Mjolner Integrated Development Tool − Reference Manual

Mjølner Informatics Report
MIA 91−34
March 2004

Table of Contents

Table of Contents

Table of Contents

Table of Contents

1 How to Get Started

The source browser that is part of Mjolner, is activated by writing one of the following (UNIX):

1. mjolner
2. mjolner myFragmentGroup
3. mjolner myFragmentGroup.bet
4. mjolner myFragmentGroup.ast

In each case a source browser window appears on the screen.

1) If there are no arguments to Mjolner the project list pane will show the users home directory and possible projects defined in ~/.ymer.pjt (See Project definitions in this manual)

2)−4) If the fragment group only has one fragment form, it will a will automatically be opened in the code viewer pane see below.

The structural representation of programs is abstract syntax trees (ASTs), that are stored on .ast files (or .astL files on the PC). The relations between the textual form and the structural form are handled in the following way: if the text file (e.g. the .bet file) is newer than the .ast file, theuser is asked whether the text file should be parsed automatically. If there are parse errors these can be corrected in a parse error editor.

2 Basic User Interface Principles

2.1 The Mouse

The Selection Button is used to select arguments, to click on command buttons, to double−click and to select commands in all menus but one: the pop−up menu.

The Pop−up Menu Button is used to activate the pop−up menu, that is used during structure editing. The menu pops up when the mouse pointer is located in the codeviewer/editor window and the Pop−up Menu Button is pressed.

2.1.1 UNIX

On UNIX systems the left mouse button is used as the Selection Button and the right most mouse button is used as the Pop−up Menu Button. The middle mouse button has currently no use.

2.1.2 Windows NT/95

On Windows NT/95 systems the left mouse button is used as the Selection Button and the right most mouse button is used as the Pop−up Menu Button. The middle mouse button has currently no use.

2.1.3 Macintosh

On the Macintosh the mouse button is used as the Selection Button and Command−Click is used as the Pop−up Menu Button.

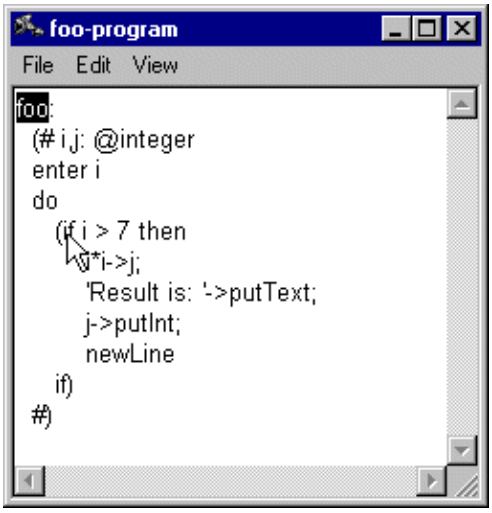
2.2 Selection

Selection is important because most operations in Mjolner are performed in relation to the current selection, which will always be marked in reverse video. There are different ways of selecting parts of a fragment.

When you position the cursor somewhere in the Code editor and click the Selection Button the nearest surrounding structure will be selected and marked as the current selection.

Some examples of how to select:

To select a name:
just place the cursor in the name and click the Selection Button.
To select a complete structure containing keywords :
place the cursor at one of the keywords and click the Selection Button:



Then the resulting current selection becomes:

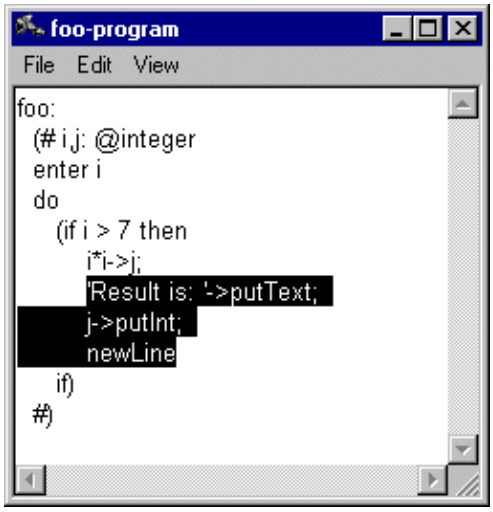
To select all elements in a list:

place the cursor at one of the list separators e.g. a ';', and click the Selection Button.

To select a sublist:

If a sublist of elements is going to be the current selection, dragging selection is a convenient technique. Dragging selection means clicking with the Selection Button at the starting point of the sublist and keeping the button down while moving the mouse to the ending point of the sublist.

At this point the button is released and the current selection will be set to the selected sublist:



In general the smallest complete enclosing structure will be selected.

3 Source Browser

The source browser makes it possible to browse in projects. A project can be a collection of files, a file directory or parts of the dependency graph of a BETA program. Links in the fragment structure like ORIGIN and INCLUDE can be followed easily.

3.1 Source Browser Windows

We will here present the most important user interface components of the source browser: windows, menus and other interaction possibilities in the following sections.

The source browser interface consists of a number of different window types:

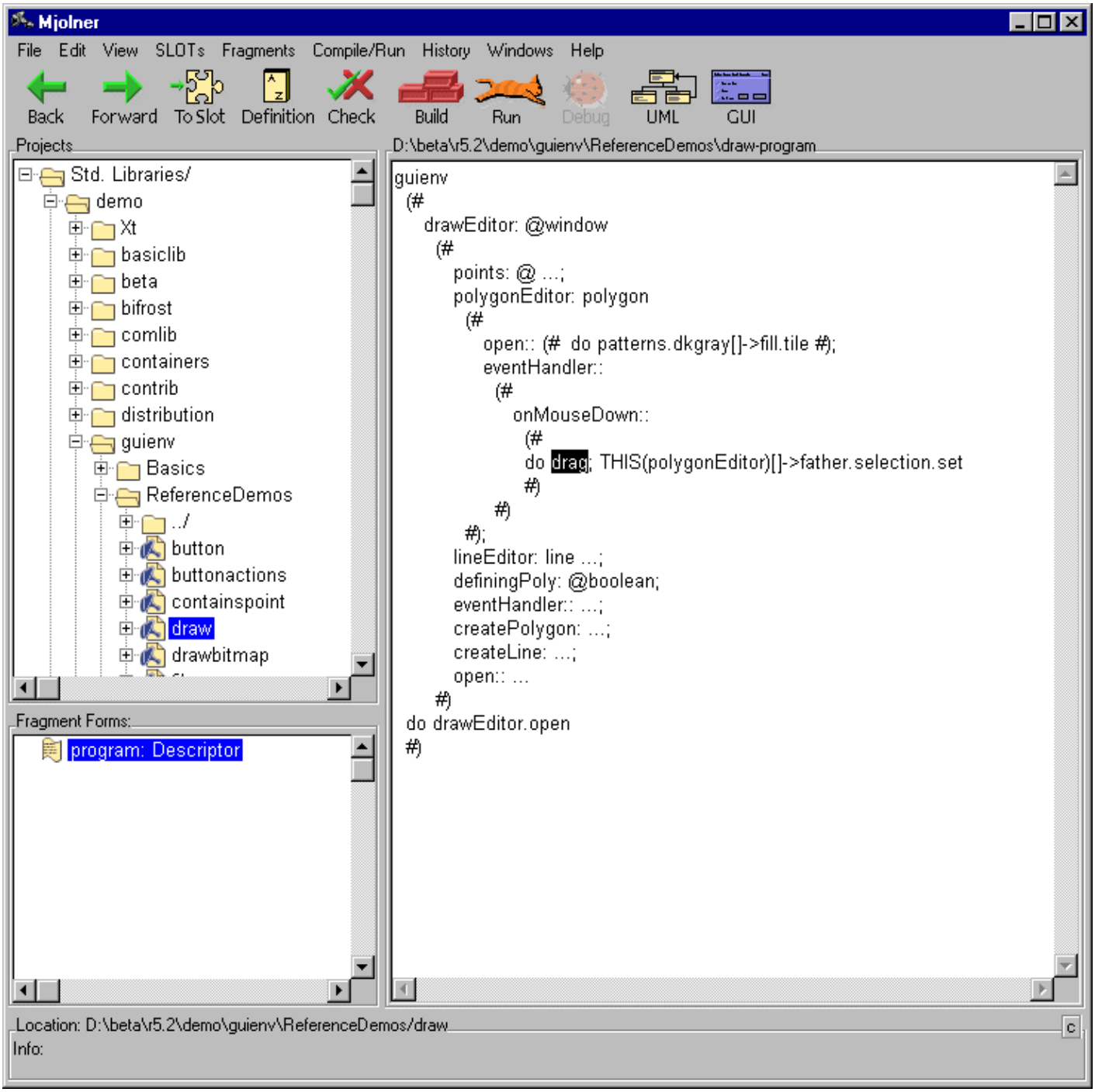
- browser window
- workspace window
- separate code viewer window
- subviewer window
- parse error editor window
- two types of semantic errors windows

where the browser window is the main window, giving access to the main browsing facilities and to the other windows. These windows will be described in the following.

3.2 Browser Window

The browser window appears like:

Figure 1: Browser window



- The upper left pane is called the Projects pane, it contains a tree view of projects. A project can contain a list of folders or files.
- The lower left pane contains a list fragment forms of the fragment group selected in the Projects pane. It is called a fragment group viewer/editor or just group viewer/editor.
- The right−hand side pane displays the BETA code of the fragment form selected in the group viewer. It is called the code viewer/editor [1]. This code viewer will be seen in many different windows and the functionalities of the viewer is described later.

Below the code viewer/editor pane there are two info panes. Location displays path of the selected fragment group in the group list pane − or in the project list pane, if nothing is selected in the group list pane.

Info displays information, such as "no fragments in the selected project"' or what the source browser currently is doing.

Browsing can be done at basically 3 levels: the project level, the group level or at the code level.

3.2.1 Menus

The following sections describe the browser window menus offered by the source browser. The other subtools supply additional menus.

3.2.2 File Menu

New...
: Makes it possible to create a new fragment group with one fragment form. You can choose between the currently available grammars. A dialog pops up.

New BETA Program...
: Creates a minimal BETA program. You are prompted for the name and location of the new fragment group.

New BETA Library...
: Creates a minimal BETA library. You are prompted for the name and location of the new fragment group.

Open...
: Opens a standard file dialog, and you can now select (a) a project−file (must have extension .pjt), (b) a directory, and (c) a .bet/.ast/.astL file. You will then in the project list pane find a project of the corresponding type A, B, or C (see Project Definitions). All other selections will be ignored.

Open Dump File...
: If your program crashes, while you are not running it under the the debugger in the Mjolner Tool, the dump file can be opened and the different activation levels can be browsed. The source code browser subwindow will for each activation level highlight the statement, that was executed at dump time.

Save
: If changes have been made to the fragment group, it is saved. A backup of the old text file (if existent) is taken on e.g. foo.bet~ and a new text file is generated e.g. foo.bet.

Save As...
: Saves the current fragment group under a another name. You are prompted for the name and location of the new fragment group. The editor continues with the new fragment group.

Save All...
: Calls Save on all open group editors.

Save Abstract...
: Saves the current fragment group on textual form at the current abstraction level. You are prompted for the name and location of the text file.

Recover
: Recovers to the latest auto save file (.ast# or .astL# on PC). This command is enabled if the .ast# exists and is newer than the .ast file.

Revert
: Reverts any changes (since the latest save) made to the current fragment group.

Print
: Print the contents of the codeviewer/editor.

Group Write Protection
: If checked, it is not possible to modify the fragment group. Group write protection is set automatically if there is no write permission on the fragment group file or if the global write protection is set.

Global Write Protection
: If checked the whole editor is in read−only mode.

Preferences...
: A preferences window pops up.

Save settings
Enables you to save the current contents of the project list pane on a project−file $USER/.ymer.pjt. This project is then automatically loaded next time you invoke a source browser application − in this way you can save your configuration. It also saves/creates another file, $USER/.ymer.rc containing information on the locations of the BETA compiler, and other tools in the Tools menu (only in the stand−alone source browser application Ymer).

Reset settings
Enables you to clear the project list pane and reset it to the last settings (or the default settings, if no $USER/.ymer.[pjt|rc] files are found.

Quit
Closes all browser windows, and terminates the application or . In both Close and Quit source browser applications will examine if any files have been changed, and prompt for whether these files should be saved before closing/quitting.

3.2.3 View Menu

One important quality of Sif is the possibility to present documents at different abstraction levels. Abstract presentation gives a good overview over a document by suppressing irrelevant details. Abstract presentation is provided by means of contractions. A contraction is a special presentation of an sub−AST, where only the root is presented. See also Abstract Presentation in Source Browser Tutorial. If the construct has an associated comment, the comment marker is included in the contraction. Browsing in the document can be done by detailing, i.e. opening the contractions selectively. This is done either by the command Detail or by double−clicking. Abstract presentation can be used to produce documentation such as a interface specifications by saving the document on textual form at different abstraction levels, using the Save Abstract... command.

Abstract
Contracts the current selection, i.e. presents the current selection as a contraction instead of the full "text".

Abstract Recursively
Traverses the current selection recursively and contracts those constructs that have a certain syntactic category. This command has only effect if abstraction levels have been specified in the grammar for the particular language. The grammar may specify a list of syntactic categories that automatically shall be contracted. For BETA the syntactic categories are Descriptor, Attributes, and Imperatives, when opening the files in the viewer/editor. For more information on specifying abstraction levels see [MIA 91−14].

Overview
Like Abstract Recursively but the current selection is reestablished after the Abstract Recursively command, i.e. the path from the root to the current selection is detailed. This command is useful if you want to see the context of the current selection.

Detail
If the current selection is a contraction, this operation opens the contraction, i.e. presents the next level of text. A contraction can also be opened by double−clicking on it. If the current selection is not a contraction any visible contraction in the current selection will be opened.

Detail Recursively
Recursively details the contractions in the current selection.

Follow Semantic Link
During the checking phase a number of semantic links are inserted in the AST. These links are available to the user. Using this command on a name application the semantic link will be followed and the current selection is changed to the appropriate node in the current or another fragment form. Double−click can also be used to activate this command.

Follow Link to Fragment

Given a SLOT definition, a binding of the definition (i.e. a fragment form with the same name and type) can be searched for. This is done by selecting the SLOT definition and using this command. The editor will now search for a fragment form with the same name in the BODY and MDBODY hierarchy of the group that this fragment is part of. Notice that the binding of Attributes SLOTs may only be found in this way if they are bound in the BODY and MDBODY hierarchy. A shortcut or double−click can also be used to activate this command.

Follow Link to SLOT
The editor searches after a SLOT definition with the same name as the current fragment form along the ORIGIN chain until it finds it or reaches the top, i.e. betaenv.

Zoom In
Changes the contents of the code viewer to the current selection, i.e. the root of the AST shown in the editor (the editor root) is changed to the sub−AST, that corresponds to the current selection.

Zoom Out
Extends the contents of the code viewer to show one level more of the structure, i.e. the editor root is changed to its father.

Zoom To Full Editor
Extends the contents of the code viewer to show the whole fragment form, i.e. the editor root is set to the whole AST of the fragment form.

Reprettyprint
Rebuilds the textual presentation of the AST and reestablishes the current selection. This command can be used if the contents of the code viewer need to be refreshed.

Format...
Only Win 95/NT. The font in the current code editor can be changed using this command.

Adaptive prettyprinting
Adaptive prettyprinting can be turned on an off. Default is off.

Show AST Dump
This command is useful for users of the metaprogramming system. It prints in the console window a dump of the sub−AST that corresponds to the current selection

3.2.4 Following links to other fragment forms

If the destination fragment form is different from the source fragment form the destination fragment form is shown in the same code viewer or in a separate code viewer, depending on the code viewer of the source fragment form. If the code viewer of the source fragment form is

- part of the browser window, the contents of the code viewer window is changed
- a separate code viewer, the contents of the code viewer window is changed
- a subviewer, a new subviewer is opened
- part of a work space window, a new code viewer is inserted in the work space window.

3.2.5 Navigation and Edit Pop−up Menu

When using the Pop−up Menu Button on an a construct that is not a nonterminal a popup menu, containing some of the most commonly used commands, is opened.

Back
Shortcut to History:Back

Forward
Shortcut to History:Forward

Follow −> Semantic Link
Shortcut to View:Follow Semantic Link

Follow −> Semantic Link (Separate Window)

Shortcut to View:Follow Semantic Link. The destination node is opened in a separate window.

Follow −> Link To Fragment
Shortcut to View:Follow Link to Fragment

Follow −> Link To Fragment (Separate Window)
Shortcut to View:Follow Link to Fragment. The destination node is opened in a separate window.

Follow −> Link to SLOT
Shortcut to View:Follow Link to SLOT

Follow −> Link to SLOT (Separate Window)
Shortcut to View:Follow Link to SLOT. The destination node is opened in a separate window.

Open Separate Editor
Shortcut to View:Open Separate Editor

Open Subeditor
Shortcut to View:Open Subeditor

Show UML Diagram
Shortcut to Toolbar:Show UML Class Diagram

Open GUI Editor
Shortcut to Toolbar:Open GUI Editor

Edit −> Cut
Shortcut to Edit:Cut

Edit −> Copy
Shortcut to Edit:Copy

Edit −> Paste
Shortcut to Edit:Paste

Edit −> Paste Before
Shortcut to Edit:Paste Before

Edit −> Paste After
Shortcut to Edit:Paste After

Edit −> Insert Before
Shortcut to Edit:Insert Before

Edit −> Insert After
Shortcut to Edit:Insert After

Reprettyprint
Shortcut to View:Reprettyprint

3.2.6 History menu

Back and Forward moves you along the path you have previously browsed.

Furthermore, this menu contains one entry for each fragment form, you have displayed in the code viewer. This is used for speedy access to previously visited fragment forms.

3.2.7 Windows menu

Open Workspace
Opens a workspace window (see later more on this).

Open Separate Browser
Opens a new browser window, identical to this one. This window is controlled by the same source browser application instance. Makes it possible to browse two different "places" at once.

Open Separate Code Viewer
Opens a separate window just like the code viewer, displaying the currently selected

fragment form in the group viewer.

Open Semantic Errors Viewer

Will open a semantic errors viewer on the selected group, if there are any semantic errors in that group. See later for more information on the semantic errors editor.

Open Semantic Errors Editor

Will open a semantic errors editor on the selected group, if there are any semantic errors in that group. See later for more information on the semantic errors editor.

The rest of the menu contains one entry for each window opened by the source browser. Selecting one of these items will bring the window to the fron

3.2.8 Double−click

In the code viewer, double−click can also used for browsing. Double−clicking on the current selection will perform different actions. If the current selection is

- a name application, the semantic link will be followed (if available). If the definition is located in another fragment form the contents of the code viewer will be changed to show that fragment form.
- a SLOT definition, the fragment link will be followed (if available) and the destination fragment form will be shown in the same code viewer.
- a comment mark (*), the full comment will be shown instead.
- an expanded comment (* ... *), the comment mark will be shown instead.
- a contraction (...), the contraction is opened, i.e. the next detail level is shown.
- anything else the current selection will be detailed, if it contains contractions (...)

3.2.9 Shift−double−click

Shift−double−click is used for browsing like double−click, except that separate windows are opened.

In the code viewer, shift double−click will do the same as double−click (except that the "result" will be presented in a separate window. Shift double−click will open different kind of separate windows depending on the current selection. If the current selection is:

- a name application, the semantic link will be followed (if available) and the destination fragment form will be shown in a separate code viewer
- a SLOT definition, the fragment link will be followed (if available) and the destination fragment form will be shown in a separate code viewer
- a comment, the comment will be shown in a separate window
- anything else a subviewer window with the current selection as its contents will be opened.

3.2.10 Information in the Info Pane

Besides the already mentioned information in the Info area, you will occasionally see status information to the right of the pane. This status information is shown as small boxes containing state information. There are three types of state information presently:

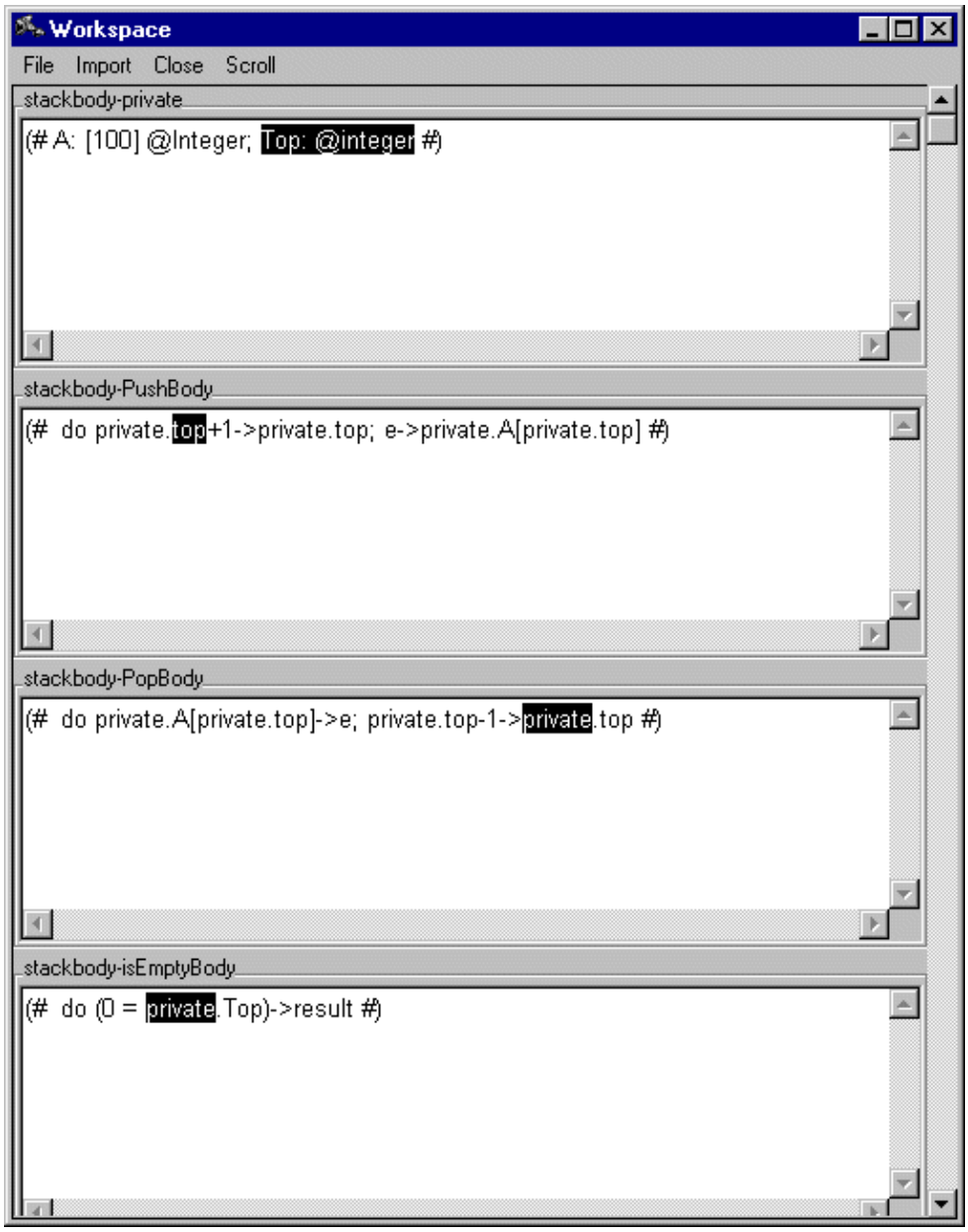
- '%' indicating that the current fragment group is locked (cannot be edited)
- '*' indicating that the current fragment group has been changed
- c' indicating that the current fragment group has been checked but a semantic checker (usually the BETA compiler), and that semantic information therefore is available for semantic browsing.

[1] The contents of the group viewer/editor can of course also be considered as part of the BETA code, but this is actually "code" written in the fragment lanuage

3.3 Workspace Window

A workspace window (you can create as many as you want) is a window in which you can view (and possibly edit) fragment forms from different fragment groups (or the same fragment group) together. The workspace window looks like:

Figure 2: Workspace window



When a fragment form has been imported to the workspace through the Import menu, it will be made visible in a view identical to the code viewer in browser window. The workspace may contain as many such views as you wish. The outer scroller enables you to scroll among the fragment forms in the workspace (try it − it is difficult to describe short).

As in the browser window, you can place the cursor between two views in the outer scroller, and thereby make the one smaller and the other bigger (and zoom the panes, as described in section

5.3.11). But in the workspace you can even more. If you press down <shift> while pressing the mouse button, you will only resize the view above the cursor − this is handy for making one view bigger without affecting the other views.

3.3.1 Menus

The above menus are those supplied by the source browser by default in a workspace window. Integrated tools may define additional workspace window menus − please refer to those manuals for details.

3.3.2 File Menu

The File menu just contains a Close entry that lets you close the workspace.

3.3.3 View Menu

The View menu is identical to the View menu in the browser window.

3.3.4 Import Menu

The Import menu will always contain at least one entry. This entry is the name of the name of the currently selected fragment group in the group list pane in the browser window. If selected, all fragment forms in that fragment group will be displayed in the workspace. Following this permanent entry, a variable number of entries are shown. Each entry corresponds to one fragment form in the currently selected fragment group in the group list pane in the browser window. If you select one of these entries, the corresponding fragment form will be displayed in the workspace. In the example above all but isEmptyBody and newBody is shown because the others are already imported.

3.3.5 Scroll Menu

The Scroll menu will contain one item for each fragment form in the workspace. If selected, the scroller will scroll such that the fragment form view is visible. In the example above these 3 fragment forms have been inserted.

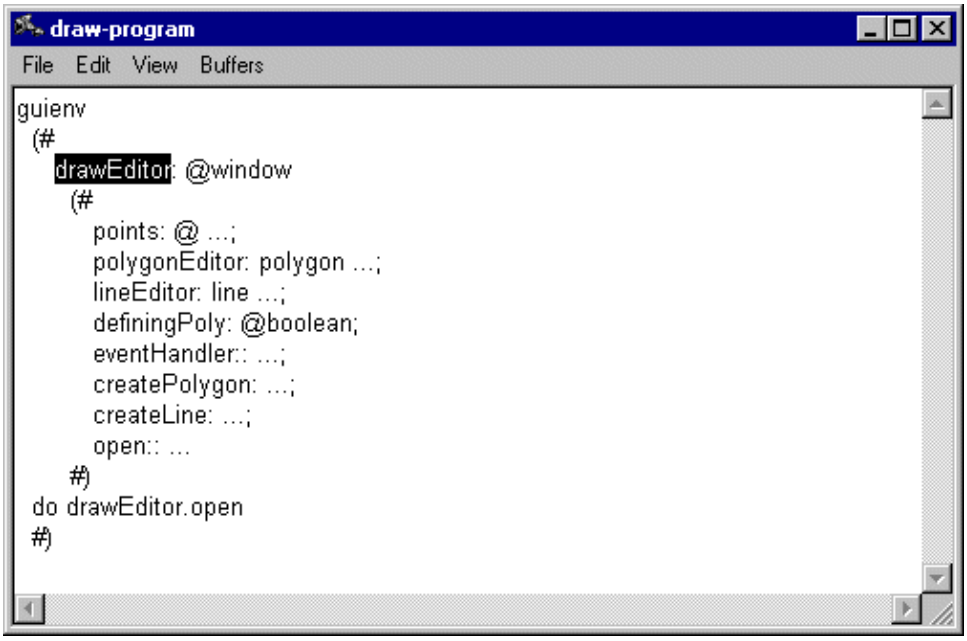
3.3.6 Close Menu

The Close menu will contain one item for each fragment form in the workspace. If selected, the chosen fragment form view is removed from the workspace. In the example above these 3 fragment forms have been inserted.

3.4 Separate Code Viewer Window

The separate code viewer window offers facilities for viewing (and possibly editing) a given fragment form, independently of any browser window. The separate code editor window looks like:

Figure 3: Separate Code Viewer Window



the code viewer is functionally identical to the code viewer in browser window.

3.4.1 Menus

The above menus are those supplied by the source browser by default in a separate code editor window. Integrated tools may define additional separate code editor window menus − please refer to those manuals for details.

3.4.2 File Menu

The File menu just contains a Close entry to enable closing the separate code editor window.

3.4.3 View Menu

The View menu is identical to the View menu in the browser window.

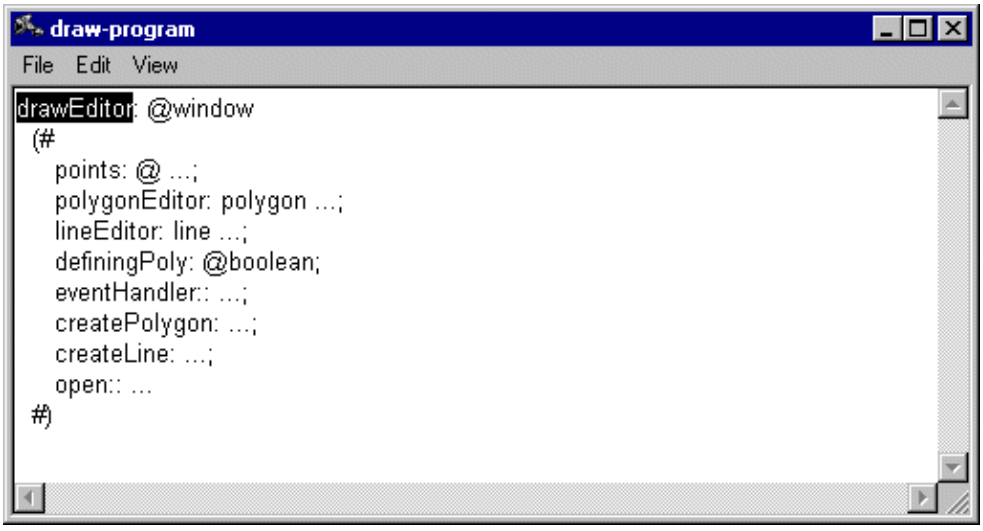
3.4.4 Buffers Menu

The Buffers menu is similar to the variable part of the History menu in the browser window. It contains one entry for each fragment form, that has been shown in this separate code editor window.

3.5 Subviewer Window

The subviewer window offers facilities for viewing (or possibly editing) a given fragment form, independently of any other code editor window. The subviewer window looks like:

Figure 4: Subviewer Window



the code viewer is functionally identical to the code viewer in browser window.

There is no Buffer menu in a subviewer. If double−click in a subviewer implies that another fragment form must be shown, it is shown in another subviewer.

3.5.1 Menus

The above menus are those supplied by the source browser by default in a subviewer window. Integrated tools may define additional subviewer window menus − please refer to those manuals for details.

3.5.2 File Menu

The File menu contains a Close entry to enable closing the subviewer window.

3.5.3 View Menu

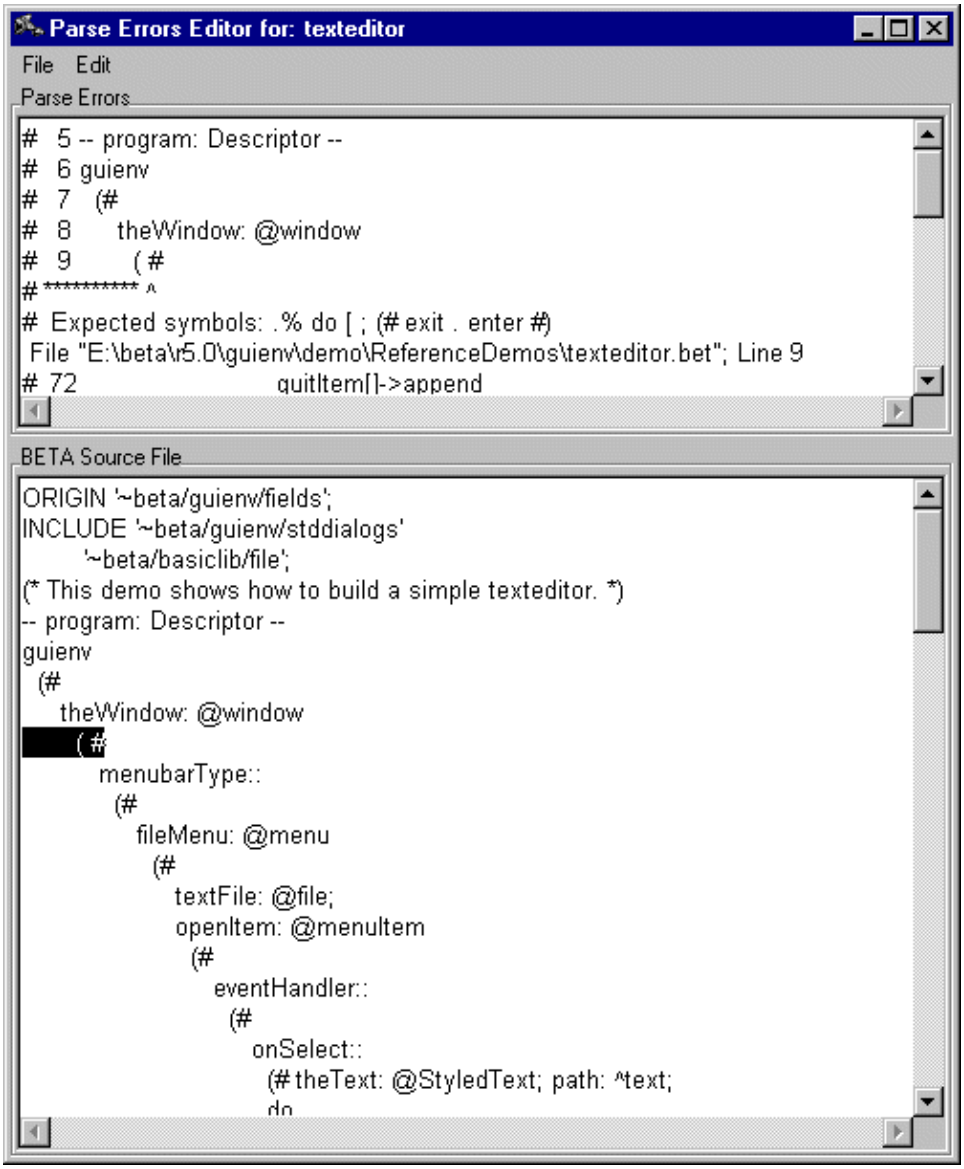
The View menu is identical to the View menu in the browser window.

3.6 Parse Editor Window

If you select a fragment group that haven't been parsed (i.e. no .ast[L] file exist, or it's outdated), a prompt will ask you if this should be done. If you say yes, the source file will be parsed.

If the source file may contain parse errors, a parse errors editor window is automatically displayed. It looks the following:

Figure 5: Parse Errors Editor



The upper pane will display the list of parse errors, with the first one selected.

The lower pane will contain the source file, with the line with the first parse error selected.

If you press <cr> in the upper pane, the next parse error is selected and the corresponding line displayed in the lower pane.

As in the browser window, you can place the cursor between the two panes, and thereby make the one smaller and the other bigger.

3.6.1 Menus

The above menus are those supplied by the source browser by default in a parse editor window. Integrated tools may define additional parse editor window menus − please refer to those manuals for details.

3.6.2 File Menu

Save

Saves the edited file.

Revert

Reads in the original file again, thereby removing the changes that might have been made to the file.

Close

Closes the parse editor window. If changes have been made to the file, and not saved, a prompt will be displayed asking whether or not to save the changes.

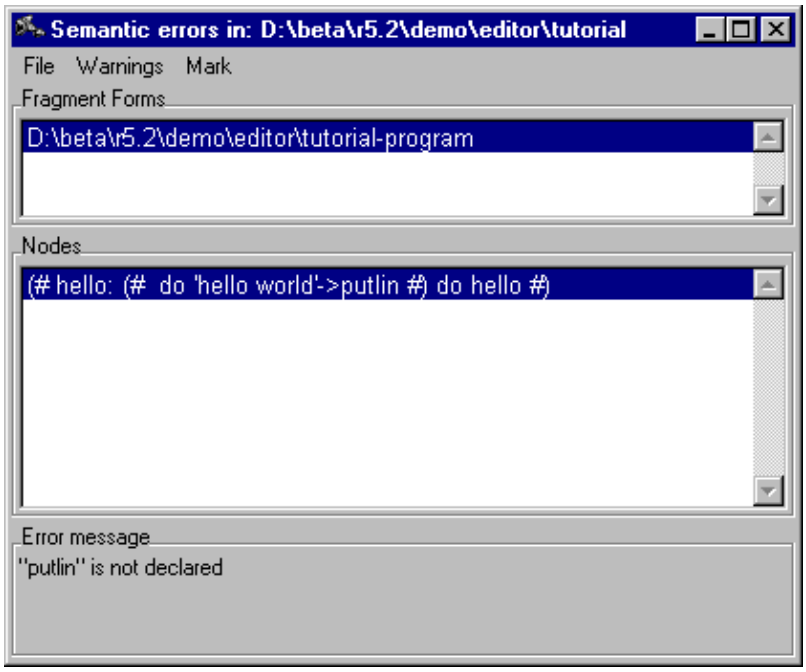
3.6.3 Edit Menu

Edit contains the Cut, Copy and Paste items with the obvious meanings (working on the contents of the lower pane).

3.7 Semantic Errors Viewer Window

The semantic errors viewer window is opened through the Windows menu in the browser window, or in those tools integrated with the BETA compiler, when a file is compiled and the compiler reports semantic errors. It appears like the following:

Figure 6: Semantic Errors Viewer



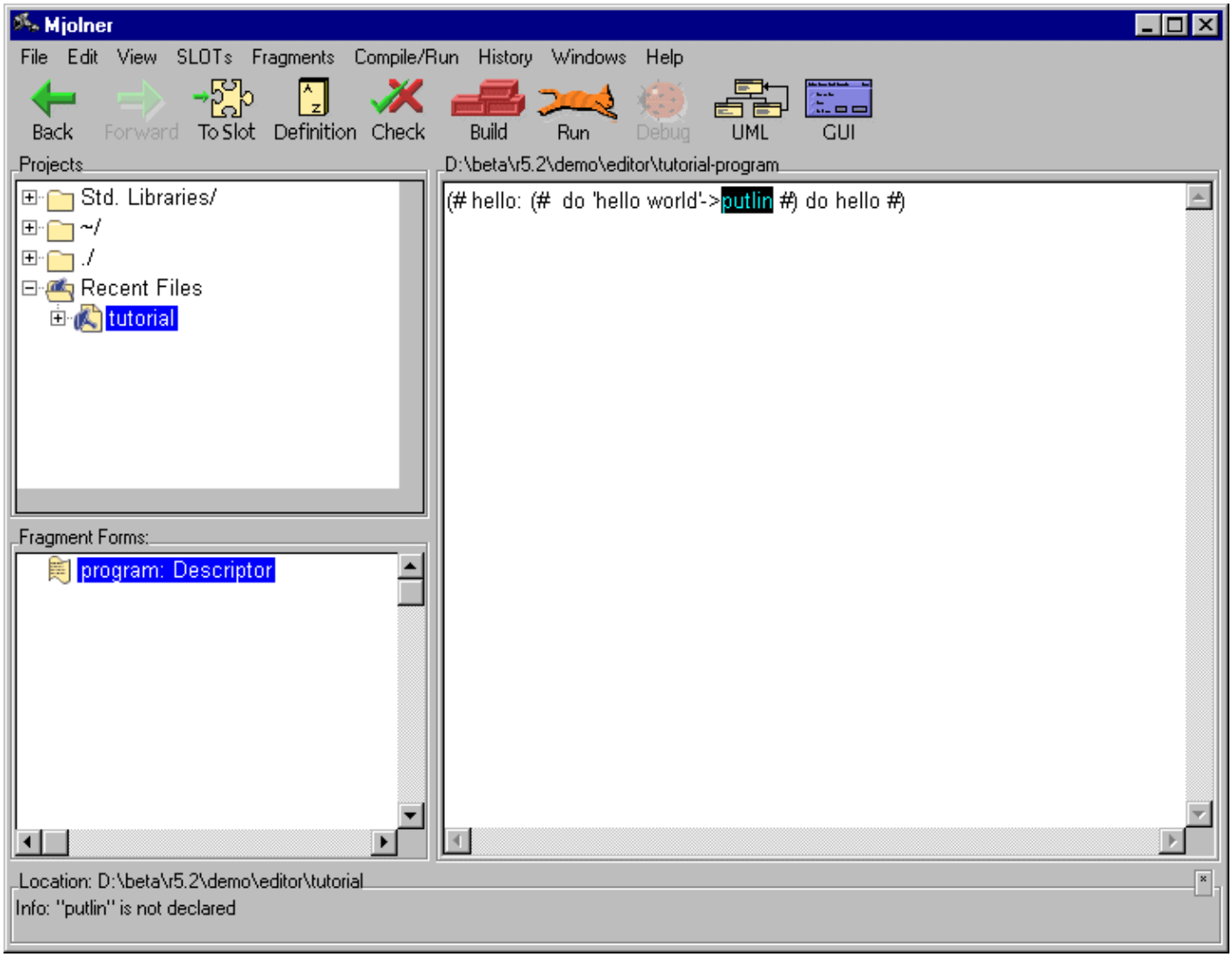
The upper pane contains a list of fragment forms containing semantic errors.

If you press <cr> in the fragment form list, the next fragment form is selected. If you press <cr> in the lower pane: the semantic error list, the next semantic error is selected.

When a fragment form is selected in the fragment form list, all semantic errors in that fragment form is listed in the semantic error list.

When a semantic error is selected in the semantic error list, the corresponding place in the source code is selected in the code viewer in the browser window:

Figure 7: A Semantic Error



As in the browser window, you can place the cursor between the two panes, and thereby make the one smaller and the other bigger.

3.7.1 Menus

The above menus are those supplied by the source browser by default in a semantic errors viewer window. Integrated tools may define additional semantic errors viewer window menus − please refer to those manuals for details.

3.7.2 File Menu

Reload
: Enables reloading the fragment group after a possible rechecking have been done.

Close
: Entry enables closing the semantic errors viewer window.

3.7.3 Warnings Menu

Include is a toggle. If toggle on, then warnings will be included in the list in the semantic error list. Otherwise, warnings are not shown.

3.7.4 Mark Menu

Mark as Corrected
: Marks the currently selected element in the semantic error list as corrected (indicated by an asterisks "*" in front of the element). This is merely a help to the user during correction of the semantic errors, enabling him to mark those errors, that he thinks he has corrected.

Unmark
: Removes such mark on the currently selected element in the semantic error list.

3.8 Semantic Errors Editor Window

The semantic errors editor window is opened through the Windows menu in the browser window. It offers the facility to view and edit semantic errors independently of any browser window. It is a semantic error viewer with a code editor adds.

When a semantic error is selected in the semantic error list, the corresponding place in the source code is selected in the code viewer in the lower part of the semantic errors editor window.

As in the browser window, you can place the cursor between the panes, and thereby make the one smaller and the other bigger.

3.8.1 Menus

The menus of the semantic errors editor window include those one defined for semantic errors viewer. View and Edit menu may also be available (depending on the tool integrated with the source browser).

3.9 Project definition

A project is either:

- A. Defined in a 'project−file' (more later)
- B. A directory
- C. A fragment group

A project may contain subprojects. These are defined differently for the three project types:

- A. A project may have defined subprojects in the project−file.
- B. The sub−directories are considered subprojects.
- C. Two subprojects are defined: [domain] and [extent]. These subprojects contain the fragment forms in the dependency−graph of the fragment group. [domain] contains all those in the domain (that is excluding BODY fragments), and [extent] contains all those in the extent (that is including BODY fragments). These two subprojects have in turn defined subprojects, namely the fragment groups in the domain (resp. the extent), sorted out into the libraries they are part of (e.g. basiclib, process, etc.).

3.9.1 Standard Projects

Assuming that you have not used source browser applications before, the project−list (1) will display three projects at start−up: "Std. Libraries", "~/" and "./".

"Std. Libraries" is a predefined project (defined in a project−file located in

~beta/configuration/r4.0/ymer.pjt), containing all the libraries in the standard MBS system (basiclib, containers, etc.). An example of this file is:

```
project basiclib '~beta/basiclib '
project containers '~beta/containers'
project guienv '~beta/guienv'
project bifrost '~beta/bifrost'
project process '~beta/process'
project persistentstore '~beta/persistentstore'
project oodb '~beta/oodb'
project distribution '~beta/distribution'
project sysutils '~beta/sysutils'
project unixlib '~beta/unixlib'
project Xt '~beta/Xt'
project mps '~beta/mps'
project objectserver '~beta/objectserver'
project contrib '~beta/contrib'
project demo '~beta/demo/r4.0'
```

"~/" is a directory project, referring to your home directory, and "./" is a directory project, referring to the current directory (i.e. the directory, where you invoked the source browser application from).

If you have invoked a source browser application with command line arguments, these will be shown in the projects list as well.

When a project/directory is selected in the project list pane, the group list pane will display all fragment files in that project. The source browser remembers the last selection in the group list pane and the group viewer, such that it will reestablish these selections next time you enters the same project or group.

When a group with one fragment form, it will automatically be selected.

3.9.2 Fold/unfold Projects

To see the subprojects/subdirectories of a project, double−click on the project (called unfold the project), then you will see the subprojects, properly indented, e.g.:

Std. Libraries
    basiclib
    containers
    bifrost
    ...
...

Double−clicking again on the "Std. Libraries" will 'fold' the subprojects, only showing the "Std. Libraries" project, and not the subprojects.

3.9.3 Dependency Graph − Domain and Extent Projects

You can make a fragment group into a project by double−clicking on it in the group list pane. It will then appear in the project list pane as a fragment group project. Double−clicking on it in the project list pane will calculate the dependency graph, and show the [domain] and [extent] subprojects.

4 Editor

4.1 Code Editor

The code editor provides structure editing on each fragment form. Many of the following menu items have keyboard shortcuts associated with them. A complete list of these can be found in the appendix.

4.1.1 Edit Menu

Undo
: Initiating this command will undo the latest performed editing operation in the code editor. All editing operations performed in a session can be undone.

Redo
: Initiating this command will redo the latest performed editing operation in the code editor. All undone editing operations performed in a session can be redone until a new editing operation is performed.

Cut
: The current selection is removed from the fragment form. If the current selection may be deleted (e.g. a statement or a declaration) it will disappear. If the current selection may not be deleted (e.g. the expression part of an if−statement) a nonterminal of appropriate syntactic category will be inserted instead of the current selection. The deleted fragment part will be put onto a clipboard and can later be pasted into the same or another fragment form.

Copy
: The current selection is copied onto the clipboard, and can later be pasted into the same or another fragment.

Paste
: Whenever the current selection and the structure on the clipboard are of exchangeable syntactic categories, Paste is enabled. Paste will replace the current selection with the structure currently on the clipboard. The content of the clipboard will not be changed.

Paste Before
: Like Paste, but preceeded by an Insert Before operation.

Paste After
: Like Paste, but preceeded by an Insert After operation.

Clear
: Same as Cut but nothing is put on the clipboard.

Select All
: Selects all the code in the window in structure editing mode and the all the code in the textediting area in text editing mode.

Textedit
: This command enters text editing mode on the current selection. See also Text editing in Editor tutorial.
Please note that text editing is only fully available if a parser for the supported language has been generated. If a parser is not available only lexems can be edited textually, i.e. name declarations, name applications, strings and constants. In that case, there is no check that the lexems are legal.

External Textedit
: Like Textedit but here an external texteditor is used. This facility is only available in Unix versions. The environment variable EDITOR is recognized by Sif. When this command is invoked the current selection is inserted in the external editor. When the external editing has finished, by saving the results, the resulting text is inserted instead of the current selection. Parse errors are detected in the same way as in the internal texteditor.

E.g. emacs can be used by setting EDITOR to /usr/local/bin/emacsclient and starting emacs as a server in the following way: emacs −f server−start.

Parse Text

Exits text editing mode and parses the edited text according to the syntactic category of the current selection before entering text editing mode. See also Text editing in Editor tutorial.

Revert Text Editing

Exits text editing mode and reestablishes the situation before text editing was entered. Alternatively the <esc> key can be used.

Insert Before/Insert After

Whenever the current selection is one or more list elements, these entries will enable you to insert a nonterminal of same syntactic category as the list elements either before or after the current selection, respectively. If the list elements are on separate lines before/after means above/below, respectively. If the list elements are on the same line before/after means left/right respectively. An alternative to Insert After is the <cr> key.

Remove Optionals

Removes all nonterminals representing optional productions from the current selection.

Show Optionals

Inserts all nonterminals representing optional productions in the current selection.

Find...

Makes it possible to search for a substring of a lexem, i.e. a name definition, a name application or a string in

- ◊ the current fragment form
- ◊ the current fragment form of the source browser, i.e. the fragment form currently presented in the codeviewer of the source browser
- ◊ the current fragment group of the source browser
- ◊ the current project in the source browser e.g. also in the domain and extent of the current fragment group

A dialog pops up. See Searching An especiallly useful facility is the browser of search hits, that collects all the search hits for easy overview and access. If the current selection is a name declaration, when the this command is activated, all applications of this name will be searched for in the selected scope. Using this facility, it is possible to search for applications of a name in the entire dependency graph.

Replace...

Extends the search dialog with a replace text field.

Open Subeditor

A subeditor on the current selection is opened, i.e. a code editor where the current selection is the root. Shift−double−click can also be used.

Form Write Protection

If checked it is not possible to modify the fragment form. Is automatically set if group write protection or global write protection is set.

4.2 Editing and contractions

Structure editing on parts of the document that contain contractions is done exactly as described above. You can for example cut, copy and paste constructs that contain contractions. When text editing constructs that contain contractions, some parser technical information is included in the contractions before the text editor is activated. This information is necessary to avoid loosing contractions during text editing. A contraction in text editing mode looks like:

<<@4711: Descriptor>>

where @4711 is an internal address of the sub−AST that is contracted, i.e. not presented, and Descriptor is the syntactic category of the contracted sub−AST.

Do not modify the contents of contractions during text editing. Likevise if the contraction contents is commented out during text editing, the corresponding code will be lost.

4.2.1 Expand Menu

This menu is a pop−up menu, that is activated by the Pop−up Menu Button on the mouse (See basic user interface principles).

The content of the Expand menu is dependent on the current selection. If the current selection is a nonterminal, the Expand menu will contain an entry for each language construct that can replace this nonterminal. If the current selection is an optional or list nonterminal, the Expand menu will contain an empty entry as well. This entry enables you to remove that nonterminal.

4.2.2 SLOTs Menu

This menu is only present in the menu bar if the current fragment group is a BETA fragment group. The menu makes use of the fragment clipboard that is used as an intermediate store for fragment forms that are 'moved around' between fragment groups. The fragment clipboard is also used in the Fragment menu.

Make DoPart SLOT...
: Substitutes the current selection with a SLOT definition and the removed part of the document is inserted in an DoPart fragment form, that is stored in the fragment clipboard. The name of the SLOT is prompted for. The default name is the name of the enclosing pattern.

Make Descriptor SLOT...
: Substitutes the current selection with a SLOT definition and the removed part of the document is inserted in an Descriptor fragment form, that is stored in the fragment clipboard. The name of the SLOT is prompted for. The default name is the name of the enclosing pattern.

Make Attributes SLOT...
: Substitutes the current selection with a SLOT definition and the removed part of the document is inserted in an Attributes fragment form, that is stored in the fragment clipboard. The name of the SLOT is prompted for. The default name is the name of the enclosing pattern.

Create Implementation File
: In a standard file dialog an implementation file can be created. It will be created as a body file to the current fragment group selected in the browser. The ORIGIN and BODY properties are automatically inserted in the respective files.

Select Implementation File
: Like Create Implementation File, but an existing file is chosen in the dialog.

Set Current as Implementation File
: The current fragment group selected in the browser is set to be the implementation file. If this file is set the Make SLOT commands and the Hide Implementation... command will automatically paste the created fragment forms that correspond to the SLOTs into the implementation file.

Unset Implementation File
: The file set in Set Current as Implementation File is unset.

Set SLOT Name Prefix...
: When one of the Make SLOT commands or the Hide Implementation... command is used the default name in the dialog for the SLOT name will be the text set by this command, followed by the name of the enclosing pattern (if any).

Unset SLOT Name Prefix
: Clears the text set in Set SLOT Name Prefix...

Hide Implementation...
For each DoPart in the current selection in the code editor the user is asked whether it should be hidden into the implementation file by making it into a SLOT and moving the DoPart into a fragment form in the implementation file.

4.3 Group Editor

The group editor is only relevant if the language in question is BETA. It is used for browsing or editing BETA programs. It is used to present and modify the structure of a group, i.e. the properties (ORIGIN, BODY, MDBODY, INCLUDE etc.) and fragments (Descriptor forms or Attributes forms).

4.3.1 Fragment Menu

The cut, copy and paste operations are using a so−called fragment clipboard.

Most commands in this menu are performed according to the currently selected item in the group editor. The selected item can be a property, like ORIGIN, INCLUDE and BODY or a fragment form,.

Undo
Undo the last executed command in the group editor. Not implemented yet.
Cut Fragment Form
The selected fragment form is deleted. A copy of the fragment form is put on the fragment clipboard.
Copy Fragment Form
A copy of the selected fragment form is put on the fragment clipboard.
Paste Fragment Form Before
The fragment form on the fragment clipboard is inserted in the fragment group before the selected item.
Paste Fragment Form At End
The fragment form on the fragment clipboard is appended to the list of fragment forms in the fragment group. It can also be pasted into a SLOT definition in a code editor (See the Paste Fragment Form command in the SLOTs menu).
Insert DoPart Fragment Form...
A new Dopart fragment form is created and inserted before the selected item. You are prompted for the name of the fragment form.
Insert Descriptor Fragment Form...
A new Descriptor fragment form is created and inserted before the selected item. You are prompted for the name of the fragment form.
Insert Attributes Fragment Form...
A new Attributes fragment form is created and inserted before the selected item. You are prompted for the name of the fragment form.
Edit ORIGIN, INCLUDE etc. ...
The properties of this fragment group can be edited in a separate code editor that supports the 'property specification language'. See Property Editor.
Edit Fragment Form Name...
The name of the selected fragment form item can be modified.

4.4 Property Editor

The property editor is able to edit the properties that can be attached to a fragment group, i.e. properties like ORIGIN, INCLUDE; BODY, MDBODY, OBJFILE, LIBFILE, BETARUN etc. In order to edit the properties of a fragment group, a structure editor supporting a grammar for the 'property language' is used. The property editor is activated in the Fragment menu by means of the Edit Properties command.

4.5 Compile/Run Menu

This menu is only present in the menu bar, if the language supported in the active code editor is BETA.

Check Current
: The current fragment group in the browser is checked. Output from the checking is written in the Log Window. Possible semantic errors are shown in semantic error viewer and the structure corresponding to the firs semantic error in the list is selected.

Check Program
: The latest selected PROGRAM is checked.

Compile Current
: The current fragment group in the browser is compiled, i.e. checking, code generation and linking is performed. Output is written in the Log Window. Possible semantic errors are shown in semantic error viewer and the structure corresponding to the first semantic error in the list is selected.

Compile Program
: The latest selected PROGRAM is compiled.

Compile and Run
: Compile and run the latest selected PROGRAM.

Run Program
: If an executable exists that has the same name as the current fragment group it is started.

Quit Current
: Kills the process that was started by Mjolner.

Semantic Errors...
: Opens the semantic error viewer.

Next Semantic Errors
: Selects the next semantic error.

Debug
: Activates the debugger on the current program in the browser.

Debug Executable
: Debug the program selected using an open dialog.

4.6 Backup

Backup group file: .ast~ (or .astL~ on PC)

When a fragment group is opened in Sif a backup of, say, foo.ast is taken on foo.ast~.

Auto−save group file: .ast# (or .astL# on PC)

When a certain number of structure editing operations has been performed, a copy of the fragment group, say, foo.ast is stored on foo.ast# Text editing including parsing is considered as 1 structure editing operation. When a fragment group is being opened in the editor and an auto−save group file exists that is newer than the .ast file, you are asked whether that file should be used instead.

Backup of text file: .bet~

When a fragment group is saved either by the user or by the compiler, a copy of the corresponding text file, say, foo.bet (if present) is taken on foo.bet~

4.7 Opening Grammar Files

Sif can also be used to edit or inspect the available grammars and corresponding prettyprint specifications. The grammar files of the Mjølner BETA System are normally located in `~beta/grammars`.

There is a naming convention for the grammar files and the prettyprint specification files. For BETA the grammar file is called beta−meta.gram and the corresponding group file is called beta−meta.ast. The prettyprint specification file for BETA is called beta−pretty.pgram and the corresponding group file is called beta−pretty.ast. To open the grammar file for BETA the entry beta−meta must be selected. To open the prettyprint specification file for BETA the entry beta−pretty must be selected. The grammar directory contains also a group file called beta.ast but this is for internal use only.

For more information on grammars and prettyprint specifications etc. see [MIA 91−14].

5 CASE Tool

5.1 How to Get Started

5.1.1 Class diagrams

Freja is launched from within the Mjølner tool one of the following ways:

- In the source browser select the *UML* toolbar button . If the code editor is empty a new diagram with corresponding code may be created. If the code editor contains an Attributes fragment form or a Descriptor fragment form with declarations, the fragment form will be shown in Freja as a reverse engineered diagram of the code in the fragment form. Note that relations, like specializations and associations, will only be reverse engineered if the fragment has been checked (e.g. using the *Check* toolbar button). Also note that if a diagram already exists for the selected fragment this will be opened and possibly be updated.
- In the source browser open a BETA fragment form. Right−click anywhere in the code editor and select *Show UML Class Diagram* from the popup menu. This will have the same effects as mentioned above.

5.2 Work Sheets

The page title of a work sheet is 'WorkSheet'. A work sheet is a page containing class diagrams. The class diagrams on a single work sheet is a reflection of the classes (patterns) contained in one or more BETA fragments.

5.3 The Group Page

If a work sheet with class diagrams is open, choosing *Windows:Group Window* a page called 'Group Window' will be opened. This page will contain one or more group diagrams.

The number of group diagrams in the Group Window corresponds to the number of currently open BETA fragment groups.

The Group Diagram displays the properties and the fragments defined in the group. The properties and the fragments can be detailed, i.e. shown in detail, by selecting a node and then select Detail in the View menu. The detailing can also be performed by double−clicking on the nodes. By e.g. double−clicking on the lib: Attributes node, the declarations in the fragment will be presented as a class diagram in on a work sheet (if it is already detailed, the detailed version will become current focus). Doubleclicking Origin, Include or Body fields in a group diagram will open that group and bring up a group diagram for it.

5.4 The Menu Bar

The majority of the commands in the menus operate on the active page. A page is made active

either by clicking in the contents of the page, by clicking the titlebar of the surrounding window or by selecting the page from the *Windows* menu.

Many of the commands of the menus also operate on the current selection of the active page. The current selection is called current focus.

5.4.1 File Menu

New Diagram...

Makes a dialog popup. Entering a file name without extension and clicking ok will result in the creation of a minimal BETA library, which is given the entered file name. The result of this will be a diagram, like the one shown above, showing up on the active work sheet. The title of the diagram is the name of the new BETA fragment group (the file name: fee) followed by a dash (−) and the name of the new BETA fragment form (lib). The single attribute shown below the title corresponds to the single class attribute of the library:

```
ORIGIN '~beta/basiclib/betaenv'
-- lib: Attributes --
<<NameDecl>>: (# #)
```

Notice:

- ◊ Diagrams displaying the attributes of a library, like the one mentioned above, are called attributes diagrams.
- ◊ Diagrams displaying the descriptor of a program are called descriptor diagrams.

A general note:

- ◊ The common name for descriptor and attributes diagrams is fragment diagrams.

New Graphics Page...

Invoking this command will create a new page entitled 'Graphics Page'.
Graphics pages are as the name implies ment for graphics only. That is all objects appearing on these pages are interpreted as having no other semantics than the purely graphical one. Basically the commands that apply on these pages are the purely graphical ones that can be found in the *Graphics* and *Align* menues. Also Clear from the Edit menu applies to purely graphical objects (whereas Cut, Copy and Paste regrettably are not yet implemented for these types of objecs).

Open...

Selecting Open... makes the open dialog popup. From here a .bet or .ast file can be selected and opened.
Selecting a .bet or .ast file will, depending on the category of the selected fragment, result in either a descriptor or attributes diagram on the active worksheet. This class diagram reflects the attributes of the opened fragment. One exception from this should be noted: When a file containing more than one fragmentform is opened the first thing to show up will be the group page with a diagram displaying the fragmentforms of the selected file (see The Group Page). Now, by doubleclicking one of these fragmentforms its attributes can be shown on the active worksheet as described above.

Notice:

- ◊ A BETA fragment is said to be related to a class diagram if the class diagram reflects attributes declared in this fragment.
- ◊ Only one .diag file can be open at the time, but the different class diagrams on the work sheets can be related to several different BETA fragments.

Save

Performs a save of all shown diagrams on all open pages to the currently open .diag file. This means that the layout of the diagrams on the pages are saved, and that the fragments related to the diagrams are saved to their .bet and .ast files. The .diag file can then later be opened for further browsing and editing and eventually be saved again.

Notice:

◊ The default name for the diagram is the name of the BETA fragment that was opened or created first during the session. To change this name use Save As (see below).

Save As...

Using this command invokes the dialog. Entering a name in the text field of the dialog with extension .diag and choosing Save will result in a save of all shown diagrams on all open pages to this file (as described above).

Revert

Reverts all edits done during the current Freja session. I.e. the diagram will appear as it did when it was initially opened.

☞**Notice**:

◊ Currently not implemented.

Save Page As Graphics...

Using this command you can, using the dialog that pops up, save the current page on disc as graphics only. The graphics only format is Encapsulated PostScript.

Page Setup...

Use Page Setup before printing. To get a satisfactory print the recommended settings are:

◊ Paper: A4 letter.
◊ Output form: Scale to fit.
◊ Omit page borders.

☞**Notice**:

◊ Currently not implemented.

Print...

Initiating the print command will bring up a dialog asking the user to indicate which pages are to be printed. If all pages are to be printed simply select the *Print all pages* field of the dialog.

You can choose to either print directly to a printer or to print to a file.

If printing directly to a printer, type in a print command − e.g. `/usr/local/bin/lpr −P<PrinterName>` on a unix system.

If printing to a file, type in the name and path to the file − e.g. /tmp/freja−print.ps.

☞**Notice**:

◊ *Print* is currently not implemented on MS Windows platforms. To print your diagram on these platforms use *Save Page As Graphics*.

Close

Hides all Freja windows.

5.4.2 Edit Menu

Undo

Choosing Undo from the Edit Menu will undo the latest editing operation on a work sheet in Freja. If this operation itself was Undo the state before the Undo will be restored.

The operations which can be undone are Cut, Copy, Paste, Insert Before, Insert After, Paste Before, Paste After, Expand.

☞**Notice**:

◊ Currently not implemented.

Cut

Initiating Cut removes the contents of current focus on the active work sheet. The deleted part of the class diagram is moved onto a clipboard and can later be pasted into any other class diagram. Only titles and attributes can be cut from a class diagram (not any of the relations). Performing a Cut with a title of a class diagram as current focus results in the removal of the entire class diagram and the removal of the attribute of which this diagram is a detailed version. Notice that this type of cut will also cut the corresponding code.

Copy

Copies the contents of current focus on the active work sheet onto the clipboard and can later be pasted into any other class diagram. Only titles and attributes can be copied from a class diagram (not any of the relations).

Paste Before

When current focus is an attribute of a class diagram this command performs an Insert Before followed by a Paste.

Paste

If current focus is part of a class diagram − i.e. a title or an attribute − and if the clipboard contains part of a class diagram, then Paste can only be performed when the contents of current focus and the class diagram part on the clipboard are of exchangeable syntactic categories. If this is the case and a Paste is performed the effect is that the contents of current focus is replaced by the class diagram part on the clipboard and the code is updated correspondingly.

In this context when cutting or copying with a title of a class diagram as current focus the syntactic category is the same as the syntactic category of the attribute of which the class diagram is a detailed version.

Paste After

When current focus is an attribute of a class diagram this command performs an Insert After followed by a Paste.

Clear

Removes the contents of current focus on the active work sheet (nothing is moved to the clipboard).

Edit...

If current focus is an attribute of a class diagram or the title of a diagram detailed from such an attribute, this command will invoke the appropriate dialog the attribute or class. If a new name is entered, it must be a lexem (i.e. some legal name in BETA). Therefore before accepting it, the name is parsed and if unlegal the dialog will ask the to change it.

Open codeeditor

If current focus is an attribute of a class diagram or the title of a diagram detailed from such an attribute, this command will invoke a codeeditor on the code that corresponds to current focus. The functionality of a codeeditor is mainly identical to the functionality of the codeviewer/−editor of the browser window of Sif. Codeeditors are therefore typically used when textual structure editing on some part of the code is called for.

Insert Before

If the contents of current focus is an attribute of a class diagram this command will insert a new unexpanded attribute before current focus.

Insert After

If the contents of current focus is an attribute of a class diagram this command will insert a new unexpanded attribute after current focus.

Remove Optionals

If current focus is part of a class diagram this command removes all nonterminals representing optional productions from the contents of current focus.

Show Optionals

Inserts all nonterminals representing optional productions in current focus, if it is part of a class diagram.

5.4.3 New Menu

In general the *New* menu contains entries that are composed of two or more ordinary commands and are provided to make some of the more frequently used series of commands easier to perform.

Class...

This command basically creates a new class pattern and makes a corresponding class

diagram show up on the active work sheet. The user is prompted for a class name and is furthermore able to decide whether the class should be declared virtual (virtual, binding of virtual or final binding of virtual).
The class is declared in what is called the current fragment diagram.
If the current focus is a title or an attribute of a class diagram P the current fragment diagram is the fragment diagram that is related to the same fragment as the diagram P.

Attribute...

This command creates a new attibute. The user is promted for name, type and whether it should be a static or dynamic reference attribute.

Operation

This command creates a new operation. It inserts the new operation in the class where current focus is set. That is, the class diagram which includes the title or attribute which is current focus. This diagram is called the current diagram. The user is prompted for a name for the new operation and is furthermore able to decide whether the operation should be declared virtual (virtual, binding of virtual or final binding of virtual).

Local Class

This command creates a new class. It inserts the new class in the class where current focus is set. That is, the class diagram which includes the title or attribute which is current focus. The user is prompted for a name for the new class and is furthermore able to decide whether the class should be declared virtual (virtual, binding of virtual or final binding of virtual).

5.4.4 Relations Menu

The commands of the *Relations* menu all relate one node to another. For this reason they are all basically used in the same manner: Choose the wanted relation, possibly a dialog appears, then fill in the wanted settings and click *Ok*, click the left mouse on the source of the relation and after that click the right mouse on the destination of the relation.

☞**Notice**:

- The settings of some of the relations (aggregations and associations) can be altered by choosing *Edit..* from the popup−menu that appears when you right−click the relation. This will bring up the appropriate dialog with settings according to the right−clicked relation.
- The source or destination of a relation can be altered after its creation by selecting the source/destination end of the relation connector and dragging it to the new endpoint

General Relationship

Any elements of a class diagram can be connected using through this relation. It has no formal semantics and no code results from creating it. It is intended as a purely informal symbol, used in situations where the user has not yet decided which type of concrete relation (typically aggregation or association) should exist between two classes.

Aggregation

When executing this command a dialog appears. The aggregation dialog makes it possible to:

- give the aggregation a name,
- specifify the multiplicities of the whole and the part,
- specify the implementation of a one−to−many [2] aggregation; that is to specify if a repetition or one of the basic container classes should be used in the resulting code,
- specify if the aggregation is to be implemented by−reference or by−value.

Note that default values are set, so that the user only has to specify the settings he actually wants to change.
When Ok has been clicked, click the left mouse on the source ("whole") of the relation and after that click the right mouse on the destination ("part") of the relation.

The destination (the part) must either be a class declaration, a virtual class declaration, a binding of a virtual class declaration or a final binding of a virtual class declaration. The destination may either be detailed or not.
If the source (the whole) of this operation is the title of a class symbol the result will be that a new attribute containing the corresponding code is inserted in the class. If the source is an attribute of a class and this attribute corresponds to a dynamic or static reference in the code the operation will result in this attribute being altered so that it now corresponds to the generated aggregation code.

Association

When executing this command a dialog appears. The Association dialog makes it possible to:

- give the association a name,
- specify role names for both sides of the association,
- specify multiplicities for both sides of the association,
- specify the implemention of a one−to−many or many−to−many association; that is to specify if a repetion or one of the basic container classes should be used in the resulting association code,
- specify if the association should be embedded or not; that is to specify if the generated code is to result in a separate association class or if it is to be embedded in the source and destination classes,

Note that default values are set, so that the user only has to specify the settings he actually wants to change.
When Ok has been clicked in the dialog, click the left mouse on the source of the relation and after that click the right mouse on the destination of the relation.
Creating this relation the source and destination must be detailed versions of either class declarations, virtual class declarations, binding of virtual class declarations or final binding of virtual class declarations.

Specialization

Creating this relation the source and destination must be detailed versions of either class declarations, virtual class declarations, binding of virtual class declarations or final binding of virtual class declarations.
The result of creating a Specialization relation will be that the descriptor corresponding to the source diagram gets the destination class inserted as its prefix.

Binding of Virtual

For this relation the source must either be an attribute containing an unexpanded attribute declaration or an attribute containing a pattern, virtual or binding declaration. The destination must be an attribute containing a binding or virtual declaration.
Having created a relation like this the source will have become a binding of the virtual of the destination.

☞**Notice**:

◊ A possible descriptor of the source will not be lost through this operation.

Final Binding of Virtual

Behaves exactly like the Binding of Virtual relation except that the source in this case becomes a final binding of the virtual of the destination.

Pattern Variable

For the Variable Pattern relation the source must be an attribute containing either an unexpanded attribute declaration or some kind of simple declaration. The destination must either be the title of a diagram that is a detailed version of an attribute containing a pattern declaration or it must be an attribute containing a pattern declaration.
The result will be that the source becomes a declaration of a pattern variable with the destination pattern as qualification.

5.4.5 View Menu

Abstract

If current focus is a title of a class diagram Abstract will remove this class diagram and all diagrams detailed from it.
If current focus is an attribute that has been detailed the command will remove the detailed version of the attribute and all diagrams detailed from that.

Abstract Recursively

If current focus is a title of a class diagram Abstract Recursively will remove all diagrams detailed from it (but not the diagram to which the title belongs).
If current focus is an attribute that has been detailed the command will remove the detailed version of the attribute and all diagrams detailed from that.

Overview

If current focus is a title or an attribute of a class diagram this command will remove all class diagrams not on the "detail path" to the diagram which is current focus.

Detail

If current focus is an attribute of a class diagram and this attribute contains an object descriptor, like e.g. a pattern declaration does, Detail will display the attribute on detailed form; that is, as a class diagram displaying the attributes of this descriptor.
If current focus is a title of a class diagram Detail will perform a Detail on all attributes that can be detailed in the class diagram.

☞**Notice**:

- ◊ Doubleclicking a detailable attribute of a class diagram will perform a Detail on the attribute. If the attribute is already detailed doubleclicking it will make the title of the detailed version the new current focus.

Detail Recursively

If current focus is an attribute of a class diagram and this attribute contains an object descriptor, like e.g. a pattern declaration does, Detail Recursively will display the attribute on detailed form; that is, as a class diagram displaying the attributes of this descriptor. Then it will perform this command recursively on the detailable attributes of the diagram that has now been detailed.
If current focus is a title of a class diagram Detail Recursively will perform a Detail Recvrsively on all attributes that can be detailed in the class diagram.

Search...

By means of the Search command it is possible to search for a substring of a lexem, i.e. a name definition, a name application or a string.
Notice that this command does not search text auxillary to the class diagrams, e.g text in user created labels.

Replace...

Extends the search dialog with a replace text field so that the found text can be replaced with a text specified in this text field.

Layout Subclasses

If current focus is the title of a class diagram and if there are currently displayed sub−patterns of this pattern, Layout Sub−patterns will rearrange the positions of these sub−class diagrams in a manner that makes them appear "nicely" in a tree structure below the selected diagram.

Edged Specialization Connectors

If current focus is the title of a class diagram and if there are currently displayed sub−patterns of this pattern, Edged Specialization Connectors will make specialization connectors only follow horizontal and vertical lines.

Show All...

- ***Attributes*** Makes all attributes of all class diagrams visible.
- ***Aggregations*** Makes all aggregation connectors visible.
- ***Specializations*** Makes all specilization connectors visible.
- ***Associations*** Makes all aggregation connectors visible.

Hide All...

- ***Attributes*** Makes all attributes of all class diagrams invisible.
- ***Aggregations*** Makes all aggregation connectors invisible.
- ***Specializations*** Makes all specilization connectors invisible.
- ***Associations*** Makes all aggregation connectors invisible.

Show Attributes With...

The four following entries concerns the appearance of the information inside the atributes of the class diagrams − that is, the textual prettyprint of the attributes.

- ***Name*** This is the default prettyprint mode for attributes. This prettyprint mode only displays the name (the NameDecl part) of the declaration.
- ***Name and Type*** In this mode both the name and the type of the declaration inside an attribute is shown.
- ***Type*** This prettyprint mode only displays the type of the declaration.
- ***Type and Kind*** The Name and Kind prettyprint mode displays the name and the kind (e.g. static item symbolised by a '@') of the declaration.

Refresh Page

Redraws the contents of the page. ***Scroll Selection Into View***

If there is a selection on the current page, this page is scrolled so that the selection is visible.

5.4.6 Graphics Menu

The Create menu commands draw non−formal graphic objects (nodes and connectors).

Connector

Creates a connector between two graphical elements. Click the left mouse on the source and after that click the right mouse on the destination. ***Rectangle***

Left click to define a corner of the rectangle, drag the mouse untill the rectancle has the desired size and shape, then left click again. ***Rounded rectangle***

Left click to define a corner of the rectangle, drag the mouse untill the rectancle has the desired size and shape, then left click again. ***Ellipse***

Left click to define the center of the ellipse, drag the mouse untill the rectancleellipse has the desired size and shape, then left click again. ***Polygon***

Left click to define each edge of the polygon, then right click to connect the first and last edge.

Label

To create a text label left click the position where you want the text to start. Type the desired text and click anywhere to finish.

5.4.7 Align Menu

All entries in this menu aligns a number of graphical elements (like f.ex. two or more classes). Holding the shift−key down click on each of the elements you wish to align, then select the desired

alignment. Alignment will be preserve the position of the first element you clicked, aligning the other elements with respect to this position.

Left side

The selected elements are repositioned so that the left−most edge (of the bounding box) of each of the selected elements appear on the same vertical axis. ***Right side***

The selected elements are repositioned so that the right−most edge (of the bounding box) of each of the selected elements appear on the same vertical axis. ***Top edge***

The selected elements are repositioned so that the top−most edge (of the bounding box) of each of the selected elements appear on the same horizontal axis. ***Bottom edge***

The selected elements are repositioned so that the bottom−most edge (of the bounding box) of each of the selected elements appear on the same horizontal axis. ***Horizontal center***

The selected elements are repositioned so that the center (of the bounding box) of each of the selected elements appear on the same horizontal axis. ***Vertical center***

The selected elements are repositioned so that the center (of the bounding box) of each of the selected elements appear on the same vertical axis. ***Spacing***

> A dialog appears and either vertical or horizontal spacing (or both) can be specified.
>
> In the case of vertical spacing the selected elements are repositioned so that the bottom−most edge (of the bounding box) of one element has a distance corresponding to the specified spacing to the top−most edge (of the bounding box) of the next element.
>
> In the case of horizontal spacing the selected elements are repositioned so that the right−most edge (of the bounding box) of one element has a distance corresponding to the specified spacing to the left−most edge (of the bounding box) of the next element.

---

[2] An aggregation is said to be one−to−many if the multiplicity of the part is one and the multiplicity of the whole is more than one

5.5 Appendix: Frequently Asked Questions (FAQ)

5.6 Contents

5.7 PART I: Frequently Asked Questions

5.7.1 Q01) What is Freja?

Freja is an object−oriented CASE tool supporting system development with BETA as the implementation language. Together with Sif, the Source Browser and Editor part of the Mjølner Tool ([MIA 99−39], [MIA 99−40], [MIA 99−34]), Freja supports a smooth transition from design diagrams to implementation code and vice versa. The design notation is a graphical syntax for part of the BETA programming language. The basic idea is to use the same abstract language for design as well as implementation. A graphical syntax is used for design descriptions and the usual textual syntax is used for program code.

Freja is part of the Mjølner tool and can initially be reached using the **Diagrams** menu.

5.7.2 Q02) What does the name Freja mean?

Many have wondered about the origins of the strange product names used for parts of the Mjølner BETA System. Due to the origin of the Mjølner BETA System, many of the components of the system bear Nordic names. These Nordic names originate from the Nordic Mythology, and are thus names within the common cultural background of people in the entire Nordic region. Freja is the name of the goddess of love. She lives in Folkvang and is the most beautiful of all women in Asgaard. She owns the golden piece of jewelry Brisingemen.

5.7.3 Q03) What are the important files?

There are two important file types when working with Freja: .bet and .diag files.

The .bet file is well−known to BETA programmers. The .diag file contains a representation of the diagrams including layout information and references to the corresponding beta code.

When running Freja there will also exist versions of the files with extensions "#" and "~". The "#" files are autosave versions that are used when recovery is called for. Autosave is performed each time the an operation altering the code is performed. The "~" files are versions corresponding to how the diagram looked when an explicit save was performed the last time.

Notice:

- Users are *not* encouraged to try to perform recovery by moving the "#" files themselves! Freja takes care of this on restart from an eventual crash; more on how to do this see Q05) How do I make recovery?
- Revert is currently not implemented in Freja. On how to do it "manualy" see Q06) How do I revert changes to the diagram?

5.7.4 Q04) How do I perform recovery?

Freja performs an autosave each time an operation that alters the code is invoked. To recover after the program has crashed you can do one of the following:

- Start the Mjølner tool and use **Open Diagram...** in the **Diagrams** menu to open the .diag file corresponding to the diagram you were working on (e.g. foo.diag).
- Start Mjølner and open the .bet file that corresponds to the diagram (e.g. foo.bet). If there is an autosave version of this file you will be prompted wether you want to recover. Answer "yes" to this question. Imediately after opening and recovering this file choose **Show Diagram** from the **Diagrams** menu. This will bring up a dialog asking you wether you want

to recover the autosaved .diag file. Answer yes to this question.

Explanation of the different file types used in Freja, see Q03) What are the important files?

5.7.5 Q05) How do I revert changes to the diagram?

Revert is currently not implemented in Freja, so if you wish to revert to the situation at the time of the last save it neccesary to manipulte the files yourself. To do this simply move .ast~ file(s) into the corresponding .ast file(s), move the .diag~ file into the corresponding .diag file and move the .bet~ file into the corresponding .bet file.

Notice:

If the diagram you wish to revert uses more than one .bet file, you *must* move all the involved .bet~ and ast~ files as explained above.

Explanation of the different file types used in Freja, see Q03) What are the important files?

5.7.6 Q06) How do I (re−)create a diagram through reverse engineering?

If no diagram is available for a piece of BETA code you can create a diagram using Freja's reverse engineering facilities. To do this open Mj'lner on the .bet file containing the code. Then choose **Show Diagram** from the **Diagrams** menu. Having opened a diagram on the code, one of two will appear:

1. If the opened file only contains one fragmentform, the Work Sheet will appear, showing a diagram containing the attributes visible in the outermost descriptor of the code.
2. If the opened file contains more than one fragmentform, the Group Page will appear, showing a diagram with entries for each fragmentform in the file.

In the first case the detailable attributes can be detailed to the wished level. If relations like e.g. inheritance connectors and references do not appear automatically during detail, it means that the program (the .ast) has not been checked. To do this, select the **Check** entry in Mjølner's **Compile/Run** menu (No nonterminals must be present in the code for the checker to be able to semantically check it). If no semantic errors are detected during checking, all relations will then automatically appear when checking is finished.

In the second of the above cases, you simply detail one or more of the fragmentform entries on the Group Page (e.g. by doubleclicking them) and then proceed as described for the first case.

5.7.7 Q7) Can I close a diagram?

No, in the current version of Freja closing of diagrams is not implemented. Several .bet files can be loaded into the same diagram though. If you wish to load another diagram however, you are forced to quit the application and start it up again on this other diagram.

5.8 PART II: Known bugs and errors

5.8.1 E01) Lists of names in declarations

Declarations containing lists names, like e.g:

```
a,b,c: @Integer
```

(as opposed to: `a: @Integer; b: @Integer; c: @Integer`)

will currently not be handled well by Freja and may cause serious instability of the program.

Workaround:
Avoid declarations of the above mentioned type in your problem domain code and use declarations of the type `a: @Integer; b: @Integer; c: @Integer` instead.

5.8.2 E02) Embedded associations and static descriptor SLOTs

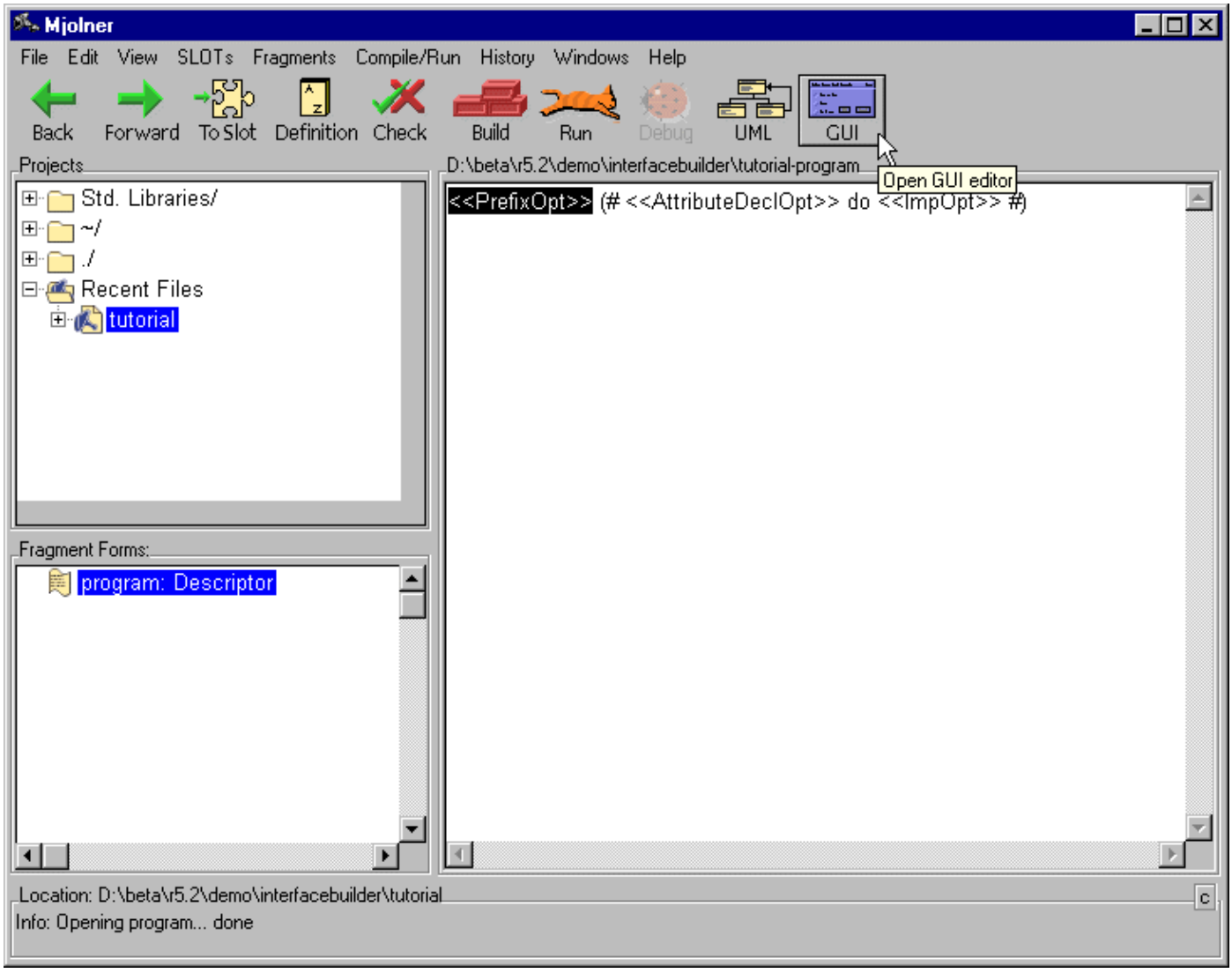
If the fragment contains embedded associations and static descriptor SLOTs (as f.ex. `private:@<<SLOT privateSlot:Descriptor>>`) Freja may crash when trying to display association relations.

Workaround:
Avoid static descriptor SLOTs in your problem domain model code.

6 GUI Builder

6.1 How to Get Started

Frigg is fully integrated with the sourcebrowser and structure editor. When Frigg is started, the sourcebrowser window is presented to the user. The 'Interfacebuilder' menu contains commands to create windows that can be edited in a graphical editor.



6.1.1 Interfacebuilder Menu

New Window Library

This command creates a new fragmentgroup that are setup to contain window definitions. The ORIGIN of the fragmentgroup are the 'guienvall' fragmentgroup, which includes all of GUIenv (Lidskjalv), the Mjølner BETA user interface framework. The fragmentgroup has a 'GUIenvLib: attributes' fragment, which therefore can contain specializations of the window pattern from Lidskjalv. Furthermore a BODY fragmentgroup is automatically created. This fragmentgroup will contain the private attributes of the window definitions.
A file dialog prompts for a filename which will be the name of the new fragment group. Assuming the name 'foo' is used, the BODY group will have the name 'foobody'.

New Canvas Library

This command does almost the same as the 'New Window Library' command. The created fragmentgroup will have a 'windowLib: attributes' fragment, which can contain canvas specialisations of the window pattern from Lidskjalv.

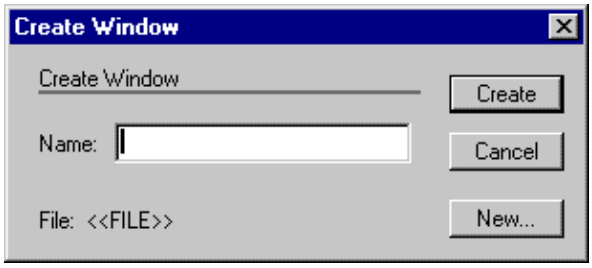
Edit object

Selecting "edit" will open a graphical editor on the window pattern that are currently selected in the structure−editor. The whole pattern definition must be selected. It is not sufficient to just select the name.

Note: If the fragmentgroup is not checked, it is not always possible for Frigg to recognize the selected pattern as a user interface object. If the 'Edit' command is disabled, the fragmentgroup should be checked via the 'Check' command in the 'Tools' menu.

Create

A specialization of window or canvas is generated in the currently selected lib fragment, dependent of the name of the fragment. If the current fragment is a GUIenvLib fragment, a window will be generated. If it is a windowLib fragment, a canvas will be generated. The following dialog are popped up:

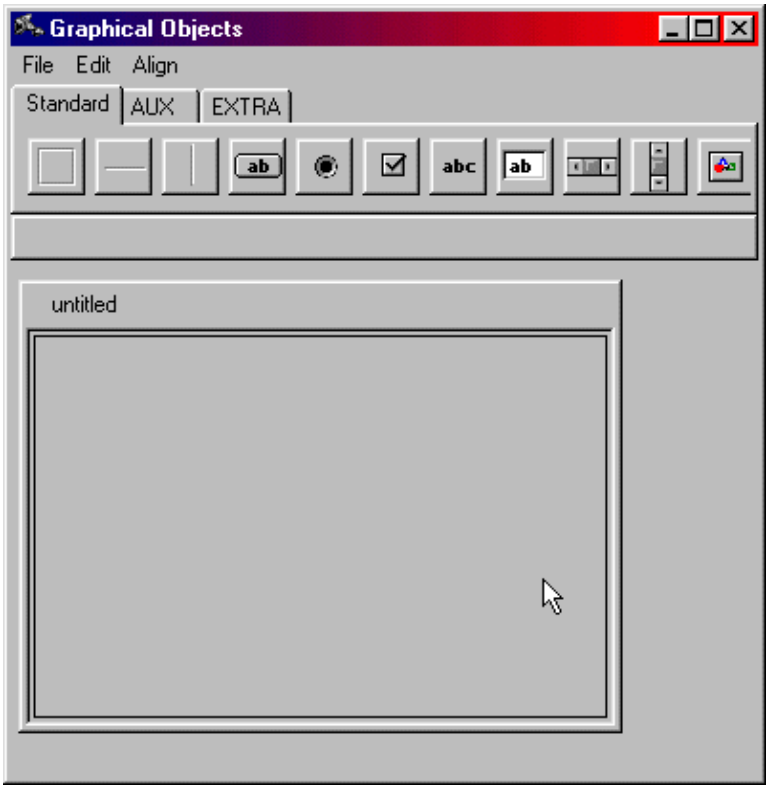


In the 'name' field the name of the new window (or canvas) should be typed in. The name is expected to be a proper BETA identifier, since this is the name the pattern will given in the source code.

A graphical editor is opened on the newly created object.

6.2 Graphical Editor

The graphical editor is the part of interface builder which allows the user interface to be constructed interactively by direct manipulation of Lidskjalv objects. The figure below shows the graphical editor on a newly created object.



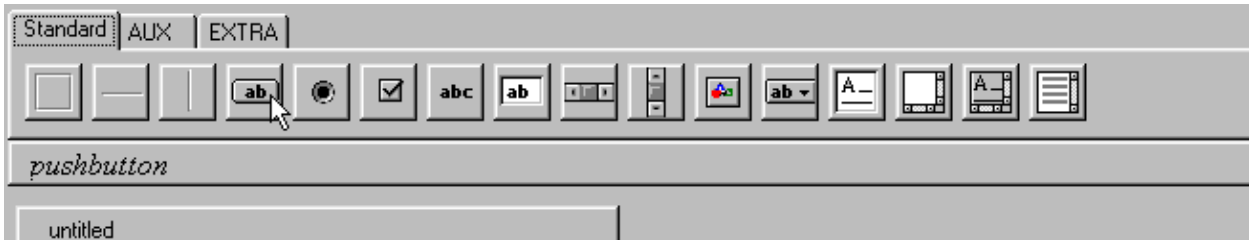
The menubar consists of three menus: File, Edit and Align. There are two areas inside the graphical editor: The object palette, and the contents area.

6.2.1 Creating Items

Items are added to the window by utilizing the palette. An instance of some item on a palette is created by dragging the item and placing it in the contents area. The canvas pattern has a border and contains other items in its local coordinate system.

When an item is dragged from the palette, the border of the receiving canvas is highlighted to indicate in which canvas the item will belong.

As seen in the figure below the pattern in Lidskjalv, corredponding to the button is posted below the palette.



These different patterns are described in the Lidskjalv reference manual.

6.2.2 Moving and Resizing

The items in the graphical editor can be moved and resized simply by dragging with the mouse. If the item is grabbed in the interior the item is moved, if it is grabbed near the border it is resized.

6.2.3 File Menu

Close

Closes the graphical editor.

6.2.4 Edit Menu

Undo

All the operations in the graphical editor can be undone by choosing the "Undo" command in the edit menu. The undo is multilevel, which means that all changes can be undone all the way back to when the window was opened for editing in the session.

Redo

A sequence of undo−commands can be redone by invoking the redo command in the edit menu − as long as no other operation has been performed after the undoing.

Cut

A copy of the selected objects is placed on the clipboard along with the underlying BETA code, and then deleted from the window. The objects can then be pasted into the window again. It is possible to copy and paste between graphical editors.

Copy

A copy of the selected objects is placed on the clipboard along with the underlying BETA code. The objects can the be pasted into the window again. It is possible to copy and paste between graphical editors.

Paste

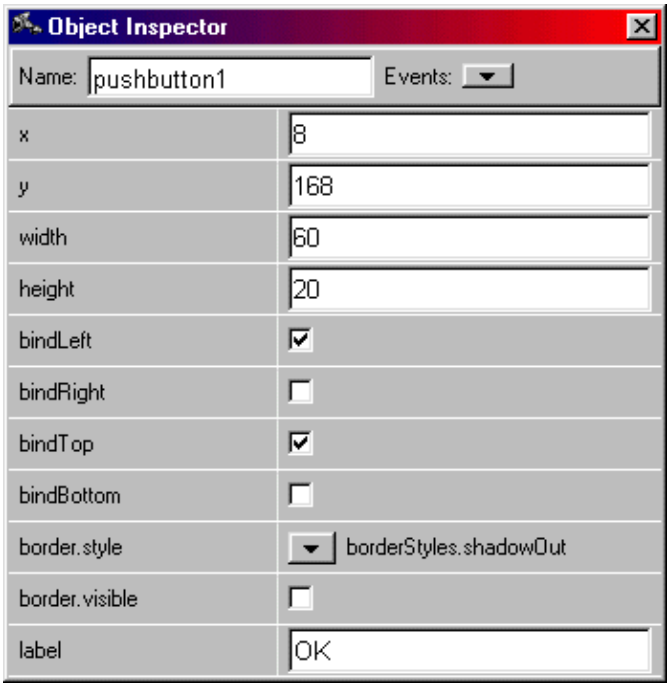
A copy of the selected objects is placed on the clipboard along with the underlying BETA code. The objects can the be pasted into the window again. It is possible to copy and paste between graphical editors.

Delete

The selected objects along with the underlying BETA code are deleted without affecting the clipboard.

Object Inspector

The object inspector dialog allows the layout properties of the objects in the graphical editor to be edited in a dialog form in contrast to direct manipulation. Most properties can only be changed via the dialog. The content of the dialog depends on the selected object. The basic info dialog has the properties that are common to all objects.



Name: The name are expected to be a proper BETA identifier, since this is the name the pattern will given in the source code.

Events: In the Events menu the different event types of the selected object are listed. When the user clicks on an object, the virtual procedure "onMouseUp" are executed in that object. There is a virtual procedure for each event that the object may receive. Selecting an event, will open a structure editor (sif editor) on the virtual procedure corresponding to the selected event type. If the specialization of the virtual procedure does not yet exist, it is first be created.

Position(x, y) and Size(width , height): The position and the size are most easily changed via direct manipulation, but sometimes it may be easier to enter the precise values via the dialog.

Constraints: The constraints control how the object reacts when the surrounding object are resized. If the bindleft is false and bindRight is true, the object will follow the right edge of the surrounding object without stretching. If bindLeft is true and bindRight is true, the object will stretch etc.

Border: All objects have a border. The visibility of the border can be controlled via the "visible" check box. There is different kinds of shaded border types, that can be selected in the "style" popup menu.

Show Source Code

Opens a structure editor (sif editor) on the code that corresponds to the selected object.
Grid
Makes an invisible grid, which grabs the object being draged into the window. **Fit to Contents**
Frequently, there is a natural size of an object, dependent of the content of the object. For example a StaticText object would have the extent of the text as the natural size. The "Fit to Contents" command in the edit menu will adjust the size of the selected object to its natural

size. Not all objects have natural sizes.

6.2.5 Align Menu

The alignment commands in the alignment menu supply facilities to align objects in a row or centered underneath each other etc. The alignment commands work on the current selection. The first object selected will stay where it is.

These alignment commands are available:

Align left side

Aligns the left sides of the selected objects to the first selected object.

Align right side

Aligns the right sides of the selected objects to the first selected object.

Align top edge

Aligns the top edges of the selected objects to the first selected object.

Align bottom edge

Aligns the bottom edges of the selected objects to the first selected object.

Align vertical center

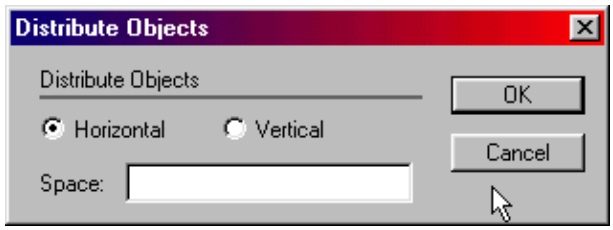
Aligns the vertical centers of the selected objects to the first selected object.

Align horizontal center

Aligns the horizontal centers of the selected objects to the first selected object.

Spacing...

This command will present a dialog that allows a group of objects to be given the same size or the objects to be arranged, so the distance between any two adjacent objects are the same.



To give a group of objects the same vertical distance do the following:

1. Select the objects that should be given the same vertical distance
2. Choose the Spacing... command
3. Check the "Vertical" check box and make sure the other check boxes are unchecked
4. Type the desired distance into the field next to the check box
5. Press OK

☞**Notice**: The objects will keep their vertical and horizontal order in the window when adjusting the distance. Furthermore, the top left object will always stay where it is.

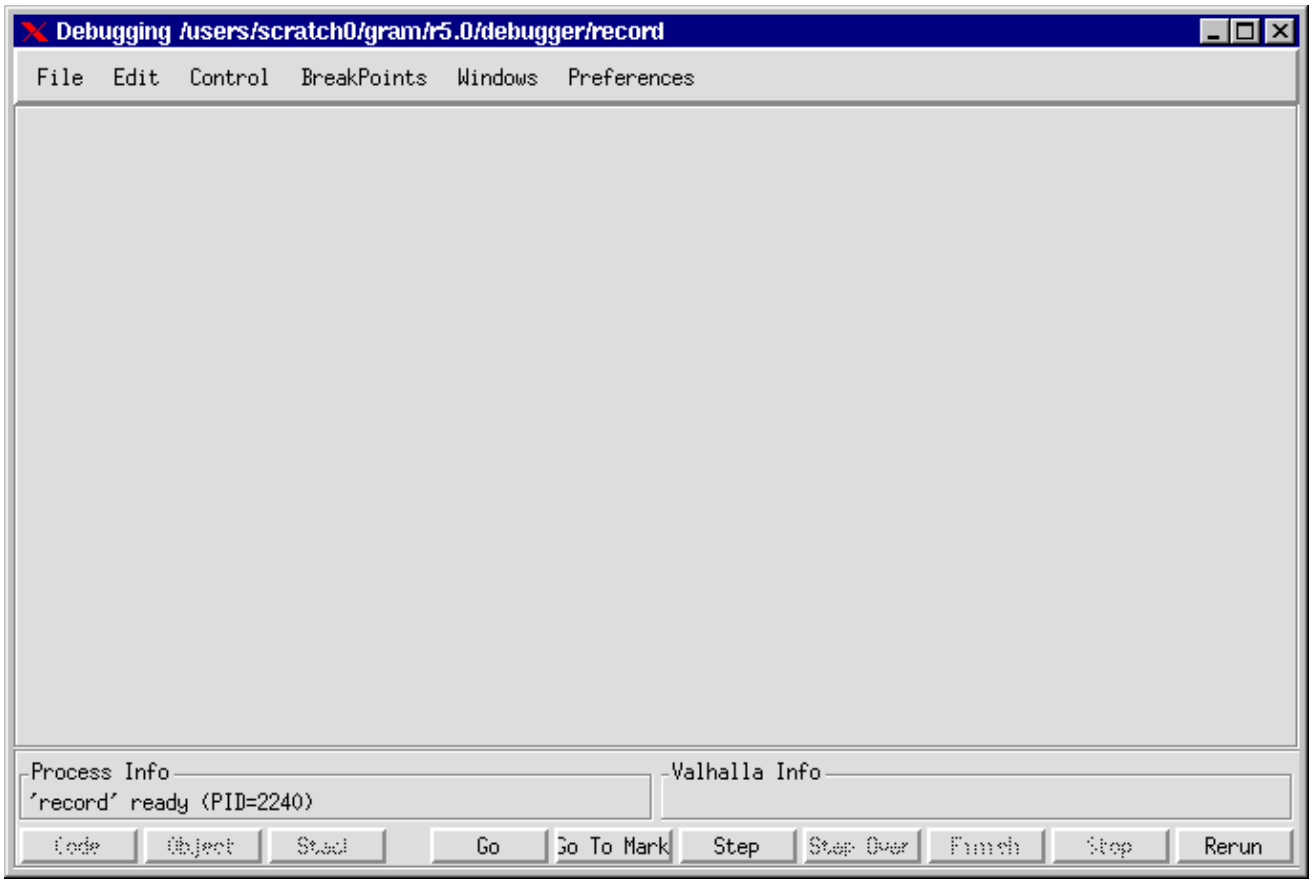
7 Debugger

In order to debug a program open the program in the mjolnertool.
Choose the fragmentgroup containing the program descriptor. Often this file has the same name as your executable. Choose "Debug" in the "Compile/Run" menu. You can only debug a program if you have an executable, therefore you must compile your program before debugging.

The debugger performs a check of all files used in the program, therefore it can take from a few seconds up to a minute to open the debugger on a large program.

When finished initialising, Valhalla opens its Valhalla Universe containing a number of menus. The figure shows the look of the Valhalla Universe when using Valhalla on the example program from the tutorial.

Figure 8: The Valhalla Universe



The top pane of the Valhalla Universe contains the menus. The middle pane contains the different views, and the bottom pane contains two info areas and a number of buttons.

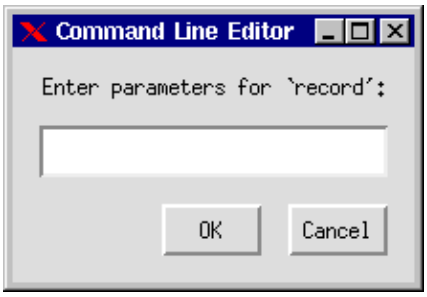
Editing is disabled during debugging. This means that if you eventually discover a bug in your program and want to correct it, you must close Valhalla. Do this by choosing "Close" in the "File" menu. After editing a new compilation is required.

7.1 Command Line Arguments to the Debugged Process and Command Line Editor

Some applications demand command line arguments in order to execute properly, and others allow command line arguments when being executed. In order to support debugging of such applications, Valhalla have facilities for handling command line arguments to the debugged process.

Command Line arguments are specified through the Command Line Editor.

Figure 9: The Command Line Editor



Here you can edit and specify the command line arguments to the debugged process before they are handed to the debugged process.

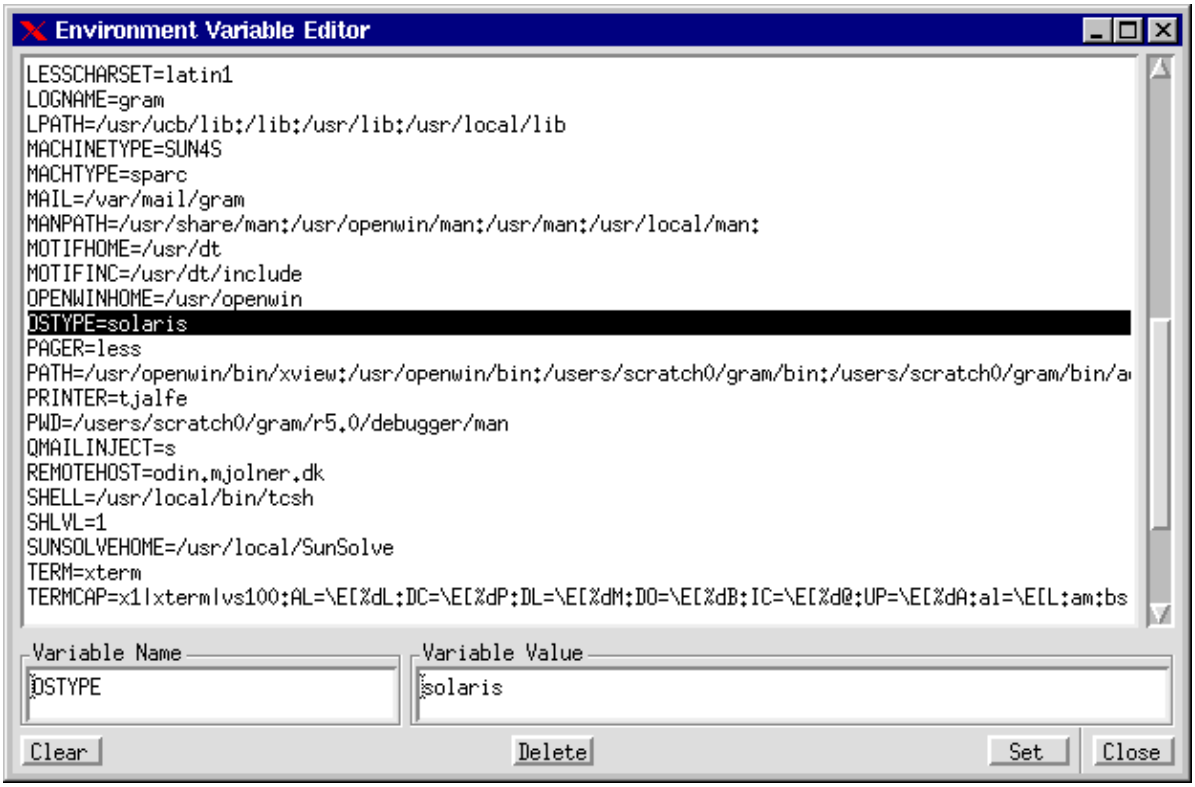
At any time during the debugging of the application, you can invoke the Command Line Editor to edit the command line arguments (by selecting *Command Line...* in the *Edit* menu). Editing the command line arguments does, however, not directly affect the process being debugged. The changes are only reflected to the debugged process when it is rerun.

7.2 Environment Editor

The debugged process is executed as a separate process, and is executed in an environment that is a copy of the environment, that Valhalla is executing in. However, in some cases, the debugged process demands special values in some of the environment variables, or it may demand some additional environment variables to be set.

In order to operate on the environment of the debugged process, the Environment Editor is available (by selecting *Environment...* in the *Edit* menu):

Figure 10: The Environment Editor



The upper pane is a scrolling list containing a name−value pair for each environment variable in the environment of the debugged process. By selecting one such pair, it is also visible in the lower pane, in the two edit fields. To the left is the name, and to the right the value. By editing the value, you can specify a new value to the environment variables. By pressing the Set button, the new value is defined.

If you wish to define a new environment variable, you can just specify its name in the left text field, and its value in the right text field, and it will then become defined when the *Set* button is pressed.

If you wish to remove the definition of an environment variable, you can select the variable from the list and press the *Delete* button.

Pressing the *Clear* button will remove any contents in the two edit fields.

Changes to the environment are not directly reflected in the debugged process, but will be reflected when you rerun the debugged process.

7.3 Interaction with Views

Views are nested inside the Valhalla Universe. Each view has a header consisting of an iconify button and a name field.

The different views share many interaction properties, that are described in this section.

Clicking in the name field puts the view on top of all other views. Clicking and dragging the name field moves the view around.

The view may be resized by placing the mouse cursor on the borders of the view (the cursor changes to a cross), and then click−and−drag to resize.

Each view has an associated menu, that is popped up by clicking the right mouse−button in the name field of the view. The contents of that menu depends on whether the type of the view. However, the menu contains in all cases *Close* and *Iconify*. *Close* closes the view, and *Iconify* iconifies the view. By selecting *Deiconify* in the menu popped up by clicking the right mouse−button on the icon, the view is displayed in full again.

7.3.1 Closing Multiple Views

By dragging a rectangle in the middle part of Universe, multiple views are selected (all views contained in the selected rectangle). If you then choosing *Close* in the *Edit* menu of the Universe, the selected views are all closed in a single operation.

7.3.2 View Shortcuts

- leftmouse−click on the iconify button in the upper left corner, iconifies the view. This corresponds to selecting *Iconify* in the name field popup menu.
- leftmouse−double−click on the icon, displays the view in full again.
- shift−control−leftmouse−click in the name field or on the icon closes the view.
- rightmouse−press on the name field or on the icon results in the view menu being popped up.
- leftmouse−drag on the name field or on the icon makes it possible to move the view or icon.
- leftmouse−click in the view name field or icon selects the view.
- Rightmouse−press on the individual lines inside the view pops up the item menu (different in the different view types).

7.3.3 View Outline

If *Outline On Open* is enabled, an outline is shown initially before a new view is opened in the Universe.

While outlining a view, the following actions are possible:

- leftmouse−drag inside the outline makes it possible to move the outline around.
- leftmouse−press outside the outline moves the closest corner of the outline to the position of the mouse (resizing the outline). Further dragging will continue the resize. When the mouse is released, the size and position of the outline is defined, and the new view will be opened in that size, and positioned accordingly.

7.4 Controlling the Execution

When an application is being debugged by Valhalla, it is executed as a separate process, under the control of Valhalla. This gives the user of Valhalla extensive control over the debugged process.

7.4.1 Interrupting the debugged process

If the debugged process executes, and you wish to interrupt the execution, you can always hit the *Stop* button in the Buttons Area of the Universe (or select the *Stop* item in the *Control* menu of the Universe). This will immediately interrupt the debugged process, making it possible to inspect it (the

state of objects, the current stack, etc.).

7.4.2 Setting Breakpoints

Breakpoints are naturally associated with the imperatives of the BETA program, and acts as points in the debugged process, where Valhalla is given some control over the execution of the debugged process. Valhalla supports three types of break points:

- Break: If a Break point is reached during the execution of the debugged process, the process will be interrupted, and Valhalla is now able to inspect the current state of the application.
- Oneshot: which is a Break point, that is only effective once (will automatically be removed, when it have been reached the first time). Otherwise it functions exactly as a Break point.
- Trace: is used to trace the execution without actually interrupting. A trace point have associated a text string. Each time a trace point is reached during the execution of the debugged process, the associated text will be printed on standard output, and the execution will be continued immediately.

Valhalla supports that break points are positioned in two positions, relative to an imperative:

- Before: when a break point is positioned before an imperative, it will be activated immediately before the imperative will be executed.
  A Before break is shown visually in the code views by >>*n*>>, where *n* is a sequence number of the break point.
- After: when a break point is positioned after an imperative, it will be activated immediately after the imperative have been executed. The only places, where using After position is the only possible solution, is when one wish to interrupt the application immediately before it reaches the terminating #) of a descriptor. In all other cases, we could just as well have placed the break point before the imperative following the one selected.
  An After break is shown visually in the code views by <<*n*<<, where *n* is a sequence number of the break point.

An important aspect of Oneshot and Break break points is, that the object− and stack views that are visible in the Valhalla Universe, are automatically updated to reflect the current state of the objects and the stack at the time of the break. Should the object shown in an object view have become inaccessible (and therefore reclaimed by the garbage collector), this fact is reflected in the object view by clearing the object view, and changing the label of the view.

7.4.3 Unsetting Breakpoints

Breakpoints may be unset essentially in the same way as they have been set.

7.5 The Universe Menus

The Valhalla Universe defines a number of menus, which will be described in this section. The buttons in the Buttons Area are merely short−cuts for some of these menu items, and will therefore not be described separately.

7.5.1 File menu

Source Browser...
This menu item will result in a source browser being displayed (or, if a source browser is already opened, it will be raised (and wriggled).

Quit
Selection this menu item will terminate the debugged process, and Valhalla will thereafter be terminated too.

7.5.2 Edit menu

Command Line...
Selecting this menu item will open the Command Line Editor. Using this Editor, you can edit the command line arguments, that will be handed to the debugged process when it is forked as a separate process by Valhalla. Note, that changes to the command line arguments will not have any effect on the debugged process, if it have been forked. However, by rerunning the debugged process, changes to the command line arguments will be reflected to the debugged process.

Environment...
Selecting this menu item will open a Environment Editor. This editor makes it possible to edit the values of the environment variables (and define new environment variables) for the debugged process. As for command line arguments, these changes will only affect the debugged process after rerunning it.

Refresh
If the Universe for some reason seems corrupted, it might help to refresh the Universe. This will totally redraw the Universe, including the views, and the contents of these views.

Close
This will close the currently selected view(s).

7.5.3 Control menu

Below the items of the Control menu of the Valhalla Universe are described. These are concerned with controlling the execution of the debugged process.

Go
The debugged process resumes execution until it hits a breakpoint, receives a signal, terminates or a runtime error is detected by the BETA runtime system.
Also available as *Go* button in the Buttons Area.

Step
The Step command makes the debugged process resume execution until it have either executed one single BETA imperative, or it have stepped into some routine. This is the basic step of the debugger (single stepping).
Also available as *Step* button in the Buttons Area.

Step Over
The *Step Over* command makes the debugged process continue execution after setting a temporary breakpoint at the next BETA imperative in the code currently being executing. *Step Over* is only available if the debugged process is currently stopped in some BETA code (as opposed to being stopped during the execution of code written in C or some other language), as 'next BETA imperative' has no local meaning otherwise..
Also available as *Step Over* button in the Buttons Area.

Stop
Stops the debugged process.
Also available as *Stop* button in the Buttons Area.

Rerun
Kills the debugged process and restarts the program in a new process. Breakpoints already

set continues to be set. Otherwise the state of the debugged process will be as if it just started execution. The command line arguments and the environment variables will be identical to the previous execution, unless they have been edited since last (re)run. Also available as *Rerun* button in the Buttons Area.

Kill

Kills the debugged process.

7.5.4 Breakpoints menu

This menu is the "bread and butter" in controlling the debugged process. It is by means of the entries in this menu, that you can make the debugged process stop at specific places in the code (or output trace information at these points). You can also gain access to all defined breakpoints through this menu, and you can save the breakpoints, to be able to load them into the debugger in a later debugging session. There are three types of breakpoints: Break, Oneshot, and Trace, and breakpoints may be places either before or after an imperative.

Go Until Mark

If you have selected an imperative in a code view, and select this menu item, then Valhalla will instruct the debugged process to resume execution until the execution reaches the selected imperative.

Set Break

This will set a break point before the selected imperative in the code view. Each time execution reaches this point, the debugged process will be suspended, and all views open in the Universe will be updated. This implies that the stack view and all open object views will display the current state of the stack (respectively objects) at the points of the break.

Set OneShot

This will also set a breakpoint before the selected imperative in the code view, but a Oneshot break point will be removed automatically when it is reached the first time (i.e. the debugged process will only be suspended at this point one single time).

Set Trace...

Selecting this menu item will display a small dialog, in which you can specify a text string to be associated with the trace point. Valhalla will then insert a trace point before the selected imperative in the code view. When execution reaches a trace point, the associated text string will be printed, and execution immediately resumed.

Set Break After

Similar to Set Break, but will set the break point after the selected imperative in the code view.

Set OneShot After

Similar to Set OneShot, but will set the Oneshot point after the selected imperative in the code view.

Set Trace After...

Similar to Set Trace..., but will set the Trace point after the selected imperative in the code view.

Erase Break

If a break point is placed before the selected imperative in the code view, it will be removed.

Erase Break After

If a break point is placed after the selected imperative in the code view, it will be removed.

Breakpoint List

Through this menu item, you can gain access to all break points, that are currently set in the debugged process. These breakpoints are all accessible through the submenu, attached to this menu item. By selecting a given break point from the submenu, the source code with the selected break point will be displayed in a code view.

Breakpoints: Save

This menu item enables you to save the current set of break points onto a file (specified

through a file dialog).

Breakpoints: Load

This menu item enables you to load a previously save set of breakpoints into the debugged process from onto a file (specified through a file dialog). This is only a safe operation, if first of all, it is the same application, and secondly no changes have been made to any of the source files in which any saved breakpoints appear. If one of these two conditions are not satisfied, this operation is not guaranteed to give any sensible results. Currently no check are implemented in Valhalla to test the legality of this *load* operation, and it should therefore be used with care (obeying the above conditions).

7.5.5 Windows menu

Through this menu, you can open views into the current state of the debugged process, i.e. inspecting state and source code of the currently executing object and component, and inspect the runtime stack of the debugged process.

Current Code

This will open a code view, displaying the source code, that was executed at the point of the break.

Also available as *Code* button in the Buttons Area.

Current Object

This will open an object view, displaying the state of the object that was executing at the point of the break.

Also available as *Object* button in the Buttons Area.

Current Component

This will open a component view into the component, that was executing at the point of the break.

Active Stack

This will display the contents of the execution stack at the point of the break.

Also available as *Stack* button in the Buttons Area.

7.5.6 Preferences menu

This menu gives a number of possibilities for setting preferences, defining the behaviour of the different parts of Valhalla. All menu items are toggle items, such that the preference is enabled if a check mark is visible to the left of the corresponding menu item.

One Line Char Repetitions

If enabled, char repetitions will be displayed as a text string. Otherwise, char repetitions will be displayed as other repetitions (i.e. on multiple lines with one index and the corresponding value on each line).

Invisible Origins

If enabled, Origin fields of objects will not be displayed.

Fast Browse Mode

If enabled, following dynamic object references will display the state of the referenced object in the same object view (replacing the existing contents). If disabled, following dynamic object references will create new object views for the referenced object.

Number Objects

If enabled, object references in object views will be specified with both the name of the pattern from which the object is instantiated and a sequence number. The sequence number helps in identifying easily that two object references refer to the same object. If

disabled, the sequence numbers are not displayed.

Short Object Names
If enabled, object names in object views will be given in short form, leaving out information on the location of the pattern, from which they are instantiated. If disabled, the object names will contain this information.

Outline on Open
If enabled, opening a view in the Universe will be done interactively by the user by dragging an outline, defining the size and position of the new view. If disabled, Valhalla will define the size and position of the new view.

Show Current Code on Stop
If enabled, Valhalla will automatically display the source code, that was being executed at the time of the break.

Short Code Names
Similar to *Short Object Names*, just for code views.

Read Labels on Fork
If enabled, Valhalla will read the labels in the executable at the time of the fork. Otherwise, reading the labels will be postponed.

Debug Valhalla
Internal debugging facility for Valhalla.

7.6 Browsing through BETA Code

7.6.1 Inspecting the current code

When the debugged process is stopped in some BETA code, and no code view is opened on that code, selecting *Current Code* from the *Windows* Menu of the Valhalla Universe opens a code view with the current BETA imperative selected. If a code view on that code is already opened, it is raised (and wriggled), and the current imperative selected.
You can also press the "CODE" button in the Valhalla Universe, this has the same effect.

7.6.2 The code view

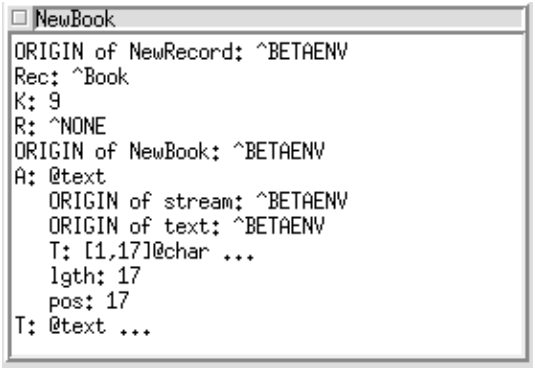
A code view displays an abstract textual description of a BETA object descriptor. Abstract means that nested object descriptors are displayed as three dots. Using the facilities of a code view it is possible to set breakpoints, perform expression searches, or simply inspect the textual description of the pattern.

Anywhere inside the code view, you can get access to the *Breakpoints* menu, simply by holding the right mouse button down. This will pop−up the *Breakpoints* menu.

We refer to the Source Browser Reference Manual in the Mjølner Tool manuals ([MIA 99−39], [MIA 99−40], [MIA 99−34]), for further details of the source browser facilities.

7.7 The Object View

Figure 11: The Object View



The state of complex objects is displayed in object views as the result of selecting *Current Object* from the *Windows* menu, double−clicking an object reference in a stack view or double−clicking a line in another object view.

The name of the pattern of which the object is an instance is displayed as the name of the object view window. The pattern name is prefixed by the origin chain from the pattern to the fragment in which the pattern is declared (if *Long Object Names* is enabled). For example, an object view having the name Object: lib.x.y displays an instance of the pattern y, declared nested in the pattern x contained in the fragment lib.

Each line in an object view corresponds to either a static or dynamic reference contained in the object. Chars, Booleans, Integers and Reals are displayed directly by value whereas attributes of more complex type are described by their pattern name.

When the current object is a "do−part−object" the surrunding object will be displayed instead.

7.7.1 Browsing objects:

The object browser uses abstract presentation of the objects presented. This means that nested part objects are initially shown contracted, i.e. as three dots. By double−clicking a line of the object view ending in '...', the hidden details will be shown. By double−clicking the same line again, the details are hidden.

Each line in the object view corresponds to some attribute of the object. Simple attributes (@Char, @Integer, ...) cannot be further detailed, whereas other kinds of attributes can. Default when double−clicking some attribute is as follows:

- Dynamic references: A new object view is opened on the referred object. If some object view on that object is already open, visual feedback (wriggling and highlighting) will signal this fact. The same behaviour goes for origin references as well.
- Static references: If the attribute is contracted, double−clicking shows an extra level of detail in the same window. Otherwise the attribute is contracted.
- Repetition references: If the repetition is contracted, double−clicking unfolds the repetition by showing a line for each index in the repetition. Otherwise the repetition is contracted.
- Pattern references: Currently a more detailed view on pattern references is not implemented.

7.7.2 Object View menus:

In addition to the default menu items for all views (described above), an object view has a number of additional items in the name field popup menu, i.e. the menu popped up by clicking the right

mouse−button in the name field of an object view. These are described in turn below:

Fit to contents
Tries to resize the object view to a reasonable size.

Show Attribute
This nested menu is a list of object attributes that are not currently visible. By default this list includes the origin attributes. By selecting an entry in this menu, the corresponding attribute is made visible. It may later be re−hidden as explained below.

Fast Browse Mode
Default when double−clicking dynamic references is to open a new object view showing the object referred. By selecting "*Fast Browse Mode*", the default is changed into showing the object referred in the same window, replacing the previous contents. This allows for fast browsing without opening unnecessary object views. Default may be reset by choosing "*Fast Browse Mode*" again.

Go Back
The browser maintains a stack of objects visited during Fast Browse Mode. By selecting "*Go Back*", the previous object on that stack in reshown in the current window.
When clicking the right mouse−button inside an object view, a sligthly different menu than the name field menu pops up. The entries of this menu, the "attribute menu" are as follows:

Open Separate
Default when double−clicking static references is to detail the corresponding attribute in the current window. Alternatively one may single−click the static attribute and then select "*Open Separate*". This opens a new object view on the part object.

Open inline
If a static reference attribute is double−clicked in order to detail the view, and another object view on that part object is already open, the existing object view is highlighted, and the attribute not detailed. If one wants to detail in the current window anyway, one may either close the existing object view, or single−click to select the attribute and then choose "*Open inline*" from the object state popup menu.

Contract Attribute
Has the same effect as double−clicking a complex attribute that is already detailed. I.e., the attribute is contracted.

Hide Attribute
By single−clicking an attribute in the object view and then selecting "*Hide Attribute*", the attribute is hidden. Note that "contracted" and "hidden" is not the same thing. Contraction replaces a complex attribute by a single line ending with '...'. Hiding an attribute means moving it completely out of sight.

Show Attribute
This entry corresponds to the menu entry with the same name in the name field popup menu. However, instead of showing a list of hidden attributes for the main object, a list of attributes hidden in the currently selected complex attribute is shown. I.e., to show a hidden attribute of a nested part object, single−click that part−object and then choose "*Show Attribute*" from the object state popup menu.

Open Evaluator
This entry appends an evaluator field to the objectview.
The evaluator field is a code−editor. When you press the "Evaluate"−button is the code in the editor compiled and linked into the debuggee and executed. The code in executed in the context the object, corresponding to the view.

Figure 12: The evaluator

7.8 The Stack View

Figure 13: The Stack View

A stack view is opened by choosing *Current Stack* from the *Display* menu of the Valhalla Universe. The stack view shows the current runtime stack of the debugged process. Note that the stack view only shows stack−frames of the currently attached component.

- BETA stack frames displayed as fragmentName: PatternName. Double−clicking on a BETA stack frame opens a code view showing the code referred.
- C stack frames. If an external procedure calls back to BETA, the part of the runtime stack used between the external call from BETA to C and the callback from C to BETA is shown abstractly as [C Stack Frame].
- Components. In the buttom of the stack is a list of the currently existing components.

By double−clicking on the BETA stack frame lines, a code view is opened, displaying the specific code of this stack frame (with the executed imperative being selected). By right mouse button hold, you gain access to the stack frame object menu. By selecting the entry, you gain access to the object executing the stack frame. By right clicking on a component you can select either the calling components stack or the calling components calling object. The former will open a stack view and the later a object−view.

7.8.1 The StackBrowser

The StackBrowser is opened when you select "Stackbrowser" in the "Windows" menu.
The StackBrowser combines a Object−View, a Code−View and a Stack−View in one window.

Figure 14: The StackBrowser

When you select a frame in the stackview the corresponding code and object is shown in the other views.

7.9 BOOLEAN OPTIONS

The following boolean options can be set by selecting "Preferences" in the file menu in the MjolnerTool. In the debugger tab you can set/unset:

- Short code names
  Whether to use truncated imperative names for code expressions shown on the stack.
  Default: TRUE.
- Read labels on startup
  Whether to read labels from executable at startup instead of on demand.
  Default: FALSE.
- Debug Valhalla
  Print out Valhalla debugging information.
  Default: FALSE.
- Show current code on stop
  Show current code on each process stop, even if no view is currently open on that code.
  Default: TRUE.
- Show char−reps on one line
  Whether char repetitions should be shown on a single line.
  Default: FALSE.
- Hide Origins
  Whether origins should initially be invisible in object views.
  Default: TRUE.
- Give objects serial numbers
  Whether objects should be given a serial number.
  Default: TRUE.
- Short object titles
  Whether to use truncated pattern names (max 2 pattern names in path) for object descriptions.
  Default: TRUE.
- Drag outline on open
  Whether a nested window is dragged on open, or is simply given a default size and position.

Default: TRUE.
- Wriggle On Open
  Whether to wriggle windows when (re)opened. If unchecked windows is only raised.
- Dynamic Compilation
  If checked dynamiccompilation is enabled.

7.10 Valhalla Environment Variables

Valhalla recognises the following environment variable:

- *VALHALLAOPTS*
  May contain command line options to Valhalla as defined above. Command line options given on the command line are given priority over command line options specified in the *VALHALLAOPTS* environment variable.

7.11 Other Issues

This section contains a number of issues not yet touched upon.

7.11.1 The BETART environment variable

The BETART environment variable is described in BETART. It may be used to control the behavior of a BETA program in several ways. To set BETART for a BETA program being debugged without affecting Valhalla [3], set the BETART environment variable either through the Valhalla command line option *ENVIRONMENT*, or using the Environment Editor.

7.11.2 Tracing garbage collections

If you wish to trace the garbage collection in the debugged process, you can do this by setting the BETART environment variable for the debugged process. See [MIA 99−36].

7.11.3 Known bugs and inconveniences

- In some cases (especially on SUN 4) the stack view does not display the full runtime stack, forgetting a number of entries. Likewise, the stack view may get confused if the debugged process is stopped while executing C code. This also has implications for Valhalla when deciding where a runtime error occurred, and might result in the wrong place being pointed out.
- When a complex attribute has been detailed, the scrolllist implementing the view may enter an "auto−scroll" mode. Click some attribute to switch off this mode.

---

[3] Valhalla is a BETA program itself

7.12 Appendix A

This appendix contains the source code for the BETA program used as example in the tutorial. The

source consists of the files record.bet and recordlib.bet.

Program 1: record.bet

```
ORIGIN '~beta/basiclib/betaenv';
INCLUDE './recordlib';
--- PROGRAM: descriptor ---
(# (* This fragment is an example of using virtual patterns in BETA.
    * The following classification hierarchy is defined in the
    * library fragment 'recordlib'
    * 1.  Record
    * 2.  Person
    * 3. Employee
    * 3. Student
    * 2. Book
    * Record has a virtual (procedure) pattern:
    * Display, that displays the attributes of a record.
    * Display is further bound in the sub-patterns.
    * The patterns NewRecord, NewPerson, etc may be used for
    * generating new instances of the record-patterns.
    * The pattern Register has two virtual attributes:
    * regCat:< record, the elements in the register.
    * Display, that displays the elements of the register.
    *
    * In this application two instances of Register are generated:
    * Greg is a general register where regCat is not further bound
    * Ereg is an employee register where only Emplyoee-records may
    * be inserted
    *)

   (**** The following are declaring a number of reference variables ****)

   Birger, PeterA, Bible, LarsB, ClausN, ElmerS, KimJ: ^Record;
   Greg: @Register; (* Greg is a general register for all types of Records *)
   Ereg: @Register (* Ereg is a specific register only for Employee records
                      (and subpatterns of Employee) *)
     (# RegCat::< Employee; (* a further binding of RegCat to restrict to
                               Employee *)
        Display::< (# (* Displays the restriction on this particular register *)
                    do '/Employee ' -> s.putText; INNER #);
     #);
do (* the following creates and instantiates a number of Record objects *)
   (0 ,'Bimmer Moller-Pedersen','Unknown',- 200,'Piccolo')
     -> NewEmployee -> Birger[];
   (1,'Peter Andersen','male','missing M.D.')
     -> NewStudent -> peterA[];
   (2,'Lars Bak','male',1000000,'Garbage collector')
     -> NewEmployee -> LarsB[];
   (3,'Claus Norgaard','male',1000010,'Senior Coder')
     -> NewEmployee -> ClausN[];
   (4,'Elmer Sandvad','male',1000050,'Senior Supporter')
     -> NewEmployee -> ElmerS[];
   (5,'Kim Jensen M|ller','male',999990,'Painter')
     -> NewEmployee -> KimJ[];
   (9,'Kristensen et al.','Object Oriented Programming in the BETA programming language'
     -> NewBook -> Bible[];
   (* the following displayes the Birger, PeterA and Bible objects on the
    * screen *)
   screen[] -> Birger.Display; (* not a nice view (: -) *)
   screen[] -> PeterA.Display;
   screen[] -> Bible.Display;
   '==========================' -> putLine;
   (* initialization of the Greg and Ereg registers *)
```

```
   Greg.init;
   Ereg.init;
   (*inserts all Record objects in the Greg and/or Ereg registers *)
   Birger[] −> Greg.insert;
   Bible[] −> Greg.insert;
   PeterA[] −> Greg.insert;
   ClausN[] −> Ereg.insert;
   LarsB[] −> Ereg.insert;
   ElmerS[] −> Ereg.insert;
   KimJ[] −> Ereg.insert;
   (* displays the Greg and Ereg registers on the screen *)
   screen[] −> Greg.display;
   screen[] −> Ereg.display;
   (if (LarsB[] −> Ereg.has) (* test if LarsB is in the Ereg register *)
    // true then 'LarsB in employee register' −> putLine
    // false then 'LarsB not in employee register' −> putLine
   if);
   (if (LarsB[] −> Greg.has) (* test if LarsB is in the Greg register *)
    // true then 'LarsB in general register' −> putLine
    // false then 'LarsB not in general register' −> putLine
   if);
   (* end of program *)
#)
```

Program 2: recordlib.bet

```
ORIGIN '~beta/basiclib/betaenv';
−− lib: Attributes −−
(* This fragment is an example of using virtual patterns in BETA.
 * The following classification hierarchy is defined
 * 1.  Record
 * 2.  Person
 * 3. Employee
 * 3. Student
 * 2. Book
 * Record has a virtual (procedure) pattern:
 * Display, that displays the attributes of a record.
 * Display is further bound in the sub−patterns.
 * The patterns NewRecord, NewPerson, etc may be used for
 * generating new instances of the record−patterns.
 * The pattern Register has two virtual attributes:
 * regCat:< record, the elements in the register.
 * Display, that displays the elements of the register.
 *
 * This fragment is a library containing the declarations
 * It is used in the program "record"
 *)
Record:
(* Record objects contain two attributes: key and Display.  Key contains
 * the ID of this record (supplied by the programmar).  Display is a
 * virtual, enabling printing Records on the screen
 *)
  (#
     Key: @integer;
     Display:< (* declaration of a virtual (procedure) pattern  *)
       (#
          s: ^stream (* the input parameter is where to display this record *)
       enter s[]
       do
          s.newline;
          '−−−−−−−−−−−−−−−−−−'−>s.putLine;
          'Record: Key      = '−>s.putText;
```

```
         Key−>s.putInt;
         s.newline;
         INNER
      #);
  #);
Person: Record
  (#
  (* Person is a suppattern of Record, declaring two additional attributes:
   * Name and Sex.  Furthermore Display (inherited form Record) is extended
   * to print the Name and Sex attributes as well as the Key attribute.
   *)
     Name,Sex: @text;
     Display::< (* a further binding of Display from Record *)
       (#
       do
          'Person: Name     = '−>s.putText;
          Name[]−>s.putLine;
          '             Sex = '−>s.putText;
          Sex[]−>s.putLine;
          INNER
       #);
  #);
Employee: Person
  (# (* analog til Person *)
     Salary: @integer;
     Position: @text;
     Display::<
       (#
       do
          'Employee: Salary   = '−>s.putText;
          salary−>s.putInt;
          s.newline;
          '          Position = '−>s.putText;
          Position[]−>s.putLine;
          INNER
       #);
  #);
Student: Person
  (# (* analog til Person *)
     Status: @text;
     Display::<
       (#
       do 'Student: Status   = '−>s.putText; Status[]−>s.putLine; INNER
       #)
  #);
Book: Record
  (# (* analog til Person *)
     Author,Title: @text;
     Display::<
       (#
       do
          'Book: Author   = '−>s.putText;
          Author[]−>s.putLine;
          '         Title = '−>s.putText;
          Title[]−>s.putLine;
          INNER
       #)
  #);
doc0: (*** Temporary initialization ***) (# #);
NewRecord: (* creation and initialization procedure for Record objects *)
  (# RegCat:< Record; Rec: ^RegCat; K: @integer; R: ^Record
  enter K
  do &RegCat[]−>Rec[]; K−>Rec.Key; INNER ; Rec[]−>R[]
  exit R[]
  #);
```

```
NewPerson: NewRecord
(* creation and initialization procedure for Person objects *)
  (# RegCat::< Person; N,S: @text
  enter (N,S)
  do N->Rec.Name; S->Rec.Sex; INNER ;
  #);
NewEmployee: NewPerson
(* creation and initialization procedure for Employee objects *)
  (# RegCat::< Employee; S: @integer; P: @text
  enter (S,P)
  do S->Rec.Salary; P->Rec.Position; INNER ;
  #);
NewStudent: NewPerson
(* creation and initialization procedure for Student objects *)
  (# RegCat::< Student; S: @text enter S do S->Rec.Status; INNER ; #);
(* This is a declaration of a register pattern.  Register objects will be able
 * to contain Records (or instances of suntarrerns of Record.
 *)
NewBook: NewRecord (* creation and initialization procedure for Book objects *)
  (# RegCat::< Book; A,T: @text
  enter (A,T)
  do A->Rec.Author; T->Rec.Title; INNER ;
  #);
Register:
(* Register is a container pattern with operations insert (insert an object)
 * scan (traverse the register), has (test for presence of an object in the
 * register), display (display the objects in the register).
 *
 * Register may contain objects, that are instances of Record (of subpatterns
 * hereof).  Specializations of Register may restrict the classes of objects
 * allowed in the specialized register pattern by further binding the regCat
 * virtual pattern
 *)
  (#
     regCat:< Record;
     regLst (* private pattern *) : (# succ: ^regLst; elm: ^regCat #);
     head: ^regLst;
     init:
     (* initialization pattern to be invoked before first usage fo an
      * register object
      *) (# do none ->head[] #);
     scan:
     (* walks through the register, executing INNIR for each element in the
      * register.  P will refer to the current element in the register.
      *)
       (# elm: ^regCat; p: ^regLst
       do
          head[]->P[];
          search:
          (if (P[] = none )
           // false then
              P.elm[]->elm[]; INNER ; P.succ[]->P[]; restart search
          if)
       #);
     Display:<
     (* display the entire register by printing header and trailer text,
      * and scanning the entire register in between, invoking display on
      * each element in the register.
      *)
       (# s: ^stream
       enter s[]
       do
          s.newline;
          '############ Register Display '->s.putText;
          INNER ;
```

```
         s.newline;
         scan
           (# do s[]->elm.display #);
         '############ End Register Display #######'->s.putLine
      #);
    Has:
    (* takes an object reference, and checks whether that object is in the
     * register
     *)
      (# E: ^regCat; found: @boolean;
      enter E[]
      do
         false->found;
         search: scan
           (#
           do (if E.key // elm.key then true->found; leave search if)
           #)
      exit found
      #);
    Insert:
    (* Takes an object reference and inserts that object in the register
     * (if not already in the register)
     *)
      (# E: ^regCat; P: ^regLst
      enter E[]
      do
         (if (E[]->Has)
          // false then
             &regLst[]->P[]; head[]->P.succ[]; E[]->P.elm[]; P[]->head[]
         if);
      #);
 #)
 (* Register *)
```

Index

The entries in the alphabetic index consists of selected words and symbols from the body files of this manual − these are in **bold** font − as well as the identifiers defined in the public interfaces of the libraries − set in regular font.
In the manual, the entries, which can be found in the index are typeset like this. This can help localizing the identifier, when the link from the index if followed − especially in the case where the browser does not scroll the line to the top, e.g. because there is less than a page of text left.
In the small table of letters and symbols below, each entry links directly to the section of the index containing entries starting with the corresponding letter or symbol.

> < A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

<

>

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

Z

Zoom In **Zoom Out** **Zoom To Full Editor**