AGILE

Automatic Generation of Instructions in Languages of Eastern Europe

---

Title ***Design specification of the user interface for the AGILE final prototype***

Authors Anthony Hartley
Richard Power
Donia Scott
Sergey Varbanov

Deliverable *INTF-2*
Status *Final*
Availability *Public*
Date *30/11/2000*

INCO COPERNICUS PL961104

Abstract:

This deliverable describes the design of the user interface to the Final Prototype of the AGILE system. The interface is designed to support three essential tasks: specifying the content of the texts to be generated by creating a graphical representation, or 'model', of the user task; setting parameters that control linguistic features of the generated output; and viewing the generated texts.

The user interface to the Final Prototype has evolved from the interface to the Intermediate Prototype through several enhancements to its functionality. Improvements include: visual distinction between the different components of a model; vertical presentation of 'steps' versus horizontal display of 'methods'; suppression of nodes in the 'optional' slots; re-design of 'cut', 'copy' and 'paste' operations. New functions include: 'hide' and unhide'; 'mark' and 'unmark'; automatic save of the SPLs with generated text; provision for creating sets of models; display of generated texts as HTML documents.

A special requirement on the AGILE user interface is its localization for Bulgarian, Czech and Russian (as well as for English for demonstration purposes). This is achieved by exploiting Windows-style menus, dialogue panels, messages and context menus, and by controlling Internet Explorer via DDE to view the generated texts.

The deliverable systematically describes the interface procedures for: opening, editing and saving a model; composing a model set; setting initial parameters and text generation options; generating and viewing the generated text.

We report the details of the implementation and some technical difficulties encountered: in some cases the behaviour of the CLIM package distributed with Harlequin LispWorks 4.1 is not as documented; internal details of support for non-English alphabets vary across different versions of Microsoft Windows operating system and are poorly documented.

Table of Content

1. Introduction 6
2. Starting the system 8
3. The user's view of the interface 8
4. Interaction with the knowledge editor 8
4.1 Changing the working language 9
4.2 Opening a new model 9
4.3 Opening an existing model 11
4.4 Editing a model 12
4.4.1 Editing operations 12
4.4.2 An example of task model editing 13
4.5 Saving a model 22
4.6 Composing sets of models 24
4.7 Setting the parameters of the output text(s) 24
4.7.1 Selecting the text type(s) 25
4.7.2 Selecting the text style(s) 25
4.7.3 Selecting the output language(s) 25
4.8 Generating and viewing text 25
4.8.1 Generating text(s) 25
4.8.2 Viewing the generated text(s) 26
4.8.3 Saving the generated texts 27
4.9 Implementation of the knowledge editor 28
References 29

Table of Figures

Figure 1. Model editor interface in Czech language. 7
Figure 2. Model editor interface in Russian language. 7
Figure 3. The Model Display Window. 8
Figure 4. Parameters specified in init-agile.lisp. 9
Figure 5. Opening a new model 10
Figure 6. A new model opened. 10
Figure 7. A model of an empty procedure. 11
Figure 8. Opening an existing model. 11
Figure 9. A model selection dialogue box. 12
Figure 10. An existing model opened. 12
Figure 11. Context menu of all appropriate values. 13
Figure 12. Goal value specified. 14
Figure 13. Context menu listing appropriate graphical objects. 14
Figure 14. Line instance created. 15
Figure 15. List of methods created. 15
Figure 16. An empty method instantiated. 16
Figure 17. Steps concept instantiated. 16
Figure 18. Every step is a procedure. 17
Figure 19. Specifying the actee of the concept 'define'. 18
Figure 20. Copying an existing component. 18
Figure 21. Pasting as reference. 19
Figure 22. 'line a6' referenced. 19
Figure 23. Copying a structure. 20
Figure 24. Pasting as copy. 20
Figure 25. Procedure 'a16' is a copy of 'a10'. 21
Figure 26. Deleting a value. 21
Figure 27. Choosing a value. 22
Figure 28. The model is complete. 22
Figure 29. Saving a model. 23
Figure 30. The Save Model dialogue box. 23
Figure 31. When saved the model changes its name. 24
Figure 32. Select generation options dialogue box. 25
Figure 33. Generating the whole model. 26

Figure 34. Generated text in Bulgarian. 27

1. Introduction

The AGILE system provides an authoring environment for the production of instructional texts in the domain of software tools. We have developed a system that allows a domain expert to construct a model of the procedural parts of manuals for CAD-CAM software, and to produce high quality drafts in any of three selected languages (Czech, Bulgarian and Russian) and five text styles (table of contents, overview, full instructions, short instructions and functional descriptions of the interface objects).

This deliverable describes the design of the user interface to the final prototype of the system. It builds on the interface to the Intermediate Prototype described in deliverable INTF1 (INTF1) and is informed by the results of its evaluation (EVAL1). The IP generated texts corresponding to the instructions for a single user task in one style only (full instructions). In contrast, the FP produces the range of styles above and is thus closer in coverage to a typical commercial manual.

The design of AGILE and its interface is strongly influenced by the DRAFTER (Paris et al., 1995) and GIST (Power and Cavalotto, 1995) systems, which support the authoring of multilingual instructions. The architecture of AGILE follows the standard pipeline organisation for natural language generation systems, with three main modules (Reiter, 1994):

- *Content determination*: the propositional content of the intended text is specified, either directly by a user or by an inference engine; both cases involve selecting information from an existing knowledge base.
- *Sentence planning* (sometimes also referred to as text planning): the structuring of the selected content over textual units such as paragraphs and sentence, and their ordering and expression in the final text. The output of this module is a text plan — a detailed specification of the linguistic and presentational properties of each sentence of the text.
- *Text realisation*: transforming the elements of the text plan into an appropriate surface linguistic form in the selected language(s) and formatting the text according to layout specifications.

The user of AGILE achieves multilingual output by performing *only* content determination. The other necessary tasks, all of which require linguistic expertise, are performed automatically by the system. The user is expected to be an expert in the application domain who may or may not have additional expertise in technical writing, and is required to have a basic proficiency in only one of the 3 supported languages. No further linguistic expertise is required.

As in the DRAFTER and GIST systems, the user interface to AGILE has to support three essential tasks:

- specifying the content,
- setting the parameters for style and language(s)
- viewing the generated texts.

These three tasks are, in turn, enabled by two main components of the system:

- the knowledge editor - which supports content determination

- the text generator - which produces the corresponding texts in the selected language and styles.

A special requirement on the AGILE user interface is that it should be *localised* for the three Eastern European languages (as well as for English, which is included for demonstration purposes only). In practice, this means that a user speaking any *one* of the supported languages should be able to produce texts in *all* of them. Localisation extends to all dialogue boxes, menu titles and menu options as well as to the labels for concepts and relations that are presented in the knowledge editing tool (Figure 1 and Figure 2).

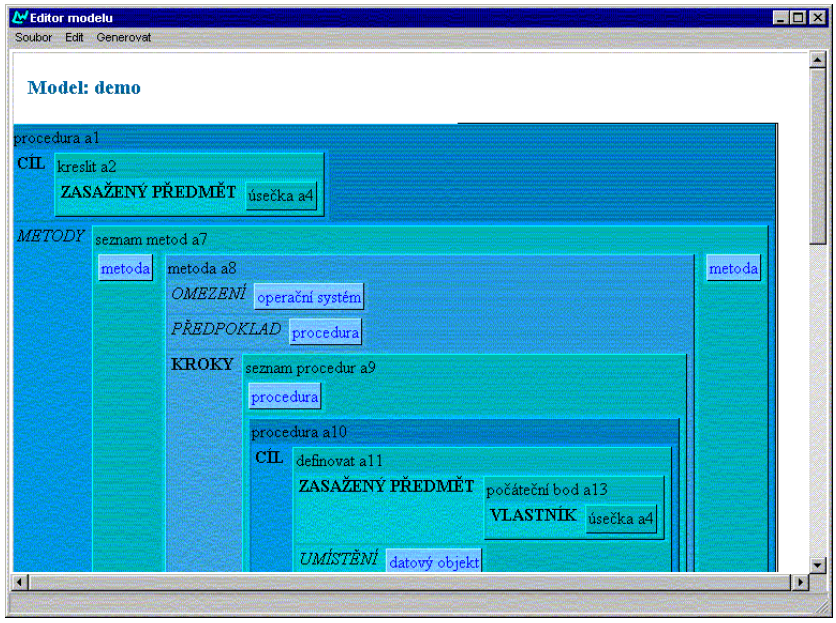

Figure 1. Model editor interface in Czech language.

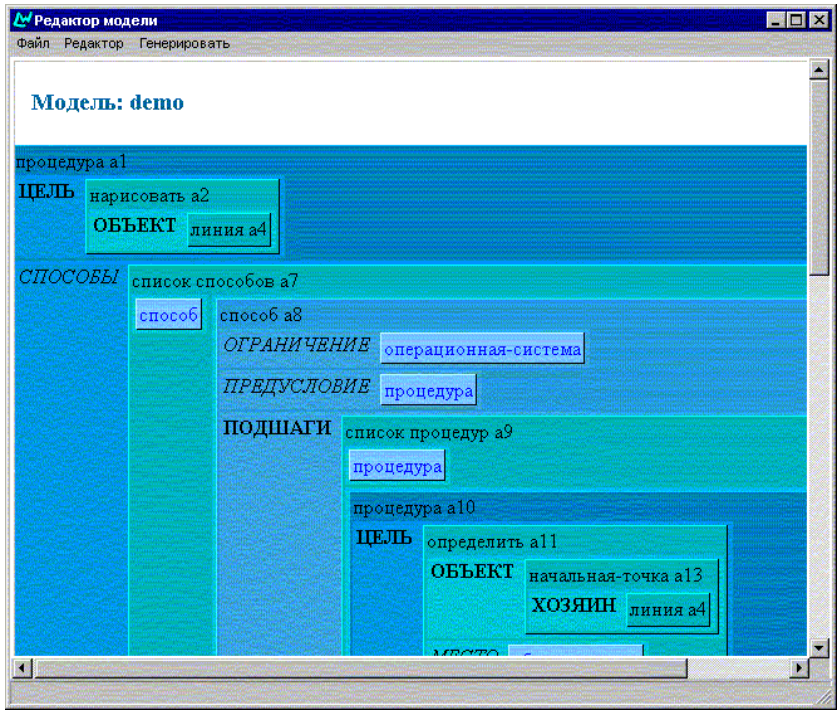

Figure 2. Model editor interface in Russian language.

2. Starting the system

Double-clicking the AGILE icon starts the kernel of the AGILE system, which in turn initiates the loading of the language resources, the text-planning module, and starts the Model Display Window.

3. The user's view of the interface

On starting up the AGILE system, the user is presented with an empty Model Display Window (Figure 3) which allows them to create, edit and save the content specification of the desired text, and to launch the text generation process. Title bars, menus and dialogue boxes[1] are localised to the user's preferred language, which defaults to the language setting of the machine.

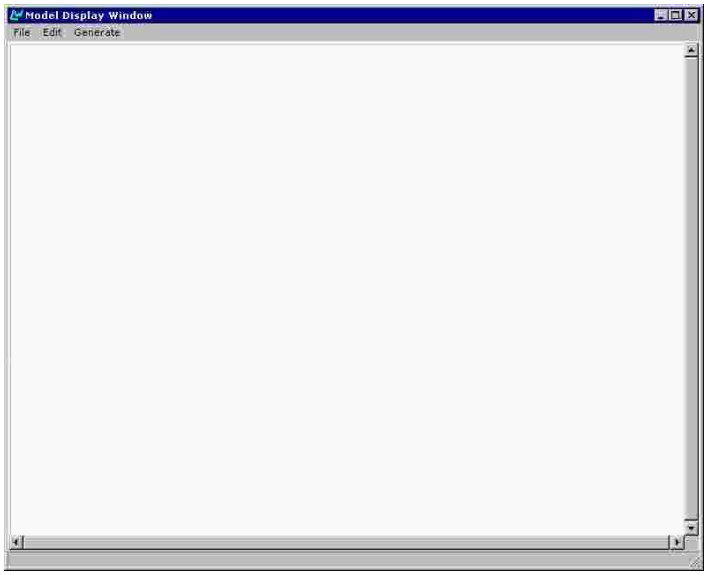

Figure 3. The Model Display Window.

4. Interaction with the knowledge editor

The Final Prototype retains the style of interaction implemented in the Intermediate Prototype. The user determines the desired content of the text through *symbolic authoring* (Power, Scott and Evans, 1998). He or she progressively builds up the assertional content of the domain model via operations on a graphical interface.

The task of knowledge editing depends in an obvious way on the nature of the underlying knowledge representation. As explained in the AGILE deliverable ([MODL1]), we follow LOOM and other languages of the KL-ONE family by making a basic distinction between *terminological* and *assertional* knowledge. The terminology, or 'T-box', defines the concepts from which a set of specific assertions (the 'A-box') can be configured. The task of defining the T-box is the responsibility of the *developers* of AGILE, not the users. It is the developers who specify that a procedure comprises a goal and a method, or that a button is a type of interface object and a possible candidate for a clicking operation. From the user's point of view, the T-box is fixed; the purpose of the knowledge editing tool is to

[1] With exception of the Windows NT dialogue boxes.

exploit the conceptual resources of the T-box in order to build an A-box. In AGILE, an A-box is a set of assertions modelling a procedure for performing some task in a CAD/CAM application.

Knowledge editing in AGILE is a process of progressively extending a model until it is potentially complete. The knowledge editing tool must ensure that the model built by the user conforms to the conceptual definitions in the T-box, and hence that it lies within the scope of the language generators. In order to enforce this compliance, the user is required to begin from a procedure entity for which the goal and method properties are undefined. By clicking on labels signalling these undefined properties, the user obtains a list of suitable values: for instance, a goal can be an action of type *create*, *draw*, *open*, etc. Making a choice from this list introduces a new entity of the specified type. Through the concept definitions in the T-box, further properties will be associated with the new entity — for instance, a *draw* action will have an 'actee' (the entity that is drawn) which must be some kind of *line*.

Since it is intended to support the production of instructional texts, the underlying domain model is essentially procedural. As described in deliverable (MODL1), the main structure of the model is a set of procedures, each consisting of a goal and a method for achieving it. Methods are decomposed into one or more substeps (obligatory), a precondition (optional) and side-effects (optional). These can be specified in any order, but generation cannot occur if any obligatory elements have been left unspecified.

4.1 Changing the working language

Some of the parameters influencing the mode of the work of the AGILE system remain unchanged during a work session with the system. They are specified in a file `init-agile.lisp` located in the root directory of the AGILE system. The user may change their values before every work session. Among these parameters is ***agile-interface-language*** which specifies the working language during a session. Its possible values are :english, :bulgarian, :czech or :russian (Figure 4).

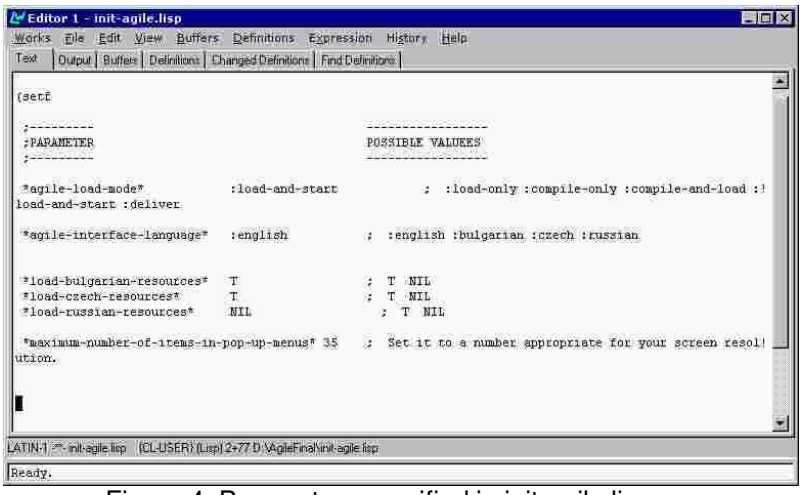

Figure 4. Parameters specified in init-agile.lisp.

4.2 Opening a new model

To open a new model, the user selects "New" from the File Menu (Figure 5). This produces a box labelled "Start" (Figure 6):

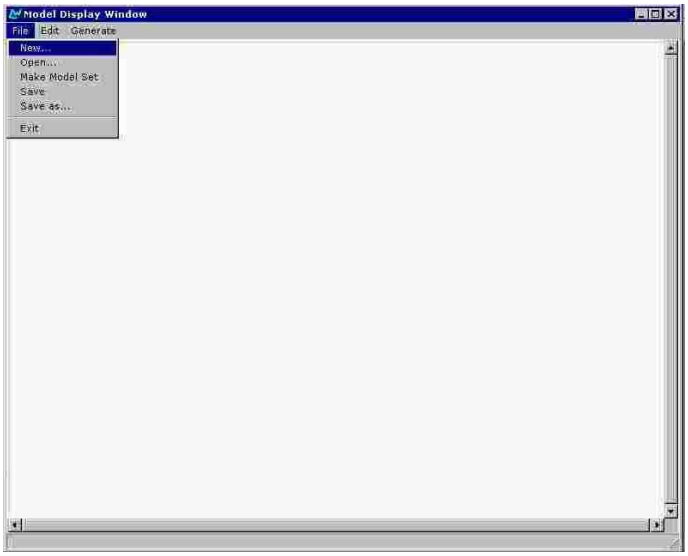

Figure 5. Opening a new model .

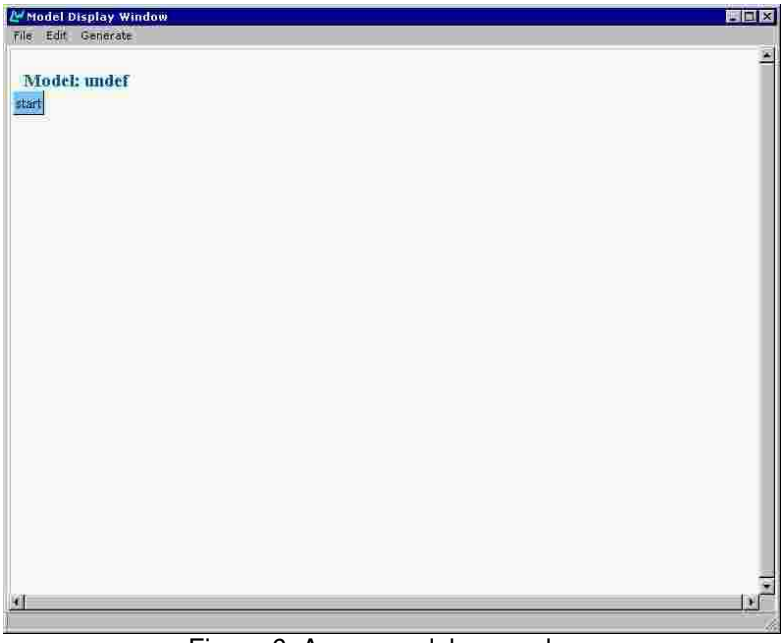

Figure 6. A new model opened.

Clicking on this produces a graphical model of an empty procedure, with fields for its three components: goal, methods and side-effects (Figure 7). The user can then proceed to construct a model by editing this empty procedure.

The name of the model appears above the graphical representation of the model. The default name is 'undef'.

Every component of a model is represented as a pair of name and value (slot). The name is displayed as a label and the value is displayed as a differently coloured box.

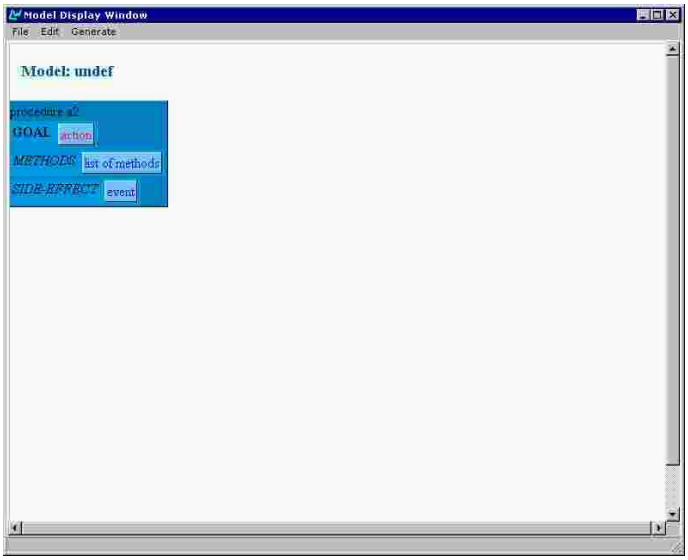

Figure 7. A model of an empty procedure.

4.3 Opening an existing model

AGILE supports users in editing models that have been previously constructed. To open an existing model, the user selects "Open" from the File Menu (Figure 8).

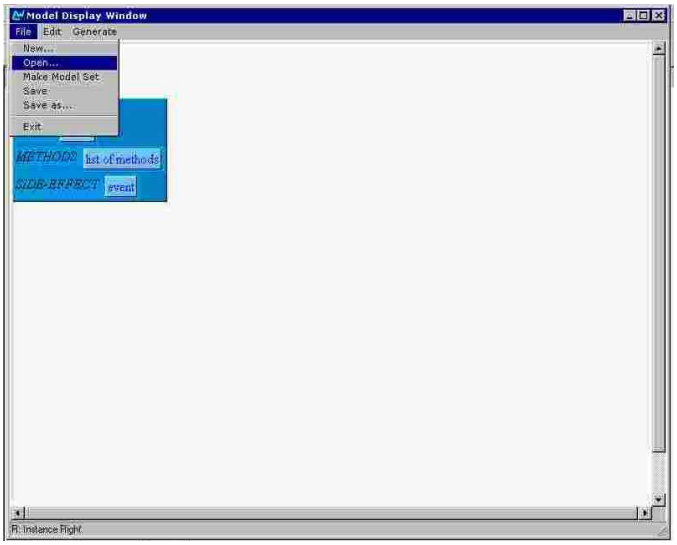

Figure 8. Opening an existing model.

A standard dialogue-box (since it is a dialogue box of MS Windows NT, it is not localised) for browsing the directory structure of the machine is then presented (Figure 9). Using this, the user selects the desired file; the model contained in this file is then presented on the screen (Figure 10).

Figure 9. A model selection dialogue box.

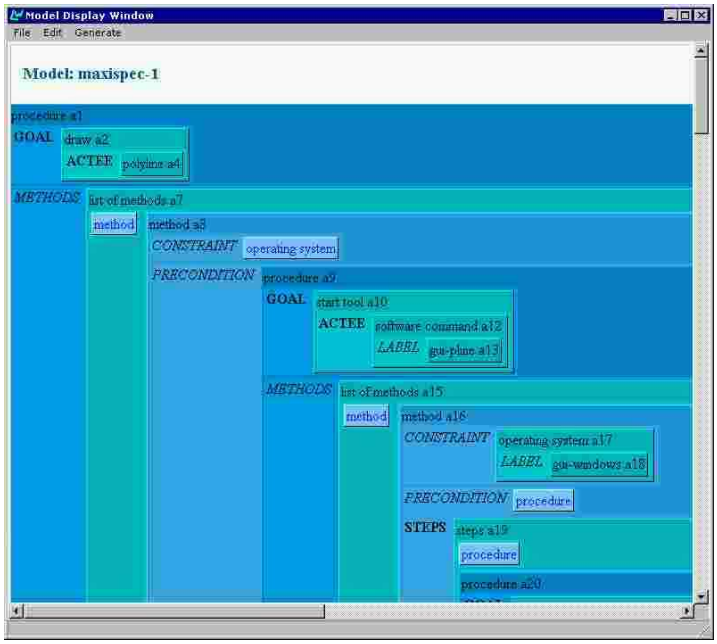

Figure 10. An existing model opened.

4.4 Editing a model

4.4.1 Editing operations

The editing of a model is achieved mostly by opening drop-down menus via left or right mouse clicking on the model components. Left click on an unfilled attribute value opens a menu with all appropriate values listed as options. Right click on a component opens menu providing some typical editing operations:

- Cut – removes the component from the model, but puts it in the editor buffer
- Delete – removes the component from the model
- Copy – puts a reference to the component in the editor buffer

- Paste As Copy (appears if the place clicked is appropriate for insertion) – creates a copy of the object referenced in the buffer and inserts it in the model at the position clicked
- Paste As Reference (appears if the place clicked is appropriate for insertion) – insert the reference from the editor buffer at the position clicked
- Mark – marks the component for generation; this operation works on procedures.
- Unmark – unmarks the component if previously marked.

4.4.2 An example of task model editing

The basic editing techniques are presented and commented in the following example:

Let us assume that the user's goal is to build a model describing the procedure of *drawing a line by specifying its end points*. The user starts a new model from the File menu as described above and clicks on the Start box. (see Figure 5, Figure 6 and Figure 7)

Note that the label 'Goal' is displayed in bold, because filling its slot is obligatory. An additional visual cue is the red colour used for the value of the slot.

The slot 'Goal' is not yet filled and 'action' appearing as its value is only the general concept, indicating the class of concepts which are appropriate.

The user clicks on 'action'. A pull-down menu listing all appropriate domain concepts appears (Figure 11). The user chooses 'draw' and the goal value changes to a box representing an instance of the concept 'draw' (Figure 12). Note the 'a3' identifier. Every new instance is identified by an unique identifier ('a3' in our example). This provides for distinction between different instances of a concept.

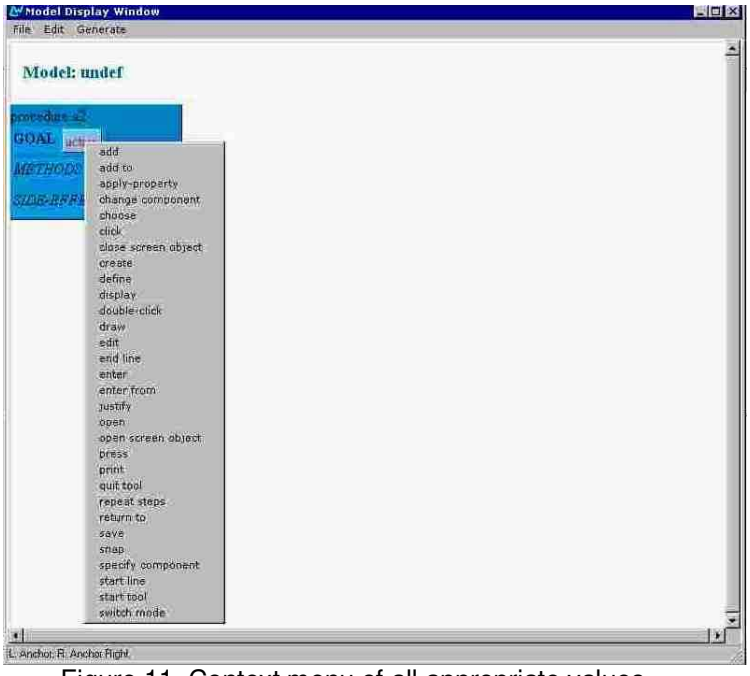

Figure 11. Context menu of all appropriate values.

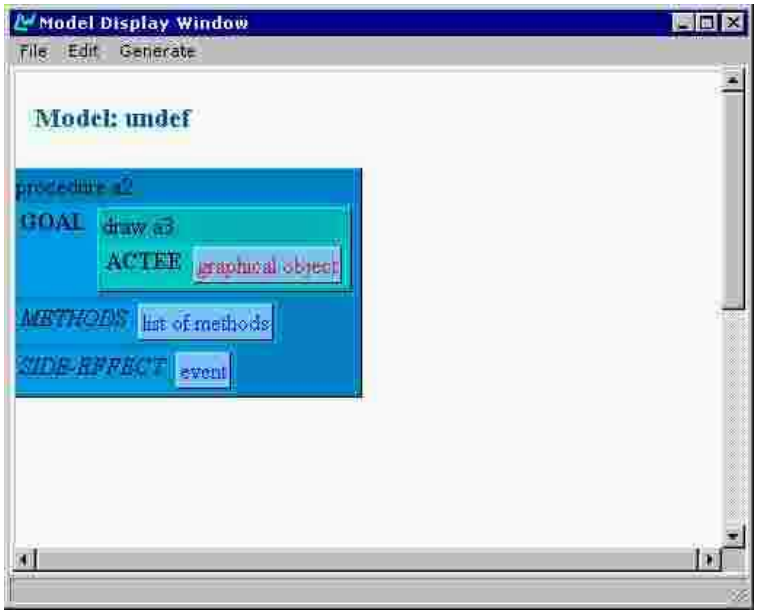

Figure 12. Goal value specified.

The slot 'Actee' is obligatory and the user clicks on its value box. A pull-down menu appears again (Figure 13). The user chooses 'line'.

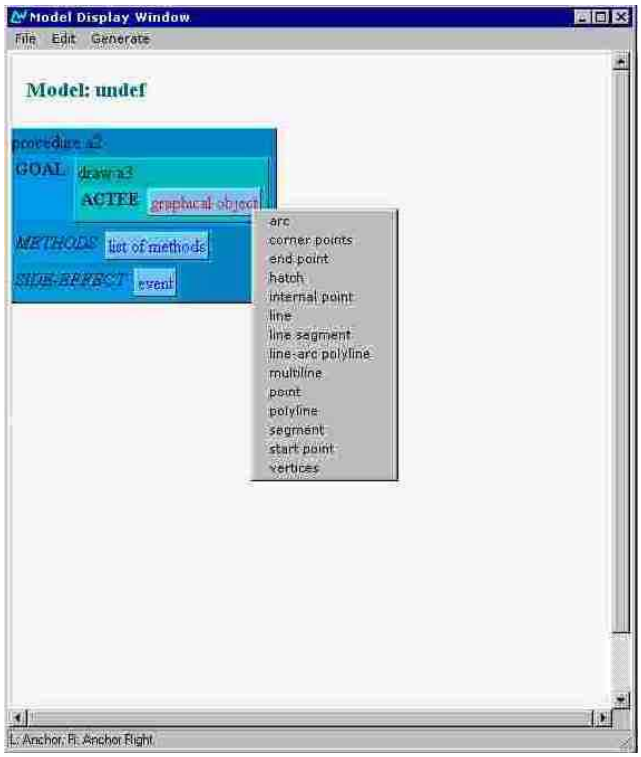

Figure 13. Context menu listing appropriate graphical objects.

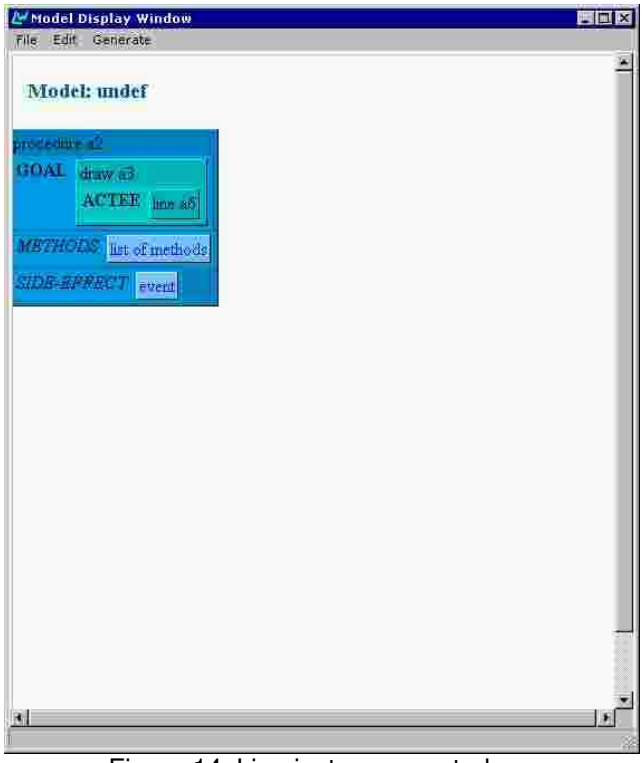

Figure 14. Line instance created.

Then the user starts to specify the methods of drawing a line by clicking on 'list of methods'. Thus a particular list of methods named 'a7' is created and it contains one empty method in the beginning (Figure 15). Clicking on 'method' instantiates the concept and the method structure is unfolded (Figure 16).

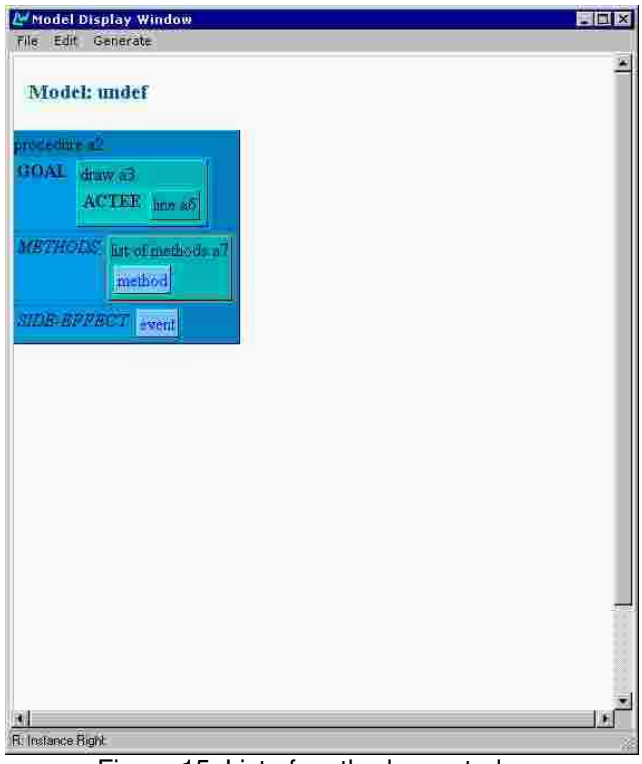

Figure 15. List of methods created.

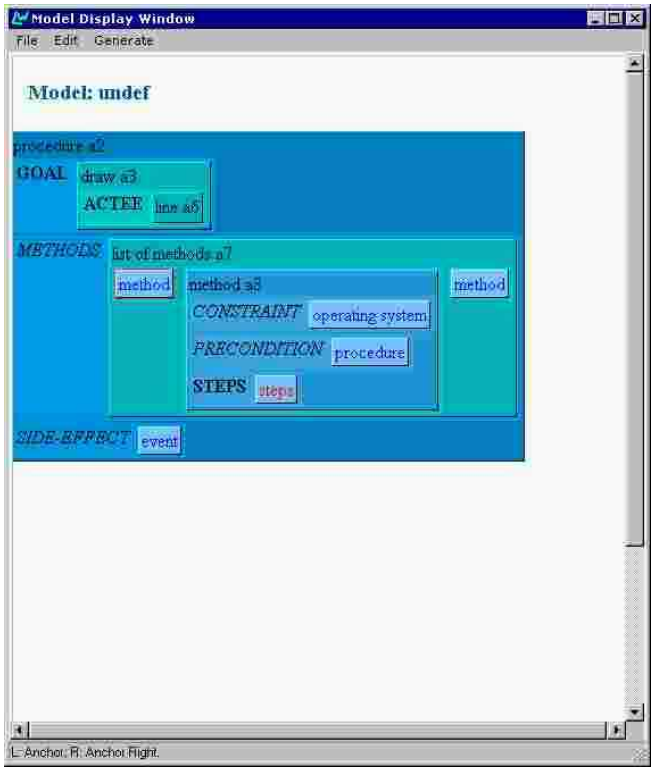

Figure 16. An empty method instantiated.

Note that two ‘method’ boxes appeared left and right to the instantiated method. These allow the user to insert methods at any position in an existing list of methods.

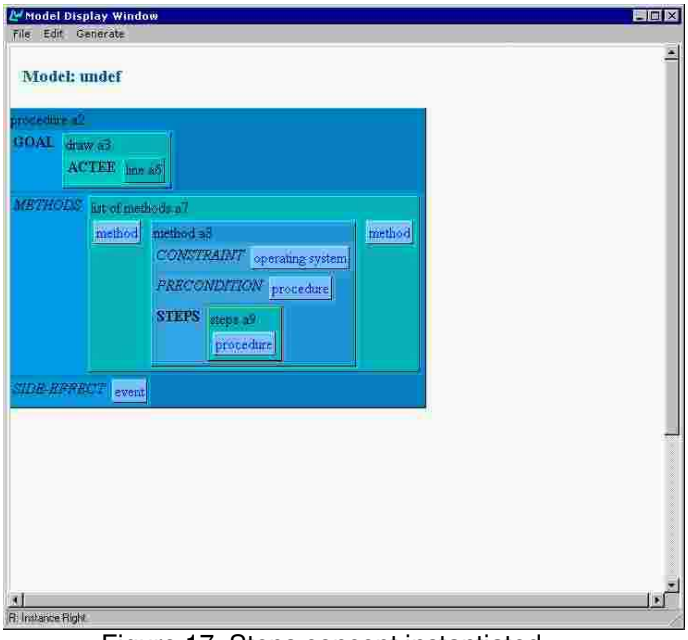

Figure 17. Steps concept instantiated.

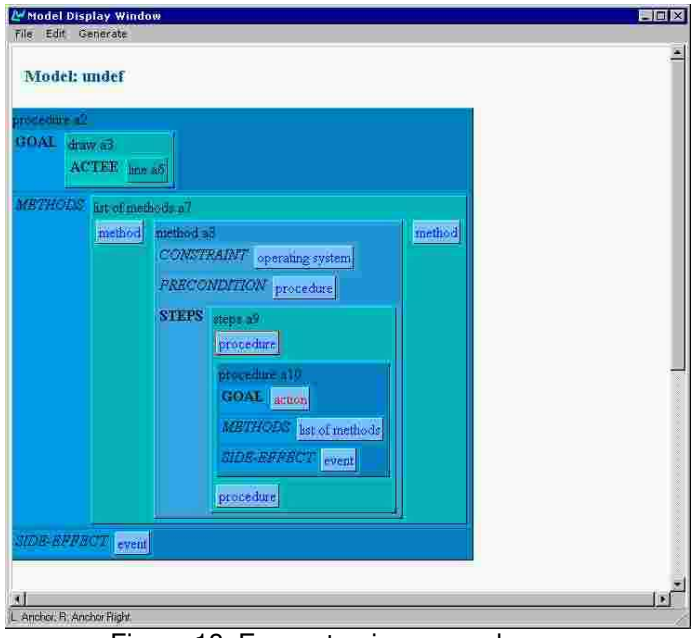

Figure 18. Every step is a procedure.

To proceed with specifying the steps of the method the user clicks on ‘steps’. The concept ‘steps’ is instantiated as object ‘a9’ and an empty procedure appears as its component (Figure 17). Similarly to the situation with methods, when the empty procedure is instantiated (Figure 18) two new empty procedures appear above and below it. Note the vertical layout of the list of procedures, providing for easier visual distinction from the horizontal layout of methods.

Then the user continues by specifying the goal of the new procedure as ‘define’ and its actee as ‘start point’ (Figure 19).

Since the start point, just specified, is a point of the line already instantiated as ‘a6’, the user has to put a reference to ‘a6’ as a value of the ‘owner’ attribute. This is achieved by the following actions:

- right-clicking the ‘line a6’ box (Figure 20) and choosing ‘copy’ puts a reference to the object in the editor’s copy buffer;
- right-clicking the ‘graphical object’ box (Figure 21) and choosing ‘Paste As Reference’ pastes the same reference as a value of ‘owner’ (Figure 22).

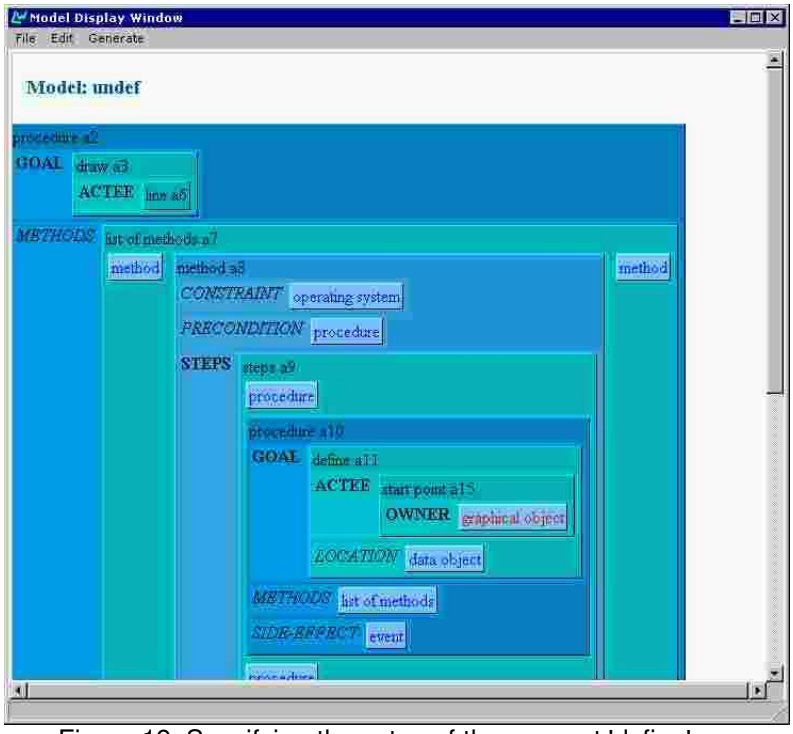

Figure 19. Specifying the actee of the concept 'define'.

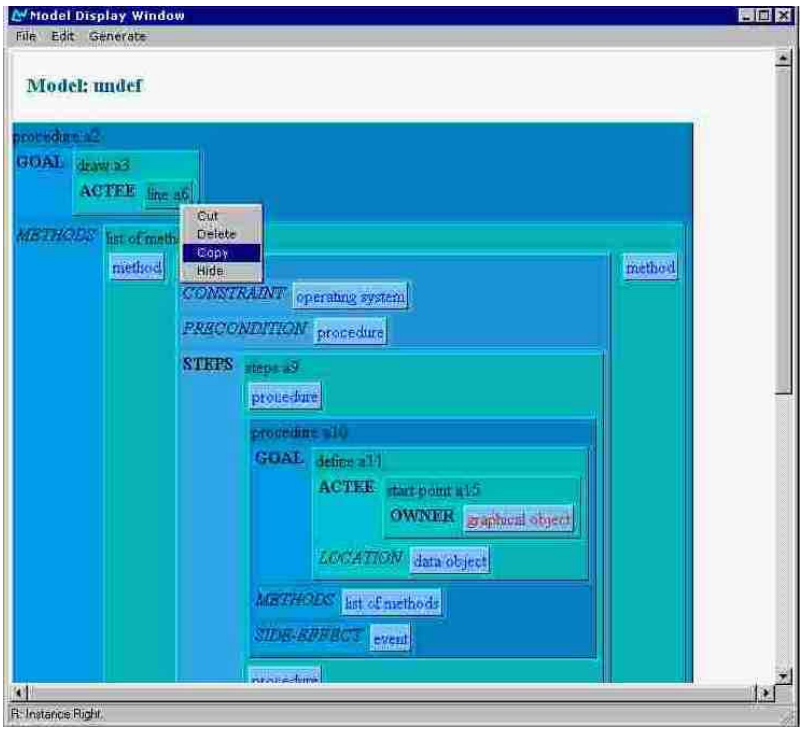

Figure 20. Copying an existing component.

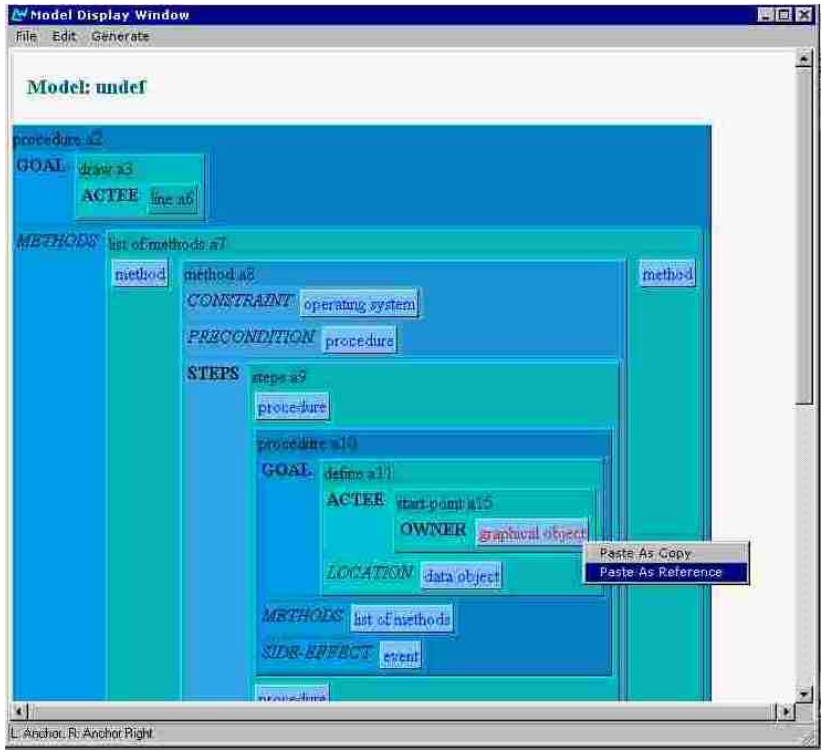

Figure 21. Pasting as reference.

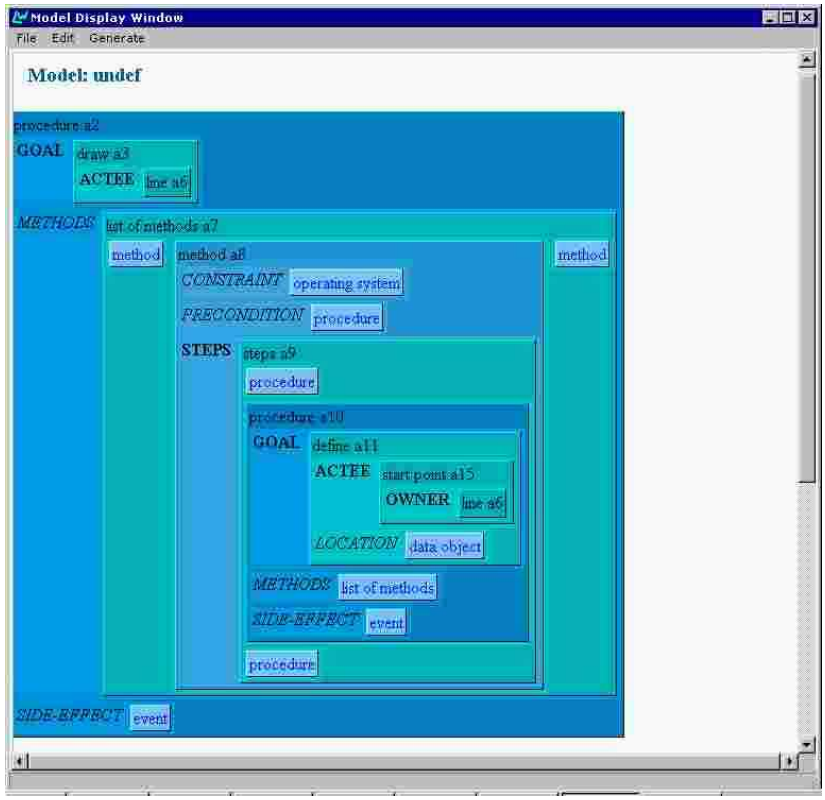

Figure 22. 'line a6' referenced.

The next step in the list of steps has to be defining of the line end-point. The corresponding procedure component could be built from scratch as the previous one, but let us consider an alternative technique in order to demonstrate the pasting of copied data.

So, the user copies the whole ‘procedure a10’ structure (Figure 23) and pastes a copy of it as next step (Figure 24). Note that every instance in the resulting ‘procedure a16’ has new identifier (Figure 25).

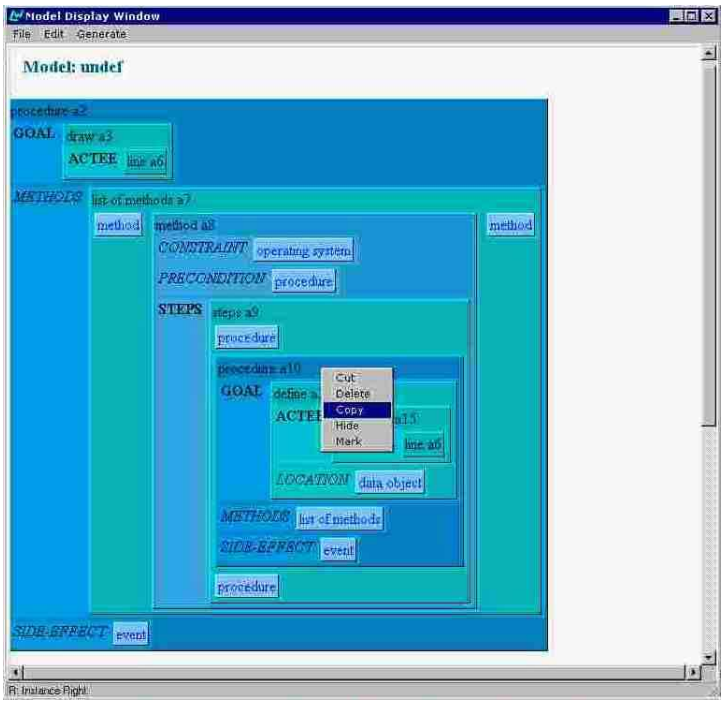

Figure 23. Copying a structure.

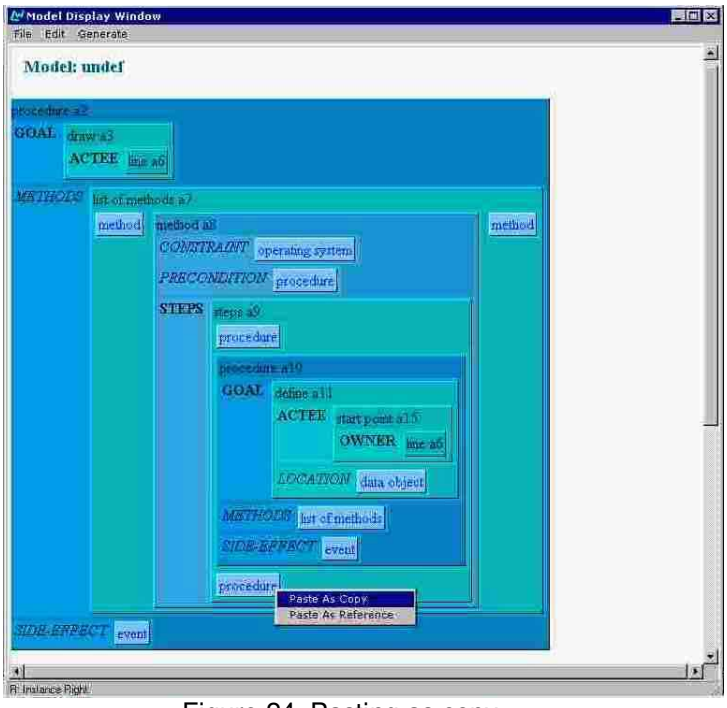

Figure 24. Pasting as copy.

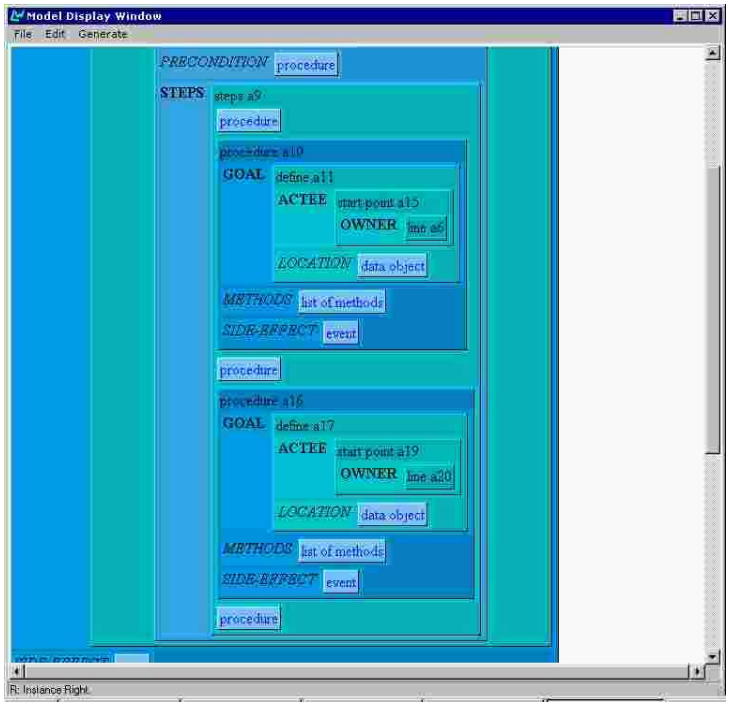

Figure 25. Procedure 'a16' is a copy of 'a10'.

Now the user deletes the ‘start point a19’ component (Figure 26) and creates an ‘end-point’ instance in its place (Figure 27 and Figure 28). Note that copying the reference to ‘line a6’ as owner of the end-point has to be done again.

Deleting components can be achieved by choosing either ‘Delete’ or ‘Cut’ from the context menu. The difference (in the Windows tradition) is that ‘Cut’ puts the object into the editor buffer, thus making its pasting possible.

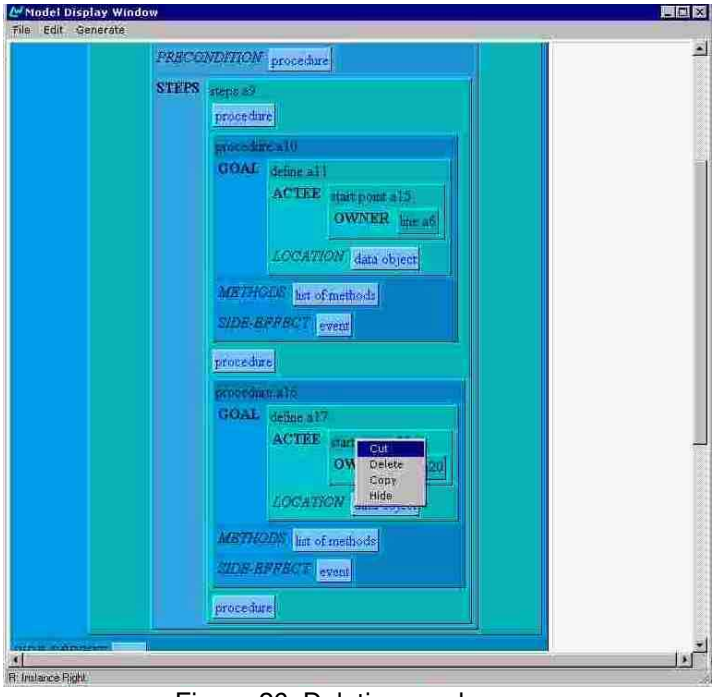

Figure 26. Deleting a value.

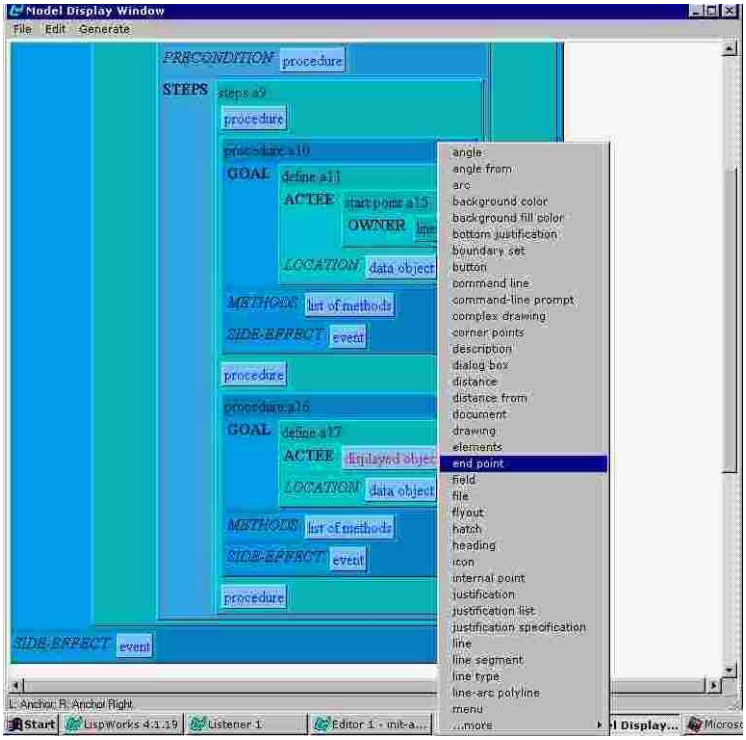

Figure 27. Choosing a value.

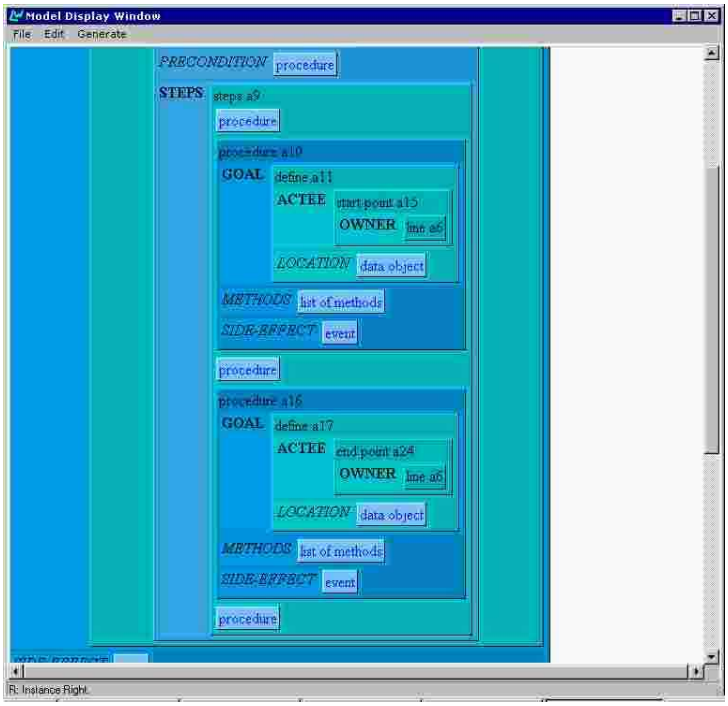

Figure 28. The model is complete.

4.5 Saving a model

The user saves the model by choosing ‘Save’ or ‘Save As’ from the File menu (Figure 29). A standard Windows NT dialogue box appears (Figure 30) where the user specifies the directory and the file name. The system maintains consistency between the model name and

the saved file name. When a model is saved under a new file name, the model name is automatically set to the same name (without the file name extension) (see Figure 31).

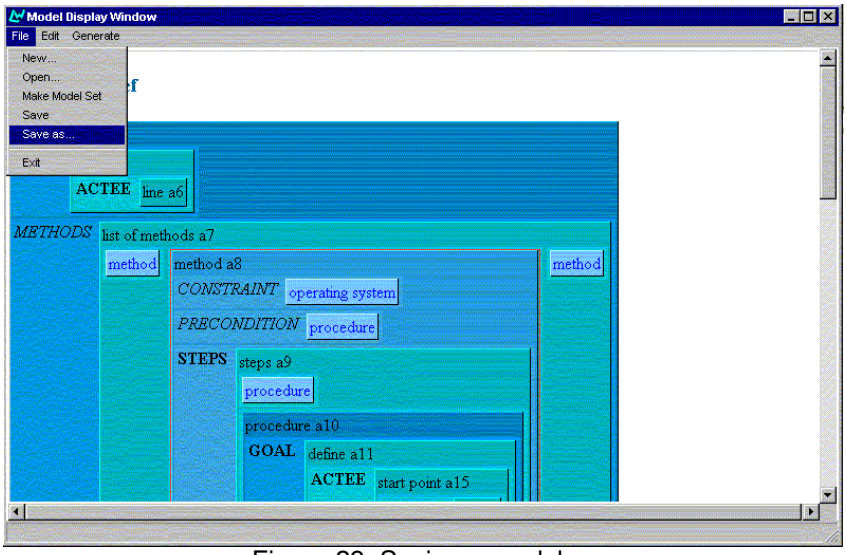

Figure 29. Saving a model.

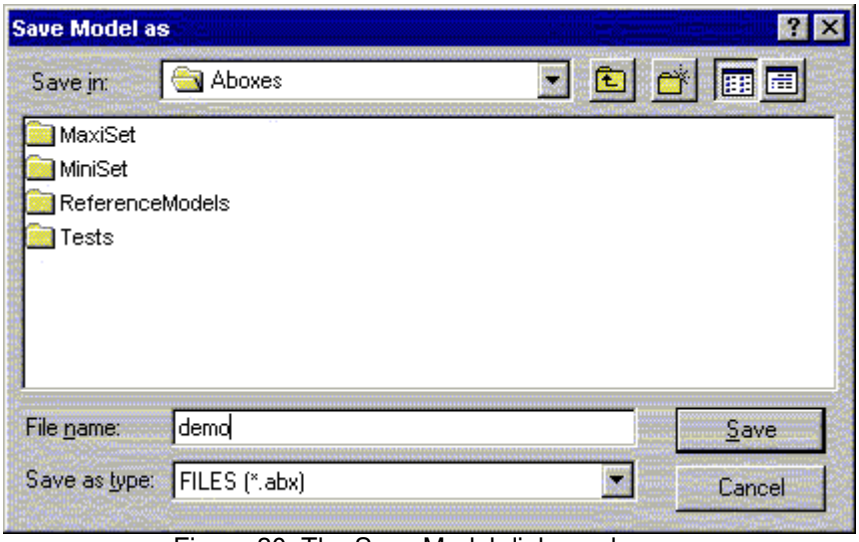

Figure 30. The Save Model dialogue box.

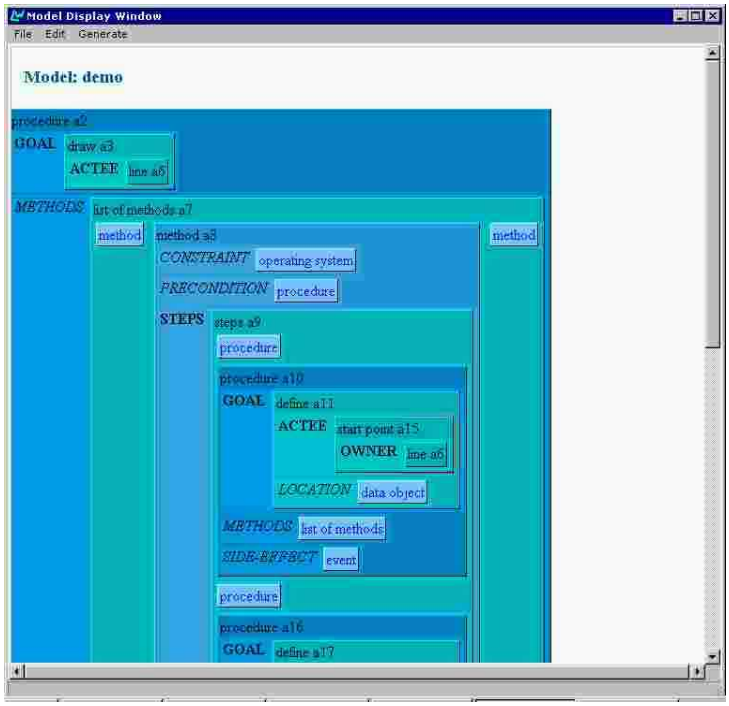

Figure 31. When saved the model changes its name.

4.6 Composing sets of models

A model in the AGILE system represents a procedure with its goal and method(s), but certain text types, namely overviews and functional descriptions, are generated over lists of procedures. The AGILE system interface provides for model sets (TEXS3) composing and editing.

The user composes a set of already created and saved models by choosing 'Make a Model Set' from the File menu. The standard dialogue box for specifying a model to be opened appears. The user selects a model and clicks 'OK'. The dialogue box appears again, giving the user a chance to select the second model, etc. This process continues until the user clicks on 'Cancel'. Then all selected models are composed and displayed in the Model Editor window as a sequence of procedures. The user may then edit the composed model set as 'normal' model and to initiate text generation.

4.7 Setting the parameters of the output text(s)

The user selects "Preferences" from the Generate Menu. A dialogue box presenting all the following options is then presented (Figure 32).

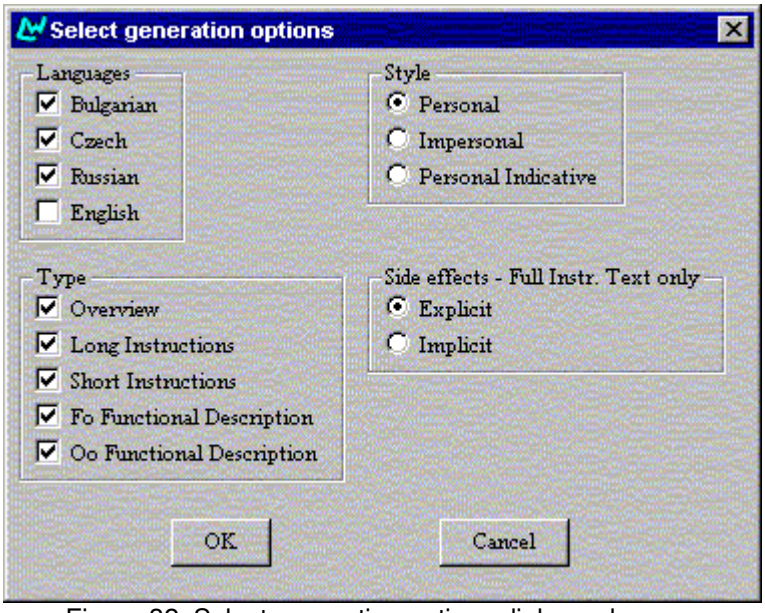

Figure 32. Select generation options dialogue box.

4.7.1 Selecting the text type(s)

The user can select any or all of the five text types (Overview, Long Instructions, Short Instructions, Function-oriented Functional Description, Object-oriented Functional Description) via check boxes.

4.7.2 Selecting the text style(s)

The user can select, via radio buttons, one of the three[2] styles for the text types Long Instructions, Short Instructions, Function-oriented Functional Description, Object-oriented Functional Description. Additionally, the user can select one of the two styles of side-effects for Long Instructions text type. Detailed descriptions of the various style parameters are provided in deliverable (TEXS3).

4.7.3 Selecting the output language(s)

The user selects via tick boxes, any or all of the four[3] output languages.

4.8 Generating and viewing text

4.8.1 Generating text(s)

The user may generate text from selected parts of a model, or from the whole model.

- partial generation (using MARK). While editing a model or a model set the user can mark one or several procedures. The 'Mark' operation appears as an option in the right click context menu, when the clicked object is a procedure (see Figure 23). If there are any marked procedures and the user selects "Marked" from the Generate menu, a new temporary list of all marked procedures is composed and passed to the generation module.

[2] The Personal Indicative style is specific for the Czech language only.

[3] Currently the AGILE System does not support generation in English.

- full generation. The user initiates the text generation over the whole model (or model set) by selecting "All" from the Generate menu (Figure 33).

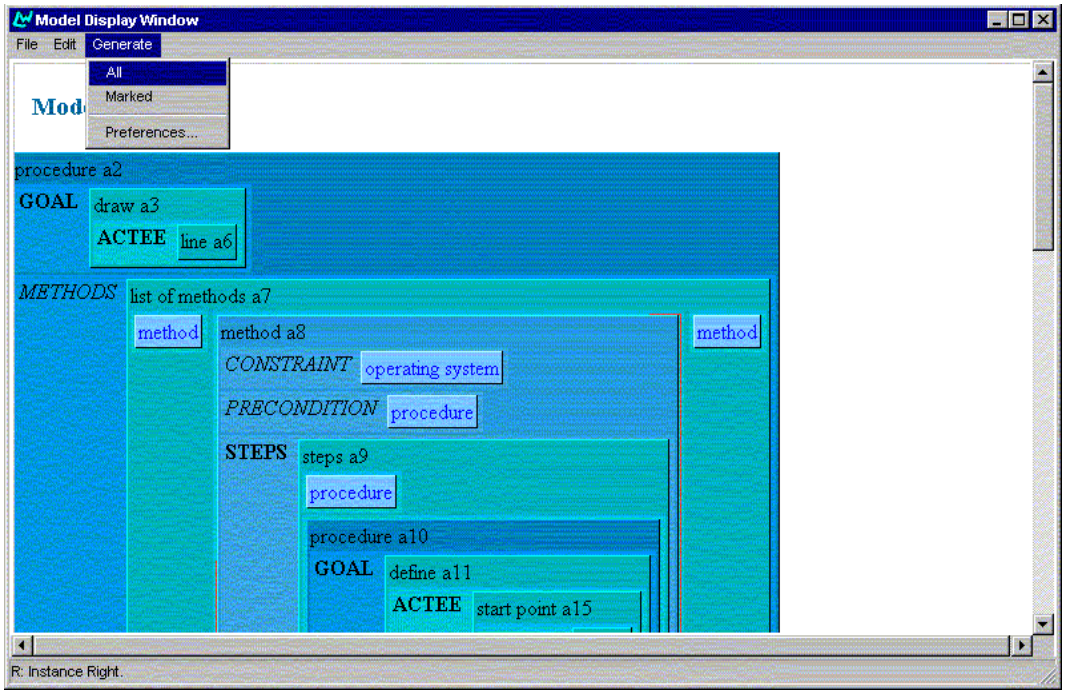

Figure 33. Generating the whole model.

4.8.2 Viewing the generated text(s)

The generation results are visualized in a separate Internet Explorer window for every generation language. Figure 34 shows the result of a generation in Bulgarian, while the working language is English. The window is split into three frames. The top frame presents information about the generation options and a time stamp. It is localized in the working language, while the left and the right frames are localized in the particular generation language. The right frame document presents the generated text. The left frame displays a table of contents, where every item is clickable and is a hyper-reference to the relevant place in the right frame.

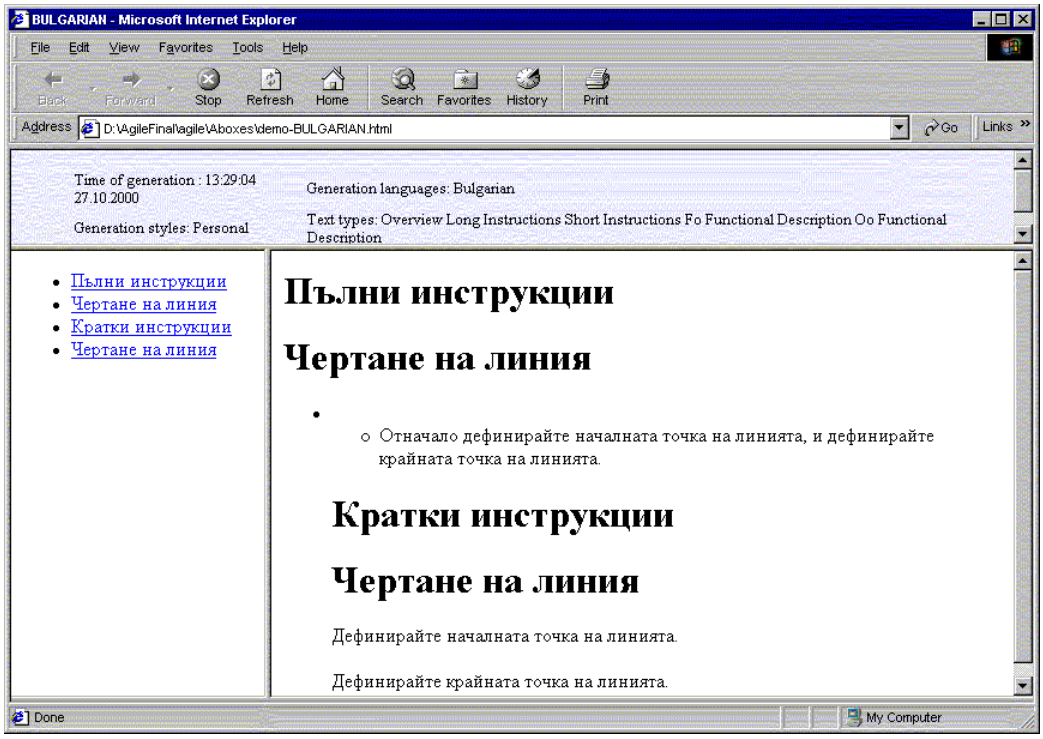

Figure 34. Generated text in Bulgarian.

A separate Internet Explorer window is opened automatically for every generation language and for every new model.

Consider the following example. The user edits a model named "demo" and starts generation in Czech and Russian. The results will be displayed in two windows. Then the user further edits the model and starts generation for Russian and Bulgarian. The result for Russian replaces the previous result in the Russian window and a new window is opened for the Bulgarian. The Czech window remains unchanged. Then the user makes some more changes to the model, saves it under a new name, and starts generation for Czech and Bulgarian. Two new windows open for the new Czech and Bulgarian results because of the changed model name.

The opened Internet Explorer windows are never closed automatically. The user has to close them manually if and when he/she wants. The reason behind this behaviour is that the user might want to inspect the generated texts for a longer time and to compare the results obtained from different versions of the model.

4.8.3 Saving the generated texts

In the process of generation, the text generation module produces texts formatted with HTML tags. The interface module composes these texts and wraps them in the appropriate HTML tags to obtain complete HTML documents for every generation language. The interface module then generates the table of contents HTML document (displayed in the left frame), the information HTML document (top frame) and the HTML document defining the frame set layout.

All these documents are automatically saved as HTML files in the directory where the model is saved. The file names are composed from the model name, generation language and the text style option used.

For instance, the generated text document (right frame) in the previous example (Figure 34) is saved as `demo-BULGARIAN-PERSONAL-text.html`. Such organization

enables further editing and combining the generated texts into larger HTML documents by means of an HTML authoring tool.

4.9 Implementation of the knowledge editor

The knowledge editor is programmed in Harlequin LispWorks 4.1 with the extensive use of CLIM (Common Lisp Interface Manager). Runs on Windows NT.

The major difficulties encountered while programming the knowledge editor were related to the following facts:

- The CLIM provides for the interface code portability, but the CLIM package distributed with the Harlequin LispWorks 4.1 appeared fragile. When certain constructions are not used in a strict object-oriented manner, their behaviour is unpredictable. There was a case when a CLIM construction did not function as documented. We contacted the Harlequin support team and they prepared and sent us specific patches solving the problem.
- The DDE (which is the protocol we use to communicate the Microsoft Internet Explorer from within lisp programs) documentation is not very detailed. For example, a problem we encountered was that when a HTML file is displayed, then updated and a DDE request for displaying again the file is sent to the Internet Explorer, the window's contents do not refresh, despite including HTML tags prohibiting the use of a cached copy. Since we could not find a solution to this problem in the Internet Explorer DDE documentation, currently the interface module of the AGILE system generates and includes in the generated HTML files relatively complex JavaScript code, which recognizes this situation and forces window refreshment.
- The inner details of supporting non-English alphabets differ (and are poorly documented) through the versions of Microsoft Windows operation system. This is the only reason for restricting the use of the AGILE system to Windows NT.

References

Paris, C.K. Vander Linden, M. Fischer, A. Hartley, L. Pemberton, R. Power and D. Scott (1995) A Support Tool for Writing Multilingual Instructions. Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI '95), Montreal, Canada, pp. 1398-1404.

Power, R. and Cavalotto, N. (1996) "Multilingual generation of administative forms" in Proceedings of the 8th International Natural Language Generation Workshop, Herstmonceux.

Reiter, E. (1994) Has a consensus generation architecture appeared and is it psycho-linguistically plausible? INLG '94, pp. 163-170.

AGILE deliverables referred to in the present deliverable:

[EVAL1] Serge Sharoff, Donia Scott, Anthony Hartley, Danail Dochev, Jiri Hana, Martin Cmejrek, Ivana Kruijff-Korbayova, Geert-Jan Kruijff (2000) Evaluation of the intermediate prototype. Deliverable EVAL1 of AGILE project PL961104.

[INTF1] Stancho Stankov, Richard Power, Tony Hartley (1999) Design specification for the interface to the immediate demonstrator. Deliverable INTF1 of AGILE project PL981104.

[MODL1] Richard Power (1998) Preliminary model of the CAD/CAM domain. Deliverable MODL1 of AGILE project PL961104.

[TEXS3] Tony Hartley, Ivana Kruijff-Korbayova, Geert-Jan Kruijff, Danail Dochev, Ivan Hadjiiliev, Lena Sokolova (2000) Text Structuring Specification for the Final Prototype. Deliverable TEXS3 of AGILE project PL961104.