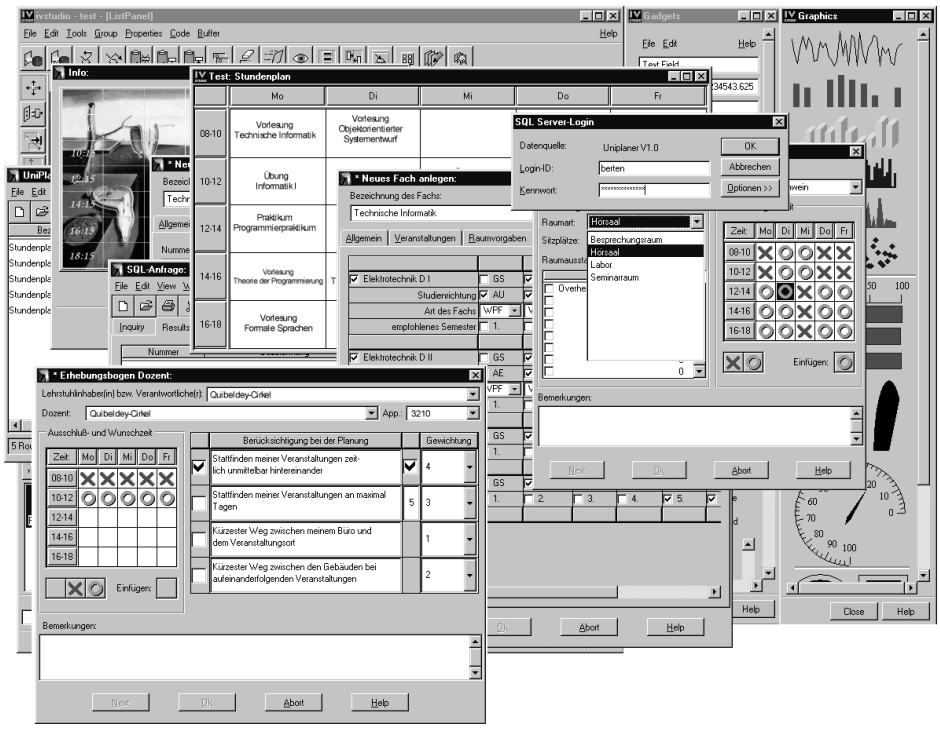
Designing a User Interface for University Timetabling: From Theory to Practice

André Berten and Klaus Quibeldey-Cirkel

Department of Electrical Engineering and Computer Science
University of Siegen – D-57068 Siegen
berten@informatik.uni-siegen.de – quibeldey@ti.et-inf.uni-siegen.de

This experience report is based on a case study about up-to-date ergonomics of human-machine interaction applied to university timetabling. Adhering to principles of software engineering, we have developed an effective graphic user interface (GUI) for our commercialised UNIPLANNER application. With the help of ILOG Views® [1], we have fulfilled the user's requirements of a simple to use and intuitive to understand one-to-one mapping from manual to electronic data gathering. The GUI component was designed as part of the *Model-View-Controller* (MVC) pattern, that is the application's data model (M), its user interface (V), and its solver component (C) are basically separated and hence easy to maintain.

Key words: GUI design, software ergonomics, form-based dialogues, MVC pattern, university timetabling.



1 Introduction

Scheduling by hand is extensive and time-consuming especially at universities. At the University of Siegen, for example, more than 3,000 courses for 12,500 students registered have to be scheduled in the face of scarce resources (< 90 lecture halls and seminar rooms). In the past, individual wishes of the lecturers, e.g. time exclusions or preferences, as well as of the students could only be considered sporadically depending on the skills and efforts of the human planner. At present, a program called UNIPLANNER, which is based on ILOG Solver®, has lighten the planner's burden [2]. It efficiently calculates a timetable that satisfies restrictions according to a list of priorities, e.g. examination regulations come first, that is compulsory courses must not be placed at the same time slot.

Motivated by the UNIPLANNER's success,[1] we are now decided to commercialise our program. To this intent, we have finished a case study on up-to-date ergonomics of human-machine interaction [3]. This experience report shows how the study's results have been applied to the design of an improved graphic user interface for university timetabling. The main objectives are fulfilled:

- ✓ The user interface satisfies the requirements of several types of German universities and can be easily modified if necessary. It complies with modern GUI ergonomics.
- ✓ The previous one-block program is now factorized into independent and extendible components.
- ✓ The program's performance could be substantially improved by implementing a client-server architecture.

Figure 1 illustrates the UNIPLANNER's configuration options. The principal components are:

- a *data base* where all data concerning scheduling and administration is stored;

---

[1] UNIPLANNER's home page: http://www.ti.et-inf.uni-siegen.de/Stundenplanung

- a *GUI front-end* used for input and output of scheduling data and for system control;
- the *solver* component that calculates timetables under prioritised restrictions.

The rest of the paper is divided into two sections. First, we summarise modern knowledge of software ergonomics focusing on the design of form-based user interfaces. Second, we introduce the UNIPLANNER's improved user interface, which is based on our study of GUI ergonomics, and outline the software architecture behind it.

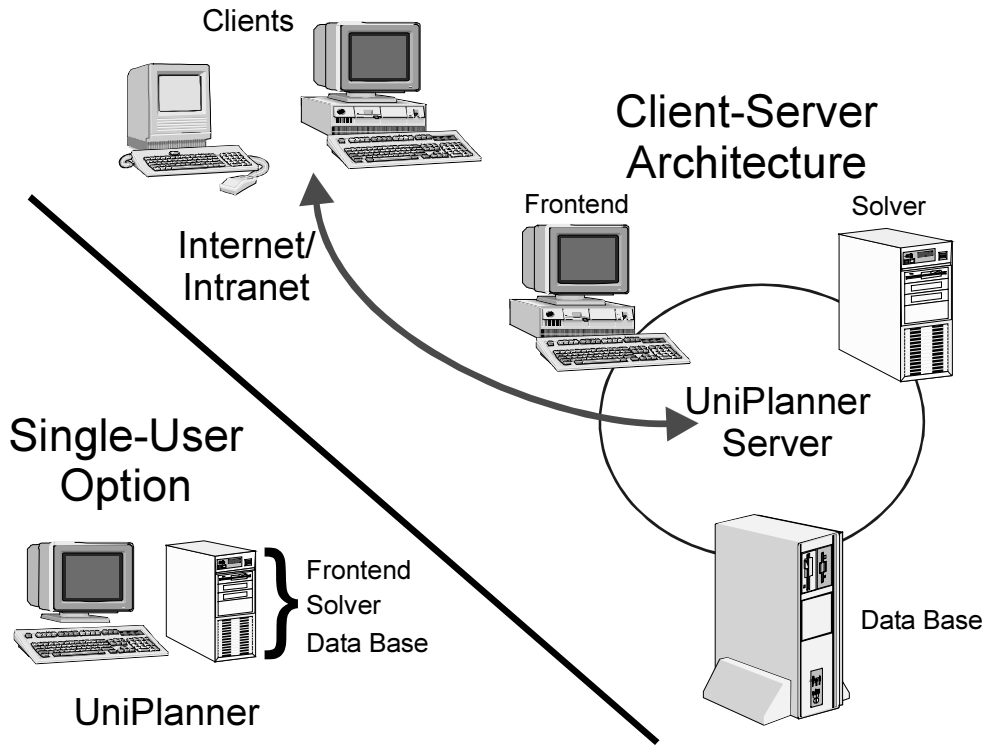


Figure 1: UNIPLANNER's configuration options

2 From Theory ...

This section outlines the knowledge of modern software ergonomics [4] that is closely related to timetabling applications. From an engineer's point of view, it might be a program's functionality that counts. This is probably why most releases only add to a program's functional features. On average, however, only 40 % of all the functions of a complex application are used. This is especially true for text processing and desktop publishing. Whether an application can be appreciated for its functionality is a question of its *operating* features. Hence, from a psychological point of view, it is the ease of a human-machine interface that decides whether a program will be used to its potential. To sum up, software ergonomics has become a decisive competitive factor.

2.1 Human-Machine Interaction

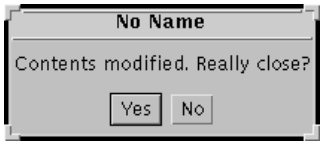
Except for some fully automated applications, computers cannot completely take on the various tasks of a human. Therefore, division of labour is the goal: where a computer is superior to a human, it is used as a supportive tool. In the domain of timetabling, a computer stores the huge amount of planning data, keeps track of the constraints, and produces consistent timetables. On the other hand, the human planner compiles the planning data and gives hints on how to satisfy the constraints more efficiently. These hints are based on experience (heuristics for variable and value ordering). Thus, the design of a user interface presupposes a deep analysis of the application domain and its particular use cases.

First of all, a user interface should reflect the working environment of a user. All terms and expressions on the window terminal should be taken from the application domain. The work flow simulated by the computer must correspond to the user's tasks. Symbols and icons must reflect the tools and materials typically used. A prime example for this *one-to-one mapping* between a working environment and its electronic representation on screen is the well-known *desktop* metaphor. Everyday office objects like files and folders, even the wastepaper basket, have found their counterparts on the

electronic desktop. That makes a human-machine interface easy to learn and intuitive to use.

To handle abstract objects without a vivid representation, e.g. numerical values, charts and diagrams help. Statistics can be easily visualised by bar or pie diagrams. Compared to text, the graphic representation of numerical values can be grasped quickly; and in addition, it simply saves a lot of screen space.

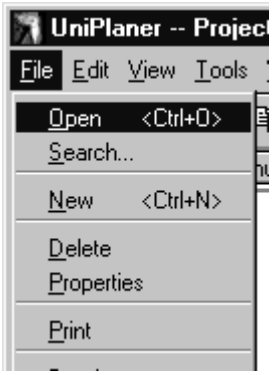
A further important aspect of any user interface is *reliability*. Any destructive action, for example deleting a file, should prompt a notice before the action is carried out. As this can be annoying sometimes, an *undo* function should be available. The undo feature gives the user self-confidence that, whatever he or she does wrong, it can be undone. Thus, a first-time user can follow an intuitive trial-and-error method to become acquainted with the program without risking corruption or loss of data.



2.2 Dialogues

Command dialogues are a relict of the past when human-machine interaction was restricted by poor hardware resources. Their existence may still be justified for expert use (see Figure 2 for an example). The casual user, however, will refuse to learn a set of commands with a cryptic syntax before he or she can do anything sensible with the program.

A much more intuitive alternative is the *menu*-driven dialogue, which has become very popular. Menus indicate the functions available to the user at a given point of time while the application is running. You do not need to learn a command language first to have some functions carried out, a single mouse click suffices. A menu system clearly exposes its structure and navigates the user through the application. The menu's tree structure supports our cognitive structure of learning. We can *see* what the program is able to do. We choose an item from the menu, and the



program will react either by carrying out the function requested or by popping up a second menu, where the function is further itemise. However, a program's menu system falls short when it comes to input and output large sets of data. For this purpose, form-based dialogues are much better suited. Figure 3 shows an example.

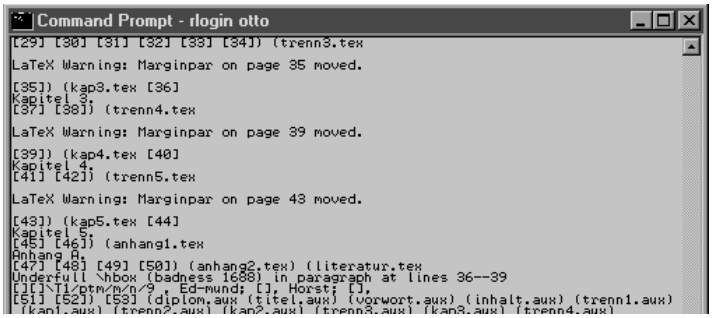


Figure 2: Relict of the past: A WindowsNT® command dialogue

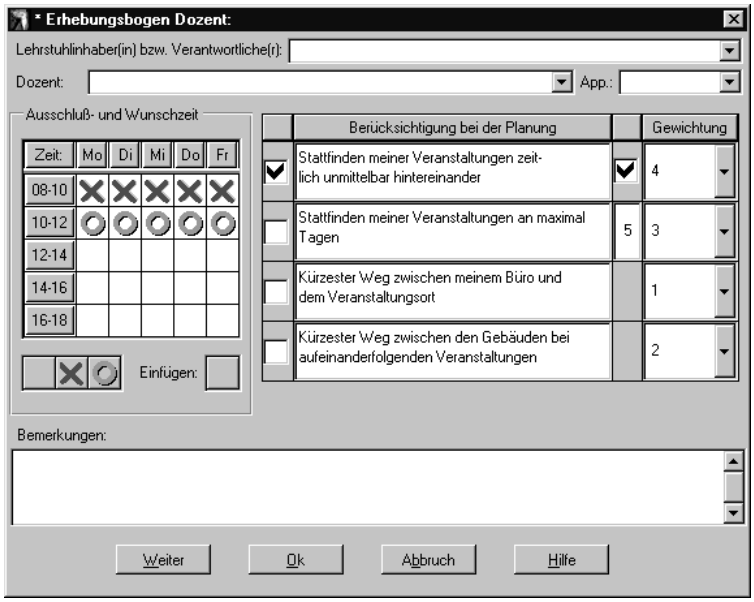


Figure 3: A UNIPLANNER's form-based dialogue

Each field is titled and may be already filled with frequently used default values. The user can either accept or overwrite them. The advantage of a form-based dialogue is its clear representation. Data sets that belong together are represented together. A form should mirror its paper counterpart as used in practice. By that, the user will quickly familiarise him/herself with the program. Figure 4 illustrates such a resemblance between a paper form and its electronic counterpart. Hence, transferring data from paper to screen becomes a straightforward task.

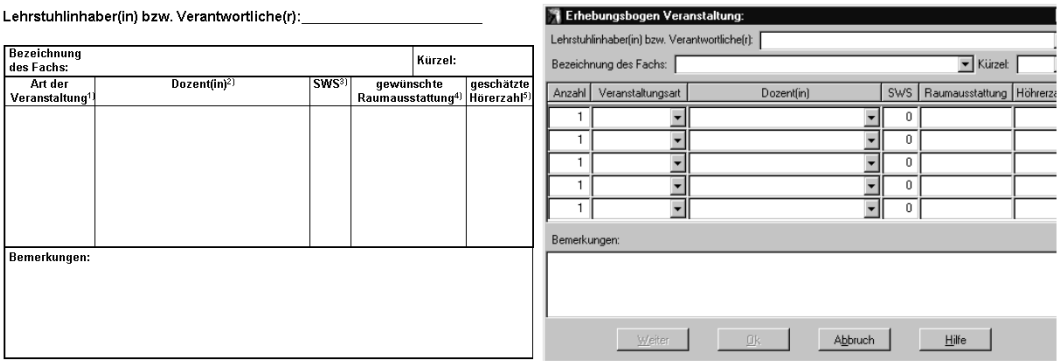


Figure 4: Paper form versus form-based dialogue

Usually, there will be a mixture of menu-driven and form-based dialogues. Both are important to keep a user interface effective. The user interfaces of modern desktop publishing programs and graphic editors are good examples for the synergetic effect between menus and forms.

3 ... to Practice

We now sketch some examples that will show how we have translated theory into action. Furthermore, we describe the software architecture that gives UNIPLANNER a better performance and a great deal of flexibility.

3.1 The UNIPLANNER's User Interface

Our program's target group are *not* computer experts, but people in charge of timetabling. Their jargon and typical tasks have led the design of the user interface. Let us summarise the main objectives:

- easy to learn and easy to modify;
- one-to-one mapping between paper and screen forms;
- all dialogues are to be operated by similar menus and buttons;
- the set of different dialogues is kept to a minimum;
- the look-and-feel is similar to office application programs;
- graphic items only where appropriate (no gimmicks).

Document Structure

The term "document" is often used in office applications, for example text document in word processing or graphic document in drawing editors. Therefore, to keep access easy for non-experts, UNIPLANNER focuses on this term. Its document structure is shown in Figure 5.

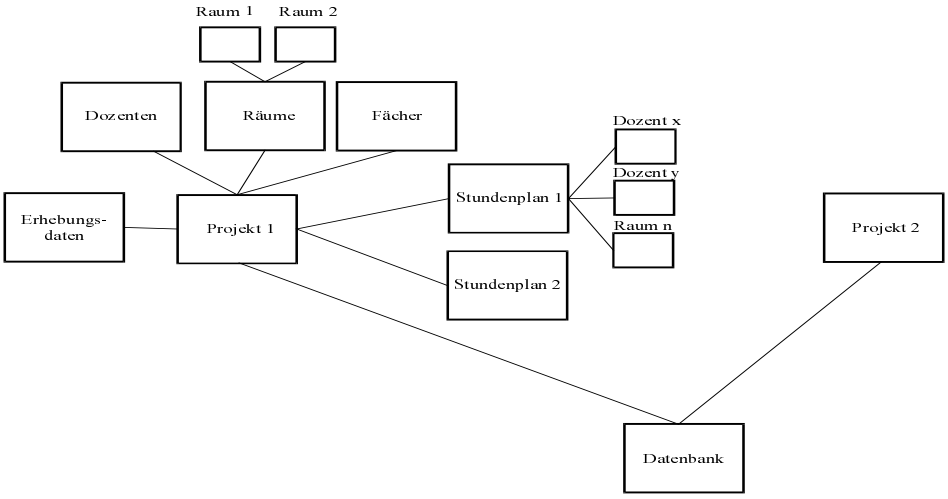


Figure 5: UNIPLANNER's document structure

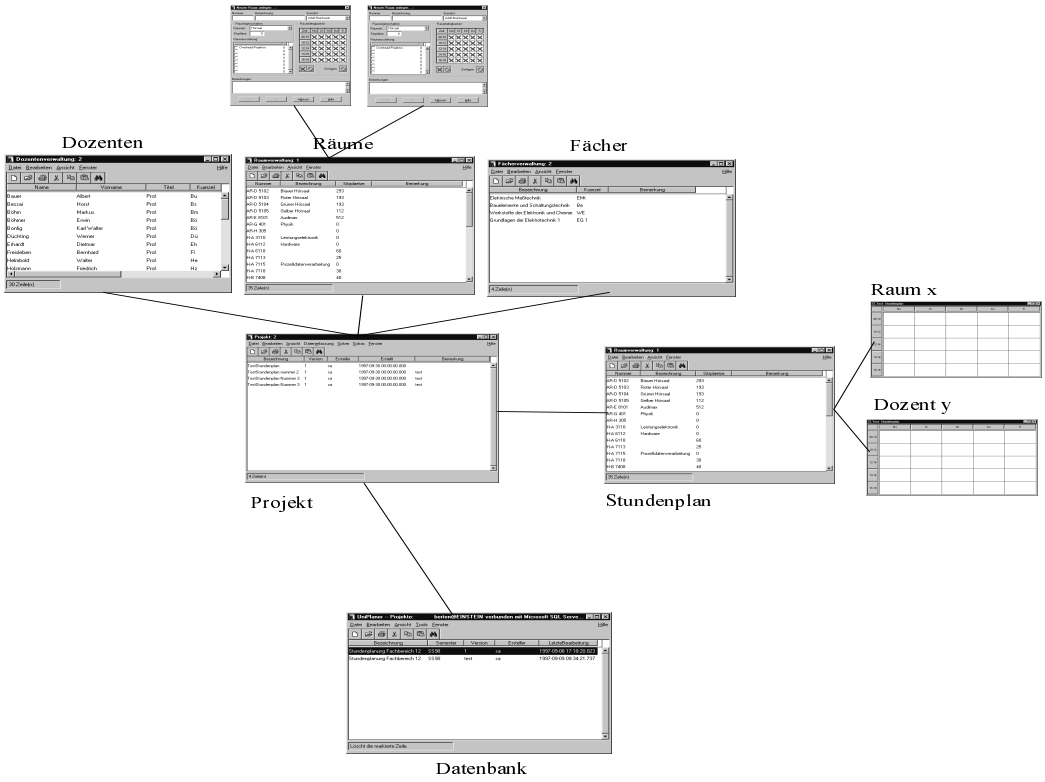


Figure 6: UNIPLANNER's dialogue structure

Here the document "data base" represents the scheduling projects as a whole. A data base document may consist of several project documents. A single project document comprises all data corresponding to the current planning session, for example lists of lecturers, courses, and seminar rooms, as well as calculated timetables. Project documents are completely independent from each other. Thus, several planning sessions can be managed concurrently.

The document structure corresponds directly to the dialogues and the application's work flow (compare Figure 5 with Figure 6). Each document has its own dialogue. Each dialogue is similar to its predecessor; and only those functions are offered that are meaningful for the displayed data. By that, a user is exclusively confronted with data and actions significant in the context of the present dialogue.

Dialogue Types

The dialogue types of UNIPLANNER are kept to a minimum. This shortens the learning process substantially. Currently, the user is confronted with just three different forms for data and action:

1. List dialogues with menu and tool bars
2. Field dialogues for data input and modification
3. Table dialogues for displaying and modifying timetables for courses, lectures, rooms, etc.

The list dialogue is central to UNIPLANNER. Lists are the basic structure for representing projects, sets of lecturers and courses. At a planning session, for example, several lists of lecturers and courses have to be handled. Timetables for students of different terms are also in list form. For each term the courses offered are assigned to rooms and time slots. Figure 7 gives an example.

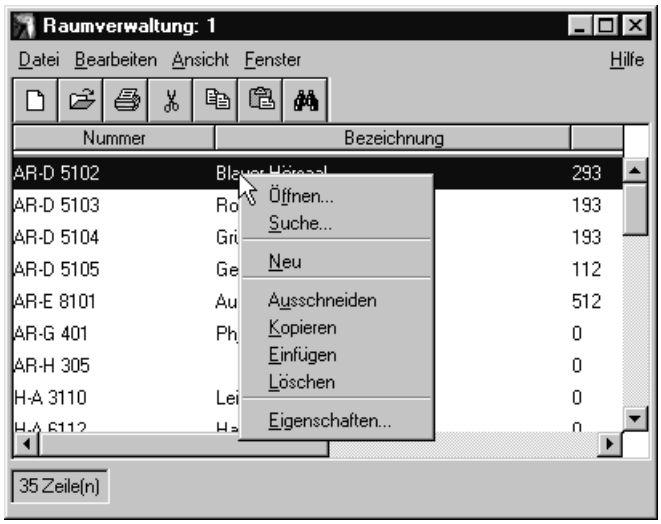


Figure 7: Example of a list dialogue

Typically, a list dialogue is divided into three sections: The *command field* consists of menu and tool bars, the *work space* is a list of data to be processed, and the *message field* at the bottom of the dialogue displays warnings and information

about the current state of the program. The list dialogue allows mouse-click selection of individual lines or group of lines. If necessary, it can be scrolled. Any action, triggered by mouse click on the menu or tool bar, only affects the lines currently selected. By clicking the right mouse button, a context-sensitive menu pops up showing all functions available for the lines selected.

The second dialogue type used by UNIPLANNER corresponds to the form-based dialogues introduced in Section 2. Compared to list dialogues, they are much more irregular as they address the manipulation of specific data. What they have in common, however, is a title bar and a set of action buttons placed at the bottom of the dialogue. The rest is kept as simple as possible and corresponds to the paper form. Figure 8 depicts an example for this dialogue type. There we see a dialogue to register a room with all its properties relevant to scheduling, e.g. equipment, capacity, and availability.

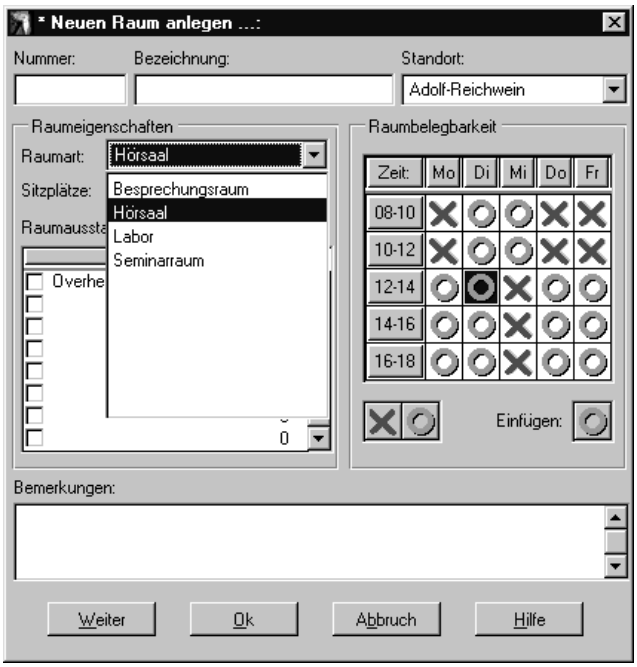


Figure 8: A field dialogue for data input and modification

The third type of dialogue is table-oriented. This dialogue consists of a two-dimensional matrix: the y co-ordinate states the time slot of a course and the x co-ordinate the weekday. Table-oriented dialogues are primarily used for time and room tables. Figure 9 gives an example of a timetable produced by our program. The tables are used for visual checking and for preparing a printer output. If necessary, a selected course may be dragged and dropped to another time slot. This, of course, will result into a new calculation of the whole timetable.

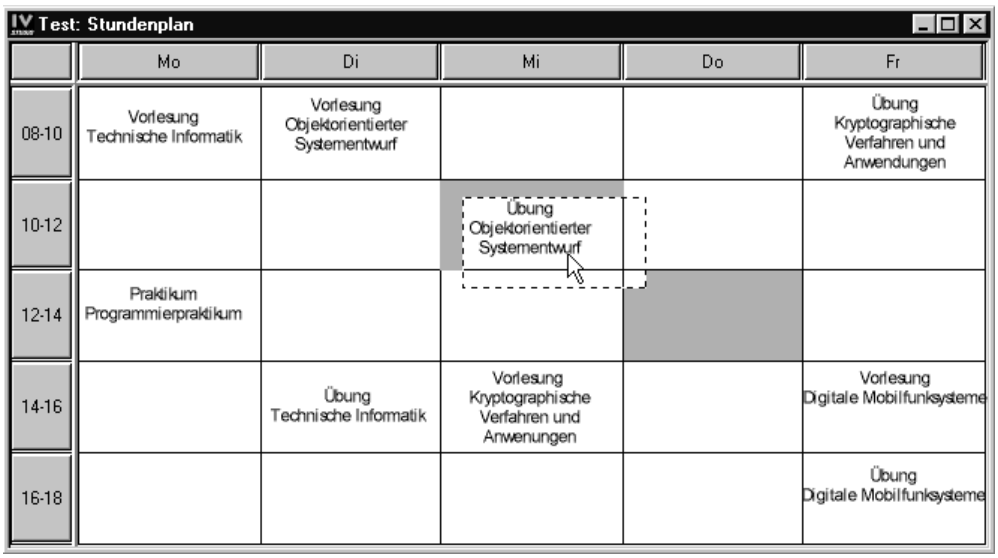


Figure 9: Drag & drop in a table dialogue

3.2 The UniPlanner's Architecture

This section outlines the software architecture of UniPlanner. Our primary goal was to keep the program's main components separate as much as possible in order to ease maintenance and further development (principle of "Separation of Concerns" [5]). In addition, a client-server structure should allow to move the solver component to a high-performance hardware platform. The well-known Smalltalk pattern *Model-View-Controller* [6] helped us in this respect. Figure 10 illustrates the components' relationships.

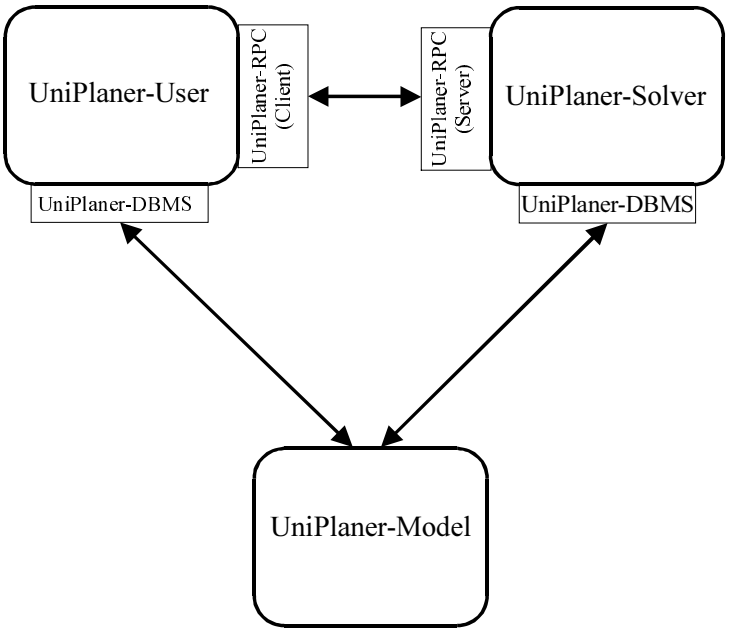


Figure 10: UNIPLANNER's MVC architecture

The *View* component corresponds to the user interface (called UNIPLANNERUSER). The *Model* component corresponds to the data model and its supporting data base management system, DBMS, (called UNIPLANNERMODEL in the diagram). And the *Controller* component corresponds to the program's algorithmic component (called UNIPLANNERSOLVER). All three components are self-contained programs, which can be run on different computers.

UNIPLANNERMODEL is the central component of the architecture. We implemented it on Microsoft SQL Server®, which supports client-server processing. The data model comprises all data relevant to university timetabling. In addition, data concerning program configuration, users management and process control is kept here. The SQL Server® can be installed on any Microsoft compatible computer. As all popular Internet and intranet protocols are supported, the data base is accessible in standard network environments. Moreover, the data model is not fixed to a particular SQL DBMS. Therefore, any low-cost option remains open, which need not necessarily be capable of multi-user or client-server processing.

UNIPLANNERUSER is the user interface of the program. It combines all options for human-machine interaction. These are:

- data input and modification
- print services
- process control
- export and import of data

UNIPLANNERUSER is a self-contained program that can be run on a WindowsNT® computer of low performance. It was implemented with the help of the class library ILOG Views® and the GUI builder ILOG Views Studio®. The advantage of implementing this component as a self-contained program is obvious: thus it can be replaced at any time. It is clearly separated from its data base and the constraint algorithm as well. Hence, it is possible to replace it with a GUI component implemented in Java, for example, in order to achieve platform independence. UNIPLANNERUSER communicates with the data base via the ODBC standard (Open Database Connectivity). Having input all necessary planning data, one or more scheduling processes can be started. As data processing is restricted to UNIPLANNERSOLVER, the user interface will not be blocked by an extensive calculation. Process control is managed by RPC techniques (Remote Procedure Call).

UNIPLANNERSOLVER is implemented with the help of the class library ILOG Solver®. This component has been reused as it was originally designed for the UNIPLANNER prototype [2]. We just wrapped it in order to function as a self-contained program. By that, UNIPLANNERSOLVER may be run on a high-performance computer over night if necessary. The solver component is completely independent from the user interface. Only the data base component is needed. Data is read from the data base and, after being processed, stored back. To access the data base, the solver uses the ODBC interface.

Altogether, we have achieved a flexible and powerful client-server architecture. The MVC pattern, which is usually used "in the small", i.e. for GUI design, has been implemented here "in the large", i.e. as an architectural pattern. The client-server features permit distributed processing and multi-tasking. If these are not needed, a single-platform installation is possible without changes to the software.

Conclusions and Outlook

Having studied software ergonomics in detail, we could significantly improve the user interface of our timetabling application. The results were evaluated by several persons in charge of scheduling at the University of Siegen. At CeBIT '98, the prototype version of UNIPLANNER was well received, which has encouraged us for the next step of commercialising the program. Several pilot projects have been initiated with representatives of different types of German universities. We are convinced that the flexibility of our software architecture will help us to serve various market requirements of university timetabling. Moreover, we are confident that we can adapt the program to fit the needs of secondary schools.

References

[1] ILOG SA:
ILOG Views® Reference Manual.
Version 2.4, 1997.

[2] Baumgart, Michael; Meyer, Sascha; Kunz, Hans Peter; and Quibeldey-Cirkel, Klaus:
Priority-Driven Constraints Used for Scheduling at Universities.
In: Proc. of the 3rd Int. Conf. on the Practical Application of Constraint Technology (PACT '97), London: The Practical Application Company Ltd, 1997, pp. 65-73.
A first version was published in the Proc. of the 2nd Int. ILOG Solver® and ILOG Schedule® Users' Conference, 9-10th July, 1996, Paris.

[3] Berten, André:
Design and Prototype Development of a Timetabling Application with Regard to Software Ergonomics and Client-Server Data Management.
Master thesis, University of Siegen, 1997. (in German)

[4] Shneiderman, Ben:
Designing the User Interface: Strategies for Effective Human-Computer Interaction.
Reading Massachusetts: Addison Wesley, 1992.

[5] Booch, Grady:
Object-Oriented Analysis and Design.
Benjamin/Cummings, 2nd edition, 1994.

[6] Krasner, Glenn E.; and Pope, Stephen T.:
A Cookbook for Using the Model-View-Controller User Interface Paradigm in Smalltalk-80.
In: Journal of Object-Oriented Programming 1, 3 (August/September 1988), pp. 26-49.