On the visualization of design notions, of notion instantiations, and of structural relationships in a design data base realized as a semantic net

by

Hans-Juergen Hoffmann

Technische Hochschule Darmstadt / Technical University at Darmstadt
Fachbereich Informatik / Department of Computer Science

Abstract

A new generation of integrated design environments is under the way in developing laboratories. They employ a network structure of design notions and of notion instantiations as a design data base to collect designers' decisions with realization in form of a semantic net/frame net. We consider an integrated environment for design of interactive programs as a particular case where the principle of specification by example is applied. Problems with the visualization of design notions relevant in this area, of notion instantiations, and of structural relationships between them are identified from the view point of computer science. Cognitive psychologists are asked for advice on how to proceed to solve the problems. Finally, object-oriented visualization is investigated in a prospective view.

Contents

Abbreviations

DDB	design data base	DS/P	design specification / program
IDE	integrated design environment	SN/FN	semantic net / frame net

" ... the challenge facing computer language designers today is to provide convenient and natural **visual** programming languages ... "
R. J. K. Jacob [21]

1. Why visualization? Visualization of what?

Design decisions of a designer/programmer*1) manifest themselves in

(i) a textual structure if he uses a traditional programming (or specification) language,
(ii) an hierarchic, mostly tree-like structure of nodes representing program constructs if he works with a tree editor, or
(iii) a network structure of "design notions" and notion instantiations ("design objects") if he manipulates a modern interface to an integrated design environment (IDE for short) where the objects are interrelated by a wide variety of structural relationships ("links").

*1) Programming is a specialized design task. When considering interactive programs, especially its interaction properties, I definitely prefer the term "designer" although the person performing the task may in other circumstances, work as a programmer (see also [33]).

The design decisions constitute what is called the design specification/program (DS/P). We have to consider not only the final specification/complete program but also intermediate forms. The DS/P, may it be intermediate or final/complete, is maintained in what we call the design data base (DDB). Appropriate realizations of a DDB in the three situations identified above provide

(i) sequential storage structures eventually broken in records (character data, linear lists, sequential files),
(ii) tree structures, and
(iii) semantic nets/frame nets (SN/FN, see, e. g., [5], [26] or, applied to design tasks, [24] and [23]), respectively.

At least the working situations (ii) and (iii) are interactive situations in the very nature. Editing*1) is involved. To support the designer in making decisions in such a way that, as long as he does not drop or modify a goal previously set, the decisions are consistent it is common practice to enforce (some of the) structural relationships ("structure oriented editing") and, in order to not overcharge the designer, to visualize (part of) the DS/P, intermediate as it is, during the course of each interactive decision step on a display unit (video screen). Some kind of pretty-printing/pretty-visualizing*2) is common practice. However, we have to be aware of the fact that a designer -- at least a professional one -- does not only want to see an echo-like representation in prettified textual form of what he has just input by some kind of typing/manipulating; rather, he additionally wants to see something like cross-reference information, storage maps, data flow/control flow information, accumulated timing information etc, all what a good compiler provides in a compilation protocol*3) (see, e. g. [20]).

The subject of the paper is not to discuss structure-oriented editing; nor what part of the DS/P is visualized; how the focus of visualization is determined and how its position is moved; how the designer controls separation in parts and focus movement; nor how the editor/IDE provides automatic control in a competent manner. Subject of the paper is how that parts of the DS/P that have

*1) See e. g. [16] and [17].
*2) Pretty-printing and pretty-visualizing, its counterpart in an interactive situation, allow to visualize one of the (hierarchic) structural relationships (contained in-relation).
*3) As an example, we consider the task of programming.

been selected for visualization are actually visualized, are actually visualized in a most supportive manner during a design session.

In text editing (situation (i) above) three categories of editing are identified (cf [32]): indirect editing, quasi-direct editing, and direct editing. Indirect editing requires to formulate editing commands without recourse to a visualized text. Direct editing means physically decomposition/composition of a textual structure, e. g., on a sheet of paper by scissors and glue. What can be realized by making use of a data processing system is quasi-direct editing, i. e., substituted editing in the visualized text instead of (direct) editing in the textual structure itself.

The criterion of visualization "in a most supportive manner" mentioned above, now, can be understood as a best approximation to direct editing (that itself cannot be achieved in a data processing system).

We want to extend these approaches from text editing to doing design work/programming by manipulating a modern interface to an IDE (situation (ii) and, situation (iii) above). Again it is not possible to access design notions, design objects, and links directly (even not if we wanted to do it physically as they are of an abstract nature). Indirect editing*1), besides utilizing commands, would require an understanding of the kind of physical representation*2) employed in the DDB for the abstract notions, design objects, and links. What is possible -- and what has to be developed -- is a substituted, quasi-direct manipulation [37] of what is visualized; thereby we assume that the kind of visualization provided allows the designer the correct association to the (abstract) design notions, design objects, and links, resp., that, idealistically, had to be manipulated directly. We understand that the correct association is facilitated if the design notions used for instantiation of the design objects and the basis of the links interrelating objects are adequately covered in what cognitive psychologists call the mental model of the design task. The search

*1) In case of indirection, obviously, one cannot speak of manipulation.
*2) The physical representation is much more complex as, to continue the analogy, the physical representation of a text in form of a sequential storage structure.

for adequate coverage is outside the scope of our work; cognitive psychologists should be concerned with that question.

There, again, has to be a best approximation to direct manipulation. This requirement leads us to search for good, best kinds of visualization of all design notions, design objects, and links that are relevant to the designer.

The developer of an IDE, has to meta-decide*1) on what design notions, what design objects, and what links have to be representable in the DDB*2) of the IDE and how they are represented if instantiated. However, he is overcharged if kinds of visualizations to the designer are the subject of meta-decisions; considering his understanding of best kinds of visualization there are problems that can only be solved in a satisfactory manner by cognitive psychologists or with good advice from cognitive psychologists.

To be able to put concise questions to cognitive psychologists we consider two kinds of IDEs. In section 3 we consider, mainly for introductory purposes, the general programming task (where some IDEs can be -- and will be -- referenced that provide some kind of visualization of programming notions, of their instantiations as program constructs as well as of structural relationships between program constructs -- especially the relationship that commonly is called the program structure*3) --). In section 4, the central section of the paper at hand, we want to identify design notions, design objects and links that are relevant in developing an IDE*4) for the user-machine interface of interactive programs; here, the situation (iii) mentioned above will be exemplified in a characterizing manner.

The importance of visualization facilities and the scope of available techniques are discussed (among others) in [12], [4], and [9].

*1) To distinguish the levels of the developer of an IDE and of the designer as a user of the IDE, we will address the former case as the meta-level.
*2) In section 2, as a supplement, we will discuss a favourable architecture of a DDB in form of a SN/FN.
*3) With respect to editing, there is usually no strict separation conceptualized between the dynamic program control structure proper and the static program structure.
*4) The DIADES system, see [18] and [19].

2. Semantic nets/frame nets as design data bases, a supplementary note

IDEs serve not only the purpose of collecting and editing design decisions related to design notions directly characterizing what has to be designed. There are further design notions that have to be considered. They originate from requirements of components like

- a design simulator (component for experimentation with the designed system, prototyping),
- a test controller (test data handling),
- a design verifyer,
- a design evaluator,
- different design editors in a phase-modelled approach (requirements/specification editor, program editor, design documentation editor, performance statistics editor), etc.

And they originate from a wider understanding what has to be covered during design. It is the intended scope of design notions to be covered and the intended variety and multiplicity of links to be established and maintained between design notions and design objects instantiated from the notions, resp. that determine the required capabilities of the DDB.

If one consideres the collection of design objects instantiated in the DDB to represent the knowledge about the system to be designed which the IDE has become acquainted with in a design session we have to look for a most powerful architecture of knowledge representation. Other, simpler architectures are an inadmissible compromise that runs the risk of becoming a limiting factor in the capabilities offered.

We advocate the utilisation of SNs/FNs according to BRACHMAN's proposal (as realized in KL-ONE, see [5]) with some modifications as sketched below. Without going into details we want to remind the reader of the set of epistemological links[*1) *2)] introduced by BRACHMAN. We characterize the links by their signature:

- DATTR: concept_node -> role_description_node
- ROLE: role_description_node -> SET_OF_INSTANTIATIONS_OF (Functional_roles)

  note: This is the most important link type that allows to introduce

*1) These are representational links of the data base that should not be mistaken for the application dependent links addressed in the rest of the paper.

*2) We will not list the link types used for concept creation.

the different "roles" of a concept[*1], roles that may be used to attach values (see below) and/or to establish the application dependent links between design notions and their instantiations as design objects, resp.

- V/R: role_description_node -> concept_node
  (V/R stands for "value restriction".)
- NUMBER: role_description_node -> PREDICATE
- MODALITY: role_description_node ->
  SET_OF_INSTANTIATIONS_OF (System_roles)
- DEFAULT: role_description_node -> V/R (self))
- INSTANCE_OF: instance_node -> concept_node
- ATTR: instance_node -> role_instance_node
- INSTANTIATES: role_instance_node -> role_description_node
- VALUE: role_instance_node -> value_node
- S/C: concept_node ->
  union (PREDICATE, SET_OF_INSTANTIATIONS_OF(Concept))
  (S/C stands for "structural condition")
- E/R: concept_node -> FORMULA
  (E/R stands for "evaluation rule"[*2])

Note that the S/C- and E/R-links are not instantiated (i. e., there is no counterpart that serves as a link between design objects).

In section 5 we will add another link type, also not to be instantiated (it is just the addition of this link type "VISUALIZE_AS" which constitutes a main contribution of the paper at hand).

Now, it is possible to represent design objects (= instance_nodes) as complex as they are conceptualized in the design notions (= concept_nodes); (application dependent) links are conceptualized in the role descriptions attached to a concept and are instantiated in the role instantiations. It is easy to form "views" of design notions/design objects by considering only some of the roles. There is no limitation to only a tree-like structuring under the design objects, as different (structuring) roles are available.

Note that the usual grammatical specification of design notions (modelling the behaviour of structural editors with a hierarchic, tree-like DDB) is a (rather simple) specialization of conceptualization by SNs/FNs:

> A grammatical production rule X -> a b ... z with X a nonterminal symbol and a, b, ... z terminal or nonterminal symbols is a conceptualization of a concept X stating that all components of any instance of X (as an ordered tuple) have to be instantiations of the concepts a, b, ... z. The usual syntactic derivation tree (parse tree) is a shorthand notation of a (tree-like) semantic net. a, b, ... z stand for roles in all the X. The S/C- and E/R-links can be understood to be context conditions and attribute evaluation

*1) Roughly, roles correspond to slots as identified in frames [26].

*2) The E/R-link is not identified in [5]. We think that, at least in the application to DDBs that have to be able to represent intermediate DSs/Ps such a link type is useful and is required.

rules, resp., as usually associated to productions in attributed grammars. Concept instantiation corresponds to syntactical derivation; rejection of an instantiation if a S/C-link is not satisfied and attribute evaluation according to an E/R-link both are handled by what is called semantic analysis in compiler design (or in design of incremental structural editors).

3. Visualization in a structural editor

For introductory purposes, as stated earlier, we consider here the general programming task.

What design notions have to be covered?

- A program is conceived as consisting of declarations of data and of imperative commands ("statements", "expressions") to access and to modify data.
- Considering the data there are design notions like declaration structuring (in array and record declarations, in structures build up by pointer chains), like scope of identifier visibility, like range of data existence, like source of a data value and accompanying sinks to which a value flows, etc.
- Considering the statements there are design notions like statement structuring (conditional, iterative, collateral), like remote specification of statements (procedure and modul references), like embedding of exception handling, like flow of control, etc.
- Both areas are intermixed in design notions like control of access rights in scopes and/or ranges, respectively, like data flow structure, etc.

Notion instantiations, i. e., design objects, are what is commonly called program constructs like

- the "integer data identified by identifier such and such in scope such and such the value of which is set or accessed, resp., in the statement such and such" or like
- the "assignment statement being successor of statement such and such and being predecessor of statement such and such all contained in the (composite) statement such and such assigning a value determined by evaluating the expression such and such to the variable such and such",

where all the terms "such and such" have to be understood as structural relationships ("links" as we call them).

We hope that the examples given clarify what we think of.

There are some classes of structural editors. They may be distinguished by the kind of visualization that they utilize:

- The class of structural editors (like GANDALF [13], CPS [42], DUAL/KEYONE

[34], PAD [15], [27], [43], to reference only some) that use a textual representation similar to what had to be written down if the program would be formulated on a sheet of paper according to the rules of a programming language. The selection menues that they usually visualize can be considered as visualizations of (some specific) design notions and of options for their instantiation. Design objects are, without exceptions, visualized textually. Only some links are visualized:

- o The hierarchic contained in-relation building a (static) program tree, in fact by indentation;
- o There are placeholders visualized in case that a contained in-link is not yet instantiated (referring to the "nonterminal" notion of a grammatical specification of the programming language).
- o Also the successor/predecessor-relation expressing (sequential) flow of control.
- o As in programs formulated on a sheet of paper the access-relation to data is only implicitly visualized by the use of identifiers invented by the programmer for that purpose.
- o All other links are not visualized, neither explicitly nor implicitly.

- The class of structural editors (like XS-i, i $\in$ { 0, 1, 2 }, [7], [1], [40]; like LITOR [44] and like others) that display the contained in-relation by a tree with some shaped boxes as visualizations of design objects and short lines as visualizations of the contained in-relation.
- The class of structural editors (like [10], COLUMBUS [11] and others) that display the contained in-relation in structogram-style*1).
- A class of structural editors with miscellaneous facilities for visualization (like, e. g., GRADE [36]).

Are the visualization facilities exemplified by these structural editors through meta-design decisions of developers best kinds of visualization? Is it sufficient to provide only facilities for visualization of some design notions (if any), of some links between design objects? By what visualization facilities can indirect editing of links (as, e. g., the access-relation through naming with identifiers) be replaced by quasi-direct editing?

4. Visualization in an integrated design environment

In the following we will concentrate on notions for designing the user-machine interface of interactive programs and on notions originating in the require-

*1) The structogram-style can also be understood as a tree visualization: Visualization of a "Spalierbaum" (wall tree) looking upon it from the sky (or from the ground whatever viewpoint one prefers).

ments of a design evaluator inspecting and evaluating designers' decisions that are relevant for the user-machine interface. From an example we derive that a SN-/FN-architecture is required for the DDB; a hierarchic, tree-like structure is not sufficient to represent and evaluate design decisions. The papers [18] and [19] on the DIADES project serve as a reference to the design tool that we have in mind.

To design an interactive program design notions like

- communicated message, symbolization in communicated messages, interaction point, weak interaction, strong interaction*1), sequence of interactions, causality of interaction, goal of interaction, timing of interactions, state explanation, explanation of interaction history, explanation of possibilities for interaction*2), attributes of a field in a communicated message, awareness level to be assigned to a field in a communicated message, type of confirmation to a communicated message requested, etc

have to be considered. In the sequel, we will use the design notion "menu" as an example (see fig. 4.1). We use the epistemological links introduced (following BRACHMAN) in section 2*3). It would be easy to continue the listing. At the moment it seems not possible to state criteria for terminating the list nor to categorize the design notions in the list.

Examples of notions originating in the requirements of a design evaluator are

- task category, application classification, user category, variance in user's expertice, variance in user's concentration, variance in user's motivation, easiness of requested manipulation, load to user's temporary memory, etc.

In the "menu"-example we have to provide (among others) design notions like

- Is a set of actions associated to menu entries constrained or not?
- Number of menu entries.
- What amount of arbitrary data is involved?
- Response time expected.
- Is a menu task-specific?
- Can a menu be bypassed by command entry for experienced user?
- Consistency in wording between options and commands.
- Number of selections in a menu?
- Reconfirmation of selection before action is taken.
- Acknowledgement that action has been initiated.
- Frequency of menu selection (compared to other means of control).
- Frequency of selection of a specific menu entry.
- Consistency in wording for designating options in menue sequences.
- Consistency in placement of options repeated in menu sequences.
- Designating menu entries in main menu/supplementing menu/general context.
- Indication of current menu in a menu sequence.

*1) See [22].
*2) See NIEVERGELT [28] for the importance of the three notions just mentioned.
*3) BRACHMAN, in his article [5], employed a pictorial visualization of concepts, of concept instantiations, and of links. This is a visualization on the meta-level. We will employ a more formalized notation.

```
CONCEPT Menu WITH
   ROLES #_of_entries: V/R = {n|n>1}, NR = 1, required, DFLT = 2;
         #_of_hidden_entries: V/R = {m|m>0}, NR = 1, optional;
         markings_for_selection: V/R = Marker_combination,
                  NR = S/C#1 ( VALUE ( #_of_entries ) ), required,
                  DFLT = {(x,-), (-,x)};
         accessed_from: V/R = Path_in_decision_network, NR ≥ 1, required,
                  DFLT =
                    Empty_path AS_AN_INSTANTIATION_OF Path_in_decision_network;
         level: V/R = Length_of_path, NR = 1, (required, derived), DFLT = 0;
         menu_identification: V/R = TEXT, NR = 1, (optional, defaulted),
                  DFLT = 'Main menu';
         explanation: V/R = TEXT, NR = 1, optional;
         prompt: V/R = TEXT, NR = 1, (optional, defaulted),
                  DFLT = 'Select one item';
         task: V/R = Tasks, NR ≥ 1, optional;
         bypass: V/R = Commands, NR ≥ 1, optional;
         #_of_defaulted_entries: V/R =
                   S/C#2 ( VALUE (#_of_entries), VALUE (#_of_hidden_entries) ),
                  NR = 1, (optional, defaulted), DFLT = 0;
         identification_of_defaulted_entry:
                  V/R = Pair_entry_identification_and_entry_marker,
                  NR = S/C#3 ( VALUE ( #_of_defaulted_entries ) ), required;
         reconfirmation: V/R = BOOLEAN, NR = 1, (optional, defaulted),
                  DFLT = true;
         entries: V/R = Menu_entry, NR =
                   S/C#4 ( VALUE (#_of_entries), VALUE (#_of_hidden_entries) ),
                  (required, ordered), DFLT#1 =
                    Uninitialized_menu_entry AS_AN_INSTANTIATION_OF Menu_entry,
                  DFLT#2 =
                    Uninitialized_menu_entry AS_AN_INSTANTIATION_OF Menu_entry;
   S/C#1: ( Lambda x: {p|p>x} );
   S/C#2: ( Lambda x, y: {p|0≤p≤x+y} );
   S/C#3: ( Lambda x: {p|p=x} );
   S/C#4: ( Lambda x, y: {p|p=x+y} );
   E/R#1: VALUE (level) := ... ;
   VISUALIZE_AS: ... .
CONCEPT Marker_combination WITH ... .
CONCEPT Path_in_decision_network WITH ... .
CONCEPT Length_of_path WITH ... .
CONCEPT Tasks WITH ... .
CONCEPT Commands WITH ... .
CONCEPT Pair_entry_identification_and_entry_marker WITH ... .
CONCEPT Menu_entry WITH ... .
```

fig. 4.1
(part of) a semantic net/ frame net for the design notion "menu"

- Option to return/undo a selection or to restart in a menu sequence.
- How are menu entries and their designations distinguished from other displayed messages?
- Grouping under menu entries if they are related.

that are relevant for design evaluation (cf [38]; only some of them are covered in fig. 4.1). Again, it would be easy to continue the listing. And -- at least under the view point of a computer scientist -- we seem to be even more away from the terminating criteria for the list.

To try to list links between design objects that would be relevant here seems

not to help much to clarify the situation. It is an open research challenge to do this*1).

In present practice many interactive programs are designed without any explicit consideration of most of the notions mentioned*2). The instantiations of the notions and their links are buried in program text formulated in a traditional language (and hence, of course, are not or only with difficulty accessible to an inspection and evaluation process, even not by a human expert, except when the interactive program is run and the messages appearing on the interface are observed after completed implementation). During design there is no visualization of relevant notions possible.

Template generators (like [6], [29], [30], [41]) partially cover some concepts and links, especially those related to the layout of communicated messages. However, the DDBs used there do not follow an architecture that by any means allows to integrate a generator with other components, e. g., for design evaluation.

As a methodological approach to overcome the disappointing situation it has been repeatedly proposed to utilize an interaction-dominant structure of interactive programs (cf, e. g., [8]). It is this approach that puts a DDB in its proper place as a central, integrating facility accessed by all components of an IDE. As an example, we show in fig. 4.2 a rough architectural diagram of DIADES as a design tool and in fig. 4.3 of the DIADES run-time system.

However, also for this approach, there are no facilities invented for visualiation of the design notions involved, of design objects, of links. A specification editor would look like an editor for textual design documentation; indirect editing is the only category available.

This situation has to be changed, has to be improved. The developer of an IDE has to invent visualizations of all the design notions involved, of design objects, of links (better: he gets hints on how to do it from cognitive psychologists!).

*1) In the scope of the DIADES project we are investigating the topic; however we do not expect to be able to present a list with proven relevance in the near future.
*2) And of further notions not mentioned.

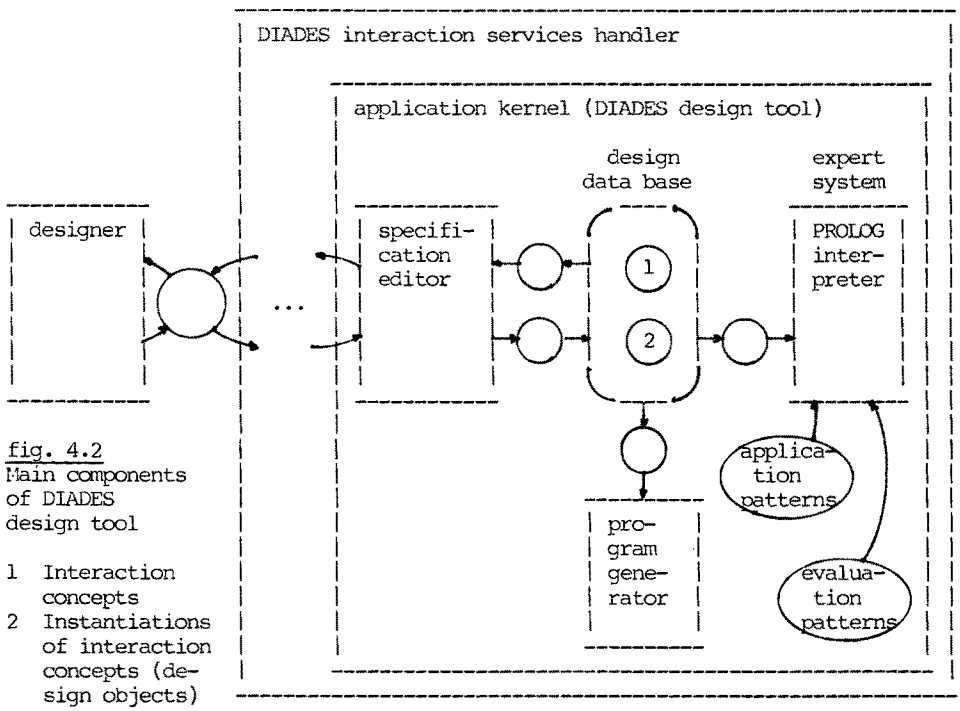

fig. 4.2
Main components of DIADES design tool

1 Interaction concepts
2 Instantiations of interaction concepts (design objects)

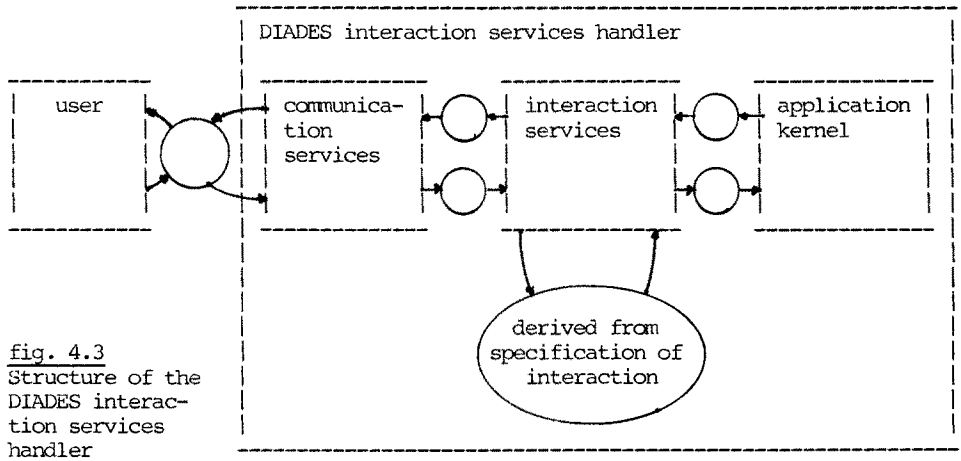

fig. 4.3
Structure of the DIADES interaction services handler

An IDE for interactive programs (like DIADES) suggests another argument: The very nature of interactive programs is to interact with a user. When the designer decides on how the interactions should take place it is only near at hand that he wants to interact in the same way, i. e., that he wants to make design decisions by interaction samples. Thus, the kind of visualization faci-

lities to be invented for design notions etc have to look similar to actual interactions of the interactive program to be generated for later use from the DS/P. We call this the specification by example-principle which has a lot of implications for how relevant design notions, design objects, and links are visualized to the designer*1).

It is one purpose of the paper at hand to make cognitive psychologists aware of the lack of appropriate facilities for visualization in this area.

The developer of an IDE, when considering the architecture of the DDB, has to consider and has to realize the implication identified in the next sentences: There has to be another epistemological link in the DDB that allows to describe the visualization facilities to be utilized during design*2). It is a link that originates in concept nodes; it is (like S/C- and E/R-links) also not instantiated in the design objects:

VISUALIZE_AS: concept_node --> SET_OF_INSTANTIATIONS_OF (Design_dialogues)

We will come back to this topic in the next section.

5. Object-oriented visualization, a prospective view

A DDB organized as a SN/FN suggests to think of an object-oriented treatment of design notions, notion instantiations and links*3). Object-orientation occurs

*1) JACOB distinguishes in [21] two categories, viz., (i) the first, where there is a one-to-one correspondence between a visual representation of the design object during design and the visual object being specified, resp., (e. g., a screen template) and (ii) the second, where something "abstract" has to be specified (like a timing scheme). Although we also are aware of the difficulties in visualization of abstract notions we want to cover both categories under the specification by example-principle (see section 5) in an approach that gives more flexibility ("design dialogue"); modern manipulative interfaces allow more than only simple visual communication, e. g., pushing a key for a time period exemplifying some design decisions on a notion related to time, or, relying to a guided design dialogue with some fading-over capabilities, exemplifying an hierarchic notion.

*2) Under a more general viewpoint it may be preferable to name the link by another word, say "DESIGNED_BY". It is only at the time being that visualization plays such an overwhelming role and that "VISUALIZE_AS" is adequate; the situation may change.

*3) In this section of the paper at hand we will speak of "objects" as a general term not restricted to notion instantiations. In a strict sense it is a question of "meta-objects".

on the meta-level. The design notions "know" how they are interrelated, how they can instantiate themselves, and how the instantiations (the design objects[*1]) are to be linked. The suggestion, of course, extends to the object visualization, i. e.,

(i) to the object visualization during the design process, also
(ii) to the visualization of the design process for the design notions considered.

A design notion (held in the SN/FN as an object) states the kind of visualization of itself and of its instantiations as well as of the process (and its visualization) to instantiate it[*2]. The VISUALIZE_AS-link introduced in section 4 serves that purpose.

It is rather easy to include the VISUALIZE_AS-links in the DDB of an IDE for interactive programs: It is possible to link to an instantiation of the concept "Dialogue" that, as a superimposed concept, always is available[*3]. Thereby, the visualization of the design process ((ii) above) is covered in form of a "Design dialogue"[*4]; the design dialogue includes appropriate visualizations of the design notions, its instantiations, and its links ((i) above). This approach will be followed in the development of DIADES.

A consequence of object-oriented visualization to the development of IDEs in general is that they all should provide a visualization structure for design processes in their respective DDB, too.

Now, it has to be discussed how an appropriate visualization, i. e., the required instantiation of the concept "Design dialogue", should look like. There should be a consistent treatment of all notions (e. g., by highlighting) and of any notion instantiations. If the designer sees a visualization of a specific notion instantiation (design object) he may ask for appropriate visualization

*1) The reader should be aware of the distinction between object-orientation of design notions and the instantiation of design notions (which we called -- being aware of the risk of confusion -- "design objects" for other reasons). Pardon!
*2) In [3] the same requirement is discussed.
*3) We will not pursue here the implications to "bootstrapping" an integrated design environment from an initial status with poor visualization capabilities to an improved, final status using the integrated design environment itself.
*4) i. e., a subconcept/specialization of "Dialogue".

of all instantiated links originating from/sinking in the design object considered as well as of all non-instantiated links the notion may be involved in. This should occur in a consistent manner for all design objects instantiated for the specific design notion.

An example of such a design dialogue and the kind of visualization we are thinking of in the development of DIADES is presented in fig. 5.1 - 5.8.

Iconic signs as proposed*1) or even already used*2) in other areas may be considered as candidates in the meta-decisions for the kind of visualization facilities we are looking for. A possible implementation could also be something like a pop-up menue, a technique already available in other interactive applications. More challenging (however far from being solved) is to guide simulated interactions in the design dialogue and to infer DS/P from them.

For developers of IDEs of the kind we are interested in (i. e., for designing interactive programs employing the principle of specification by example) it would be of great help if cognitive psychologists would give answers on the appropriateness of the object-oriented approach sketched above as approximation of direct manipulation of the relevant notions, notion instantiations, and links during design.

5. Literature references

[1] G. Beretta et al.: XS-1, an integrated interactive system and its kernel; Proc. 6th Intl. Conf on Software Engineering, 1982, 340 - 349.

[2] W. L. Bewley et al.: Human factors testing in the design of XEROX's 8010 'STAR' office workstation; Proc. Human Factors in Computing Systems, 1983, 72 - 77.

[3] A. Bigl, P. Szalapaj: Saying what you want with words & pictures; Proc. INTERACT '84, 1985, 275 - 280.

[4] H.-D. Boecker: Visualisierung als Problemloese- und Programmiertechnik. In H. Schauer/M. Tauber (Hrsg.): Psychologie des Programmierens, Oldenbourg-Verlag, 1983, 96 - 110.

[5] R. J. Brachman: What's in a concept, structural foundations for semantic networks; Intl. J. Man-Machine Studies 9 (1977), 127 - 152.

*1) To give some literature references: [31], [39], [35], [25], also [14] and [2].

*2) Look out for a XEROX STAR and all its "visual compatibles".

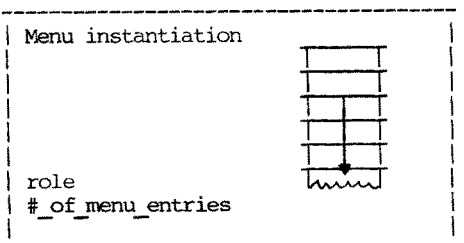

fig. 5.1 Pull-down under mouse control

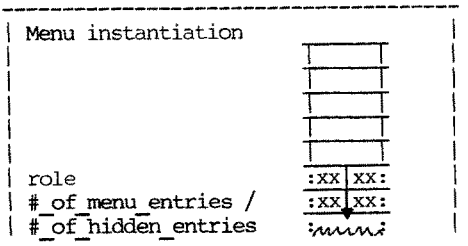

fig. 5.2 Pull-down continued, hidden entries identified

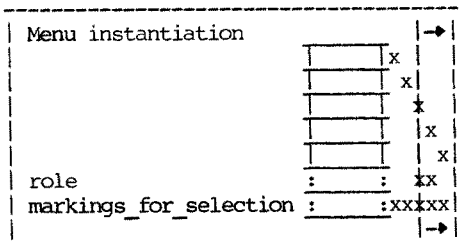

fig. 5.3 Pull-right, marker combination samples added

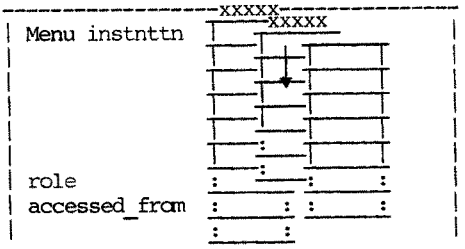

fig. 5.4 Pull-down in (one of the) predecessors

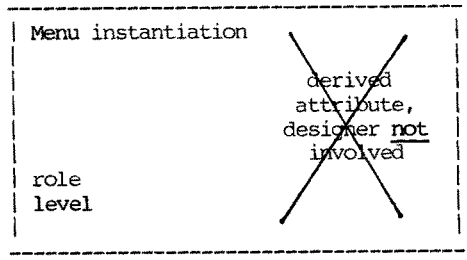

fig. 5.5 -- not included in design dialogue! --

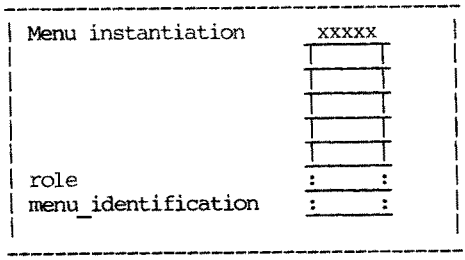

fig. 5.6 typing-in of identification for menu instantiation

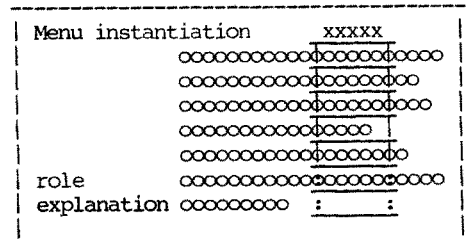

fig. 5.7 Typing-in of explanatory text (text editing facility)

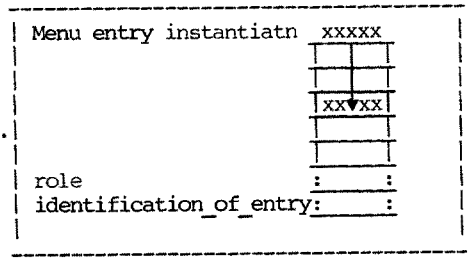

fig. 5.8 selection of entry under mouse control, typing-in of identificator of entry

xxx / ooo place where typed-in text appears

[6] R. Buchholz: GERDA, ein Generator fuer Programme mit interaktiver Schnittstelle; Tagungsband d. 15. WASCO-Jahrestagung, 1983.

[7] H. Burkhart, J. Nievergelt: Structure oriented editors. In P. R. Wossidlo (Hrsg.): Textverarbeitung und Informatik; Informatik Fachberichte Nr. 30, Springer-Verlag, 1980, 164 - 181.

[8] R. W. Ehrich: DMS, a system for defining and managing human-computer dialogues; Automatica 19 (1983), 655 - 662.

[9] M. Fitter, T. R. G. Green: When do diagrams make good computer languages?; Intl, J. Man-Machine Studies 11 (1979), 235 - 261.

[10] H. P. Frei et. al.: A graphics-based programming support system; ACM SIGGRAPH Newsletter 12 (1978) 3, 43 - 49.

[11] K. Gewald et al.: COLUMBUS, strukturierte Programmierung in der Praxis; Elektronische Rechenanlagen 19 (1977) 1, 30 - 34.

[12] P. Gorny: Zur Manipulation visueller Information. In H. Schauer/M. J. Tauber (Hrsg.): Psychologie der Computerbenutzung; R. Oldenbourg-Verlag, 1983, 55 - 88.

[13] A. N. Habermann: The GANDALF project. In H. Morgenbrod/W. Sammer (Hrsg.): Programmierumgebungen und Compiler; Teubner-Verlag, 1984, 281 - 284.

[14] K. Hemenway: Psychological issues in the use of icons in command menues; Proc. Human Factors in Computer Systems, 1982, 20 - 23.

[15] H.-J. Hoffmann: Programming by selection; Proc. Intl. Computing Symposium 1973, 1974, 59 - 65.

[16] H.-J. Hoffmann: Ueber die Benutzung moderner Editoren fuer die Programmentwicklung und Textbearbeitung. In H. Schauer/M. J. Tauber (Hrsg.): Psychologie der Computerbenutzung; R. Oldenbourg-Verlag, 1983, 101 - 148.

[17] H.-J. Hoffmann: Ueber Editoren fuer die Programmentwicklung und Textbearbeitung; Bericht PU1R6/83, FG Programmiersprachen und Uebersetzer, TH Darmstadt, 1985 (submitted for publication).

[18] H.-J. Hoffmann: DIADES, ein Entwurfssystem fuer die Mensch-Maschine-Schnittstelle interaktionsfaehiger Systeme; Notizen zu interaktiven Systemen, 1984, Heft 12, 59 - 69.

[19] H.-J. Hoffmann: DIADES, a design tool for interactive programs with an integrated expert system for design evaluation; Bericht PU1R5/85, FG Programmiersprachen und Uebersetzer, TH Darmstadt, 1985.

[20] H.-J. Hoffmann: Informierende Nebenleistungen eines Uebersetzers; Kap. F5 des Vorlesungsbegleittexts Uebersetzerentwurf II, unveroeffentlicht.

[21] R. J. K. Jacob: A visual environment for designing user interfaces: private communication, 1984.

[22] I. Kupka: Functions describing interactive programming; Proc. Intl. Computing Symposium 1973, 1974, 41 - 45.

[23] R. Lutze: Eine Programmierumgebung auf der Grundlage einer Objektflussmaschine; Doctoral dissertation, Techn. University at Darmstadt, 1985

[24] H. M. Markowitz et al.: The EAS-E application development system, principles and language summary; Comm. ACM 27 (1984) 8, 785 - 799.

[25] G. Matheis: Konzeption und Realisierung der graphischen Ausgabe von Spezifikationen; Diplomarbeit, Univ. Kaiserslautern, FB Informatik, Februar 1985.

[26] M. Minsky: A framework for representing knowledge. In P. Winston: The psychology of computer vision; McGraw-Hill, 1975, 211 - 277.

[27] D. Muth: Erstellung strukturierter Programme im Dialog; Lecture Notes, German Chapter of the ACM, 1974/3, 1 - 10.

[28] J. Nievergelt: Design of man-machine interfaces, towards the integrated interactive system; Notizen zu interaktiven Systemen, 1984, Heft 13, 3 - 12.

[29] NN: Benutzerhandbuch IFG fuer FHS; Siemens AG, Transdata, Bestell-Nr. U765-J1-Z75-3.

[30] NN (E. E. Iacobucci): Application display management system; IBM Personal Computer Software, Order-Nr. 6322129, 1984.

[31] NN: "'Ein Bild sagt mehr als viele Worte.' Und ein Pictogramm? Urteilen Sie selbst. Ihre Mitarbeit hilft uns weiter. Und Ihnen auch ..."; Triumph-Adler AG, Basisentwicklung, Fuerth, Fragebogen von der Hannover-Messe 1984.

[32] H. Oberquelle: Objektorientierte Informationsverarbeitung als Grundlage benutzergerechten Editierens; Doctoral Dissertation, Univ. Hamburg, 1979.

[33] D. R. Olsen: A context for user interface management; IEEE Computer Graphics & Applications 4 (1984), 12.

[34] L. Petrone et al.: Program development and documentation by stepwise transformations, an interactive tool; Proc. Intl. Computing Symposium 1983, 1983, 268 - 285.

[35] G. Rohr, E. Keppel: Iconic interfaces, where to use and how to construct; Proc. 1st Intl. Symp. on Human Factors in Organisations, Design and Management, 1984.

[36] H.-E. Sengler: Programmieren mit graphischen Mitteln, die Sprache GRADE und ihre Implementation; IFB 53, Springer-Verlag, 1982, 67 - 78.

[37] B. Shneiderman: Direct manipulation, a step beyond programming languages; IEEE Computer 16 (1983) 8, 57 - 69.

[38] S. L. Smith, J. M. Mosier: Design guidelines for user-system interface software; the MITRE Corp., Bedford, Mass., Report ESD-TR-84-190, 1984

[39] M. J. Staufer: Pictogramme fuer Buerocomputer; Triumph-Adler AG, WISDOM-Projekt, FB-TA-85-6, 1985.

[40] J. Stelovsky: XS-2, the user interface of an interactive system; Dissertation ETH Zuerich, 1983.

[41] W. Straub: Ein Programmsystem zur leichteren Erstellung von Dialogprogrammen mit PL360; Studienarbeit PU1S070, FG Programmiersprachen und Uebersetzer, TH Darmstadt, 1979.

[42] T. Teitelbaum, T. Reps: The Cornell program symthesizer, a syntax-directed programming environment; Comm. ACM 24 (1981) 9, 563 - 573.

[43] G. Winkler: Entwurf, Implementierung und Gebrauch eines Dialogsystems zum Erstellen von Programmen unter Steuerung von kontextfreien und kontextsensitiven Grammatiken; Doctoral dissertation, Techn. University at Darmstadt, 1977.

[44] G. D. Zincke: CAS System LITOR, Konzept und Realisierung einer Arbeitsumgebung fuer den interaktiven, graphisch unterstuetzten Softwareentwurf; H. Morgenbrod/J. Sammer (Hrsg.): Programmierumgebungen und Compiler; Teubner-Verlag, 1984, 225 - 247.

Address of the author: Prof. Dr. Hans-Juergen Hoffmann
Fachgebiet Programmiersprachen und Uebersetzer
Fachbereich Informatik, Techn. Hochschule Darmstadt
Alexanderstr. 24, D-6100 Darmstadt