The User Interface of NUT[1]

Bo Andersson, Benjamin Volozh

June 1994
Updated December 1996

1. This document is available from `http://www.it.kth.se/labs/se/nut.html`

The User Interface of NUT

1.0 Introduction

The user interface of the NUT system offers the users a number of advanced functions, e.g. direct graphical interface to objects including automatic generation of menus with message names from class descriptions, algorithmic debugging of program specifications, visual tools for defining classes etc. The present document describes the most important parts of the user interface of NUT.

The user interacts with the system through the following windows:

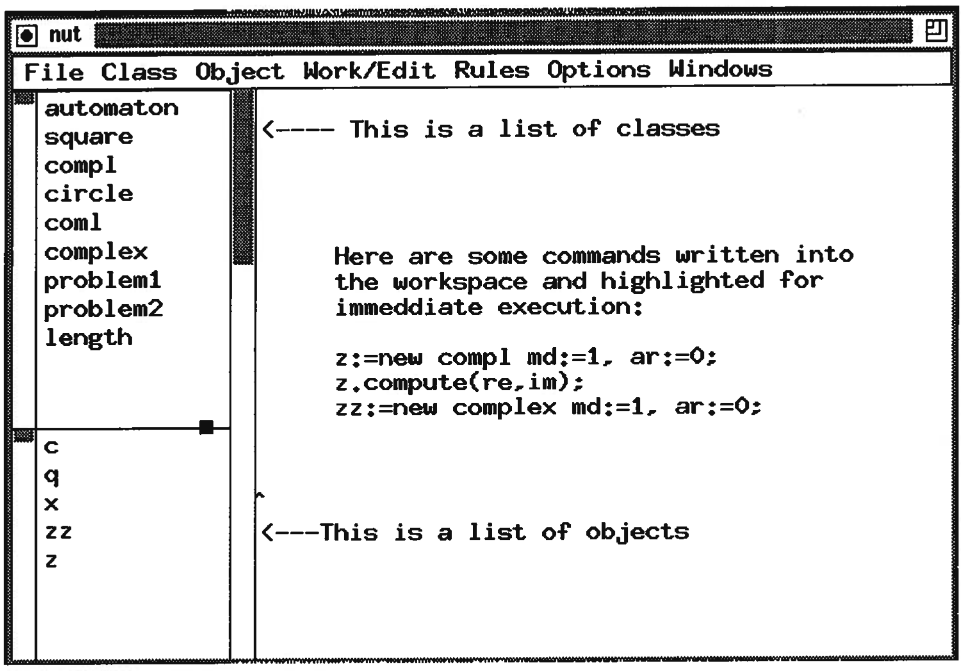
- the NUT main window (Sec 2)
- class windows, associated to classes (Sec 3.1)
- object windows, associated to objects (Sec 3.2)
- zoom windows, associated to components of schemes; they are a particular case of object windows (Sec 3.3)
- Graphics Editor windows: the main Graphics Editor window and image windows associated to classes (Sec 4.1)
- Scheme Editor windows: the main Scheme Editor window and scheme windows associated to classes (Sec 4.2)
- the Algorithm window (Sec 5)
- the Diagnostic window
- the Debug window
- the Rules window
- message and dialog windows

Since working with graphical editing windows has been thoroughly described in a separate document, *NUT Graphics*, our discussion in Sec 4 about these is very brief. The Diagnostics, Debug, and Rules windows are not discussed in the present document. Message and dialog windows are self-explanatory.

2.0 The NUT Main Window

The NUT main window (also called the package window) is opened when the NUT system is started. Closing the main window means exiting from the system.

.


nut
File Class Object Work/Edit Rules Options Windows
automaton
square
compl
circle
coml
complex
problem1
problem2
length
c
q
x
zz
z
<---- This is a list of classes
<---This is a list of objects

```
Here are some commands written into
the workspace and highlighted for
immeddiate execution:

z:=new compl md:=1, ar:=0;
z.compute(re,im);
zz:=new complex md:=1, ar:=0;
```

FIGURE 1. The NUT main window

In NUT, software is developed in the form of packages. In the system, one can work with one package at a time. The active package does always have a name. If the package was never saved, its name is *untitled*. Otherwise, the package assumes the name under which it was last saved. Packages are stored in the following file structure:

- the global state of the package (involving classes in text and compiled form, object states, the main window workspace text) in the file *package.mem*,
- icon representations of classes in the directory *package.icodir*,
- image representations of classes in the directory *package.imadir*,
- scheme representations of classes in the directory *package.schdir*,

where *package* is the name of the active package.

The main window is the "control panel" of the active package. This window contains a menu bar, and the following three areas:

- the class list area
- the object list area
- the workspace

The class list area contains a list of all classes of the active package. The object list area contains a list of all (global) objects of the active package. To select an object (class) in the NUT main window means to highlight its name in the list of objects (classes) by a click with the mouse.

The workspace is a text area intended for writing short programs by the help of which the user can interact with the active package. To perform a program, the user has to select its text with the mouse, and then apply the Perform command of the Work/Edit menu. All statements of the procedural part of the NUT language are available for writing programs in the workspace.

Below are explanations of the commands in the pull-down menus of the NUT main window. Opening an object (class) means opening a window for it. Closing an object means closing its window.

File

About	- displays a message window with information about the active package
New	- discards the current active package (the user is asked if he/she wants to save it); starts an empty package with the name *untitled.mem*
New (param.)	- the same as New
Open ...	- loads a package from a file (default suffix *.mem)*
Restore	- reloads the active package from a file
Save	- saves the active package
Save as ...	- assigns the active package a new name, and saves under this new name (the user is asked for the file name; default suffix is *.mem)*
To file ...	- writes the workspace and class texts of the active package into a file (the user is asked for the file name; default suffix is *.pac)*
From file ...	- merges workspace and class texts from a file into the active package (the user is asked for the file name; default suffix is *.pac*)
Exit	- exit from the NUT system

Class

New ...	- creates a new empty class (the user is asked to provide a name for it), and opens it
Open	- opens the selected class, if it is not open; a shortcut equivalent is double-clicking on the name of the class in the class list
Rename ...	- renames the selected class (the user is asked to provide the new name) [the selected class must be closed]
Duplicate ...	- makes a duplicate of the selected class (the user is asked to provide a name for it), and opens the duplicate class [the selected class must be closed]
Delete	- closes the selected class, if it is open, and deletes it [the selected class must be compiled]
Compile	- compiles the selected class [the selected class must be closed]
Make	- compiles all classes that need compilation, resolving cross-references; asks user confirmation for each class to be compiled [all classes must

	be closed]
Silent make	- same as Make, but doesn't ask for confirmations
Close all	- closes all classes that are open [all classes must be compiled]

Object

New ...	- creates a new instance of the selected class (the user is asked to provide the name for it), and opens it
Show	- opens the selected object, if it is not open, and sets the object's window into the show mode; a shortcut is double-clicking on the name of the class in the class list
Edit	- opens the selected object, if it is not open, and sets the object's window into the edit mode
Delete	- closes and deletes the selected object [all value fields in the window of the object must be closed]
Delete all	- closes and deletes all objects [all value fields in the windows of the objects must be closed]
Close all	- closes all objects [all value fields in the windows of the objects must be closed]

Work/Edit

Perform	- executes the selected text [it must be a syntactically correct program]
Cut	- moves the selected text into the clipboard
Copy	- copies the selected text into the clipboard
Paste	- pastes the text from the clipboard into the text area where the cursor is
Undo	- undoes the effect of the last editing command
Select all	- selects all text in the workspace

Rules

Open	- opens the Rules window
Print rules	- shows rules of the selected class in the Rules window
Produce	- runs the production system

Options

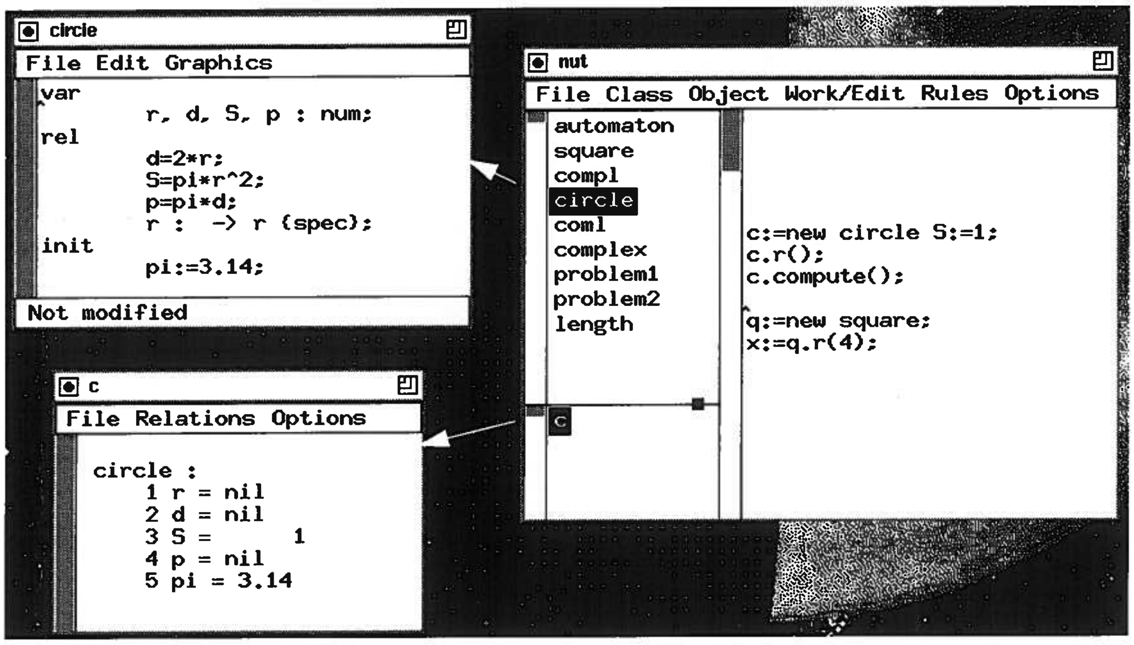
Gen. code	- switches code generation on/off
Remove garbage	- switches garbage collection on/off
Minimize	- switches minimization of synthesized algorithms on/off
Expand subproblem model	- switches the global expansion of specifications during planning on/off
Wide context	- switches the wide context for planning on/off
Equation systems	- switches solution of equation systems on/off
Interpreter info	- switches the printing of the interpreter information into the xterm window on/off
Object reduction	- switches the automatic coercion of objects in arithmetic expressions on/off

Windows

Algorithm	- opens the Algorithm window
Diagnostics	- opens the Diagnostics window
Debug	- opens the Debugging window
Graphics	- opens the main Graphics Editor window
Scheme editor	- opens the main Scheme Editor window
Rules	- opens the Rules window.

3.0 Class and Object Windows

Packages are developed in NUT by editing class texts in class windows and interacting with the package through object windows which can be opened for each class and object listed in the NUT main window. Fig. 2 shows a typical situation on the screen when working with NUT. After double-clicking on a class name, a class window is opened. When an object of interest is encountered, double-clicking on its name in the object list opens a window for this object.



```
var
        r, d, S, p : num;
rel
        d=2*r;
        S=pi*r^2;
        p=pi*d;
        r :  -> r {spec};
init
        pi:=3.14;
Not modified
```

```
  circle :
      1 r = nil
      2 d = nil
      3 S =      1
      4 p = nil
      5 pi = 3.14
```

```
c:=new circle S:=1;
c.r();
c.compute();

q:=new square;
x:=q.r(4);
```

FIGURE 2. Class and object windows

3.1 Class Windows

Class windows are used for the textual definition of classes. Each class is edited and compiled in its own class window. To open a class means to open a class window for it. To close a class means to close its window. Whenever an open class is deleted, it is first closed.

A window can be opened for a class in one of the two following ways: by double-clicking with the mouse on the class name in the class list area of the NUT main window, or by using the Open command of the Class menu in the same window.

The window of a class contains a menu bar, a text area for the text of the class and a message bar. The class text can be edited and compiled. The message bar displays Modified or Not modified depending on whether the class text has or has not been edited since it was last compiled.

The following commands are available from the pull-down menus of class windows:

File

Compile	- compiles the class
Revert to previous	- restores the class text to what it was when the class was last compiled successfully, or to empty text, if the class was never compiled successfully
Close	- closes the class window

Work/Edit

Cut	- moves the selected text into the clipboard
Copy	- copies the selected text into the clipboard
Paste	- pastes the text from the clipboard into the text area where the cursor is
Undo	- undoes the effect of the last editing command
Select all	- selects all text in the text area of the window

Graphics

Edit icon...	- starts the standard *bitmap* program of the X Windows System (see its manpage) for editing the icon of the class (Save saves the icon under the appropriate name); the icon represents the class in the palettes of Scheme Editor windows
Edit image...	- opens a Graphics Editor window for the class for editing of the image of the class; the image represents the class in the drawing areas of Scheme Editor windows
Edit scheme...	- opens a Scheme Editor window for the class with the purpose of editing its scheme representation

3.2 Object Windows

Object windows serve as interaction media between the user and the state of the active package. They can be used in two different modes: *show* and *edit*. The first mode is used when only the visualisation of values of components of the object of interest is needed. The second mode is intended for data exchange with the object. In this mode, values of components of the object can be edited (changed). In both modes, messages can be sent to the object. This invokes computations on the object which in their turn can invoke computations on other objects. The pull-down menu Relations is built in accordance with the class of the object and it contains the names of all its named procedural relations. A message is sent to the object by selecting the relation name from the menu.

3.2.1 Opening an Object Window

Every object window is associated with a particular object. Whenever an object is deleted, first its window is closed.

To open an object window, you must, first, go to the NUT main window (package window), and select an object in the object list, and, second, from the **Object** menu group choose one of the commands:

Show	- to view the object
Edit	- to edit the value(s) of the object or its components.

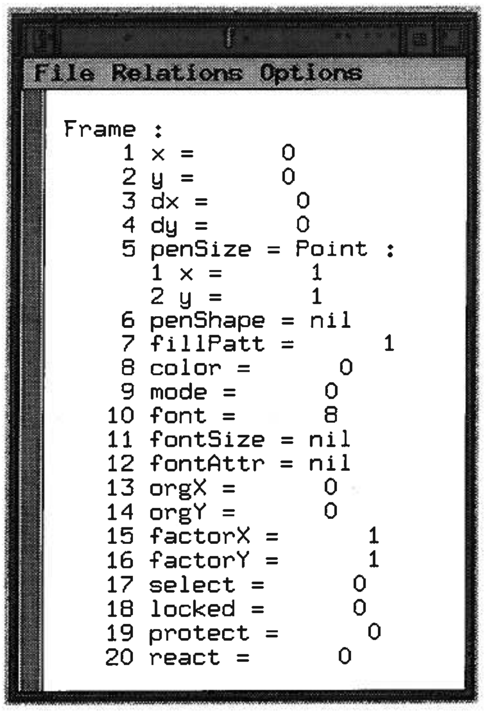
Both commands open the same object window in corresponding modes. These commands will be disabled in the menu, if no object is selected in the list.

The mode for an opened object window can be changed later either from its **Options** menu or from the **Object** menu of the NUT main window. Instead of creating a new object window that will bring up the existing one and set the appropriate mode for it.

Double-clicking with the mouse on an object's name in the list of objects of the NUT main window is equivalent to the **Object | Show** command.

A layout for an object window is generated automatically according to the class of the object. With default options, the window layout will be equivalent to the that of the object printout produced by the *NutPrint* function. If needed, the object window can be resized and its contents scrolled.

The layout of an object window is controlled by a set of options that can be selected from the **Options** menu.



```
Frame :
    1 x =       0
    2 y =       0
    3 dx =       0
    4 dy =       0
    5 penSize = Point :
      1 x =       1
      2 y =       1
    6 penShape = nil
    7 fillPatt =       1
    8 color =       0
    9 mode =       0
   10 font =       8
   11 fontSize = nil
   12 fontAttr = nil
   13 orgX =       0
   14 orgY =       0
   15 factorX =       1
   16 factorY =       1
   17 select =       0
   18 locked =       0
   19 protect =       0
   20 react =       0
```

FIGURE 3. An object window in the *show* mode.

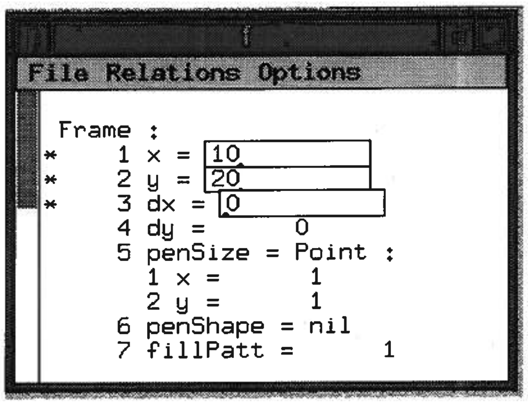
3.2.2 Editing Object Component Values

To edit a value displayed in an object window, just click on the corresponding line with the mouse. Make sure that the window is in the *edit* mode: in the Options menu you can check or change the mode.

After a value has been selected for editing, a text field with a rectangular frame will surround it. You can edit the value just as any other text: by typing on the keyboard and selecting characters with the mouse. Keep the cursor (the I-beam) within the field to direct the keyboard input into that field. On typing outside a text field, an object window will normally react with a beep.

An asterisk (*) appears in the first position of a line being edited.

There can be more than one field edited at a time, and the window can be scrolled, if needed. While scrolling, the framed fields with edited values are temporarily removed from the screen, and the asterisks highlight the lines with unsaved changes. Long values in these lines may appear truncated to their previous length while scrolling.



```
 Frame :
*    1 x =  10
*    2 y =  20
*    3 dx =  0
     4 dy =       0
     5 penSize = Point :
       1 x =      1
       2 y =      1
     6 penShape = nil
     7 fillPatt =      1
```

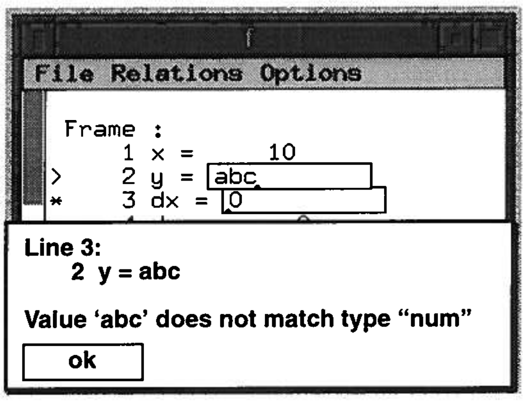
FIGURE 4. An object window in the *edit* mode.

Applying changes

To apply changes to the object, do one of the following:

- From the File menu choose Apply to update all values being edited. As a result all fields will be closed.
- In a field press the **Enter** key to update the corresponding value. That will close only the particular field.

For each field NUT checks whether the specified value matches the corresponding class. If the check fails, an error message will be displayed, and the > sign will highlight the line where the error occurred:



```
Frame :
     1 x =       10
>    2 y = abc
*    3 dx = 0
```

Line 3:
2 y = abc

Value 'abc' does not match type "num"

FIGURE 5. A sample error message.

For objects of class *text* the value can be entered as it is or included into single quotes, e.g. `'the string'`. However, the quotes are obligatory when a text string is assigned to an object of class *any*.

If an empty value is applied, that will mean *nil* for numeric objects (*num*) and an empty string for other classes, i.e. *text* and *any*.

A value can be cancelled, i.e. *nil* assigned, in one of the following ways:

- Type `nil` as the value and apply it.
- In the field press the **Delete** key.

Cancelling changes

To cancel changes before applying them to the object:

- From the File menu choose Revert to Previous to close all fields.
- In the particular field press the **Escape** key.

3.2.3 Executing Object Methods

From the Relations menu you can perform computations on the object or execute its methods (relations).

Choosing the Compute command is equivalent to executing the *compute* method for the object in the workspace window, for example:

```
    obj_name.compute( arg_list );
```

where *obj_name* is the object name and *arg_list* is the list of arguments (here - names of the components to be computed) as specified with the Arguments command, if any.

The **Relations** menu contains the names of all named procedural relations of the object's class as commands. Execution of such a command is equivalent to a usual relation call. E.g. in the workspace of the NUT main window one could perform:

```
    obj_name.rel_name( arg_list );
```

where *obj_name* is the object name, *rel_name* is the relation name and *arg_list* is the list of arguments as specified with the **Arguments** command, if any.

The list of actually available relations may change when the class of the object is recompiled. To update the **Relations** menu you must close the object window and reopen it (make sure that the object really corresponds to the new class specification!).

With the **Arguments** command you can specify a list of arguments to be used with subsequent commands invoking *compute* or class-specific relations. This list is not saved when you close the window.

3.2.4 Menu Commands

File

Apply	- updates all values and closes all fields. This command is available only if some values are being edited.
Write to File...	- writes contents of current window to a specified file (the user is asked to provide the file name; default suffix is *.txt*)
Revert to Previous	- reverts to the object's previous state (closes all fields without applying changes). This command is available only if some values are being edited.
Close	- closes the object window. If there are any fields open, asks for applying changes. You can select one of the options: Yes: apply changes and close the window; No: close the window without applying changes; Cancel the command.

Relations

Compute	- performs the *compute* method on the object using the arguments specified with the **Arguments** command.

(relations)	- an optional group of class-dependent commands: each item is a name of a procedural relation in the object's class; choosing a name invokes performing of the corresponding relation on the object using the arguments specified with the **Arguments** command
--------------	
Arguments...	- opens a dialog box where the user can tell the arguments for a *compute* or a relation to be invoked through this menu

Options

Show	- sets the object window mode to *show* (default mode)
Edit	- sets the object window mode to *edit*
Virtuals & Aliases	- switches the displaying of virtual components and aliases on /off (default if off)
Row	- switches the displaying of row elements on/off (default is off).
Lines...	- sets the limit for the number of displayed lines (default is 200).
Levels...	- sets the limit for the number of displayed levels of the object structure (default is 4)
Save for Class	- saves the current display options (not the mode!) as the default for the windows of objects of the given object's class

3.2.5 Object Window Options

Object display options, i.e. Virtuals & Aliases, Row, Lines and Levels, are normally associated with a particular object window. If the associated object value is changed by some computations, this will not affect the options. When the associated object is deleted, the window is closed, and the settings are lost.

Invoking Options | Save for Class command in an object window stores the current display options in the object's class. All subsequently opened windows for objects of the same class will employ these settings.

3.3 Zoom Windows

Zoom windows are a special case of object windows, associated to components of schemes. They are a little bit special, since they do not represent global objects, but rather components of schematically represented classes. The usage of the value editing facility of zoom windows is different from that of normal object windows: value editing is used for adding initialisations to the class associated with the scheme. Zoom windows can also be used for viewing the results of computations performed on this class (correctly speaking, on an instance of it).

The default mode for zoom windows is *edit*. Open value fields in a zoom window are stored with the scheme whenever the scheme is read in to the class or the scheme is closed. In the class text, open value fields translate into initialisation amendments. Usual application of value changes to an object does not work. A thorough discussion of the particularities and the usage of zoom windows can be found in *NUT Graphics*.

The zoom window for a component of a scheme can be opened by selecting the image representing the component on the scheme and then selecting the Zoom in command in the Elements menu of the scheme window.

4.0 Graphical Windows

There are two kinds of graphical windows in NUT: Graphics Editor and Scheme Editor windows.

4.1 Graphics Editor Windows

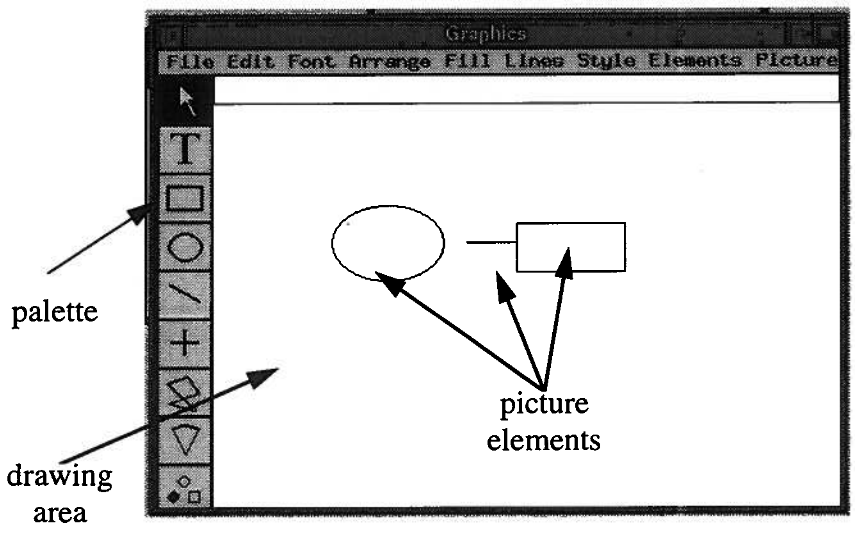
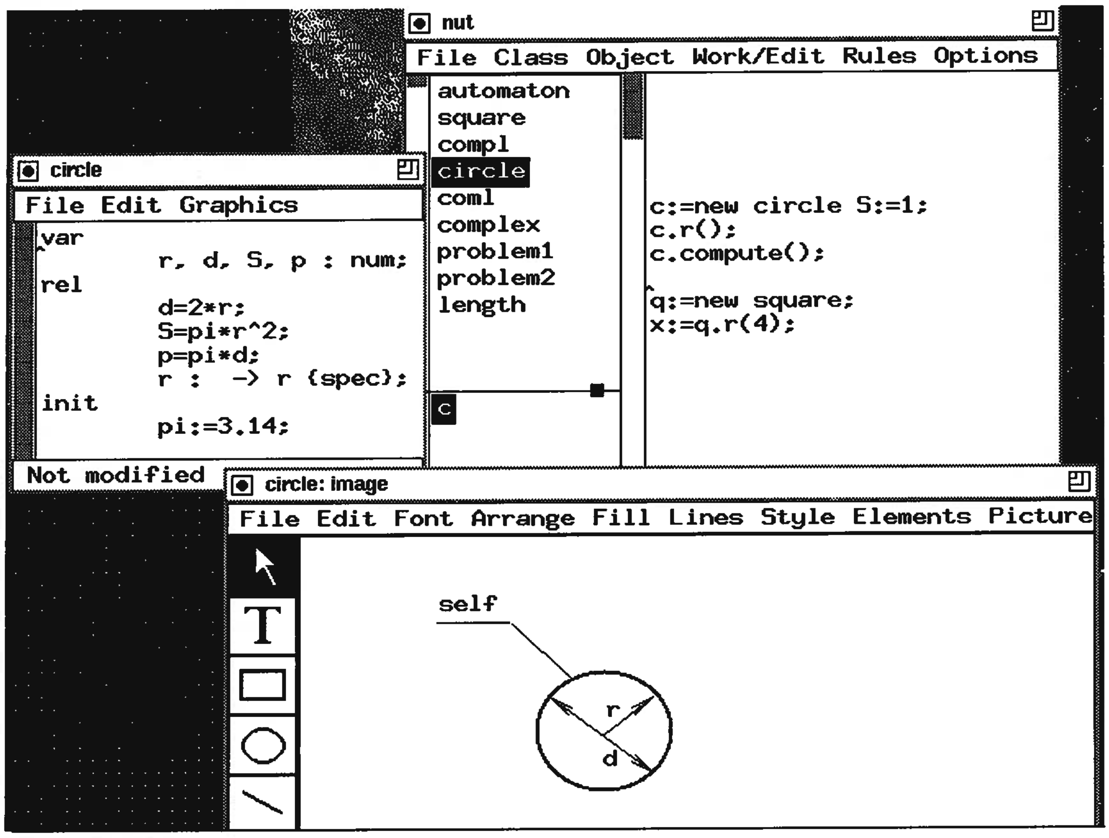


FIGURE 6. A Graphics Editor window

The Graphics Editor of the NUT system is a 2.5D graphics editor by means of which pictures formed out of simple picture elements such as lines, polylines, rectangles, circles, arcs, pies, chords and groups of these can be drawn. The usage of the Graphics Editor is twofold in NUT, and correspondingly there are two kinds of Graphics Editor windows.

The main Graphics Editor window (also called the default Graphics Editor window) serves as a graphical interaction window between NUT programs and the interactive user. This is possible due to the existence of standard functions in the NUT language for interacting with the main Graphics Editor window: a program in the NUT language can manipulate the picture in the window by calls to these functions while the user can manipulate the picture manually using the drawing tools from the palette and the menu commands of the window. It is important that picture elements in the main Graphics Editor window can be linked to objects of the package. The main Graphics Editor window is opened with the Graphics command of the Windows menu of the NUT main window.

Image windows can be opened for user-defined classes. The image window of a class is used only for defining the graphical image of the class (the user has to draw the image, it is not possible to draw anything into the image window from a NUT program). On the schemes of other classes, this graphical image represents components having this class. The image window for a class can be opened with the Edit image command of the Graphics menu of the text window of the class.



```
var
          r, d, S, p : num;
rel
          d=2*r;
          S=pi*r^2;
          p=pi*d;
          r :  -> r {spec};
init
          pi:=3.14;
```

```
c:=new circle S:=1;
c.r();
c.compute();

q:=new square;
x:=q.r(4);
```

FIGURE 7. **Editing the image of a class**

The operation and usage of the Graphics Editor is explained in full detail in *NUT Graphics*. The standard functions of the NUT language for interaction with the main Graphics Editor window are described in *The NUT Language Report*, and in *NUT Graphics.*

4.2 Scheme Editor Windows

The Scheme Editor is and extension of the Graphics Editor and is used as a tool for schematic (visual) definition of classes.

The main picture elements of which schemes are formed are images of classes. On schemes, class images represent components (having the corresponding classes) of the class represented by the scheme. Class images themselves are pictures formed of simple picture elements and defined in class image windows using the Graphics Editor (see Sec 4.1). Other important picture elements much used on schemes are different kinds of connections that represent equivalence links between components in the class represented by the scheme.

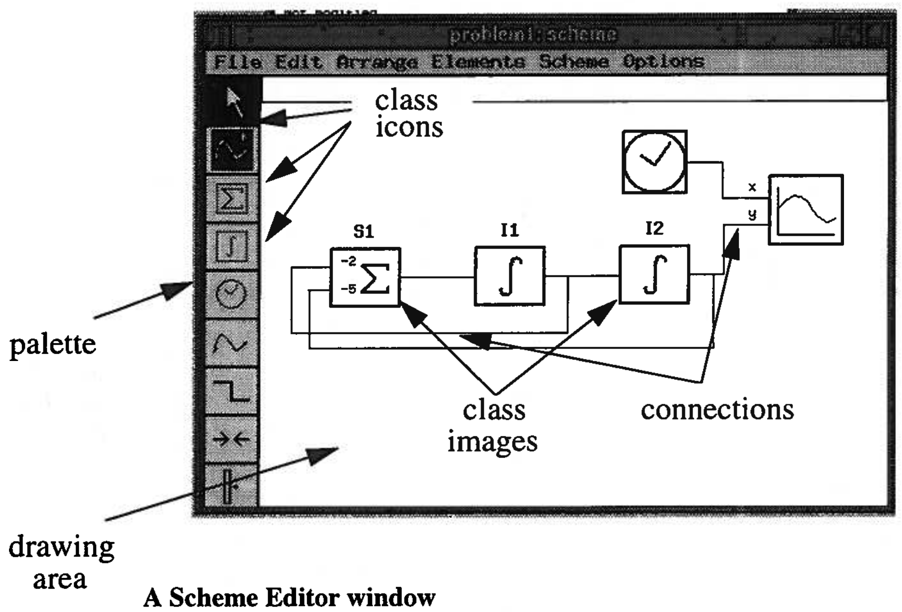
In the palette of the Scheme Editor windows, classes are represented by their icons. Class icons are bitmaps and they are defined using the standard *bitmap* editor of the X Window

System. The palette of any Scheme Editor window contains icons of all classes of the active package for which an icon and an image have been defined.

The scheme of a class is defined in a scheme window for that class. The user draws the scheme of the class using the tools from the palette and the menu commands of this window. The scheme window of a user-defined class can be opened with the **Edit scheme** command of the **Graphics** menu of the text window of the class.

Besides scheme windows associated to classes, there is the main (or default) Scheme Editor window. This window can be opened with the **Scheme Editor** command of the **Windows** menu of the NUT main window. If the user asks the system to compose or compile a class from the scheme in this window, or to compute something on the scheme, the class name *SCHEME* will be used (a class with this name is created).

.



A Scheme Editor window

The operation and usage of the Scheme Editor is explained in full detail in *NUT Graphics.*

5.0 The Algorithm Window

The Algorithm window is used in the NUT system to visualise the algorithms synthesized by the NUT planner in computations. The window contains a menu-bar and a text area (non-editable).

The Algorithm window can be opened with the Algorithm command of the Windows menu of the NUT main window. At any moment, the window displays the skeletons of the algorithms that were synthesised by the planner during the last program execution invoked by the Perform command of the Work/Edit menu of the NUT main window or *compute* or method execution on an object invoked by the corresponding commands in the Relations menu of the window of the object.

Below is a description of the commands in the menu-bar:

Close	closes the window
Clear	clears the text area
To file	saves the algorithms in the text area into a file (the user is asked for a name; default suffix is *.txt*)

The mechanism of automatic synthesis of component values and relations in NUT is explained in *The NUT Language Report.*

6.0 References

B. Volozh, M. Kopp, E. Tyugu. (1993) NUT Graphics. Technical Report TRITA-IT R 93:05, Dept of Teleinformatics, The Royal Institute of Technology, Stockholm.

T. Uustalu et al. (1994) The NUT Language Report. Technical Report TRITA-IT R 94:14, Dept of Teleinformatics, The Royal Institute of Technology, Stockholm.