An Approach to Structured Display Design - Coping with Conceptual Complexity

Morten Borup Harning

Informatics and Management Accounting, Copenhagen Business School, Howitzvej 60, DK-2000 Frederiksberg, Denmark
Phone: +45-3815-2431 – Fax: +45-3815-2401
E-mail: harning@cbs.dk
WWW: http://www.econ.cbs.dk/people/harning/

Abstract

The methods that provide a structured approach to user interface design, often more or less ignores the aspects of display design. The structured display design approaches that exist, seems to have problems coping with conceptually complex interfaces. Building on the relationship between the system data model and the display design, this article proposes a structured approach to display design. The design is divided into three steps: conceptual design, logical window design and physical window design. This structure seems to be a way of coping with the design of conceptually complex user interfaces.

Keywords

Display design, conceptual design, conceptual prototypes, user data model, visual data dictionary, logical window design, structured design method, user interface design.

Introduction

We know from software engineering that some sort of structured design method (e.g., Modern Structured Analysis [Yourdon89], OMT [Rumbaugh91] or OOSE [Jacobson92]) is needed in order to manage the design process involved in developing large and complex computer systems. The structured design methods make design and development of such systems manageable by dividing the process into smaller and more focused design tasks. The result is that the design team can focus on a smaller sets of design issues, one set of issues at the time. This is necessary to maintain a general view of the design, as well as to produce time plans and cost estimates.

Most software engineering methods do not deal with the user interface aspects of the design process. Those that do include such aspects treat them very superficially, e.g., Multiview [Avison90], Sommerville [Sommerville95]. There exists a number of

structured methods for designing user interfaces, e.g., Foley et. al. [Foley90], MUSE [Lim92, Lim94], Sutcliffe [Sutcliffe95]. These methods usually describe only the functional design, often with a focus on task analysis, whereas they only present a set of ideas to how the display design should be conducted, e.g. use of metaphors.

There has been presented a huge amount of guidelines concerned with display design (Nielsen [Nielsen94] lists 100 common usability heuristics, and tries to identify the most important ones). The general problem of using guidelines or usability heuristics as Nielsen calls them is that they do not help the designer structure the design process. This means that the guidelines are of little help, when the problem is coping with the design of large and complex systems. This is not to say that guidelines and design guides are useless, these are just not enough.

A few recent structured user interface design methods, such as EFDD [Lauesen-93], DIANE+ [Tarby94, Tarby96], TRIDENT [Bodart95a], OMT++ [Jaaksi95], addresses display design. EFDD and OMT++ have their offspring in software engineering, whereas DIANE+ and TRIDENT have evolved from the HCI tradition. However there still remain several problems. The most serious problem is that only EFDD addresses the issue of *early usability testing*. The other approaches do not facilitate usability testing until a prototype has been built. This is a major problem, because usability testing has turned out to be the most efficient way of identifying usability problems [Desurvire92]. Another major problem is that the methods do not seem to address the problems of designing displays for interfacing with large amounts of conceptually complex information.

In systems that need to support complex problem solving it is important that the users' mental model of the system matches the actual conceptual model of the system [Staggers93]. Norman [Norman86] suggests that users infer mental models of the device they work with. Interfaces that need to support this kind of complex problem solving, hence should help the user to infer an appropriate mental model.

Looking at user interfaces, the information modelled by the systems data model dominates the visual appearance of an application (e.g. as values shown in entry fields, tables or graphs), whereas task information such as procedures and operations only appear indirectly (e.g. through window transitions and greying of command buttons). Green and Benyon [Green92] among others describe how display designs can be interpreted as data models. Based on these observations the best way of supporting the inference of an appropriate mental model seems to be to focus on how the user interface reflects the structure and contents of the data model, because this is the most apparent part of the conceptual model in the interface.

This article describes an approach to display design that helps coping with conceptually complex interfaces. The design is based firmly on the data model to ensure a conceptually clear design, while task efficiency is ensured by taking in to account the information required to perform the tasks the system should support.

The complexity of existing user interfaces can be measured by constructing an Entity Relationship Model of the Information Artefacts (ERMIA) [Green92]. A con-

ceptually complex user interface is, in this article, a user interface that gives access to a data model with 10-20 entities or more, and where the user needs to see/work with information from several entities or several instances of entities simultaneously. The complexity of a user interface is, however, also influenced by a several other factors, e.g. the number of relationships in the data model.

1 Method

The display design method described in this article was developed as part of a full user interface design method. The full method called Entity Flow Dialogue Design (EFDD) was described in an early version by Lauesen and Harning [Lauesen93].

The revision of the method presented here is based on several sources of experience. One source of experience has been a series of professional design courses and tutorials based on EFDD offered to professional system designers. Three master level courses, where the participants spend five weeks in groups developing a user interface using EFDD, has been another important source of experience.

However, the primary source of experience stems from the design and implementation of a Classroom Scheduling System (CSS). The design and development time involved in this project was in the proximity of two man years. The system is today operational and is in daily use for scheduling and administrating the 150 classrooms at Copenhagen Business School. The design involved designing a graphical user interface according to the CUA'91 guidelines [CUA91].

When it was decided to decentralise part of the classroom administration, the system was redesigned and later implemented using Oracle SQL*Forms. The user interface in the final system was character based, because the system needed to run on the existing VT-220 terminal-based hardware platform, and had to comply with the interface style of the existing administrative applications used at the business school.

The CSS project made it possible to develop and mature the method in settings similar to the ones found in industry. The project has provided insight in the applicability of the method in large development projects. The high conceptual complexity of the system combined with the need for both simple form-based windows and windows visualising the large amounts of information made the project ideal for the development of a display design method.

2 The Full EFDD Method

The approach to display design presented here is, as stated earlier, part of a complete interface design method, called EFDD. The method divides the design process into the following four phases:

1. Compiling a task list and designing a user data model.
2. Designing logical windows and user functions.

3. Initial design of physical windows and dialogue state-transition diagrams.
4. Detailed physical design.

Iteration between the phases is assumed, as in any other structured design method, but the phases help define where the focus of attention should be in different stages of the design. The part of the method presented in this article is the design of a user data model (part of phase 1) and of the logical windows (part of phase 2), but also briefly the design of physical windows (part of phase 3 and 4). The issues discussed in this article have been added since the method was first described [Lauesen93, Lauesen94].

3 Coping with Complexity and Large Amounts of Information

There seems to be two main types of approaches to display design. The first type of approach is to derive the display design from task analysis, based on the information needed to perform the tasks, e.g., AUI [Kuo88], DIANE+ [Tarby94], MUSE [Lim94], TRIDENT [Bodart95a] and [Sutcliffe95]. Designing the displays to be included in the user interface involves identifying, for each step in the design procedure, which of the attributes in the data model needs to be accessed, e.g. by using Activity Chaining Graph (ACG) as proposed by [Bodart93]. As the number of tasks and the size of the data model grows, this becomes a very cumbersome process. Ensuring task efficiency using these approaches seems to be fairly straightforward, however maintaining a conceptually clear design becomes almost impossible.

The second type of approach is to use the data model or object model as the basis for the display design. Examples of this type of approach are GENIUS [Janssen93], Semantic Database Prototypes (SDP) [Baskerville93] and User data modelling [Lauesen93]. A conceptually clear design can easily be achieved using these approaches, even for large data models. Whereas it is more difficult to ensure task efficiency. If a task needs information from several entities, the user will have to jump between the corresponding windows to collect the information. One of the important advantages of this type of approach is that it is possible to design a prototype very quickly, and before the functional design has been done. This makes it possible to submit the design to usability testing, and hereby eliminating possible conceptual errors [Baskerville93, Lauesen93].

Jaaksi [Jaaksi95] presents in OMT++ an approach that try to balance the two design goals: (1) designing a conceptually clear display design; (2) ensuring task efficiency. But this approach deliberately tries to minimise the use of usability testing, and appears to have problems coping with conceptually complex interfaces, similar to the task driven approaches. The various works done on visualisation is a good example of how interface design can cope with high conceptual complexity. These visualisation techniques make it possible to work with amounts of data that would otherwise have been unmanageable. Tweedie [Tweedie95] describes a number of visualisations (Interactive Visualisation Artefacts), and shows how these can be in-

terpreted as depiction of a complex data model, showing otherwise "hidden" aspects of the data.

User data modelling as described in EFDD [Lauesen93] was designed to cope with the window design that was reasonably close to the data model. However, in the design of the classroom scheduling system the design had to include window designs like the visualisations described by Tweedie [Tweedie95]. This meant that the design had two parallel design goals: (1) designing a set of windows that could serve as the conceptual model of the application and (2) designing a set of windows tailored to the information demands of the tasks.

In an attempt to balance these design goals, five heuristics were introduced [Lauesen94]. However, these often contradictory goals resulted in a more unstructured design, because the designer had to focus on a set of contradictory design issues. This made it difficult to determine, when the design of the user data model was complete. In the design of the CSS the new heuristics resulted in frustration rather than creativity.

The solution was to divide the initial display design into two design steps: *a step focusing on the conceptual design* and *a step focusing on the design of logical windows*. The first step ensured that the design of the conceptual model was clear. The second step ensured task efficiency, and this step made it natural to consider the need for visualisation techniques, like the ones discussed by Tweedie. By letting the design of logical windows build on the conceptual design, it was possible to ensure that the interface remained conceptually clear. The following sections describe these two steps in detail, using a system for monitoring development activities in a software company, as the general example.

4 Conceptual Design

Data modelling, e.g. as described by Chen [Chen76], has within software engineering proven its worth as a tool for conceptual design. Data models following the Chen or similar notations (e.g., object-oriented notations), will in the following be referred to as technical data models. Figure 1 shows an example of a technical data model following the Chen-notation. The data model describes the concepts of a system for monitoring development activities.

There are a number of shortcomings in the technical data model if it is to form the basis for a structured design of the visual parts of the user interface. First of all the notation was not meant to and does not facilitate involvement of the users and hence any kind of usability testing of the conceptual design [Baskerville93]. This is a major problem, as mentioned in the introduction. However using design rules such as those used in GENIUS [Janssen93], Semantic database prototypes [Baskerville93] or TRIDENT [Bodart95c] it is possible to generate the first version of a user data model. This makes it possible to submit the conceptual design to usability testing.

To ensure a conceptually clear design, the user data model is designed according to following three heuristics:

1. Eliminate technical details.
2. Identify the information the user associates with a concept.
3. Identify the appropriate level of detail.

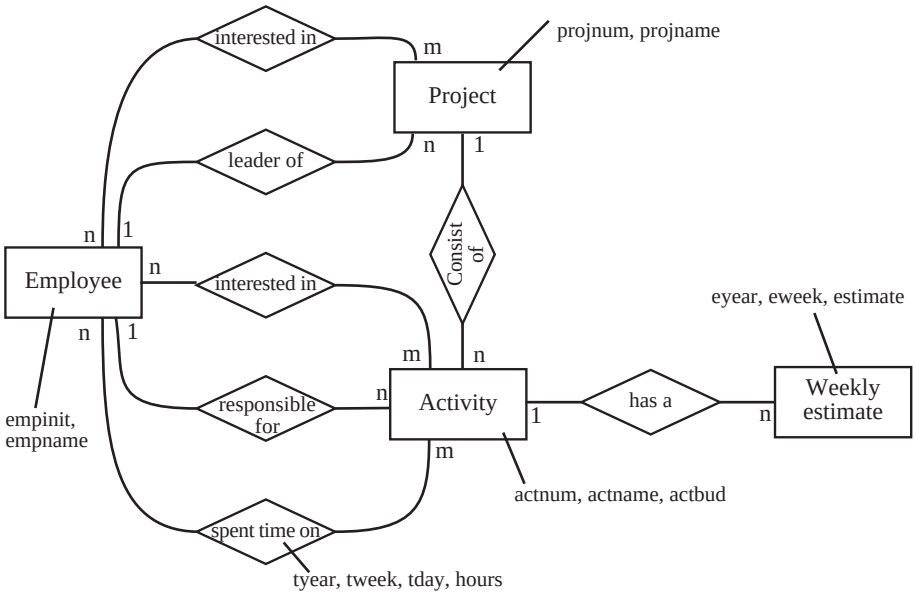

Figure 1. A data model describing the concepts of a system for monitoring development activities in a software company, using the Chen notation [Chen76]

4.1 Generating the First Version of the User Data Model

The general idea of producing a user data model is to transform the technical data model into a simple display design, by applying a simple set of design rules. The goal of the user data model is to form and evaluate the conceptual design, to be included in the user interface. Semantic database prototypes as proposed by Baskerville [Baskerville93] were introduced exactly for this purpose. A semantic database prototype was according to Baskerville [Baskerville93] an effective way of enabling a user driven evaluation of the conceptual design.

The semantic database prototypes are built from the information in the technical data model. Figure 2 shows two windows from the semantic database prototype corresponding to the technical data model shown in figure 1. The transformation is based on a set of simple design rules, guiding how the different parts of the data model should be represented in the prototype. A very simplistic description of the process is that all attributes in the data model are included in the prototype as data entry fields. The relationships (1:1) between entities are represented by including the identifying attribute (key field) of the related entity in the display design. The

key fields in figure 2 are marked by a button on the right side of the data entry field, this button made the prototype jump to the related entity. Other rules specify that 1:n relationships are represented in the display design as tables. A similar set of design rules can be found in GENIUS [Janssen93] and TRIDENT [Bodart95a].

This way of presenting the data model made it possible for users without prior knowledge of data modelling to relate the structure of the data model, and the way it models the domain concepts.

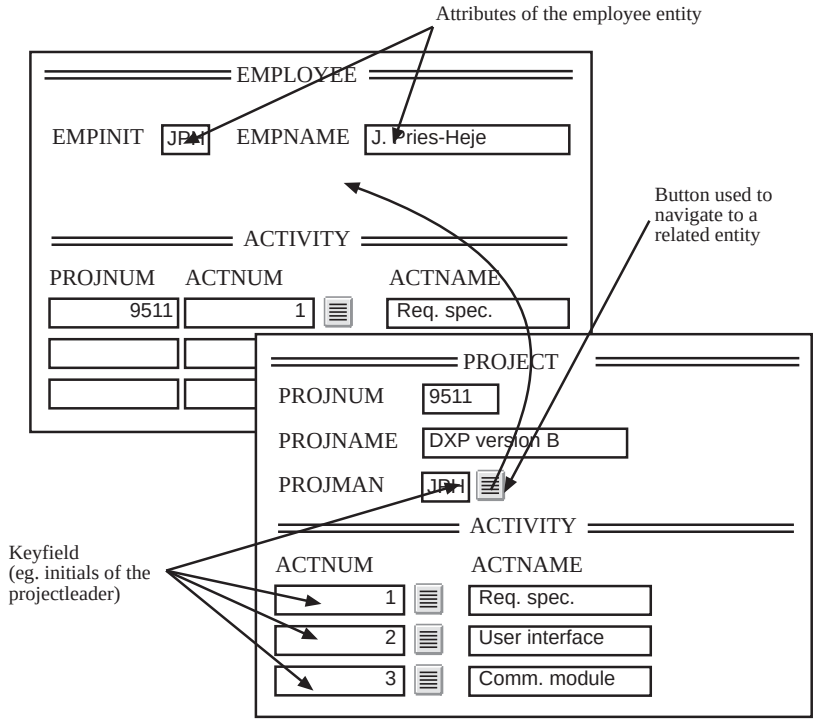

Figure 2. A semantic database prototype [Baskerville93] for the entities 'employee' and 'project' in figure 1

The user data model is built according to the same rules used in the semantic database prototype, but goes a step beyond that by applying the three heuristics mentioned above. The notation is a bit more relaxed, meaning that any information artefact [Green92] that reflects the attributes and relationships in the technical data model can be used. The only requirement is that the ERMIA of the display design should match the technical data model. This means that graphical representations can be used instead of tables when appropriate. In a drawing application the user data model would usually be purely graphical. The more relaxed notation used in the user data models facilitate including aspects that lack in order to make the conceptual design useful in the later, more windows oriented, part of the display design. The following subsections will describe how the heuristics mentioned in the beginning of this section can be applied to the conceptual design.

4.2 Heuristic 1: Eliminate Technical Details

The first problem is that the technical data model usually contains a number of technical details that are of no concern to the user, but are necessary in order to implement the technical algorithms. If the technical data model is to be used in the display design, it is important to identify the parts of the data model that concern the user. This is done in order to avoid including unnecessary technical details in the user interface design, that would only increase the possibility of misunderstandings.

An example of technical details, that should be avoided in the display, are internal record number used to implement the relationship between records in a relational database. Such record numbers are often used in the user interface, because entering these numbers makes the lookup of records easier. However using query languages like SQL there is no reason to burden the user with such information.

The user data model lets the designer determine through usability testing which parts of the technical data model that should be excluded from the user interface. The remaining part of the display design can then be focused on the relevant information, because the user data model serves as a visual data dictionary in the design of the logical and physical windows.

4.3 Heuristic 2: Identify the Information the User Associates with a Concept

The second problem is to find the appropriate level of normalisation. The technical data model will typically have been normalised to avoid technical problems like redundancy. But when used as a conceptual model, the normalised data model will seem unnatural to the user. The reason for this is, that when used for user interface design purposes, any redundancy in the conceptual model will indicate a relationship between the involved concepts.

E.g., when the name of an employee happens to be the same as the name of the project leader, the user assumes that this employee is in fact the project leader. This is a very simple example, but the principle seems to apply to other kinds of redundancy as well. This perception of relationships, based on the redundant information, is what make the semantic database prototype described above readable to the user. This may seem like a trivial aspect of the design.

However, in practise many systems exclude information, that is not directly used when performing a task. By excluding such information it becomes harder to understand the relationship between the information used in different tasks, hence making it difficult for the user to infer an appropriate mental model.

The design rules used to produce the user data model has been extended compared to the rules described in the design of a semantic database prototype. This is done in order to identify which parts of the data model the users associate with a con-

cept in the conceptual model. The design rules are basically the same, but are used recursively.

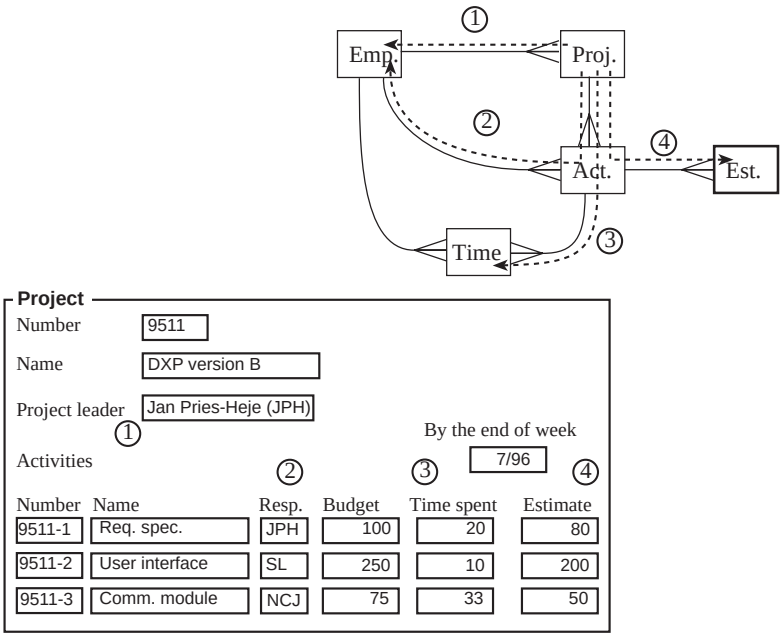

Figure 3. The project form in the ***user data model****, as it might look in the activity monitoring system*

Figure 3 show the project form in the user data model as it might look in the system for monitoring development activities. The form includes all of the information found in the semantic database prototype, as shown in figure 2. The identification of the project leader has been extended, to include the full name of the project leader, because the users perceived this as an attribute of the project. Secondly several columns have been added to the activity table.

The first new column is the person responsible for the activity. This information can be found by following the relationship between a project and one of the associated activities, and then the relationship from this activity to the employee responsible for the activity (this is shown by the dashed arrow in the ERD in figure 3). Using a similar process columns "Time spent" and "Estimate" was added.

However, because the activity can be related to several time records the column includes the *sum* of all related records. Instead of a sum, all of the individual values could have been shown graphically, because the ERMIA of this design would still match the technical data model.

Looking at the technical data model, it might seem unnatural to include, in the form describing the project, status information like "Time spent" and "Estimate" related to an activity (as shown in figure 3). However, from the user's point of view, such status information is a natural part of the project description. The status

information is used to assess the overall progress of the project. To the user it would be unnatural to have to look at each of the activity forms to get this kind of information. Using the design shown in figure 3, the user would not look at the activity form unless the status information indicated some kind of problem.

Redundant information is a natural part of the user data model, but should be avoided in the technical data model to ease the technical implementation and maintenance of the information stored by the system. The redundancy can easily be simulated using query languages like SQL.

4.4 Heuristic 3: Identify the Appropriate Level of Detail

The technical data model is a very detailed model of the domain. The high level of detail is maintained in order to make it easy to use different aspects of the information in the technical algorithms, such as searching. One example of this high level of detail is the way you see a set of intervals represented in the data model (see figure 4 and discussion below). The technical data model does not have an explicit representation for a set of intervals, so the way intervals tend to be modelled is as pairs of start points and end points, even though the user thinks of the whole set of intervals as one single attribute.

Another example of the high level of detail in the technical data model is the implementation of multivalued attributes. This can be modelled either by defining a maximum number of values or by introducing a new entity, where the entity with the multivalued attribute is related to multiple instances of the new attribute entity.

An address is an example of a multivalued attribute and hence could be modelled either by defining a maximum number of address lines (the model might then contain e.g. two attributes called ADR1 and ADR2) or by defining an 'address line' entity related to the original entity. Which of the two alternatives is the most appropriate depends on the technical requirements. The user will however, no matter what alternative is selected, perceive the address as one concept - an address, and it would be unnatural if the user interface made references to the concept address line.

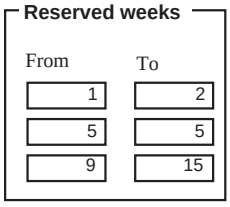

Figure 4. A form that reflect the typical way of modelling intervals in data models. A single entry field using a syntax like '1-2,5,9-15' might be more appropriate

It can be an advantage, when identifying the appropriate level of detail to be used in the user interface, to think of the user data model form as a paper form. Looking at the interval example mentioned above, it would be unnatural to the user, if

the design of a paper form included a series of fields corresponding to the way the intervals are modelled in the data model (figure 4 uses this design).

It would be more natural to have one field, where all the intervals could be written as one entry, e.g., '1-2, 5, 9-15'. A notation like that would be just as natural in a user interface, but is seldom considered because the generally available user interface tools lack a standard interaction technique that handles this kind of custom-made syntax. This should however not be an obstacle, because such an interaction technique easily can be implemented.

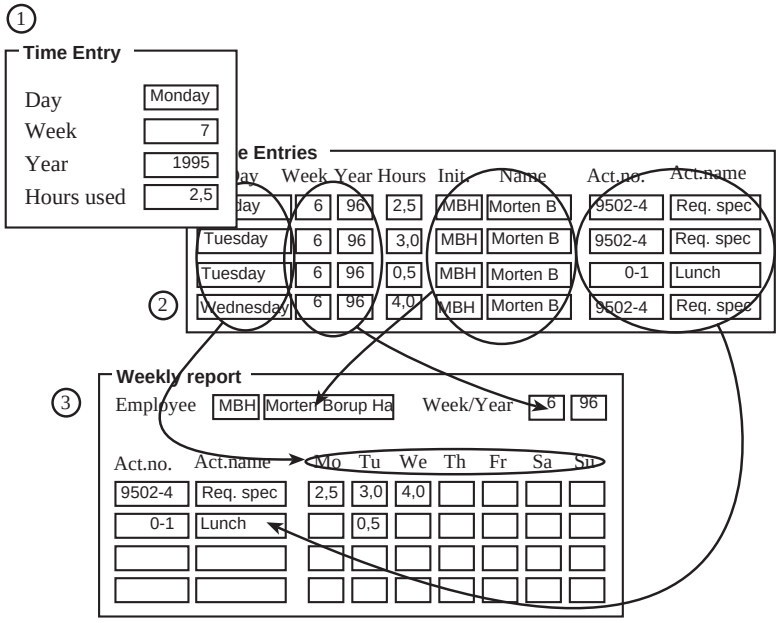

Figure 5. The transition from a single form presenting a single time entry to a weekly report presenting the same information in a way close to what the user is used to

Another problem of the high degree of normalisation is that a number of concepts used by the user are not included in the data model. Looking at the data model of the activity monitoring system in figure 1, the information bearing relation called 'spent time on' between the entities 'employee' and 'activity' lacks a reference to the meta concept the user uses to refer to this group of data - a weekly report. The technical data model is hence too detailed with respect to the tasks the user needs to perform. The user does not have any tasks that need to refer to the number of hours spent by a specific employee on a given activity at a given date.

A design close to the data model would be something like the design shown in figure 5 (1). This is just a simple form with room for a date, an activity, an employee and the number of hours spent on the activity. Using this design the user would, by the end of the week, have collected a large pile of these small and unrelated notes. The design in figure 5 (2) is slightly better, but the table will when filled include a lot of repetitions, e.g. the day. Imagining a possible paper form design matching

the tasks the user needs to perform, a design like the one in figure 5 (3) would be more natural. This design eliminates these repetitions, by using the qualities of the data shown. This is how the concept lacking in the technical data model would appear.

4.5 Using the User Data Model as a Visual Data Dictionary

The user data model should include all the necessary information needed to present information in the conceptual model consistently during the design of the logical and physical windows. Such information includes naming of the individual attributes, because the technical data model often only includes names like PROJNAME, where "Project name" might be more appropriate to the user. It should also include the natural grouping of attributes and layout of the information used when referring to a concept, e.g. name and initials when referring to an employee (used in figure 3).

If possible most of such design decisions should be made during the conceptual design, because these decisions will make it easier to present a consistent view of the concepts in the window design. The idea of collecting this kind of information is very similar to the idea of a data dictionary, and the user data model might hence be thought of as a *visual data dictionary*.

Thinking of the user data model as a visual data dictionary, it is important to include information on how to present the contents of the attributes. Should a numerical attribute be presented as a number and if so with how many decimals? Should it be encoded as a colour and if so, what values should correspond to which colour. De Baar et al. [de Baar92] and Bodart and Vanderdonckt [Vanderdonckt93, Bodart94c] provides an idea of how to make this kind of decision according to precision, scale and other parameters.

It is important to include examples of typical ways of filling out a user data model form. The examples should include messages shown in status fields, because such messages are not included in the technical data model. If included, the data model will typically model messages using a numerical code, or as an enumeration type as known from languages like Pascal. One of the reasons why including examples in the user data model are important, is that it helps determine the dimension of tables and field width etc. Another reason is that it makes it easier for both designers and users to understand and relate to the content and structure of the model.

If the user data model is used to supplement the technical data model with the kinds of information mentioned above, it is the experience from the design of the classroom scheduling system and from several case studies, that the user data model can be used as a visual data dictionary guiding the design of logical and physical windows.

GENIUS [Janssen93] includes information on grouping in the data model, TRIDENT [Bodart95a] includes some of this information in the specification of ab-

stract interaction object. However, how the information is specified is of less importance, than using the information in the design of logical and physical windows.

5 Designing Logical Windows - Tailoring the Display to the Information Needs of each Task

The user data model is designed to model the information needed to perform each of the tasks the system must support, and hence fulfils the information needs of each task. However, the user data model only presents the information in a way that makes it easy to understand the content and structure of the conceptual model, and this may not be the most efficient way of grouping the information, when it comes to performing a specific task [Lauesen94].

It is often the case, that a task requires access to information from several forms in the user data model, both several instances of the same form and several different forms. In these cases it would be more efficient to gather all of the necessary information in one window.

Windows tailored to the information needs of a specific task becomes more important, as the amount of information needed to perform a task grows. If there is not a window that combines the necessary information, the user will have to go through several windows to collect the information, increasing the likelihood of errors and misunderstandings.

The reason for designing logical windows are to ensure the task efficiency of the display design, while using the user data model as a visual data dictionary to ensure that the design remains conceptually clear.

Sutcliffe [Sutcliffe95] and Bodart et al. [Bodart95b] design the Logical Windows (only Bodart et al. uses this term) by selecting the attributes a task needs from the technical data model. However, this increases the design task significantly compared to the approach presented here, especially in the design of conceptually complex interfaces. At the same time selecting individual attributes, makes it harder to maintain the conceptual clarity of the user data model.

5.1 Analysis of Information Needs of each Task

The instances of the user data model forms that a task requires access to are specified for each of the tasks identified in the task analysis. An example of such a list is shown in table 1. The design of logical windows are then based on this list. When designing the logical windows, one of the goals is to keep the number of windows to a minimum, so if the information needs of two tasks are very similar, designing only one window should be considered, if such a window would fulfil the information needs of both tasks.

Task	Information need
1. Check the progress of all the projects and activities you are interested in	*Employee (status of the project), historical activity reports (development in 'number of hours used' and in 'number of hours used'+'estimate')*
2. You are asked to check the progress of a specific project, e.g. "9511 DXP version B" (not one of the projects you are normally watching	*Project (sum of all activities), historical activity reports (development in 'number of hours used' and in 'number of hours used'+'estimate')*
3. A specific project, e.g. "9511 DXP version B" is out of control - find the reason.	
3.1. Identify activities, that are having problems	*Project (all activities line by line), historical activity reports (development in 'number of hours used' and in 'number of hours used'+'estimate')*
3.2. Find the people, that have been working on the activity.	*Activity report, Weekly activity report (3-4 weeks back)*
3.3. Check what the person responsible for the activity have been doing apart from working on the activity in question.	*Activity report, Weekly activity report (3-4 weeks back), Employee report*

Table 1. Analysis of information needs for all tasks in the scenario 'Activity surveillance'

The names in the column called 'Information need' in table 1 refers to forms in the user data model. The note in brackets describes what instances of the form are needed or if only a small part of the form is needed, what part of the form that needs to be available. 'Development in "number of hours used"' and 'number of hours used'+'estimate', '3-4 weeks back' are examples of such notes.

If the forms in the user data model fulfil the information needs, they should be used as logical windows. If a frequently performed task needs access to information scattered over several windows, one or more logical windows should be designed, that fulfils these information needs in a manner, that suits the user.

This is also the case if a task requires an overview of large amounts of data, in such cases some kind of visualisation technique should be applied. Examples of visualisations that might be included in the design of logical windows can be found in Tweedie [Tweedie95] and Ahlberg [Ahlberg95].

However, in most cases only a simple visualisation is needed. For example to fulfil the information needs of the tasks 1, 2 and 3.1 in table 1, the designer could include a graphical presentation like the one shown in figure 6. The graphical presentation supplies the user with a number of progress indicators that would be difficult to obtain just from looking at the numbers shown in the forms.

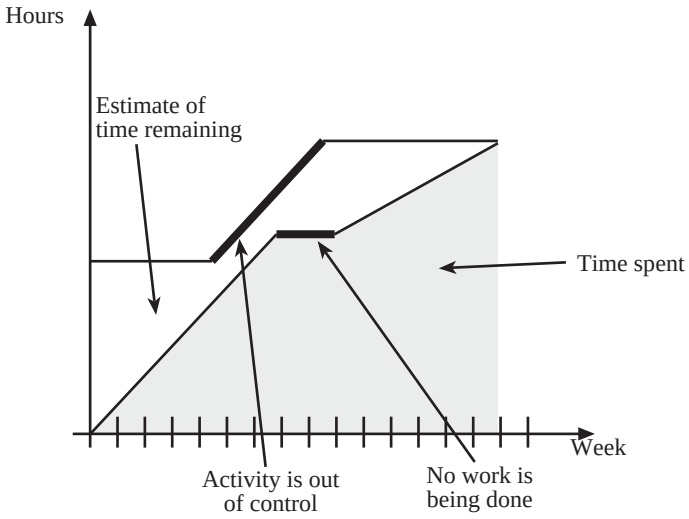

Figure 6. Development in 'number of hours used' and 'number of hours used'+ 'estimate' showed in a graphical manner.

The design of logical windows does not necessarily mean radical changes from the design found in the user data model, more often only a number of minor modifications are needed. Figure 7 shows an example of how the employee form can be changed to meet the task requirements. The only changes needed are the order of two columns and a redefinition of the meaning of the estimate column.

Finally a column has been added that show the portion of the budget that has already been used. The 'percent done' field provides the user with valuable insight on the state of the project, e.g., if the work has just begun. This kind of information is important when assessing the importance of problem indicators.

By letting the design of the logical windows take its origin in the user data model, it is possible to maintain the conceptually clear design while obtaining a higher degree of task efficiency. In order to maintain the conceptually clear design, it is important to let the design reflect the conceptual model and that the relationship between the concepts included in a window is accentuated. This can be achieved by keeping the grouping and layout established by the user data model. It is also important to use the names found in the user data model consistently.

6 Designing Physical Windows - Compliance with User Interface Standards

Having designed the logical windows only the design of the physical windows remain in order to complete the display design. The design of the logical windows aims at ensuring task efficiency, whereas the compliance with user interface standard has been deliberately postponed to the design of the physical windows.

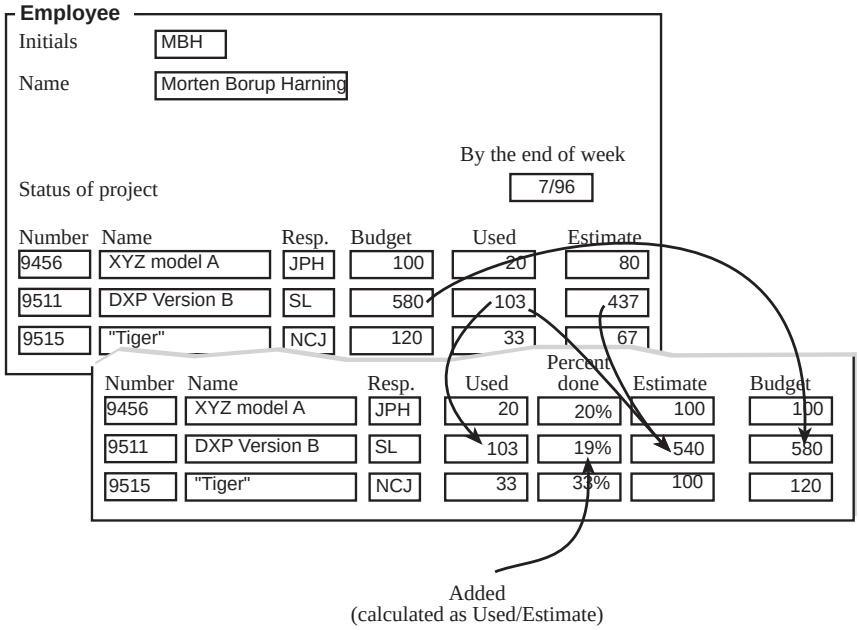

Figure 7. Tailoring the logical windows showing the user entity 'employee' to fit the information needed to perform task 1 in table 1.

The transition from the logical display design to the physical display design will usually be straight forward, because a major part of the design has already been done during the design of the logical windows and the user data model, such as layout of information and design of the necessary graphical presentations. The design of the physical windows is hence primarily a question of finding the best way to implement the logical design, in a given user interface tool or the user interface standard the interface design has to comply with.

The physical design will include finding a way to organise the windows, this includes adapting the layout in the logical windows to possible physical limitations like screen size or a CUI.

To complete the physical design the functional design and the design of dialogue state transitions have to be done. The design of these aspects of the user interface are included in the full EFDD method outlined in section 2.

The identified user functions needs to be implemented as buttons, menu items or drag-and-drop operations as suggested by the user interface standard. Finally the design must include a way of visualising the current state of the dialogue, e.g. through enabling/disabling of fields and buttons, cursor changes.

The design of user functions and dialogue state-transition diagrams are described in Lauesen and Harning [Lauesen93].

7 Utilising the Human Ability of Perceptual Organisation

To reinforce the users perception of the relationship between the information presented in the logical windows (and in the user data model form for that matter) it might help to cast an eye over knowledge available from the gestalt psychology (e.g., [Palmer94]). Some of the well known observations found in gestalt psychology are that things presented close to each other or things within a frame are perceived as belonging together (they form a gestalt).

The gestalt psychology also proposes a number of other rules that appear to guide the human perceptual organisation. These rules can be of great use when applied to the display design. The opposite is of course also true, that if the design violates some of these rules (e.g. misuse of frames and a random layout of fields), it results in errors and misunderstandings.

Conclusion

The proposed approach to structured display design have by now been used in several design cases and in a large development project. The conclusion is that the approach helps structure the display design. The result of this seems to be a conceptually clear display design with good task efficiency. The approach also shows how the existing structured interface design methods, with a focus on task analysis, can be combined with a structured display design.

Both the user data model, the logical windows and the physical windows facilitate usability testing, which helps eliminating usability problems very early in the design process.

The proposed way of designing the user data model increases the creativity by dragging the designer/user through all possible views of the data modelled by the system. At the same time the two step design process makes it more obvious to consider appropriate kinds of visualisation techniques. This seems to be helpful, both when designing user interfaces with high conceptual complexity as well as designing rather simple interfaces.

By using the proposed approach it is possible to do a structured display design as a natural part of large development projects, and hence increase the focus on user interface design. User interface design in such large development projects is otherwise a problem because it is difficult to apply the traditional HCI approaches like guidelines, prototyping and usability testing, while maintaining control of cost and development time.

Acknowledgements

I would like to acknowledge that the initial idea of a user data model was developed together with Søren Lauesen at Copenhagen Business School. Søren was also

the person that coined the phrase. I would like to thank Jan Pries-Heje for his constructive criticism reading the first drafts of this article, and the CADUI'96 referees that provided huge amounts of constructive criticism. This project has been funded by a grant from the Danish Technical Research Council.

References

[Ahlberg95] Ahlberg, C., Truvé, S., *Tight Coupling: Guiding User Actions in a Direct Manipulation Retrieval System*, in [Proceedings of British Conference on Human-Computer Interaction HCI'95 « People and Computers X » (Huddersfield, 1995), M.A.R Kirby, A.J. Dix, J.E. Finlay (Eds.), Cambridge University Press, Cambridge, 1995, pp. 305-322.

[Avison90] Avison, D.E., Wood-Harper, A.T., *Multiview: An Exploration in Information Systems Development*, McGraw-Hill, 1990.

[Baskerville93] Baskerville, R., *Semantic database prototypes*, Journal of Information Systems, Vol. 3, 1993, pp. 119-144.

[Bodart93] Bodart, F., Hennebert, A.-M., Leheureux, J.-M., Sacré, I., Vanderdonckt, J., *Architecture Elements for Highly-Interactive Business-Oriented Applications*, in Proceedings of the East-West International Conference on Human-Computer Interaction EWHCI'93 (Moscow, 1993), L. Bass, J. Gornostaev and C. Unger (Eds.), Lecture Notes in Computer Science, Vol. 753, Springer-Verlag, Berlin, 1993, pp. 83-104.

[Bodart94c] Bodart, F., Vanderdonckt, J., *On the Problem of Selecting Interaction Objects*, in Proceedings of British Conference on Human-Computer Interaction HCI'94 « People and Computers IX » (Glasgow, 23-26 August 1994), G. Cockton, S.W. Draper, G.R.S. Weir (Eds.), Cambridge University Press, Cambridge, 1994, pp. 163-178. http://www.info.fundp.ac.be/cgi-bin/pub-spec-paper? RP-94-018

[Bodart95a] Bodart, F., Hennebert, A.-M., Leheureux, J.-M., Provot, I., Sacré, B., Vanderdonckt, J., *Towards a Systematic Building of Software Architectures: the TRIDENT Methodological Guide*, in Proceedings of 2nd Eurographics Workshop on Design, Specification, Verification of Interactive Systems DSV-IS'95 (Château de Bonas, 7-9 June 1995), R. Bastide and Ph. Palanque (Eds.), Eurographics Series, Springer-Verlag, Vienna, 1995, pp. 262-278. http://www.info.fundp.ac.be/cgi-bin/pub-spec-paper?RP-95-019

[Bodart95b] Bodart, F., Hennebert, A.-M., Leheureux, J.-M., Vanderdonckt, J., *Computer-Aided Window Identification in TRIDENT*, in Proceedings of the 5th IFIP TC13 Conference on Human-Computer Interaction INTERACT'95, Lillehammer, 25-29 June 1995, K. Nordbyn, P.H. Helmersen, D.J. Gilmore and S.A. Arnesen (Eds.), Chapman & Hall, London, 1995, pp. 331-336. http: //www.info.fundp.ac.be/cgi-bin/pub-spec-paper?RP-95-021

[Bodart95c] Bodart, F., Hennebert, A.-M., Leheureux, J.-M., Provot, I., Zucchinetti, G., Vanderdonckt, J., *Key Activities for* a *Development Methodology of Interactive Applications*, in « Critical Issues in User Interface Systems Engineering », D. Benyon, Ph. Palanque (Eds.), Springer-Verlag, Berlin, 1995, pp. 109-134. http://www.info.fundp.ac.be/cgi-bin/pub-spec-paper?RP-96-025

[Chen76] Chen, P., *The Entity-Relationship Model - Toward a Unified View of Data*, ACM Transactions on Database Systems, Vol. 1, No. 1, 1976, pp. 9-36.

[CUA91] *CUA Systems Application Architecture: Common User Access Advanced Interface Design Reference*, SC34-4290-00, IBM, October 1991.

[de Baar92] de Baar, D.J.M.J., Foley, J., Mullet, K.E., *Coupling Application Design and User Interface Design*, in Proceedings of the Conference on Human Factors in Computing Systems CHI'92 « Striking a balance » (Monterey, 3-7 May 1992), P. Bauersfeld, J. Bennett, G. Lynch (Eds.), ACM Press, New York, 1992, pp. 259-266. ftp://ftp.gvu.gatech.edu/pub/gvu/ tech-reports/91-10.ps.Z.

[Desurvire92] Desurvire, H.W., Kondziela, J.M., Atwood, M.E., *What is Gained and Lost when using Evaluation Methods other than Emperical Testing*, in Proceedings of British Conference on Human-Computer Interaction HCI'92 « People and Computers VII », A. Monk, D. Diaper, M.D. Harrison (Eds.), Cambridge University Press, Cambridge, 1992, pp. 89-102.

[Foley90] Foley, J.D., van Dam, A., Feiner, S.K., Hughes, J.F., *Computer Graphics: Principles and Practice*, Addison-Wesley, Reading, 1990.

[Green92] Green, T.R.G., Benyon, D., *Displays as Data Structures: Entity-Relationship Models of Information Artefacts*, Technical Report no. 92/22, The Open University Computing Department, Milton Keynes, 1992.

[Jaaksi95] Jaaksi, A., *Object-oriented Specification of User Interfaces*, IEEE Software - Practice and Experience, Vol. 25, No. 11, 1995, pp. 1203-1221.

[Jacobson92] Jacobson, I., *Object-oriented Software Engineering, A Use Case Driven Approach*, ACM Press-Addison-Wesley, New York, 1992.

[Janssen93] Janssen, C., Weisbecker, A., Ziegler, J. , *Generating User Interfaces from Data Models and Dialogue Net Specifications*, in Proceedings of the Conference on Human Factors in Computing Systems INTERCHI'93 « Bridges Between Worlds » (Amsterdam, 24-29 April 1993), S. Ashlund, K. Mullet, A. Henderson, E. Hollnagel, T. White (Eds.), ACM Press, New York, 1993., pp. 418-423.

[Kuo88] Kuo, F.-Y., Karimi, J., *User Interface Design From a Real Time Perspective*, Communications of the ACM, Vol. 31, No. 12, December 1988, pp. 1456-1466.

[Lauesen93] Lauesen, S., Harning, M.B., *Dialogue Design Through Modified Dataflow and Data Modelling*, in Proceedings of Vienna Conference on Human-Computer Interaction VCHCI'93 (Vienna, September 1993), Lecture Notes in Computer Science Vol. , Springer-Verlag, Berlin, pp. 172-183.

[Lauesen94] Lauesen, S., Harning, M.B., Grønning, C., *Screen Design for Task Efficiency and System Understanding*, in Proceedings of OZCHI'94, S. Howard, Y.K. Leung (Eds.), Melbourne, 1994, pp. 271-276.

[Lim92] Lim, K.Y., Long, J.B., Silcock, N., *Integrating Human Factors with The Jackson System Development Method: An Illustrated Overview*, Ergonomics, Vol. 35, No. 10, 1992, pp. 1135-1161.

[Lim94] Lim, K.Y., Long, J., *The MUSE Method for Usability Engineering*, Cambridge University Press, Cambridge, 1994.

[Nielsen94] Nielsen, J., *Enhancing the Explanatory Power of Usability Heuristics*, in [CHI94], pp. 101-107.

[Norman86] Norman, D.A., *Cognitive Engineering*, in « User Centered System Design », D.A. Norman, S.W. Draper, Lawrence Erlbaum Associates, Hillsdale, 1986.

[Palmer94] Palmer, S., Rock, I., *Rethinking perceptual organization: The role of uniform connectedness,* Psychonomic Bulletin & Review, Vol. 1, No. 1, 1994, pp. 29-55.

[Rumbaugh91] Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., Lorensen, W., *Object-Oriented Modeling and Design*, Prentice Hall, Englewood Cliffs, 1991.

[Sommerville95] Sommerville, I., *Software Engineering*, Addison-Wesley, Reading, 1995.

[Staggers93] Staggers, N., Norcio, A.F, *Mental Models: Concepts for Human-Computer Interaction Research*, International Journal of Man-Machine Studies, Vol. 38, No. 1993, pp. 587-605.

[Sutcliffe95] Sutcliffe, A.G., *Human-Computer Interface Design*, Macmillan Press, London, 1995.

[Tarby94] Tarby, J.-C., *The Automatic Management Of Human-Computer Dialogue And Contextual Help*, in [EWHCI94]. ftp://trg03.univ-lille1.fr/FTP/pub/Publis/JC.TARBY/ewchi.ps.gz

[Tarby96] Tarby, J.-C., Barthet, M.-F., *The DIANE+ Method*, in this volume, pp. 95-119.

[Tweedie95] Tweedie, L., *Interactive Visualisation Artifacts: How can Abstractions Inform Design?*, in [HCI95], pp. 247-265. http://www.ee.ic.ac.uk/research/information/www/LisaDir/DIVA/DIVA.html

[Vanderdonckt93] Vanderdonckt, J., Bodart, F., *Encapsulating Knowledge for Intelligent Automatic Interaction Objects Selection*, in [InterCHI93], pp. 424-429. http://www.info.fundp.ac.be/cgi-bin/pub-spec-paper?RP-93-005

[Yourdon89] Yourdon, E., *Modern Structured Analysis*, Prentice Hall, Englewood Cliffs, 1989.