AN EXAMINATION OF THE USE OF DIALOG CHARTS IN SPECIFYING CONCEPTUAL MODELS OF DIALOGS

by

Gad Ariav
Information Systems Area
New York University
90 Trinity Place
New York, NY 10006

and

Linda-Jo Calloway
Information & Communications
Management
Fordham University
Lincoln Center Campus
New York, NY 10023

August 1988

Center for Research on Information Systems
Information Systems Area
Graduate School of Business Administration
New York University

Working Paper Series

CRIS #185
GBA #88-81

Abstract[1]

The conceptual design of user interfaces focuses on the specification of the structure of the dialog, independent of any particular implementation approach. While there is common agreement with respect to the importance of this activity, adequate methods and tools to support it are generally unavailable. The Dialog Charts (DCs) yield high level dialog schemas that are abstract enough to support the conceptual design of dialog control structures. They combine dialog concepts with widely accepted design principles, in a uniform diagraming framework. Specifically, the DCs distinguish between the dialog parties, provide for hierarchical decomposition and enforce a structured control flow.

A clear set of guiding principles for the conceptual design of dialogs has yet to emerge. In this paper we have elected to focus on the notions of descriptive power and usable power, as they apply to conceptual dialog modeling tools. The conceptual descriptive power of the DCs is informally examined by applying them in a varied set of examples and relating them to their lower level counterparts, namely implementation dialog models like augmented transition networks or context-free grammars. The usable power of the DCs has been examined empirically through a qualitative study of their actual use by system designers. The Dialog Chart models were found by dialog designers to be a useful conceptual design tool, which exhibit the essential attributes identified for conceptual models.

---

[1] Ken Marr's comments on an earlier version of this paper are gratefully acknowledged.

1. Conceptual Models of Dialogs

The intensifying discussion of conceptual dialog models is an inevitable result of the recent consolidation of a dialog management paradigm. This paradigm partitions a system/user dialog into three linked generic functions: the handling of syntax, the handling of control and the handling of the applications (Figure 1-1). This conceptualization essentially underlies a wide array of contemporary dialog models, expressed in various terminologies (e.g., [37], [4], [34], [18], [1], [38], [19] [35], [14] and [16]). The set of dialog concerns is parcelled out as follows: the *syntax* defines the valid set of user inputs and captures presentation aspects, including the delivery of outputs to the user; the handling of the *applications* entails the definition of the interface to the required application modules and the passing of information to and from these modules; finally, the *control* aspect of dialog management is concerned with the maintenance and enforcement of the **dialog structure**, practically defining the set of interaction contexts and the permissible sequences of user-system activities.

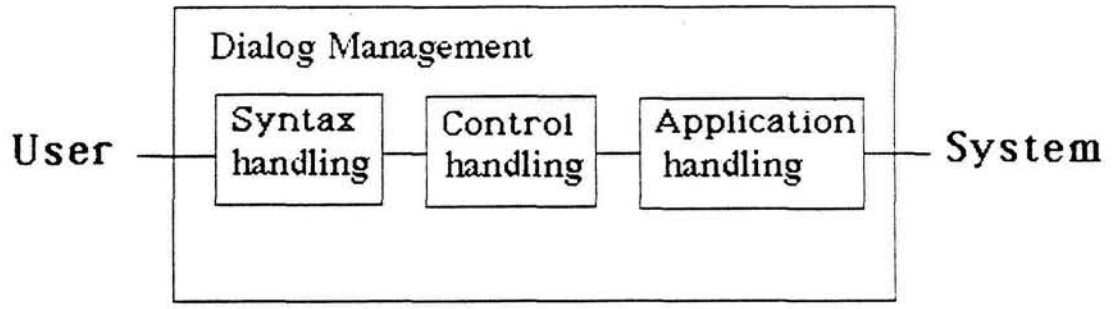


Figure 1-1: The Generic Structure of Dialog Management

One implication of the three-partite model of dialog management is that the design of the dialog structure of a system can be handled somewhat independently of both the design of the application as well as the interaction style or implemented appearance of the user interface. A model of the control structure of the dialog is, therefore, a stable abstraction of the dialog: it outlines possible sequences of system/user interactions without being bound to a specific

implementation. Concretely, such a conceptual model of the dialog captures the essential decisions about the nature of the dialog, decoupled from related decisions about, say, the variety or scope of data management services, or the provision of specific facilities for users' input/output. These models therefore guide the actual implementation of the dialog component, and allow the examination of dialog designs for correctness, consistency and simplicity prior to (expensive) implementation [12].

This paper presents and examines an approach for the specification of conceptual models of dialog. Although several methods have been suggested for modeling and specifying human/computer interactions, they are generally oriented towards programmers. These methods typically address implementation aspects of dialog design, and furthermore, they do not directly support the *process* of dialog design. The general state of the art of conceptual modeling of dialogs is rather problematic: "While there is nearly universal agreement that [conceptual design] is the most critical point in the process, there is also a nearly universal lack of adequate tools and formalisms to aid the designer at that task" (p.314 in [36]).

In developing the elusive notion of conceptual models of dialogs, analogous concerns in the area of database design provide some useful insights. The contemporary view of database design clearly differentiates among three types of data models [46]: the conceptual model (e.g., Entity Relationship Model), the implementation model (e.g., Network Model), and the physical model (e.g., file organization and access methods). The essence of a proper database design process is the gradual refinement of system specifications through the development of a consistent set of corresponding models (Figure 1-2). Conceptual models capture users' views and outline fundamental system requirements; these models are ideally expressed in ways which are directly examinable by users. Implementation models add formality and thereby disambiguate, within the framework established by the conceptual model, any implementation issues. Moving closer to the realm of computing, physical models further ascertain the feasibility of the system by translating the implementation model into concrete data and software structures, relating them to available hardware options. The analogy, it seems, can form a useful agenda for the discussion of proposed methods for constructing conceptual dialog models.

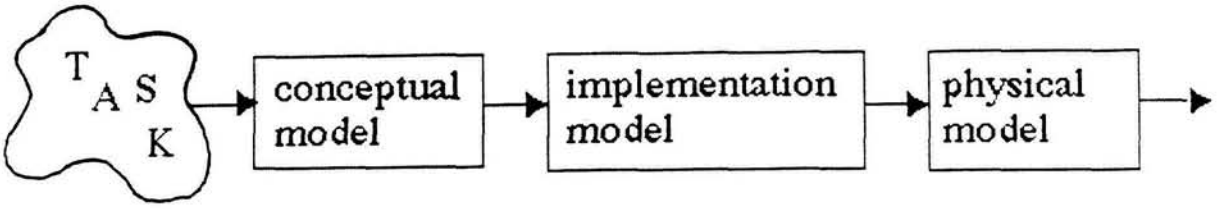


Figure 1-2: Model Hierarchy in Database Design Processes

By far the most influential conceptual data model is Chen's *Entity/Relationship Model* (ERM) [8]. Date's critical remarks concerning this model (pp.611-612 in [10]) anticipate the likely criticism of conceptual dialog models. In particular, ERM is said to be vague, imprecise, loose and not well-defined; its definition may not meet all the requirement considered necessary to qualify as "true" data model; it is said to be a "thin layer on top" of the much more rigorous relational data model; that it leaves crucial modeling aspects implicit; and that its popularity could be attributed to the diagraming technique, rather than to the ERM "per se". One can plausibly argue that precisely these deficiencies make the ERM so useful: They directly correspond to the quintessential attributes of the early stages of the analysis and design of database applications. The Dialog Charts discussed in this paper were conceived to facilitate similarly the early stages in the design of dialog structures, and critiques like the above can be rightly leveled at them. Nevertheless, the charts seem to provide an effective vocabulary for the specification of conceptual dialog models and for solving dialog design problems.

The Dialog Charts (abbreviated henceforth as "DCs"), are introduced in Section 2. One manifestation of the early formative stage which characterizes the area of conceptual modeling of dialogs is the lack of commonly accepted criteria for assessing different modeling approaches. In this paper the DCs are examined with respect to their descriptive power -- in Section 3 -- and usable power -- in Section 4. The *descriptive power* of a conceptual modeling notation, to paraphrase [16], is the set of dialog situations that can be modeled by the notation, and in turn

the set of implementation models of dialog that can be described by the notation. "The larger this set is, the more powerful the notation" (p.245). Specifically, we study a range of dialog examples as they are modeled by the DCs, and examine their implementation oriented counterparts. The *usable power* of a conceptual modeling notation, in a similar paraphrase, captures the ease of applying the notation -- it identifies a subset of the describable dialog situations which are conveniently modeled by the notation. Section 4 describes an empirical investigation of the usable power of the DCs and their usefulness, as indicated by actual user experience. The discussion in Section 5 highlights our findings in the context of contemporary conceptual dialog design issues and relates them to an alternative approach explicitly aimed at conceptual dialog modeling (UIDE [12]).

2. Dialog Charts -- Notation

Dialog Charts constitute a tool for solving dialog design problems. The concepts formulated as the framework for the Command Language Grammar [31] are used in the DCs to identify the structural elements of human/computer interactions. The design discourse assumed and supported by the DCs is made of cycles among the basic design activities of *goal elaboration*, *design generation* and *design evaluation*, until a satisfactory specification is found [26]. Finally, the types of control flows in the DCs and their diagramatic nature correspond to some key notions of the Syntax Charts [23]. A more complete discussion of the DCs viz. its underlying "ideologies" is included in [2].

The principles that define the adequacy of a conceptual *data* model seem relevant in the discussion of conceptual dialog models. Specifically, a conceptual model in our context should facilitate the identification, examination and discussion of concepts that are useful in an informal discourse about the world surrounding the application. It should provide a set of corresponding symbolic object representations; a set of rules which define proper composition and linkage of these notations; and a set of operators for manipulating those symbolic objects and their compositions [10]. The DCs are correspondingly made of diagramatic elements that are structurely linked together, and in their development they are subjected to a set of permissible manipulations, i.e., the rules that govern the refinement and decomposition of a chart.

Six distinct constructs make up the DC notation. In order to facilitate reference and manipulation of these constructs, they are associated with sets of graphic symbols (Figure 2-1). Specifically, the constructs are:

1. A decomposable user activity, i.e., a composite gesture (indicated by a box).
2. A non-decomposable user activity, i.e., "terminal" (an oval).
3. A decomposable system activity, i.e., a program (a double box).
4. A non-decomposable system activity, i.e., a reasonably "closed" and well-defined subroutine (a double oval).
5. An activity that combines user activities and system activities, i.e., a task or a method that involves user and system interaction. Such tasks could be either user-led or system-led (indicated by different combinations of a half single, half double box).
6. Direction of flow (indicated by an arrow). The basic flows permissible are selection, iteration, sequence and case. These can be combined arbitrarily.

The arrows represent the directions of the sequences, and thereby play a critical role in the DCs' capacity to explicate structure. By limiting the repertory of flows to those commonly associated with structured programming approaches, a measure of desired quality is enforced on the result of the design. Specifically, structured flows can aid in identifying robust dialog logic and modular dialog design. Similar arguments have motivated the inclusion of these constructs in lower levels of dialog modeling (e.g., [3]). Junctions in the diagrams represent decision points, and are resolved by whomever holds the initiative at that point. The party (i.e., either user or system) whose range of actions is specified in the routes that branch out of the junction holds the dialog initiative and selects the actual dialog path to be followed. This approach requires the adherence to ***homogeneity constraint***, namely that all the paths that emanate from a junction will be either all user-led or all system-led. It also brings out the fundamental decision on the assignment of dialog initiative, and explicitly calls for its resolution.

The major manipulation in dialog charting is the transformation of an element in a DC into a detailed chart. The range of these manipulations and the associated rules have been kept intentionally limited, to preserve simplicity. Specifically, any box can be further decomposed. It can be decomposed into more boxes or into boxes and ovals, or into ovals. However, once a box is either "all user" (i.e., single-lined box), or "all system" (i.e., double-lined box), it can only be

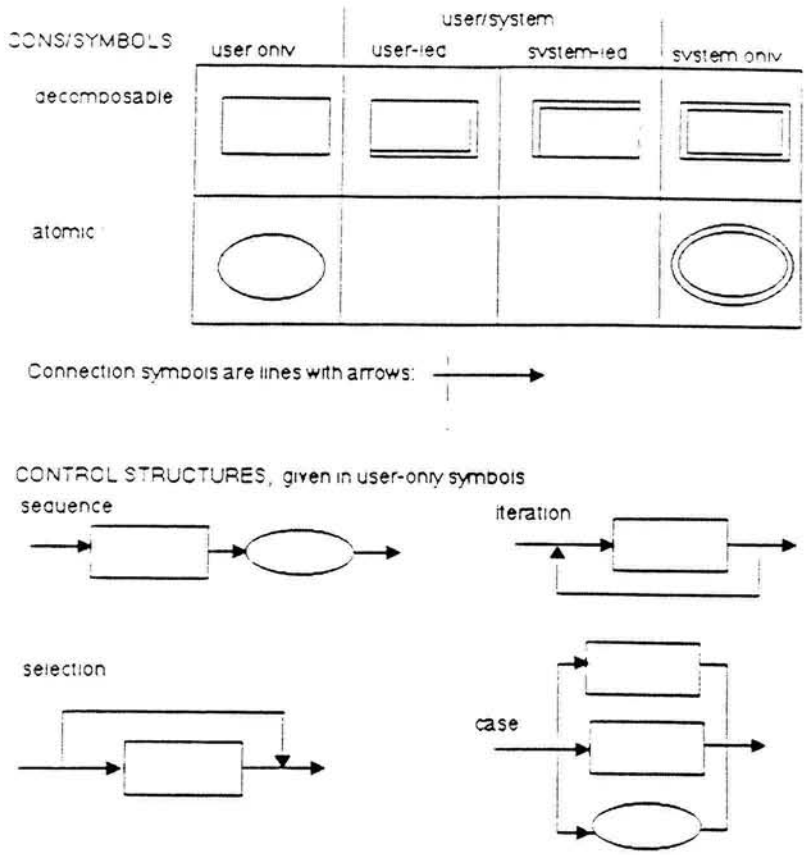


Figure 2-1: Dialog Charts notations and icons

decomposed into more boxes and ovals of the same kind. Ovals are atomic and can't be further decomposed. An additional restriction, the homogeneity constraint, applies to the choice of the first ("left-most") element in the decomposition of user/system activities or "mixed party" tasks. Specifically, this leading element has to reflect the definition of the original task as either user-led or system-led. Therefore, for instance, the decomposition of a user-led task should be led by (i.e., "start with") only user activities (either boxes or ovals), or other user-led mixed-party tasks. The purpose of this constraint is to facilitate the inheritance, consistency and homogeneity of dialog initiative among the various levels of specification and decomposition.

The specification of error handling procedures within the general structure of a dialog tends to obfuscate designers' and programmers' views of the underlying structure. To avoid this confusion, the DCs follow the notions embedded in the Syntax Charts of Jensen and Wirth [23] and support the concept of designing only the permissible dialogs. Only the accepted, proper flows through a dialog are considered. Error handling procedures are added to the DCs as annotations at the appropriate system level. If a procedure applies throughout the system it is stated at the highest level of specification, and if it applies only to a specific junction it is noted at that junction only.

A classical issue in design, and especially in conceptual design, is how deeply should the structure be decomposed. A related issue is when an element is declared as a terminal rather than as a further decomposable. There is no clear stopping rule for the elaboration process. Design common sense, however, indicates that the process should stop either when further decomposition does not offer new relevant insights, or when there are no more decomposable dialog elements (i.e., "no more boxes").

The Dialog Charts belong primarily to the category of analytic methods for dialog design, namely those methods which employ an abstract and somewhat formal representation of an interaction. The DCs focus exclusively on conceptual dialog modeling, and address its essential aspects by integrating simple visual concepts, structured flows, hierarchical decomposition and distinguishable dialog parties. While no single tenet of the DCs is in itself novel (as clearly indicated by the citations earlier in this section), their **integration** in the context of dialog design is. The Charts were initially developed in 1982 and were used since then in dozens of system development projects where interactive decision support systems and online database systems were designed. The DCs are typically taught and demonstrated in about an hour of class instruction, during which sufficient proficiency is gained. The work of selected teams of designers has been studied more carefully as part of an ongoing research project on the patterns of actual use of the DCs [6]. A summary of this research is included in Section 4 below.

3. The Conceptual Descriptive Power of the DCs

The descriptive power of a notation for the conceptual design of dialogs ultimately relates to the range of dialogs situations which can be described by the notation. Obviously complete enumeration of dialog situation is a formidable task, if at all feasible, so current discussion of descriptive power has to resort to illuminating examples. In Section 3.1 two common dialog situations are modeled, demonstrating primarily the variety of conditions in which the DCs are applicable.

A conceptual model should also be the broad generalization of "all" of its corresponding implementation models, and ideally generalized to the point that it is independent of any *specific* implementation model. At the same time, it should provide a concrete but semantically rich framework upon which the implementation models will be defined -- in principle the result of the conceptual design should directly provide the basis for the "first iteration" in the subsequent implementation design. Evaluation of this aspect of the descriptive power of the DCs can therefore be accomplished through a critical examination of the mapping between a conceptual model and corresponding candidate implementation models. This is the subject of section 3.2 below.

3.1. DC Models of Dialog Situations

The two cases in this section demonstrate the use of the DCs in the design of new dialogs or the analysis of existing ones. First, the DCs are applied to the conceptual design of a LOGIN command in a Military Message System (Section 6.4 in [21]). In a second example the DCs are used to model and describe the structure of basic dialog of the popular Lotus 1-2-3 product. For demonstration purposes this section deliberately focuses on simple examples.

In the LOGIN task a user enters into a dialog with a computer in order to establish a session [21]. The specific scenario is as follows: The user enters his or her name. If the system doesn't recognize the name, the user is prompted to try again. When the user enters a valid name, the system prompts for a password. The user gets two tries to enter a correct password

and proceed. If an incorrect password is entered twice, the user must begin the whole command again. On receipt of a correct password, the user must select a security level for the session, which must be no higher than the user's security clearance. "If he enters a level that is too high, he is prompted to reenter it, until he enters an appropriate level. If he does not enter an appropriate security level, he is given the default level unclassified." (p.44 in [21]). Note that the specification is somewhat ambiguous with respect to the dialog logic – there are two consequences of entering an inappropriate security level.

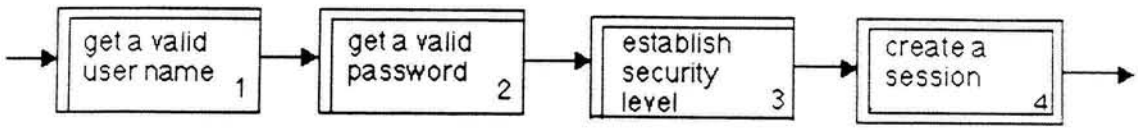


Figure 3-1: MMS LOGIN Session, Top-most level DC

The DCs for this scenario are provided in Figures 3-1 through 3-3, with each figure representing a different level of system elaboration. Figure 3-1 represents the top-most view of the session. It allows the designer to partition clearly the overall flow into well defined concerns.

In some cases, the first level of elaboration may be enough. However, in order to gain more insight into the LOGIN procedure a further elaboration should be worked out. Figure 3-2 includes two successive levels of elaboration for the box numbered 2 in Figure 3-1. In another example, Figure 3-3 represents a second and third level "explosion" of the box numbered 3 in Figure 3-1.

Note how the use of the structured DCs forces the designer to disambiguate the verbal description of the session. In the DC, the interpretation is explicit: The user is either allowed to indicate no security clearance, or is allowed to enter a valid security clearance level.

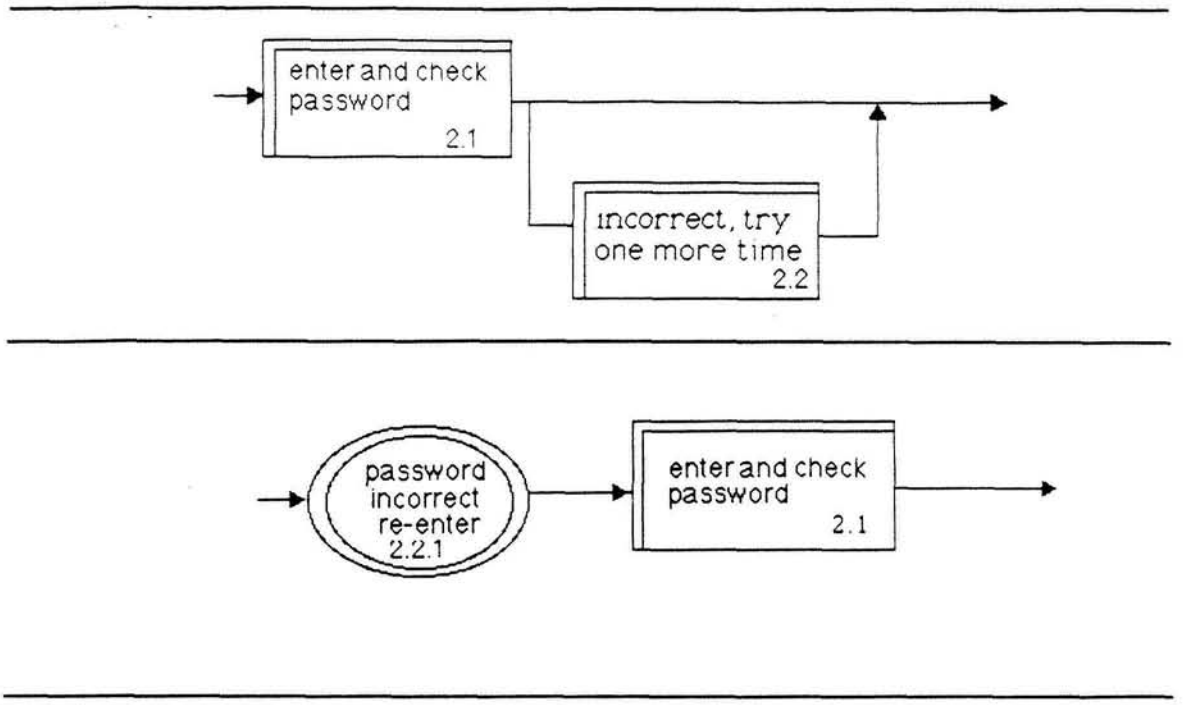


Figure 3-2: Levels 2 and 3 DCs for "Password Getting" subtask

The complete set of DCs for the LOGIN session shows which system modules and subroutines need to be programmed, those in double-lined symbols. The collection of the double boxes and ovals therefore serves as a preliminary blueprint for the detailed design of the applications and the application processor. If, however, all double boxes and ovals are removed from the charts, the remaining set of connected user actions (i.e., the single-lined elements) constitutes a broad definition of the user interface syntax, as it practically identifies the complete valid user-generated syntax.

A (partial) description and analysis of the popular spreadsheet package 1-2-3 (by Lotus Development Corporation) is conducted in Figures 3-4 through 3-6. In Figure 3-4 the top level of interaction is specified, indicating clearly the extent of choices available to the user. Figure 3-5 is an explosion of the user-led task labeled **Commands** in Figure 3-4, highlighting the choices available to the user at that stage. Figure 3-6 further elaborates on the structure of the function **Copy** that has been offered to the user at the **Commands** level dialog.

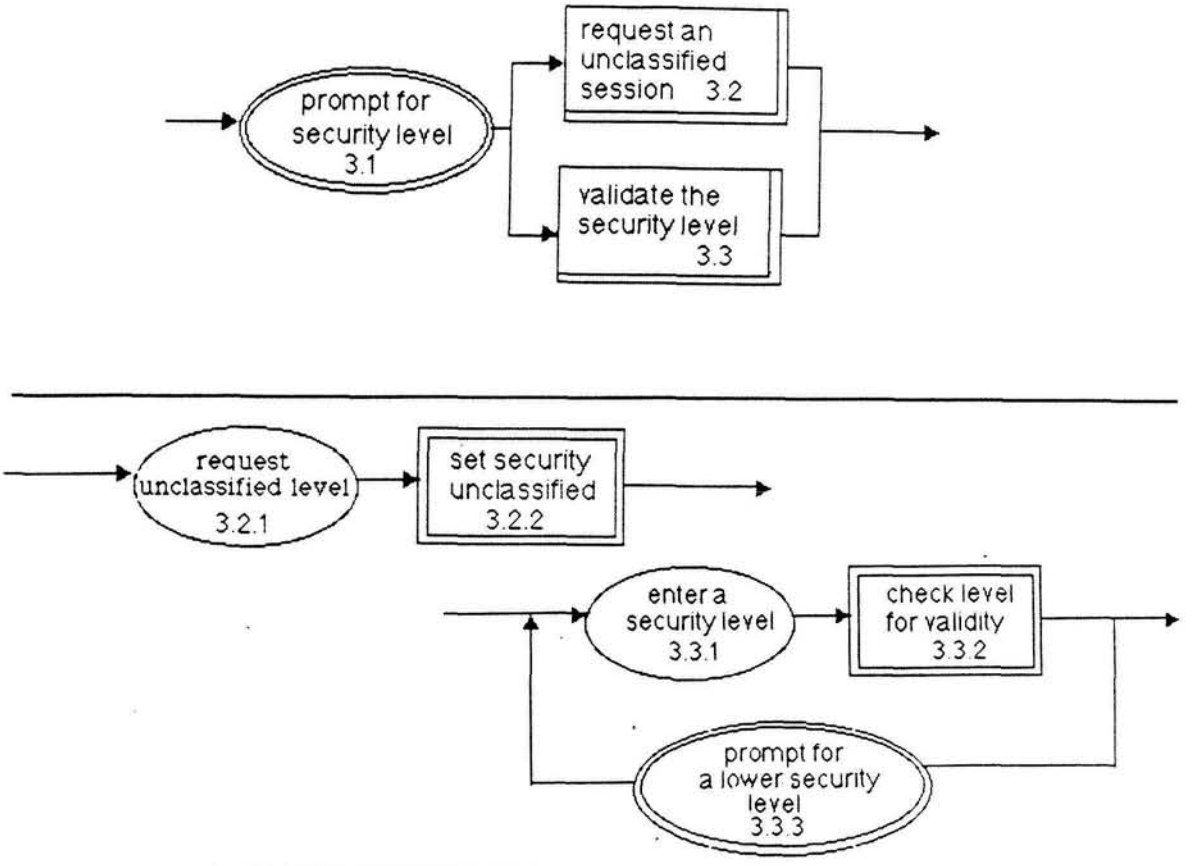


Figure 3-3: Levels 2 and 3 DCs for "Security Establishing" subtask

This analysis of an existing dialog highlights some interesting observations about the DCs. As far as the explication of the extent of control goes, the three figures are visibly different -- the taller the figure, the looser is the structure, and the user has to confront a wider set of choices. This in itself is neither "good" nor "bad", but rather indicates instances in the design where tradeoffs between freedom and confusion should be evaluated. The structure of the **Copy** command is markedly different from the other two -- it is closer to a linear, tightly controlled sequence, with relatively limited extent of user choices in carrying out the task involved. The DCs also render explicit the lack of "structuredness" in the sequence of activities that leads to quitting the session (the "extra" exit from the bottom box in Figures 3-4 and 3-5). Again, the DCs bring the unstructured sequences to the designers attention, adding it to the design agenda.

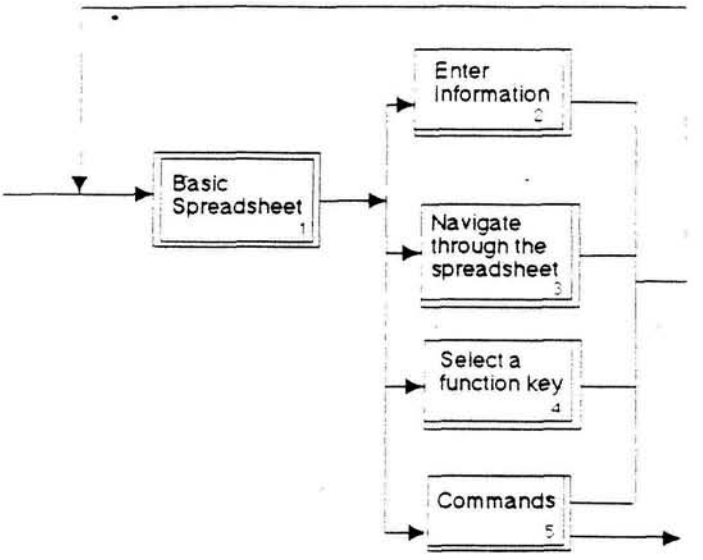


Figure 3-4: DCs for Top Level Lotus 1-2-3 Dialog

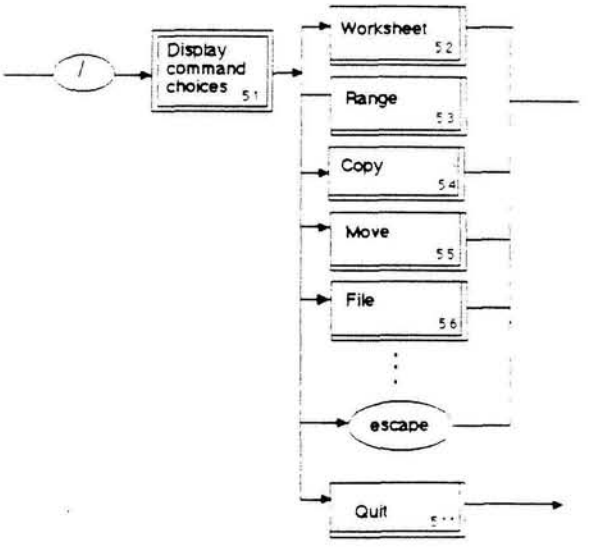


Figure 3-5: DCs for Lotus 1-2-3 **Command** Dialog

The final decision whether to retain that structure or "correct" it is a question the designer has to ultimately decide upon.

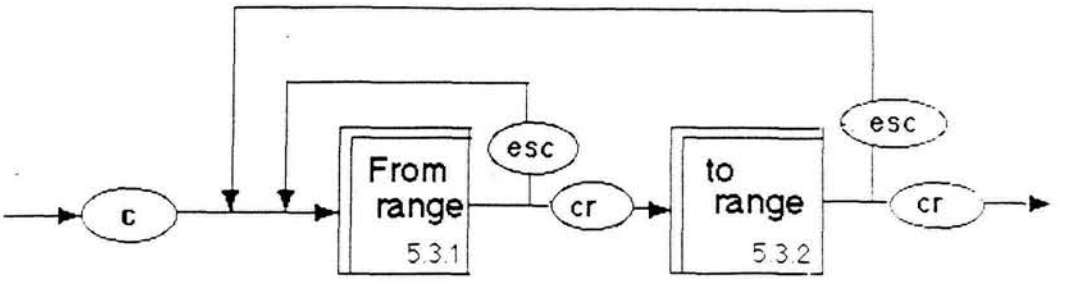


Figure 3-6: DCs for Lotus 1-2-3 **Copy** Dialog

Another comment relates to the wide range of implementation possibilities addressed with a DCs-based model. As it turns out, each of the three dialog models in Figures 3-4 through 3-6 is implemented in a different interaction style: the top-level dialog is implemented as an unprompted interaction, the **Command** follows primarily a menu-style interaction, with an alternate unprompted and abbreviated style, while the **Copy** command is implemented in a Question/Answer style, with direct manipulation being an optional type of user's gestures. The actual decision about interaction style is probably affected somewhat by the fundamental properties of the dialog as they are picked-up by the DCs, but the determining factor is a set of assumptions about the user. Otherwise, the sharp difference between the implementation of the top-level and the **Command** dialogs cannot be easily explained.

A more rigorous examination of the DCs is obviously called for. In the following section the issue of descriptive power is further discussed with respect to the correspondence between conceptual and implementation models of dialog.

3.2. Relating Conceptual and Implementation Models

In this section we reflect on the extent to which a conceptual model expressed as DCs provides a generalized description of its semantically corresponding implementation models. The core of the discussion is a critical examination of the mapping between a conceptual model and

corresponding candidate implementation models. Such an examination may indicate any limitations on the extent of correspondence between the two levels of modeling involved. This "correspondence criteria" is clearly only a necessary condition for a conceptual modeling tool, but the examination in this section also addresses the complementary issues of communication effectiveness and description efficiency. The discussion further highlights the visible differences in the way conceptual and implementation models express the same dialog situation.

A single example is featured in this section, adopted from [16], for which four models are formulated, using in turn a context-free grammar (BNF-based), an ATN, an event model and the DCs. Methodologically we use Green's article [16] as the broad definition of the realm of currently acceptable methods for describing dialog control. The task in this example is the "rubber band" drawing of a line on the screen. This example is particularly interesting since it takes the issue of conceptual dialog modeling deep into the "territory" of user interface implementation -- the task is an exercise in direct manipulation that is typically described in terms of specific user gestures. In this type of drawing, a line is anchored at one point, and extends to the current position of the cursor. It moves with the cursor until a user gesture nails it down and fixes it. Figure 3-7 includes a DC model for the rubber-band dialog. It only provides a top-level view of the interaction, which seems to fit the level of complexity (or the lack of it) in this specific user/system exchange.

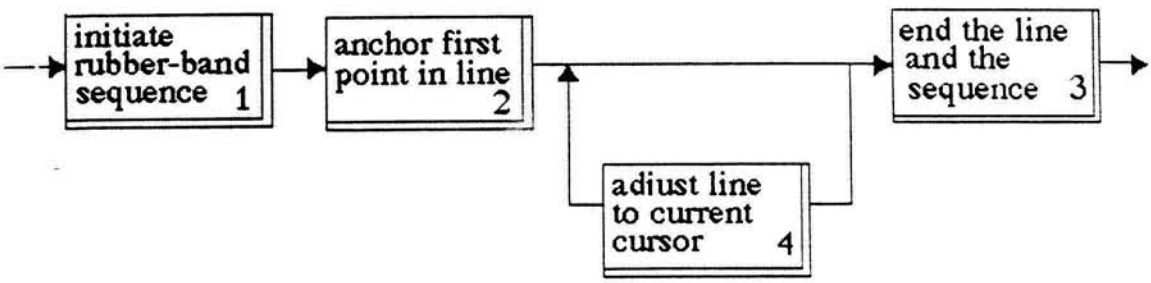


Figure 3-7: A DC Model of the Rubber Band Line Task

Backus-Naur Form (BNF) is an example of a context-free, production rule grammar.

Languages are described with this grammar as a set of rules, each specifying a substitution of a composite term by its constituent terms. The chain of substitutions eventually results in a fully specified string in the language [21]. Since dialogs are carried out through preordained expressions ("languages" of sorts), this type of grammar has an obvious appeal in the modeling of dialogs.

Reisner's Action Language Grammar (ALG) [39] is a characteristic methodology for analytic dialog modeling. It uses a BNF notation to define a formal grammar that describes actions taken by the user while interacting with the system. ALG could not serve as an implementation model for DC-based conceptual model since they are fundamentally incompatible -- ALG only deals with one part of the dialog. The Multiparty Grammar [43] extends the Action Language Grammar, and allows the designer to explicitly identify human actions and computer actions. The model in Figure 3-8 is an implementation model of the rubber-band dialog, expressed through a multiparty dialect of BNF.

---

line → button end_point

end_point → move end_point
| button

Context-free grammar for rubber band line **example.**

Rubber band line example with program actions.

line → button d1 end_point

end_point → move d2 end_point
| button d3

d1 →
{ record first point }

d2 →
{ draw line to current position }

d3 →
{ record second point }

Figure 3-8: BNF Models for Rubber Band Line Task (from [16])

State transition diagrams are used for describing finite state machines, and have been used for quite some time to model dialogs [37]. When defining an interface with these diagrams, the

nodes of the network correspond to different states or modes of the interaction. Arcs linking nodes have one or more input events, output events, or application actions associated with them [19]. This basic form of transition networks has been augmented in a number of different ways with a variety of additional features, forming what is referred to as *augmented transition networks* (ATN). For instance, the nodes in ATNs may also represent subnetworks, recursion and calls to other nodes [21] [22], *Super states* [9], and *subconversations* [48]. Figure 3-9 represents the rubber-band drawing task as an ATN-based dialog model.

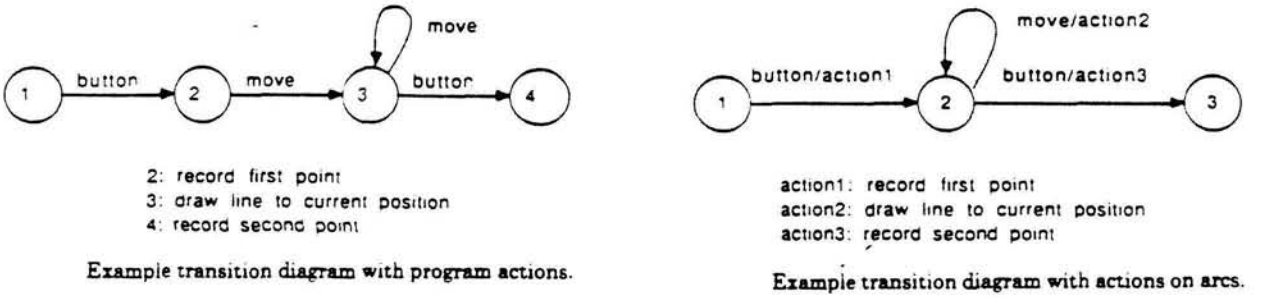


Figure 3-9: An ATN Model for Rubber Band Line Task (from [16])

The event model of dialogs decomposes the interaction into independent descriptions, each of which encapsulates a single, somewhat self-contained activity of the user (e.g., a gesture), the control processor, or the application [16]. As befits an implementation model, the event model is strongly associated with a UIMS approach in which active *event handlers* are responding to identified events by invoking the associated procedures to "process the event". Such event processing can compute, generate events, and activate or deactivate other event handlers. An event handler is defined by the set of events it can handle, so activation and deactivation of a handler practically determines the set of permissible events – all the events recognized by the concurrently active handlers at a given point in time.

The event handler in Figure 3-10 outlines an event model for our running rubber band line example. The TOKEN section in it associates external events with event handling procedures,

the VAR section defines the local variables, the repeating section EVENT defines the event handling procedures themselves, and the INIT section sets up initial conditions. The disjoint nature of an event model makes it a natural choice for modeling a concurrent processing environment, or *multithreaded dialogs* [16].

```
EVENT HANDLER line:

TOKEN                VAR                  EVENT Button DO {              INIT
 button Button:       int state:           IF state == 0 THEN             state = 0;
 move Move:           point first, last:    first = current position:
                                            state = 1;
                                           ELSE
                                            last = current position:
                                            deactivate(self);
                                           ENDIF;
                                          };

                                          EVENT Move DO {
                                           IF state == 1 THEN
                                            draw line from first to current position:
                                           ENDIF;
                                          };
```

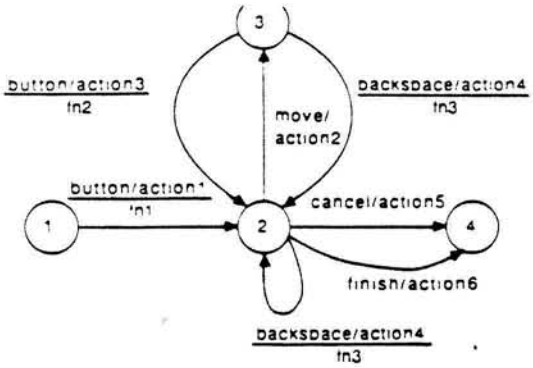
Figure 3-10: An Event Model for Rubber Band Line Task (from [16])

In general it seems that BNF, ATN and event models address the implementation design of dialog control, and therefore do not directly *compete* with the DCs in the "conceptual arena." They rather complement each other in the overall process of interface design. Nevertheless, contrasting the DCs with the three other modeling approaches illuminates some key concerns in the conceptual modeling of dialogs. The following comments highlight two of them, specifically the explication of dialog semantics and dialog control structure, and the formulation of models with varying levels of detail.

BNF-based models describe all possible grammatically valid dialogs. As noted by [21], BNF-based representations can not explicitly represent control structure. BNF-based models, hence, do not differentiate among meaningful dialogs and dialogs that are meaningless to the system or user [39] -- the meaning of the dialog, i.e., its semantics, have to be handled elsewhere. This is resolved most naturally within a DC-based conceptual model: with their explicit specification of interaction context the DCs constrain the dialog description beyond solely

grammatic validity. Although the control structure is explicit in an ATN, it is nevertheless obscured by implementation-related details and large numbers of arcs that can form an unconstrained network (as opposed to a structured network). The augmentations in the ATNs are efforts to use them to capture more of the semantics and complexities of the various dialog components. The event model explicitly manages these aspects of dialog control, but the details of the control structure itself are "buried" disjointly in the actual event handling code. Event handling approaches result in descriptions of dialog that are difficult to examine observed, and input/output languages that cannot readily be determined without inspecting the actual code of the event-handler itself. The dialog's control structure is not directly observable and has to be pieced together outside the model. The conceptual design, as expressed in Dialog Charts, makes the control structure of the interaction explicit. It is important to notice that the basic flow, structure and connectivity of the design are apparent even *before* any particular interaction syntax is specified. Moreover, as we saw in the Military Message System example in the previous section, a detailed specification of interaction syntax can easily be derived from the DCs.

The Dialog Charts also directly address the issue of high level modeling of the control structure by allowing the designer to defer decisions about specific interaction sequences and applications interface issues. Approaches that emphasize formal accuracy cannot easily accommodate such deferment. Both the problems of obscured control structure and of overwhelming implementation details become more acute when a more complex task is analyzed. Consider a slightly more complex example, extending the line drawing into a rubber-band polyline drawing with cancel and backspace options. The rubber-band polyline drawing itself is a collection of individual lines that are created as rubber-band lines and connected end-point to end-point. Figure 3-11 presents functionally equivalent DC and ATN models for this task (the ATN model is directly quoted from [16]). The DCs are more abstract than the ATNs and therefore they incorporate the options easily in a higher level abstraction. As has been shown, this structured diagram can be decomposed to specify any particular interaction syntax that is desired.




```
action1: record first point
action2: draw line to current position
action3: record next point
action4: erase last point
action5: erase polyline
action6: return polyline

fn1: count:=1; return(true);
fn2: count:=count+1; return(true);
fn3: if count = 1 then
        return(false);
     else
        count := count-1;
        return(true);
```

Polyline dialogue with cancel.

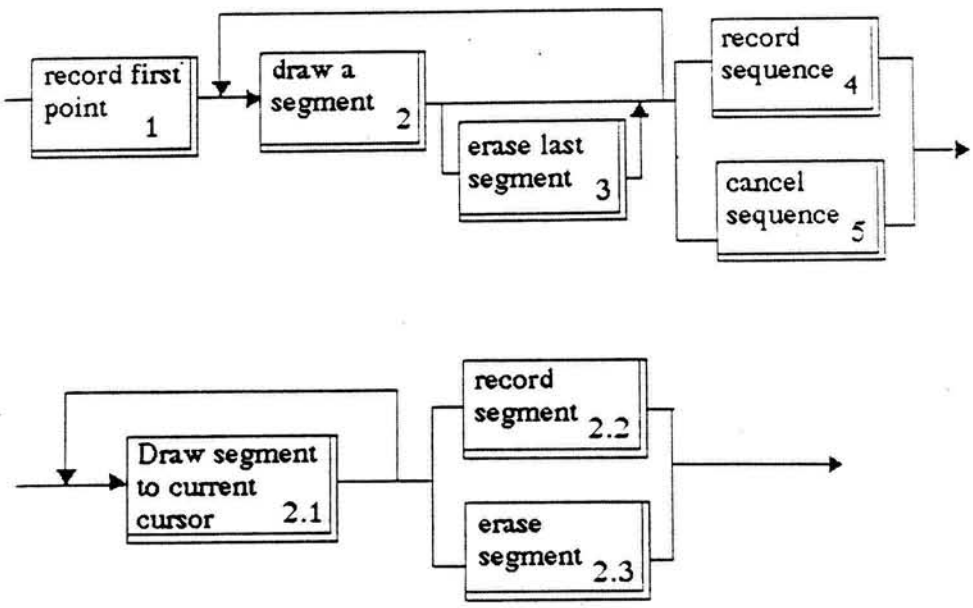


Figure 3-11: ATN and DC Models for "Polyline" Task

4. The Conceptual Usable Power of the DCs

Against the backdrop of the conceptual descriptive power of the DCs, this section explores the ease with which designers can employ the charts in conducting system design activities. Although the DCs appeared to be well received by their users in varied design situations and a wide range of applications, there was no methodical basis for substantiating this anecdotal evidence. Apparently this is not an unusual situation: "Most people who have built tools for interface development claim that these tools enhance designer performance. The authors are not aware of any empirical evidence to support these claims" (p.233 in [20]).

The empirical investigation reported in this section was explicitly aimed at answering the questions of **how designers actually use conceptual design tools** and therefore **what makes dialog design tools useful**. In the absence of existing firm theory of the use of conceptual dialog modeling tools, the immediate need was the development of a more concrete set of research "concerns" or questions that operationalized the notion of usefulness or the essence of "usable power" [16]. The corresponding questions that eventually guided our study were:

1. For what meaningful design **purposes** are the DCs used?
2. In what **stages** of system development do designers find the DCs to be helpful?
3. What are the perceived effects of the DCs on the **products** or results of the design effort?
4. What are the perceived effects of the DCs on the nature of the design **process**?
5. How do designers feel about the DCs and about using them? Have any **attitudinal patterns** developed towards the DCs?

These questions are meant to encapsulate the notion of the tool's usefulness [6]. The overall vantage point is that of the designer, and therefore subjective terms such as "meaningful," "useful," "perceived," or "attitude" are to be interpreted from the perspective of a designer who actually uses the tool, the DCs in this case.

The research strategy adopted in this exploratory study has been qualitative, as outlined in Section 4.1. In seeking valid responses to the above questions our approach draws primarily on concepts of grounded theory [15], [27], qualitative analysis methods [30], and qualitative content

analysis [25]. In Section 4.2 we present the results of applying this methodology in studying a team of designers who had just concluded a system development project in which they used the Dialog Charts.

4.1. Research Methodology

The overall research design was a field experiment [28], which occurred over a period of about three months. The experimental task was the analysis, design, development and demonstration of an interactive database application. The application's scope, complexity and development mode were realistic -- a team-based development setting of a system of about 1000 lines of high-level code. The team was made up of 4 undergraduate students in their senior year, all Information Systems majors, who were enrolled in a course on the analysis and design of interactive systems. The course included a review of various methodologies (including the DCs) for disciplined design of information systems and databases, and design teams were generally expected to apply these tools. Participants' inexperience (relative to practicing information systems professionals) does not seem to limit the generalizability of the results. In the era of end-user computing many designers of interactive systems, especially those engaged in specifying the decomposition of the task, are not thoroughly trained in systems design. As it turned out, most of the participants took up jobs that required them to participate immediately in designing interactive systems.

The basic premise of our approach to data collection and analysis is that to be "useful" to the designer, a tool and/or methodology must **be** used, **aid** the designers in achieving their goals and objectives, and finally be **perceived** as being useful. If it is not perceived as useful, there is no reason to suspect that it will be used again regardless its actual benefit [11]. This translates into the assumption that inferences can be made from "revealed perceptions" about the usefulness of the target tool as they surface in the designers' retrospective reflection on the design and building of their system -- after the tool has been actually used.

The main objective while capturing the data was to solicit designers perceptions of the DCs

in an unobtrusive fashion. Following the completion of the development of their system, the team participated in an open-ended, semi-structured and funneled interview with a hidden-agenda [7], [17], [45]. In such an interview questions are prespecified, but the answers are not, and the broad range of questions masks the identity of the actual topic under study. The funneled interview begins by asking questions about a general area or domain, and then pursues areas that have been mentioned by the interviewed team more specifically. The results of this approach are that each issue on the interview's hidden agenda is approached with the broadest and most open questions first. These are followed by more specific questions, often rephrased according to the specific language used by the informants.

The interview provided, therefore, a loose structure within which the designers related to their system development experiences. The first segment of the interview established the overall context (i.e., What does the system do?), and then focused on the work that the team accomplished in the various stages of system development -- from conceptual design all the way to actual coding. The second segment of the interview raised two issues. It started with a discussion of the problems encountered in specifying, designing and implementing the dialog. It then brought up the topic of design tools, and future intentions regarding tools that have been used. Throughout the interview no direct focus was placed on the DCs in order to preserve the hidden agenda, and to guarantee that information about how the designer used the DCs was, to the extent possible, voluntary. The interviewer was an outsider who did not participate in any of the previous stages of the experiment itself.

The audio-taped interviews provided the archival raw data for analysis. Basically, qualitative analysis consists of progressively reducing and categorizing raw data into various forms of *display*, i.e., "an organized assembly of information that permits conclusion drawing" [30]. The analysis is an iterative process of data reduction, display, and conclusion refinement. In this way, the data which at first seems vague and inchoate gradually becomes more explicit and "grounded" [15]. Initial data reduction of the taped interview was achieved through a structured content analysis; i.e., semi-mechanical tracking, extraction, transcription and

categorization, of explicit "mentions" of the DCs [24]. A *mention*, the basic unit of analysis in this study, is a group of utterances made by the designers about the tool, within a design context and categorization. A change in the broad context or major category signals the end of the mention. Mentions occurred in *sequences*, i.e., one or more mentions that are contiguous. If a sequence arises spontaneously, it is an *unsolicited* sequence of mentions, while if it is triggered by a follow-up probe, it is a *solicited* sequence, and all its mentions are therefore solicited as well.

The mentions were encoded by studying their relationship to the concept of usable power, as operationalized by the set of five research concerns listed earlier. These concerns were further differentiated into a set of *seed categories*, representing a concrete contextual framework for categorizing the empirical data. The initial set of categories used in this study is summarized in Figure 4-1. The formal statement of seed categories makes explicit assumptions and expectations about the nature of the researched phenomena [30]. The seed categories in this study were adopted from the "parent disciplines" of systems design and software engineering.

- **Q1. Purpose**
  Gathering intelligence, goal elaboration, design generation, design evaluation, communication
- **Q2. Stage**
  Documentation and analysis, logical/conceptual design, implementation design, programming/coding, testing
- **Q3. Product**
  Modularity, control structure, data structure/architecture
- **Q4. Process**
  Design philosophy, constraints
- **Q5. Attitudinal Patterns**
  Learning, task performance, subjective satisfaction, retention, errors

Figure 4-1: The Original Seed Categories

Seed categories are further differentiated during the data analysis, as categories and data are subjected to *constant comparison* with new information [15]. The encoded mention frames

either fit in any of the existing categories or a new category is declared. In the process, the initial categories are partitioned or combined, new ones are added, and new properties and value sets are noted as suggested by the mentions that are encountered during data collection, reduction and analysis [30]. A mention can relate to more than one question or category. Such an overlap might represent a complex perception or a link among the categories. As far as possible, the findings are articulated by formulating the data-context relationships so that the data appear as independent variables and the context targets appear as the dependent variables [25].

The basic assumption is that mentions, because of the unobtrusive and free-response interview form, faithfully represent and reveal the *perceptions* of the designer. Making inferences from mentions about *actual* use is problematic, though -- the linkage between mentions of use and actual usage is not directly observable. For example, it is possible that some users will not voluntarily mention using the DCs. In this case we assume that although the DCs were used, it is unlikely that they were perceived as either useful or as a significant part of the development process.

The empirical research reported here is in a sense a case study of a single dialog design tool and a single team of designers. Even though a case study is scientifically "weaker", it nevertheless rich and unconstrained, as befits the preliminary state of understanding dialog design processes.

4.2. The Usability of the DCs

The summary of the findings is presented in three complementary fashions. Following a brief discussion of the broad distribution of mentions into categories, we consider the observations category by category. Finally, we comment about the observed relationships among the categories indicated by the data.

In all, there were 49 mentions of the Dialog Charts throughout the interview. Ideally, we wanted mentions to fit unambiguously into only one category. It turned out that the identification of such mentions often led to mentions of very few words that were stripped of

meaningful-context. Therefore, we operated under a more relaxed guideline that a mention should provide enough context to be understandable in itself, and yet should fit, i.e., be coded, *primarily* into not more than two categories. The 49 mentions received 80 such codings. We considered a mention as "reliably coded" if there was no disagreement about the applicability of the codings, although there might be other codings that could apply. A "coding consensus" among the researchers was sought in the cases where mentions could be interpreted in more than one way.

Figure 4-2 summarizes the mentions of the DCs by the team and underscores the richness of the information that is under study. Although no major categories had to be added (c.f. Figure 4-1), some new secondary categories have emerged (marked with an asterisk in Figure 4-2); In particular, the DCs were mentioned in connection to *system maintenance*, a system development **stage** that was originally thought to be far removed from a conceptual design tool; *user orientation* was identified as another characteristic of dialog design **products**, and *task clarity and comprehensibility* was added to **attitudinal patterns**. All third-level categorizations have been suggested by the empirical data. They basically refine their corresponding categories and give them a more precise and concrete interpretation. Since the analysis reflects a single site, no categories are being dropped, although some are currently "empty." For instance, there was no mention of the use of the DCs for the purpose of *intelligence gathering*.

The 49 mentions occurred in 23 sequences (Figure 4-3). Counter to our expectations, the extent of mentioning the DCs in the two segments of the interview was similar. The total number in the first segment -- largely unsolicited mentions -- were 27 mentions and 44 codings, in 11 sequences. During the second segment, in which somewhat more direct questions were posed, there were 22 mentions in 12 sequences, which were coded into 35 categories. Figure 4-3 shows the mentions, sequences and codings grouped according to whether or not they were solicited.

Interestingly, Figure 4-3 does not reveal substantive disparity with respect to codings in each segment, or along the solicited/unsolicited dimension. Our original emphasis was on the

Purpose	totals by sub-category		
		Unsol	Solic
Intelligence			
Goal elaboration		0	2
Goals into sub-goals	2		
Design generation		6	9
DCs as step before pseudocode	2		
Queries and prompts	1		
Menus	3		
Designing the "user interface"	2		
User orientation in interface design	4		
Error processing	2		
Coding	1		
Design evaluation		1	3
For completeness	1		
Menu structures	2		
Flow of control	1		
Communication		5	4
To later stages	2		
Logical to physical	4		
Logical to coding	2		
Logical to maintenance	1		
Stage			
Documentation and analysis			
Logical or Conceptual design			
Physical/Implementation design	1	0	1
coding	2	2	0
Testing			
*Maintainance	1	1	0
Products structures			
General modular/hierarchical	2	1	3
DCs as a map, diagram	2		
*User orientation to product	2	1	1
Control structure	2	0	2
Data structure/data flow	2	0	2
Process of design			
Constraints on design		1	1
Thoughts forced into something concrete	1		
Help maintain strong control	1		
Philosophy of design		1	3
Distinguish the parties	2		
Top-down decomposition	1		
Iteration	1		
Attitudinal patterns			
Learning		6	0
Dialog Charts/should have learned value earlier	6		
Recall/retention	2	2	0
Task performance		2	0
Time to do menus/little	1		
Time to do DCs/lots, because "had" to re-do them	1		
*Task comprehensibility		0	3
Saw all levels of depth	1		
Saw control structure	2		
Subjective satisfaction/dissatisfaction		8	9
Intend to use DCs	4		
Surprised, because tool useful	4		
Adjectives valuable, important, etc	8		
Confident in code	1		

Figure 4-2: Tabulation of Mention Codings

		Sequences	Mentions	Ccodings
SegmentI	probed	2	13	18
	unprobed	9	14	26
SegmentII	probed	4	14	25
	unprobed	8	8	11
	Totals	23	49	80

Figure 4-3: Tally of sequences, mentions and codes by interview segment

authenticity of unsolicited mentions. At least for this team it seems that the intensity and nature of reference to the DCs were not sensitive to the solicitation or the lack of it. One possible interpretation is that the team has formed fairly stable opinion about the DCs, and therefore related to them consistently across the different modes of evidence gathering. An alternative interpretation is that the unobtrusive funneling was not actually so unobtrusive. The wide dispersion of the mentions through the conversation and abundance of references to other tools and aspects of the design process make the latter interpretation rather implausible. Asking for clarification of the information by soliciting with a probe led though to relatively longer sequences of mentions, which is not unexpected.

Distribution of Codings

By far, the majority of the mentions related to **purpose** (29 mentions) and **attitudinal patterns** (30). With respect to the **purpose** of use category, Dialog Charts were mentioned most frequently in the context of ***design generation*** (15 mentions). While this could be expected, somewhat unexpected was the intensity of mentioning the use of the tool for the *communication* of design information (9 mentions). Although communicating is a characteristic of a usable development methodology [47], there was no requirement that the team use DCs as a communications vehicle.

The 30 mentions categorized under **attitudinal patterns** came as a surprise: *subjective satisfaction* was the context for 17 of the mentions. Eight of these referred to the DCs as *valuable and important*, while 4 referred to the *surprise* of the team and the usefulness of DCs. Needless to say, no such information was requested in the interview.

Category by Category Summary of Mentions

In the following paragraphs, the content of mentions in each category is summarized and illustrated by examples. The mentions are reproduced in their entirety in [6], and the numbers in the parentheses following mention quotations refer to the mention sequence number in the format of *(team-id.Mention-seq)*.

Purpose: *Goal elaboration* includes decomposing goals into potential sub-solutions or subgoals. The team did not mention their process of *Goal Elaboration* until late in the interview, when asked explicitly about dialog design. There they described using DCs to "differentiate between system -- response or function, and user response or function, or something that's a combination of both" *(C.29)*. The products of *goal elaboration* are the functional requirements of the system, as paraphrased in the following mention:

> - And you start out with the very simplest, the hightest level... break that down, and you go down and down and down until you hit the lowest level. You hit every possible situation.
> - You can't explode anymore.
> - Until you don't need to prompt the user for anymore information.
> - ...and you can just perform the necessary functions. *(C.30)*

Design generation was the most frequent context in which the team mentioned the DCs. More specifically, the DCs were mentioned in the context of designing queries and prompts, menus, the "user interface", the control structure, the code, and error handling.

Early on the team mentioned designing *queries and prompts* in conjunction with a *user orientation* towards eliciting necessary details from a user during system operation:

> - ...you have to set up queries, or question answer things, whether its menus or

whatever. Somehow, the system has to egg-on the user, know what I mean? Lead them into what the system means.
- How the system will prompt you into getting to what you want.
- That's the whole idea behind the Dialog Charts. *(C.3)*

This context of *user orientation* surfaced elsewhere in the interview, particularly in reference to the *design generation* of the dialog. In response to the question "how did you go about specifying and designing your interface?" the team answered,

- How you would most feel comfortable if you would put yourself in the user role.
- That basically came out of the Dialog Charts too... *(C.28)*

One recurring theme in the interview was the surprise expressed at the usefulness of the DCs. It surfaced while discussing *menu* design, in the logical design phase of the interview: "We really did use them [the DCs] as far as designing menus" *(C.12)* and "The menus really came out of that [the DCs]." *(C.25)*

It is interesting to note that the team extended the DC design vocabulary to designing *error processing*. One mention in particular described how the team had integrated error processing with general control structure design:

- Because you have to diagram where things are comming in from and going through. And you can see right where to check, and where to direct the flow of control to.
- If it was wrong, you can direct it back to one particular point. I mean if you have something, wrong, some wrong input, and you direct it to the wrong place back, you may--
- you may crash the whole system.
- or else you may change something that--
- should we go back to the very beginning menu? Or should we go back to where they got to that point, where they made the mistake. *(C.33)*

As indicated in Section 2 above, the DCs avoid cluttering the description of the dialog with its entire collection of alternative paths of error handling. The team's response to a question related to the issue of error routines is therefore somewhat surprising:

- ...I dont know whether it's supposed to include it or not, but we included the error, because-- well I dont know, to be quite honest with you, but we got very familiar with it [dialog charting, and, you know, we just took to it, you know. I mean we just thought that it was just a logical extension of it. *(C.43)*]

Apparently the team felt free to change the tool to suit their purposes, an interesting statement indicative of the team's familiarity and comfort with the DCs.

Finally, the team mentioned using the DCs, among other design products, in *coding* their system:

> - ...We took our Dialog Charts, and our files, and our menus that we designed, and we... actually started to code them. Coded the record layouts, coded the file description statements.
> - Set up the user interface.
> - Coded the menus, yes. *(C.27)*

The mentions of the DCs in the context of *design evaluation* seem to re-emphasize the familiarity that the team found with the charts, because they could use them in a flexible fashion, and even comment on suggested improvements to the design process. They mentioned using Dialog Charting in an iterative fashion to evaluate their designs for completeness:

> - ...What we found was the Dialog Charts really needed to be an interactive process. Because as you go through them and through them and through them--
> - You realize things that you havent thought about before, or different ways. *(C.20)*

The team also commented that DCs were used to re-evaluate their menu hierarchy. First, they asked when the user makes an error, "...Should we go back to the very beginning menu? Or should we go back to where they got to that point -- where they made the mistake?" *(C.33)*. Their response was, "...we had originally gone back to the original menu, and then decided that that's boring" *(C.34)*. They were apparently satisfied with their restructuring, because as the mention continues, they note that the control structure of the system had become "more flexible" and "efficient".

The DCs were often mentioned in the context of *communication* from task to task. The DCs were used to derive menus, as input to the coding phase, and in determining how to prompt the user. Succinctly: "They really gave us a basis for so many of the next steps." *(C.37)*. As one member commented in a voluntary mention, when there was a question of the value of the different design tools (i.e., dataflow diagrams, flow charts and dialog charts), the DCs "seemed to

be the most helpful, though, because when we did get into the later stages, we did actually use them. Much to my surprise." *(C.10)*. Similar comments were repeated later in the conversation.

Interestingly, one team member, while indicating his intention to use the DCs in the future, focused on using them to communicate with users and in system maintenance:

> -... In terms of helping them maintain their system, I do keep in the back of my mind the Dialog Charts, which I thought were great. In terms of helping explain myself to them, what ideas I had. Whereas before maybe it was just kind of haphazard. Now I have some structure for explaining, and why I'm thinking what I'm thinking. *(C.48)*

Stage: References in that category were particularly scant. Nevertheless, three of the four mentions related to communicating information among the various system development stages; for example, after the team commented on realizing the value of the DCs, they were asked "What was the value?" and the responses were "Just for the later stage, and actual physical design." *(C.22)*, and "It helped in the implementation. How we were going to prompt the user." *(C.23)*. A third mention related to communicating to system maintainers *(C.48)*, and the last indicated confidence in the coding stage *(C.36)*.

Product: Mentions in this category link the use of the DCs to the structure of the resulting system architecture and dialog structure. The DCs were described by the team as "like a sketch of coding" *(C.4)*, "a beefed up data flow diagram with the user in it" *(C.7)*, and "a map of the system" *(C.24)*. The team made an interesting comparative comment:

> - ...we never used those data flow diagrams because they were all disjointed.
> - You know. This [the DCs] is at least connected and you could see different levels... *(C.6)*.

Reference to the *Control Structure* was made in response to the probe: "What were these Dialog Charts that you mentioned?"

> - It's like diagrams of how the system should work. At what point you would intercept the user to get a response. And based on that response what would be the next step. *(C.5)*
> - You kinda see the flow of everything.
> - ...And try to get an idea of what information you did have to prompt the user for... *(C.6)*.

> - ...you saw all the levels of depth. You saw all things that you would really have to do and ask for to perform the functions that you proposed. *(C.31).*

The DC vocabulary is not intended to be used as a language for modeling *data structure* and architecture. Nevertheless, the following mention indicates that they helped in conceiving data structure as well as the general hierarchical and modular structure of the system:

> - [you saw] which information you needed to determine which file you had to access, what calculations needed to be done on the data. *(C.32)*

Process: References to how using Dialog Charts put *constraints* on the design noted how "You were forced to put all the ideas you had into something concrete" *(C.17)*, and how the charts helped the team "to keep a very strong control over what was going on." *(C.35)*. Comments about the direction and *philosophy of design* were made in mention *(C.30)*, where the DCs were brought up in the context of the functional decomposition of the system until "you hit every possible situation", and also that it is decomposed according to party *(C.29)*. Recapping mention *(C.2)* in design evaluation, the team put forth the idea that the specification of the DCs really needed to be an *Iterative process*.

Attitudinal Patterns: The main theme that cut through the mentions in this category is that the DCs were found to be surprisingly valuable. It is interesting to note that the value was not discovered until the charts were used during stages subsequent to the conceptual/logical design. The team mentioned that the DCs allowed them to feel "confident in our code" and made them feel that the tasks of implementation went fast. Specifically, all of the *Learning* mentions point out that the usefulness of the DCs was not apparent to the team until the later stages of the development process, where they were actually used, e.g.,

> - I think we would have concentrated more on getting those right the first time, instead of going back and having to re-do them, not knowing the value of them the first time... cause we did, we went back and did them, like twice. *(C.16).*

Three more mentions express the view in a similar fashion. It looks as though the team experienced the value of the DCs when they learned that the tool would concretely guide them in building their system.

Re-doing Dialog Charts and some record descriptions was credited with positive *Task performance* in the following mention:

> - ...after we had gone back the physical worked out very well.
> - Very well, see how fast it went though.
> - Yea, but if we hadn't gone back we would have been stuck
> - I think we would have really trudged through that one, so it paid off.
> - Yea, that's for sure".
> *probe:* and what did you redo again?
> - The Dialog charts. That was the main one. And some record descriptions. *(C.44).*

The team related to the ease in which menu design is derived from dialog charts *(C.26),* and also mentioned the DCs in the context of *Task clarity and comprehensibility.* For example:

> - Because, you saw all the levels of depth. You saw all the things that you would really have to do and ask for to perform the functions that you proposed. *(C.31).*
> - The main thing is that it helped us to-- see the control. *(C.35).*

Clearly, the team members derived *Subjective satisfaction* from using the Dialog Charts. It was expressed in the intent to use the Dialog Charts in the future, in their happy surprise at their usefulness, and in the perception of the DCs as valuable.

The idea that the DCs proved to be useful surprised and pleased the team, and they mentioned it four times. One mention is interesting in particular:

> - probably the best way to show the contrast is that in the beginning, like when you first starting programming, they made you do flow charts. and you were supposed to do a flow charts before you programmed, and most people programmed and then drew the flow charts afterwards....
> - So, I mean, this was totally the opposite.
> - ...thats why it's so surprising. For once, we actually used it further on. *(C.41).*

They were also surprised that the Dialog Charts functioned as a mapping tool for system structure: "And it really is a true map, which is--surprising." *(C.24).* The *value and importance* of the dialog charts were mentioned three times, twice in connection with the learning process. For example:

> - But I think, when we went to the next step, we realized how valuable they were.

- Right.
- And then we redid them. *(C.17).*

Relationships Among Categories

Generally speaking, a *link* is some co-occurrence of categories within a mention. As indicated earlier, mentions were categorized with the minimal number of categories, but in some cases more than one category adequately keyed the mention. Figure 4-4 summarizes the co-occurrence of categories in coded mentions. In the following paragraphs we briefly comment on some interesting double-coded mentions in the current set of data.

	purpose	stage	product	process	attitude
purpose	25,27 28	22,23 48	5,33 3	20,29 30,30	10,17 26,38
stage	22,23 48	0	0	0	36
product	5,33 3	0	0	14	24,31 49
process	20,29 30,30	0	14	0	35
attitude	10,17 26,38	36	24,31 49	35	16,18 21,44

Figure 4-4: Tally of Multiple-Coded Mentions, by Category

Purpose of using the DCs linked to **Stage** with respect to communicating information among stages of design. In particular, one mention indicates that the DCs "helped in the implementation" *(C.23).* Another mention linked **Product** to **Purpose** in reference to the results of designing the control structure: "It's like a diagram of how the system should work" *(C.5).* Yet another evidenced a user orientation while designing the queries: "...you have to know the kind of user you're dealing with and formulate those queries accordingly." *(C.3).*

Co-mentions of **purpose** with **attitudinal patterns** occurred 4 times, which interestingly centered on communication. Two such multiple-coded mentions indicate surprise because the DCs were helpful or useful in later stages of system implementation *(C.10)* and *(C.38)*. One used the term "valuable" about the role of DCs in "so many" following steps. A fourth mention related DCs to the ease of menu design *(C.26)*.

Product linked to **process** in a mention that expressed the constraint that the DCs forced them to put their ideas into "something concrete" *(C.14)*. It also linked to **attitudinal patterns** in three mentions. In *(C.24)*, the team expressed surprise by the idea that the DCs are a "true map" of the system, and *(C.31)* relates similarly to clarity of the structure and functions. User orientation in designing the product is expressed in *(C.49)*, along with the intention to use DCs in the future: "No doubt about that." *(C.49)*. One **Process** mention linked with **attitudinal patterns**. The DCs "helped us to see the control" as well as "to keep a strong control over what was going on" *(C.35)*. Four **Attitudinal patterns** mentions linked to other aspects in that category. All four are *learning* mentions, three of which are linked to the *value and importance* subcategory, and the fourth mention was linked with *task performance*.

By now the richness of the data gleaned from this single team's experience is apparent. What do all these observations really mean? In the following discussion section we attempt to interpret our findings and relate them to issues currently on the evolving agenda of conceptual dialog modeling.

5. Discussion: Conceptual Dialog Modeling in Perspective

The sections above examine -- from a number of complementary perspectives -- a methodology for the conceptual design of dialogs. There is actually a dearth of such studies not only in the domain of dialog design, but in the entire field of information system design. Without a clear guiding tradition, this section focuses on three questions, namely whether conceptual modeling of dialogs is at all relevant, whether the pervasive notion of direct manipulation defies conceptual modeling and therefore undermines its validity, and lastly how do

the DCs contrast with another contemporary approach to conceptual dialog modeling. Through this discussion the examination of the DCs is related to contemporary concerns in dialog design.

5.1. The Relevance of Conceptual dialog models

Is the conceptual structure of interaction specifiable? Some argue that there is no meaningful way to abstract an interaction, and that any attempt to strip it of application or implementation detail renders such description worthless. The question has not been dealt with directly so far; in this section the case for conceptual modeling is informally reviewed. Key sources for observations are the recent debate around User Interface Management Systems (UIMS) and our own empirical study reported above.

User interface implementation is concerned with syntactic and lexical levels of design, command names, screen and icon design, menu organization, sequencing rules, and interaction techniques. It is being increasingly recognized that UIMS's postulate is that the syntactical aspects of the dialog can be **extracted** from its ultimate realization, thereby separating the user interface from the application's functionality [41]. Such a separation, so claim UIMS supporters, promises a clearer and more modular system architecture, as well as a more "consistent" interface. Conceptual design of dialogs in general, and the DCs in particular, share the same premise, but treat the **structure** of the interaction as a prime target of an independent design effort. Conceptual dialog design, therefore, refines the UIMS postulate by separating dialog structure from any of the application details as well as the syntactical aspects.

The conviction that interaction style design can be meaningfully discussed independently of the specific application context is widely held among user interface designers. Nevertheless, it is being recognized more and more clearly that the user interface design problem and the application design problems evolve together from the initial task analysis all the way to the implementation, and an effective design tool has to address them in an integrated way. UIMS's emphasis on the user-visible, syntactical issues instead of broader concerns for system or dialog functionality has been criticized recently. Jim Miller in [41] identifies the following difficulties as the "real bottlenecks" in the areas of interface design and development:

1. The portrayal of the application's semantics in a way that allows users to carry out their tasks.
2. Support for the activities of *design*. "If the role of interfaces is to help users understand and work with the semantics of a task domain, we need tools that will let interface designers represent these domains and make their important properties explicit in the interface." (p.199).

In its core, the conceptual modeling of dialogs directly addresses these concerns.

The above comments are somewhat theoretically motivated, but does conceptual modeling actually work? Although it relates to a single team only and is necessarily preliminary, the empirical portion of this study does support the case of conceptual design of dialogs. Specifically, designers have addressed, in their reference to the DCs, the fundamental attributes of conceptual models and their use.

Conspicuously, a frequent -- and unsolicited -- reference was made by the team to the DCs in the context of communicating between the logical and the physical stages. This idea relates directly to the essential role of conceptual modeling, namely guiding the design by establishing the conceptual framework within which the dialog is to be implemented. Furthermore, the general recognition of the value of the DCs was tied to using the tool as a vehicle for learning -- "going back and modifying" -- and "growing" a system description and specification. The team's reference to the use of the DCs as a tool for evaluation is also interesting, since evaluation per se was not part of the project, and was therefore completely motivated by the team itself. The team also indicated a number of times that the DCs brought in the **users** as a focus of the modeling process, making them an un-ignorable part of the deliberation. There is, as already noted in Section 1 of this paper, wide agreement that the support for the above activities is the ultimate purposes of conceptual dialog modeling.

Modeling in general, and conceptual modeling of computer based implementations in particular, are typically "disturbing" in the sense that constructing such models involves a series of decisions about what details to neglect. Dealing with abstraction can easily create dissatisfaction and frustration. In this light designers' emotional responses to the DCs are very

relevant and rather interesting -- seventeen mentions reflected various forms of subjective satisfaction with the DCs. The team, members of which took part in a number of system development efforts before, expected the DCs to be "ritualistic" like other conceptual design tools. In fact, they expressed surprise that the DCs were actually advantageous and valuable, and that they actually used them during later stages of design.

In light of the earlier discussion of UIMS, it is interesting that another frequent comment of the team was with respect to the relationship between the implementation and conceptual designs. It occurred to the team that the DCs capture the essence of the menu in a convenient fashion -- captured in three unsolicited mentions. In fact, two are categorized under *design generation* in the logical stage.

One area where conceptual modeling has become the norm is database design -- sound database design begins with a logical data model. Experiences that can fuel similar development in the area of dialog design are still under-reported. An interesting exception: according to its developer, Fredrik Brooks, IBM System/360 job control interface (the notoriously unfriendly JCL) has assumed such a bad reputation because it lacked a clear conceptual basis [5]. Although the analogy between database design and dialog design is appealing, its limits should be explored -- how literally can it be taken? For example, data modeling focuses on capturing the underlying static structure and typically fails to portray dynamic constraints and relationships, while dialog modeling must address the dynamic essence of the dialog process. Are these differences between data modeling and dialog modeling significant?

The idea of separating logical, implementation and physical models, which has become a cornerstone of data modeling seems to be directly relevant to dialog modeling, probably due to the fact that both apply to the process of system development in computerized environments. Could a more dialog-idiosyncratic yet generic model, like the Seeheim Model [38] (essentially Figure 1-1), be useful in dialog design? The following section is dedicated to the discussion of this matter.

5.2. The Direct Manipulation Benchmark

The Seeheim model (Figure 1-1), which underlies both UIMS and conceptual modeling, has recently become a target for critical discussion (e.g., in [41]). It has been argued that the partitioning of the dialog task suggested by this view raises more problems than it solves in dealing with direct manipulation interfaces, concurrent processing and multiple input output channels. Since direct manipulation is probably the most pervasive concept in the contemporary scene of dialog design, it serves here as a benchmark. In this section we examine the "threat" that direct manipulation presents for conceptual modeling of dialogs in general and the DCs in particular.

By way of introduction it should be noted that the notion of "conceptual" can be interpreted in a number of different ways. One possible interpretation is that it is an abstract representation of the dialog which corresponds to the central element in the Seeheim model. A conceptual model in that sense is then **realized** in a particular syntactical form. A different interpretation is in line with the multi-layered modeling approach commonly used in database design procedures [46]. According to this interpretation the conceptual or logical model captures the structure of the dialog as as close as possible to the user's functional view of it. Such a conceptual model is then **translated** into a more detailed and more formal implementation model, which is translated in turn into a concrete physical model. These two interpretations are closely related and commonly used as interchangeable, but there are also subtle differences between them.

Direct manipulation [44] is a principle of dialog design and it cannot be interpreted in a narrow sense in either of the above approaches. As far as the Seeheim model is concerned, direct manipulation addresses all three aspects of dialog design, namely it suggests direct gestures, it defines the range of actions available, and it defines the relationship between the visible availability of dialog "items" and the applications that they could be subjected to. From the database-inspired perspective direct manipulation conceptually fosters a dialog which centers around manipulable objects and identifies what can happen to these objects. At the

implementation level it entails the formal identification of objects and the formal specification of procedures that can be applied to them, while in the physical level it implies the design of mechanisms for depicting and picking visible objects.

The role of the DCs in the database-inspired interpretation is somewhat easier to understand. From a more theoretical perspective, Section 3.2 demonstrated the clear correspondence -- as well as the apparent distinction -- between the DC-model and its related implementation models. Moreover, it should be noted that the examples in Section 3.2 are all dealing with direct manipulation interactions. The empirical observations, as reported in Section 4.2, clearly support that view. One amply supported observation is that the DCs indeed provide a means for effectively communicating information from one phase of the design process to the next -- a quintessential service of conceptual modeling.

The same empirical evidence could be equally construed as supporting the other, Seeheim-inspired role of conceptual modeling as the guidance for the actual syntactical realization of the interface. A directly relevant observation is the correspondence that designers found between the DC-based model and the menus, the actual syntax of interaction, that "came out" of it. However, the separation assumption in the core of the Seeheim model is in conflict with the reality of direct manipulation. Direct manipulation rests on "semantically rich" application dependent feedback, and may suggest therefore a very tight coupling between the syntactical and the application aspects of the dialog. Is it too tight for meaningful separation? Does direct manipulation unveil, therefore, the fundamental limitations of conceptual dialog modeling?

These somewhat more theoretical problems with direct manipulation persist only if the Seeheim model is interpreted too literally and too narrowly. The model can mislead in the sense that the user-visible interface is indeed inseparable from the application, and artificial decoupling of these two intimately coupled elements may hinder implementation, rather than help it. A clear linkage between the syntactical aspects and the application is needed, in terms of bridging application requirements and system's appearance. Stated from the vantage point of

implementation, for a UIMS to be able to handle the dialog's syntactical aspects, it necessarily depends on the "flow of semantics" from the application. Actually the Seeheim model can be looked upon from a different perspective, namely that it explicitly **connects** the application and the user-visible interface, but suggests that for design purposes (e.g., a "cleaner" design), they should be discussed independently as part of the design agenda, but not in an unrelated fashion. Figure 5-1 depicts this perception. There is a rather compelling argument for the importance of the dialog structure (i.e., the "conceptual model") as the intermediate entity.

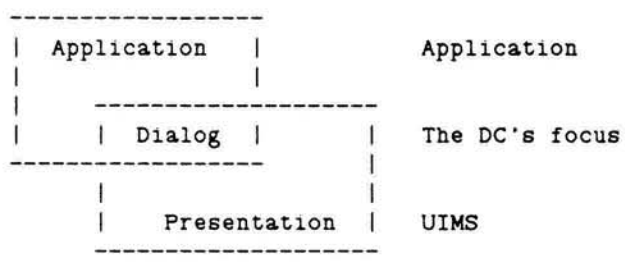


Figure 5-1: An Alternative Depiction of the Seeheim Model

The essence of this argument is that the Seeheim model is a **design framework**, and the actual construction of the dialog may call for differing levels of integration or decoupling. It provides a relatively clear and valid agenda for thought about the problem of dialog design, and as such it entails some partitioning of the overall problem into manageable sub-problems. There should be a clear distinction between **design** -- the clear identification of the environment of the system, its role and its *functional* components, and **construction** or implementation -- resource allocation and the arrangement of working modules that together fulfill the role and functional requirements set forth during the design. Such a view identifies very clearly the role of conceptual modeling as a design discipline which focuses designers' attention on semantic and functional aspects of the interactive system under consideration.

5.3. Different Disciplines of Conceptual Modeling of Dialogs

Generally, there seem to be three typical approaches for designing dialog structures, specifically those which focus on a "proper" design process, those which prescribe a "proper" set of dialog attributes, and finally those which provide tools for dialog modeling and analysis. *Procedural approaches* describe sequences of activities that dialog designers should follow. These approaches sometimes use formal or informal representations, but the emphasis is on how to approach the design and on how to decompose the task (e.g., [9], and [29]). *Guidelines sets* are loose collections of principles, policies and rules to be used in dialog design (e.g., [49], [13], [32], [33], [40] and [42]). A "guideline" advises about the proper conduct for the dialog; for instance, "Control should always remain with the user." *Analytic methods* employ an abstract and somewhat formal representations of the interaction, along with rules for manipulating these representations. The Dialog Charts and the User Interface Design Environment (UIDE) [12] belong primarily to this category.

The UIDE has been presented as a candidate approach for the conceptual design of user interfaces. Its intended use -- and for that matter of any conceptual model -- is to represent conceptual design, to provide an abstract representation of dialogs (i.e., be a basis for a *set* of functionally equivalent interface implementations), provide a specification for a UIMS, a means to ascertain correctness and completeness, a means to evaluate the design with respect to speed of use and ease of learning, and provide run time help to the user [12].

UIDE is the conceptual analog for the event model of interaction discussed in Section 3.2 above. Its basic approach is to decompose the description/specification of the dialog into autonomous but linked frames which describe:

1. Objects and properties,
2. Actions which can be performed on these objects,
3. Information required by these actions, and
4. Pre- and post- conditions for the actions.

The overall approach is of formal specification, actually formal enough to be eventually accessible to a UIMS.

UIDE includes all information and context needed to carry out operations, as contrasted with lower level specification syntaxes (such as BNF and ATN), where such knowledge cannot be explicitly represented, and therefore cannot be used in interface design and implementation. Specifically, pre- and post conditions schemas are UIDE's means for capturing application semantics and represent the *evolution of context* in system's use. Such a specification is the basis for context management, context sensitivity in menu presentation and help functions. UIDE decomposes the essence of the dialog flow into paired sets of pre and post conditions of actions.

UIDE and DCs represent different approaches to the conceptual design of dialogs. UIDE is analytic -- it focuses on the details, and its general design philosophy is "bottom-up." The DCs are synthetic -- they focus on "holistic" description, following a "top-down" design process. The two approaches maintain the notion of transformation, or the manipulation of dialog representations. More formally, a *transformation* is a gradual or marginal modification of a consistent set of schemas into another consistent set of schemas. Specialization and generalization transformations correspond to the refinement of DC "boxes" into their underlying elements and regrouping DC elements into an aggregate dialog element, respectively.

In the current state of conceptual dialog modeling a more rigorous comparative assessment of these two conceptual design schemes is rather difficult. If one applies superficially the criteria of descriptive and usable power as a tentative agenda for evaluation, it seems that the two approaches are compatible with respect to their descriptive power, while the DCs seem to provide more usable power, i.e., how *convenient* is UIDE for **end**-users specification or inspection of dialog design? Further study and research is obviously called for, and some pertinent questions are therefore highlighted in the following concluding section.

6. Conclusion and Further Research

The Dialog Charts yield high-level dialog description that is abstract enough to be useful for more than one implementation technique or strategy. The DCs also combine two types of decompositions in the same hierarchy, namely a functional decomposition, which is a common

design practice, and a decomposition of parties, which is a distinct dialog modeling requirement. They model the functional requirements of the system, capture the sequencing and control of the interaction, while clearly differentiating between user gestures (i.e., the inputs) and system responses (i.e., the outputs). The DCs are simple and complement other design tools like DFDs, ERM in their respective areas of applicability.

The examination in this paper is comprehensive but rudimentary. Unfortunately, there is no well-defined, validated theory to guide the evaluation of the various methodologies and tool vocabularies that are used for designing conceptual dialog models. Further research questions are clearly raised by this paper, and few key issues are highlighted in these concluding comments.

First and foremost are fundamental questions with respect to the assessment of conceptual design tools. The links in the design chain that are addressed by this paper are delineated in Figure 6-1. In Section 3 the notion of descriptive power has been mainly applied to the outcome of the conceptual design, and the DCs were assessed viz. their implementation counterparts. In Section 4 the complementary notion of usable power was applied mainly in the context of the ease of translating user requirements into a conceptual dialog model. Ideally, both criteria should have been applied at both ends of the conceptual design activity. Nevertheless, assessing the descriptive power of a conceptual design tool with respect to the universe of user and application requirement is a perplexing task -- it is premised on the existence of some "complete" classification of user and application requirement. Usable power, representing a proper subset of the descriptive power, is therefore a close surrogate measure of quality.

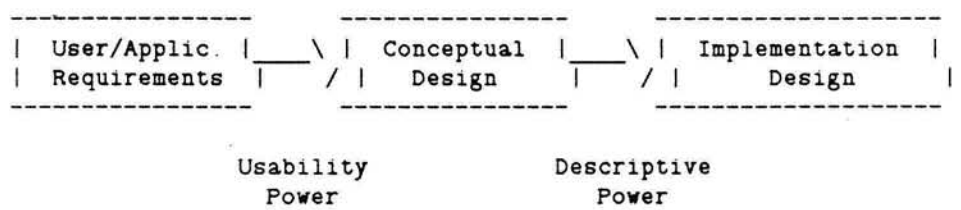


Figure 6-1: Process Environment of Conceptual Dialog Modeling

As far as the step of moving from a conceptual design to implementation design, one can argue that given the somewhat mechanical nature of the transformation, assessment of the descriptive power at that juncture is conceivably straight forward. Although usable power is bounded by the descriptive power, discovering which types and forms of conceptual models translate *easily* into implementation models is an interesting question, which calls for further empirical inquiry.

A related topic is the relationship between *usefulness* and *usability* of a conceptual design tool in general and the DCs in particular. A study of a single situation addresses usability, especially in it somewhat more formal sense, in a rather limited fashion. Not all situations are "amenable" to the DCs, so the essence of the question is in ascertaining the limits of the tool's applicability, e.g., what type of design situations are more "prone" to dialog charting, or which range of applications calls for DC use. It has been commented that "traditional" data processing systems are probably more amenable to the DCs technique. There is a need for a rigorous assessment of the relationship between the variety of tasks and contexts in which tools are used and the perceived usefulness of tools. Such an examination will allow the prediction of a tool's behavior in a particular design environment, and also allow the designer to select appropriate design situations for using the tool. The empirical part of this study is currently being repeated with more teams [6]. Ultimately, it is going to address the concept of usable power more directly. Upon analysis of more and varied cases it will become clearer in which situations the DCs are perceived as most valuable.

Although the target tool in this study is the Dialog Charts, the research is an in-depth study of the dialog design process. How do people go about dialog design? What are the requirements for designing dialog structure and control processor components? Ultimately these insights will form the basis for a set of assessment criteria to guide the development and evaluation of dialog design methodologies, and the development of sounder and more robust human/computer interaction.

References

[1] Ariav, G. and Ginzberg, M.
DSS Design: A Systemic View of Decision Support.
Communications of the ACM 28:1045--1052", 1985.

[2] Ariav, G. and Calloway, L.
A Normative Analysis of Approaches to the Conceptual Modeling of Human/Computer Dialogs.
Technical Report, New York University, N.Y., 1988.

[3] Benbasat, Izak and Wand, Yair.
A structured approach to designing human-computer dialogues.
International Journal of Man-Machine Studies 21:105--126, 1984.

[4] Britton, K.H., Parker, R.A., and Parnas, D.L.
A procedure for designing abstract interfaces for device interface modules.
Proc. 5th Int. Conference Software Engineering , 1981.

[5] Brooks, Jr., Frederick P. .
Grasping Reality Through Illusion: Interactive graphics serving science.
In *Conference on Human Factors in Computing Systems*, pages 1--11. Washington, D.C., May, 1988.

[6] Calloway, L.
An Approach for Assessing Tools for Designing Dialog Structures: A Study of the Dialog Charts.
PhD thesis, New York University, 1988.
(In progress).

[7] Campbell, D.T. and Stanley, J.C.
Experimental and Quasi-experimental designs for research.
Rand McNally, Chicago, 1963.

[8] P. P.-S. Chen.
A Preliminary Framework for Entity-Relationship Models.
ACM TODS 1, No. 1:, March, 1976.

[9] Cheriton, D.R.
Man-Machine Interface Design for Timesharing Systems.
In *Proceedings: Annual Conference of the Association for Computing Machinery*, pages 362-366. Houston, Texas, October 20-22, 1976.

[10] Date, C.J.
An Introduction to Database Systems, Volumn 1, 4ed.
Addison-Wesley, Reading, Massachusetts, 1986.

[11] Davis, Fred D. and Olson, Judith Reitman.
Integrating User Motivation and Task Performance Theories of Information Systems Design.
Technical Report, The University of Michegan, April, 1986.
Private correspondence.

[12] Foley, James, Gibbs, Christina, Kim, Won Chul, Kovacevic, Srdjan.
A Knowledge-Based User Interface Management System.
In *Chi'88 Conference Proceedings: Human Factors in Computing Systems*, pages 67--72. Washington, D.C., May 15-19, 1988.

[13] Gaines, B.R. and Facey, P.V.
Some experience in interactive system development and application.
In *Proceedings of the IEEE Vol63(6)*, pages 894-911. June, 1975.

[14] Gaines, Brian R. and Shaw, Mildred L.G.
From timesharing to the sixth generation: the development of human-computer interaction. Part II.
International Journal of Man-Machine Studies 24:101-123, 1986.

[15] Glaser, B. and Strauss, A.L.
The Discovery of Grounded Theory: Strategies for Qualitative Research.
Aldine, Chicago, 1967.

[16] Green, M.
A Survey of Three Dialogue Models.
ACM Transactions on Graphics 5, No. 3:244-275, 1986.

[17] Dunnette, Marvin D. (editor).
Handbook of Industrial and Organizational Psychology.
Rand McNally, 1976.

[18] Hartson,H.R., Johnson, D., and Ehrick, R.W.
A Human-Computer Dialogue Management System.
In *Proceedings of INTERACT '84, First IFIP Conference on Human-Computer Interaction*. september, 1984.

[19] Hayes, P.J., Szekely, P.A., Lerner, R.A.
Design Alternatives for User Interface Management Systems Based on Experience with Cousin.
In *CHI '85 Proceedings*. ACM, April, 1985.

[20] Hix, D. and Hartson, H.R.
An Interactive Environment for Dialogue Development: Its design, use and evaluation-or-Is AIDE useful?
In *CHI '86 Proceedings*. ACM, April, 1986.

[21] Jacob, R.J.K.
Survey and Examples of Specification Techniques for User-Computer Interfaces.
Technical Report NRL Report 8948, Naval Research Laboratory, Washington, D.C., April, 1986.

[22] Jacob, R.J.K.
A Specification Language for Direct-Manipulation User Interfaces.
ACM Transactions on Graphics 5, No. 4:283-317, 1986.

[23] Jensen, K. and Wirth, N.
Pascal User Manual and Report, Second Edition.
Springer-Verlag, New York, 1978.

[24] Kelley, John F.
Videotex Information Packagers: A field study aimed at tomorrow's videotex authoring interface.
SIGCHI Bulletin 19.3:37--47, 1988.

[25] Krippendorff, K.
Content Analysis: An Introduction to its Methodology.
Sage, Beverly Hills, CA., 1980.

[26] Malhotra, A., Thomas, J.C., Carroll, J.M. and Miller, L.A.
Cognitive Processes in Design.
International Journal of Man-Machine Studies 12 no 2:119-140, February, 1980.

[27] Martin, Patricia Yancey and Truner, Barry A.
Grounded Theory and Organizational Research.
The Journal of Applied Behavioral Science 22, No. 2:141--157, 1986.

[28] McGrath, J., Martin, Joanne, Kulka, Richard A.
Judgement Calls in Research.
Sage, Beverly Hills, 1982.

[29] Mehlmann, M.
When People Use computers: An Approach to Developing an Interface.
Prentice Hall, Inc., Englewood Cliffs, N.J., 1981.

[30] Miles, Matthew B. and Huberman, A. Michael.
Qualitative Data Analysis, A sourcebook of new methods.
Sage, Beverly Hills, CA., 1984.

[31] Moran, T.P.
The Command Language Grammar: A Representation for the User Interface of Interactive Computer Systems.
International Journal of Man-Machine Studies 15:pages 3--50, 1981.

[32] Morse, A.
Some Principles For the Effective Display of Data.
Computer Graphics 13(2):94-101, August, 1979.

[33] Nickerson, R.S.
Why interactive computer systems are sometimes not used by people who might benefit from them.
International Journal of Man-Machine Studies 15:469-483, 1981.

[34] Norman, D.A.
Design principles for human-computer interfaces.
In *CHI '83 Proceedings*, pages 28-34. ACM, December, 1983.

[35] Olsen, D.R., Jr.
Presentational, Syntactic and Semantic Componenets of Interactive Dialogue Specifications.
User Interface Management Systems.
Springer-Verlag, Germany, 1985, pages 125--136.

[36] Olsen, D.R., Jr.
Whither (or wither) UIMS?
In *CHI '87 Proceedings*, pages 311-314. ACM, April, 1987.

[37] Parnas, D.L.
On the use of transition diagrams in the design of user interface for a interactive computer system.
In *Proceedings of the 24th National ACM Conference*, pages 379-385. ACM, New York, 1969.

[38] Pfaff, G.E., Ed.
User Interface Management Systems.
Sprinter-Verlag: Berlin, 1985.

[39] Reisner, P.
Formal Grammar and Human Factors Design of an Interactive Graphics System.
IEEE transactions on Software Engineering SE-7, No. 2:229-240, March, 1981.

[40] Reitman, J.O.
Expanded Design procedures for learnable, usable interfaces.
In *CHI '85 Proceedings*. ACM, San Francisco, April, 1983.

[41] Rosenberg, Jarrett, Hill, R., Miller, J., Shewmake, D.
UIMSs: Threat or Menace? (Panel).
In *Chi'88 Conference Proceedings: Human Factors in Computing Systems*, pages 197. Washington, D.C., May 15-19, 1988.

[42] Shneiderman, B.
Software Psychology: Human factors in computer and information systems.
Little Brown and Co., Boston, MA., 1980.

[43] Shneiderman, B.
Multiparty Grammars and Related Features for Defining Interactive Systems.
IEEE transactions on Systems: Man and Cybernetics smc-12, no 2:148-154, March-April, 1982.

[44] Shneiderman, B.
The Future of Interactive Systems and the Emergence of Direct Manipulation.
In Vassiliou, Y. (editor), *Human Factors and Interactive Computer Systems*, pages 1-27. Ablex Publishing Co., Norrwood, New Jersey, 1984.

[45] Spradley, James P.
The Ethnographic Interview.
Hold, Rinehart and Winston, 1979.

[46] Teorey, T.J., and Fry, J.P.
The logical record access approach to database design.
ACM Computing Surveys 12, 1980.

[47] Wasserman, A.I. and Shewmake, D.T.
Rapid Prototyping of Interactive Information Systems.
Technical Report, Medical Information Science: UC, San Francisco, 1982.

[48] Wasserman, A.J., Pircher, P.A., Shewmake, D.T., and Kersten, L.
Developing Interactive Information Systems with the User Software Engineering Methodology.
IEEE Transactions on Software Engieering se-12 No. 2, February, 1986.

[49] Williges, B.H. and Williges, R.C.
Dialog design considerations for interactive computer systems.
Human Factors Review: 1984.
Human Factors Society, Santa Monica, California, 1984.