User Interface Design - Addendum
Window patterns

Soren Lauesen
E-mail: slauesen@itu.dk

Version 2.1: August 2009

This paper contains additional sections for the book User Interface Design by Soren Lauesen, Addison-Wesley, 2005. These sections show various patterns for connecting a data model to windows.

Printing instructions
Print on A4 paper, two-sided printing. Most figures are placed on a right-hand page with the corresponding text on the left-hand page.

Contents

Preface

I have used my book User Interface Design (and its precursors) for more than ten semesters. In general, students perform very well, but their main weakness is coming up with good virtual windows - windows that give adequate overview, are easy to understand and efficient for the user's tasks. The issue is not so much the fancy graphical data presentations, but the mundane text-based ones made from forms and tables.

It has taken me a long time to realize that many things that I find obvious are not obvious to the average student. One of them is the relationship between the data model and the virtual windows. I look at the pattern in the data model and intuitively imagine various ways to present it. Based on the user needs, I pick a suitable presentation. Probably I have a tacit collection of patterns in my mind.

This paper makes these patterns explicit. Some of the patterns are widely used, for instance the form-subform pattern, which can show two E/R classes connected with a relationship. Others are less common, for instance showing a many-to-many relationship as a matrix of data.

The paper is structured as new parts of the book, sections 3.8 to 3.12. Sections 3.1 to 3.7 of the book deal with graphical data presentations using curves, colors, etc. We will barely deal with those here, but focus on the text-oriented presentations of large amounts of data, for instance as they are found in a database.

3.8 Window patterns for a single table

How can we present data from a single class of the data model? In Figure 3.8A we use a class of projects as an example. We assume that each project has a name, a longer textual description, a start date, an end date, a budgeted cost, a current cost accumulated until now, and an estimated rest cost.

Simple form

At the top right, we see the first presentation - a classical form view. Each attribute is shown as a label and a text box. The user may edit the data through the text boxes or we might present the data without a box so that the user cannot change it. According to the virtual window principles, we should as far as possible use the same windows for display of data and for edit. This ensures that the user imagines the virtual windows as the objects stored inside the computer.

In the example, notice the *Estimated total*. It is a computed value and the user cannot edit it. Also notice the large text box for the project description.

The advantage of the form view is that we can show many attributes and arrange them in many ways. The disadvantage is that we see only one record at a time. If we try to show several records in this way at the same time, lots of precious screen space is wasted because labels and unused space between fields occur over and over.

Table view - one record per row

The table view allows us to show many records at the same time. This view corresponds to the table concept in a relational database. In principle we have a fixed number of columns (one per attribute) and a variable number of rows (one per record).

The advantage is that we get a good overview of several records. Very little screen space is "wasted" on labels and spacing.

One disadvantage is that it is harder to provide visual patterns to help the user. The table is just one big gestalt. Another disadvantage is that we can only show relatively few attributes. Long attributes such as the project description are not suited for this data presentation. Many GUI systems offer ways to overcome the limited number of attributes: They allow the user to hide columns, rearrange them, and change the column width. However, this gives only a modest improvement of the overview.

Table view - one record per column

The bottom of Figure 3.8A shows an unusual table view with one record per *column*. It is an advantage if we have many attributes and only a few records to show. This happens in some applications.

It is somewhat harder for the user to compare records in this way. For instance it is hard to scan the starting dates to find the first date. With a record per row it is much easier. On many GUI platforms it is technically difficult to implement the record-per-column view. This is for instance the case with MS-Access.

Tab sheets

Figure 3.8B shows a variant of the simple form view. We have put most of the attributes on tab sheets. This provides space for 5 to 10 times as many attributes on the same screen space. (More than 10 tabs make the window real hard to use.)

The disadvantage is that there is less overview. Often the user has a hard time guessing the right tab. (One example is MS-Word's dialog box for setting options.) The designer has to come up with "logical" ways of grouping the attributes - at the same time trying to make all groups roughly the same size to utilize the space on each tab. Not an easy job.

Matrix presentation

The matrix presentation is quite different from the table views and gives another kind of overview. We have a variable number of rows and a variable number of columns. In the first example we have a row for each project name and a column for each month in the calendar. Each project occupies a single cell in the matrix according to its name and starting date. The cell shows the budget for the project, but might show any short attribute. With some effort we might compress a few attributes into the cell. We might for instance color the budget according to the state of the project.

The main disadvantage is that we can show very few attributes: one for each axis (in the example *name* and *startDate*) and one or a couple inside the cell. We also waste a lot of screen space on the many blank cells if the matrix is sparsely filled. This is the price for the visual gestalts inside the matrix.

In the second matrix example, we have shown each project as a block of cells ranging from the start date to the end date. In this way we have squeezed one more attribute into the data presentation, the end date.

The hotel system's room screen and the paper lab form (Figure 3.6A in the book) are examples of matrix presentations.

Notice that these matrix presentations gradually blend into true graphical presentations such as *bubble diagrams* (Figure 3.5E) and *Gantt charts* (Figure 3.7C).

Fig 3.8A Window patterns for a single table

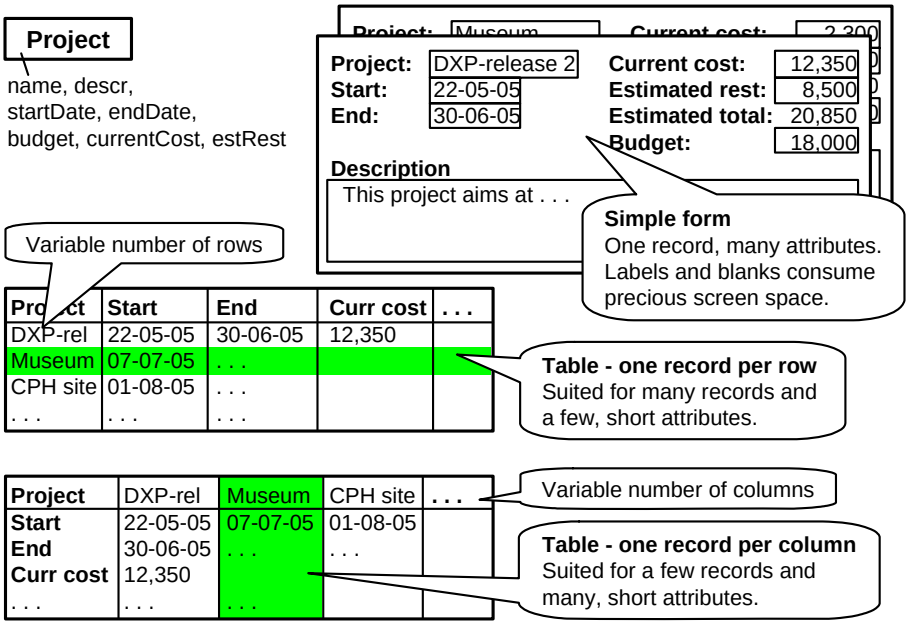
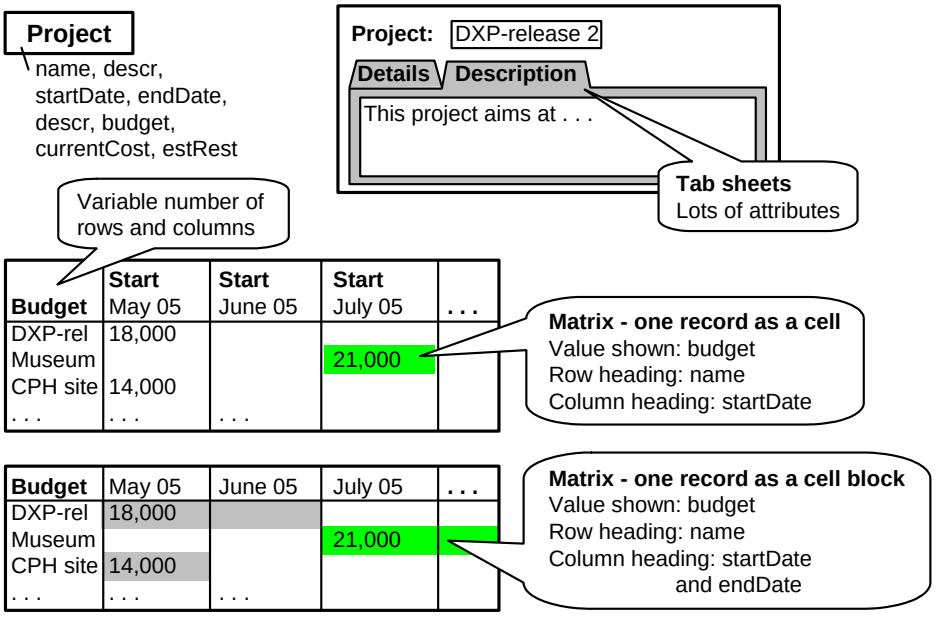


Fig 3.8B More window patterns for a single table



Graphical presentations

In section 3.5 of the book we didn't discuss how the graphical presentations related to a data model. Actually, many of them show data corresponding to a single class. As an example, the business data in Figure 3.5D consists of one record per year. Each record holds a value for customer satisfaction, employee satisfaction, sales and profit.

A note on scroll bars and search criteria

Often the data takes so much screen space that the window cannot fit on the screen. Scroll bars come in handy here, and are often added automatically by the system. However, don't rely on scroll bars. The beginner may easily overlook data that is "outside the screen". Plan the windows so that they usually will appear without scroll bars.

When a table frequently is too long to fit on the screen, use search criteria to reduce its length. See more in section 6.7 of the book.

A note on screen space and paper mockups

Above we have mentioned the precious screen space. Yes, the screen is small compared to other work areas we use, such as the office desk, the space in the kitchen or the well-equipped hobby room.

Yet most novice designers underestimate what the screen can hold. As a result they come up with a few small windows that give a poor overview, rather than one larger window with a good overview. I hate to say it, but the paper mockups seem to encourage this. I often suggest to students that they use the paper in landscape mode to get a better impression of what the screen can hold:

> **A medium-sized screen** (1280 * 1024 pixels) can hold around 60 lines of 150 characters.

3.9 Window patterns for 1:m relationship

Figure 3.9 shows a data model with two classes. We have a set of departments, and each department may run a number of projects. The department has a name and a mission statement (a longer text). The department is the **parent** of the relationship.

Form-subform

One way to show this structure is to show each department as a form - the *main form*. On this form we embed a table of projects belonging to the department. The table is usually called a *subform*.

We can show many attributes for the department, but only a modest numbers of attributes for the projects, due to the limitations of the table structure.

At the top right, we see a variant of this. Instead of showing each department in its own window, we show a list of departments and their projects as one long window. In this case we waste space because we have to show the labels of the department fields repeatedly down the form.

Table with detail window

An alternative presentation is to show the departments as a table. When one of these departments is selected, the screen shows the projects in this department in a detail window. It may be as a separate window that can float around, or as a frame adjoining the department table (see Figure 3.5A in the book).

Form with parent data

The presentation forms at the top of Figure 3.9 show data by department. The bottom of the figure shows the alternative: data by project.

The first version shows a form with the project attributes. The form also shows attributes from the parent (department data). Notice that we might allow the user to move the project to another department by choosing the department from a drop-down list, as shown. Allowing the user to type in another department name, might give him the impression that he could change the name of the department.

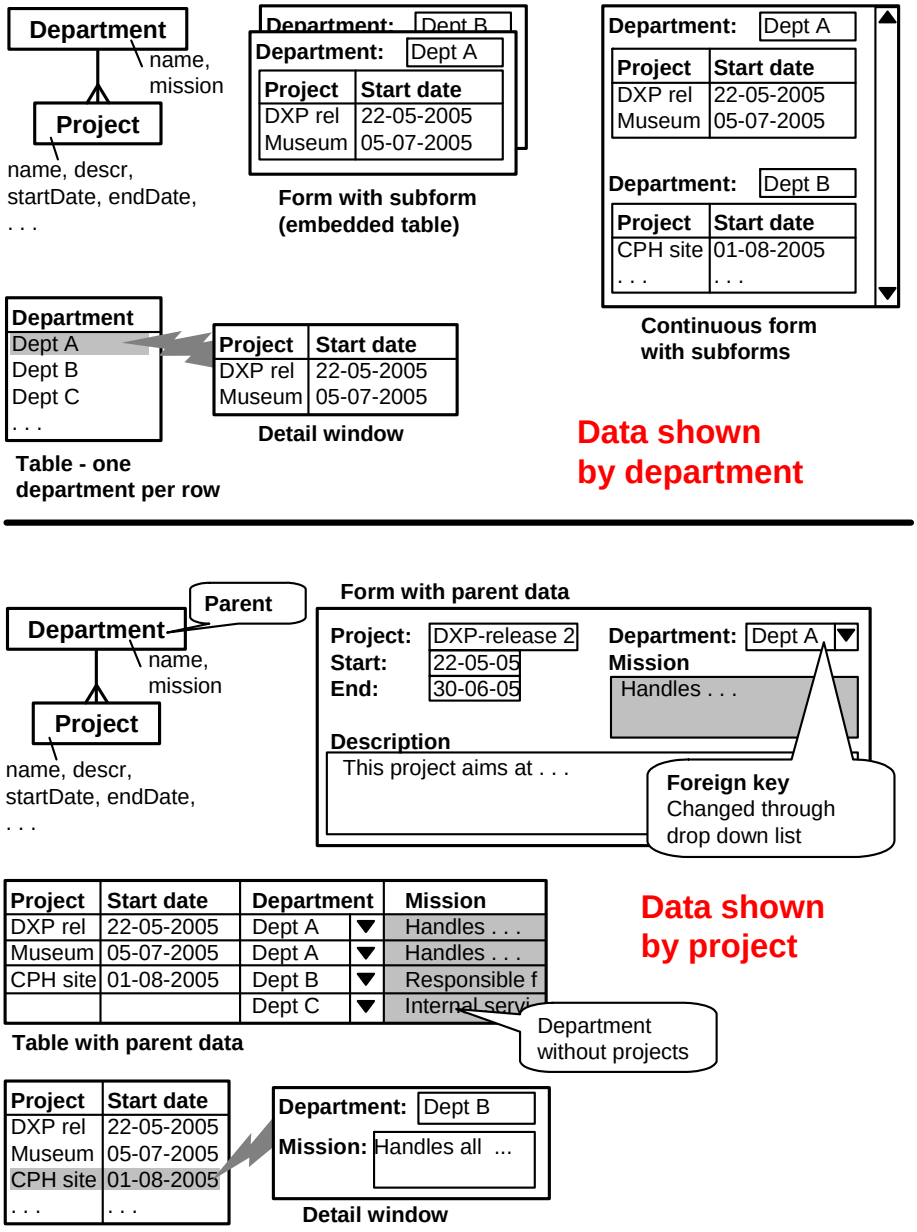
Changing the mission text might cause similar usability problems. In principle, we might allow the user to edit the mission text. However, the user cannot see whether it is department data or project data. Users are normally not conscious of the data model behind the windows, but the user might be surprised if the change influenced what he sees for other projects in the department. To prevent this problem, we have grayed the mission field to show that it cannot be changed here.

Table with parent data

The next example is a table view of all the projects. Each line also shows some data from the parent class, the department. In the example we show the name of the parent department and maybe a few more attributes of the department. Again we should be cautious about letting the user change department data this way.

It may happen that a department has no projects. In this case the department would not appear in the table of projects. Sometimes we want the parent (the department) to appear anyway. This is easy: we just show it as a row without a project. (Technically speaking this is handled by an outer join of the two tables.)

Fig 3.9 Window patterns for 1:m relationship



The figure also shows a variant where the table contains only project data. When a project is selected, a detail window shows the department attributes. In this case we are able to signal to the user that this is department data and allow him to change it.

3.10 Two 1:m relationships with the same parent

When we have two relationships with the same parent, we can use much the same patterns as above. Figure 3.10 shows an example. Each department has a set of employees as well as a set of projects.

The first pattern shows a department form with two subforms, one for projects and one for employees. If space doesn't allow this, it would be obvious to use two tab sheets as shown in the background.

The two windows at the bottom show the data by project and by employee. Notice that in the window with data by project, we cannot see the employees - and vice versa. This accurately reflects the data model, because there is no relationship between projects and employees in the model. However, such a relationship will most likely exist in the domain, and we will look at it in section 3.12.

3.11 Two nested 1:m relationships

Figure 3.11A shows two nested relationships. Each department has a set of projects and each project has a set of activities. It is easy to show each level by itself, but how can we show both levels on the screen?

Nested subforms?

With nested relationships, the logical solution would be to use nested subforms. Unfortunately, this doesn't work well in practice. In the top right window, we try something like it. We show the department as a form and its projects as a subform. Then we squeeze the project activities into a single field as a list of the activity names. You might consider the list a very rudimentary subform inside the project subform.

The solution suffices when there are only a few activities and we just need the activity names. The planning screen of Figure 3.7A (in the book) successfully uses this trick to show a list of week intervals because usually there are only a few intervals. When there are many, it shows three dots to indicate *more*. The user will then have to navigate to the full list.

In Figure 3.11A we show one more activity attribute as the color of the name. The color indicates the activity status.

An alternative to the list of activity names, is a detail window with the activity data. This pattern is shown to the lower right in the figure. These two solutions are the closest we can get to nested subforms.

Hierarchies

The first pattern above (with the list field) doesn't generalize well to more than two levels. Using parentheses in the list is a way to handle more levels, but it soon gets a mathematical flavor that would scare most users.

The second pattern (with the detail window) might be generalized to a few levels by means of several adjoining detail windows.

Simple hierarchy

A more general solution is to use an indented hierarchy. The first pattern shows the departments aligned to the left, the projects indented a bit and the activities indented two bits. We can easily generalize to several levels.

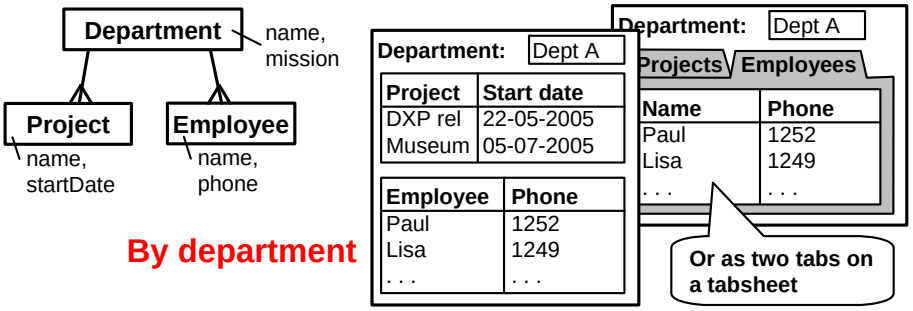
Showing the attributes is more difficult. For the activities we have shown the status. This particular attribute doesn't need a label, but in general we would need field labels. Just a few attributes would make the whole thing very hard to read and waste a lot of screen space.

We might show the lowest level as a table, in that way showing the field labels only once. But if the higher levels have different attributes, they need field labels of their own.

In some cases all levels have the same attributes, for instance if we have a recursive data model (a tree, see section 16.9 of the book). Then we can show all attributes as one table and just indent the data in the left-hand column. One example is the typical managerial reports that show costs, etc. broken down by division, department, etc. Here is an example:

Unit	**Cost**	**Employees**
America	12,345	1,300
Sales	2,300	230
West	1,300	120
East	1,000	110
Development	10,045	1,070
Texas dept.	5,045	600
Washington dept.	5,000	470
Asia	7,123	1,600
Sales	. . .	

Fig 3.10 Two 1:m relationships with same parent



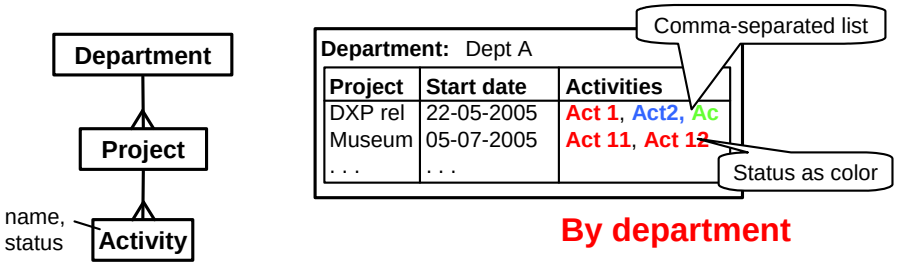
By project

Project	Start date	Department
DXP rel	22-05-2005	Dept A
Museum	05-07-2005	Dept A
CPH site	01-08-2005	Dept B
. . .		

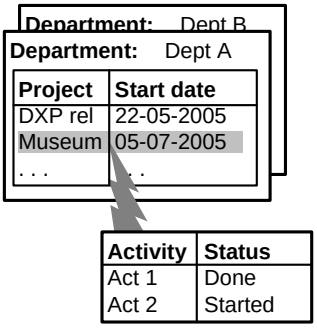
By employee

Employee	Phone	Department
Paul	1252	Dept A
Lisa	1249	Dept A
Jun	1300	Dept B
. . .		

Fig 3.11A Two nested 1:m relationships



By department



Hierarchy

Department	Dept A
Project	DXP release 2
Activity	Act 1: Done
Activity	Act 2: Started
Activity	Act 3: Planned
Project	Museum
Activity	Act 11: Done
Activity	Act 12: Done
Activity	Act 15: Planned
Department	Dept B
. . .	. . .

Expand-collapse hierarchy

Figure 3.11B shows a variant of the hierarchy. Each level (except the bottom level) expands to show the lower level when the user clicks the plus, and collapses when the user clicks the minus.

This pattern gives an excellent overview and becomes more and more used. However, showing attributes is still a problem. Often the hierarchy is combined with a detail window that shows all records at the selected level as a table. The well known file browser and many email systems work this way.

XML browser hierarchy

One way to get around the attribute problem is to consider the attributes the lowest level of the hierarchy. The right-hand window shows such an example. We can see all the attributes for project DXP release 2, all its activities, and all the attributes for the activities.

Due to its generality and the popularity of XML, this pattern is becoming popular too. Unfortunately, it is extremely poor from a user point of view. Lots of screen space is wasted from repeated attribute labels, the gestalt patterns are poor and there is little overview.

I have seen developers use such an XML browser as the user interface to a system for managing a large mobile network. The user interface was easy to develop, and it could deal with all the many kinds of nodes in the mobile network. But from the user's point of view it was a disaster.

3.12 M:m relationship

Figure 3.12 shows an m:m relationship. An activity is staffed by several employees and each employee may be involved in several activities. This m:m relationship has been resolved by means of the *participant* class into two 1:m relationships. The only attribute of *participant* is the number of hours the employee works on the activity.

A participant has two parents: an activity and an employee.

Form-subform pattern

The first pattern to deal with this has a window per activity. The window shows all the activity attributes and a subform with participants.

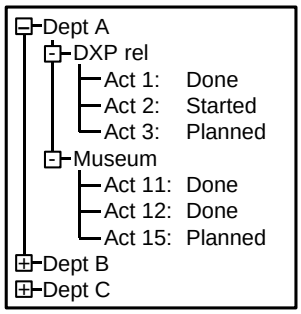
For each participant we can see not only the hours worked, but also the participant name and phone. These two fields are obtained from the employee record. We have the same problem as for other cases where parent data is shown: it may cause usability problems if we allow the user to edit the parent data.

Matrix

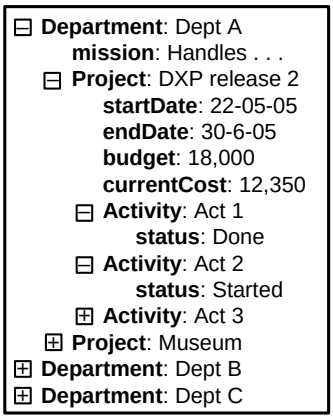
The second pattern shows the data as a matrix. Each activity has its own row in the matrix and each employee his own column. A cell in the matrix corresponds to a participant record. It shows the number of hours worked.

Notice that we show two attributes for an activity: name and status. The first two columns hold these attributes. In the same way we might show more than one attribute for the employees.

Fig 3.11B More hierarchies

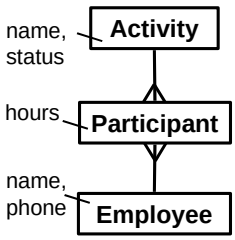


Expand-collapse hierarchy



XML browser hierarchy
One line per attribute

Fig 3.12 M:m relationship



Activity: Act 1 **Status:** Done

Participant	Hours	Phone
Paul	12	1252
Lisa	28	1249
. . .	. . .	

Variable number of rows and columns

Activity	Status	Paul	Lisa	Jun	. . .
Act 1	Done	12	28		
Act 2	Started	6			
Act 3	Planned		53	30	
Act 11	Done			98	
. . .	. . .				

Matrix - one participant per cell
Value shown: hours
Row heading: activity name
Column heading: emp. name