On Formalising Graphical User Interfaces (GUIs)*

Dines Bjørner
Department of Information Technology
Technical University of Denmark
DK–2800 Lyngby, Denmark
E–Mail: db@it.dtu.dk

September 18, 2000

Abstract

Ww show, by example how simple it is to formalise the notion of the window graphics of a graphical user interface with windows, windows within and windows next to windows, with simple icons, icons with scroll down bars, and icons with windows. We define a class of window types and window values. The presentation evlves from a *Rough Sketch* scenario, via a combined *Analysis* and *Narrative*, to a *Formalisation*.

This technical note also illustrates some specification and desgn principles. These are dutifully recorded.

Contents

This document is presently in a very DRAFTy stage !

*This technical note has been pre[ared, March 13–14, 2000, for the benefit of Mr. Ths. Andersen's MSc Thesis work.

1 Background

Formal specification has shown itself very useful in the abostract modelling of domains, requirements and software designs. Classically abstraction hs been applied to conceptual matters: *What is a Programming Language ?, What is a Banking System ?, What is a Railway System ?* etcetera. In [1] and many later references [2, 3, 4, 5, 6, 7], formalisation was first shown useful in connection with user Interface specification.

The author of this note finds that many "formalists", students and engineers alike, still do not see the benefit of abstraction — especially when it comes to such seemingly concrete, operational, ideas as graphics user interfaces, hereafter abbreviated, sorry, by GUI. The simple fact that windows, icons, curtains, scroll bars (See however [8]), etc., whether clicked or not, and their screen effect, can be simply formalised, still escapes most.

This little technical note shall therefore serve as a temporary repository for notes on GUI specification: A teaser, a sampler.

It is expected that the message of this note, in some form or another, will find its way into the author's current lecture notes's chapter 10, section 10.5 (Interface Requirements) and later chapters (Software Architecture, Program Organisation, etc.).

2 Rough Sketch Scenario: An Example GUI Session

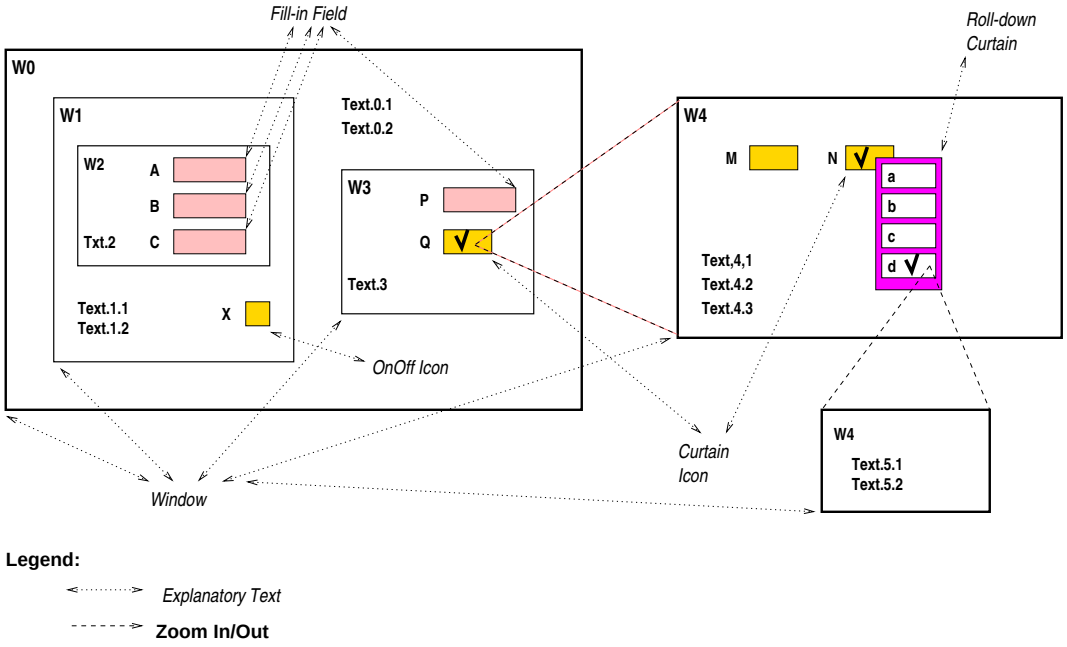
Figure 1 on the facing page shows an example GUI session: Window W0, to the left, is to be thought of as the first window. Window W4, to the right, came about as a result of clicking an icon inside W1. W1, when first displayed on the screen looks very much like you see it: With properly embedded, fixed position sub–windows (or really just aggregate fields). These are labelled W1 and W3. Window W1 again has a proper, fixed aggregate field W2. Windows Wi, for i=0,...,5, has some text, Text.i.j as well as the window labels Wi (for those i). Window W1 has an icon, labelled X. Window W2 has three fields, lablled A, B and C. The user is supposed to fill in the grey fields next to these labels. Window W3 also has a subfield, labelled P for text to be filled-in, and an icon. Pressing icon X (of Window W1) may be thought of as changing an underlying state, the icon is highlighted when clicked (X is shown as not clicked !). Icon Q of Window W3 has been clicked ($\surd$, the highlighting symbol), and when clicked it led to the creation of window W4. It has two icons. Icon N has been clicked and led to the appearance of the scroll down curtain, also referred to as N. It has four fields: a, b, c and d. Icon d has been clicked ($\surd$) and led to the creation of Window W5.

3 An Analysis and a Narrative

3.1 Entities Etcetera !

- **Entities:** The basic entities (things, individuals) are as follows:
  - *The GUI:* The graphical User Interface manifests itself as a series (a set) of uniquely named windows.
  - *Windows:* Windows have (i) a position (relative to the embedding window, if embedded, and otherwise are freely movable), (ii) a horisontal width and a vertical height. Windows consists of (iii) some fixed texts, positioned and of some maximum

Figure 1: A Snapshot GUI: Graphical User Interface



An Example Graphical User Interface: A Snapshot

length, and of some type. Windows also consists of zero, one or more embedded uniquely named window items.

– *Window items:* A window item is either a (iv) window, (v) a field [to be filled in by users], (vi) a simple on/off icon, or (vii) a curtain icon. [Embeddedness and uniquely naming applies to each kind of window item (iv–vii).] All window items (iv–vii) have position, width, height and possibly other attributes (colour, hue, etc.). Window items may have horisontal and/or vertical scroll bars. These are used to reposition the conTents of a "large" window item within the "frame" of a "small" window. (We leave the double quoted terms unexplained for now.)

– *Field:* Fields are pairs of (field) names and a text field, where this text field is intended to be filled-in by a user. Fields have:

  ⋆ Type: [a] character, [b] text [ie. characterstring], [c] integer, [d] Boolean, or some other kind (could be vector of such texts, matrix { table }, or some higher dimension array), etc.

  ⋆ Value: Either unspecified or of the type (somehow) indicated.

– *Icon:* We exemplify three kinds of icons:

  ⋆ On/Off Icon: On/off icons are either set ("on") or reset ("off"). The setting/resetting of an icon is reflected in the value state of the window, and hence the GUI. The value is some enumerated type or Boolean.

  ⋆ Window Icon: A window icon is either set ("on") or reset ("off"). The setting/resetting of an icon is reflected in the value state of the window, and

hence the GUI. When set, another window, "disjoint" from the window of the clicked window icon, is shown. When reset it is removed.

⋆ Curtain Icon: A curtain icon is either set ("on") or reset ("off"). The setting/-resetting of an icon is reflected in the value state of the window, and hence the GUI. When reset, a curtain is not shown on the screen, but its value is still part of the state. When being set, and when set, the curtain appears somehow "next" to its icon. We exemplify two kinds of curtains.

– *Fixed Curtain:* Fixed curtains consists of a fixed number of uniquely named sub-icons (which are icons), and which are either set or reset.

Fixed curtains may have a vertical, and even a hortisontal, scroll bar.

The setting of a fixed, or, as we shall see next, a scroll-down, curtain icon results in the display of its syntactic "meaning": some window item.

– Scroll–down Curtain: Scroll down curtains consists of a variable number of uniquely named sub-icons (which are icons), and which are either set or reset.

- **Operations on Windows:**

– *Definition:* The application system developer can define windows: naming them, giving their contextual descriptions: Positions, dimensions, fixed texts, number and types of window items, and their "embedded" quantities. So we speak of a window type. Window type definitions are named and can be filed, retrieved (for further editing, ie. defining) and instantiated (for future use). So we speak of a window type definition catalogue and a database of instantiated windows.

– *Use:* The user can operate upon an instantiated window: Reading the window, an operation that does not change the window state, (i) filling-in fields, (ii) setting and resetting icons, (iii) and browsing — using scroll bars — the window items. Operations (i–ii–iii) change the window state.

- **The GUI System:** The GUI system consists of a window type definition catalogue, a database of instantiated windows, and one or more uniquely named sessions.

– *GUI Session:* A GUI session consists of a root, the initial, window

⋆ (Definition Session:) type definition and a set of window type definitions, or

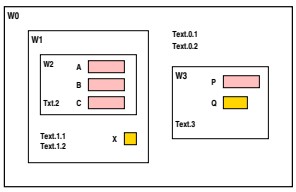
⋆ (User Session:) — which is then an instantiated window, a set of instantiated windows,

"derived" from, ie. related to the root window.

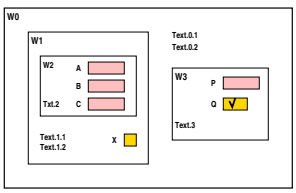
3.2 A Session Example

We show, without explanation, a session sequence:

- Stages 1, 2:

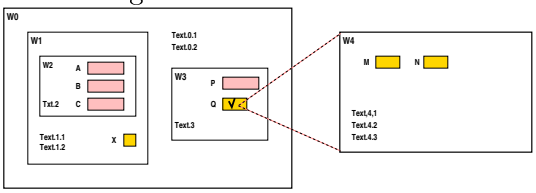


An Example Graphical User Interface Session: Stage 1

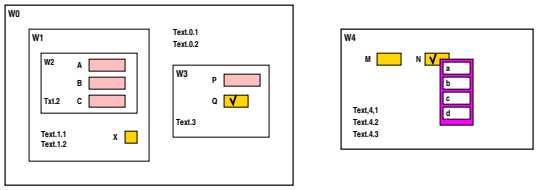


An Example Graphical User Interface Session: Stage 2

- Stages 3 and 4–5:

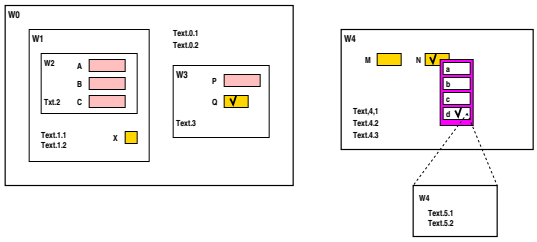


An Example Graphical User Interface Session: Stage 3



An Example Graphical User Interface Session: Stages 4-5

- Stage 6:



An Example Graphical User Interface Session: Stage 6

4 A Formalisation

4.1 Window Types: Static and Dynamic

Types of windows can be defined. A window type expression is a static semantics value which can be manipulated. A window type expression denotes a possibly, ie. most likely infinite set of window values. First we deine

4.1.1 Window Type Expressions

In designing the syntax of the types that govern the form and shape of windows and their embedded items, we follow a principle of homomorphic and recursive design: The recursion principle states that whenever an embedded quantity is desired, then one should examine whether one wants that embedded quantity to be of the same kind as the "whole thing".

type

0. Wn, Enum
1. WTEx = TT-**set** × (Wn $\overrightarrow{m}$ (TT × WI))
2. WI == FixeWI | FieldWI | IconWI | CurtWI

3a. FixdWI :: (nil|embedded|linked) × WTEx

4. FieldI :: FT
5. IconWI :: IconTp
6. CurtWI :: (FT × WI)*
7. IconTp == mkOnOffT(oo:Enum-**set**) | mkFixCurT(f:CurtWI) | mkVarCurT(v:CurtWI)
8. TT = XY × HW × FT × HSc × VSc
9. FT == symbol | text | integer | boolean
10. HSc, VSc == nil | mkSc()
11. XY, HW = **Nat** × **Nat**
12. WTCtlg = WTNm $\overrightarrow{m}$ WTEx

Annotations:

- A window type expression, WTEx, expresses that a window has some texts, TT-**set,** and zero, one or more uniquely named, Wn, positioned, sized, annotatable and possibly horisontally and possibly vertically scrollable, TT, window items, WI.
- A window item is either a fixed window item, a field window item, an icon window item or a curtain window item.
- A fixed window item, FixdWI, is a window type expression and defines a possibly optional ("hidden") window that is either properly embedded in the window immediately surrounding the window of the window item, or is linked to it, but appearing as a separate window possibly overlapping with it.

  Later we shall name some, the defined window type expressions. Then we shall replace equation [3] by another equation [3′].
- A field window item is positioned, sized and annotated and defines that annotation (to appear at that position and in a "box" of the deined size).
- An icon window item is of either of three kinds: A simple finite set of one or more, usually less than 10, enumerations which define such a set of screen "button" to be either clicked ("on") or not clicked ("off"), or is a fixed curtain or a variable curtain window item which both define a finite set of curtain window items.
- A curtain window item is a list of window items, each is annotatable, and defines the possibility, through the "clicking" of repective curtain list element "annotation buttons", that the corresponding window item value can be displayed.
- The positioned, sized, annotated and scrollable annotations define either single characters and other symbols, or variable length texts, or integers or Booleans.
- Window items have a frame (only implicitly defined) which allow for scrolling window item images horisontally, HSc, and/or vertically, VSc.

4.1.2 Window Values

In designing the semantics of types, we follow a principle of homomorphism: The value structure follows the type structure.

The window value definitions i/j below thus correspond to window type expression definitions i above.

type

1/12. WV = TV-**set** × (Wn $\overrightarrow{m}$ (TV × WIV))
2/13. WIV == FixdWV | FieldWV | IconWV | CurtWV
3/14. FixdWV :: OptWV
3a/15a. OptWV == nil | mkENilWV(w:WV) | mkEWV(w:WV)
| mkLNilWV(w:WV) | mkLWV(w:WV)
4/16. FieldV :: Field
4/17. Field == nil | mkNilField(f:VAL) | mkField(f:VAL)
5/18. IconWV == mkOnOffV(oo:Enum-**set**) | mkFixCurV(f:CurtWV) | mkVarCurV(v:CurtWV)

6/19. CurtWV :: (TV × CW)*
7/20. CW == nil | mkNilCW(w:WV) | mkCW(w:WV)
8/21. TV = XY × HW × VAL × HScroll × VScroll
9/22. VAL = **Char** | **Text** | **Int** | **Bool**
10/23. HScroll,VScroll = **Nat**

Annotations:

- A window value, WV, corresponds to some definining zero, window type expression: Has texts, TV**-set,** and one or more uniquely named, Wn, window item values, WIV. The texts are positioned, occur inside sized boxes, and otherwise consists of text values of a defined type.
- Window item values are either fixed, field, icon or curtain values.
- A fixed window value is either present, and is then shown or is not shown, or is absent. In the showable case the fixed window value is, indeed, a window value. It can either be properly embedded, or appears as a separate (movable) window.

  Later we shall name some, the defined window values. Then we shall replace equation [3a/15a] by another equation [3b/15b].
- A field window value is either present, and is then shown or is not shown, or is absent. In any case the field window value is an annotation, which — if not present — remains to be filled-in by the user, and if present, can be so edited.
- An icon value is either a simple on/off set of clicked enumerations — a possibly empty subset of the enumerations given in a corresponding window type expression, or is a fixed curtain value, or is a variable curtain value.
- A curtain value list is a list of annotations and optional curtain values. An optional curtain value which is present, ie. non–Nil, corresppponds to the list item annotation being clicked. Not clicked means that window value is not shown.

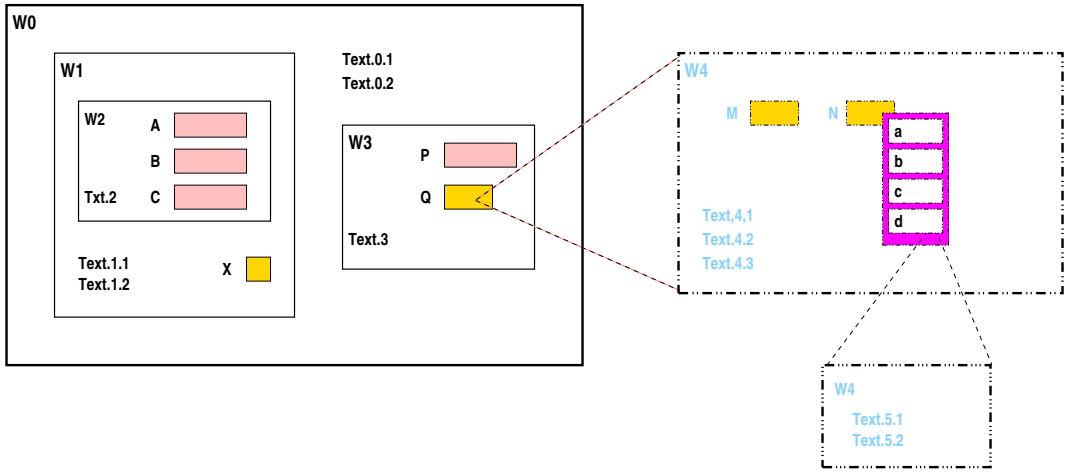
Figure 2 on the next page shows a window value where all icons have been reset. The window items that have thus been "turned off" are shown with dash-dot-dotted lines. They are part of the full window value but "hidden" from view.

4.1.3 GUI: The Graphical User Interface System State

We now define the configuration of a GUI system as consisting of an evironment and a state (22.). The environment consists of all defined and hence name window types and all created and hence window values (23.). The state consists of one or more uniquely named sessions, with one of these session being singled out (ie. named) as the one currently being operated upon (26.). A session consists of a list of window/sub-window, etc., snapshots (27.–28.).

type
22. GUI = ENV × Σ
23. ENV = WTCtlg × WVDaBa

Figure 2: A Window Value: Shown and "Hidden" Items



An Example Graphical User Interface Session: "Hidden States"

24. WTCtlg = WTNm $\overrightarrow{m}$ WT
25. WVDaBa = WVNm $\overrightarrow{m}$ WV

3b. FixdWI :: WTNm

3b/15b. OptWV == nil | mkENilWV(wn:WVNm) | mkEWV(wn:WVNm)
| mkLNilWV(wn:WVNm) | mkLWV(wn:WVNm)

26. Σ = Sessions × SNm
27. Sessions = SNm $\overrightarrow{m}$ SLst
28. SLst = (WV × (Wn $\overrightarrow{m}$ WV))*

value

29. gui:GUI = ((ρ,δ),(sess,s))

30.0 inv_GUI: GUI → **Bool**
30.1 inv_GUI((ρ,δ),(sess,s)) ≡ ...

Annotations:

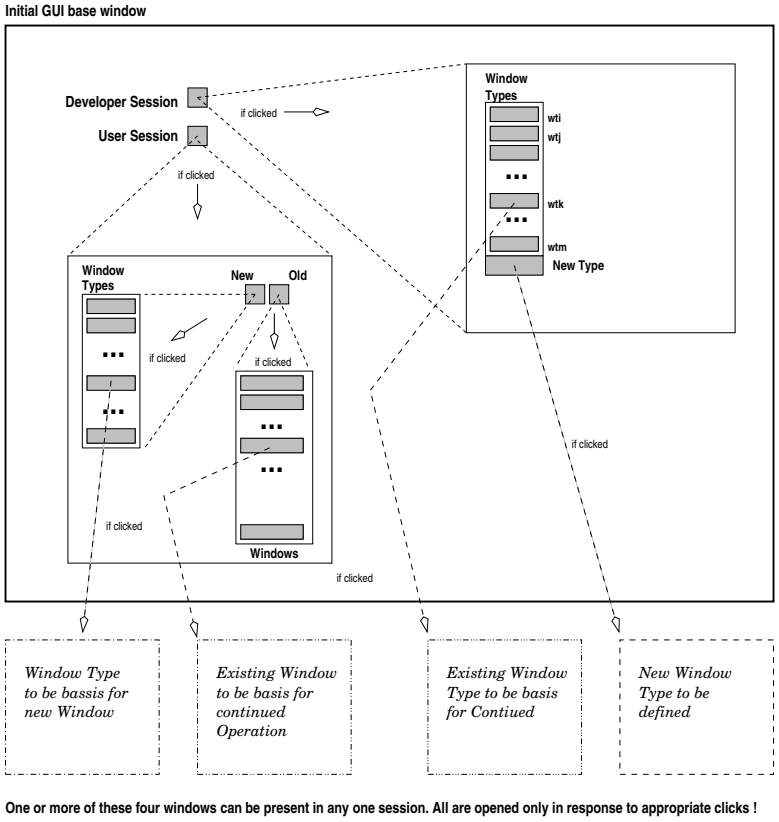
- We redefine window type expressions:
  - Instead of "recursing" fixed window type expressions in eneral window type expressions, we "recurse" via names of defined such expressions.
  - That is: Any embedded window is always of defined type.
- This allows, obviously, for arbitrary recursion in types, which we accept, but not in values:
  - That is not the same as saying that the denoted, instantiated values "recurse" — in which case we would have infinite loops. We shall rule that out.

– Every "recursed" (ie. recursively type defined) window value corresponds to an own window value (ie. node in a possibly infinitely expandable tree).
– The user is thus obliged to name such windows — whereby they become data based values.
– But such a value is not allowed to contain, somewheres, its own window value name.

The well–formedness function (inv_GUI) is to be defined.

Figure 3 shows an initial GUI: One from which either developers or users can "reach" their goal: Accessing and editing window type definitions, respectively windows.

Figure 3: An Initial Graphical User Interface

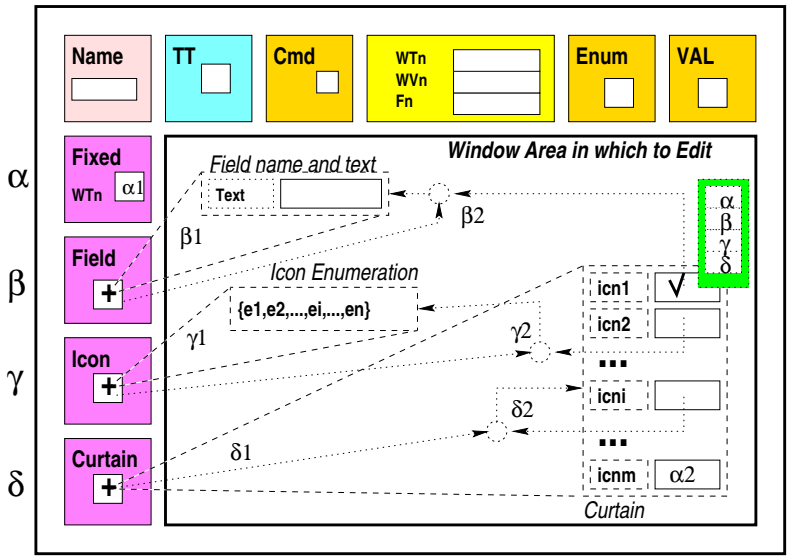


4.2 Operations on Windows

4.2.1 Informal Explication of Editor Window Functions

Figures 4 on the next page and 5 on page 11 illustrate how the editing commands can be composed using icons and curtains.

Figure 4: Facets of a possible Window Type Expression Editor



A Window Type Editor Window: Window Item Editing

α1,β1,γ1,δ1,α2,β2,γ2,δ2: **Eight window item editing cases**

In Figure 4 clicking $(\alpha_1, \beta_1, \gamma_1, \delta_1)$ one of the left column icons gives rise to the creation and display (ie. appearance) of a window item value "template" of a kind corresponding to the window item type being clicked. This template can be "dragged" into a position, with a width and height, etc., as defined below.

- α: Clicking the fixed window item icon, α_1, (illustratively, "outside the full window", labelled[1] α) means that the editor has to give the type name, wtn:WTn, of a supposedly already defined window type expression.
- β: Clicking the field window item icon[2] leads, β_1, to the *field name and text* frame appearing somewhere in the editing area. The editor is supposed to fill in some appropriate field name, see explanation below [having clicked the appropriate upper row Fn icon], and give type to the field text to be inserted by future window users [having clicked the appropriate upper row VAL icon and one of its fixed names: integer, text, symbol, Boolean]. The editor, by moving ("dragging") the *field name and text* frame can (or may) determine position and size parameters.
- γ: Clicking the icon window item icon[3] leads, γ_1, to an icon frame appearing somewhere in the editing area.

[1] The possibly indexed $\alpha, \beta, \gamma, \delta$ labels are not part of the GUI. They are used by us in explaining, in referring to, the four window item concepts.
[2] Illustratively, "outside the full window", labelled β
[3] Illustratively, "outside the full window", labelled γ

- δ: Clicking the field window item icon[4] leads, δ_1, to a curtain frame appearing somewhere in the editing area.

  One of four possibilities now arise:

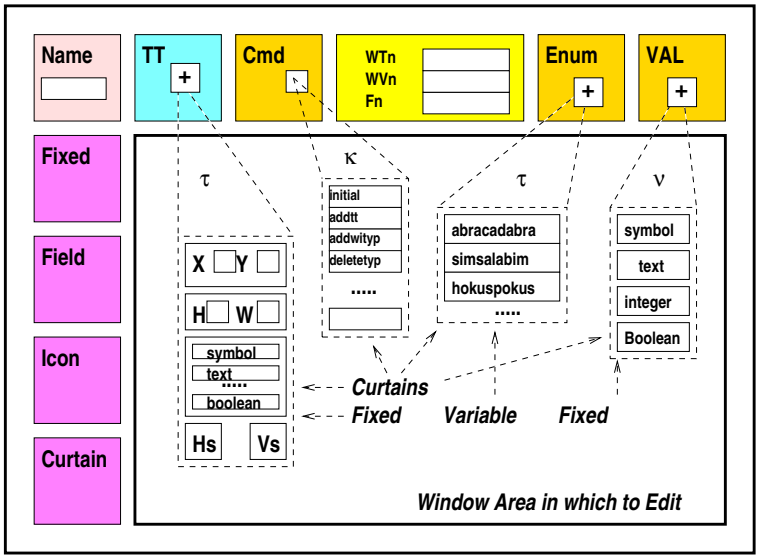
  - α_2 : Recursively a new, fixed window item, is selected, and "the story" is as above.
  - β_2 : Recursively a new, field window item, is selected, and "the story" is as above.
  - γ_2 : Recursively a new, icon window item, is selected, and "the story" is as above.
  - δ_2 : Recursively a new, curtain window item, is selected, and "the story" is as above.

WE have, informally, explained the syntax, semantics and some of the pragmatics of the window type editing facilities of the proposed GUI.

We not that the homomorphism principle transends: That recursively defined concepts are explained recursively.

Clicking window item icons is not enough. Some "minor", to some tedious, information, must be provided. We now turn to that issue. Figure 5 introduces this part.

Figure 5: Facets of a possible Window Type Expression Editor



There are four facilities for providing "administrative" information: θ, κ, τ and ν :

[4] Illustratively, "outside the full window", labelled δ

- θ: In Figure 5 on the page before clicking (θ) the TT icon, in the top row, number 2 from left, allows the user to define, for a window item template either already in the edit window, or the next to be "dropped" there, the position, height, width, annotation type and whether scrollable horisontally and/or vertically.

- κ: Clicking (κ) the Cmd icon, in the top row, number 3 from left, results in a fixed curtain being shown. That curtain lists all available commands. Clicking one of these results in the appropriate command being executed with parameters "fetched", somehow — to be explained below — from the edited window area and from the α and the top rown fields and the other curtains their clicking gave rise to. Clicking a command name closes the curtain.

- τ: Clicking (τ) the Enumeration icon, in the top row, number 2 from right, results in a variable curtain being shown. It consists, initially, of one blank field, or, depending on the history of previous Enumeration icon clickings of zero, one or more filled-in fields. If a next, ie. a first field is filled-out, by the editor, the filled-in text becomes an enumerated item and another blank field appears "below", etc. Clicking the Enumeration icon once temporarily "hides" the curtain. Clicking the Enumeration icon twice blanks it out and closes it. Clicking the Enumeration icon thrice corresponds to first a twice click, then a once click. At any one time the window are being edited thus has an enumeration set value — which may be empty. Duplicate enumeration values are ignored.

- ν: Clicking (ν) the value (type) icon, in the top row, number 1 from right, results in a fixed curtain being shown. It has four icons labelled symbol, text, integer and Boolean. At most one of these can be selected. The state of the window area now has that type name as its type name value.

4.2.2 The Main Interpreter Function

The editor function is a process. It non-deterministically [internal choice] "cycles" between selecting one or another left column, to row or editing area icon or filling out respective fields, etc. I'll probably take 1/2 page to describe that function. The internal choic non-deterministic nature reflects a widest possible choice of dialogues. One may wish to restrain these.

value
  editor: GUI $\xrightarrow{\sim}$ GUI
  editor((ρ,δ),(sess,s)) ≡
    let ... **in** ... **end**
    pre s ∈ **dom** sess

4.2.3 Auxiliary Functions

At the moment it is not quite clear whether the below attempts is on the right track or not. But it seemed a good idea to write these selector functions down. Also I am not quite sure whethere the idea of modelling ewv, wim, row, etc., is right. I shall think that over a bit more.

We define a number of auxiliary functions that extract from such a most recent editor window appropriate parts that go into composing editing commands.

value
  ewv: (,wi:wim), **type** WV
  wim: (r:row,c:col)
  row:
    [tt ↦ ((x,y),(h,w),txttyp,(hs,vs)):TT,
     cmd ↦ command:Cmd,
     wtn ↦ n:WTn,
     wvn ↦ n:WVn,
     fn ↦ f:Fn,
     en ↦ {e1,e2,...,em}:Enum-**set**,
     val ↦ v:VAL]
  col:(wi:IEnum)
type
  IEnum == fixed|field|icon|curtain

value
  sWV: GUI → WV
  sWV((ρ,δ),(sess,s)) ≡
    let (wv,wl) = sess(s) **in** wv **end**
    pre s ∈ **dom** sess ∧ wl=⟨⟩
    post wv ″`is of structure`″ ewv ″`as above`″

  sTT: WV ~→ TT, sTT(wv) ≡ tt(r(wi(wv)))
    post wfTT(tt(r(wi(wv))))

  sCmd: WV ~→ Cmd, sCmd(wv) ≡ cmd(r(wi(wv)))
    post cmd(r(wi(wv))) ≠ nil

  sWTn: WV ~→ WTn, sWn(wv) ≡ wtn(r(wi(wv)))
    post wtn(r(wi(wv))) ≠ nil

  sWVn: WV ~→ WVn, sWVn(wv) ≡ wvn(r(wi(wv)))
    post wvn(r(wi(wv))) ≠ nil

  sFn: WV ~→ Fn, sFn(wv) ≡ fn(r(wi(wv)))
    post fn(r(wi(wv))) ≠ nil

  sEn: WV ~→ Enum-**set**, sEn(wv) ≡ en(r(wi(wv)))
    post en(r(wi(wv))) ≠ {}

  sVAL: WV ~→ VAL, sVAL(wv) ≡ val(r(wi(wv)))
    post val(r(wi(wv))) ≠ nil

4.2.4 Editing Command Formalisation

This section now is to define all the functions that were appealed to, referenced, in the editor function Section 4.2.2 on page 12.

We now define the editing commands formally.

- *Initial Window Type:*

  type
  Cmd = initial | ...

  value
  I_Cmd: Cmd → GUI → GUI
  I_Cmd(initial)((ρ,δ),σ) ≡
  let n = sWn((ρ,δ),σ) **in**
  ((ρ ∪ [n ↦ ({},[ ])],δ),σ) **end**
  pre n ∉ **dom** ρ

  Initialising a window type means entering a pair of a void set and a voild map into the type catalogue.

- *Adding Type Clauses:*

  One can add either TT elements or named window items.

  type
  Cmd == | |

- *Deleting Type Clauses:*

- More to come, much more, 1 page or so !

4.2.5 Redefining Windows: WT

- *Fetching Window Type:*
- *Updating Type Clauses:*
  - *Adding Type Clauses:*
  - *Deleting Type Clauses:*
- *Catalogueing Window Types:*
- More to come, much more, 1 page or so !

4.2.6 Window Session

- *Opening Window:*
- *Operating upon Windows:*
  - Single Clicking Icons:
  - Scrolling Curtains:
  - Double Clicking Icons:
  - Filling–in Fields:
- *Closing Window:*
- More to come, much more, 1 page or so !

5 Bibliographical Notes

References

[1] D. Beech, editor. *Concepts in User Interfaces: A (VDM) Reference Model for Command and Response Languages*, volume 234 of *Lecture Notes in Computer Science*. Springer-Verlag, 1986.

[2] L.S. Marshall. *A Formal Description Method for User Interfaces*. PhD thesis, University of Manchester, Oct. 1986.

[3] Bernard A. Sufrin. Formal methods and the design of effective user interfaces. In M.D. Harrison and A.F. Monk, editors, *People and Computers: Designing for Usability*. Cambridge University Press, UK, 1986.

[4] C.B. Jones and R. Moore. Muffin: A user interface design experiment for a theorem proving assistant. In *[9]*, pages 337–375, September 1988.

[5] VIP Project Team. Man machine interface: Final specification. Report VIP.T.E.8.3, VIP, Praxis Systems, Bath, England, December 1988.

[6] Andrew W. Wood. A Z specification of the MaCHO interface editor. Memorandum no. 4247, RSRE, Ministry of Defence, Malvern, Worcestershire, UK, November 1988.

[7] Gregory D. Abowd, Jonathan P. Bowen, Alan Dix, Michael Harrison, and Roger Took. User interface languages: a survey of existing methods. Technical Report PRG-TR-5-89, Oxford University Computing Laboratory, 11 Keble Road, Oxford, UK, October 1989.

[8] Daniel Jackson. Structuring Z Specifications with Views. *ACM Transactions on Software Engineering and Methodology*, 4(4):365–389, October 1995.

[9] R. Bloomfield, L. Marshall, and R. Jones, editors. *VDM – The Way Ahead*. Proc. 2nd VDM-Europe Symposium 1988, Dublin, Ireland, Springer-Verlag, Lecture Notes in Computer Science, Vol. 328, September 1988.