Design of Structure Presentation Layer
Editors in
LOGGIE
by
Mikael Benzinger and Anna Nordström
Design of
Structure Presentation Layer

Editors in

SICS

LOGGIE

by

Mikael Benzinger and Anna Nordström

February 1989

SICS, Swedish Institute for Computer Science
Box 1 263
S–164 28 KISTA
Sweden
# Table of contents

1 Abstract .................................................................................................................. 1
2 Sammanfattning ...................................................................................................... 1
3 LOGGIE™ ............................................................................................................... 3
   SPL .......................................................................................................................... 3
4 Introduction ............................................................................................................. 5
   HCI Overview ......................................................................................................... 5
   Editor Design ......................................................................................................... 6
5 Human–Computer Interaction with graphical editors ............................................. 7
6 Design of SPL Editors ............................................................................................ 10
   Conceptual Models .............................................................................................. 10
   Window-titlebar ................................................................................................. 10
   Mouse-commands .............................................................................................. 10
   Menues and Palettes ........................................................................................... 11
   Palettes .................................................................................................................. 11
   Menues ................................................................................................................... 12
   Icons vs. Text ....................................................................................................... 12
   Active window ..................................................................................................... 12
   Shape .................................................................................................................... 12
   Scroll .................................................................................................................... 12
   Refresh .................................................................................................................. 12
   Hidden Events ..................................................................................................... 12
7 Implementation ....................................................................................................... 13
   CLASS: SPL-Editors .......................................................................................... 13
   CLASS: SPL-E-DATA ......................................................................................... 13
   CLASS: SPL-E-VIEW ........................................................................................ 13
   CLASS: SPL-E-COMMANDS ............................................................................ 13
   Undo-mechanism ............................................................................................... 14
   CLIPBOARD ......................................................................................................... 15
   Menues ................................................................................................................... 15
   The Symbol Editor .............................................................................................. 17
      Selection ........................................................................................................... 17
      Alignment ........................................................................................................ 18
      Draw objects ................................................................................................. 18
      Edit objects ................................................................................................. 19
   The Connector Editor ......................................................................................... 21
   Selection .............................................................................................................. 21
   Draw objects ....................................................................................................... 21
   The edit menu ..................................................................................................... 21
   The Window Editor ............................................................................................ 23
      The attached menu ....................................................................................... 23
      The menu sub-editor ..................................................................................... 24
      The editor menu ............................................................................................ 26
8 References .............................................................................................................. 27
   Design .................................................................................................................. 27
   Implementation ................................................................................................. 27
Appendices .............................................................................................................. 29
   SPL Editors Version 1
   Manual for SPL Editors
1 ABSTRACT

This is a Master thesis work in Computer Science at the University of Stockholm, accomplished at SICS (Swedish Institute of Computer Science) during autumn and winter 1988—89.

In the area of Human-Computer Interaction, HCI, there is currently a lot of research. We are discussing some important aspects of HCI, mainly concerning design of graphical editors, as WYSIWYG, event driven actions, undo-mechanism and menu-arrangements.

The report also describes a design and implementation of some graphical editors used in LOGGIE\(^1\).

These editors are used to design symbols, windows, menus etc. to be used in LOGGIE.

2 SAMMANFATTNING


Arbetet har utförts som en del av LOGGIE-projektet. LOGGIE är en grammatikbaserad generator för grafiska interaktiva editorer. En användare kan definiera meta-beskrivningar, grammatiker, specifikationer m.m. som går att beskriva grafiskt. LOOGIE är uppbyggt kring en hierarkisk grafmodell. Utseende, attribut och interaktionsbeskrivningar för de noder och bågar som bygger upp denna modell definieras/redigeras med de editorer som specificeras i denna rapport.

LOGGIE är uppdelat i tre lager, Deriverings-Lagret (DL), StrukturPresentation-Lagret (SPL) samt det Grafiska StödSystemet (GSS). SPL presenterar de strukturer som skapas i DL med hjälp av grafik från GSS.

Rapporten beskriver följande viktiga aspekter inom området Människa-Dator-interaktion, speciellt avseende grafiska editorer: vem som bestämmer, händelsesyt, menydesign, dolda händelser, funktionalitet, hjälp, avbrottsmöjligheter, felhantering, enkla dialoger, obzeroende kommando-ordning, WYSIWYG (vad du ser är vad du får), enkla bilder, svarstid.

Rapporten beskriver också design och implementering av tre editorer i SPL. Dessa är skapade med en objektorienterad design och implementerade i LOOPS. Implementeringen föröker följa de krav på god användarmiljö som finns specificerade i rapporten.

\(^1\) LOGGIE\(^\text{TM}\), Language Oriented Generator of Graphical Interactive Editors, is a research-project at SICS in cooperation with Televerket Radio and Televerket PV.
Den är gjord enligt en metod som kallas Modell-Vy-Kontroll-paradigmen dvs. man separatorar klasserna i huvudgrupper för
- hantering av datastrukturen
- hantering av fönster och menyer, och interaktion
- hantering av kommandon.
3 LOGGIE™

LOGGIE [BACKLUND88] is a system for construction of design environments in which the design is based on formal design techniques given in an interactive graphic syntax. The user of LOGGIE can interactively specify and use meta notations, specification/programming languages, specifications/programs and program transformation. Since LOGGIE itself is an interactive tool, several types of editors are provided.

LOGGIE is divided into three layers, the Derivation Layer, DL, the Structure Presentation Layer, SPL, and the Graphic Support System, GSS.

DL is specified in terms of a hierarchic graph model based on typed nodes connected by typed links. A graph can represent a meta notation for specification of an attribute grammars formal language, an attribute grammar or a program written according to a specific grammar. The graphics are realized by GSS, LOGGIEs Graphical Support System [BOGEBY89], which is built upon an implementation of PHIGS [GULLESTAD88].

SPL is the organizer in the middle which presents the DL-structures, as pictures, with help from GSS-graphics.

Fig 3.1 LOGGIE-hierarchy

SPL

SPL - Structure Presentation Layer [FORSLOUN88] visualizes structures of any kind, but especially those generated in DL, in form of symbols and connectors (nodes and edges). SPL is designed for visualizing datastructures as graphs. Structures are associated with pictures in SPL. This graphical representation can be presented using all kinds of com-
mon graphical objects (boxes, circles etc.) as well as text. Attributes of an object in the structure can as well be associated with, and visualized by, the graphical presentation. Graph layout is also realized in SPL.

SPL offers three types of pictures: symbols, connectors and windows.

Each picture can have different aspects, that can be presented in different views.

Fig 3.2 A symbol

- Symbols represents nodes in a graph. Each node can be connected with other nodes, through a connector, and moved around freely in the window.

Fig 3.3 A connector

- A connector represents an edge connecting some nodes. These have a dynamic graphical appearance, i.e. when a connected symbol is moved, the connector will dynamically change its appearance and will still be attached to the symbol.

Fig 3.4 A window

- A window is the drawing surface of the LOGGIE editor, where the editing of the structures takes place.

The editors are fundamental tools. Their purpose is to let the user edit the graphical image of each symbol, connector and window to be used in an application built upon SPL.

We have used an object oriented design and implementation. The implementation [BENZINGER89] [BACKLUND88] is done with Envos Symbolic Development Environment\textsuperscript{1} [INTERLISP] using Envos Loops [LOOPS] [CLIL] on XEROX 1109 and on XEROX 1186.

\textsuperscript{1} Available for XEROX 1109/1186, FUJI XEROX 1021 and SUN 3/4
4 INTRODUCTION

The master thesis-project was divided into two major parts:

- Study of existing graphical editors and literature on Human-Computer Interaction.
- Design and implementation of three graphical editors, the SPL editors for symbols, connectors and viewports.

HCI OVERVIEW

Human-Computer Interaction, HCI, is currently a very active research area. A lot of new ideas, tools as well as completely new environments are presented. XEROX Interlisp-Machines and XEROX wordprocessor STAR introduces a new way to communicate with the computer, a window system. It broke through when the ideas got more available, at a lower cost, with the introduction of Apples Macintosh.

The computer window is thought of as a desk where the user freely can manipulate documents (= files), folders (= directories) etc. This technique uses high resolution screens together with a pointing input-device called "mouse" and pop-up menus on the screen. The fundamental idea is to let a cursor on the screen follow the mouse movements and making selections with a button, when the cursor is pointing in the area of interest. The technique tries to follow the principle "What you see is what you get". Lots of efforts it made today in HCI-research to make this environment better.

There are several different ideas of what is a good environment. Some aspects have been carefully studied combining computer science with knowledge from areas as psychology and linguistic, but other aspects are not yet fully studied. The knowledge of these results are still not wide-spread and arguments like "I like my environment best – therefore it is the best for everybody" still flourish.

HCI can be studied reading different literature, especially the proceedings from the SIGCHI1, but should of course also be studied by testing different environments and tools within them.

---

1 SIGCHI — ACM Special Interest Group on Computer & Human Interaction
EDITOR DESIGN

The editors are designed in the purpose to be easy and friendly but still very powerful.

Principally there are four things that are interesting to edit in the SPL editor:

- Appearance — How the pictures look like.
- Behaviour — How the user can interact with the pictures.
- Attribute presentation — How the attributes from DL will be presented.
- Placement — How the pictures shall be placed.

Using all graphical primitives from GSS, a picture — symbol or connector — can be described as a circle, line, bitmap, arc etc. as well as a combination of these. Presentation of associated attributes, placement constraints and interaction descriptions are editable as well. Editing of GSS-hierarchy, constraints and presentation of attributes is done with subeditors. The symbol-editor also have subeditors for editing markers and shades.

The window-editor makes it possible to design the windows outlook and behaviour. To edit its menus there is a subeditor.

Library routines for pictures, windows, markers and menus are supported as well.
5 HUMAN–COMPUTER INTERACTION WITH GRAPHICAL EDITORS

Some of the most important aspects of a good environment can be described by discussing the following conceptions in a workstation environment: Who is in charge, Event-driven, Menu-arrangements, Hidden Events, Functionality, Help, Abort, Undo, Error Handling, Clean dialogs, Independence of command order, WYSIWYG, Simple icons, Request-time and Do not overuse.

Who is in charge:

The user must feel that he makes all the important decisions and that he can edit all aspects of a picture or document. The user is doing the job, the computer is just a tool [DIALOG86] [SEVERINSON86].

Event-driven:

Each action that the user makes should be treated as an event. An action can be a sequence of one or more pushes on the mouse-button or typing on the keyboard, a combination or sequence of both or an interaction with a sub-system. When the event is finished it is treated as a unit, which for instance could be undone. Using an object-oriented environment and treating each event as an unique object makes it easy to implement this [BERGERON87].

Menu-arrangements:

There are several different aspects concerning presentation of menus.

- Placements: Menues could pop-up directly when pushing a button, placed on the screen just under the cursor, or they can be reached after moving the cursor to an specified area, like a title-bar or a palette, and then pushing a button. Menues can also be fixed, continuously shown or attached to a specific window.

- Contents: Either all menu-items are visible every time the menu is shown, whether or not all items are chooseable, or the item-list is recomputed each time to hold only items that are useful/possible to choose.

- Subject-arrangements: The menu items could be arranged in groups according to subject, or by sorting them by probability of selection.

It is wise to not overuse fixed menues, only show important and often used items that way [SEVERINSON86]. Results from studies [SOMBERG87] states that fix placement of menues and items is important. For a beginner, subject-arrangement is more important than for the more experienced user. When a user gets more used to an environment it is more convenient with fix placement. The user knows what he wants and where to get it. All items should be shown, the menu-handler should prevent the user to choose not allowed items or prevent an action to take place when the user chooses such an item.
Hidden Events:
It is important that the meaning of the text of a menu-item or the image of an icon is clear and that any hidden events does not mess things up for the user. A hidden event could be some side effect of choosing the item, or something that happens if you "double-click". If the hidden event is not undoable or obvious the user should be warned, or be able to confirm in some way [CASNER87].

Functionality:
There should be a good visual, and perhaps also sounding, feedback when interacting with the system. Marking in a box or choosing from a menu should for example result in highlighting of an area, and then the action is, if possible, visualized as well. For example, when moving an object the object is continuously redisplayed at the cursors new position instead of just letting the object pop up in the new place. When an event takes long time the user should be informed in some way, for example by changing of cursor-image, displaying an hour-glass or by showing some sort of increasing index indicating how much of the action that has passed [KINDBORG87]. It is important that this is implemented in an effective way.

The main purpose of the program should, of course, not be slowed down by the feedback. The user should also be able to choose level of feedback.

Help:
At all stages and at all levels there should be a mechanism for help. The information given should be short, descriptive and relevant.

Abort:
A possibility to abort commands and dialogs at all stages is important.

Undo:
An undomechanism is very useful. All actions should be logged on a history stack to make it possible to regret one or many actions. At best there is a possibility to undo the undo as well.

Error Handling:
When an error occur, there should be a possibility for the user to recover to a stage before the break took place. Displaying an error message, telling what went wrong, is of course valuable.

Clean dialogs:
When creating the visual display it is important to not overload with information. It should be presented with an artistic insight concerning choice of shapes, proportions etc. To present the functionality is important, there should not be any unwanted surprises [CARDIELI87]. If possible the user should be able to reshape the dialog window. Another requirement for clean dialogs is that there always should be a possibility to refresh/redisplay the screen.

Independence of command order:
Often situations occur when the user must make a sequence of choices, for example drawing a figure with a selected tool and with a selected pattern. If possible the order of choosing, (tool ⇒ pattern ⇒ action) or (pattern ⇒ tool ⇒ action) should not matter [LIEBERMAN85].
WYSIWYG:
This principle "What You See Is What You Get", WYSIWYG, is holy. It means that if you make a picture with an editor the result outside the editor should look EXACTLY the same on paper as on screen. Many systems tend to treat this as "What You See Is Almost What You Get".

Simple Icons:
There is no need to overuse icons and pictures if their meaning is not obvious. Many times it is more reliable and clear to use a short textual description, a word, instead. Use icons when it is possible, text otherwise.

Request time:
It is important that the user feels that the system is fast, even if it is not. Good feedback is essential for this.

Do not overuse:
The graphical systems, nowadays, often supports a lot of different shades, colours and fonts. It is, though, important not to overuse these possibilities. At best there is only two or three different fonts, patterns displayed in a window, otherwise there is a great danger that the message/picture gets lost.

It is important that designers always has the HCI-requirements in mind, the computer and the computer programs, are only tools for different activities. A human-friendly interface makes them easier and nicer and thereby safer to use.
6 DESIGN OF SPL EDITORS

The design of the SPL editors follows as far as possible the requirements stated in previous chapter. The editors are event-driven. They have a history-stack that permit the user to undo more than just the last command. Independence of command order holds for most commands. Abort is possible in all interaction dialogs as well as in the subeditors. Error handling is only implemented in a simple low level manner. WYSIWYG holds for everything designed to be shown on the screen.

CONCEPTUAL MODELS

The symbol and connector editors can be described as a drawing desk with a paper, tools and pencils. The paper is surrounded with tools like pencils for circle drawing etc. and patterns for them. Some tools are still in the drawer and the user can get them from there.

The window in the window editor can be regarded as a paper on which you paste items such as border, title, size etc.

The menu, constraints and attribute sub-editors are formulas.

The hierarchy sub-editor is a graph where the lines between the nodes are the hierarchy-edges.

WINDOW-TITLEBAR

The window titlebar is a popular place for presentation of information and for interaction with an editor. We have choosen to have three separate parts.

- Shrinkbutton: The left part, indicated by a box indicating the picture-type in it (i.e. SYMBOL, CONNECTOR and VIEWPORT). Clicking in the box, makes the editor update the definition it is editing and then shrink the editor into its icon.
- Movebutton: The middle part, indicated by the name of the object edited. When the name is surrounded by asterixes (i.e. *MY-PICTURE*) there are changes to the picture that are not yet applied to its definition.
- Menuesbutton: The right part, indicated by a box with the word MENUES in it. This is an entrance to a collection of all menues that are separately available in different areas of the editor.

MOUSE-COMMANDS

The mouse of the XEROX machines has three buttons. In the SPL editors each button invokes the same type of event independent of context.

- The left button, here called Activity-button, invokes some sort of action. In the selectionpalette it means creating a new graphical object, in the other palettes it starts an event. Clicking on a object means selecting it, clicking on a selected object means moving it.
- The middle button, here called Information-button, gives either information about global values or selected items (Active, ED-items\(^1\) etc.) or help about what will happen if you click with the A-button in a palette or in a menu. Inside a window the information given is a presentation of the attributes of a selected object.

- The right button, here called Menu-button, pops up a context-dependent menu.

**MENUES and PALETTES**

There must always be a choice between a larger drawing surface or a lot of palettes, the user can not have them both. Often used menu-items have been choosen to be put in the palettes, if this choice is not sufficient the user could make any pop-up menu stay continuously visible.

**PALETTES**

Palettes in the SPL editors are menues, attached to the main window, with mostly graphically displayed items. When editing in more than one SPL editor at the same time only the active (= current) editor shows it palettes. Palettes holds often used commands. In the symbol and connector editors items for creating new graphical objects and changing SPL attributes (i.e. connectable, active etc.) are shown as palettes. Patterns for dashing and lines are also shown as palettes. It is possible for the user to bring up items from a menu to a palette to make them easier to reach.

Fig 6.1 Tool-palette

---

\(^1\) See FORSLUND88S chapter 2 for an explanation.
MENUES

Menues are the usual pop-up menues. There are two main menues. One is reached with the M-button in the header. It holds items for editor behaviour and commands that operate directly on the SPL picture object. Inside the drawing-area the M-button gives a menu which holds items concerning editing on selected GSS objects. Pushing the A-button in the MENUES box in the title gives the main-menu, which is a combination of both.

Fig 6.2 Menu dialogue

All menues have a paper-clip, , as their first item. When it is chosen the menu stays open and the item becomes a close box. Clicking in that item closes the menu.

ICONS VS. TEXT

It is a good idea to use icons when possible — the recognition of pictures are faster than that of texts. But, an icon should be a good description of what will happen if choosing it. It is better with textual description than with bad icons. Therefore icons and textual items are combined.

ACTIVE WINDOW

It is possible for the user to have several editors open at the same time, but it is not possible to interact with more than one at a time. The currently active editor is marked, i.e. all inactive editors have their palettes closed. When changing from one editor to another the new one shows its palettes and the previously used closes its.

SHAPE

All windows have a shape-button, at its bottom right corner. A push there gives the user possibility to redraw the area for the window.

SCROLL

Scrolling is implemented using the standard XEROX scrollbars, i.e. when the user get near the left or bottom border of the window a scrollbar pops up.

REFRESH

There is always a possibility to redisplay the window.

HIDDEN EVENTS

In our design there are a few hidden events, all associated with updating of the definition that is edited. Clicking in the shrink-button also updates. Updating always implies emptying of the history-stack.
7 IMPLEMENTATION

To make a simple and legible implementation we have used the Model-View-Controller paradigm [Krause88], i.e. to separate classes that handles the datastructure, the visual presentation and the commands, used in Smalltalk-80 [Model88] and in MacApp on the Macintosh [Schmucker86]. The SPL editors uses four main classes SPL-EDITORS, SPLE-DATA, SPLE-VIEW and SPLE-COMMANDS.

CLASS: SPL-EDITORS

This class handles all communication between data, view and commands. It also takes care of the interface to SPL. Therefore it is natural that the main method which controls the whole editor and the history-stack handling is placed here.

This class’ main method is EDITOR-MAIN which controls the flow for each command-object (see SPLE-COMMANDS) that is created. It is invoked from the SPLE-VIEW class, when some action (menu-item choice, for instance) has been done. The different methods of the command-cycle are executed, and thereafter the history-stack is updated.

When invoking an editor from SPL, all communication with the editor is done with this class.

CLASS: SPLE-DATA

Holds the internal datastructure for the editor, and methods to handle it. Each GSS-object which is added to the datastructure is regarded as a unique definition, and therefore assigned a unique GSS-definition name. The clipboard facility is also handled by this class.

CLASS: SPLE-VIEW

All palettes, menus and windows are controlled from this class. That makes it natural for the command objects to be invoked from this class and sent to SPL-EDITORS. This class also holds methods for Activate/Inactivate editor and selection of objects. All menus and palettes are associated with SPLE-VIEW. The menus and palettes are realized using Interlisp menus and Interlisp Freemenus [Backlund88y].

The main methods are DO-A-BUTTON-COMMAND, DO-I-BUTTON-COMMAND and DO-M-BUTTON-COMMAND which are invoked with the corresponding mouse button.

CLASS: SPLE-COMMANDS

One of the four main classes are specialised to handle all sorts of commands. Anything that happens can of course be a command, but in this case, a command changes something, a placement or angle for example. A command consists mainly of four parts: NewInstance, TRACK-MOUSE, TRACK-CONSTRAINTS and DO-IT, and they are
called in that order. One other important feature is that the user is able to undo most of the commands.

NewInstance makes a new instance of a command-object and saves some further important information. Since NewInstance is the only method in a command that can be called with different number of parameters it is also important to use this method to save information for the command object that will be needed later.

The method always called after NewInstance is TRACK-MOUSE. This method does not only track the mouse, but also collects all sort of information necessary for the command that NewInstance does not bring, like patterns, positions, linewidths. TRACK-MOUSE always collects two values for the same attribute: the old value of the attribute and the new value. Most commands can be interrupted and that possibility is handled by this method.

TRACK-CONSTRAINTS controls that objects, if possible, will be drawn completely inside the window and that they will not be clipped. Otherwise the method resizes the editor window.

Last, DO-IT is called. It updates the data-structure and the window.

![Diagram of command-chain](image)

**Fig 7.1 A command-chain**

**UNDO-MECHANISM**

Since it is important to have the possibility to undo commands an implementation is made for an undomechanism. There are two stacks; one for all commands that the user has done (history-stack) and one for the commands that he has made undone. The latter is always emptied when the user starts a new command. Every command object that is not interrupted is pushed on the history-stack, which only will be emptied when the user says that he is pleased either by closing the editor, shrinking the editor, saving the definition on file or applying the changes to the definition. When the user does an undo-command, the command, of course, is made undone and the command-objects is moved from the history-stack to the undone-command-stack. Vice versa happens if he does an undo command to an undone command-object. Every time a command is made undone the values fetched by TRACK-MOUSE (old and new values for involved attributes) are swapped.
CLIPBOARD

There is one global clipboard for each type of editor. The user can cut or copy to and paste from the clipboard.

COPY
  Copies the object and places the copy in the clipboard.
PASTE
  Copies the object in the clipboard and returns the object to the editor where the user will place it.
CUT
  Moves the object from the editor to the clipboard.

MENUES

<table>
<thead>
<tr>
<th>SYMBOL-ME</th>
<th>NU</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPEN</td>
<td></td>
</tr>
<tr>
<td>SHRINK</td>
<td></td>
</tr>
<tr>
<td>SAVE&amp;EXIT</td>
<td></td>
</tr>
<tr>
<td>SKIP</td>
<td></td>
</tr>
<tr>
<td>MOVE WINDOW</td>
<td></td>
</tr>
<tr>
<td>SAVE</td>
<td></td>
</tr>
<tr>
<td>HARDCOPY</td>
<td></td>
</tr>
<tr>
<td>CLEAR</td>
<td></td>
</tr>
<tr>
<td>APPLY</td>
<td></td>
</tr>
<tr>
<td>SHOW HOT SPOTS</td>
<td></td>
</tr>
<tr>
<td>CHANGE NAME</td>
<td></td>
</tr>
<tr>
<td>INPUT-DEFINTIONS</td>
<td></td>
</tr>
<tr>
<td>DL-ATTRIBUTES</td>
<td></td>
</tr>
<tr>
<td>SHADES</td>
<td></td>
</tr>
<tr>
<td>MARKERS</td>
<td></td>
</tr>
</tbody>
</table>

There are two kinds of pop-up menus; one for editing objects in the editor and one for editing and handling the whole definition. The first mentioned handles the selected object. The latter have a lot of common functions handling the whole definition:

Fig 7.2 The editor menu

➡️ Makes the menu stay continuously open.

Makes the continuously open menu close.

OPEN
  Clears the definition and loads a new one into the current editor.

SHRINK
  Shrinks the editor to an icon, clears the history-stack and applies the definition.

SAVE&EXIT
  Saves the definition and exits the editor.

SKIP
  Cancels all commands that have been done (since last emptying of history-stack) and exits the editor.

APPLY
  Updates the definition.

SAVE
  Saves the definition to file.

MOVE
  Moves the editor.

CLEAR
  Clears the window.

REDISPLAY
  Redisplays the window.
<table>
<thead>
<tr>
<th>HARDCOPY</th>
<th>Prints the definition on paper.</th>
</tr>
</thead>
<tbody>
<tr>
<td>INPUT-DEF</td>
<td>The user can choose how the symbol will interact with the user in SPL.</td>
</tr>
<tr>
<td>CHANGE NAME</td>
<td>It is always possible to change the name of the definition.</td>
</tr>
<tr>
<td>HIERARCHY</td>
<td>Since it is simpler to handle hierarchies rather than a lot of different objects, the objects are linked in a simple hierarchy. If the user wants to change the hierarchy for some reason, then there is a simple hierarchy editor where the user can build a new hierarchy with the already drawn objects.</td>
</tr>
</tbody>
</table>
THE SYMBOL EDITOR

![Symbol Editor Diagram]

Fig 7.3 The symbol editor

SELECTION

Selected items are boxed. This is a "command" that is not undoable.

![Selection Diagram]

Fig 7.4 A window with one selected and one not selected item
ALIGNMENT

One way to be able to align objects is to define nine visible and pickable points for each object: leftbottom, centerbottom, rightbottom, leftcenter, centercenter, rightcenter, lefttop, centertop and righttop. The idea is that when an object is marked for alignment with these points (reference items), the user is able to choose one point per object and then tell them how they will be placed relatively each other.

![Diagram of placement points]

**Fig 7.5 Placement points**

DRAW OBJECTS

When the user wants to draw a new object, a new instance, which is assigned a unique GSS definition name, of a GSS-object is created. All global defaults are applied on the new objects attributes and an interaction for collecting specific values like position and image is started. The user is able to draw different things in the symbol editor:

- circles
- polygons
- bitmaps
- closed curves
- ellipses
- lines
- curves
- lines
- rectangles
- texts
- polylines

The tools for drawing figures are displayed as a palette to the left of the window. All objects are linked in a simple hierarchy. The first drawn object becomes the root and all other objects are automatically linked as its children.
EDIT OBJECTS

<table>
<thead>
<tr>
<th>SYMBOL</th>
<th>EDIT-MENU</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>UNDO</td>
</tr>
<tr>
<td></td>
<td>UNDO undone</td>
</tr>
<tr>
<td></td>
<td>MOVE</td>
</tr>
<tr>
<td></td>
<td>SCALE</td>
</tr>
<tr>
<td></td>
<td>ROTATE</td>
</tr>
<tr>
<td></td>
<td>FONT</td>
</tr>
<tr>
<td></td>
<td>LINES</td>
</tr>
<tr>
<td></td>
<td>CLIPBOARD</td>
</tr>
<tr>
<td></td>
<td>DUPLICATE</td>
</tr>
<tr>
<td></td>
<td>DELETE</td>
</tr>
<tr>
<td></td>
<td>BURY</td>
</tr>
<tr>
<td></td>
<td>TO-TOP</td>
</tr>
<tr>
<td></td>
<td>PRIORITY</td>
</tr>
</tbody>
</table>

The user has the possibility to change the outlook for the objects if he is not pleased with them. He can choose to select a few objects and change an attribute at a time or he can put a new value as default for all new objects. The idea is that the user always will be able to choose a tool and an object to manipulate, despite of order. Therefore, most of our commands act in this way. If there is no object chosen when the user selects a tool, he will be asked for an object and if the user does not select an object, the command will be interrupted. Here is a list with all commands that act in this way:

The menu will stay continuously open.

The continuously open menu will close.

The object will buried.

The object will be put on top.

Deletes the object.

Duplicates the object and lets the user interactively place the new object.

The user moves the extent of the object and when he is pleased the object moves to that place.

Rotates the object towards a position with an angel that the user gives.

Scales an object with different scale in X and Y. Circles could of course only be scaled in one direction.

Align two objects.
The four functions below are placed as the four upper palette items in the left palette:

**ACTIVE/INACTIVE**
Outside the editor the symbol will have active parts, which behave in a certain way when they are clicked on. All parts (objects) are initially active.

**CONNECTABLE OUT**
A symbol have parts that a connector can attach to, an out-item. The user can choose which parts (objects) which shall be out-item. All parts are initially out-items.

**CONNECTABLE IN**
A symbol have parts that a connector can attach to, an in-item. The user can choose which parts (objects) shall be in-items. All parts are initially in-items.

**ED-VISIBLE**
Makes an object visible in the symbols or connectors exported display. All parts are initially ed-visible.

There is a few commands that do not act in the usual way. If there is no selected object they change default values instead of asking the user to select an object.

**DASHING**
Sets the dashing on lines.

**LINE-WIDTH**
Sets the line-width.

**PATTERN**
Sets the pattern on circles and rectangles.

**FONT**
This sets the face, family and size of a text.
THE CONNECTOR EDITOR

![Diagram of CONNECTOR editor with SPLE-CONN-DEMO and MENUES]

Fig 7.7 The connector editor

SELECTION

Selected items are boxed. This is a "command" that is not undoable.

DRAW OBJECTS

The connector editor differs from the symbol editor with the possibility to draw:
- arrowed polylines
- arrowed lines
- arrowed curves
- arrows

THE EDIT MENU

There is a lot of likeliness between this and the edit menu in the symbol editor:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BURY</td>
<td>The object will be buried.</td>
</tr>
<tr>
<td>TO-TOP</td>
<td>The object will be put on top.</td>
</tr>
<tr>
<td>DELETE</td>
<td>Deletes the object.</td>
</tr>
<tr>
<td>DUPLICATE</td>
<td>Duplicates the object and lets the user place the new object.</td>
</tr>
<tr>
<td>MOVE</td>
<td>The user moves the extent of the object and when he is pleased the object moves to that place.</td>
</tr>
<tr>
<td>ROTATE</td>
<td>Rotates the object towards a position with an angel that the user gives.</td>
</tr>
<tr>
<td>SCALE</td>
<td>Scales an object with different scale in X and Y. Circles could only be scaled in one direction.</td>
</tr>
<tr>
<td>DASHING</td>
<td>Sets the dashing on lines, if no object choosen, default values are changed.</td>
</tr>
<tr>
<td>LINE-WIDTH</td>
<td>Sets the line-width, if no object selected, default values are changed.</td>
</tr>
</tbody>
</table>

21
These items could be chosen in the left palette:

**CONNECTABLE FROM**  A connector have parts that out-items from the symbol can be connected to, a from-item. The user can choose which parts (objects) which shall be from-items. Initially all items are from-items.

**CONNECTABLE TO**  A connector have parts that in-items from the symbol can be connected to, a to-item. The user can choose which parts (objects) which shall be to-items. Initially all items are to-items.

**ACTIVE/INACTIVE**  Outside the editor the symbol will have active parts, which behave in a certain way when they are clicked on. All parts (objects) are initially active.

**DYNAMIC**  Makes the object dynamic. A connector must have the possibility to reach between to different nodes either the distance is short or long. Initially all items are dynamic.

There is a possibility to see the outlook of the connector when it has different number of "legs". The user can choose how many out-items and in-items he wants to see in two small windows in the upper left corner of the editor.
THE WINDOW EDITOR

<table>
<thead>
<tr>
<th>TITLE</th>
<th>VIEWPORT</th>
<th>WINDOW-DEMO</th>
<th>MENUES</th>
</tr>
</thead>
<tbody>
<tr>
<td>MENUES</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ICON</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Fig 7.8 The window editor

THE ATTACHED MENU

TITLE

Define the title-line of the window. Text and pattern are editable.

MENUES

This item gives the possibility to add, delete and edit menu definitions. The default menu only holds the items YES and NO. The user can change all attributes on which a menu depends.

ICON

Defines the icon of the window definition. The icon depends on a bitmap, a text and an icon-mask. If both bitmap and text is present the text is not displayed.
THE MENU SUB-EDITOR

```
DERIVATION MENU
Add State
Add Transition
Remove

Derivation Menu Editor

TITLE:  DERIVATION MENU
TITLE-FONT:  MODERN

CONTEXT:  STRUCTURE
MODE:  SEND
HOW:  PINNED
FN?:  NO

POSITION:  [5, 255]
ROWS:  3
COLUMNS:  1
BORDER:  1
CENTER?:  YES
CHANGE-OFFSET?:  YES

ITEMS

FAMILY:  MODERN
SIZE:  10

APPLY-ITEM:  **APPLY**
CANCEL-ITEM:  **CANCEL**
ITEMS:  ["Add State" "Add Transition" "Remove"]

SHOW MENU  ADD  APPLY  QUIT  ABORT
```

Fig 7.9 The menu editor

TITLE  The user can change the title of the menu.
TITLE-FONT  The user can change the font of the title.
APPLY-ITEM  There are always two items that have to be in a menu when it is in continue-mode; apply-item and cancel-item. This will change the name of the apply-item to something that the user want.
CANCEL-ITEM  This will name the item that cancels the command(s) that the user have chosen.
BORDER  This field holds a menu of integers with which the user can tell how wide the border around every item in the menu should be.
CENTER?  Centres or left positions the text in the items.
CHANGE-OFFSET?  If the user choose YES this makes the menu "start" on the item he choose the last time he used this menu.
ROWS  The user do not need to show all items in the menu, he/she can choose to show more or less rows than the number of items. This field will give a small menu were he/she can choose to how many rows he/she want to choose.
COLUMNS
A menu do not need to have one column, the user can here form the menu in as many columns he/she like.

FAMILY
All text in the items are by default written in Helvetica. Here the user can choose any, of the system known, family.

SIZE
Sets the size of the text in the items.

POS
By clicking in this field the user move the menu and thereby change the position interactively.

CONTEXT
The item that the user choose in the menu is sent to this context.

MODE
The user can choose between gather- and send-mode. Send will execute the command immediately by sending the name of the item as a message to the object in context. Gather collects all information and then waits for an apply-command for execution.

HOW
There are three different kind of menus; pinned, pop-up and continue, that the user can choose between. A pinned menu is opened and attached to the window. The continue menu also is fixed-placed but on a arbitrary place and the pop up menu pops up on the cursors place.

FN?
The menu items calls a function instead of methods.

ITEMS
When the user click here he will get an editor where he can write all items that he want to have in he menu.

At the bottom of the formula there are a few buttons that take care of the definition:

SHOW MENU
Displays in the way HOW indicates.

ADD
Adds the definition of the menu to the window definition.

APPLY
Applies the users choices on the definition. It also empties the history-list and therefore he can not undo things that he has applied.

QUIT
Applies and quits the menu-editor.

ABORT
Abort everything that the user have done with the definition until last APPLY and quits the menu-editor.
THE EDITOR MENU

The editor item has a few items that is different from the other editors:

In the subitem BEHAVIOR you are able to set some attributes for the window. They are all of the type ON/OFF:

- **CLOSEABLE**: If the window are closeable or not.
- **EXPAND&SHRINKABLE**: Sometimes it is useful to have a window that the user can shrink when he need the space for it and expand when he want to use it again. Here the user can tell if the window is expandable.
- **OPENED**: The window is opened when the instance is created.
- **PICKABLE**: If the user makes the window scrollable he do not need to have as large window.
- **SCROLLABLE**: The window can be shaded.
- **SHADED**: The user can resize the window when he find it too small or too big.
- **SHAPEABLE**: Will the window be buried or to-top when it is created.
- **TO-TOP**: Sets the window to continuously update the structure in it. Otherwise it only updates when the user send an update message.
- **UPDATABLE**: The user can reshape the editors window.

At last there is an item left:

- **SHAPE**: The user can reshape the editors window.
8 REFERENCES

DESIGN


CARDELLI87 Building User Interfaces by Direct Manipulation, Luca Cardelli, System Research Center of Digital Equipment Corporation, Palo Alto, California, 1987


DIALOG86 Människa-datorinteraktion i grafiska system, Dialog-projektet, slutrapport, TRITA-NA-E8686, NADA, KTH, Stockholm, 1986


 IPLAB86 IPLab - Projektansökan, TRITA-NA-E8659, NADA, KTH, Stockholm, 1986


KJELLDALH83 Requirements for Interactive Editing of Diagram, Lars Kjell Dahl & Staffan Romberger, TRITA-NA-8303, NADA, KTH, Stockholm, 1983

LIEBERMAN85 There are more to menu systems than meets the screen, Henry Lieberman, Proceedings of SIGGRAPH’85 (San Fransisco, July 22—26). ACM, New York, 1985, pp. 181—189.


SEVERINSON86 Utvärdering av Dialogtyper för Grafisk Interaktion, Kerstin Severinson-Eklundh, TRITA-NA-E8667, NADA, KTH, Stockholm, 1986


IMPLEMENTATION

BACKLUND88I Implementation Guidelines for LOGGIE, Björn E. Backlund & Pär Forslund, SICS, 1988

BACKLUND88L SICS - LOGGIE, a Language Oriented Generator of Graphical Interactive Editors, Björn E. Backlund, Pär Forslund, Olof Hagsand & Björn Pehrson, SICS, 1988

BACKLUND88Y YAFFY USER’S GUIDE, Björn E. Backlund, SICS, 1988

BENZINGER89 Technical Documentation for SPL Editors, Mikael Benzinger & Anna Nordström, To be published as SICS technical report, ISSN 1100—3154

BOGEBY89 Implementation of Basic Graphical Support System, Hans Bogey & Harri Vuorela, SICS technical report T89005, ISSN 1100—3154


FORSOUL388S Structure Presentation Layer - Specification, Pär Forslund, SICS, 1988
<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GULLESTAD88</td>
<td>ALPHA: Implementation of a subset of PHIGS, <em>Ylva Gullestad</em>, SICS technical report T89001, ISSN 1100-3154</td>
</tr>
<tr>
<td>LOOPS</td>
<td>ENVOS LOOPS REFERENCE MANUAL, Envos 1989</td>
</tr>
</tbody>
</table>
SPL EDITORS VERSION 1.0

The SPL Editors are implemented in a first version (February 1989), that fulfil the requirements of this design description except for the following topics:

• Selection of more than one object at the same time. It is desirable that the user can select a number of objects and work with them parallel. The code is prepared in some extent to this.
• Alignment of two objects.
• The user can choose a tool and start to use it and after a while decide to choose a new tool without being finished. In version 1.0 the user has to stop the tool before he/she can choose another.
• Track-constraints are not implemented at all.
• The connector can not be shown with different numbers of in-items and out-items. That is because the routing in SPL is not implemented throughout.
• Hierarchy, attributes and constraints sub-editors are not implemented.
• The possibility to view the history-stack and the undo the last $N$ commands.
• No sort of arrows can be drawn in the connector editor.
• Hardcopy
• The user defines dashing by a numeric list, in the future it will be a bitmap editor.
• Help/Information is only implemented for a few commands.
• Boxing a symbol
• Libraries
• The user can not pick up items from the menus and add them to palettes.
Users guide to
Structure Presentation Layer Editors

by

Mikael Benzinger and Anna Nordström

February 1989

SICS, Swedish Institute for Computer Science
Box 1 263
S–164 28 KISTA
Sweden
Table of Contents
1 Manual for the SPL-editors ................................................................. 1
2 The Mouse .............................................................................. 1
3 A Presentation of the Editor ....................................................... 1
4 Title-line .................................................................................. 2
   Shrink-button .......................................................................... 2
   Move-button ........................................................................... 2
   Menu-button ........................................................................... 2
5 Menues ....................................................................................... 3
   The edit menu .......................................................................... 3
   The editor menu ...................................................................... 5
6 Symbol and Connector ................................................................. 6
   To Create Objects ................................................................... 6
   Patterns ................................................................................... 6
   To Draw An Object .................................................................. 6
   Edit menu ............................................................................... 7
   The editor menu ...................................................................... 8
7 Symbol ....................................................................................... 8
   Left Upper Palette ................................................................... 8
   The editor menu ...................................................................... 9
8 Connector ................................................................................... 10
   Left Upper Palette .................................................................. 10
9 Window ....................................................................................... 11
   The fixed menu ....................................................................... 11
   Menues .................................................................................... 13
   The editor menu ...................................................................... 15
Index ............................................................................................ 17
1 MANUAL FOR THE SPL-EDITORS

This manual first describes the general behaviour and commands for all editors. Since the symbol editor and the connector editor are alike, they have a small part described together. The manual ends with special descriptions for each editor.

2 THE MOUSE

The mouse has three buttons. Each button has a certain behaviour. You use the left button, when you want to select an object or tool, the middle button when you want some help or information, the right button when you want a menu.

3 A PRESENTATION OF THE EDITOR

The window is surrounded by some tools that will help you to design a symbol. The title holds three buttons for different functions:

• You can shrink the editor with the left button, named symbol, connector and window respectively. This button first applies the changes on the definition and then shrinks.

• The middle button is used when you want to move the editor and it also holds the name of the definition.

• If you push the right button when the cursor is inside the window or on an object a menu pops up. The menu holds commands that edit different parts of the definition. The upper item (the one with a paper-clip on it) keeps the menu open, and it will be fixed until you closes it. You close the fixed menu by choosing the upper item (with a box on it).

At the left of the window there is a fixed menu of tools placed that is useful when you define a definition. Instructions and other information for the commands and some help pops up in a little window at the top of the editor.
The order between selection of the tool and the object that the tool operates on are in most cases not determined. Some tools check if there is any object selected and operates on it, otherwise it changes a default value.

4 TITLE-LINE

SHRINK-BUTTON

If you want to work with something else than the definition and you need the space that the editor takes you can shrink it by clicking with the left button in the shrink-box. The icon of the editor will be placed on the same place it last were placed. If the editor have not been iconised before you have to place it with the mousemovements and the push the left button.

MOVE-BUTTON

The middle button in the title-line holds the name of the definition of the symbol. While you are pushing the left mousebutton you can move the editor with the movement of the mouse.

MENU-BUTTON

There are two categories of menues. One that operates on the whole editor and one that edits parts of the definition. Since it shall be easy for you to remember where the menues are, they are collected in one button. It is created by all items in the edit menu and holds the editor menu as a submenu.
5 Menues

There are two different menus that may appear when you push the right mouse-button. The menu that works on objects appears when the cursor is in the window and the other one in the title-line and other places in the editor that do not have an own menu. It handles things concerning the editor and the whole definition.

The main menus have a paper-clip as the first item. If you choose that item, the menu stays open and you can place it where ever you want. You can close it by choosing the upper item, which has become a box. If you press the right mousebutton in an open menu, you will get a menu where you can choose to move the open menu.

Fig 5.1a Part of an pop-up menu
Fig 5.1b Part of an fixed menu

The Edit menu

The edit menu collects a variety of functions you can apply on an object. Some commands operates between a clipboard, where you can save an object, and the editor. Some operates graphically on an object. Common for most of them is, that if there is no object selected it will ask for one, but some changes default values instead.

Fig 5.2 The symbol editors edit menu.
UNDO - UNDO undone  Since it is desirable to have the possibility to easy undo commands the undo-item, 'UNDO', is placed on the top of the edit-menu. Almost any commands are undoable, even undo-commands, but then you have to choose the item named 'UNDO undone'. There are two stacks that takes care of the undo-mechanism; one for all new commands, history-stack and one for undone-command. When a command is made undone it will be moved from the history-stack to the undone-commands-stack and it will be moved back if you choose 'UNDO undone'. The undone-command-stack will be emptied when you give a new command. Things like creation and editing an object counts as a command and are therefore undoable, but things like selection are not counted as a command.

![Diagram of undo mechanism]

**Fig 5.3 A sketch about undo.**

**CLIPBOARD**  This item has three items that handles the clipboard:

- **Paste**  Copies the object from the clipboard to the window where you have to place it.
- **Cut**  Moves the selected object from the editor to the clipboard.
- **Copy**  Makes a copy of the selected object and places the copy in the clipboard.

**DELETE**  Deletes the selected object from the editor.

**DUPLICATE**  Copies the selected object and let you place by yourself.
The editor menu takes care of things that depends on the whole definition:

**OPEN**
Clears the editor and reads in a new definition from library or file.

**SHRINK**
Shrinks the editor and updates the definition. It also clears the history-stack.

**SAVE & EXIT**
Saves the definition and exits the editor.

**SKIP**
Aborts all changes of the definition since you last applied and exits the editor.

**SAVE**
Saves the definition on file.

**CLEAR**
Clears the editor.

**APPLY**
Sends and update message to SPL and clears the history-stack.

**HARDCOPY**
Prints the definition on paper.

**INPUT DEFINITIONS**
You can tell the definition how you are going to interact with it in SPL. You get an window where you for example can define which button will move the picture.

**DL ATTRIBUTES**
Edit presentation of DL-attributes.

**CHANGE NAME**
Renames the definition. The name is entered as a string.
6 SYMBOL AND CONNECTOR

TO CREATE OBJECTS

The lower part of the left palettes are made for object creation. You choose tool with the left button and then draw. There is some help-text written in a window at the top of the editor for your help. The menus with a little arrow to the right have a little submenu connected to it. You can there choose to draw an object like that in the item or the object in the submenu. All values, like line-width and dashing, are taken from defaults. These defaults are changeable. You change the defaults by choosing the item in the menu without having an object selected.

Fig 6.1 The symbol draw-tool palette

A newly created object is selected. All operations, that needs an object to operate on, will operate on the selected item. The selection disappear when you click in the window.

PATTERNS

This palette changes patterns on selected circles and rectangles. If there is no selected object it changes the default value to the pattern chosen and if the selected item is not a circle or rectangle nothing happens.

TO DRAW AN OBJECT

Every time you want to draw an object there will be instructions in a window on the top of the editor. This is what the commands will expect of you:

<table>
<thead>
<tr>
<th>TEXT</th>
<th>Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>Circle</td>
<td>First you have to give the center-position and then a position that is on the edge of the circle, i.e. the radius.</td>
</tr>
<tr>
<td>Ellipse</td>
<td>The ellipse draw command expects 3 points: the center-position, the position for the first axis, this position also gives the rotation for the ellipse, the last position tells how long the second axis will be.</td>
</tr>
<tr>
<td>Line</td>
<td>Two positions are expected.</td>
</tr>
<tr>
<td>Polygon/Closed curve</td>
<td>You give as many knots the polygon/curve will hold. You tells the editor to stop reading points by clicking outside the window.</td>
</tr>
<tr>
<td>Polyline/Curve</td>
<td>This commands act just as polygon and a curve respectively, but in this case closed figures will be drawn instead.</td>
</tr>
</tbody>
</table>
You give two knots that the rectangle has as opposite corners.

First you can choose between old bitmaps, creating a new one or to modify an old bitmap. If you choose to create a new one or to modify an old, a bitmap editor will be invoked and you can edit the bitmap in the way you want it. When the bitmap is designed you have to give a position were the center of the bitmap will be placed.

First a center-position is required, then the radius and the rotation of the arc. After that you shall give the angel of the arc.

**EDIT MENU**

![Symbol Edit Menu](image)

**Fig 6.2 The symbol edit menu**

**MOVE**

You will be able to move the extent of the object. The extent is moved with the mouse movements and placed when you push on the left mouse-button. The move command also start when you select an already selected object.

**SCALE**

The selected object will be scaled in two dimensions, horizontally and vertically. You will be asked for factors in both directions. Circles has one scaling-factor.

**ROTATE**

Rotates the selected objects towards a position that you gives, concerned to the left bottom corner of the objects extent. You will also be asked for the angle of the rotation.
LINES
There is two parameters that can be changed on lines: dashing and width. The item for width can be expanded for fixed widths and if you not are pleased with them you can choose the item WIDTH and you will get a little numeric keypad where you can tell the editor which line-width you want. When you choose DASHING you will be able to give list of numbers, ex (1 2 3), where every other number are values for coloured pixels and the other are for uncoloured pixels. These two commands changes the default values if there is not any selected object, and they are not undoable.

FONT
This command changes face, family and size of a string, and if there is no object selected the default values of those attributes changes.

bury
Buries the object.

to-top
Moves the object to top.

priority
You can change drawing priority on objects, choose between high and low priority.

THE EDITOR MENU
SHOW REFERENCE ITEMS
If you think that the selected item is marked indistinct you can choose to have nine points that marking center and corners of the object.

Fig 6.3 The figure that shows that an object is selected with reference items.

7 SYMBOL

There are two different kind of palettes to the left of the window. The four upper items forms one palette that influences on the behaviour in the symbol outside the editor. The rest forms one palette for creation of the symbol. At the bottom there is a palette for the pattern of circles and rectangles.

Fig 7.1 The symbol editor pattern palette

LEFT UPPER PALETTE
All items are from the beginning out-items and in-items, i.e. all items can be connected to connectors for flow in and flow out. With the two upper items you can tell the object to be (or not to be, it toggles) an in-item or an out-item. The third item makes the object active outside the editor and the fourth makes the item visible in the exported display.
All items toggles. If you want to have information about which objects that have the property you just pushes on the middle button on the item and all objects with that property is high-lighted.

THE EDITOR MENU

The symbol editors' editor menu differs from the connector editors' editor menu with just two items:

**SHADES**

Each symbol can have a set of shades that can be used in the derivation layer. A shade is a pair of a pattern and name. Shades are created in a simple bitmap editor. You can also edit parts of a shade, the name or the pattern and of course delete shades. If you want to create a shade you have to choose the subitem ADDSHADE. Then you will get a simple bitmap editor for you use: Click with the left button for black bits and the middle for white bits. Sometimes it is desirable to edit a shade, either the pattern and either the name. If you choose the item EDITSHADE you get a menu of shades to edit and then you get a menu mad of the name and the shade for you to choose between and change. When you choose to delete shades you get a menu of all shades bound to your definition.

**MARKERS**

Each symbol can also have a set of markers that can be used in the derivation layer. A marker is a pair of a pattern and name. Markers are treated as the shades and therefore you can add, edit and delete markers. The only difference is that it is a little bit more advanced bitmap editor.
8 CONNECTOR

In the first version the connector is a simple polyline.

LEFT UPPER PALETTE

All items are from the beginning from-items and to-items, i.e. all items can be connected to symbols for flow from and flow to. With the two upper items you can tell the object to be (or not to be, it toggles) a from-item or a to-item. With the first item you are able to tell how many parts of from-items and to-items respectively that are going to be shown in the editor. Usual there is a connector with one to-item and one from-item. The fourth item makes the object active outside the editor and the fifth makes the item dynamic. It is important to have at least one dynamic item in the connector, otherwise you can not move symbols and still keep the connector visually connected.

All items toggles. If you want to have information about which objects that has the property you just pushes on the middle button on the item and all objects with that property is high-lighted.
9 WINDOW

This editor is made for designing window to show graphs in. You are able to define everything that the windows functionality is dependent of. You can shape the window (and therefore the editor) by clicking with the left button in the small button on the right bottom of the window.

Fig 9.1 The window editor

The header is described before: shrink-button, move-button and the menu-button. To the left of the header is a menu with items for forming the outlook of the window.

THE FIXED MENU

You choose the item TITLE when you want to put a title on the window. Then you got a small window that you have to place by moving the mouse and paste it with the left button.

Fig 9.2 The title editor for the window.
In the upper field, prompted 'TEXT:', you can name the window. When you push the left button on the second field, 'PATTERN:', you will get a menu of patterns were you can choose the pattern that fits you best. You confirm your choices by clicking in the OK-button and cancel you choices with the ABORT-button.

![Diagram of window editor with options for TEXT, PATTERN, OK, and ABORT]

Fig 9.3 The title editor when a pattern is selected.

The title of the window will be placed in the upper left corner and the pattern fills out the rest of the title-line.
MENUES

When you want to add, edit and delete menu definitions you choose the item 'MENUES'. In the 'EDIT'-item you can choose to change the old definitions or create new ones. All items gives a formula (the same) to fill in but the item *NEW* also prompt for a name of the definition. End the definition name with a return. The formula will pop up in the left bottom corner of the screen and you have to place it by yourself.

![Derivation Menu Editor](image)

Fig 9.4 The menu editor

The following description will describe the formula. All fields that demands a written text as answer wants an answer from the small prompt-window on the top of the editor. End all field that you write in with a return.

**TITLE**
You can write the name of the menu.

**TITLE-FONT**
To change the font of the title you have to write a known family.

**APPLY-ITEM**
There is always two items that have to be in a menu in gathermode; apply-item and cancel-item. This will
cancel the name of the apply-item to something that you want. They will not be seen in pop-up menus for example.

CANCEL-ITEM
This will name the item that cancels the command that you have chosen.

BORDER
This field holds a menu of integers that you use when you want to tell how wide the border around every item in the menu.

CENTER?
Centers or left positions the text in the items.

CHANGE-OFFSET?
If you choose YES this makes the menu "start" on the item you choose the last time you were in this menu.

ROWS
You do not need to show all items in the menu, you can choose to show more or less rows than the number of items. This field will give a small menu where you can choose to how many rows you want to choose.

COLUMNs
A menu do not need to have one column, you can here form the menu in as many columns you like. You probably must change number of rows too, otherwise you get empty items in the menu.

FAMILY
All text in the items are by default written in Helvetica. Here you can choose any, of the system known, family.

SIZE
The size of the letters in the items.

POS
By clicking in this field you move the menu and thereby change the position interactively.

CONTEXT
The item that you choose in the menu is send to this context.

MODE
You can choose between gather- and send-mode. Send will execute the command immediately by sending the name of the item to the object in context. Gather collects all information and then waits for an apply-command for execution.

HOW
There are three different kind of menus; pinned, pop-up and continue, that you can choose between. A pinned menu is fixed-placed and always open. The pop-up menu pops up where the cursor is placed and disappears when an item is selected. The continue menu acts like a pinned but do not be attached to the window.

FN?
Send a call to a function instead of message to context.

ITEMs
When you click here you will get an editor where you can write all items that you want to have in your menu.

At the bottom of the formula a few buttons is placed that take care of definition:
ADD
Adds the definition of the menu to the window definition.

APPLY
Applies your choices on the definition. It also empties the history-list and therefore you can not undo things that you have applied.

QUIT
Applies and quits the menu-editor.

ABORT
Abort everything that you have done with the definition until last APPLY and quits the menu-editor.

The last item in this menu defines the special ICON that the window can have bound to itself. In this small editor you can edit a bitmap for the icon or write a text that will be in
the center of the icon and change the ICON-mask. If you choose to have both a bitmap and a text on the icon the text will not be seen. The ICON-mask tells how the bitmap will be shown. A black bit tells that the corresponding bit in the icon bitmap will be shown as it is, but a white bit lets the bitmap to have the background pattern in that place.

THE EDITOR MENU

The editor item has a few items that is different from the other editors:

In the subitem BEHAVIOR you are able to set some attributes for the window. They are all of the type ON/OFF:

CLOSEABLE    If the window are closable or not.
EXPAND&SHRINKABLE Sometimes it is useful to have a window that you can shrink when you need the space for it and expand when you want to use it again. Here you can tell if the window is expandable.
OPENED       If the window will open on creation or not.
SCROLLABLE   If you makes the window scrollable you don not need to have as big window as it hold the whole graph.
SHADED
SHAPEABLE    You can resize the window when you find it too small or too big.
TO-TOP       Will the window be buried or to-top.
UPDATEABLE   Sets the window to continously update the structure in it. Otherwise it only updates when you send an update message.

At last there is only a few left:
COORDINATES Set local coordinates.
SHAPE        You are able to shape the editor.
abort 14
active 8
add 14
apply 5; 14
apply-item 14
arc 7
bitmap 7
border 14
bury 8
cancel-item 14
center? 14
change name 5
change-offset 14
circle 6
clear 5
clipboard 4
closable 15
closed curve 6
columns 14
commands 3
context 14
coordinates 15
copy 4
create objects 6
curve 6
cut 4
default values 6
delete 4
dl attributes 5
draw 6
duplicate 4
ed-visible 8
editor menu 15
eclipse 6
exit 5
expand 15
family 14
fn? 14
font 8
hardcopy 5
how 14
icon 15
icon-mask 15
icons 2
in-items 8
input definitions 5
items 14
line 6
lines 8
markers 9
menus 2; 13
menus creation 13
mode 14
mouse 1
move 2; 7
open 5
out-items 8
paste 4
patterns 6
pickable 15
polygon 6
polyline 6
pos 14
priority 8
prompt-window 1
quit 14
rectangle 7
reference items 8
rotate 7
rows 14
save 5
scale 7
scrollable 15
selected items 6
shaded 15
shades 9
shape 15
shapeable 15
shrink 2; 5
shrinkable 15
size 14
skip 5
text 6
title 1; 2; 11; 13
title-font 13
to-top 8; 15
tools 1
undo 4
updateable 15
window 11