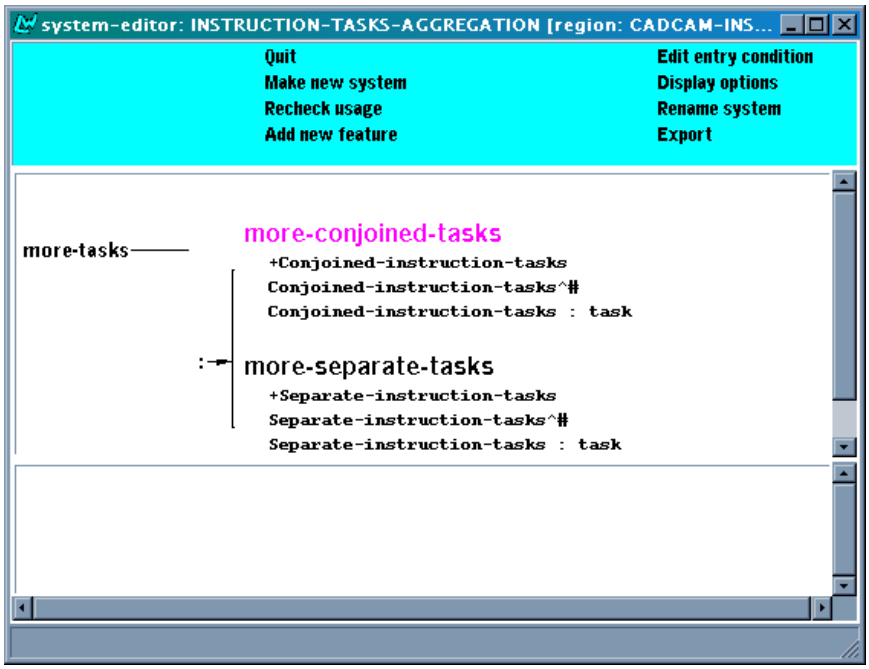


Figure 5 - INSTRUCTION-TASKS-AGGREGATION system

The entry-condition to the system is the feature "more-tasks" - it only makes sense to consider the possibility of aggregating Tasks using conjunction if there *is* actually more than one Task. The chooser implementing the system's decision process is displayed in Figure 6.

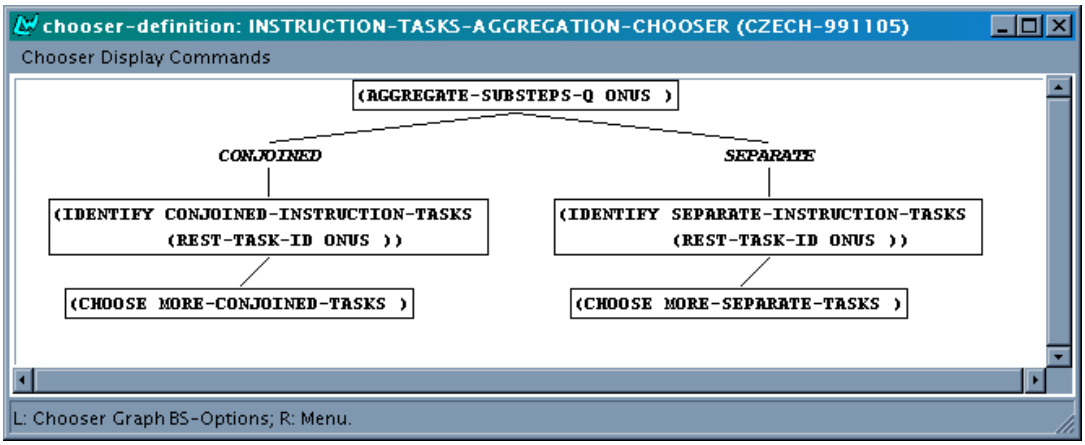


Figure 6 - INSTRUCTION-TASKS-AGGREGATION chooser

The actual decision for aggregation Tasks is made by the inquiry AGGREGATE-SUBSTEPS-Q, which asks of the onus (which at this point pointing to a PROCEDURE-LIST) whether the PROCEDUREs contained therein are SIMPLEX in the way defined above. If so, the chooser decides to conjoin the tasks, after which the system plans that aggregation by inserting a CONJOINED-INSTRUCTION-TASKS node in the textplan. This node gets identified with the relevant PROCEDURE-LIST in the A-box, so that we know what part of the A-box's content we are dealing with.

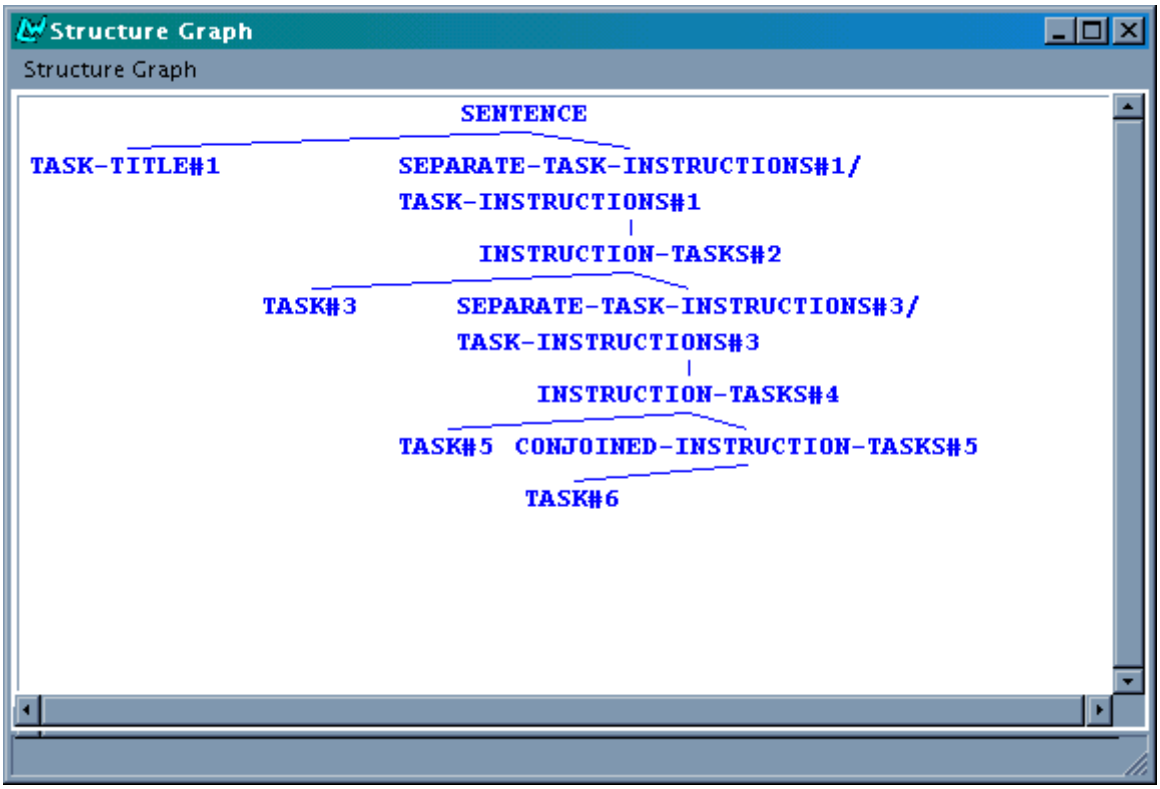


Figure 7 - Text plan including conjunction

The sentence planner interprets the CONJOINED-INSTRUCTION-TASKS node, and creates the proper SPL code for it. For example, for the textplan given in Figure 7 we would get the following SPL code:

```
(G6306 / CONJUNCTION :DOMAIN
       (|a19| / DM::CLICK :PREFER-MENTION-AGENT-Q WITHHOLD :ACTEE
              (|a23| / DM::DRAWING :CONTEXTUAL-BOUNDNESS DM::YES)
              :SPATIAL-LOCATING
              (|a24| / THREE-D-LOCATION :LEX NABI2DKA
                     :CONTEXTUAL-BOUNDNESS YES)
              :ACTOR
              (HEARER / DM::USER :IDENTIFIABILITY-Q DM::IDENTIFIABLE))
       :RANGE
       (|a27| / DM::ENTER :PREFER-MENTION-AGENT-Q WITHHOLD :ACTEE
              (|a31| / THREE-D-LOCATION :LEX SOUBOR)
              :ACTOR
              (HEARER / DM::USER :IDENTIFIABILITY-Q DM::IDENTIFIABLE)))
```

Just as an aside: the field " :CONTEXTUAL-BOUNDNESS YES" identifies for example the drawing as a contextually bound element, i.e. it has already been mentioned in the previous discourse. This information is taken into account when realising the sentence's word order [IMPL3, Kruijff-Korbayová et al, 2000].

The system dealing with disjunctions has been constructed in a similar way, though this time we take as entry condition the feature signifying the fact that there are more Instructions (i.e. more than one METHOD in the currently examined METHOD-LIST):

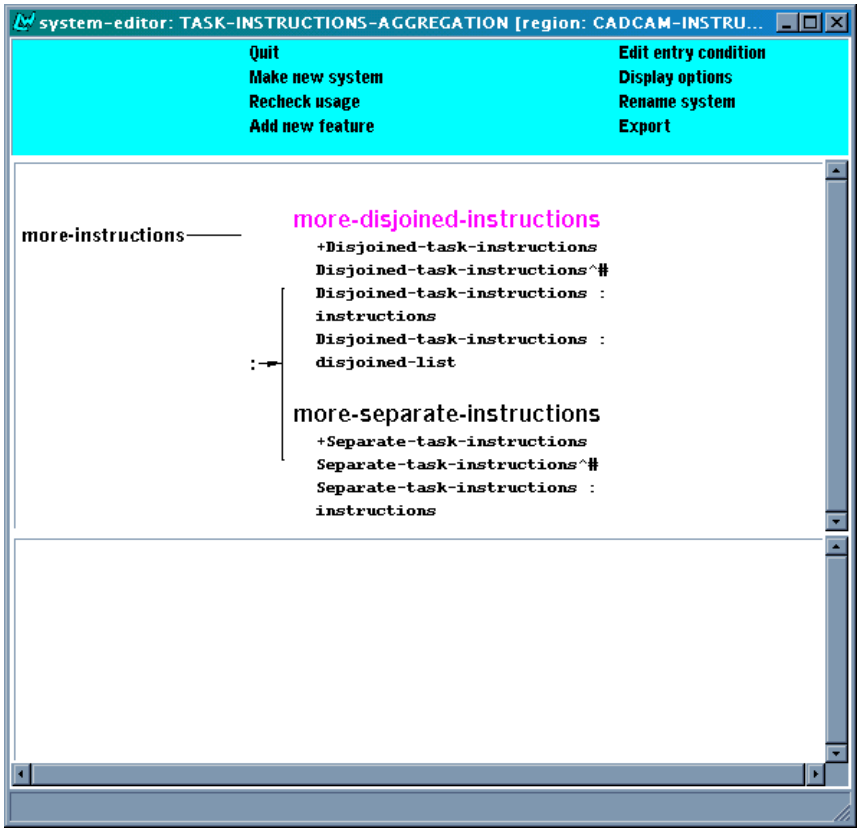


Figure 8 - TASK-INSTRUCTIONS-AGGREGATION system

The chooser underlying the system in Figure 8 is displayed in Figure 9.

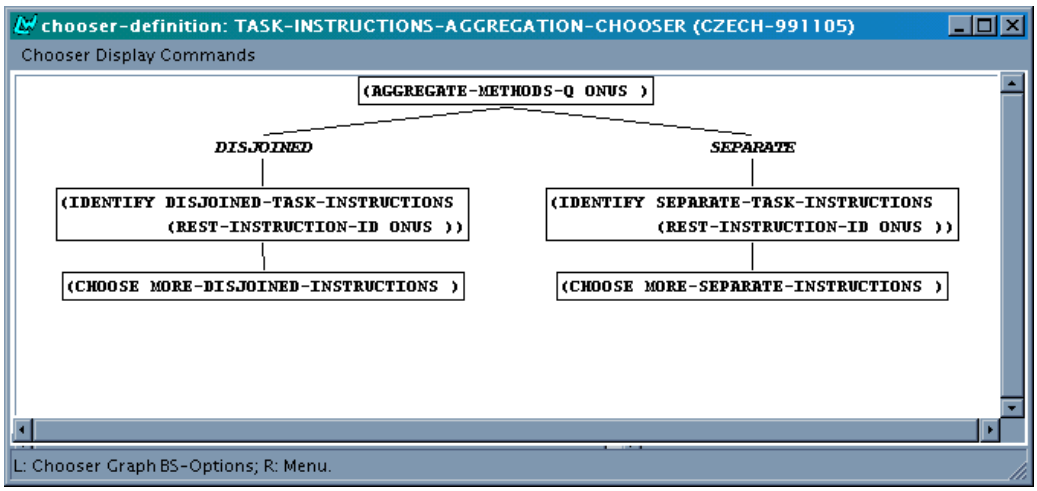


Figure 9 - TASK-INSTRUCTIONS-AGGREGATION chooser

The inquiry AGGREGATE-METHODS-Q follows the specification, and examines the Onus which (at the point of investigation) refers to a METHOD-LIST in the A-box. Once the chooser decides for introducing a disjunction, the system introduces a DISJOINED-TASK-INSTRUCTIONS node in the textplan, which gets identified with the examined METHOD-LIST in the A-box.

The SPL code realising a disjunction looks as follows, for example as for the disjunction shown in the textplan in Figure 2:

```
(G6319 / DISJUNCTION :DOMAIN
      (|a19| / DM::CLICK :PREFER-MENTION-AGENT-Q WITHHOLD :ACTEE
           (|a23| / DM::DRAWING :CONTEXTUAL-BOUNDNESS DM::YES)
           :SPATIAL-LOCATING
           (|a24| / THREE-D-LOCATION :LEX NABI2DKA
                  :CONTEXTUAL-BOUNDNESS DM::YES)
           :ACTOR
           (HEARER / DM::USER :IDENTIFIABILITY-Q DM::IDENTIFIABLE))
       :RANGE
       (|a27| / DM::ENTER :PREFER-MENTION-AGENT-Q WITHHOLD :ACTEE
            (|a31| / THREE-D-LOCATION :LEX SOUBOR)
            :ACTOR
            (HEARER / DM::USER :IDENTIFIABILITY-Q DM::IDENTIFIABLE)))
```

3.2.5 Implementation of explicit sequence marking of conjoined tasks

The system which makes the choice between using explicit sequence marking in addition to conjunction is implemented as shown in Figure 10.

Two example SPLs obtained for the two cases are shown below. The first SPL yields the Czech realisation of 'Add an element and save the style', the second SPL yields the counterpart of 'Add an element, then save the style and then press OK'.

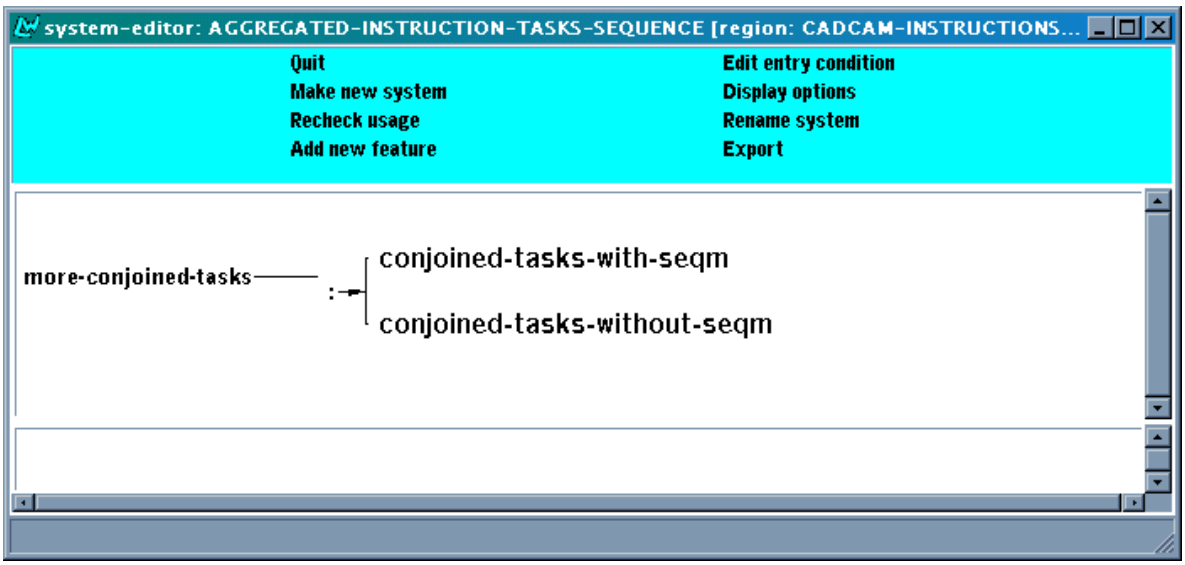


Figure 10: Aggregated-Instruction-Tasks-Sequence system

```
(EXAMPLE
:NAME   5-add-conj
:SET-NAME   CONJ
:TARGETFORM   "Přidejte element a uložte styl"
:LOGICALFORM
    (c1 / conjunction
    :Domain
          (S1 / DM::ADD :SPEECHACT IMPERATIVE
                :ACTEE (P1 / DM::style-element))
    :Range
          (S2 / DM::SAVE :SPEECHACT IMPERATIVE
                         :ACTEE (P1 / DM::STYLE))))

(EXAMPLE
:NAME   5-add-RSTseq
:SET-NAME   CONJ
:TARGETFORM  "Přidejte element, potom uložte styl a potom stikněte OK."
:LOGICALFORM
(c1 / RST-SEQUENCE
    :Domain
          (S1 / DM::ADD :SPEECHACT IMPERATIVE
                         :ACTEE (P1 / DM::style-element))
    :Range
          (c2 / RST-SEQUENCE
                :Domain (S2 / DM::SAVE
                            :SPEECHACT IMPERATIVE
                            :ACTEE (P1 / DM::style))
                :Range (S3 / DM::PRESS
                            :SPEECHACT IMPERATIVE
                            :ACTEE (P2 / OBJECT :NAME gui-ok)))))
```

3.3 Planning of discourse aggregation

3.3.1 Description of discourse aggregation in the final prototype TSM

Conjunction and disjunction hold between text structure elements that are of the same type, namely TASKs respectively INSTRUCTIONs. The discourse relations RST-PURPOSE and RST-MEANS are between text structure elements that are of *different* types. Both discourse relations can be used, under certain restrictions, to couple a TASK to an INSTRUCTION.

Following [Hartley et al, 1996] we make a distinction between *enablement* and *generation*, as we discussed already at some length in [TEXS3]. Here, we make a distinction between the two relations based on the types of concepts involved, along the following lines:

- If the concept of the Task that will act as satellite is of any of the following types, then the proper RST-relation is RST-MEANS: Click, Double-Click, Enter, Press, and Repeat-Steps.
- Otherwise, the RST-PURPOSE is the most appropriate relation to be used.

In the IMD TSM these relations were *not* planned by the text planner, but by the sentence planner. As a consequence, there was very little flexibility in specifying when these relations could, or should, be used. We already described in the previous chapter how this has changed. Compare for example the two textplans below: Figure 11 shows a textplan without any explicit RST relations planned (as produced in the intermediate prototype), and Figure 12 shows a textplan for an almost identical A-box, now *with* the RST-relation made explicit.

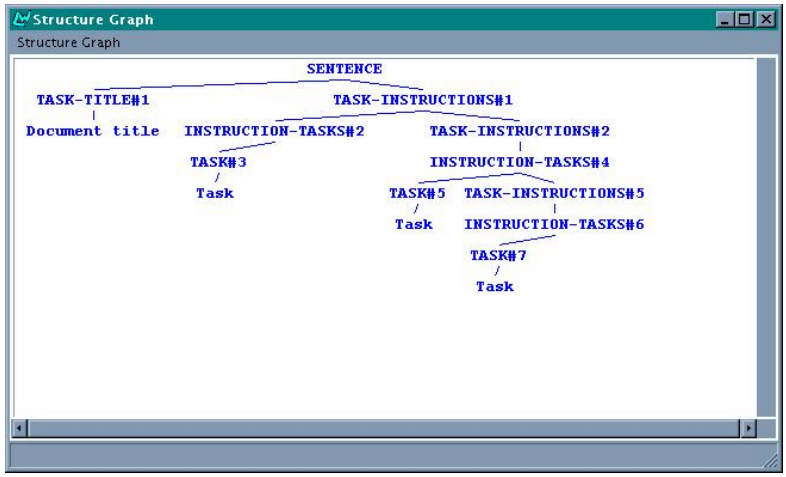


Figure 11 - IMD TSM textplan without RST relation

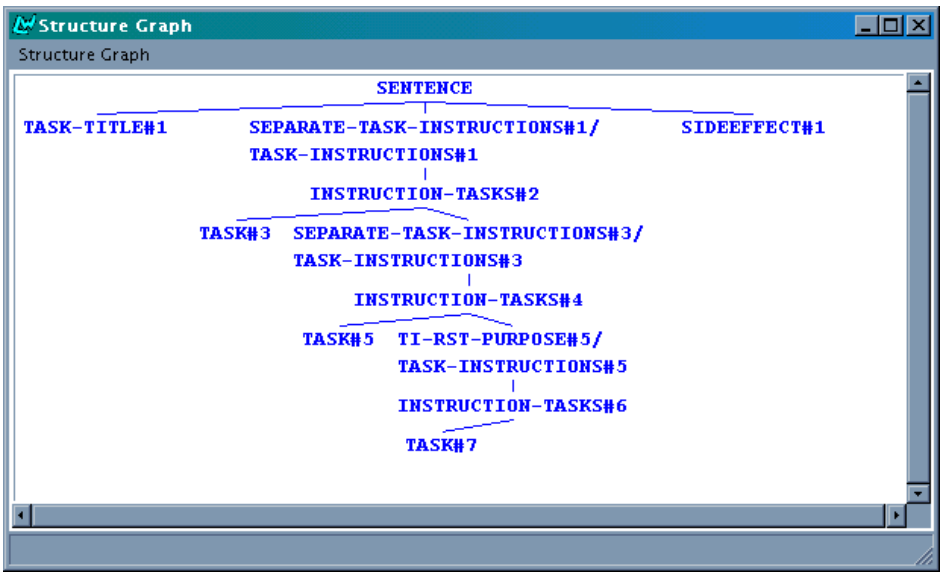


Figure 12 - FP TSM textplan, with RST relation

3.3.2 Specifications for RST-relations

In [TEXS3], we specified the systems needed for planning RST-relations explicitly, as follows.

In the text plan, a TASK is related to an INSTRUCTION by means of a TASK-INSTRUCTIONS node, as was shown in Figure 11 above. The TASK-INSTRUCTIONS node corresponds to the METHOD-LIST that fills the METHODS slot of the PROCEDURE to which the TASK corresponds. As we pointed out in [TEXS3], we will use this node to signify explictly the discourse relation that is to hold between the TASK and an INSTRUCTION.[1] We already saw in Figure 3 that the first TASK-INSTRUCTIONS would get inserted in the TASK-TYPE system, depending on whether the underlying PROCEDURE has a filled METHODS field or not.

Instead of directly inserting a TASK-INSTRUCTIONS node in the TASK-TYPE system, we propose to apply a similar construction here as we employed above for conjunctions, namely deferring the decision for insertion of a particular type of TASK-INSTRUCTIONS node to a different system to which the feature *with-instructions* acts as entry condition.

We specified the system TASK-INSTRUCTION-DISCOURSE-RELATIONS where we wouldddecide whether a TASK and its INSTRUCTION(s) should be explicitly related through a discourse relation, and if so, which relation. For each discourse relation we want to be able to distinguish, there is a choice in the system. Each discourse relation gets signified in the text plan by a node bearing its name, for example TI-RST-MANNER or TI-RST-PURPOSE (with “TI” indicating aggregation of a TASK and its INSTRUCTION(s)).

The system can be formally specified as follows:

[1] More specifically, the TASK-INSTRUCTIONS node that connects TASK to its INSTRUCTION(s) corresponds to the first METHOD in the METHOD-LIST. As we discussed in the previous section, subsequent METHODs in that list will be placed under nodes that signify either disjunction or separation. The very first TASK-INSTRUCTIONS node obviously does not signify any such relation.

```
System: [TASK-INSTRUCTIONS-DISCOURSE-RELATIONS]
    with-instructions →
    (ti-separate
          insert TASK-INSTRUCTIONS
          preselect TASK-INSTRUCTIONS INSTRUCTIONS
          orderatend TASK-INSTRUCTIONS
    )
    (ti-means
          insert TI-RST-MEANS
          preselect TI-RST-MEANS INSTRUCTIONS
          orderatend TI-RST-MEANS
    )
    (ti-purpose
          insert TI-RST-PURPOSE
          preselect TI-RST-PURPOSE INSTRUCTIONS
          orderatend TI-RST-PURPOSE
    )
```

In [TEXS3] we also discussed the possibilities for different orderings of the satellite and nucleus in an RST-relation, apart from the canonical order. As it turned out, the important decisions to be made there could not be made -even- at the level of sentence planning, since they mostly relied on how a grammar would decide to realise particular content. Below we will therefore defer these decisions to the grammar.

3.3.3 Implementation

The system implementing the formal specifications given above is TASK-INSTRUCTION-DISCOURSE-RELATION, as displayed in Figure 13 below.

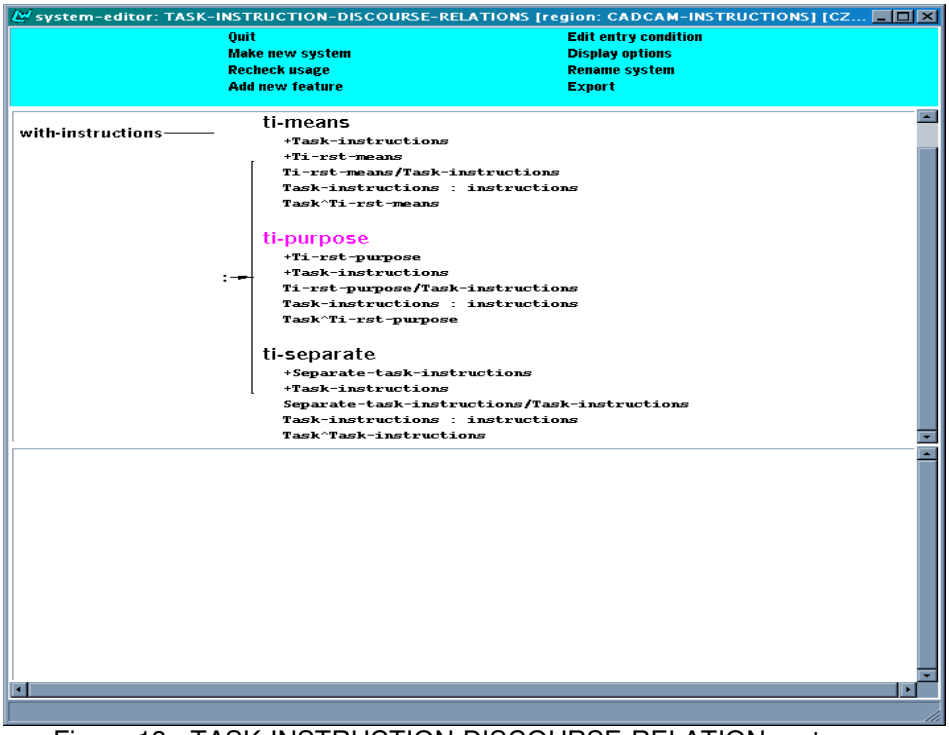


Figure 13 - TASK-INSTRUCTION-DISCOURSE-RELATION system

The system's chooser is as follows, Figure 14.

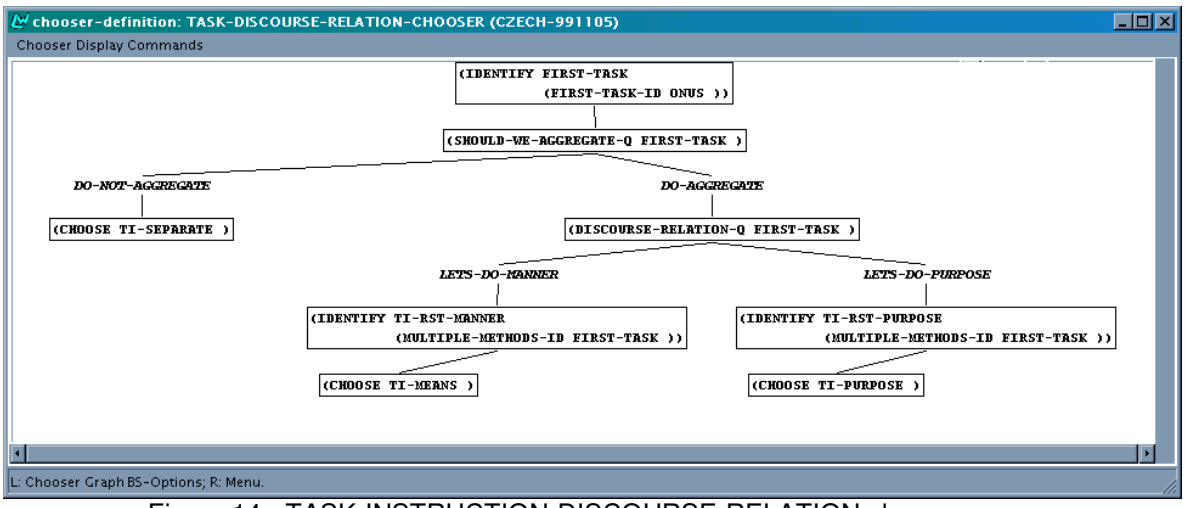


Figure 14 - TASK-INSTRUCTION-DISCOURSE-RELATION chooser

At the point that the system is called, the onus is pointing to a PROCEDURE-LIST and not a PROCEDURE. The first thing that happens, therefore, is that we pick up the first PROCEDURE of that list, and identify FIRST-TASK with it.

Subsequently, the chooser invokes the inquiry SHOULD-WE-AGGREGATE-Q on FIRST-TASK. The decision whether or not to aggregate essentially follows the above specification. (The exact implementation being more explicit in that it traverses the A-box structure to check the complexity of the content under FIRST-TASK.) Once aggregation is possible, the kind of RST-relation is determined on the basis of the concept that is instantiated by the PROCEDURE of the METHOD of the METHOD-LIST filling the METHODS field of the PROCEDURE to which FIRST-TASK corresponds. We identify the *node* in the textplan with the METHOD-LIST that fills FIRST-TASK's PROCEDURE's METHODS field, such that later on the Task corresponding to the satellite of the RST-relation can be planned.

The SPL code produced for the RST-relations is similar to that produced in the intermediate prototype, for example:

```
(G6384 /
       (RST-PURPOSE)
       :DOMAIN
       (|a28| / DM::CHOOSE :PREFER-MENTION-AGENT-Q WITHHOLD :ACTEE
           (|a30| / OBJECT :LEX TYP :GENERALIZED-POSSESSION-INVERSE
                     (G5796 / DM::LINE)
                     :GENERALIZED-POSSESSION-INVERSE
                     (|a15| / DM::LINE :CONTEXTUAL-BOUNDNESS DM::YES)
                     :CONTEXTUAL-BOUNDNESS DM::YES)
           :ACTOR
           (HEARER / DM::USER :IDENTIFIABILITY-Q DM::IDENTIFIABLE))
       :RANGE
       (|a21| / DIRECTED-ACTION :PREFER-MENTION-AGENT-Q WITHHOLD :LEX
            ZAC3I2T :ACTEE
            (G5778 / OBJECT :LEX KRESLENI2)
            :ACTEE
            (|a15| / DM::LINE :CONTEXTUAL-BOUNDNESS DM::YES)
            :ACTOR
            (HEARER / DM::USER :IDENTIFIABILITY-Q DM::IDENTIFIABLE)))
```

3.4 Granularity

3.4.1 Description and specification

Under the heading of aggregation we also discussed, in [TEXS3], the idea of *granularity*. In our case, we can consider the integration of a METHOD's CONSTRAINT directly into the sentence that realises the METHOD's SUBSTEPS, as in the following examples:

> In **Windows,** choose *Multiline style* from the *Object Properties* toolbar.
> In **DOS or UNIX,** choose *Multiline style* from the *Data* menu.
>
> Using **Windows,** you should choose *Multiline style* from the Object Properties toolbar.
> Using **DOS or UNIX,** you should choose *Multiline style* from the Data menu.
>
> If you are using **Windows,** you should choose *Multiline style* from the Object Properties toolbar.
> If you are using **DOS or UNIX,** you should choose *Multiline style* from the Data menu.

Here, we consider a CONSTRAINT to identify a specific context of application, and as such can be realised as a location (as in the first pair of sentences), as an instrument (as in the second pair of sentences), or by a conditional construction (as in the third pair of sentences). All three realisations are consistent with the intuitions behind the CONSTRAINT concept as it is formulated in the Domain Model.

In the text plan we can already plan the inclusion of the CONSTRAINT, like in the examples above. We can do so by differentiating different types of CONSTRAINT-nodes in the text plan, in a similar vein as we did above. Here, we will use SEPARATE-

CONSTRAINT, LOCATION-CONSTRAINT, MEANS-CONSTRAINT and CONDITION-CONSTRAINT. The reason for making the differentiation already during text planning rather than deferring to the stage of sentence planning, is that the decision for one type over the other depends on

- the overall type of text we are planning, i.e. running text or using enumeration; and,
- what style the text is to be generated in.

In case the text is to be realised in imperative voice using enumeration, then the only natural choice appears to be to use SEPARATE-CONSTRAINT, which keeps the CONSTRAINT separate from the sentence realising the METHOD's SUBSTEPS. The reason being that there is no explicit mention of the hearer (ruling out "*If you are using **Windows**, choose ...*"). Hence, in this case the text planner should insert SEPARATE-CONSTRAINT into the text plan.

On the other hand, if we are to generate a running text, we can use LOCATION-CONSTRAINT if the imperative voice is used, or either MEANS-CONSTRAINT or CONDITION-CONSTRAINT, depending on the grammatical constructions available in a given language, if active voice is used. For example, the MEANS-CONSTRAINT realisation is not available in Czech, but is possible in Russian.

Upon encountering a constraint under one of these nodes in the text plan, the sentence planner should act as follows:

- SEPARATE-CONSTRAINT: A separate sentence-plan should be created, realising the CONSTRAINT. This is the way it is done already in the TSM for the intermediate prototype. The corresponding SPL contains the following statement:
  ```
  (P / object :NAME gui-windows)
  ```
- LOCATION-CONSTRAINT becomes a Spatial-Locating Circumstance within the clause realising the TASK. It is reflected by the following bit of SPL code:
  ```
  :spatial-locating(P / object :NAME gui-windows)
  ```
- MEANS-CONSTRAINT is realised by the Range of RST-Means, where the Domain corresponds to the TASK. The following bit of SPL code reflects this:
  ```
  (RST / RST-Means Manner
     :Domain …
     :Range(Range / dispositive-material-action
       :lex use :ACTEE (P / object :NAME gui-windows)))
  ```
- CONDITION-CONSTRAINT is realised by the Domain of RST-Logical-Condition, where the Range corresponds to the TASK. The following bit of SPL code reflects this:
  ```
  (RST / RST-Logical-Condition
     :Domain(Range / dispositive-material-action
       :lex use :ACTEE (P / object :NAME gui-windows)
     :Range))
  ```

3.4.2 Implementation

The above specification has been implemented as the following system, Figure 15.

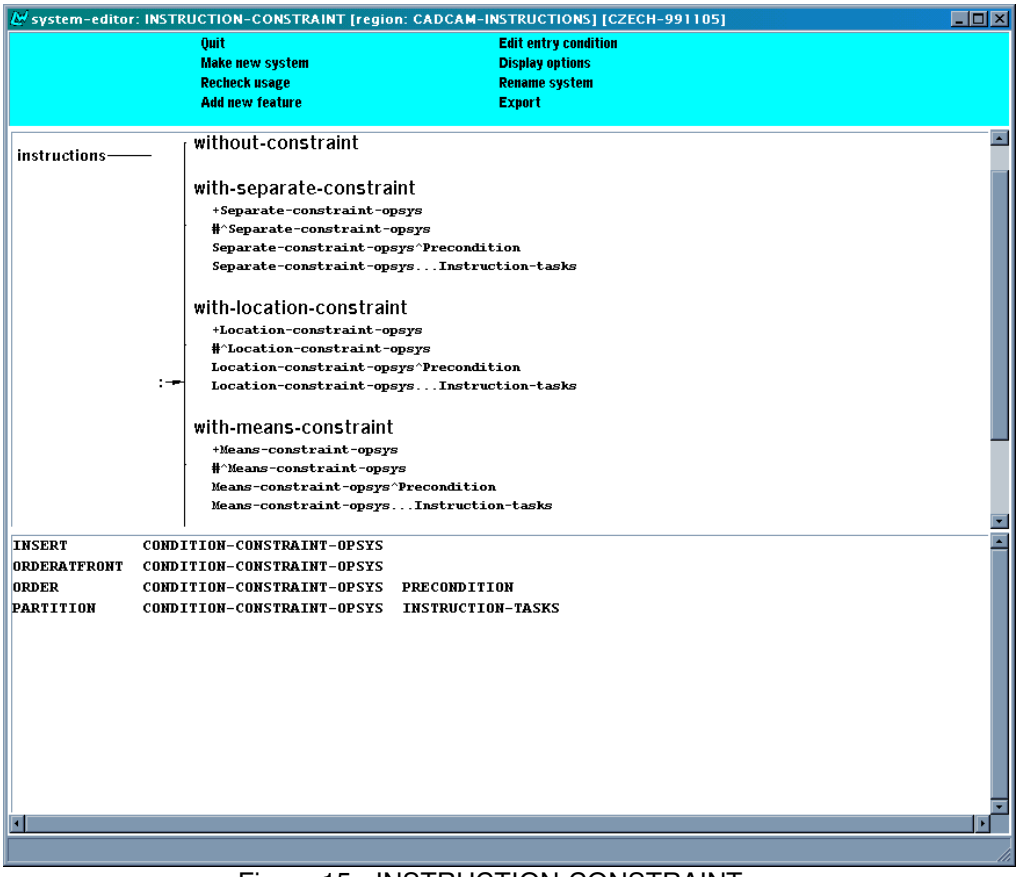
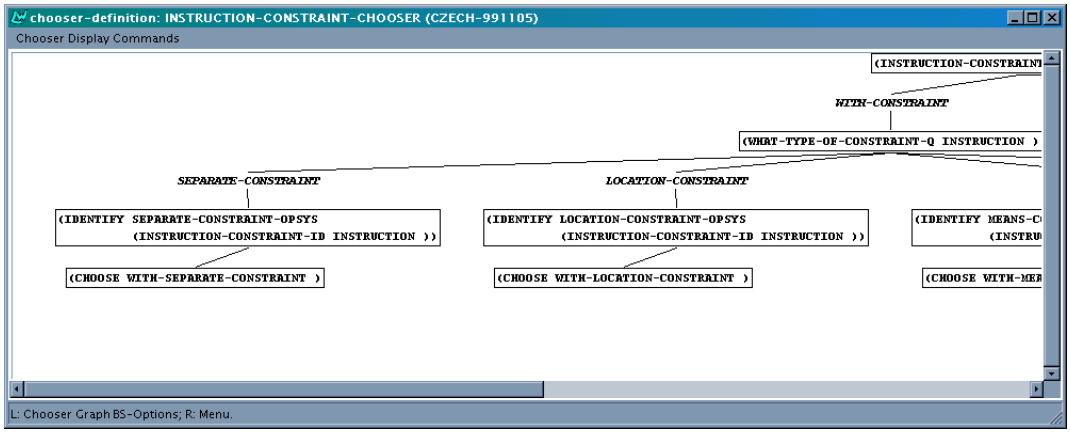


Figure 15 - INSTRUCTION-CONSTRAINT

The chooser underlying the INSTRUCTION-CONSTRAINT system is displayed (in two parts) below.



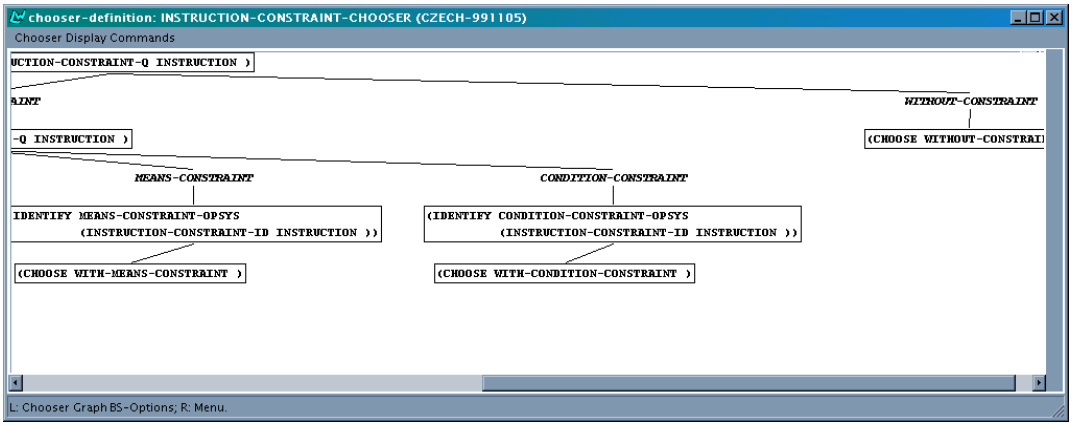


Figure 16 - INSTRUCTION-CONSTRAINT chooser

The choice whether or not to plan a constraint first of all depends on the presence of any such content in the CONSTRAINT slot of the METHOD to which the Onus refers. If there is no such content, there is no Constraint to be planned. Otherwise, a decision is to be made what type of Constraint. This decision has been implemented following the specifications, as follows:

- SEPARATE-CONSTRAINT if the text is not "running";
- LOCATION-CONSTRAINT if we are planning "running text" for the current level of steps, and the text is to be in personal style;
- MEANS-CONSTRAINT if the text is to be in active voice, and the language either Russian or Bulgarian;
- CONDITION-CONSTRAINT if the text is to be in active voice, and the language Czech.

Whether a text at a current level is to be planned as "running text" or not is a decision made when planning sequencing, and is discussed in the next section.

The SPL code generated by the sentence planner has already been given in the specifications above, at least as far as the folding in of the constraint into another SPL is concerned. We would like to remark here that, again, complexity is not an issue here - the text planner is free to create its text plan to any level of recursive depth, and the sentence planner will deal with. Hence, we can for example obtain an SPL combining a CONDITION-CONSTRAINT and a sequence of conjunctions, as in "When using Windows, do A, B, and C."

Particularly, as the specifications already discussed, if two methods have identical constraints to indicate that in the same operating system there are alternative ways of obtaining the stated goal, then we should not create a sentence plan *two* identical constraints. Instead, the methods should be headed by the constraint, and then appear as a disjunction. The text plan in Figure 17 shows how the textplan looks like. Note that besides the disjunction there is also a conjunction.

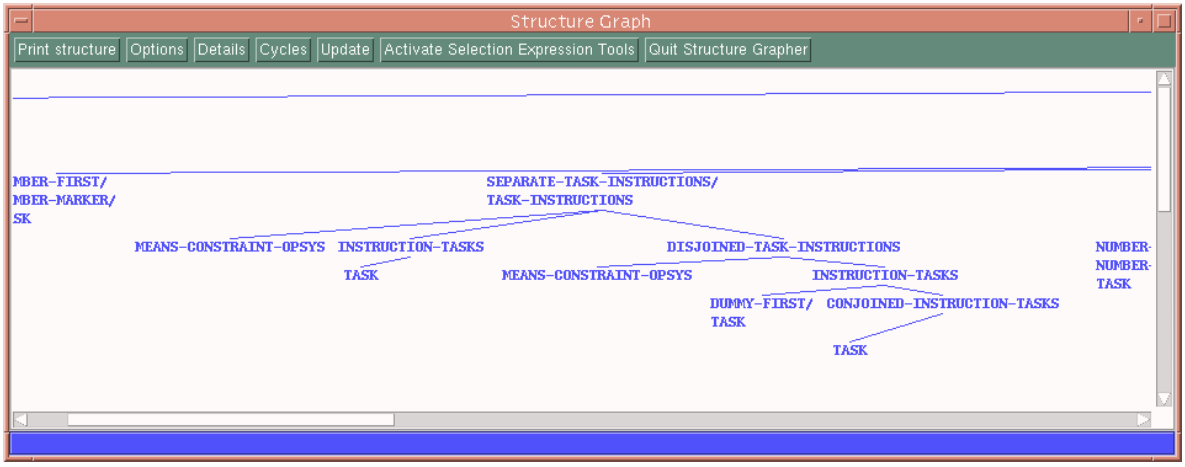


Figure 17 - Disjunction over identical constraints

The SPL code that specifies the disjunction in text plan (for Czech) is given below. Roughly, the sentence translates into "If you use Windows choose the 3points-option from the X menu in the Draw Arc panel or choose the Arc menu and choose the 3d-point option".

```
(G1323412 / DISJUNCTION
:DOMAIN
  (G1320889 / RST-LOGICAL-CONDITION
  :RANGE (G1320887 / DISPOSITIVE-MATERIAL-ACTION
        :LEX USE
        :ACTEE
          (|a18| / DM::OPERATING-SYSTEM OBJECT :LEX ELLIPSISZERO
           :CLASS-ASCRIPTION
       (|a19| / DM::GUI-WINDOWS :CONTEXTUAL-BOUNDNESS DM::YES)))
  :DOMAIN
     (|a22| / DM::CHOOSE
        :PREFER-MENTION-AGENT-Q WITHHOLD
        :ACTEE
             (|a24| / DM::OPTION
               :CLASS-ASCRIPTION
               (|a25| / DM::GUI-THREE-POINTS
                      :CONTEXTUAL-BOUNDNESS DM::YES)
               :CONTEXTUAL-BOUNDNESS DM::YES)
        :SOURCE
             (|a27| / THREE-D-LOCATION
               :LEX NABI2DKA
               :PROPERTY-ASCRIPTION
                   (G122405 / QUALITY
                       :LEX IKONOVY2
                       :PROPERTY-ASCRIPTION
                              (G122406 / QUALITY
                                    :LEX PLOVOUCI2))
                       :SPATIAL-LOCATING
                              (|a28| / THREE-D-LOCATION
                                    :LEX PANEL
                                    :PROPERTY-ASCRIPTION
                                     (G122436 / QUALITY :LEX NA2STROJOVY2)

                                 :CLASS-ASCRIPTION
                                        (|a29| / DM::GUI-DRAW
                                     :CONTEXTUAL-BOUNDNESS DM::YES)
                           :CONTEXTUAL-BOUNDNESS DM::YES)
                    :CLASS-ASCRIPTION
                              (|a30| / DM::GUI-ARC
                              :CONTEXTUAL-BOUNDNESS DM::YES)
                 :CONTEXTUAL-BOUNDNESS DM::YES)
        :ACTOR (HEARER / DM::USER
            :IDENTIFIABILITY-Q DM::IDENTIFIABLE)))
:RANGE
   (G1323397 / CONJUNCTION
      :DOMAIN
    (|a38| / DM::CHOOSE
           :ACTEE
            (|a40| / DM::OPTION
                 :CLASS-ASCRIPTION (|a41| / DM::GUI-ARC))
        :SOURCE (|a43| / THREE-D-LOCATION :LEX NABI2DKA)
        :ACTOR (HEARER / DM::USER
                  :IDENTIFIABILITY-Q DM::IDENTIFIABLE))
:RANGE
  (|a47| / DM::CHOOSE
     :PREFER-MENTION-AGENT-Q WITHHOLD
     :ACTEE (|a49| / DM::OPTION
              :CLASS-ASCRIPTION (|a50| / DM::GUI-THREE-POINTS))
     :ACTOR (HEARER / DM::USER
```

```
:IDENTIFIABILITY-Q DM::IDENTIFIABLE))))
```

3.5 Planning of sequence realisation styles

3.5.1 Description of the phenomena

In order for a reader to easily understand how to accomplish the task to which the instructions pertain, the instructional text needs to reflect in a perspicuous fashion the hierarchical organization of the tasks at hand as well as the sequence of steps to be carried out. The hierarchical organization of the tasks is reflected straightforwardly by the hierarchical organization of the text, or by the hypotactic relationships between clauses within clause complexes. The former is an issue of layout, while the latter is a part and parcel of aggregation (see Sections 3.3.1 and 3.3.2). The latter requires the generated text to reflect the sequencing of the bits of content that are being expressed, and mark the sequencing of steps overtly when necessary.

The bits of content as specified in an A-box are inherently sequentially related. The assumption is that the order of sub-steps specified in an A-box corresponds to the order in which the steps are to be carried out. Therefore, the generated instructions need to reflect this underlying "sequencing" of steps. The straightforward way of reflecting the step sequences is by ordering of the sentences in the output. However, when the input content gets more complex, more elaborate structuring of the generated text becomes needed. Two alternatives of realising such more complex content by a well-structured text are demonstrated in Figure 18 and Figure 19 (repeated from [TEXS3], section 3.2.1, where they were discussed in more detail).

To draw a line and arc combination polyline

1. Draw a line segment.

   First start the PLINE command using one of these METHODs:
   Windows: From the Polyline flyout on the Draw toolbar, choose Polyline.
   DOS and UNIX: From the Draw menu, choose Polyline.

   Then specify the start point of the line segment and the endpoint of the line segment.

2. Draw an arc segment.

   First switch to Arc mode by entering **a**. The Arc mode confirmation dialog box appears. Select OK.
   Then specify the endpoint of the arc.

3. Draw another line segment.

   First return to Line mode by entering **l**. The Line mode confirmation dialog box appears. Select OK.
   Then enter the distance and angle of the line in relation to the endpoint of the arc.

4. Press Return to end the polyline.

Figure 18: A combination of sequence styles: using numbered list for the top-level GOALs, and explicit sequence discourse markers for the lower-level GOALs, along with aggregation.

To draw a line and arc combination polyline

First draw a line segment.

1. Start the PLINE command using one of these METHODs:
   Windows: From the Polyline flyout on the Draw toolbar, choose Polyline.
   DOS and UNIX: From the Draw menu, choose Polyline.
2. Specify the start point of the line segment.
3. Specify the endpoint of the line segment.

Then draw an arc segment.

1. Enter **a** to switch to Arc mode. The Arc mode confirmation dialog box appears.
2. Select OK.
3. Specify the endpoint of the arc.

Then draw another line segment.

1. Enter **l** to return to Line mode. The Line mode confirmation dialog box appears.
2. Select OK.
3. Enter the distance and angle of the line in relation to the endpoint of the arc.

Finally, press Return to end the polyline.

Figure 19: A combination of sequence styles: using explicit sequence markers for the top-level GOALs within the list of steps, and a numbered list for the lower-level steps.

We have discussed in detail various alternative ways of realizing complex content in Section 3.2 in the [TEXS3] deliverable. In [TEXS3] section 3.2.1, we substantiated the need for explicit marking of sequences as a means to facilitate a reader's orientation in text describing complex structured tasks. Essentially, we proposed to distinguish between the following *sequence realization styles*:

- Running text sequence
  - Unmarked running text sequence (realised by a continuous paragraph where the elements in the sequence do not include any overt sequence markers)
  - Linguistically marked running text sequence (realised by a continuous paragraph where the elements in the sequence include overt sequence discourse markers)
- List sequences
  - Unmarked list sequences (realised by bullet lists in which the elements in the sequence do not include any overt sequence discourse markers)
  - Numbered list sequences (realised by numbered lists in which the elements in the sequence do not include any overt sequence discourse markers in addition to the numbering, which in itself constitutes sufficient overt sequence marking)
  - Linguistically marked list sequences (realised by lists of elements in which the elements in the sequence include overt sequence discourse markers)

By *(linguistic) sequence discourse markers* we mean expressions like 'firstly', 'secondly', 'first', 'second', 'third', etc., 'now', 'then', 'finally', 'lastly', 'further', etc. (and their Bulgarian, Czech and Russian counterparts, see Figure 21 below). The following examples (1), (2) and (3) illustrate simple sequences employing the three different sequence realisation styles of the same fragment of content in English, Czech, Bulgarian and Russian. The corresponding content model is shown in Figure 20.

(1) **Numbered list**

(a) *English*

1. Choose Element Properties. The Element Properties dialog box appears.
2. Enter the offset of the multiline element in the Element Properties dialog box.
3. Select Add to add the element.

(b) Czech

1. Vyberte Element Properties. Objeví se dialogové okno Element Properties.
2. V dialogovém okně Element Properties zadejte offset elementu multičáry.
3. Pro přidání elementu vyberte Add.

(c) Bulgarian

1. Изберете Element Properties. Появява се диалоговият прозорец Element Properties.
2. Въведете отместването на елемента на мултилинията в диалоговия прозорец Element Properties.
3. Изберете Add, за да добавите елемента.

(d) Russian

1. Нажмите кнопку Element Properties. Появится диалоговое окно Element Properties.
2. В диалоговом окне Element Properties введите смещение элемента мультилинии
3. Нажмите кнопку Add, чтобы добавить элемент.

(2) **Linguistically marked list**

(a) *English*

First choose Element Properties. The Element Properties dialog box appears.
Then enter the offset of the multiline element in the Element Properties dialog box.
Finally select Add to add the element.

(b) Czech

Nejprve vyberte Element Properties. Objeví se dialogové okno Element Properties.
Potom v dialogovém okně Element Properties zadejte offset elementu multičáry.
Nakonec vyberet Add pro přidání tohoto elementu.

(c) Bulgarian

Отначало изберете Element Properties. Появява се диалоговият прозорец Element Properties.
След това въведете отместването на елемента на мултилинията в диалоговия прозорец Element Properties.
Накрая изберете Add, за да добавите елемента.

(d) Russian

Сначала нажмите кнопку Element Properties. Появится диалоговое окно Element Properties.
Затем диалоговом окне Element Properties введите смещение элемента мультилинии.
И наконец Нажмите кнопку Add, чтобы добавить элемент.

(3) **Linguistically marked running text**

(a) *English*

First choose Element Properties. The Element Properties dialog box appears. Then enter the offset of the multiline element in the Element Properties dialog box. Finally select Add to add the element.

(b) Czech

Nejprve vyberte Element Properties. Objeví se dialogové okno Element Properties. Potom v dialogovém okně Element Properties zadejte offset elementu multičáry. Nakonec vyberte Add pro přidání tohoto elementu.

(c) Bulgarian

Отначало изберете Element Properties. Появява се диалоговият прозорец Element Properties. След това въведете отместването на елемента на мултилинията в диалоговия прозорец Element Properties. Накрая изберете Add, за да добавите елемента.

(d) Russian

Сначала нажмите кнопку Element Properties. Появится диалоговое окно Element Properties. Затем диалоговом окне Element Properties введите смещение элемента мультилинии. И наконец Нажмите кнопку Add, чтобы добавить элемент.

As is apparent from the classification of the sequence styles above, issues of *layout* and *linguistics realisation* are intertwined in the area of sequence realisation: While the distinction between a list and a running text is a matter of layout choice, the distinction between numbering and discourse markers is a matter of linguistic realisation choice, accompanied by explicit marking of the text structure. The text planning approach we are presenting here tries to balance out the use of the various sequence styles in order to achieve naturally structured and easy to comprehend texts.

As we have also argued in [TEXS3] section 3.2.1, it is not satisfactory to use one and the same fixed style in a complex text. Rather, the realisation of complex task-content in a single continuous text requires a combination of various sequence realisation styles, alternating between layout and explicit marking to reflect sequences at different levels within the hierarchical structure of described task described in the text. Figure 18 and Figure 19 above show two alternative realisations of a complex text, where different combinations of sequence styles are employed in a suitable and natural way. The text in Figure 18 uses numbered list for the top-level GOALs, and explicit sequence discourse markers for the lower-level GOALs, along with aggregation, while the text in Figure 19 uses explicit sequence markers for the top-level GOALs within the list of steps, and a numbered list for the lower-level steps.

The decisions concerning sequence realisation styles are closely related to the decisions concerning aggregation. In this section, we discuss explicit sequence marking on separate sentences in a procedural text; note however, that every sentence can be a complex one, i.e. a result of aggregation. The explicit sequence marking described in this section results in using a particular sequence realisation style for such a sentence as a whole. The specifications and implementation regarding explicit marking of sequences within aggregated clause complexes using conjunction are discussed in Sections 3.2.3 and 3.2.5, respectively.

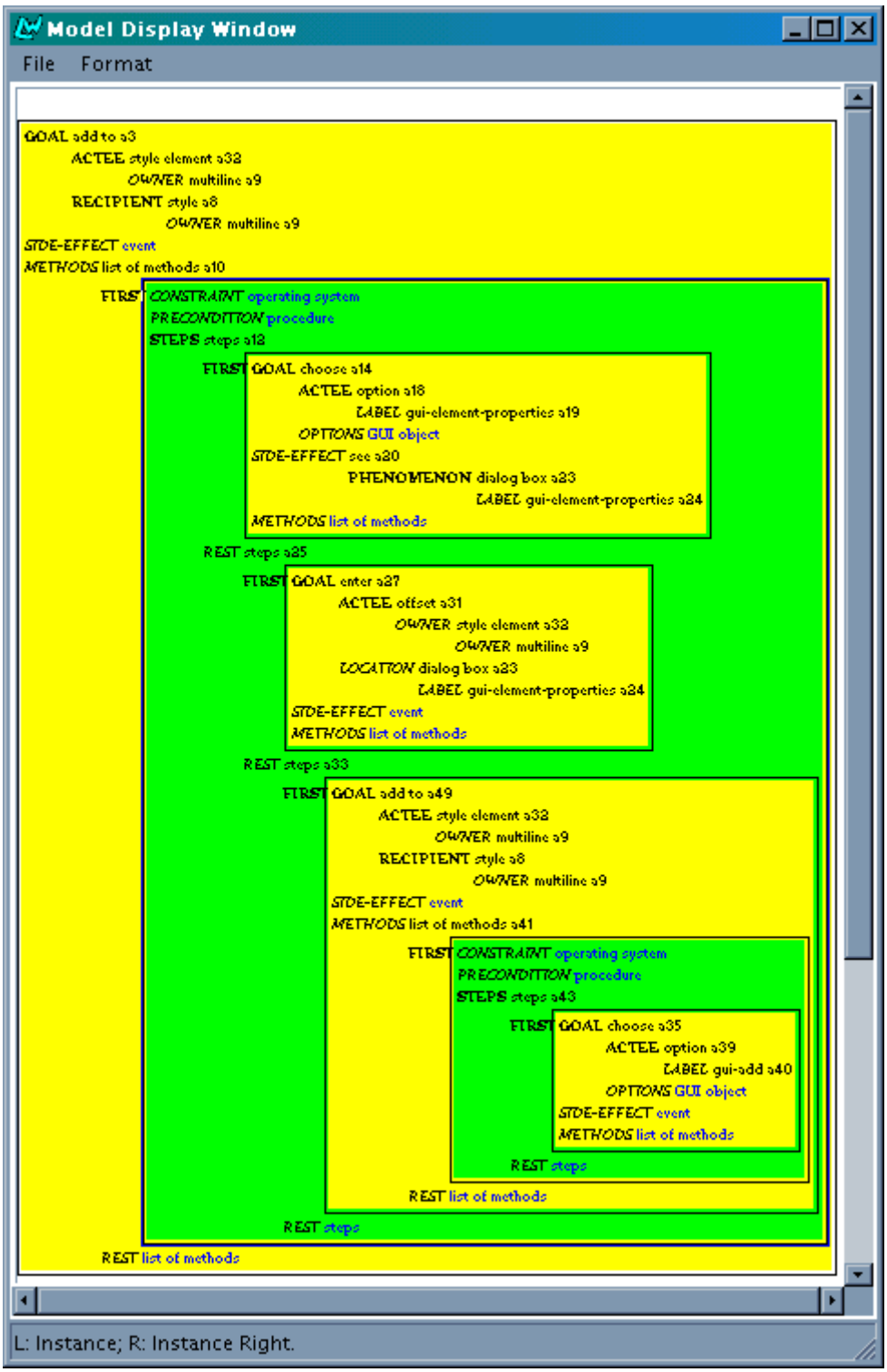


Figure 20: Content model fragment for the text fragment realised in (1), (2) and (3).

3.5.2 The specifications for sequence realisation styles

In [TEXS3] section 3.2.2 we presented an initial formalisation of our approach to planning sequence realisation styles in the full procedural texts generated in AGILE. As the work progressed, we developed better elaborated specifications, which we present in this section.

Obviously, the decisions concerning sequence realisation need to be closely coupled with the decisions concerning aggregation. Therefore, the entrance condition for the planning of

sequence style realisation is that we are generating a sequence of at least two separate, i.e. non-aggregated SUBSTEPS.

The decisions about the sequence marking style are only made when the first element of a sequence is reached. The style for any subsequent elements in the same sequence needs to be the same. For this and other reasons, we need distinguish between the first and subsequent elements of a sequence. The following system allows us to keep track of sequence-continuing elements:

```
System: SEQUENCE-MEMBERSHIP
    Task →
          [sequence-next-member]
          [not-sequence-next-member]

Chooser: SEQUENCE-MEMBERSHIP-CHOOSER
    If the element continues a sequence
    Then choose sequence-next-member
    Else choose not-sequence-next-member
```

Similarly, we need to distinguish between the first and subsequent elements of a conjoined sequence of tasks. The following system allows us to keep track of sequence-continuing elements in a conjunction:

```
System: CONJUNCTION-MEMBERSHIP
    Task →
          [conjoined-next-member]
          [not-conjoined-next-member]

Chooser: CONJUNCTION-MEMBERSHIP-CHOOSER
    If the element continues a conjoined sequence
    Then choose conjoined-next-member
    Else choose not-conjoined-next-member
```

The first system that makes a decision about sequence realisation is the following one, which decides whether to use a running text or a list. The input condition is quite complicated, in order to capture all elements in any sequence, including the last one or the first one of a group of conjoined tasks.

```
System: SEPARATE-INSTRUCTION-TASKS-SEQUENCE:
    More-Separate-Tasks OR
    (no-more-tasks AND sequence-next-member) OR
    (more-conjoined-tasks AND not-sequence-next-member) OR
    (not-conjoined-next-member AND sequence-next-member) →
          [Running-Tasks]
          [List-Tasks]
```

The choice between list and running text is to a great degree determined by the genre of step-by-step instructions in procedural style. The top-level of procedural instruction steps is realised as a list, provided there are at least two steps. At any lower level, simple content is realised as a running text, while complex content is realised as a list. The following chooser reflects this decision.

```
Chooser: SEPARATE-INSTRUCTION-TASKS-SEQUENCE-CHOOSER:
    If at the top-level in A-box,
    Then choose List-Tasks
    Else
          If at any lower level in A-box
          which has no more than four SUBSTEPS
          and the SUBSTEPS are simplex
          and have no Constraints,
          Then choose Running-Tasks
          Else choose List-Tasks
```

Note that the top-level sequence is always realised as a list. Any further embedded level can be a list or a running text (if it contains only simplex SUBSTEPS with no alternative METHODs distinguished by Constraints).

The next decision to make is whether to mark a sequence explicitly or not. This decision can also be made when a separate task which is a sequence member is being inserted. For running text we can choose between explicit marking and no marking. For lists of tasks, we always use explicit marking. The reason for this is to avoid confusion of a step sequence with a list of alternative methods. When alternative methods are realized as separate ones, they are formatted as a bullet list in the output. So, we have two systems making this decision, specified as follows:

```
System: SEPARATE-LIST-INSTRUCTION-TASKS-SEQUENCE-MARKING:
    List-Tasks →
          [Explicit-Marking-List-Tasks]

System: SEPARATE-RUNNING-INSTRUCTION-TASKS-SEQUENCE-MARKING:
    Running-Tasks →
          [Explicit-Marking-Running-Tasks]
          [No-Marking-Running-Tasks]
```

The top-level of instructions always uses explicit markers, for the lower levels we make the following more elaborate decision: we use explicit marking of sequences in lists, no marking of sequences in running texts which have less than three elements (i.e., in sequences of just two elements), and explicit marking of sequences in running texts which have more than two elements. The criteria for a running text are captured as follows.

```
Chooser: RUNNING-INSTRUCTION-TASKS-SEQUENCE-MARKING-CHOOSER:
    If there are less than three tasks,
          Then choose No-Marking-Running-Tasks
          Else choose Explicit-Marking-List-Tasks
```

Explicit sequence marking is only used if there are at least two SUBSTEPS. However, this does not need to be expressed as a separate condition, because the sequence realisation decision process is only triggered for sequences containing at least 2 elements.

Moreover, there is some interplay between the choice of list vs. running text and explicit sequence marking, so we introduce the following systems:

```
System: RUNNING-INSTRUCTION-TASKS-SEQUENCE-MARKING:
    Running-Tasks AND Explicit-Marking-Tasks →
          [Linguistically-Marked-Running]

System: LIST-INSTRUCTION-TASKS-SEQUENCE-MARKING:
    List-Tasks and Explicit-Marking-Tasks →
          [Linguistically-Marked-List]
          [Numbered-list]
```

Although it is to a certain degree a matter of stylistic choice whether to use numbering or linguistic discourse markers for the top level, we decided to make this decision automatically, as to relieve the user of the AGILE system of this decision burden. We decide as follows: if there are up to three elements in the top-level sequence, linguistic markers are used, otherwise we use numbering.

The realisation of the lower-level SUBSTEPS can then take a complementary form, in order to avoid repetition. So, if the top-level uses linguistic markers, the second-level can use numbered list. If the top-level uses numbered list, the second-level can use linguistic markers. And so on for further lower levels.

```
Chooser: LIST-INSTRUCTION-TASKS-SEQUENCE-CHOOSER:
    If at the top-level in A-box,
    Then
          If there are at most three top-level PROCEDUREs,
          Then choose Linguistically-Marked-List
          Else choose Numbered-List
    Else
          If the top-level used Linguistic-Markers-Tasks,
          Then choose Numbered-List
          Else choose Linguistically-Marked-List
```

Last but not least, we have to plan the exact realisation of the sequence marker. This happens in alike fashion for numbered lists and linguistically marked lists. Below, we discuss each of them in turn.

For numbered lists, the numbering itself is not an issue for the text planner, because numbering of list elements is done through HTML mark-up which is then interpreted by a suitable viewer, e.g. Microsoft Explorer in the Final Prototype of the AGILE system. The HTML mark-up is inserted as an annotation into the SPLs by the sentence planner (cf. [TEXM2] and [TEXM1]). However, we do perform some planning of the numbering, in order to aid the sentence planner in inserting the appropriate HTML mark-up in accordance to how many-th element of a list is being marked.

```
Gate: NUMBER-MARKER-INSERT
    Numbered-List →
    Number-Inserted
       (INSERT NUMBER-MARKER)
       (CONFLATE NUMBER-MARKER TASK)
```

The following systems insert the appropriate realisation constraints depending on how many-th element of a list is being marked:

```
Gate: NUMBER-FIRST
    Number-Inserted AND Not-Sequence-Next-Member →
          Number-First
            (INSERT Number-FIRST)
            (CONFLATE Number-First Number-Marker)

Gate: NUMBER-LAST
   Number-Inserted AND Sequence-Next-Member AND
   no-more-tasks AND Not-Conjoined-Next-Member →
          Number-Last
            (INSERT Number-Last)
            (CONFLATE Number-Last Number-Marker)

Gate: NUMBER-RELATIVE
   Number-Inserted AND Sequence-Next-Member
   AND more-tasks →
          Sequence-Marker-relative
            (INSERT Number-relative)
            (CONFLATE Number-relative Number-Marker)
```

The decisions made for sequences marked by explicit linguistic discourse markers, proceed likewise. First, an explicit sequence discourse marker is inserted into the text plan.

```
Gate: SEQUENCE-MARKER-INSERT
    Linguistically-Marked-List OR Linguistically-Marked-Running →
    Sequence-Marker-Inserted
       (INSERT SEQUENCE-MARKER)
       (CONFLATE SEQUENCE-MARKER TASK)
```

Unlike the list numbering, the linguistic markers in running text as well as in an explicitly linguistically marked list are realised by the lexico-grammar. In particular, they are realised as textual conjunctive expressions. The SPL produced by the TSM thus needs to contain sufficient information to enable the grammar to realise the textual conjunctive for a given element in a sequence appropriately. The choice depends not only one the position in the sequence, but also on the length of the sequence. Figure 21 shows the different linguistic markers used in each language depending on how many elements there are in a sequence.

- **English:**
  2 elements: *First, Now*
  3 and more elements: *First*, (*Then*)[+2], *Finally*

[2] (Then)+ means one or more occurrences of 'then'.

- **Bulgarian:**
  2 elements: *Отначало, Сега*
  3 and more elements: *Отначало, След това, Накрая*
- **Czech:**
  2 elements: *Nejprve, Nyní*
  3 and more elements: *Nejprve,* (*Potom*)$^{+}$, *Nakonec*
- **Russian:**
  2 elements: *Сначала, Затем* (*Теперь* is also possible, but the context of its usage is restricted to the here-and-now interaction style)
  3 and more elements: *Сначала , Затем, И наконец*

Figure 21: Different linguistic markers depending on the number of elements in a sequence

The following systems insert the appropriate realisation constraints:

```
Gate: SEQUENCE-MARKER-FIRST
    Sequence-Marker-Inserted AND Not-Sequence-Next-Member →
          Sequence-Marker-First
          (REALIZE-WITH SEQUENCE-MARKER :conjunctive sequence-first)

System: SEQUENCE-MARKER-LAST
   Sequence-Marker-Inserted AND Sequence-Next-Member
   AND no-more-tasks AND Not-Conjoined-Next-Member →
          Sequence-Marker-Last
          (REALIZE-WITH SEQUENCE-MARKER :conjunctive sequence-last)
          Sequence-Marker-Other
          (REALIZE-WITH SEQUENCE-MARKER :conjunctive sequence-other)

Chooser: SEQUENCE-MARKER-LAST-CHOOSER
    If the element ends a sequence of two elements
    Then choose Sequence-Marker-Other
    Else choose Sequence-Marker-Last

Gate: SEQUENCE-MARKER-RELATIVE
   Sequence-Marker-Inserted AND Sequence-Next-Member
   AND more-tasks →
          Sequence-Marker-relative
          (REALIZE-WITH SEQUENCE-MARKER :conjunctive sequence-relative)
```

This completes the presentation of the revised specifications for sequence realisation. In the next section, we present the implementation details.

3.5.3 Implementation

In this section, we provide the details pertaining to the implementation corresponding to the specifications given in the preceding section. In the implementation phase we of course had to code the specified systems, and choosers and the underlying inquiries and inquiry code. In addition, there were the following important issues that had to be tackled:

- recognizing the first, intermediate and last task in a sequence, in order to assign the appropriate realization constraints to the sequence markers
- ensuring a consistent sequence realization style throughout a sequence, i.e., making a decision when the first element is reached, and then percolating the decision to the subsequent elements
- discrimination between the sequences at the highest level in an A-box and the lower-level sequences, in order to make specific decisions concerning the top-level

We now present the systems and chooser as we have implemented them, and point out how they contribute to solving the above issues.

The system SEQUENCE-MEMBERSHIP was introduced in order to keep track of elements which continue an ongoing sequence. Together with other features, the two features distinguished in this system also enable us to discriminate between the intermediate and the last elements in a sequence. The choice in this system is made on the basis of either a default specified in the inquiry (**not-sequence-next-member**), or a preselection imposed by a preceding element in the sequence (**sequence-next-member**).

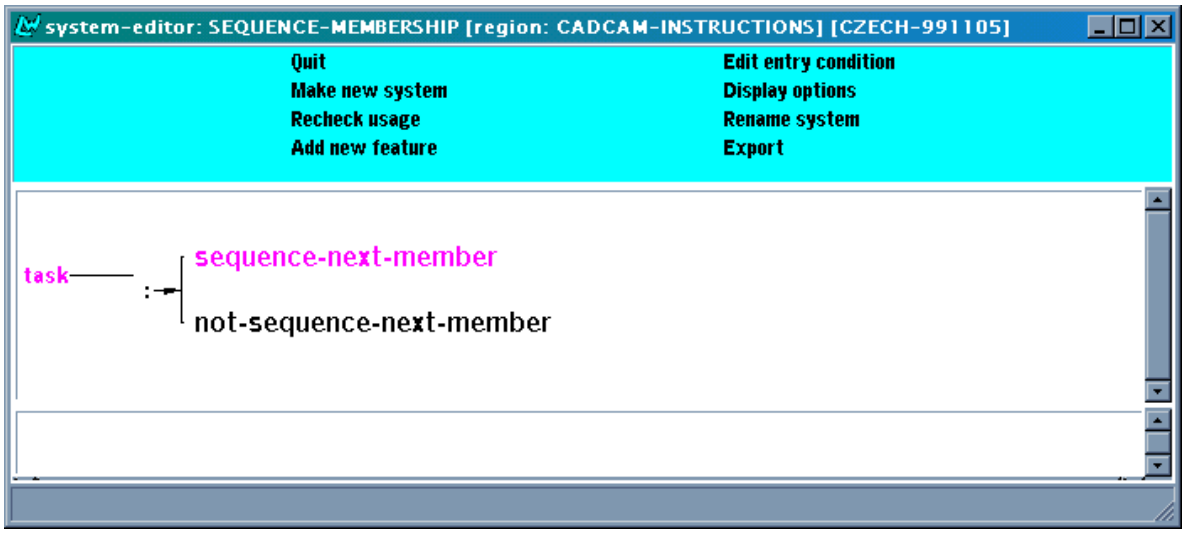


Figure 22: Sequence-Membership system

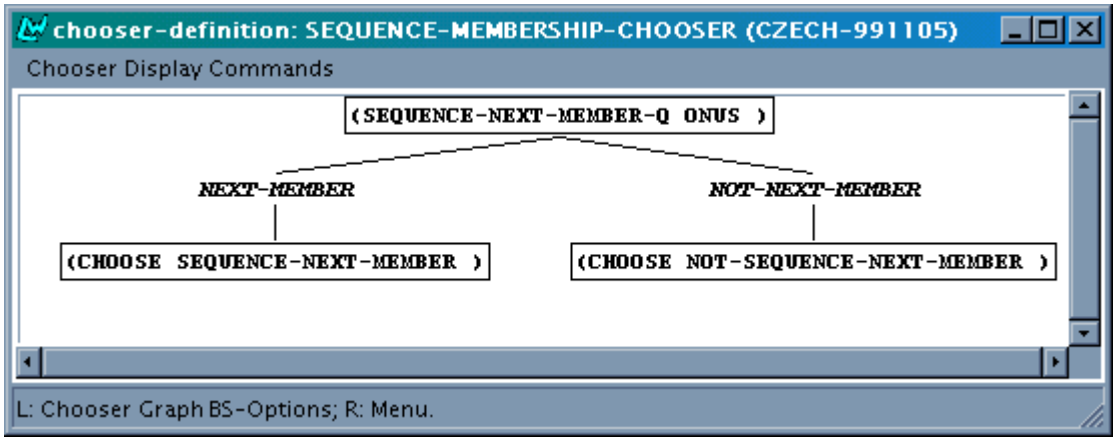


Figure 23: Sequence-membership chooser

We also need to recognize the case of an interleaved sequence and conjunction (i.e., when an element in a sequence consists of conjoined tasks). The system CONJUNCTION-MEMBERSHIP was introduced for this purpose, and in order to keep track of elements which continue an ongoing conjunction of tasks. Together with other features, the two features distinguished in this system also enable us to discriminate between the intermediate and the last elements in a conjoined sequence of tasks. The choice in this system is made on the basis of either a default specified in the inquiry (**not-conjoined-next-member**), or a preselection imposed by a preceding element in the sequence (**conjoined-next-member**).

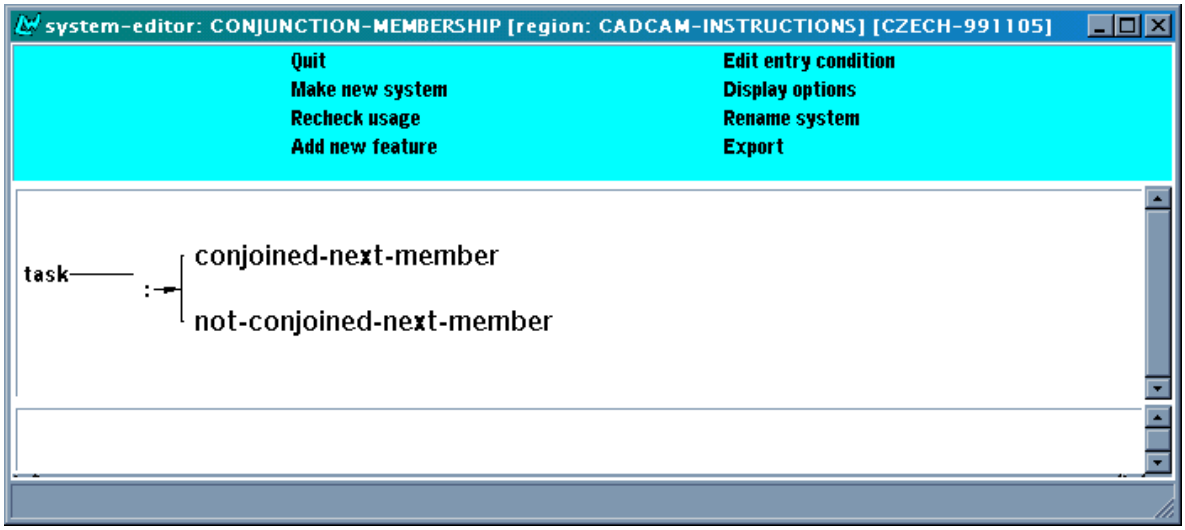


Figure 24: Sequence-Membership system

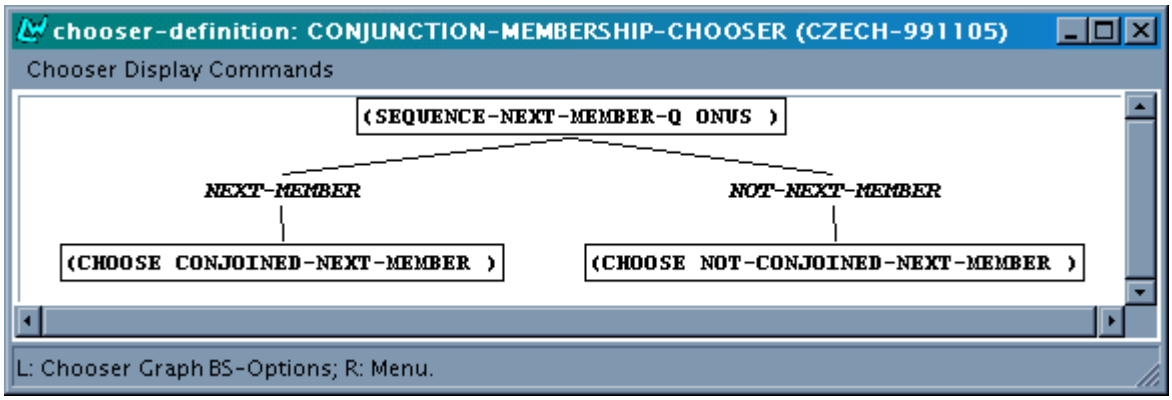


Figure 25: Conjunction-membership chooser

The following group of systems deals with the choices of particular sequence realization styles under specific circumstances, as described and discussed in the specifications. These systems also take care of maintaining a consistent sequence realization style throughout a sequence. This is achieved as follows. For the first element in a sequence, a decision about the sequence realization style is made, using the corresponding chooser, inquiry and inquiry code. Once the choice is made, a preselection is imposed on the next element in the same sequence. Then, for every next element in that sequence (recognized by the feature **next-sequence-element** introduced in the SEQUENCE-MEMBERSHIP system) the sequence realisation style choice is determined on the basis of the preselection and not any more by the inquiry code.

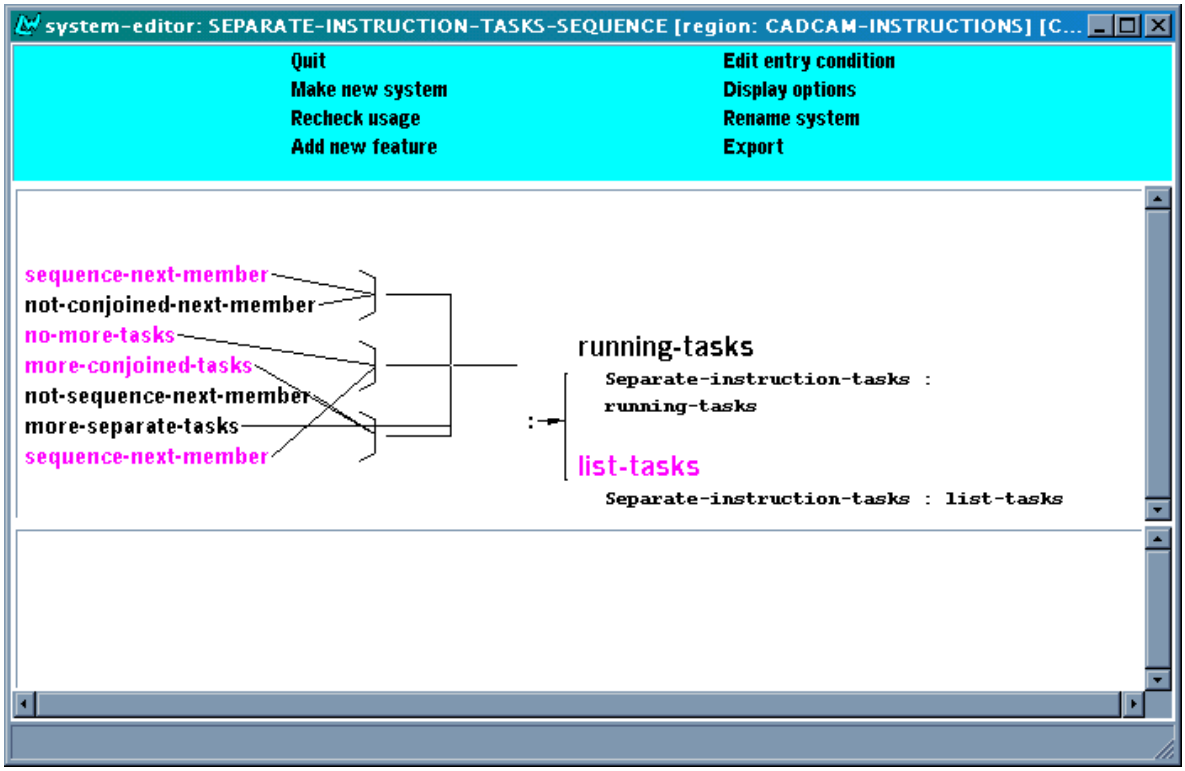


Figure 26: Separate-instruction-task-sequence system

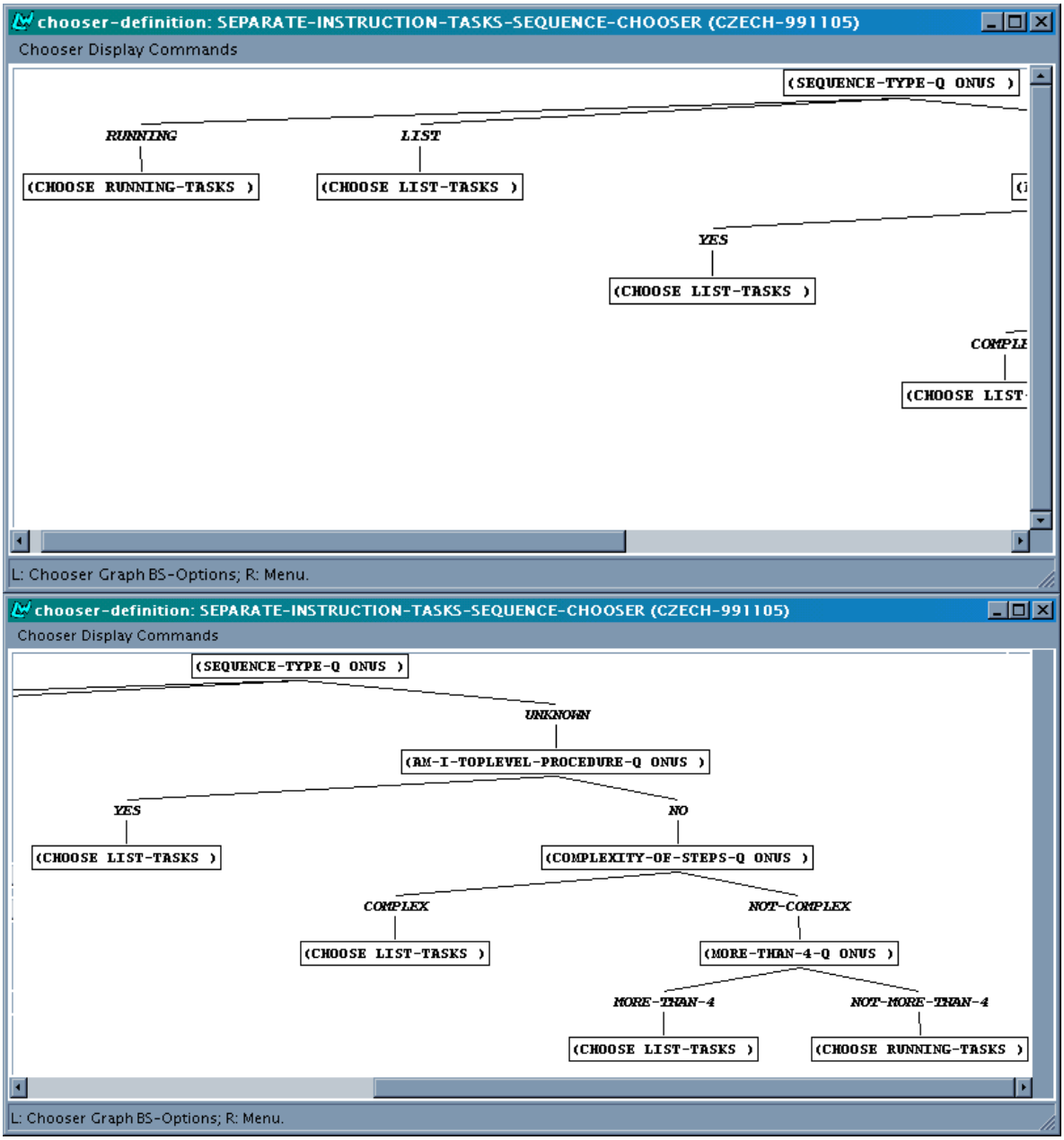


Figure 27: Separate-instruction-tasks-sequence chooser (part 1 and 2)

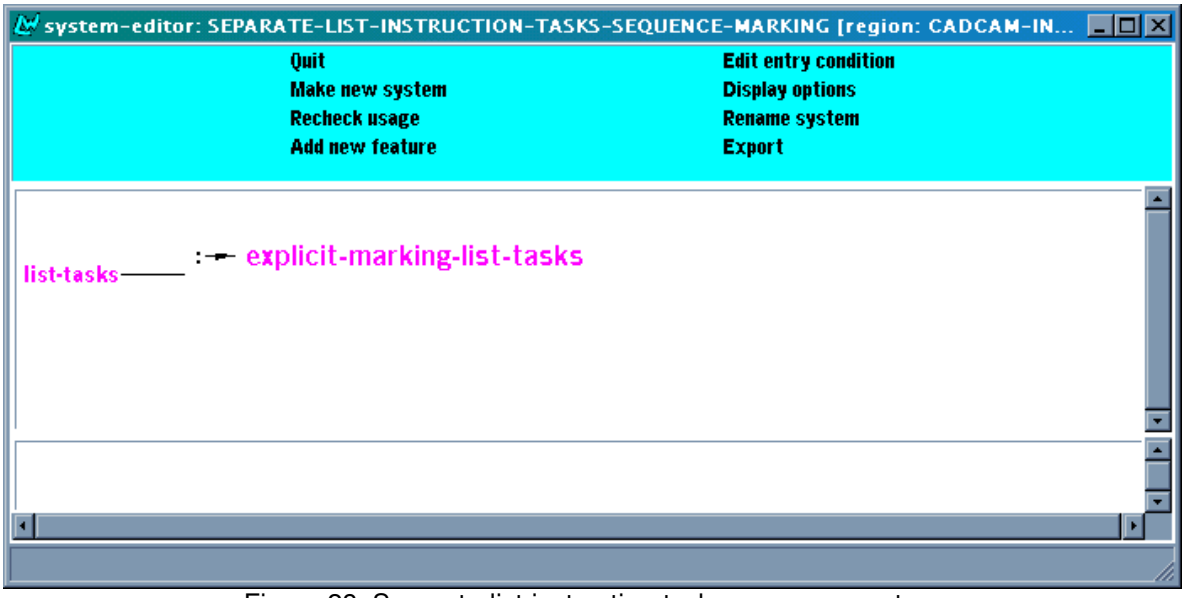


Figure 28: Separate-list-instruction-tasks-sequence system

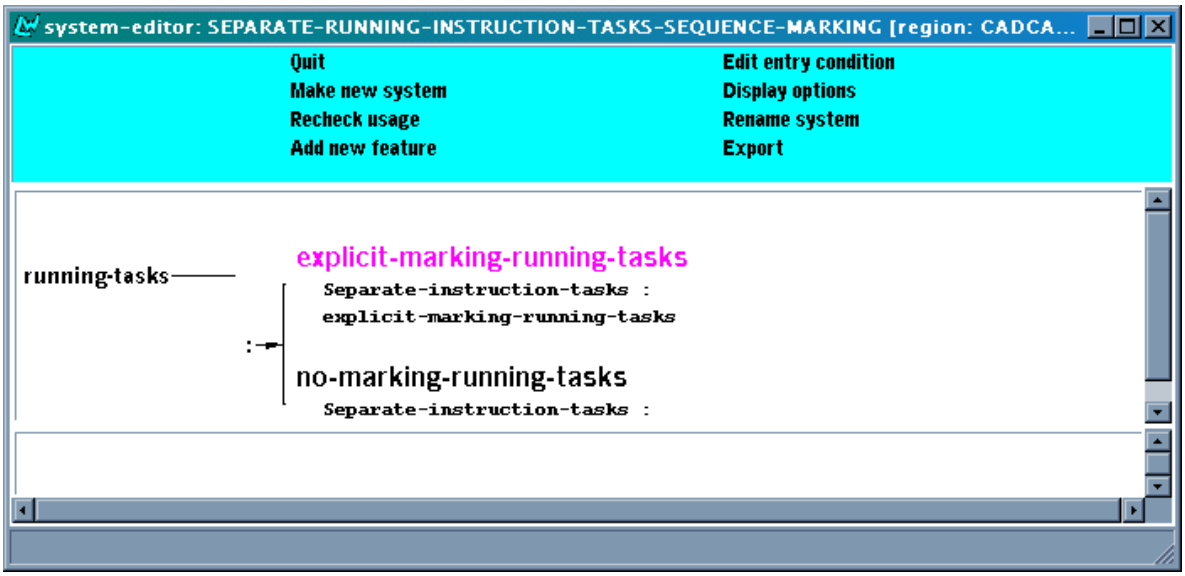


Figure 29: Separate-running-instruction-tasks-sequence-marking

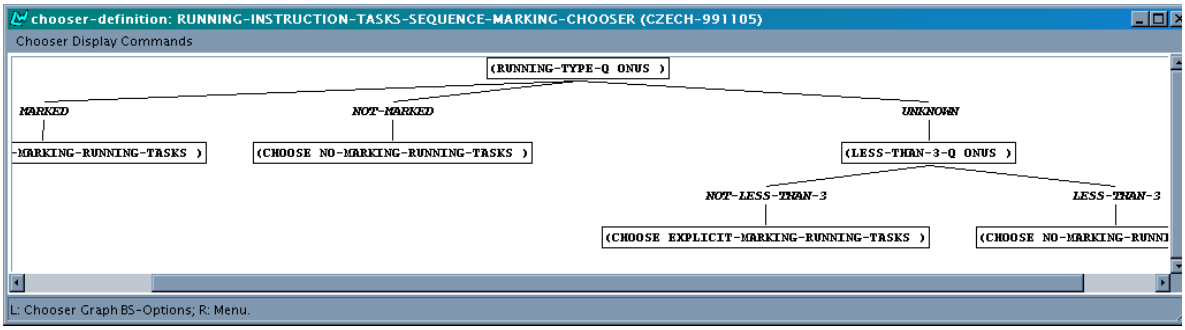


Figure 30: Running-instruction-tasks-sequence-marking chooser

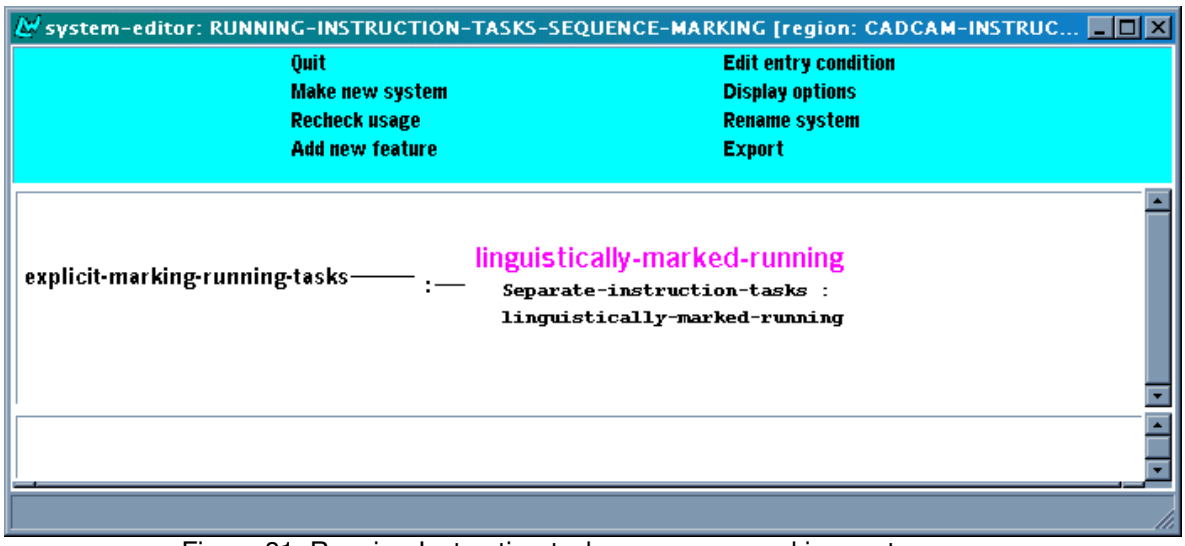


Figure 31: Running-Instruction-tasks-sequence-marking system

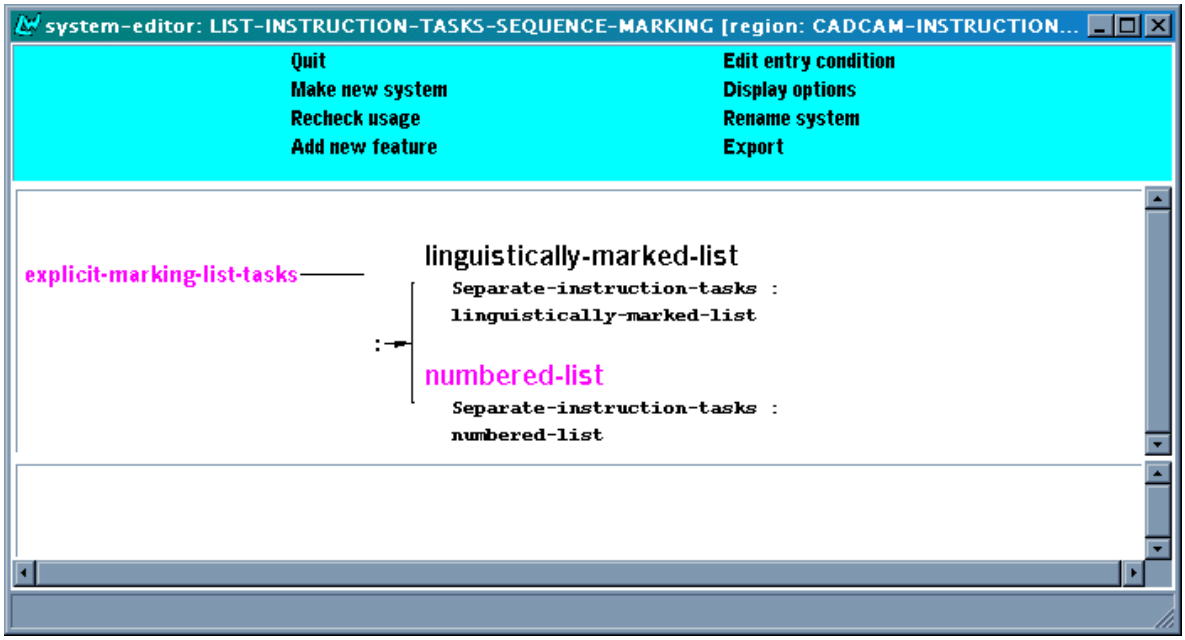


Figure 32: List-Instruction-Tasks-Sequence-Marking system

The following group of systems serves the purpose of inserting different kinds of sequence markers, namely discourse markers and numbers, and imposing the appropriate realization constraints on them when necessary.

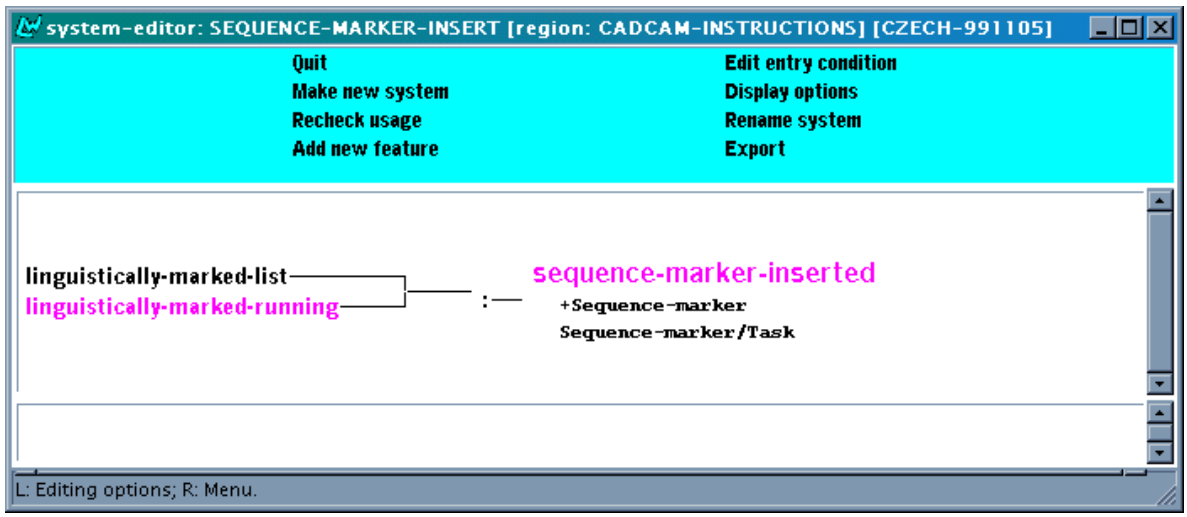


Figure 33: Sequence-marker-insert system

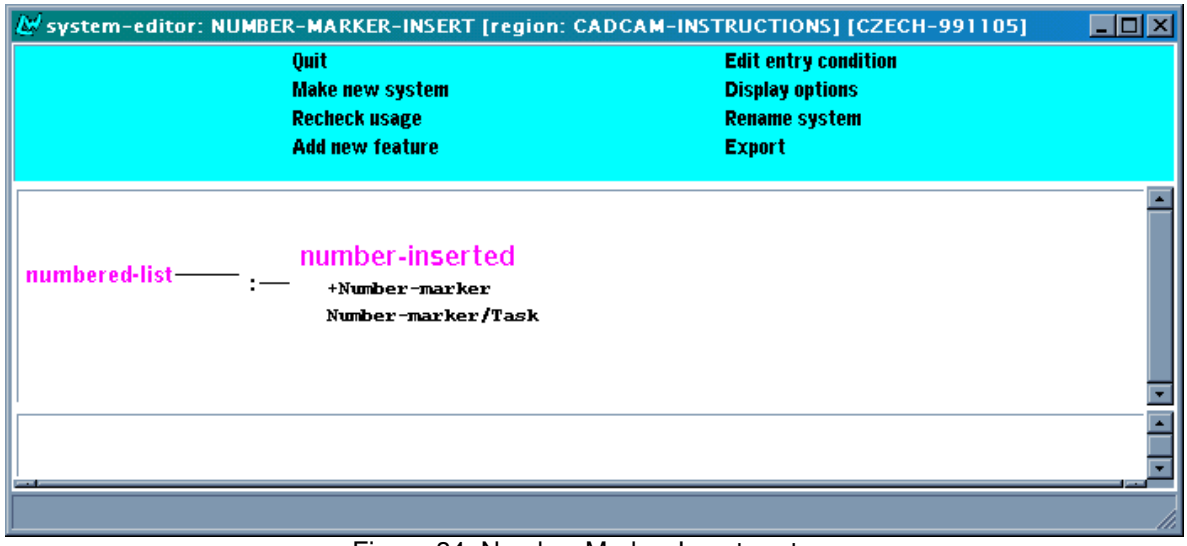


Figure 34: Number-Marker-Insert system

As we discussed in the specifications, we use different discourse markers for different positions in an explicitly linguistically marked sequence. The systems bellow ensure this. The input conditions for each of the systems are different in accordance to how many-th sequence element the system handles. Then, the appropriate constraint is imposed which tells the lexicogrammar what conjunctive element to use in the generated sentence.

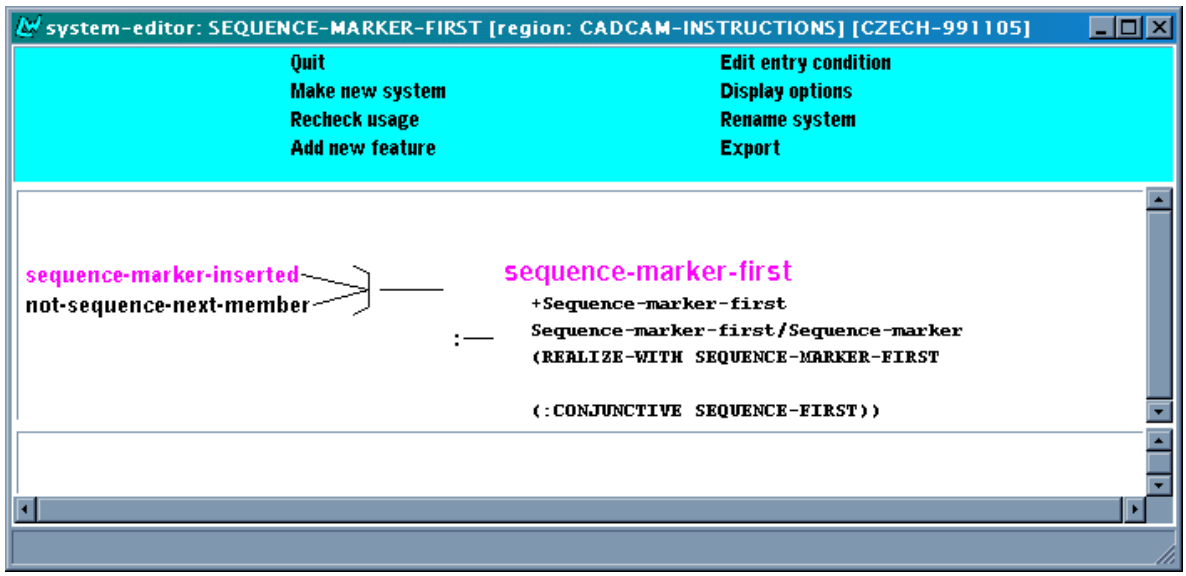


Figure 35: Sequence-Marker-First system

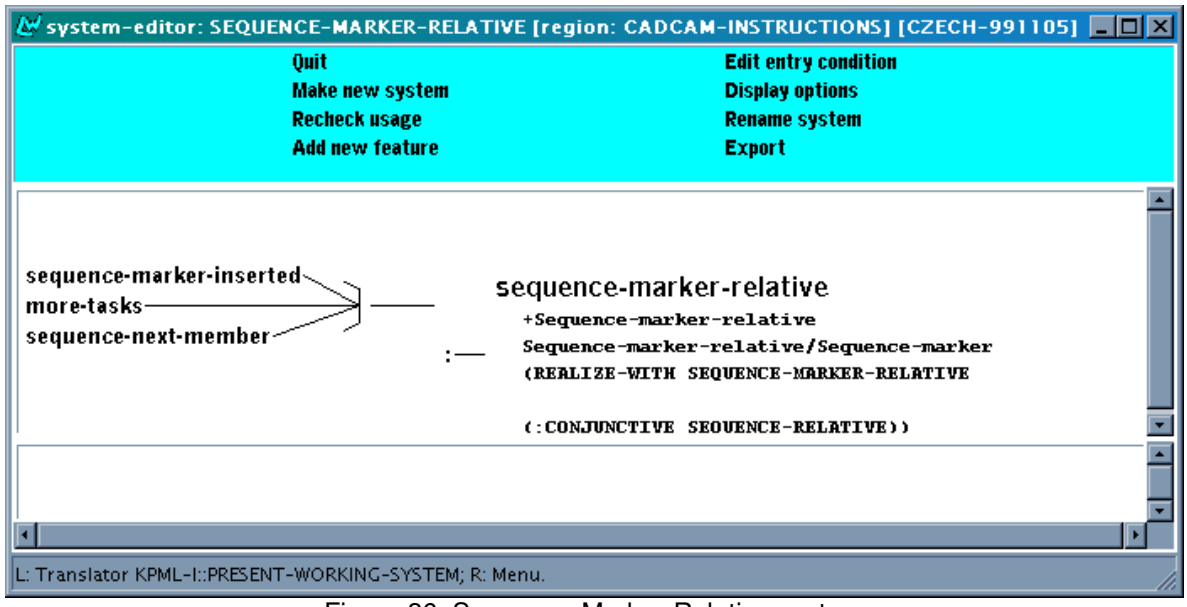


Figure 36: Sequence-Marker-Relative system

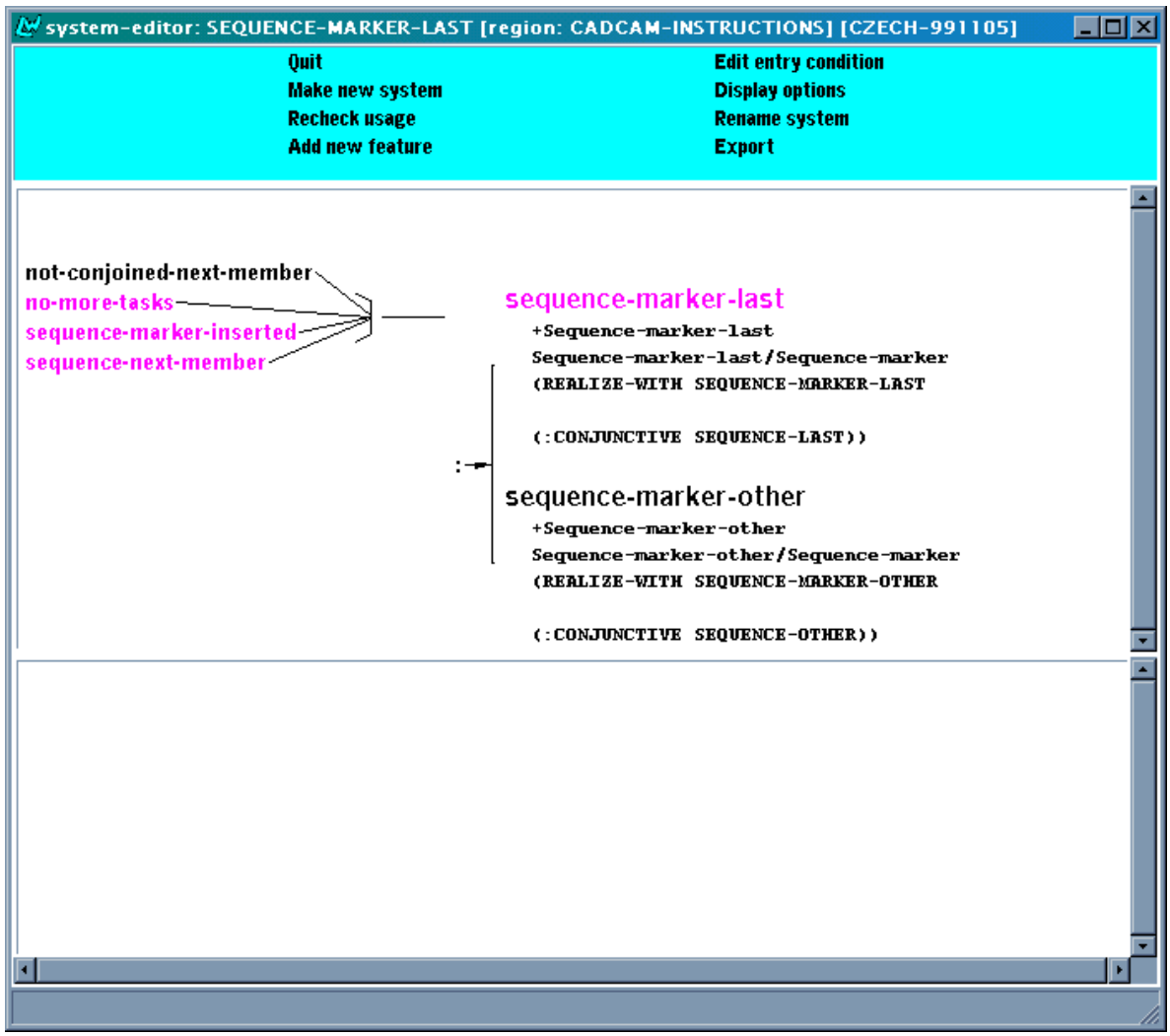


Figure 37: Sequence-Marker-Last system

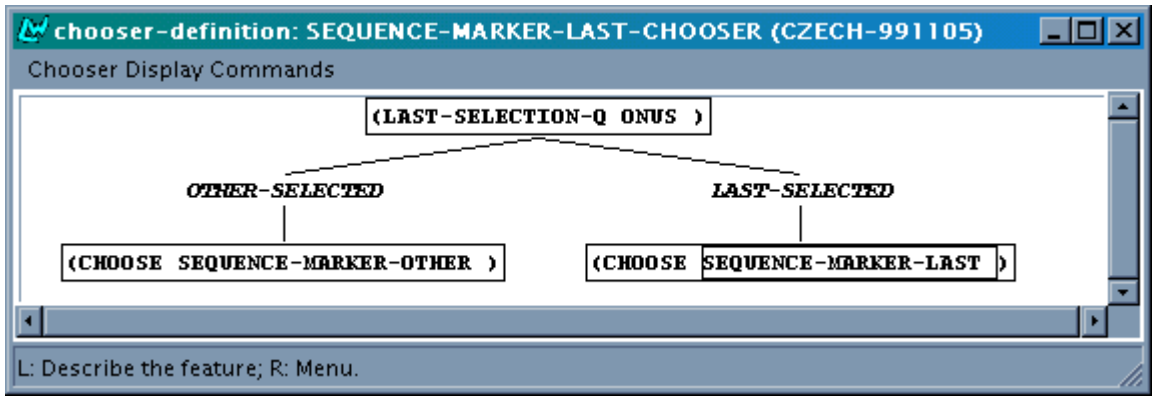


Figure 38: Sequence-Marker-Last chooser

Notably, we use a different realization of the last element in a sequence depending on whether the sequence consists of just two or of more elements. This decision is made in the above system and chooser. The choice made thereby is determined by a default or a preselection made by the preceding sequence element. The important decision is in fact made as soon as the first element in a sequence is being handled, in the SEQUENCE-MARKER-LAST-SELECTION system shown below. At that point, we count how many elements the sequence has. If there are only two, the preselection for **sequence-marker-other** is made.

Otherwise, i.e. without the preselection, the default (**sequence-marker-relative**) is chosen in the SEQUENCE-MARKER-LAST system above.

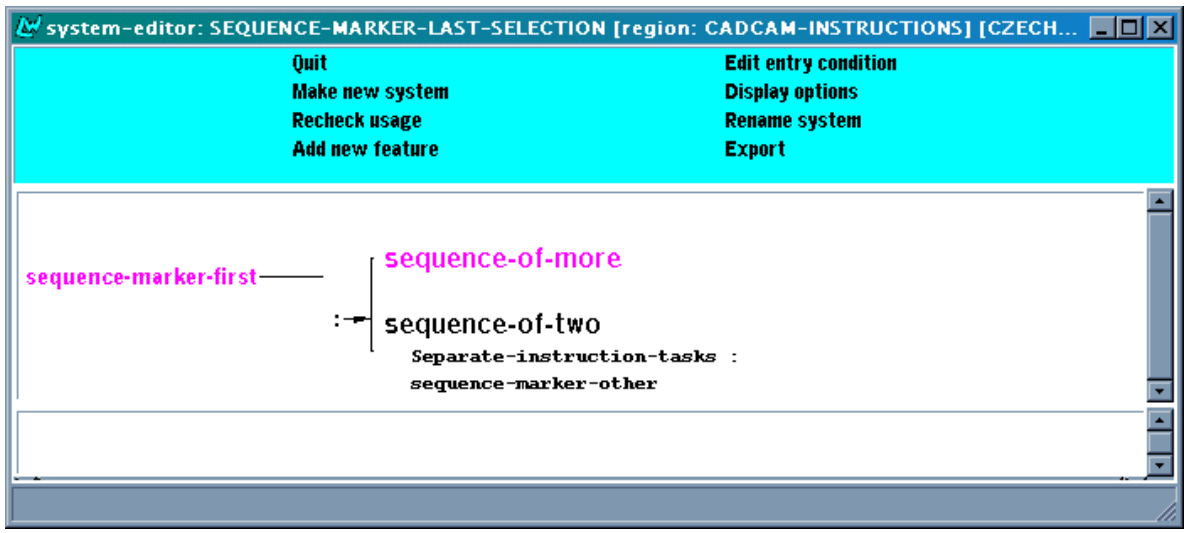


Figure 39: Sequence-Marker-Last-Selection system

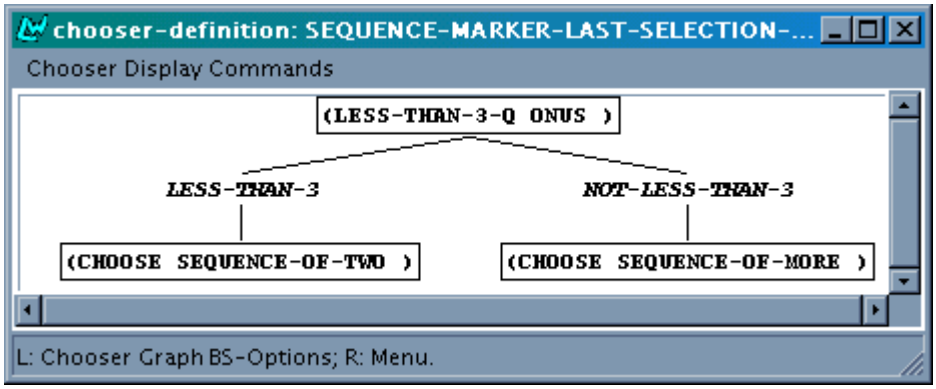


Figure 40: Sequence-Marker-Last-Selection chooser

The sequence marker realisations are abbreviations for the following full SPL code:

```
Conjunctive-first =
         (:conjunctive-relation-q conjunctive
          :conjunctive-relation-id (?rr / rhetorical-relation)
          :process-regulated-q processregulated
          :necessity-q nonecessity
          :sequence-q sequence
          :absolute-position-q absolute
          :extremal-position-q extremal-first)
Conjunctive-last =
        (:conjunctive-relation-q conjunctive
         :conjunctive-relation-id (?rr / rhetorical-relation)
         :process-regulated-q processregulated
         :necessity-q nonecessity
         :sequence-q sequence
         :absolute-position-q absolute
         :extremal-position-q extremal-last)
Conjunctive-other =
         (:conjunctive sequence-other
          :conjunctive-relation-q conjunctive
          :conjunctive-relation-id (?rr / rhetorical-relation)
```

```
            :process-regulated-q processregulated
            :necessity-q nonecessity
            :sequence-q sequence
            :absolute-position-q absolute
            :extremal-position-q nonextremal)
Conjunctive-relative =
          (:conjunctive-relation-q conjunctive
           :conjunctive-relation-id (?rr / rhetorical-relation)
           :process-regulated-q processregulated
           :necessity-q nonecessity
           :sequence-q sequence
           :absolute-position-q notabsolute
           :relative-position-q immediate)
```

Next, we present the group of systems which ensure appropriate marking of numbered-lists. The systems are similar to the above systems dealing with linguistic marking. However, the numbering itself is not realized by the lexicogrammar, but it is handled by HTML mark-up inserted in the sentence planner. Therefore we do not impose any realization constraints on the number markers.

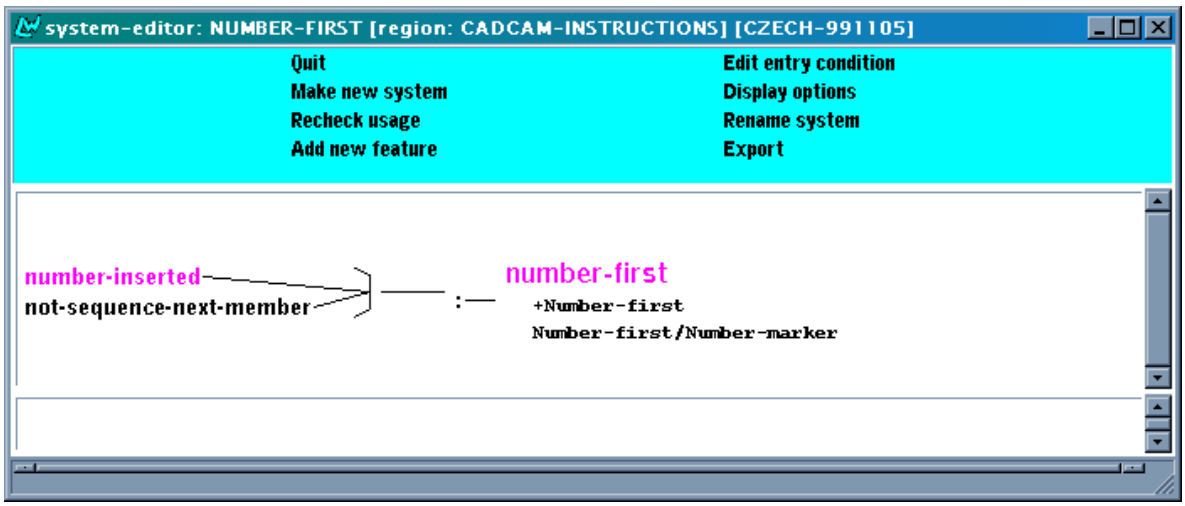


Figure 41: Number-first system

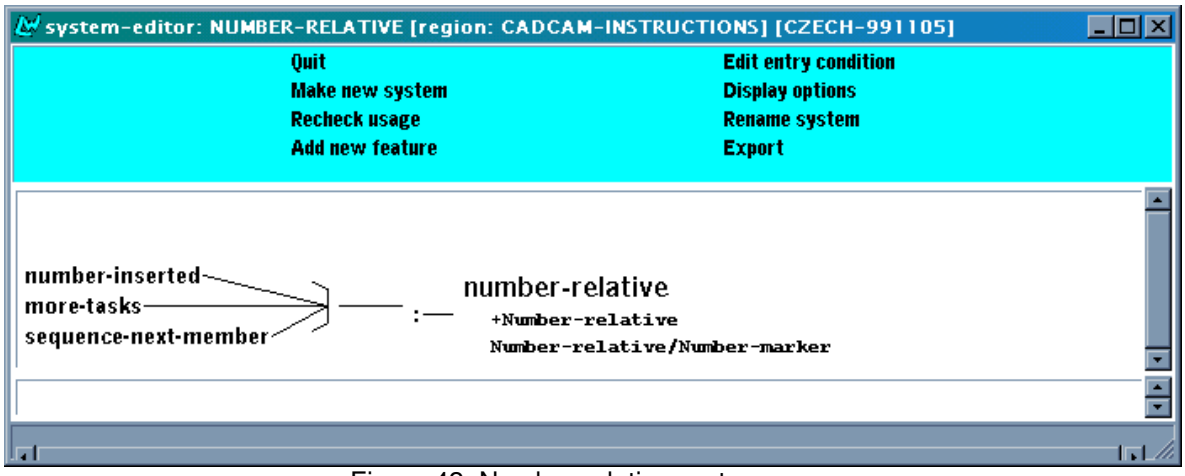


Figure 42: Number-relative system

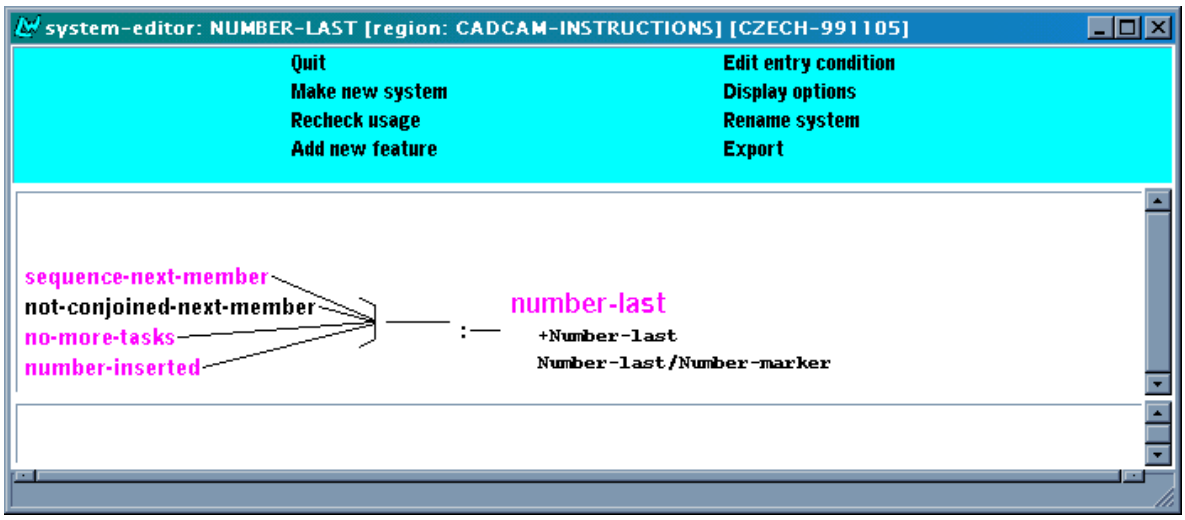


Figure 43: Number-last system

Last but not least, there is the following group of systems which are introduced purely in order to facilitate the insertion of the HTML mark-up in the sentence planner (SPLizer). These systems enable the recognition of a running text sequence and its proper formatting.

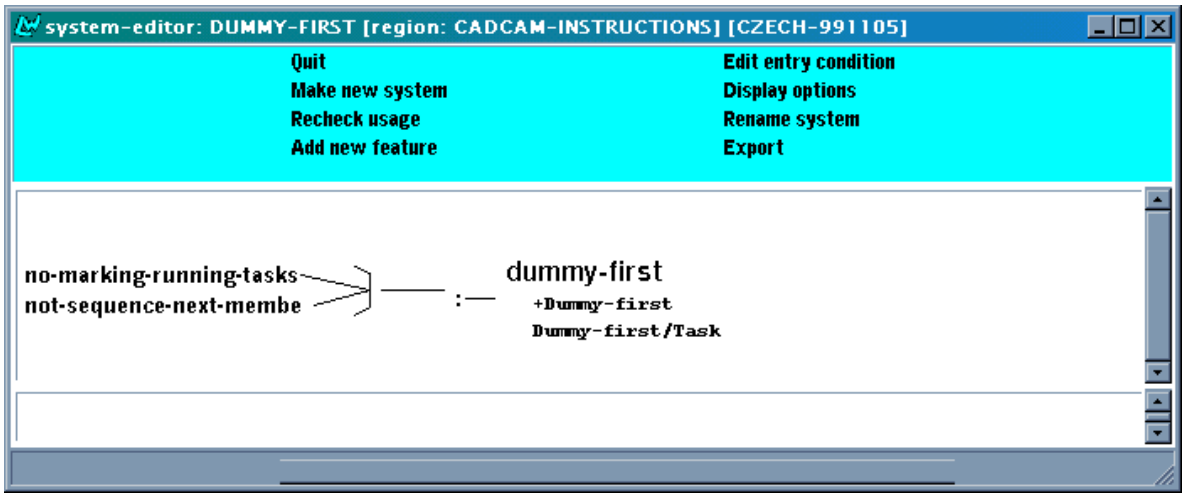


Figure 44: Dummy-first system

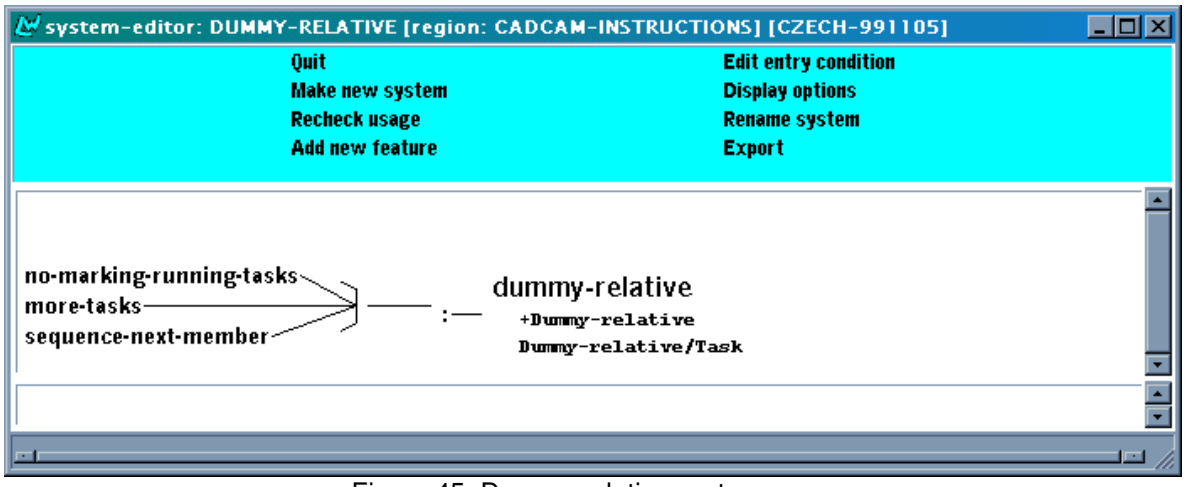


Figure 45: Dummy-relative system

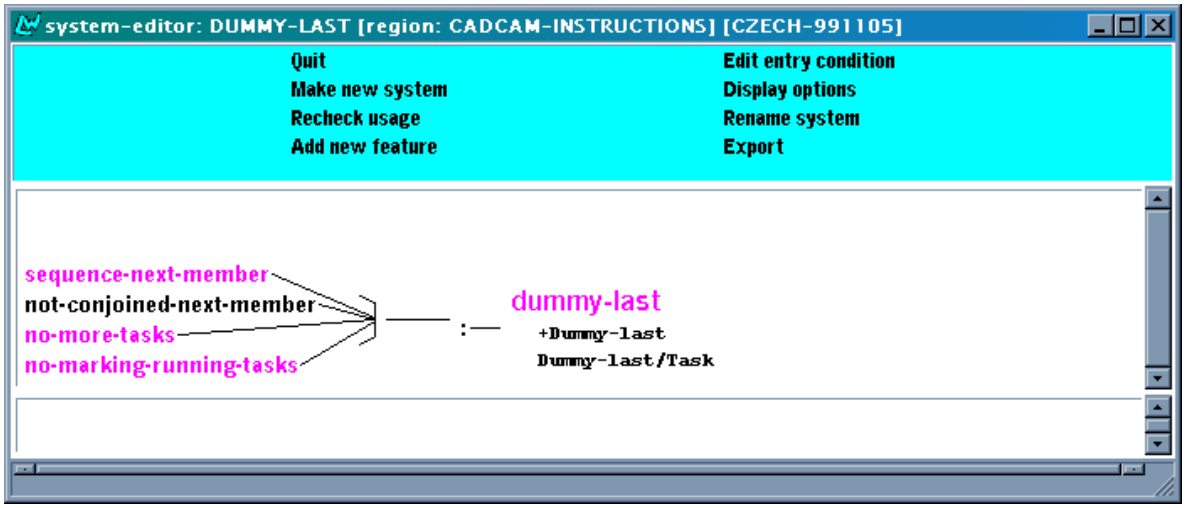


Figure 46: Dummy-last system

For the sake of proper output formatting in the case that a sequence of simple tasks is conjoined, we introduce a marker with the last of the conjoined tasks. This is just to ensure that the proper HTML mark-up can be inserted also in this case.

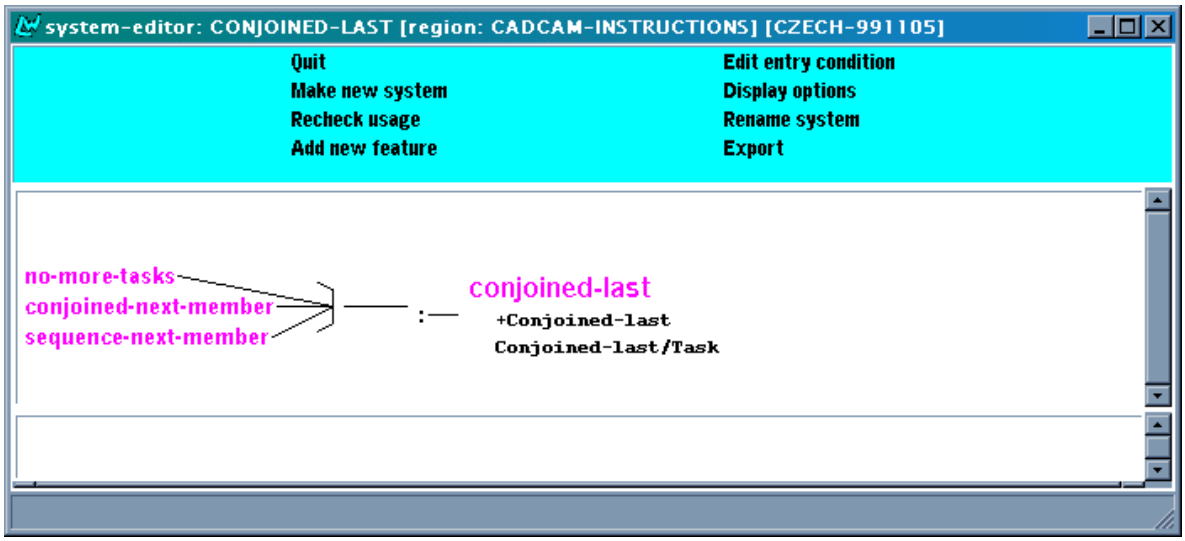


Figure 47: Conjoined-last system

3.6 Additional style

3.6.1 Personal style, indicative

In the [TEXS2] we also described an additional text *style*. Next to the two styles already implemented in the IMD TSM, being *personal style* (imperative) and *impersonal style* (indicative mood, medio-passive voice), we discussed the possibilities of indicative mood in personal style. For example, a personal style of procedural instructions in Czech very often employs indicative mood in active voice in first person plural:

(4) **Cz:**

Abychom	čáru	ukončili,	stiskneme	Enter.
so-that-would-1pl	line	end-1pl	press-ind-1pl	Enter

In order to end the line we press Enter.

Also second person plural is possible, and is used in some manuals:

(5) **Cz:**

Abyste	*čáru*	*ukončili, stisknete*		*Enter.*
so-that-would-2pl	line	end-2pl	press-ind-2pl	Enter

In order to end the line you press Enter.

Formally, there is little to specify here: the way to obtain the above style is to state that the speech act to be used should be "assertion".

3.6.2 Implementation

To implement the above addition, we have extended the TASK-REALISATION-STYLE system:

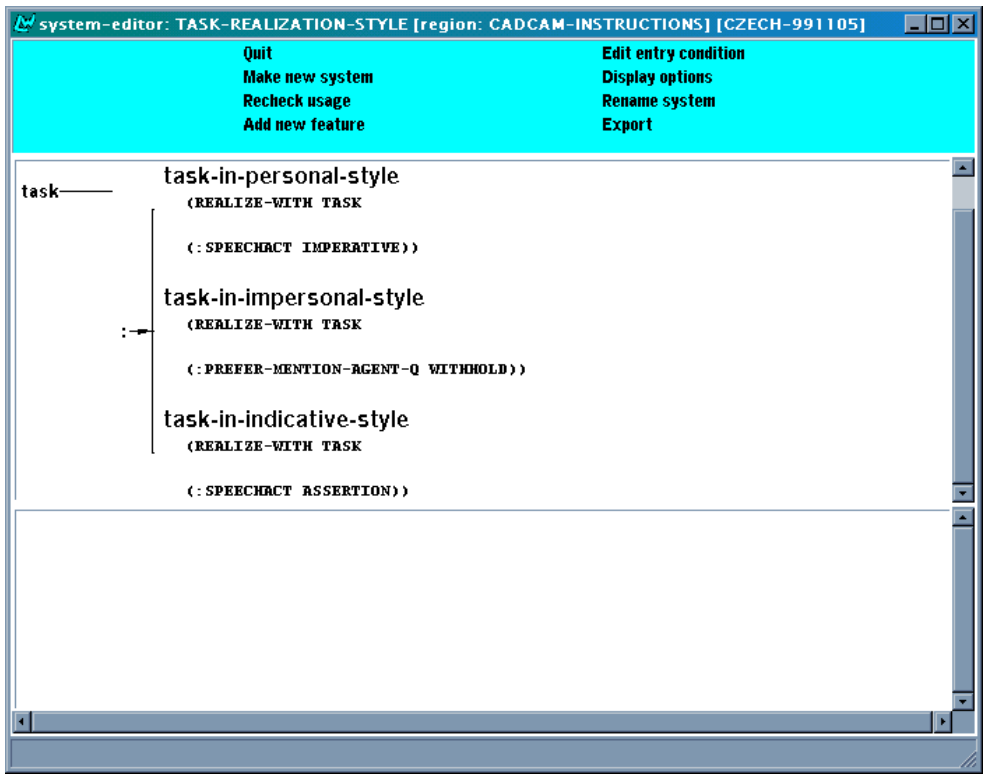


Figure 48 - TASK-REALIZATION-STYLE system

The chooser underlying the system simply checks the style(s) selected in the interface, as we already described in [TEXM2].

4. The CADCAM-SUMMARIES region

In this chapter we describe the CADCAM-SUMMARIES region. This region is responsible for generating text plans for describing one of possible abbreviated styles (look TEXS3) – basic steps. We describe here systems, choosers and inquiries required for creation of a tree of elements constituting a text plan for summaries. The realization of text plan elements by the sentence planner in the case of summaries doesn't differ from the full instructions realization, so nothing to add to the sentence planner.

4.1 A text plan tree for summaries

We are not absolutely reasonable to apply the procedure of abbreviation to our A-box reflecting the activity of technical writer to create the instruction text, because he/she also insert in the representation some degree of content balance and delicacy. But the procedure would be quite reasonable if applied to a full specification of text content, for example, got from the formal specifications of graphical redactor interface. Nevertheless as we can see in the example below the abbreviated text got from procedural text of instruction is quite readable. In the example below we show full instruction text where the clauses that will be "abbreviated" are marked by red and then we show the result of abbreviation as a summary text.

To draw a line and arc combination polyline

Start the PLINE command.

Windows:

From the Polyline flyout on the Draw toolbar, choose Polyline.

DOS and UNIX:

From the Draw menu, choose Polyline.

Draw a line segment.

Specify the start point of the line segment.

Specify the endpoint of the line segment.

Draw an arc.

Switch to Arc mode.

Enter a.

The Arc mode confirmation dialog box appears.

Select OK in the Arc mode confirmation dialog box.

The Arc mode confirmation dialog box disappears.

Specify the endpoint of the arc.

Draw a line segment.

Switch to Line mode.

Enter l.

The Line mode confirmation dialog box appears.

Select OK in the Line mode confirmation dialog box.

The Line mode confirmation dialog box disappears.

Enter the distance.

Enter the angle.

Press Return to end the polyline.

To draw a line and arc combination polyline

Windows:

From the Polyline flyout on the Draw toolbar, choose Polyline.

DOS and UNIX:

From the Draw menu, choose Polyline.

Specify the start point of the line segment.

Specify the endpoint of the line segment.

Switch to Arc mode.

Enter a.

Select OK in the Arc mode confirmation dialog box.

Specify the endpoint of the arc.

Enter l.

Select OK in the Line mode confirmation dialog box.

Enter the distance.

Enter the angle.

Press Return.

4.2 Specifications for SUMMARIES

An example of a text plan for functional descriptions is given in Figure 49

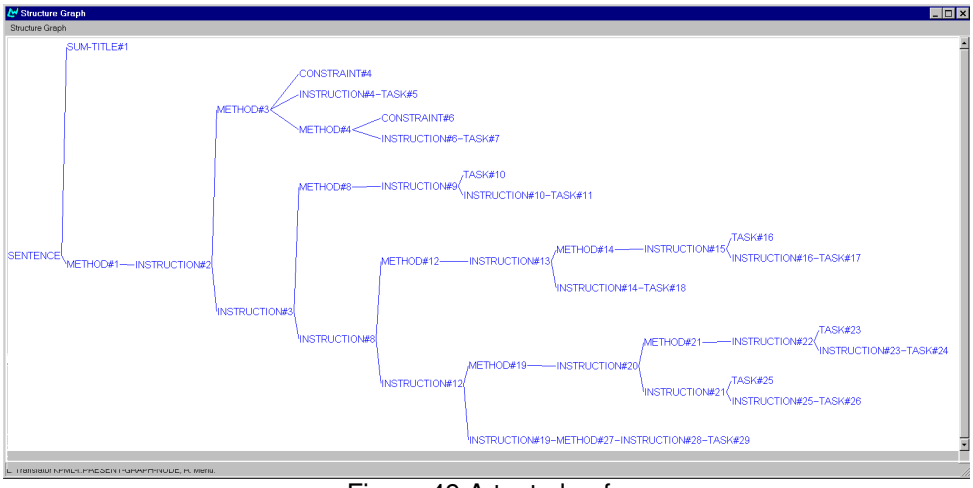


Figure 49 A text plan for summary.

The traversal takes into account five types of constituents:

- **Method** – corresponds to the concept *method* in a-box. If there are several methods of achieving one goal, they are involved one by one recursively. So, one method-node can have as a leaf another method-node that is the other method to one goal as the first one.

  METHOD ←→ METHOD of PROCEDURE

- **Instruction** – corresponds to the concept *procedure* in a-box. The content of the procedure is abolutely indifferent. That means that we do not separate procedures inside precondition from procedures inside substeps. All the procedures a considering in order of their meeting in a-box. The solution with multiple instructions is solved the same way as methods are solved.

  INSTRUCTION ←→ PROCEDURE as one of the SUBSTEPS of METHOD or its PRECONDITION

- **Sum-title** – title of the text

  SUM-TITLE ←→ GOAL of topmost PROCEDURE

- **Constraint** – corresponds to the concept *constraint* in a-box. Is inserted into textplan everywhere it is meeting as a slot under method concept.

  CONSTRAINT ←→ CONSTRAINT of METHOD

- **Task** – corresponds to the concept *goal* in a-box. It corresponds to the part of the a-box appropriate to the sentence of final text.

  TASK ←→ GOAL of PROCEDURE.

4.3 Implementation of SUMMARIES

The CADCAM-SUMMARIES region allows a traversal of an A-box to gather from it all information relevant for creating a summary instruction. The system which achieves selection between the types described in the previous section is given in

Figure 50

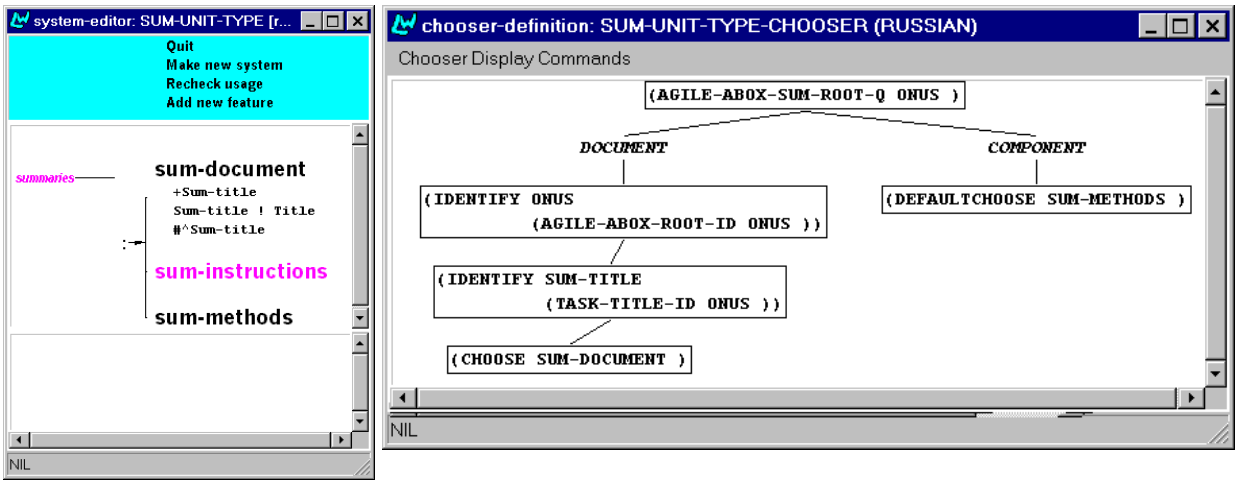


Figure 50 SUM-UNIT-TYPE system and chooser

The node METHOD of the A-box can recursively insert other METHOD nodes with preselect SUM-METJHODS in the text plan if there are other methods to achieve the goal associated with the node above (Figure 51):

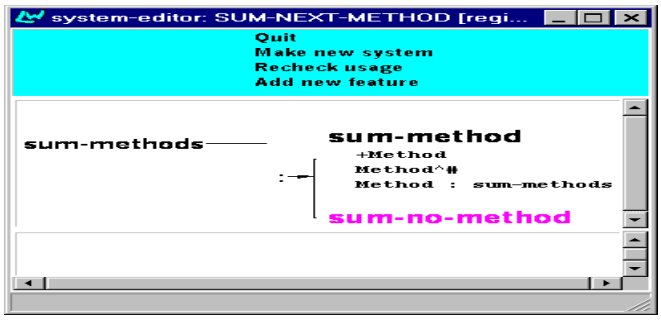
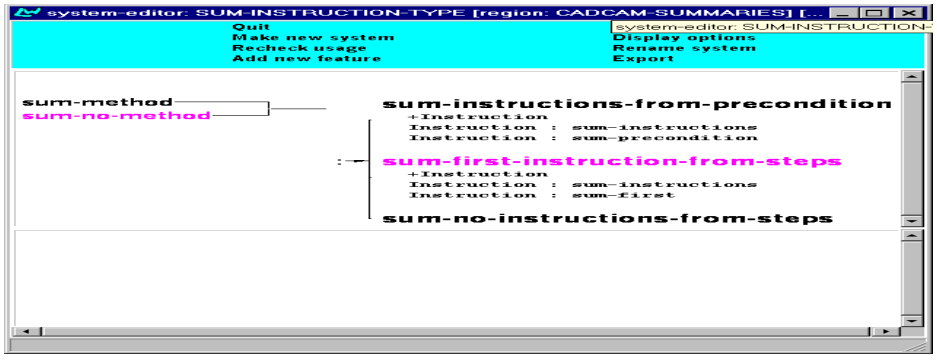


Figure 51 SUM-NEXT-METHOD system.

In the A-box every Method is described as the list of instruction-steps, so the text plan node METHODS can insert besides METHOD a node INSTRUCTION. The context of the instruction in A-box is significant. We use SUM-FIRST preselect to mark that the node inserted is the first step in a Method (except the case the FIRST STEP goes after precondition. In this case FIRST STEP is not marked by appropriate precondition), or



SUM-PRECONDITION to mark that the node inserted is precondition (Figure 52).

Figure 52 SUM-INSTRUCTION-TYPE system.

Association with the A-box context is realized by two gates: SUM-LOCALIZE-CONT-FOR-FRST-INSTRUCT and SUM-LOCALIZE-CNTXT-FOR-PRECOND since they are met in different contexts of A-Box structure. The localization is realised in the following choosers (Figure 53):

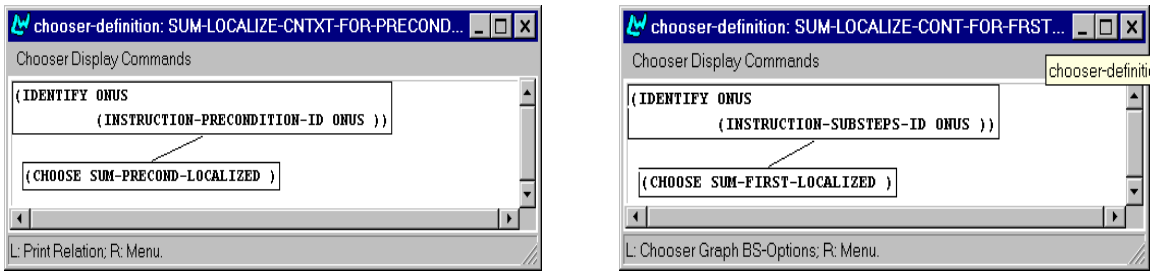


Figure 53 Choosers for the context localization.

If there is the TASK node which corresponds to the INSTRUCTION node its lexicalization is made by SUM-TASK-REALIZATION-STYLE system (Figure 54)

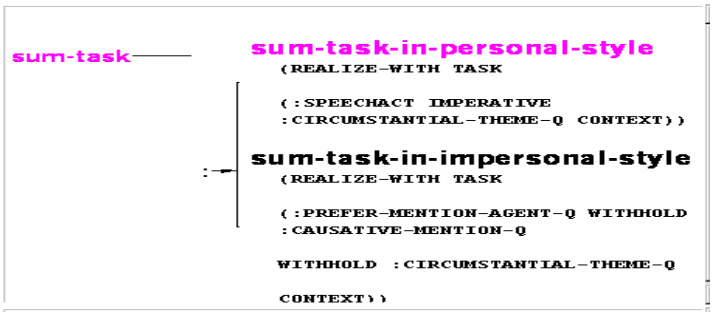


Figure 54 SUM-TASK-REALIZATION-STYLE system.

Besides the Instruction and Methods METHOD node can insert a terminal node Constraint in the text plan. The appropriate system is shown in Figure 55:

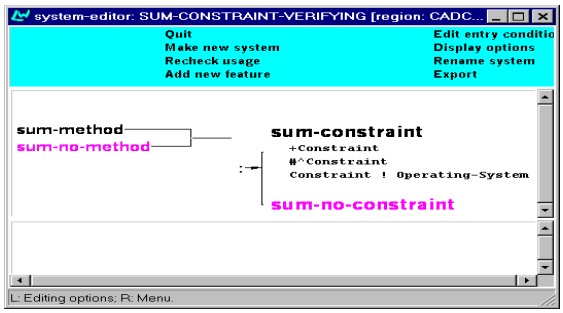


Figure 55 SUM-CONSTRAINT-VERIFICATION system.

Realization of terminal text plan nodes is identical to the full-instructions-style.

5. The CADCAM-FUNCTIONAL DESCRIPTIONS region

5.1 Overview

In this chapter we describe the CADCAM-FUNCTIONAL-DESCRIPTIONS region. This region is responsible for generating text plans for describing the functionality of the target application. We describe here:

- systems, choosers and inquiries required for the creation of a tree of elements constituting a text plan for functional descriptions;
- realization of text plan elements by the sentence planner;

5.2 A text type of functional descriptions

Functional descriptive texts that are possible for our A-box specifications can be grouped into two classes:

- "Descriptions of what Goal(s) a Method (user-action) achieves"
- "Descriptions of what Goal(s) a GUI object (...) achieves".

Either of them applies to actions with GUI objects, which are linked to a specific function. In our DM the list of concepts for such objects includes: BUTTON, KEY, OPTION and SOFTWARE-COMMAND. The first type describes situations, in which the user triggers a functional object and through that action realises the function defined for it. This type is called action-oriented functional descriptions:

(6) *By selecting the Add button, an element is added.*

(a) *Bu:*

С избиране на клавиша Add се добавя елемент.
By selection of button-def Add refl add-3sg element.

(b) *Cz:*

Vybráním tlačítka Add se přidávají elementy.
Selecting-ins button-gen Add refl add-3pl elements
By selecting the Add button one adds elements.

(c) *Ru:*

Нажатие кнопки Add добавляет элемент.
Selecting-nom button-gen Add add-3sg element-acc

The second type is aimed at associating a GUI object with its function in the interface. This type of functional descriptions is called object-oriented:

(7) *The Add button allows you to add an element.*

(a) *Cz* Tlačítko Add umožňuje přidat element.
Button Add enables add elements.

(b) *Bu:* Бутонът Add позволяв да добавите елемент.
button-det Add allows add-"da-construction"element

Бутонът Add позволяв добавяне на елемент.
button-det Add allows add-nominalization of element

(c) *Ru:* Кнопка Add позволяет добавить элемент.
button-nom Add allows add-inf element-acc

A functional description just presents a pair of actions one of which has an Actee expressed by a functional object of any type and the second presents the goal of the action as expressed in the A-box. The former is presented in the text by a noun – nominalisation (the first example) or just reduced only to the functional object name (the second example). The latter action in both cases is realised in impersonal form – infinitive clause for Czech and Russian and the functional substitute of the infinitive clause in Bg.

It follows from the discussion about the composition of functional descriptions that this style cannot be derived from just any A-box specifying the content of an instructional text. It must contain interface functional object as Actee in some action and this action must be a method of a goal.

5.3 Specifications for functional descriptions

An example of a text plan for functional descriptions is given in Figure 56.

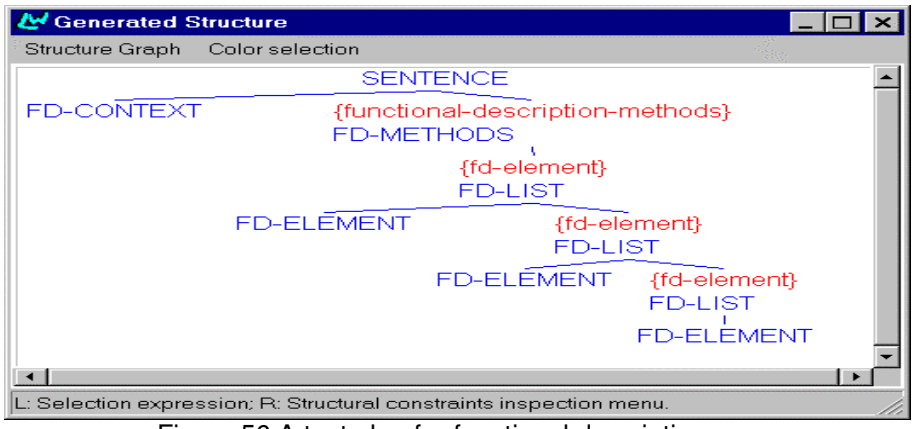


Figure 56 A text plan for functional descriptions

So, we model a traversal of an A-box to gather from it all information relevant for creating a list of functional descriptions (FDs) for functional objects (FOs).

The traversal takes into account four types of constituents:

- FD-title, which defines headers for the four sections of FD lists: actions with buttons, keys on the keyboard, options in the menu and software commands typed in the command line prompt.

- FD-list, which forms a list of functional descriptions and consists of first/rest slots; and
- FD-element, which is associated with the content to be delivered by a single functional description.

The FD-title element of the text plan is associated with the TEMPLATE class, which is accompanied by a language-specific pattern. The need to associate an FD-element, as an element of a text plan, with its sense requires addition of a new configurational concept into the CAD-CAM DM:

```
(define-concept FD-configuration  (INSTRUCTION-SCHEME)
  ((METHOD-ACTION :type USER-ACTION)
   (TARGET-ACTION :type USER-ACTION)
   ))
```

Its first slot (Method-action) corresponds to the method (*click the Add button*), while its second slot (Target-action) corresponds to the goal the user wants to achieve (*add an element*). Thus, we have the following mapping between text plan pieces and semantic pieces:

FD-TITLE ←→ TEMPLATE

FD-METHOD ←→ a METHOD of a PROCEDURE

FD-LIST ←→ a STEP of a METHOD

FD-ELEMENT ←→ FD-CONFIGURATION

The system which achieves selection between those types is given in Figure 57.

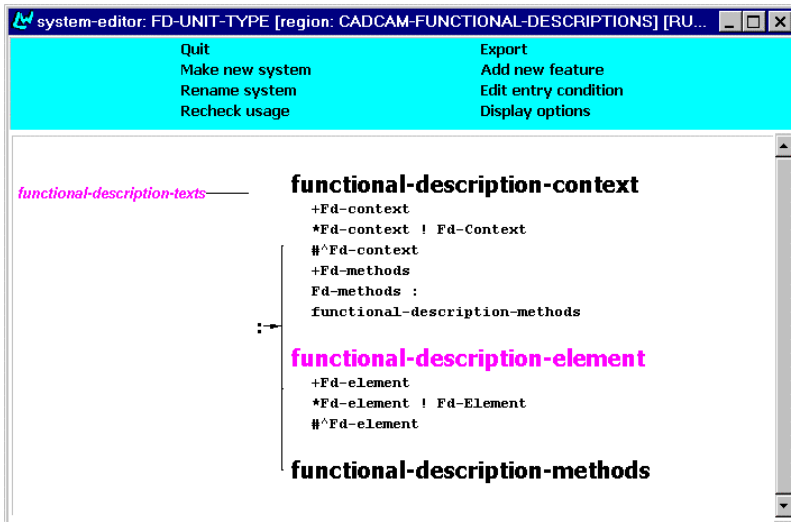


Figure 57 FD-UNIT-TYPE system

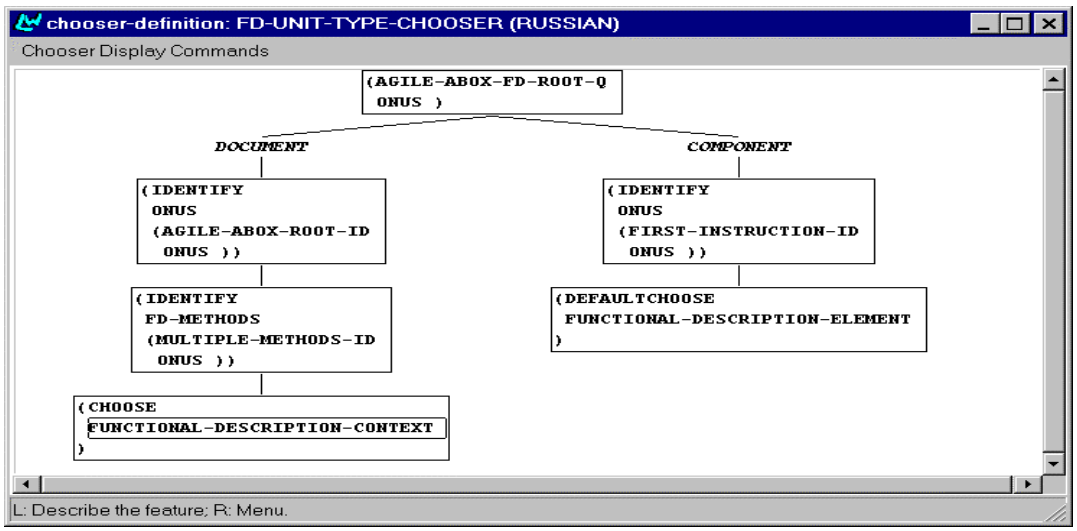


Figure 58 FD-UNIT-TYPE chooser

The selection is based on a difference between entering the root of an A-box or traversing its methods. In the first case, the FD methods constituent is identified with a list of respective methods, while in the second case, the FD-list constituent is identified with the next action. Also this leads to realization of subchapters of the ready-reference section.

Figure 59 FD-list-complexity system

When an element of an FD-list is being realized, the traversal checks the possibility to extend the list by adding more elements to it (Figure 59). Note that functional descriptions cannot be derived from just *any* A-box. The reason being that sometimes there just is not any kind of information available in that A-box that could be planned as a functional description.

The condition for selecting an instruction procedure description suitable for a functional description is formulated as follows:

> If there is a Goal, which has a method with one step and the step is achieved by a GUI object, which concept belongs to the list of BUTTON, ICON, KEY and OPTION, then this is a candidate for inclusion into the list of functional descriptions of an A-box.

That is you need this construction to be available in the A-box for the region to be able to distil a "text plan" from it.

The above conditions on the form of the content is formally expressed in the implementation of the LONGER-LIST-OF-FDS-Q inquiry, which is used in the chooser of the FUNCTIONAL-DESCRIPTION-ELEMENTS-COMPLEXITY system. The chooser is displayed below.

In this case, the existing FD-element is identified with an A-box piece, otherwise the recursive traversal of the A-box continues until it is finished. In the latter case, no-more-functional-description-elements is selected (Figure 60).

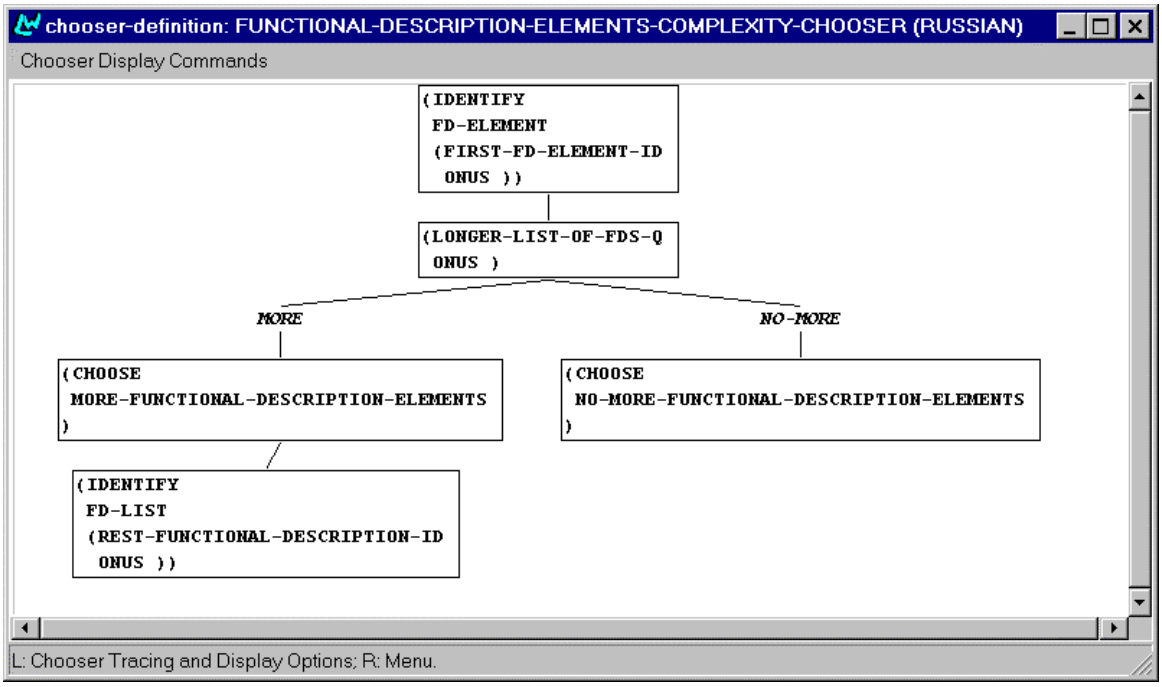


Figure 60 FD-list-complexity chooser

The association between an FD-configuration as a semantic specification piece and an FD-element, as a text plan piece is defined in the code for the inquiry that is used in the chooser given above:

1. an FD-configuration is produced;
2. its target action slot is assigned to the goal of the current procedure;
3. its method action slot is assigned to the goal of the first substep of the thirst method for achieving the current procedure;

Finally, the style of functional descriptions is chosen depending on the option in the interface. This leads to addition of realization statements shown in Figure 61.

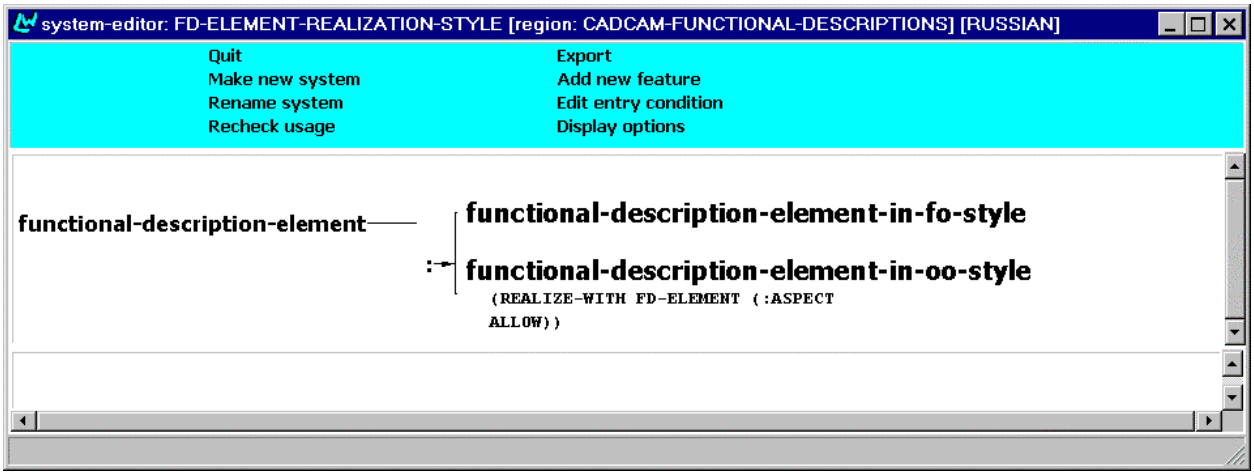


Figure 61 FD-element realization style

5.4 Realization of text plan elements

The sentence planner provides mapping of A-box pieces associated with text plan elements into SPL structures, which are served as input to the respective tactical generators (cf the overall architecture of the AGILE system, and the place of the TSM therein, as discussed in [TEXM2]). The conditions for getting proper SPLs out of FD-configurations involves setting the method action as the actor of the target action, using the following instruction:

1. provided that we have a proper FD configuration with the filled target action and method action slots,
2. the target action slot is splized to form the major structure for the target SPL;
3. depending on the style of functional descriptions, either the method action slot of the FD configuration (to achieve the style of '*clicking a button does X*') or the actee of the method action slot of the FD configuration (to achieve the style of '*a button allows you to do X*') is splized to fill the :actor slot of the target SPL.

This also assumes the need to suppress the expression of the user as the default actor of the target action in the case of functional descriptions.

The SPLs, which are produced by the sentence planner for a typical functional description in two styles, are as follows:

```
(EXAMPLE
    :NAME   TSM-RU-FD-1
    :LOGICALFORM
(|a130| / DM::END-LINE :ASPECT ALLOW :ACTEE
         (|a132| / DM::POLYLINE)
         :ACTOR
         (|a138| / DM::PRESS :ACTEE
                  (|a140| / DM::KEY :CLASS-ASCRIPTION
                           (|a141| / DM::GUI-RETURN))))
    :SET-NAME   TSM-RU-FD
)

(EXAMPLE
    :NAME   TSM-RU-FD-2
    :LOGICALFORM
(|a130| / DM::END-LINE :ASPECT ALLOW :ACTEE
         (|a132| / DM::POLYLINE)
         :ACTOR
```

```
        (|a138| / DM::PRESS :ACTEE
                (|a140| / DM::KEY :CLASS-ASCRIPTION
                        (|a141| / DM::GUI-RETURN))))
    :SET-NAME   TSM-RU-FD
)
```

This produces the following output in Russian:

Нажатие клавиши Return завершает полилинию.

Клавиша Return позволяет завершить полилинию.

6. The CADCAM-TOCS region

6.1 Overview

This chapter describes the CADCAM-TOCS region, which implements the generation of text plans for TABLE OF CONTENTS (TOC) text type. The discussion covers the features and the structure of the TOC as it has developed since the specifications and first implementations described in [TEXS3].

The texts included in the AGILE generated documents are supposed to form a consistent part of a technical manual (see [TEXS3]). The Table-of-Contents part of a manual is constructed by a numbered list of the Task titles from the list of task descriptions. As the basic texts in AGILE manuals are full instruction procedural texts, the Task titles present the main goals of the procedures. The layout of TOC within the AGILE project is designed considering the common type of Internet style TOC. Each TOC element represents hyperlink to some main part (chapter) of document bundle.

6.2 Description of the implementation

The implementation of this text type is achieved through the CADCAM-TOCS (Figure 62) region built within the TSM resources.

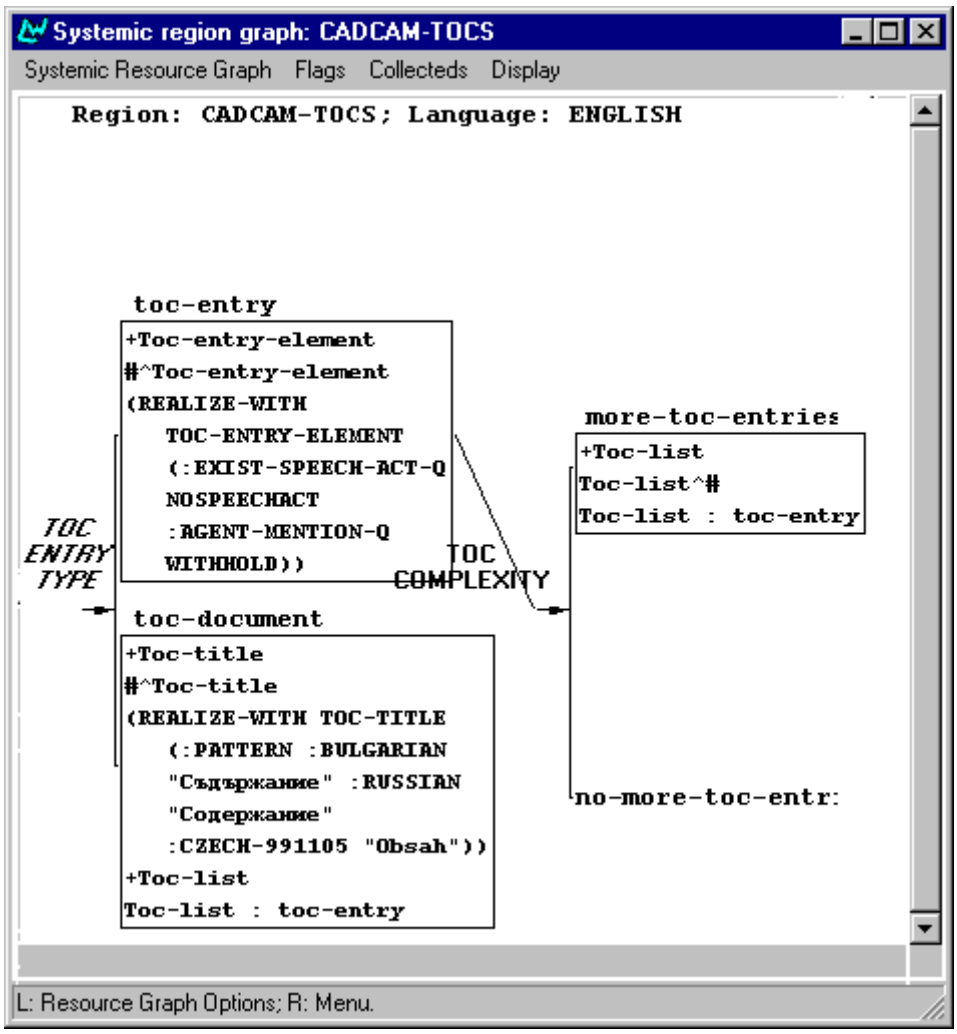


Figure 62 - CADCAM-TOCS

The region follows previously defined structure based on the CADCAM configuration concept PROCEDURE-LIST, which was described in [TEXS3]. The system TOC-ENTRY-TYPE (Figure 63) starts the cycle of TOC gathering. Here we define whether we are dealing with TOC-ENTRY-ELEMENT or TOC-TITLE. At this point we should mention that TOC-TITLE is fixed to “Table of Contents” expression, realized as a template for each language. After the realisation of TOC-TITLE, each TOC-ENTRY-ELEMENT is traversed and realized as nominalization. The operator REALIZE-WITH achieves that in the TOC-ENTRY branch of the system, which adds the necessary nominalization features.

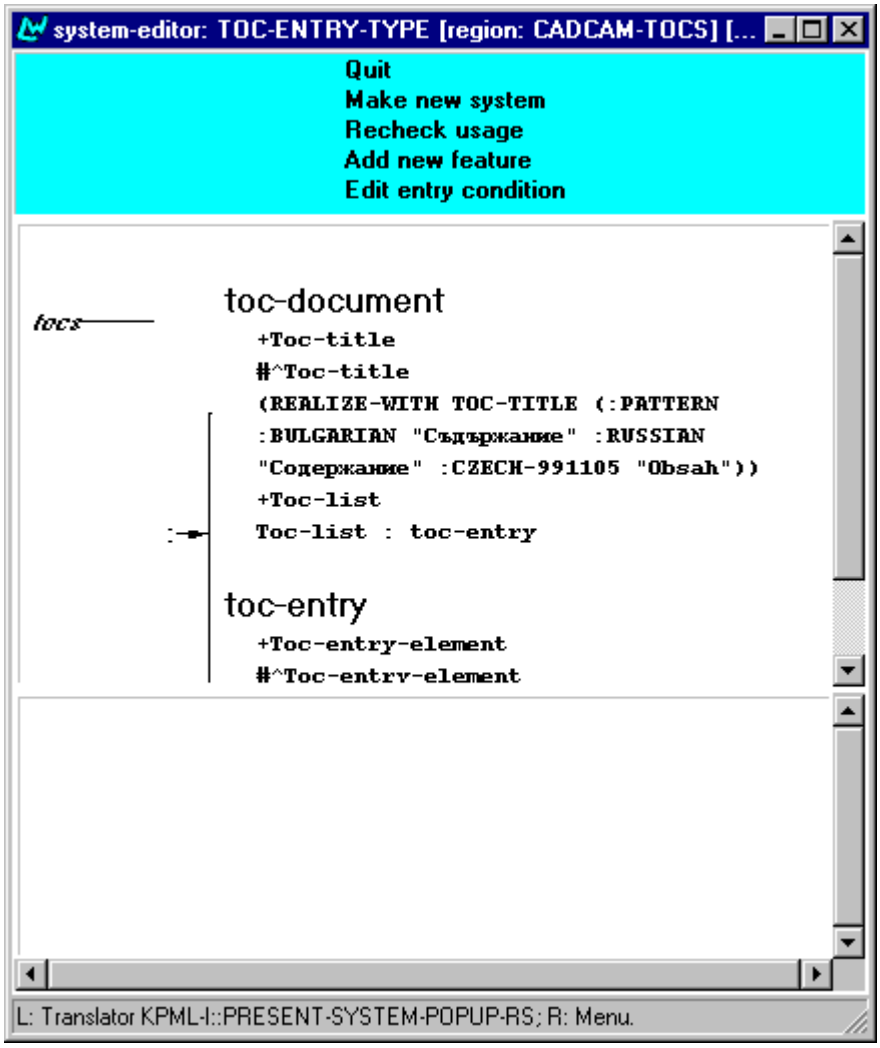


Figure 63 - TOC-ENTRY-TYPE system in the CADCAM-TOCS region

The chooser of the system – TOC-ENTRY-TYPE-CHOOSER uses the TSM inquiry AGILE-ABOX-ROOT-Q to determine what kind of TOC element we are dealing with. If the process is at the beginning of the structure then this is TOC-TITLE case. The rest of the elements are realized as nominalised elements, which represent the main GOAL of each PROCEDURE in the structure. The chooser graph is given on Figure 64.

As it is seen from Figure 62 the next system is TOC-COMPLEXITY. The main purpose of the system is to check what is the current element within the structure processed by the text planner and hence determine whether there will be further need of realizing more elements. This is achieved by traversing the CADCAM concept PROCEDURE-LIST. While there are more PROCEDUREs in the PROCEDURE-LIST, new TOC items are inserted and processed. The scheme of the system is given on Figure 65.

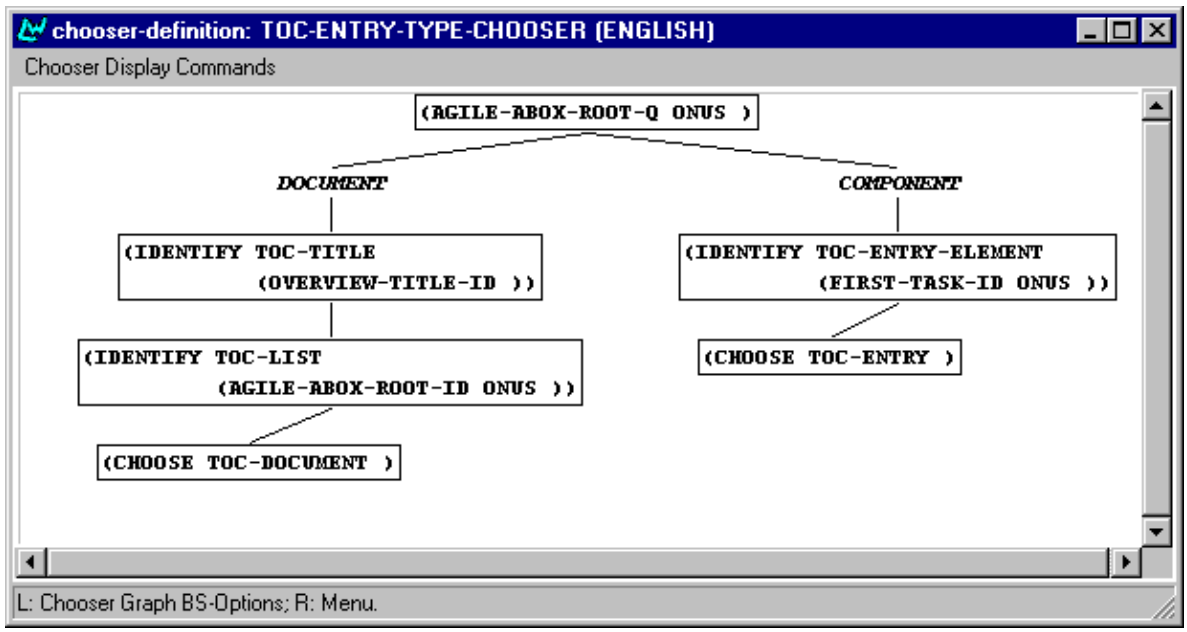


Figure 64 - TOC-ENTRY-TYPE-CHOOSER

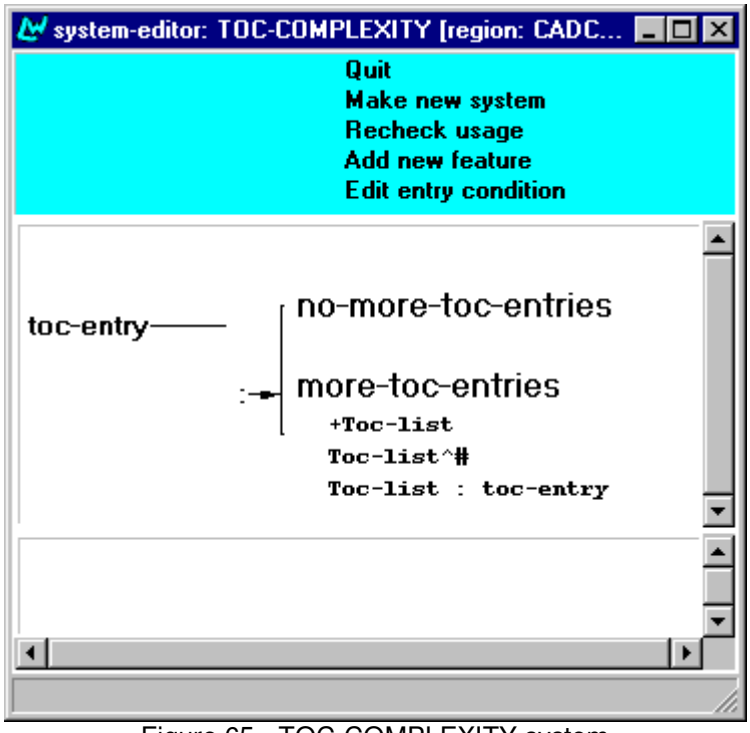


Figure 65 - TOC-COMPLEXITY system

The chooser TOC-COMPLEXITY-CHOOSER (Figure 66) uses the TSM inquiry LONGER-LIST-Q, which checks whether there are more PROCEDUREs left in the PROCEDURE-LIST. The elements are inserted recursively, following the recursive structure of the PROCEDURE-LIST. On Figure 67 is shown the structure of a TOC received as a text plan. The whole structure is one recursive list bundle, which contains the TOC elements. At first level we can see the TOC-TITLE and the first TOC-ENTRY-ELEMENT. The rest of the list contains all the other TOC-ELEMENTs. For each elements

new TOC-LIST is inserted. That way we can achieve an infinite number (as many as we needed) of TOC elements.

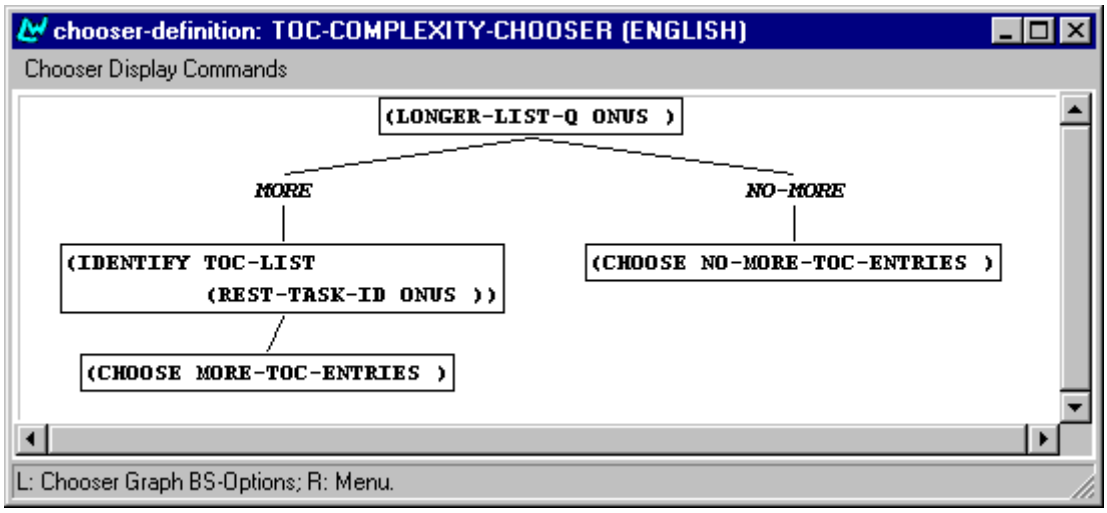


Figure 66 - TOC-COMPLEXITY-CHOOSER

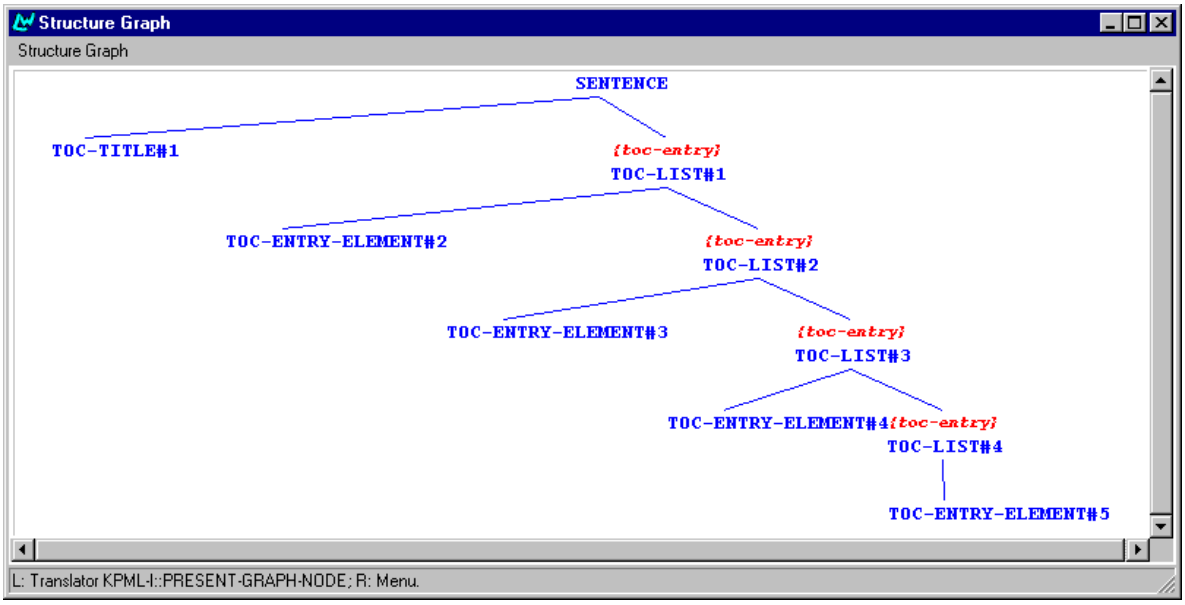


Figure 67 - TOC structure tree

6.3 HTML marking/punctuation

The mechanism for HTML marking within the TSM allows the writer of new text types to easily make HTML marking. In order to achieve such marking, we have to define the mapping between the different text plan elements and the desired HTML tags. In the case of TOCs the mapping is the following:

6.3.1 Pre-punctuation rules:

TOC-TITLE → <FONT SIZE=+2>

TOC-LIST → <OL>

TOC-ENTRY → <LI>

6.3.2 Post-punctuation rules

TOC-TITLE → </FONT>

TOC-LIST → </OL>

TOC-ENTRY → </LI>

That way the TOC realisation covers the specification in [TEXS3]. (Figure 68).

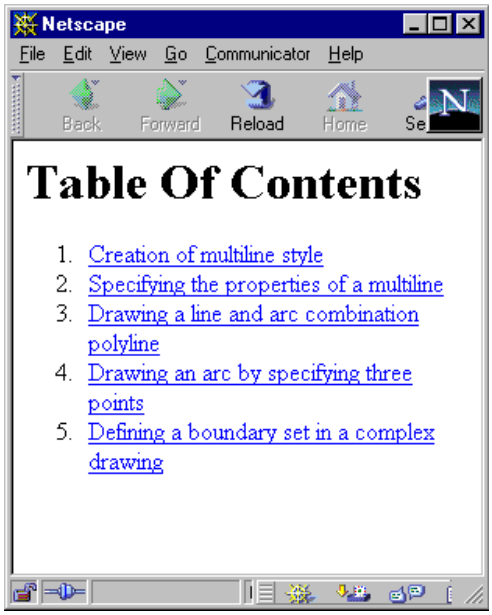


Figure 68 - TOC HTML marking

7. The CADCAM-OVERVIEWS region

7.1 Function and text type

This chapter covers description of the mechanism implemented for generation of Overview text type. In AGILE the aim of Overview text is presenting a summary of the operations described in a technical manual or chapter containing a set of procedural texts (see [TEXS3]). The realisation of this text type is achieved in the CADCAM-OVERVIEW region, which is now part of the text planner regions. The main subject of this chapter is to describe the features of the Overview text type as they are implemented. The Overview text type depends (the same way as TOC does) on previously defined structure, based on the CADCAM concept PROCEDURE-LIST (see [TEXS3] for further details).

An overview text is composed by complex sentences. In order to achieve a sensible running OVERVIEW text the text planner has to consider some constraints over the aggregation, resulting in limited length sentences. The implementation leads to sentences, containing three dependent elements at most. The ordering of the elements in the OVERVIEW sentences do not necessarily follow the user ordered selection of A-boxes. Semantic grouping was considered as a possibility in TEXS3. The AGILE user may choose to group the elements in the OVERVIEW text either by the same actee or by actees, participating in the same semantic hierarchy.

Examples:

(8) **Personal style, with explicit 2nd person plural reference to the user**

(a) *English*

The AGILE-AUTOCAD system allows you to create multiline style, to edit multiline style and to specify the properties of a multiline. You can also draw a line and arc combination polyline and change a component of polyline.

(b) *Czech*

Systém AGILE-AutoCAD Vám umožňuje, vytvořit styl multičáry, ……styl multičáry, určit vlastnosti multičáry. Také můžete nakreslit křivku složenou z čar a oblouků, …..

(c) *Bulgarian*

Системата AGILE-AUTOCAD позволява да създадете стил на мултилиния, да редактирате стил на мултилиния, да зададете характеристиките на мултилиния. Вие можете също да начертаете полилиния, съставена от отсечки и дъги, да промените елемент на полилиния.

(d) *Russian*

Система AGILE-AUTOCAD позволяет создавать стиль мультилинии, редактировать стиль мультилинии, задавать свойства мультилинии. Вы также можете рисовать полилинии, состоящие из отрезков прямых и дуг, изменять элемент полилинии.

(9) **Personal style, with explicit 2nd person plural reference to the user**

(a) *English*

The AGILE-AUTOCAD system allows creating multiline style, editing multiline style, specifying the properties of a multiline. You can also draw a line and arc combination polyline and change a component of polyline.

(b) *Czech*

Systém AGILE-AutoCAD Vám umožňuje, vytvořit styl multičáry, ……styl multičáry, určit vlastnosti multičáry. Také můžete nakreslit křivku složenou z čar a oblouků, …..

(c) *Bulgarian*

Системата AGILE-AUTOCAD позволява да се създаде стил на мултилиния, да се редактира стил на мултилиния, да са зададат характеристиките на мултилиния. Възможно е също да се начертае полилиния, съставена от отсечки и дъги, да се промени елемент на полилиния.

(d) *Russian*

Система AGILE-AUTOCAD позволяет создавать стиль мультилинии, редактировать стиль мультилинии, задавать свойства мультилинии. Возможно также рисовать полилинии, состоящие из отрезков прямых и дуг, изменять элемент полилинии.

7.2 Planning of syntactic aggregation (conjunction)

There are two types of syntactic aggregation – conjunction and disjunction (see 3.2.1 for a detailed description of these phenomena). The OVERVIEW case implements a CONJUNCTION type of syntactic aggregation. The planning of syntactic aggregation or conjunction for Overviews is achieved by doing aggregation while following the structure of the CADCAM concept PROCEDURE-LIST. Each PROCEDURE-LIST has two slots FIRST and REST. The slot FIRST holds a PROCEDURE, and the REST slot contains the list with all the other PROCEDUREs. Here, we are interested only in the GOAL slot of each procedure. The CONJUNCTION role is to plan the GOALs as running text giving summarised information for the Aboxes (PROCEDUREs).

7.3 Description of the implementation

The CADCAM-OVERVIEW region envelops the specifications of the Overview text type. We start with the system OVERVIEW-ENTRY-TYPE. It comes to determine whether there will be OVERVIEW-TITLE or OVERVIEW-ELEMENT realized.

Both elements, OVERVIEW-TITLE and OVERVIEW-ENTRY-ELEMENT are planned to have either PERSONAL or IMPERSONAL style. This is done by adding the necessary 'REALIZE-WITH' statements, which provides the formation of the proper statements within the SPL structure for each style.

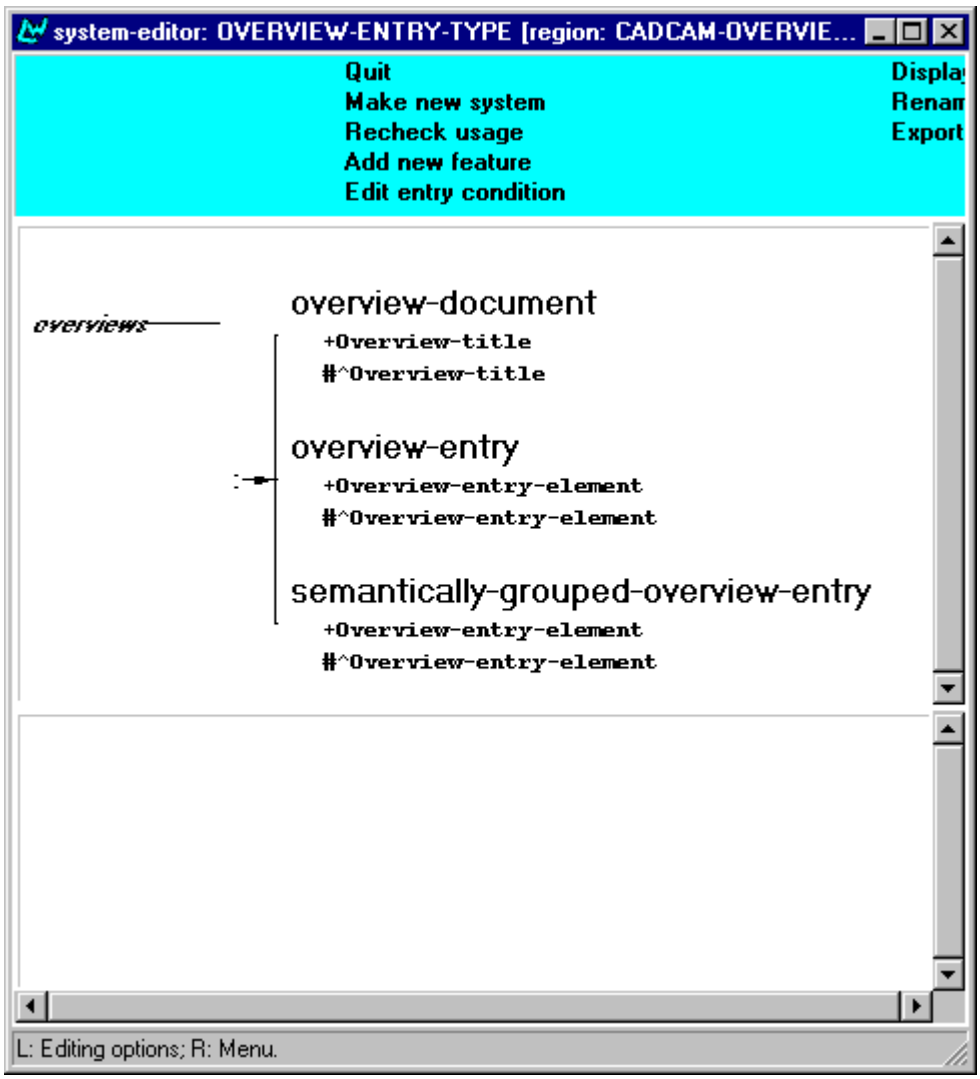


Figure 69 - OVERVIEW-ENTRY-TYPE system

As it is seen from Figure 69 the system has three outputs – OVERVIEW-DOCUMENT, OVERVIEW-ENTRY and SEMANTICALLY-GROUPED-OVERVIEW-ENTRY. OVERVIEW-DOCUMENT introduces title for the Overview. The title is realized by using template for each language. The other two outputs OVERVIEW-ENTRY and SEMANTICALLY-GROUPED-OVERVIEW-ENTRY introduce new OVERVIEW-ENTRY-ELEMENT. The decision for whether we need OVERVIEW-TITLE or OVERVIEW-ENTRY-ELEMENT is done by the chooser OVERVIEW-ENTRY-TYPE-CHOOSER (Figure 70), which uses the TSM inquiry AGILE-OVERVIEW-ROOT-Q and determine whether we are at the beginning of the Overview planning or not. If we are at the beginning then an Overview title is inserted and realize as a template. Otherwise new Overview element is introduced. Further the chooser checks what kind of element should be inserted. The inquiry WHAT-OVERVIEW-GROUPING-STYLE-Q checks what style of grouping the user has requested – semantic or limited/non-limited.

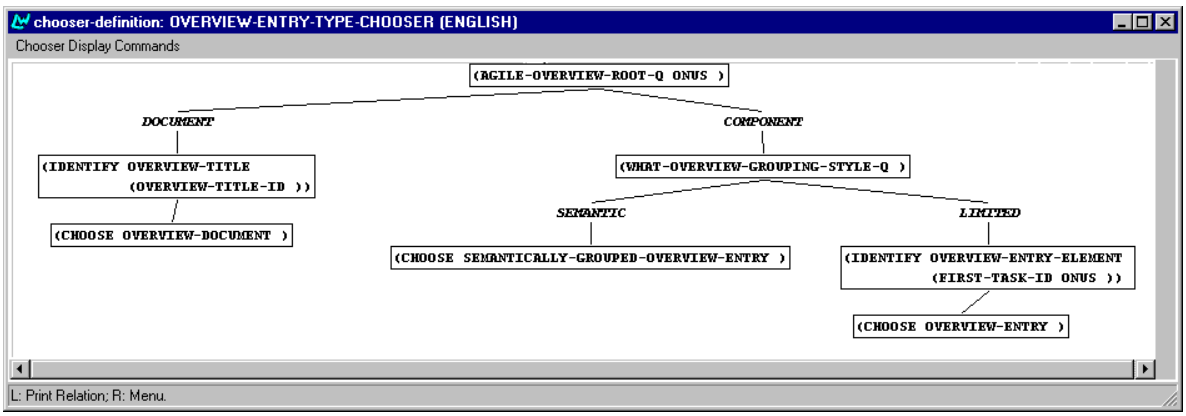


Figure 70 - OVERVIEW-ENTRY-TYPE-CHOOSER

The semantic grouping relies on previously formed list, which contains the GOALs of the PROCEDUREs in the PROCEDURE-LIST (the A-box structure which is used for the planning of the Overview) ordered either by actees that are the same or by actees, which have the same type.

The limited grouping makes separation of the elements according to a previously defined fixed number. This number defines how much elements are planned for each group to have.

7.3.1 Semantic realization of elements

7.3.1.1 Preparation for semantic grouping

At this point we describe what is done within the implementation in order to prepare a structure, which will be the main “wheel” for planning of semantic aggregation. The structure type is simply a list of GOALs, found in the PROCEDURE-LIST, used as a base for text planning of Overview. Between the semantic groups along the list there are separators, which mark end/start of different semantic groups. The list is received by doing traversal of the PROCEDURE-LIST and reordering the GOALs in the way we want to produce them – ordered either by the same actee or by actee type (see [TEXS3] specifications for more details on this topic).

7.3.1.2 Implementation of semantic grouping

Once the OVERVIEW-ENTRY-ELEMENT is introduced in the branch SEMANTICALLY-GROUPED-OVERVIEW-ENTRY (Figure 69) it is identified with the current member of the semantically grouped list, defined in advance.

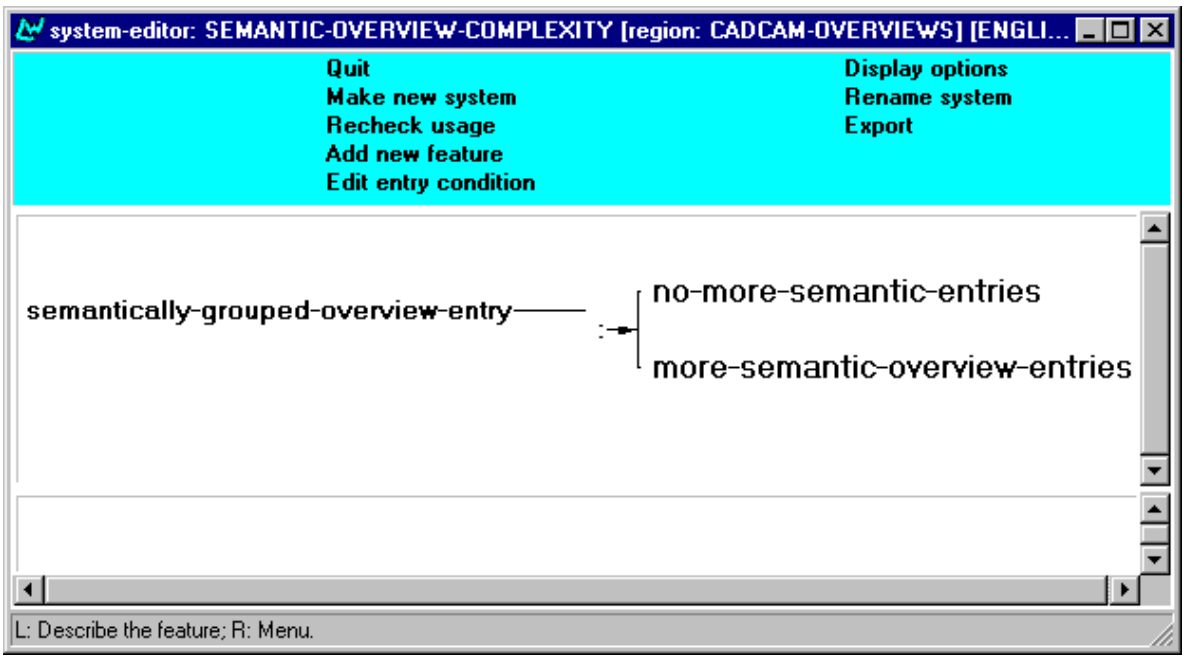


Figure 71 - SEMANTIC-OVERVIEW-COMPLEXITY system

The system SEMANTIC-OVERVIEW-COMPLEXITY (Figure 71) decides whether there will be more Overview elements realized. Using the inquiry LONGER-SEMANTIC-GROUPED-LIST-Q implemented in the chooser – SEMANTIC-OVERVIEW-COMPLEXITY-CHOOSER, the system either continues to introduce MORE groups of elements or stops the process (Figure 72).

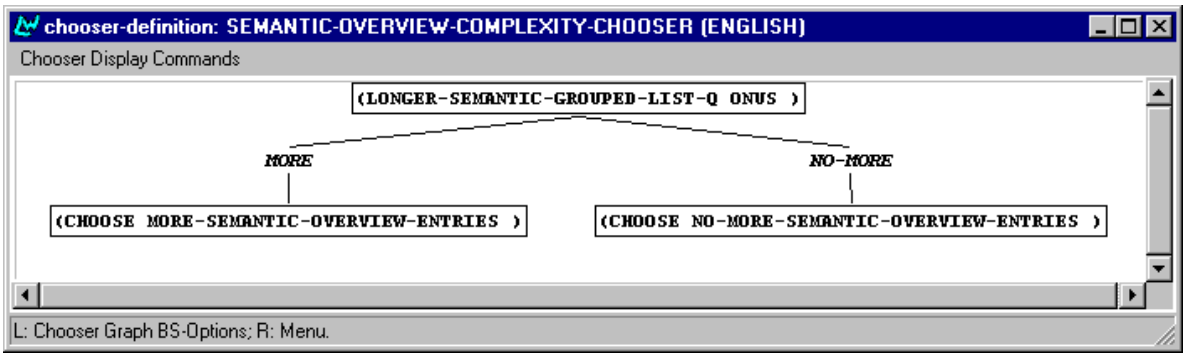


Figure 72 - SEMANTIC-OVERVIEW-COMPLEXITY-CHOOSER

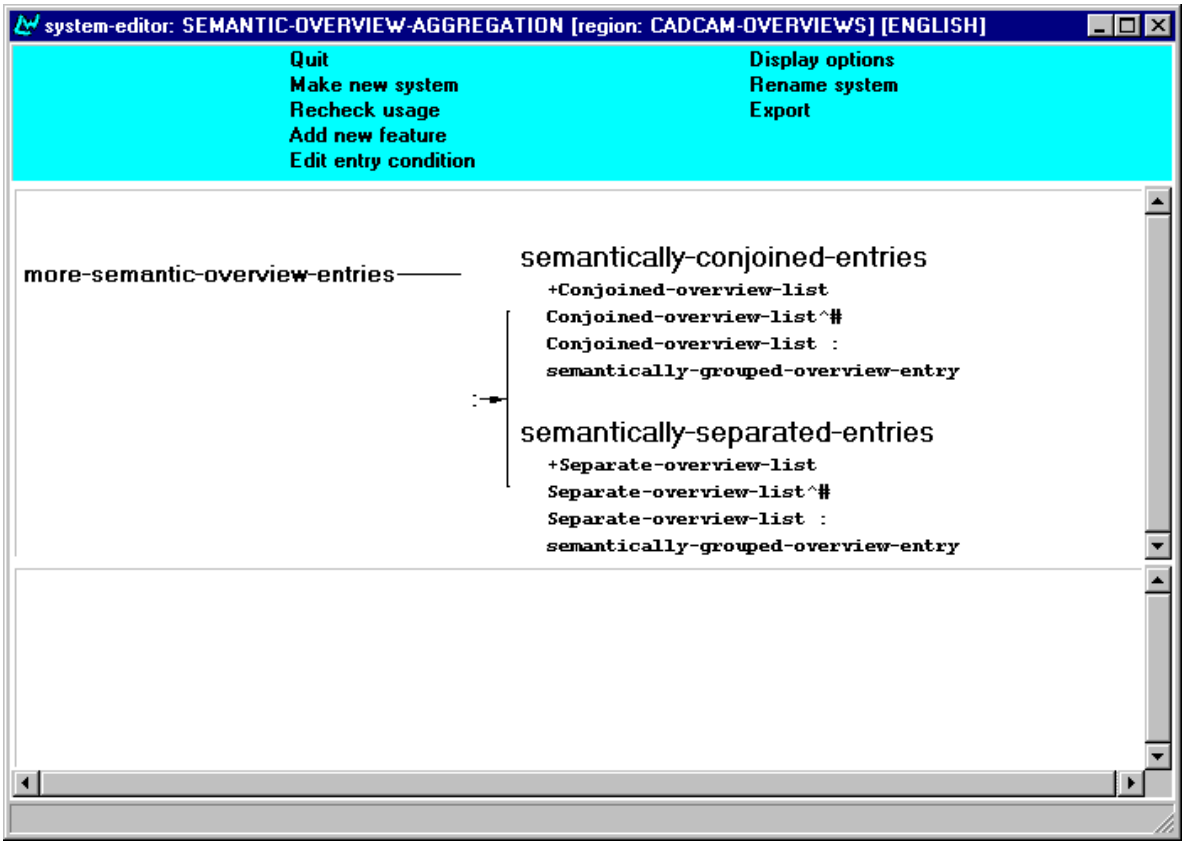
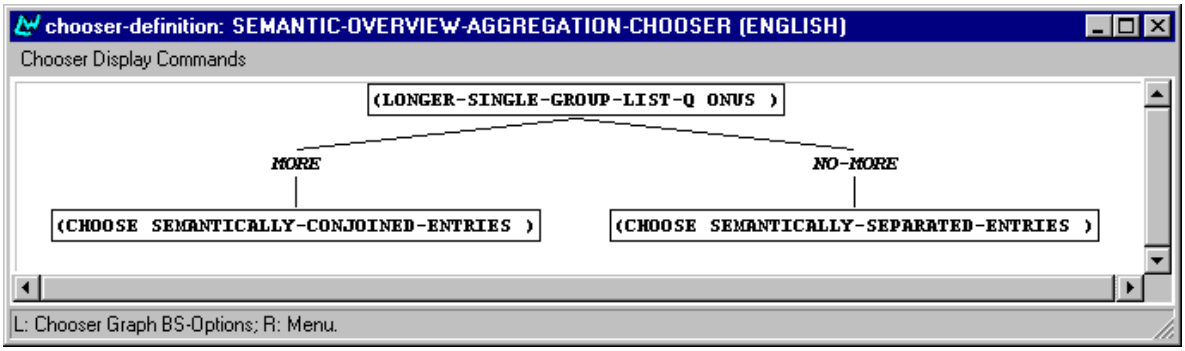


Figure 73 – SEMANTIC-OVERVIEW-AGGREGATION system

Figure 73 shows the system SEMANTIC-OVERVIEW-AGGREGATION, which handles the conjunction of the semantically grouped elements. By using the inquiry LONGER-SINGLE-GROUP-LIST-Q. The chooser of the system SEMANTIC-OVERVIEW-AGGREGATION-CHOOSER decides whether there is more elements to be gathered together. If there is no more elements, a new list is inserted, which marks the beginning of a new semantic group of elements.



This way we achieve the desired specifications for semantic grouping of the elements described in [TEXS3].

7.3.2 Limited and non-limited realization of elements

The other way of grouping specified in [TEXS3] and implemented here is by limiting the elements, according to their number.

The implementation of this type of grouping starts within the system OVERVIEW-ENTRY-TYPE (Figure 69). The output OVERVIEW-ENTRY introduces new

OVERVIEW-ENTRY-ELEMENT, which is identified with the GOAL of a PROCEDURE from the PROCEDURE-LIST (the A-box structure which is used for retrieving the text plan).

Next we follow the system OVERVIEW-COMPLEXITY (Figure 74), which checks whether there are more elements left in the list of PROCEDUREs, and if that is the case realize them. The control over whether we have more elements is done the same way as in the system TOC-COMPLEXITY (see the previous chapter – CADCAM-TOCS region). The TSM inquiry LONGER-LIST-Q-CODE checks whether there are more elements left.

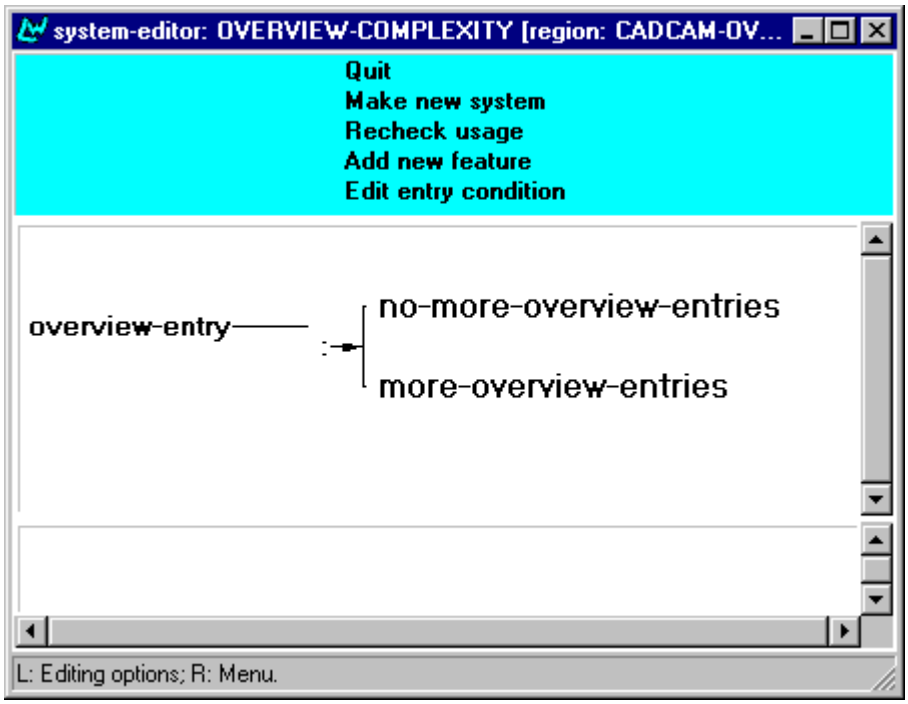


Figure 74 - OVERVIEW-COMPLEXITY system,

The next system to which the control is given is OVERVIEW-AGGREGATION (Figure 75). Here is the starting point where we start planning OVERVIEW syntactic aggregation by using conjunction. The conjunction is planned to be performed either as a bundle of conjunction elements or as distributed conjunction groups.

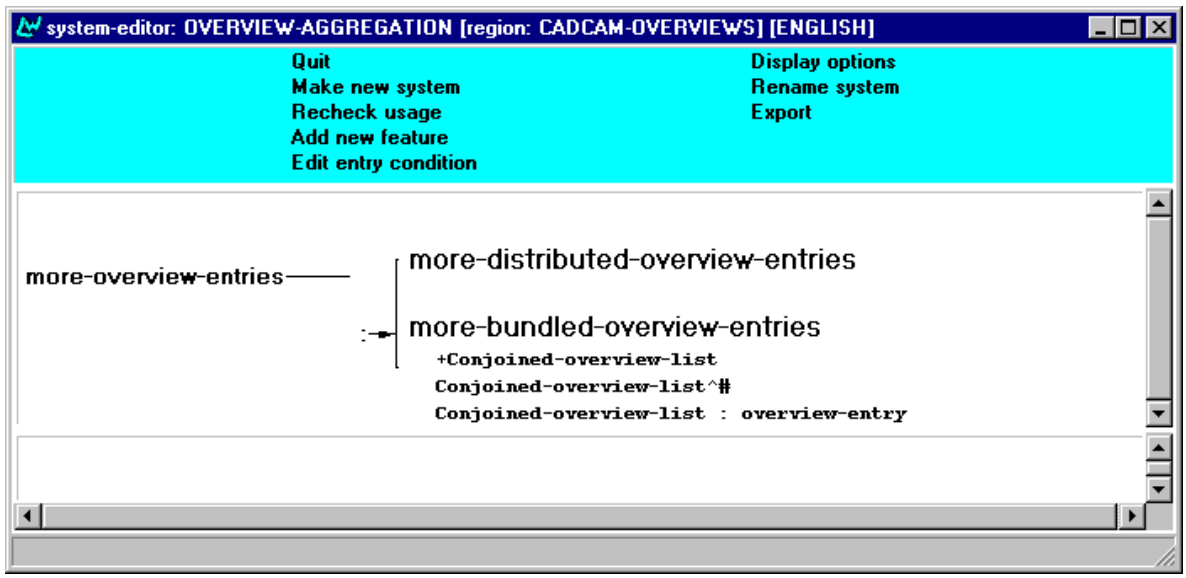


Figure 75 - OVERVIEW-AGGREGATION system

The output of the system - MORE-BUNDLED-OVERVIEW-ENTRIES gathers the processed elements as a whole bundle. It just realises them as one conjunct list of elements. Here we insert the function CONJOINED-OVERVIEW-LIST, which when encountered by the sentence planner is realized as a conjunction of the elements within the CONJOINED-OVERVIEW-LIST.

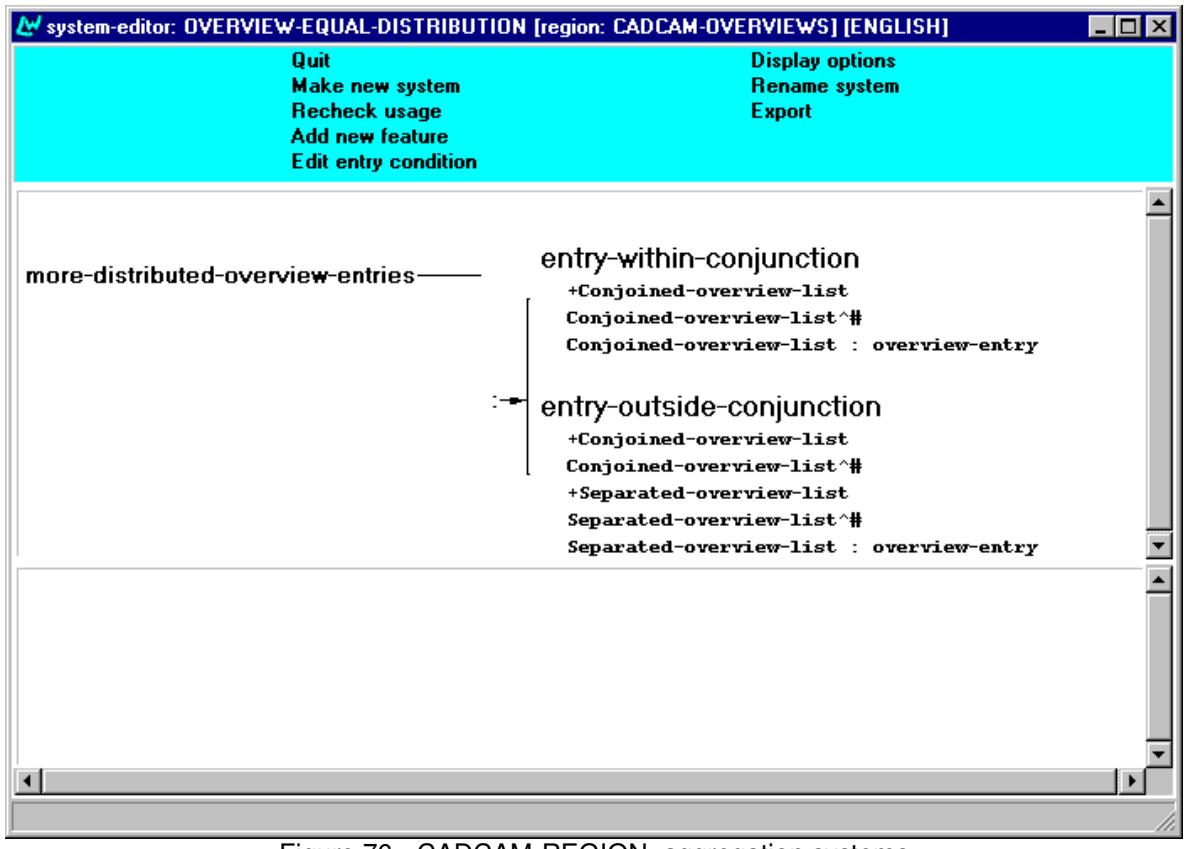


Figure 76 - CADCAM-REGION, aggregation systems

The other output of the system – MORE-DISTRIBUTED-OVERVIEW-ENTRIES is the beginning place, where we start to plan the distribution of the conjunct elements. Further, the processing is led by OVERVIEW-EQUAL-DISTRIBUTION system (Figure 76). The planning for the distribution of the elements is done in OVERVIEW-EQUAL-DISTRIBUTION. At this moment the distribution is based simply on the number of conjunct elements. The sentences planned represent groups of equally distributed conjunct elements. This type of grouping is done by the inquiry HOW-MANY-OVERVIEW-ENTRIES-Q-CODE, which is counting the number of elements realized up to a certain point. On Figure 77 is shown the chooser of this system – OVERVIEW-EQUAL-DISTRIBUTION-CHOOSER that decides where the output of the system should be directed by using the inquiry described above. If previously defined number of elements is exceeded new group of conjunct elements is created. That way we achieve equally spread aggregation.

Different styles of syntactic aggregation (conjunction) – distributed and bundled are shown on Figure 78 and Figure 79. They represent graph structures received during the generation. The difference between the two schemes is the function SEPARATED-OVERVIEW-LIST, which ensures the separation of conjunct elements.

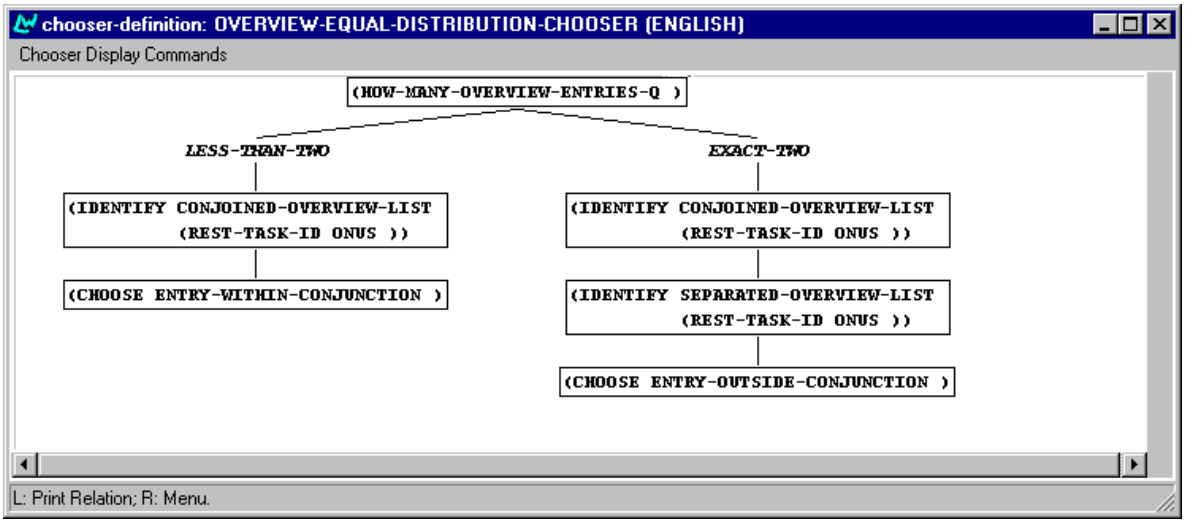


Figure 77 - OVERVIEW-EQUAL-DISTRIBUTION-CHOOSER

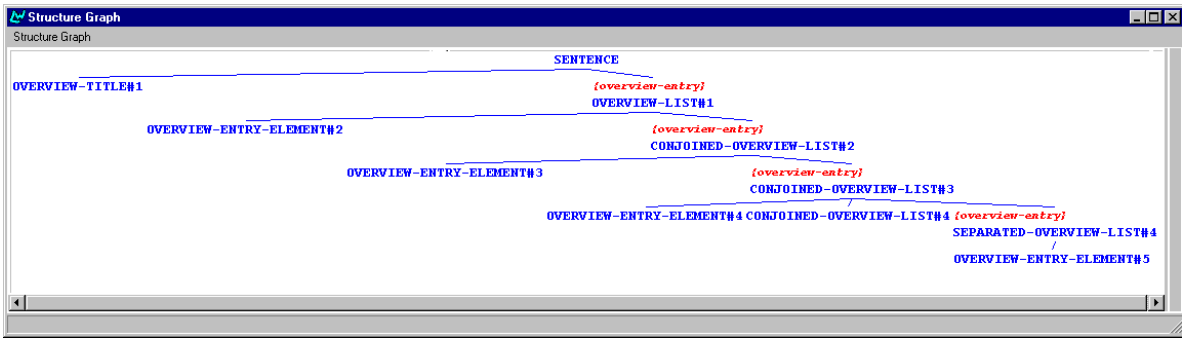


Figure 78 - GRAPH STRUCTURE WITH DISTRIBUTED CONJUNCTION

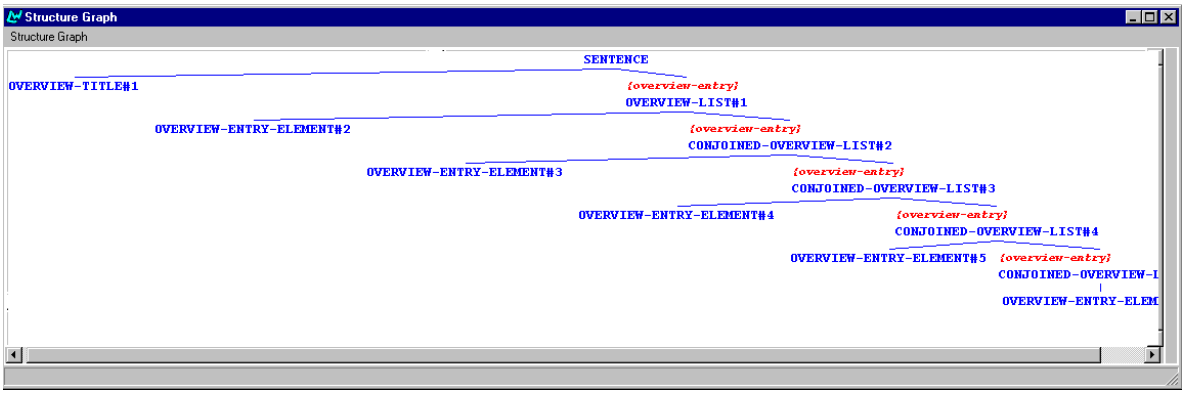


Figure 79 - GRAPH STRUCTURE WITH BUNDLED CONJUNCTION

8. Conclusions

The aim of the WP5.3 task was to develop a framework that would enable the integration of different approaches to discourse structuring, and provide a more general perspective on strategies for text generation that would be applicable beyond the immediate domain of application defined for AGILE. In [TEXS3] we proposed detailed specifications of an approach that would answer to this aim, by:

1. enabling the development of a *text grammar* to specify the generation of text plans for a particular text type, including the planning of complex textual phenomena like discourse aggregation, syntactic aggregation, and sequence marking;
2. developing an approach to discourse structuring based on Halliday's Systemic Functional Grammar (SFG), Mann and Thompson's Rhetorical Structure Theory (RST), and the Praguian Functional Generative Description (FGD), that would make it possible to construct complex but coherent text plans and have textual (information) structure reflected by contextually appropriate word order; and,
3. illustrating the flexibility of the entire approach on a variety of text types commonly found in software manuals.

In this deliverable we showed how the TSM for the final prototype achieved this complex aim. We accomplished flexible development of text grammars (point 1, above) first of all by reimplementing the sentence planner. For the final prototype, the sentence planner is capable of interpreting complex text plans, and facilitates a principled and straightforward way of extending its coverage of text plan configurations. With the sentence planner in place, we showed how a variety of text types could be planned by different regions, and how the sentence planner could -indeed- be easily extended to cover the configurations specific to a particular type of text plans.

Besides the full procedural instructions text type, familiar from the IMD TSM [TEXS2/TEXM2], we also developed modules that are capable of producing different text types, all using the same type of content-specification as input. In many a case, the principled approach taken in the development of the CADCAM-INSTRUCTIONS region dealing with the generation of text plans for full procedural texts, paid off its dividend: New regions did not have to start from scratch, but could reuse many of the basic implementations already present in the CADCAM-INSTRUCTIONS region. We believe that this "ease of reuse" coupled to the shown "ease of extendibility" provides a good illustration of the flexibility of the entire approach - arguably in no small part due to properly thought-through specifications and ramified implementations.

Finally, how does the TSM advance the state of the art in the field of Natural Language Generation? We can summarise its potential contributions as follows.

To begin with, the architecture of the TSM conforms to the recently proposed (and agreed upon) ideal architecture for natural language generation systems [Reiter and Dale, 2000]. The TSM thus not only obtains a certain perspicuity in its functioning, i.e. its interaction with other modules of the integrated system. What is more, precisely because its modules for text planning and sentence planning are implemented in one and the same system as the lexico-grammars, the notorious problem of the "generation gap" can be overcome - no longer is it impossible for text planning to exert control or issue guidance

about how a text should be realised. In fact, the text planning can control the realisation as much as needed in any particular case, as it can impose constraints of any level of delicacy on any of the text plan elements, and these constraints are directly interpretable by the lexico-grammars.

Also, the final prototype TSM enables one to specify *text grammars* using the intuitive tools provided by KPML. These tools allow for a relatively abstract level of specification, thus allowing the user to focus at the task at hand without having to worry about the nitty-gritty of low-level implementations. The sentence planner supports this view. What is more, we have presented in this deliverable various text grammars (i.e. regions) that plan a variety of text types.

Finally, the implementation of the CADCAM-INSTRUCTIONS region illustrates how various approaches to discourse phenomena, i.e. SFG, RST and FGD, can be integrated to handle information structure and discourse relations, thus optimizing coherence and cohesion (or: *contextual reference* in its broadest understanding). We would like to argue that each approach on its own would find it difficult to construct a coherent and sufficiently detailed explanation for the broad range of phenomena considered here (cf. also [Kruijff-Korbayová et al, 2000]. Thus, the synthesis of the three approaches has resulted in a design (or a method) that adds value over and above the contributions made by each individual approach.

References

[MODL2, AGILE 1.2] Richard Power. *Final model of the CAD/CAM domain*. Agile Deliverable 1.2, July 1999.

[SPEC3, AGILE 6.3] Serge Sharoff et al. Agile Deliverable 6.3, June 2000. (Deliverable comprises LSPEC3-Cz, LSPEC3-Ru, LSPEC3-Bu). In preparation.

[TEXS1, 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.)

[TEXS2, 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)

[TEXM2, AGILE 5.2] Geert-Jan Kruijff, Ivana Kruijff-Korbayová and John Bateman. *The Text Structuring Module for the Intermediate Prototype*. AGILE deliverable 5.2, July 1999. (Deliverable comprises TEXM2)

[TEXS3, AGILE 5.3] Tony Hartley, Ivana Kruijff-Korbayová, Geert-Jan Kruijff, Danail Dochev, Ivan Hadjiiliev, Lena Sokolova. *Text Structuring Specification for the Final Prototype.* AGILE deliverable 5.3, January 2000. (Deliverable: comprises TEXS3-Cz, TEXS3-Bu, TEXS3-Ru)

Geert-Jan Kruijff *et al.* 2000 " A Multilingual System for Text Generation in Three Slavic Languages". In *Proceeding of the International Conference on Computational Linguistics COLING2000*. July 31 - August 4, 2000.

Geert-Jan M. Kruijff & Ivana Kruijff-Korbayová. 1999. "Text Structuring in a Multilingual System for Generation of Instructions." In: *Proceedings of the Second Workshop on Text, Speech and Dialogue, Mariánské Lázně, September 1999*. Springer and Verlag.

Geert-Jan M. Kruijff & Ivana Kruijff-Korbayová. 2000. "Aggregation and Contextual Reference in Automatically Generated Instructions." In: *Proceedings of the Third Workshop on Text, Speech and Dialogue, Brno, September 13-16 2000*. Springer and Verlag.

Ivana Kruijff-Korbayová & Geert-Jan M. Kruijff. 1999. "Contextually Appropriate Ordering of Nominal Expressions." In: *Proceedings of the ESSLLI¨"99 Workshop on Generating Nominal Expressions, Utrecht, August 1999*.

Ivana Kruijff-Korbayová, Geert-Jan M. Kruijff & John Bateman. 2000. "Contextually Appropriate Ordering of Nominal Expressions." Contribution to a volume based on the ESSLLLI¨"99 Workshop on Generating Nominal Expressions, Utrecht, August 1999. Kees van Deemter and Rodger Kibble (eds.). Under review.

Ehud Reiter & Robert Dale. 2000. *Building Applied Natural Language Generation Systems*, Cambridge University Press, Cambridge UK

Petr Sgall, Eva Hajičová, & Jarmila Panevová. 1986. *The Meaning of the Sentence in its Semantic and Pragmatic Aspects.* Dordrecht, the Netherlands: Reidel.