FAKULTÄT FÜR !NFORMATIK

Designing a Graphical User Interface for a 3D Data Visualisation Application

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Magistra

im Rahmen des Studiums

Wirtschaftsinformatik

eingereicht von

Karin Theresia Wendelin
Matrikelnummer 8825132

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung:
Betreuer/Betreuerin: O.Univ.Prof. Dipl.-Ing. Dr.techn. Herbert Grünbacher
Mitwirkung: Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Herbert Hutter
DI Klaus Nowikow

Wien, 28.11.2008T

(Unterschrift Verfasser/in) (Unterschrift Betreuer/in)

Danksagung

Ich danke Prof. Herbert Grünbacher für die Möglichkeit, diese sehr interessante Aufgabe unter seiner Anleitung durchzuführen.

Weiters danke ich sehr herzlich DI Klaus Nowikow, der mich zu dieser Arbeit im Rahmen seiner Dissertation gebracht hat und mir jederzeit mit Rat und Tat zur Seite stand und immer Lösungen für verzwickte Angelegenheiten schuf. Ohne Klaus wäre es nie zu dieser Arbeit gekommen.

Meinen Dank auch an Prof. Herbert Hutter, der mir — als Betreuer von Klaus Nowikow — bei der Umsetzung dieser Arbeit ebenfalls sehr behilflich war.

Ganz besonders danke ich meinen Eltern Georg und Helga Wendelin, die noch immer hofften, dass ich mein Studium eines Tages abschliesse. Und ebenfalls danken möchte ich all meinen Freunden, wenn auch mancher nicht mehr an die Umsetzung glaubte, so haben sie mich dennoch bis zu letzt unterstützt.

Contents

List of Figures

Chapter 1

Introduction

The present document was gathered within a project developing a 3D data visualisation application for the new TOF.SIMS 5 device at the Institute of Anorganic Chemistry at the Technical University of Vienna. It focuses on the essential steps of designing the graphical user interface for this application.

For a better understanding of the special requirements of the GUI, first it offers an introduction into the Time of Flight Secondary Ion Mass Spectrometry (TOF-SIMS), a way to analyze chemical materials. In the following section it will handle a theoretical treatise about GUI-Design before it offers you a short insight into wxPython the toolkit used for the implementation the user interface.

After this theoretical treatise the paper focuses on the development of the user interface for the 3D-TOF SIMS-Application called VISIMS. The paper will follow theoretical software engineering approaches from the definition of the use cases up to conceptional design of a user interface.

Chapter 2

The Time of Flight Secondary Ion Mass Spectrometry — TOF SIMS

The technological development of the past strengthened the need for processing new high quality materials, which derive their properties not only from the concentration of their main components but also from the three–dimensional distribution of so called trace elements [9] and from the prevention of contanimation. Semiconductors e.g. are examined within failure analysis to find contaminations which can have massive influence in their usage [4].

The TOF–SIMS analysis is one method to process such examinations. The following chapter will give a brief overview of the principles of this method to analyse materials.

2.1 Introduction into TOF–SIMS

TOF–SIMS is an acronym for the combination of the analytical technique SIMS (Secondary Ion Mass Spectrometry) with Time of Flight mass analysis (TOF) [3].

The simplified principle of the SIMS (Secondary Ion Mass Spectrometry) analysis is that, a solid surface is bombarded by ultra–short primary ion beams. The energy of these ions generates a so–called collision cascade within the target atoms via the atomic collisions with the bounding primary ions. At last this energy transfer leads to an overcoming of the surface binding energy of surface atoms or moleculars. The so sputtered secondary ions are quite accelerated and pass through a drifting zone the "flight tube" where they can be detected. As it is known that different atoms or moleculars need different times for passing this flight path dedicated to their masses by measuring the exact time they need to reach a detector

at the on of this zone. On the hereupon based mass analysis it is possible to get detailed information an the elemental and molecular composition of the surface [3].

SIMS is said to be the most sensitive surface analysis technique of all known analysis techniques as it is able to run micro analysis with very high lateral resolution and to detect elements present in the parts per billion range.

At last SIMS–analysis are obtaining a mass spectrum of the sample containing all atomic masses over a range of 0–10,000 amu [10].

TOF–SIMS–anlysis often are made for failure analysis and quality control but there are even wide ranges of use in research and developing. Materials that can be examined range e.g. from semiconductors, polymers to ceramics, glass, paper and metals. Some examples of its application ranges are [4]:

- detection of contaminations
- providing information of the wettability of solids
- analysing the diffusion of materials within an example
- explore the corrision of materials
- analysing the cell chemistry

Another appliance, as particles are removed from the surface what means hat SIMS affects the solids in a destructive way, is to erode the pattern in a controlled way to get information of the distribution of elements within deeper layers of the solids. This function provides the possibility of generating depth profiles of the examined sample [3].

2.2 The basic principles of the TOF–SIMS

As mentioned in the previous chapter the principle of TOF analysis bases on the fact that the travel time of ions with the same energy is depending on their masses. The lighter their masses the shorter will be the time of arrival at the detection point, where the time is being recorded. Based upon already known flight times of ions a mass spectrum can be generated from these recordings. The repetition time of this process is depending on the flight time of the highest mass being recorded [3].

A first result after collecting all sputtered ions of the surface of a TOF–SIMS analysis run can be a mass spectrum as shown in Figure 2.2.

Analysis can be made in following modes of operation [5]:

- Imaging or ion microscope mode: means that the pulsed beam focuses on a small spot of interest on the specimen and the detector is presented as kind of mirror of the specimen's surface. That means every of the detector corresponds with a point on the surface. Applying this mode it is possible to get some 2D "ion images" of the specimen. 3D images can be received while pictures are taken during the sputtering process, which removes material from the surface. Combining all generated pictures to a "stack" a 3D visualisation will be possible.
- raster scan: implies a fine focused primary ion beam that is rastered on the predefined area of the surface e.g. to determine a lateral distribution of elements. Executing this mode the primary ion beam is lower than in focused analysis. Therefore it gains a low amount of sputtered material and cannot be used for 3D imaging.
- line scan: this mode is quite similar to the raster scan with its parameter of the primary ion beam to follow a specified straight line at the surface of the specimen. [9]
- depth scan: to execute this mode two ion beams operate in a dual beam mode, i. e. one beam has to sputter a crater into the specimen's surface while the other has to analyse the bottom of this crater.

Concluding can be said, that even the limitation of the immense output of generated data from a TOF–SIMS analysis — every single point on a picture generates a full mass spectrum of it which makes a optimized data handling necessary — the formidable strengths of TOF–SIMS cannot be dismissed. There are e.g. that

- all masses on a surface of the specimen can be detected including single ions (positive or negative) up to molecular compounds
- it offers a high mass resolution. That means species of similar nominal masses can be distinguished.
- TOF–SIMS is a non–destructive analysis.
- retrospective analysis are possible.

Processing the enormous amount of TOF–SIMS data is usually done by the software of the manufactorer of a TOF–SIMS system. But they are mostly focused on 2D analysis and offer only limited 3D analysis of the specimen. Based on this fact the overall aim of the main project is focused to solve these limitations.

Find subsequently some figures of TOF–SIMS analysis.

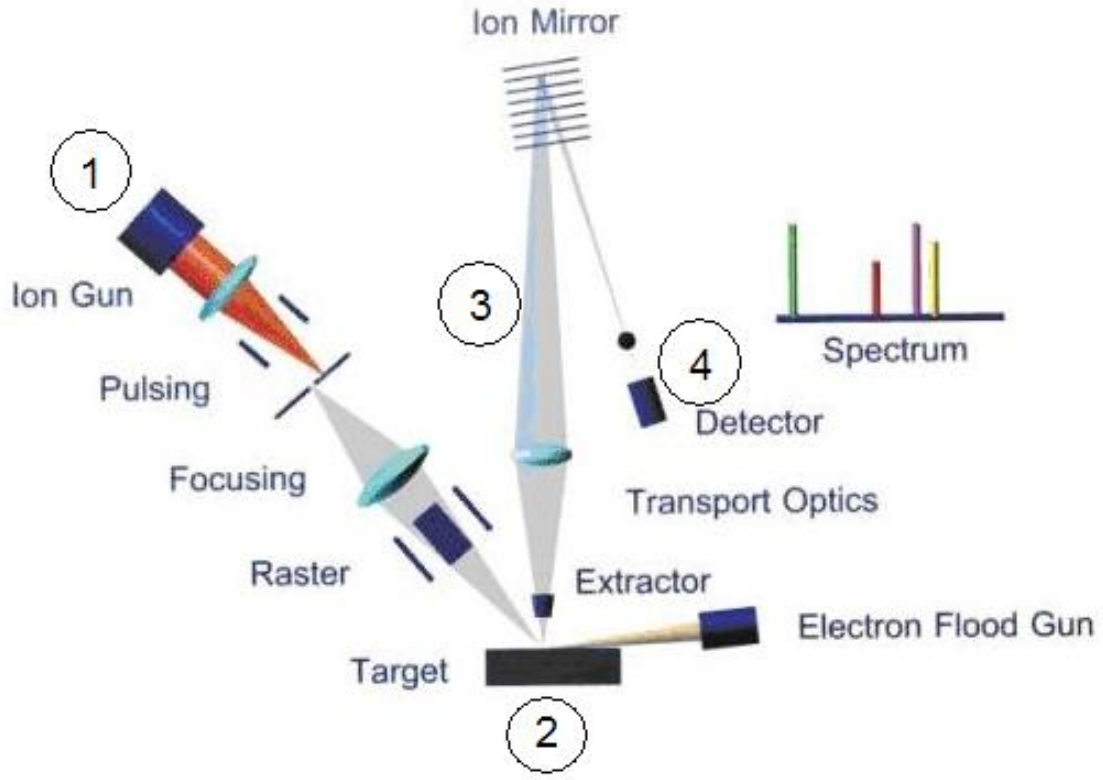

Figure 2.1: The functionality of TOF–SIMS which shows the typical configuration of a TOF–SIMS device. (1) The ion gun supplies the high energy primary ions. The pulsed primary ion beam defines the start time of all secundary ions. This primary ion beam can be focused to a small area or rastered to determine a lateral distribution of elements on the target sample (2). The secondary ions sputtered off the surface drift through the flight zone (3) where they are directed to the detection zone (4) which records the arrival time of the single elements.

[7]

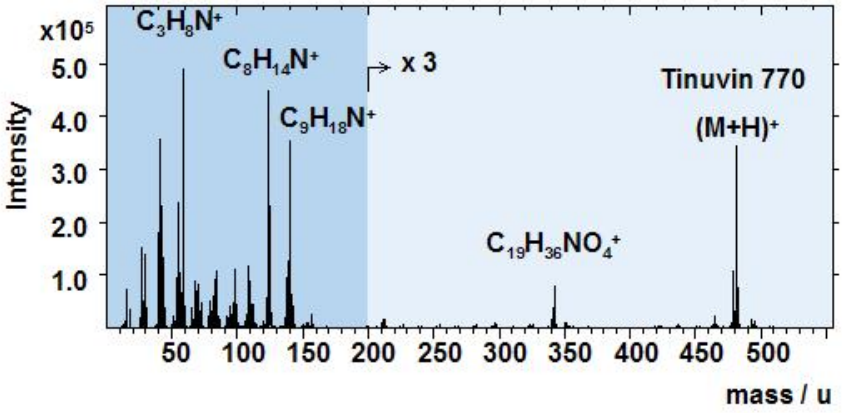

Figure 2.2: Example of a mass spectrum of polymer additives

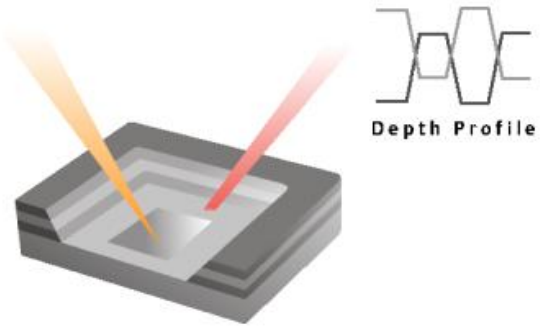

Figure 2.3: Visualisation of a depth scan
[5]

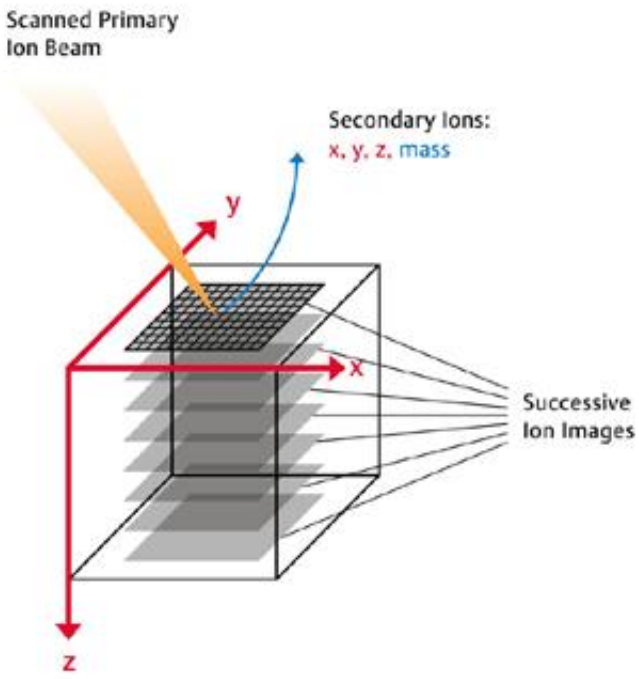

Figure 2.4: Schichten TOF–SIMS
[6]

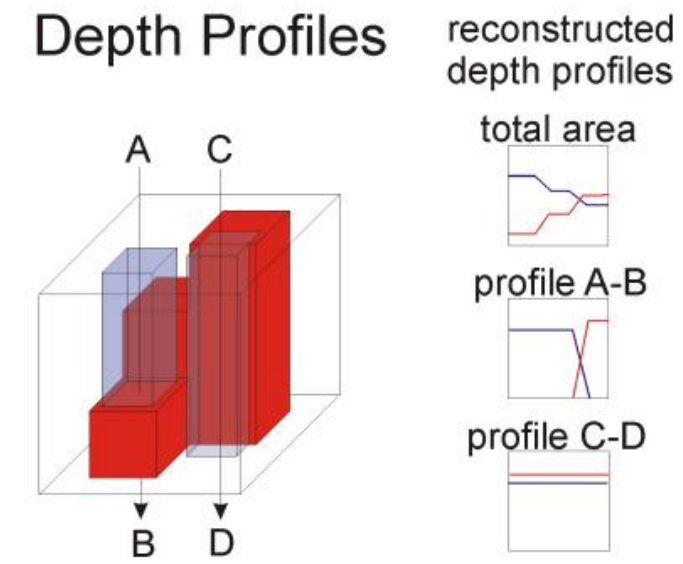

Figure 2.5: depth profiles TOF–SIMS
[6]

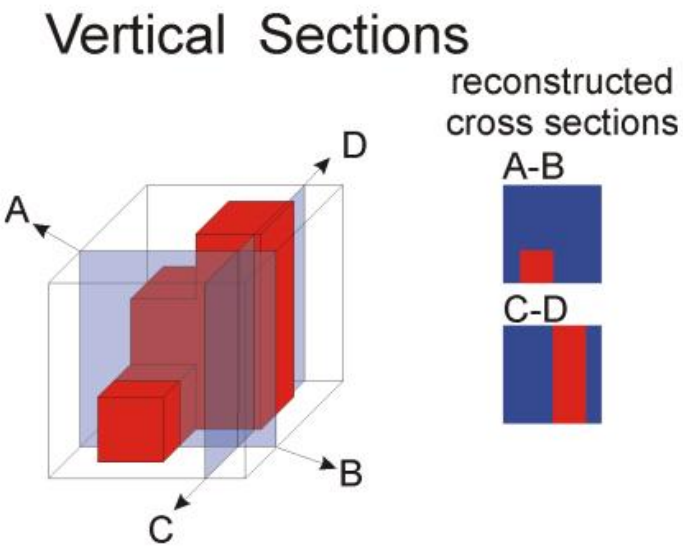

Figure 2.6: vertikal TOF–SIMS
[6]

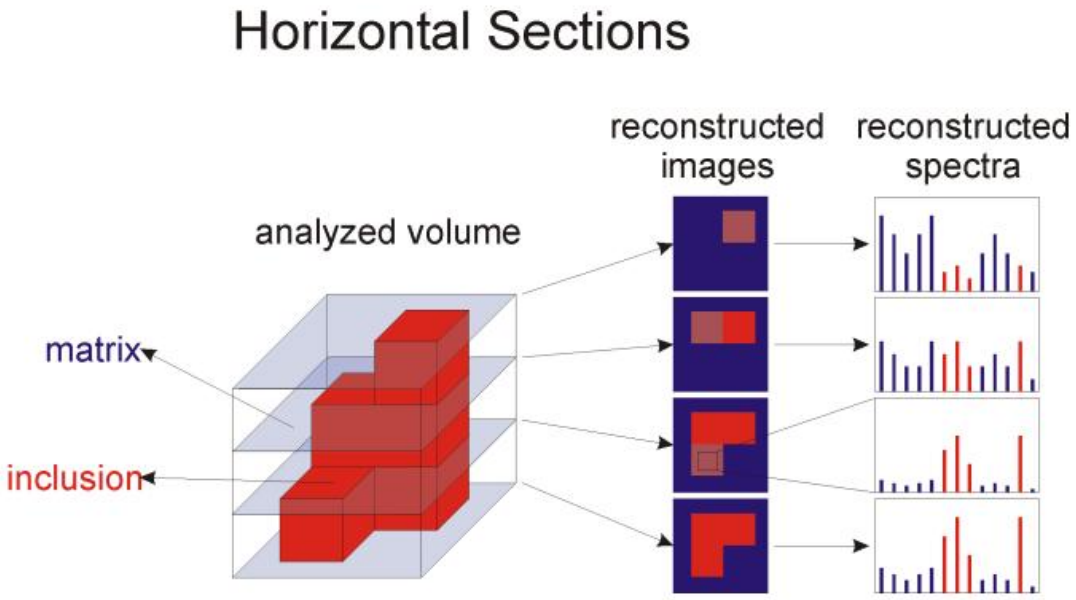

Figure 2.7: horizontal TOF–SIMS
[6]

Chapter 3

Principles of GUI Design

In this section you will get an introduction into the fundamental principles and techniques for designing user interfaces particularly with regard to graphical user interfaces (GUI).

User interfaces are a collection of techniques and mechanisms where the human user interacts with the machine. That means the GUI represents the part of the application which the user sees and interacts with. The frontend only represents the underlying algorithm and functionality and is not the calculating part of the software. The user inputs made through a GUI are sent to the processing layer of the application and the output will be sent back to the user interface as result of the user's acting [14] .

A good user interface design means to ensure usability and therefore it is the base for user's subjective satisfaction, reduction of the error rate during use of the system, better memorability of the system and therefore easier for e.g. casual user to return to the system without having to learn everything all over again [14].

The term usability features the following items [13]:

- Learnability

  A system has to be easy to learn and intuitive in handling that a user can rapidly get started with some

- Efficiency of use

  The effort of a system should show — once a user has learned the system — a higher level of productivity when using the system

- Memorability

  A system has to be easy to remember, that an average user do not have to "relearn" the system after not working with it for a while.

- Errors

  A system should have a low error rate to support the user not making any errors. In case of an error it must be possible to recover data quite easily. It must be able to except catastrohpic errors.

- Satisfaction

  The system should be pleasant to use and that users like to work with this system.

On the other hand a bad design can lead to the refuse by users of a functional excellent application.

3.1 The characteristics of the Graphical User Interface

3.1.1 Elaborate the visual presentation

As the GUI is what the user sees on the screen a sophisticated GUI represents a composition of a toolkit. That means it is made up of a variety of character fonts, graphics, colours, windows, menus, controlboxes or icons with its objective to reflect visually on the screen the users expectations of an application [1].

3.1.2 Pick–and–click interaction

Action between user and machine is performed via graphical elements on the screen. They must be identifiable for the user as element of a proposed action. This recognition and positioning of the mouse pointer on the element represents the pick-mechanism. Whereas the activating of the element usually starts with a click on the mousebutton.

This pick–and–click mechanism usually can also be performed via given keyboard inputs [1].

3.1.3 Restriced set of interface options

This characteristic represents the concept of "WYSIWYG — What you see is what you get", formerly used for printing layouts, where the user got a print out looking like the document on the screen. In this context it means that the user can retrieve available outputs through representing elements on the screen and no other meanings possible [1].

3.1.4 Visualization

Depending on the user's needs presenting difficult information in graphical images can help a user to understand e.g. information which is voluminous or very abstract. It is rather more important to know the relevant information that has to be shown in the visualization then creating a realistic graphical image [1].

3.1.5 Object orientation

Objects are the part of a user interface a user sees and handles with. A graphical user interface hast to be designed to keep the user's focus on these objects and not on the action behind them. The processes running behind an object are not a focus of user needs. Objects can contain other objects, called subobjects. They can even be combined with and hold other objects [1].

3.2 Guidelines for the User Interface Design

3.2.1 Aesthetic appereance and clarity — Look&Feel

As it is known that an attractive design draws attention, it is important to desing a pleasent graphical interface depending on a number of common and simple principles. Doing so you have to provide a meaningful contrast between screen elements, create groupings, and use color and graphics simply but targeted. Following these simple principles you can attract the user's attention, convey easier messages of the graphical elements, and make the system inviting for a user.

As most of the human-computer communications occurs nowadays through the graphical user interface it is more than ever necessary not to overload a graphical user interface with the effect of disorientation, misinterpreting of meanings, and slowing down or confusing the users.

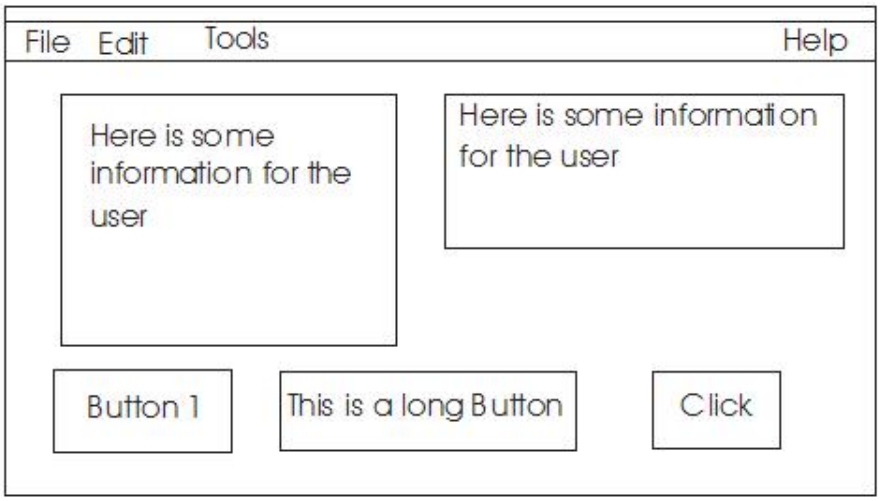

Figure 3.1: An example for a bad visual appereance

Figured you can see an interface example with a disharmonious look and feel of its frontend: the user may be disoriented by different button sizes, different styles of windows and even different menu spacing. This kind of user interface looks carelessly done and confuses a user instead of offering a clear application. A solution of this design problem might be to rearrange the menue with the same spacing, resizing the buttons while probably reviewing their labeling, and rearrange and resize the windows.

As you can see, only by following these mentioned simple steps the user gets an improved visual appeareance of the interface. The adjusted elements used provide the user a more harmonious look and feel.

Another important fact is, that wording, concept, and visual appearance of an user interface have to be linquistically clear and understandable. They have to relate to the user's real-world and to be simple and free of computer jargon.

While focussing on the layout of a graphical user interface the developer may not forget another area of aesthetics — the temporal dimension. It is well known that users do not like handling slow or only felt slow programs. e.g. displaying the progress of work done gives the

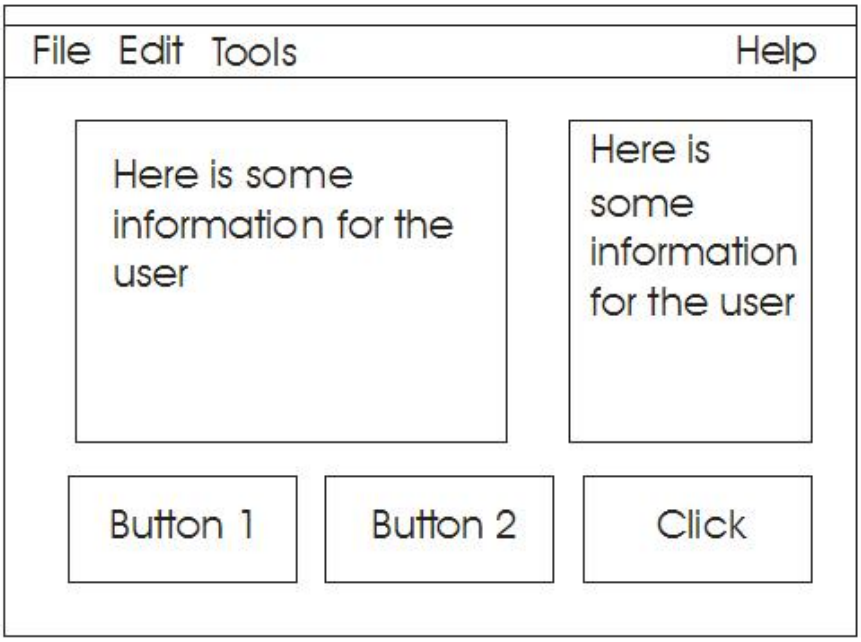

Figure 3.2: Improved visual appereance

user a better feeling while he has to wait until the process is finished. Another example is, that waiting for the underground without knowing the arrival time of the next train usually leaves the customer with ambiguous feelings whereas having the arrival time on a display gives the customer an assurance of the arrival event within a certain time and therefore usually a better feeling [12].

3.2.2 User's Compatibility

As users are not all alike and commonly have a different approach to applications one of the first questions to be answered in the user interface designing process is to find out the typical user for the application being developed — e.g. a design made for a technically skilled user will not be accepted of non–technicians like doctors or financial consulters.

Another fact is that the user's behavior usually differs from the developer's behavior as a developer knows about the aim of a functionality he made. While developing the single functions they get a deeper insight as a common user ever will have and furthermore the application follows the thinking of the developers.

To find out the average user of the application and his needs it helps to answer detailed following questions:

- What are the detailed user's goals?
- Do you know the user's skills and experience?
- What are really the user's needs?

With the result of this "brainstorming" iterative process you can generate an average user profile which includes details of possible users of the application. Doing this the developer can create so called user models.

This information is the basis to find out generalized way to create a user interface helpful for the users the achieve their goals expected of the application.

Fact is that designing an application for a special group of users may be easier than designing an application with a large general purpose, where one cannot define a specific range of users. In this case it may be helpful to form some plenties of e.g. from skilled to unskilled or old to young.

For a developer it can be helpful to talk directly to some real users to get through this contact different sights of the application's goals. The direct contact between end-users and developers in fact often radically transforms the development process.

3.2.3 Using common habits and metaphors

A possibility to make a graphical user interface easier to understand for the user is to use metaphors that resembles common habits from the users everyday life. That means the developer borrows for the application's interface some behaviors from systems which are quite close to the system's needs and a user knows how to handle. A good example to imagine this principle is the use of audio software. These programs usually have a similar interface metapher with real audio players. The user finds the standard transport controls like play and rewind.

The developer has to consider several factors when using a metaphor:

- A chosen methapher shall not be used only as single specific point in an application interface. To realize the wanted effect it has to spread throughout the interface. In the best case the metapher used spreads over several applications that the user interacts with kind of a known environment.
- It is possible to combine several different metaphors in one application. Doing so the developer has to take care that the different metaphors used do not clash and therefore derange the user's habit of use. If this case occurs it will finally reduce user's acceptance of the application.
- It is not a must have to transport software functions into metaphors. If the functions of a software are quite self-explanatory or easy to understand it is not a necessary condition to adapt a metaphor to the application's function or to strain program features to make them suitable to a chosen metaphor.
- When chosing a metaphor a developer has to consider possible disadvantages of the real-world environment. Fact is, that one assumes a risk of transporting a real world kit into a graphical user interface without knowing its advantages and disadvantages.
- In combination with knowing the users of a software it is also important to know their cultural habits for chosing a kind of metaphor. Europeans may have different approaches to a metaphor then Asians or Americans have.

3.2.4 Consistency

A program should behave consistent — internal and external.

One important way to allow users to manage quickly and properly their tasks with a software is that a developer has to retain consistency within the tool. That means, only a software which is logical, consistent, and easy to follow can optimize the user's tasks. There are two types of consistency: internal and external.

Internal consistency means that procedures, layout, and even terminology draw a continous line throughout all parts of the programm. inconsistencies could be annoying at the very least, and might cause a user to choose the wrong option the program's behaviors make "sense" with respect to other parts of the program. For example, if one attribute of an object (e.g. color) is modifiable using a pop-up menu, then it is to be expected that other attributes of the object would also be editable in a similar fashion. One should strive towards the principle

of "least surprise".
The following figures show an internal inconsistency within a reporting tool, where in the first session the user quits his report only by confirming "Yes" to be sure to leave the report. In the second report in an other part of the program the user confirms by selecting "Yes" to save a copy of his report. He quits the report by pressing "No".

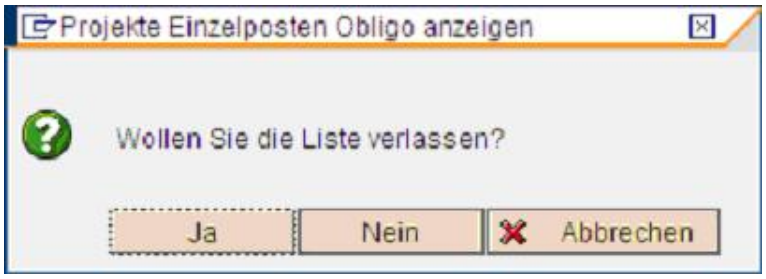

Figure 3.3: Control window before closing a report (C) SAP

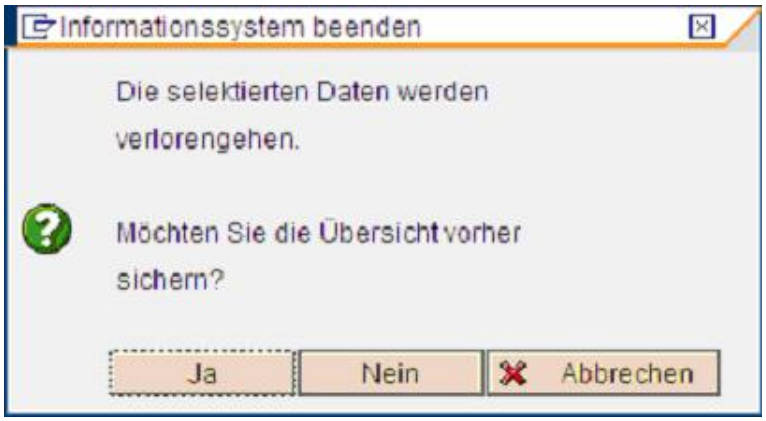

Figure 3.4: Control window before closing another report (C) SAP

External consistency means that the program is consistent with the environment in which it runs. This includes consistency with both the operating system and the typical suite of applications that run within that operating system. One of the most widely recognized forms of external coherence is compliance with user-interface standards. There are many others, however, such as the use of standardized scripting languages, plug-in architectures or configuration methods.

3.2.5 Simplicity

First of all, it is not simple to keep an application simple as in the last few years the complexity of computer programs has grown with new technical possibilities. It is the developer's duty to keep simple tasks simple for the user but even to implement complex tasks in a way they keep usable for the users.

A way to get a solution for this problem is, that the developer has to determine a solution by breaking complex tasks into smaller units, where the use of natural objects common to the user will be possible. The user will get the result of the complex problem by following the interaction objects from bottom up [2].

3.2.6 Visualise changes of states and behavior

The interface of a program should clear visually distinguish changes of the behavior of the program. This principle implies that the visualization of state changes must be in response to a behavior change. The changes of state visualization also have to be distinguishable and comprehensible for the user. Otherwise he will no longer rely on the changes of appearance as he does not get efficient benefits for the use of the program which will lead furthermore to a refusal of the application.

A good example of state changes is to highlight the current selection, wether a window, cell, or even a set of objects the user wants to interact with next. A user might get confused, if the objects e.g. several windows would keep all the same active state. He could not focus on the application he is working with as he would not find any focus because of the diffusion of "active" windows.

For a developer it is important to create a consistent and clear way to show any changes of states or behavior. A common way is to dim non–active objects with changing their colour from any painted into any kind of grey. A developer would be misadvised to generate active objects in grey as users are already used to the common standards saying grey means an inactive object [12].

3.2.7 Standardised exposing of functions and features

It is a fact that software developers know the complex model with all its tasks, dependencies and possible results and have therefore an other approach to handle functionalities of an application. Their logical way to refine and perfect abstract models, functions and databases differs to the sensual way of analysing sensory details of the environment. The approach of so called intuitives to user interfaces is a more rational way by using command lines, scripts, plug-ins, or macros.

On the other hand the sensables use their perceptual abilities and therefore they tend to applications with a front end built with toolbars, dialog boxes, and icons e.g. But it is not said that only developers are intuitive persons and that they are only intuitive.

This difference may not lead the developer to generate a user interface which covers any possible user action wether sensable or intuitive. This problem cannot be solved with the above mentioned user profile. Based on this user profile it could be possible to derive a tendency model basing on known human studies analysing the coherence of profession and human personality [12].

It is necessary to find a way not to focus the users thoughts on the technical deatails of the system. Information about the workings inside the computer should keep invisible to the user, whereas features of the program like the underlying printer state can be exposed [1].

Analogous to the principle of simplicity where the developer is said to break down complex tasks into smaller ones, the developer can generate a system from "beginners" to "advanced users". To avoid that a beginning user will be overwhelmed with too much information about functions and features the developer has the possibility to make a kind of a layer system, where the user can discover detailed but "hidden" functionalities depending on his growing abilities in using the application. A solution of this problem could look like following items:

- a completely exposed toolbar
- menu item which will be exposed by trivial user gesture
- submenu item which will be invoked by some more user gestures
- explicit addressable dialog box
- secondary dialog box embedded in the first dialog box
- "advanced options" embedded in dialog boxes

- run a shell

It is important to develop a primary interface which does not show the whole complexity of the underlying programm and architecture. The user has to have the possibility to "grow" with the application according to his level of use. And on the other hand it is the programmer's challenge to generate an application that can even grow with the user's experience [12].

3.2.8 Optimise user operations

The more a user is interacting with an application the more experience he gains and the more he learns about the results evoked from any particular user gestures. The user gets in the position to predict actions and procedures a specified handling will cause. With the growing knowledge the user is getting a different approach to the application and may come to know that the simple steps broken down from complex actions are quite cumbersome and that interacting with the application through the application's tool set last much longer then the user's acting and thinking would be.

An application should be prepared for this case and offer so–called "shortcuts" to the user to provide a faster handling with the procedures and functions and even — according to the section above — to reach a next userlevel with more powerful functions.

Even shortcuts do have different levels of use depending on the knowledge level of the user with the aim to get faster and easier to the designated results. Shortcuts are not necessarily only keystrokes. They can also be some kind of add-ins in the command line which the user can individually activate [12].

3.2.9 Building a Focus

In building graphical user interfaces it is necessary to consider results and knowledge of science, research and studies analysing the human and his functions. As graphical means that it is focused on optical cognition, to understand the function and attraction of the human eye may not be neglected.

One of the scientific knowledges is, that the human eye is being attracted from action in its surroundings. This fact implies that the user's eye will be rather drawn to animated areas than to static ones and their changes will be promptly noticed.

A very good example of focussing on the screen is the use of the mouse. It is interesting how quick mouse users learn to track the mouse sign with their eyes to navigate through and focus points on the screen. Using this property a software developer can signal state changes of the application through changing the appearance of the mouse coursor. Well–known examples are the "hourglass cursor" showing that the system is occupied and no inputs are possible, the arrow to search and focus points on the screen, and the cursor shown as text bar, to signal text inputs are possible [12].

Even though the eye is attraced from action on the screen a developer should avoid to build too much action on the screen. From the user's sight this would lead to a loss of the focus and the impression of an cumbersome application.

3.2.10 Modelling the User Interface Grammar

The order of functions and operations in interaction with objects and icons within a user interface can be compared with the grammar of a language: the subject is the object to be operated on and the verb is the operation being performed on the object. Analog to the development of computer languages it can be supposed being confronted with an grammatical order. It is the developers duty to find out what the rules are.

Apparently seen can be the two standard situations [12]:

- modality — action impacting object

  In this case the operation or tool (i.e. the action) is selected before the object is chosen. Whenever an object is selected the operation chosen before will immediatly operate on the object. The operation can be executed to many objects on one by one as the selection of the tool persists from one operation to the next.

  Selecting the tool is the mode which changes the operation of the program, that's why it is called modality. Graphical programs have good examples for this mode of operation: chosing a paintbrusher it can be applied to all objects until another mode is chosen, e.g. the eraser.

- non–modal — object impacting action

  This incident means that the object is being selected before one or more operations are chosen to interact with the object. The object is the persisting part in this case

and a variety of affecting actions alternate. As example can be mentioned a text processor where the user can select parts of his text and apply some different formatting operations.

This mode is called non–modal as all actions that can be chosen to interact with the object are always available. The more common style one can find in applications is the non–modal style

3.2.11 Provide proper Help

It would be arrogant from the developer to assume building an application everybody knows to handle and not any question will arise. The acceptance of an application from the user will decrease because of a missing possibility to find some help. Therefore it is necessary to reconsider and implement a helping module within the application.

Analysis of the human machine interaction behavior came to the result that there are five basic types of help, corresponding to the five basic questions asked:

1. Goal-oriented: "What is the aim of the function?"

   One of the best ways to come up these questions is to install "about boxes" within the single objects.

2. Descriptive: "What is this? What is it doing?"

   This kind of question is best to be answered with a standard context–sensitive help like a "help browser" or "tool tips" valid for the total application.

3. Procedural: "How can I get my result?"

   Similarily to the questions type 2 these questions can also be answered via a help browser. But in this case they found out that it is more useful to offer the user interactive help possibilities which will guide from the problem to the solution, like interactive guides, help wizards or cue cards.

4. Interpretive: "Why did this happen?"

   Studies found out, that this kind of question is already not well addressed within applications. One proposal to come up to the user's needs of help would be to analyse possible error sources during the user interface designing process and to prepare appropriate pop–up error messages.

5. Navigational: "Where am I?"

   A proposal to come up to this type of help is e.g. to create an application roadmap — a kind of landscape for the use of the application.

Even though lots of analysis have been made to describe different strategies for answering these questions in combination with the development of the adequate help interface, none of the mentioned solutions is already final but they are the common standards of helping objects [12].

Unfortunately it is still common that the helping modules are treated like a stepchild as the development focuses on the completion of the application itself, often accompanied from delay within the development or cost determined.

Another thing that has to be considered in case of the helping modes is the wording of the messages. The language of helping tools should be adjustet to the user's wording which commonly differs from the developer's phrasing. In particular the developer has a deeper insight in "his" application what can cause that a helping module written from the developer is missing information clear for the developer but not for the user.

In this part of the user interface design it is necessary to work close with users or non developers to gain a different sight of probable sources of error and the user's outcoming helping needs of this process.

3.2.12 Providing a Safety System

As above mentioned that the user interface designing process is quite influenced of scientific studies of human behavior it is necessary to mention that even the human need for safety impacts the design of graphical user interfaces.

To this fact has to be mentioned that every human being has a range of minimum and maximum risk–levels. That means if a person comes into a situation where the risk confronting the human exceeds the maximum level of risks, he will take action to reduce the risks and modify his actual situation. On the other hand, if the risk level drops under the minimum zone, usually the human will even take action to change the current situation.

These ranges differ for the people itself and for their living or educational situations. Whereas the level of safety depending on human characteristics and living situations is hardly measured, the influence of education and experience on the range of risk levels can be allocated within user groups. That means, security requirements can be grouped from inexperienced users up to "power–users".

The developer has to find a way to combinate the needs and ideas regarding security of these different users within the application. An unskilled user must feel safe whenever treating the application. They have to feel being safe from their own lack of skills. The safety net offered from the programm has to make these users feel comfortable and inspirational in using the application. Otherwise the programm will get no acceptance of these users. Secure questions like "Are you sure?" and multi–level undo options are the core of the security net for these type of users.

For the experienced users the application should include the options to choose wether to use the security components or to turn them off — according to the principle to optimise user operations. The option, to turn off safety checks should even have an undo function, for the case e.g. of a mistaken break of these options [12].

3.2.13 Limitation of User Activities

This guidline says that a user action only takes place within a given context — e.g. the current document or selection. The validation of selected operations is depending to the context they have been chosen and they may not be valid in another context. Usually e.g. in applications the user cannot not select a text object simultaneously with selecting individual characters of another text field to format them at once in the same style.

This principle is founded on the reason that the users — even skilled users — might get confused of impacts of the current context to all selections made and this may lead to unexpected errors [12].

3.2.14 Testing the Design with Users

During the software developing process the developer has several steps of testing the application. Usually a software designer is able to find fundamental and logical defects of the user interace. As a developer does not have the same approach to the software he is developing

and which follows his own logical way he cannot spot defects in a way a average user will do [12].

Only the people who will use the application can tell how the quality the interface will be. The point is to find out what will happen when real users start using the system. Therefore it is necessary to chose a representative number of users out of the user profile defined some steps before [8].

Statistics have shown that the testing of user interfaces with defined end-users is one of the most effective techniques to discover design defects [12]. But there are also some ethical concerns in working with test user. First of all acting as test user may not be distressing. Tests made under stress can lead to false results. Another factor is, that the information collected may not harm the privacy of the test users and at furthermore it can be a problem to have friends, relatives, or co–workers (especially subordinates) as test users as it is not secure that they really feel free to act [8].

However, following specific techniques can lead to maximize the effectiveness of end-user testing. The following steps show a summary of these techniques:

- Setting up the observation: first of all test tasks have to be designed based on the assumption these are pretty like real tasks. For next it is necessary to recruit end-users being a quite typical sample of the real user. One precondition is to avoid the recruiting of users who are familiar with the product [8].
- Describing to the user the purpose of the observation: according to the above mentioned ethical concerns, the tester has to know the aim of the test series. That the topic is not to test the test users, and that — if any inconvenient situation happens — they can quit at any time. The test user has to know that faults are not up to him but they are necessary to know for the developer to find the problem behind in the design [12].
- Provide a test system: user test should already start in very early phases of the development process because error found very early are easier and cheaper to correct. Therefore a test system can even be some paper prototypes where the developer has to explain and demonstrate how this test serie will run. According to the development progress the test environment will change.
- Deciding what data to collect: before starting the test unit the developer has to decide which data are helpful to collect. Process data will give information of what is happening step–by–step whereas bottom–line date inform about what happened. [8].

- The thinking aloud method: The basic idea of this method is, that the developer asks the test users to verbalize what they are thinking about using the product, what they are trying to do, or which questions arise during their work with the product. The comments of the users will be recorded. The observer stays with the user but does not offer any help. His duty is to remind the users to comment their doings [8].
- The observer of the testing series will not provide help.
- Explaining the given tasks and introduce the product.
- Any probable questions are being discussed before starting the tests.
- The conclusion of the observation implies to tell the test users what was found out and to answer any of their questions.
- Using the results.

As already mentioned user testing can occur at any time during the development cycle, and however, it is more efficient to start these test series as soon as possible in form of building mock-ups or prototypes of the application to find out principle design errors as it is much easier dealing with a design defect before the application implementation. This proceeding of quite early user tests can avoid high developing costs and product delays because of the time intense complexity of a later developing point.

3.2.15 Filtering information

As developer it is important even to gain further information concerning the application by simply watching other people using this program, listening to their opinions, and proposals. Even if it is not necessary to implement every feedback a developer gets from the user, it is a possibility to gain further valuable insights to the program being developed.

A possibility would be to take these amount of user opinions, comparing them with the special knowledge the developer has about the application, combining them, and at least reducing them to a kind of greatest common denominator.

With this method it is possible to combine the different perceptions and to balance the fact that each part in this developing process has his own picture of the system to be developed.

It is already known that it makes no sense to develop applications based only on customer feedback, which would tend to be mediocre tool. But on the other hand it is even not good to

only rely on the developer's intuition to know what the principles of a good design and a bad design.

Some things designers have to know about their users:

- According to the social and occupational environment people tend to have a biased idea what an "average" person is alike, resulting of the fact that most of our interpersonal relationships are in some way self-selected. Only a rare number of human brings their daily life into contact with other people from a full range of personality types and backgrounds. This leads to our preconceived opinion, that others think "mostly like we do." Designers and developers are no exception of this fact.
- Usually people have some core competancies, where they can be expected to perform well within that domain.
- Computer literacy: that means the skill of using a computer is in fact much harder than it seems to be.
- A lack of "computer literacy" cannot not educe an indication of a lack of basic intelligence. Native intelligence contributes to one's ability to use a computer effectively. But there are other factors being just as significant, such as a love of exploring complex systems, and an attitude of playful experimentation. One has discovered that much of the fluency with computer interfaces derives from playing computer games, but also from dedicating themselves to "serious" tasks such as running a business may lack the time or patience to be able to consecrate themselves to effort to it.
- Studies say that a quite high proportion of programmers are introverts, compared to the general population. It does not mean that they are only solitary people, but there are some specific social situations to find that make them uncomfortable. A lack of social skills could have been discovered combined with a retreat into the world of logic and programming. This leads to the fact that mostly a developer might not be an experienced people–watcher.

A solution for a developer to learn from end users about the behavior of using computers, is to spend some time with them, watching their behavior and probably getting a "fee" for the thinking and acting of an non–technical person. This knowledge can be the basis for setting up user test series.

3.2.16 Further Guidelines

The already mentioned guidelines describe only an abstract of guidelines for user interface developing. Here will be some more mentioned but not discussed in detail:

- Involving the user via participatory design: that means the user is getting involved into the designing process of the application and not only into the testing series. This will offer different sights of the necessities right from starting the designing process.
- Considering the human limitations of memorability: the capacity and duration of a human's short–term or working area must be considered in the designing process.
- Recognition is better than recall of information: fact is that it is easier for human to recognize things than having to recall them. It is e.g. easier to refind an alread made choice from a menu than to recall all the elements within the menu.
- Using user–centered wording in messages: the case occurs that the system is sending messages to the user. Basically the developer has to take care not to express these messages for non–techniciens.
- Using positive, nonthreatening wording in error messages: as error messages may have the greatest psychological impact on users, words like "disastrous, fatal, catastrophic,..." have to be avoided in order not to disconcert the user any more [2].

3.3 Advantages and Disadvantages of Graphical Sysmtes

The development of graphical user interface was accompanied with a lot of promises being improved according to previous user interfaces. Let us comparise some advantages and disadvanteges graphical user interfaces brought.

Some advantages commonly reported in literature are that [1]

- symbols are faster recognised than text
- it supports a faster learning as pictorial representation can easily be learned
- it is easier to remember the system because of its greater simplicity
- reversible actions are easier to realise
- it requires low typing skills

- it is more attractive
- it is more natural as actions and visual skills are emerged before languages.
- it is exploiting visual cues, so that spatial relationsships are seen to be understood more quickly than verbal

On the other hand there are some disadvantages mentioned [1]:

- a greater design complexity is necessary as the number of available tools is unlike higher than found in text–based systems
- there is still a lack of experimentally–derived developing guidelines also being due to the fact that developers will not publish their results to maintain a competitive advantage
- there are still a lot of inconsistencies in technique and terminology
- they are still inefficient for experienced touch typists
- they do not use always the fastest style of interaction
- they may consume more screen space
- they can cause a problem of hardware limitations if one has not the adequate power

3.4 The Life Cycle for User Interface Development

A life cycle for user interface development is a less rigid sequential process than software developing processes. The life cycle shown in figure 3.5 intends to be equally supportive to a top–down or bottom-up developing processes with the possibilities to perform inside–out and outside–in development.

This model implicates that a user interface engineer can theoretically start with almost any development activity. It even offers the possibility to move on to nearly any other element in the life cycle. The activities include the analysis of the system, the users, needs and the aims of a system as well as task and functional analysis.

The designing process needs the inputs from the task analysis and the features of the system. Starting from an initial conceptual design with highly abstract objects the process runs through iterations up to a detailed design with fully functionalities.

Another activity shown is the process of prototyping with its goal to provide early impressions of the nature of the final product. Via prototyping one can elaborte different evaluating ideas and define weightings of alternatives. A main factor of this activity is to gain feedback from users to have the possibility to creat a refinement of the product based on the feedback.

The principle of rapid prototyping says that from the beginning of an evaluating process of the prototype through the user the interaction time leaves time for substantial changes, which allows multiple iterations for fine tuning.

The evaluation of usability aims to gather information about the usability or potential usability of a system to improve therefore features of an interface in a developing process or even to assess a completed interface. It is not possible to get a relationship between design and use, if the usability is not evaluated [2].

3.5 Interaction Styles

Designing user interfaces the developer needs some devices with which the user can communicate with the application. These objects make it possible, that a user can make some input in an application or gather some information of the software he is interacting with.

The collection of interface objects and associated techniques appropriate for the use to design user interfaces for the communication with human users are called interaction styles. These objects are the basis on how the interaction component will communicate with the user and they are being integrated in a software designing process [2].

Earlier a developer had to implement each interaction style itself. Nowadays it exists a wide range of toolkits, which can be integrated as a widget. This makes the reuse of software and even the modification much easier [2].

In the following sections some common interaction styles are presented.

3.5.1 Windows

A window is a screen object. It provides the screen space for other interaction objects. All userinteractions with an application are occuring through a window. Two kinds of windows are known:

- Primary windows: the primary is the window from which all other windows of this application are generated — it could also be said the main– or root–window. See figure 3.6
- Secondary windows: these are the windows generated through a primary window.

If the user has multiple windows open, usually only one is active and can accept user inputs. A (active) window can be closed via mouseclick. Designing a user interface one has to consider several windows design guidelines, like to minimize to amount of windowmanipulation. Another one is, that the primary window should stay consistent, whatever the user is going to do. That means, the appearance of it will not change. Different tasks have to be implemented into different windows to provide the user the possibility of a visual distinction. [2]

3.5.2 Menus

The menu is a list of items with operations behind, which the user can chose with the aim to derive certain actions. There are different types of menus, here are some examples: the push–button menu, where the user makes choices over physically separate buttons like "ok"–button or "cancel"–button, or the pull–down menu, usually found in the so–called menubar, where the menu pulls down, when the user clicks onto an item with the mouse, see Figure 3.7 [2].

3.5.3 Forms

Forms are special screens with labeled fields where the user can make some input. There are several possibilities a user can make input into forms [2]:

- User–typed strings, where the user can type any free–form character or a specified syntax is necessary e.g. only numeric input possible
- User choices from a list, depending on a preceding action the system offers a specified list of options where the user can chose his option
- Default values for fields, the functionality of this form will be best explained with an example: a flight online booking system has the function to chose the traveldate. By clicking the calender the system sets a default date which the user can change by clicking through the calender but he cannot type some values into the calender

- Required and optional values, if the form needs a required value, the system will not execute any actions until a valid value is filled in. Whereas optional value must not be filled in - it is necessary to distinguish these two fields by different appearence on a form.
- Dependent values, values have to be entered in dependency to another form or field. e.g. the user marks "married" the form of the date of marriage gets active for the input of the date of marriage.

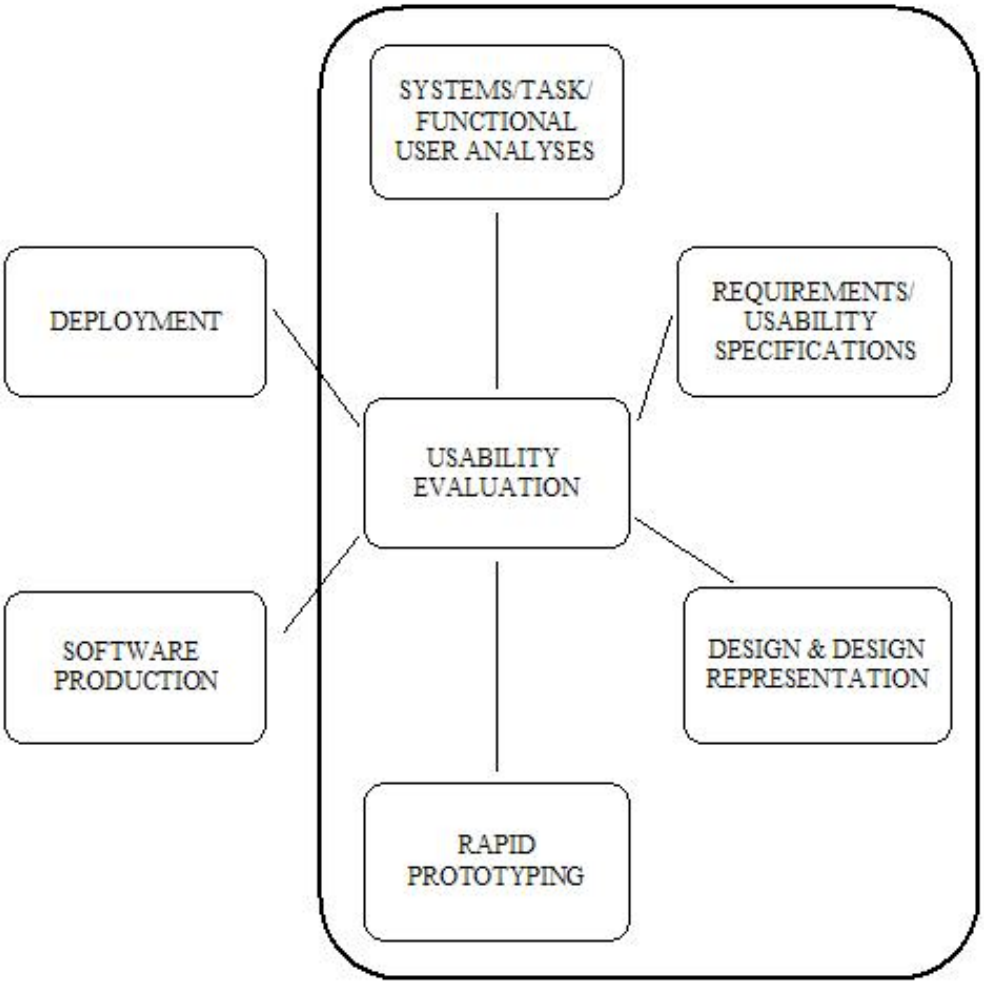

Figure 3.5: The star life cycle for user interaction development [2]

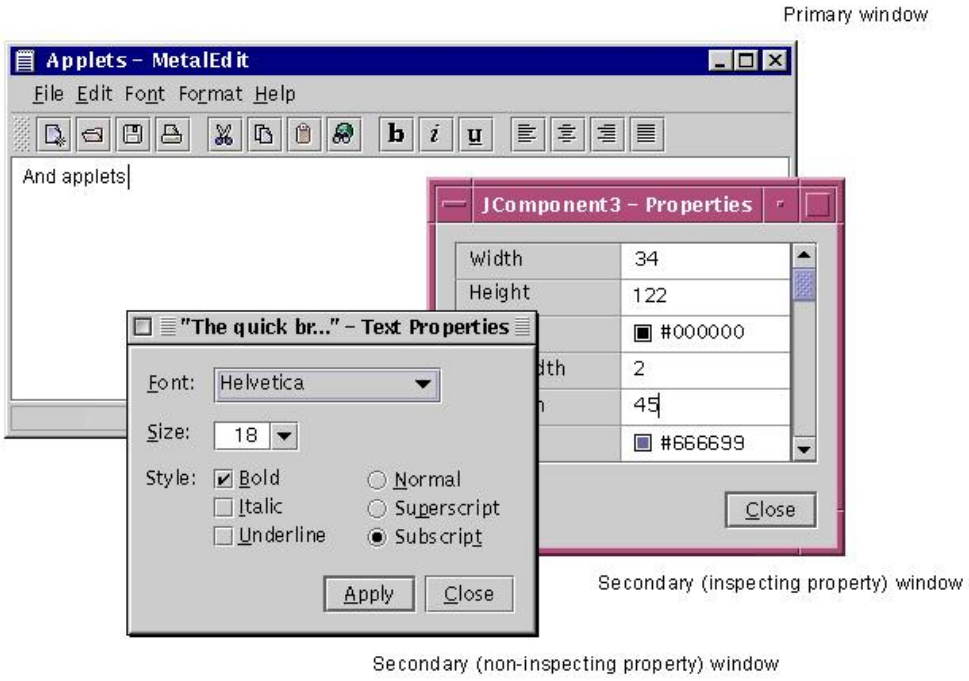

Figure 3.6: The smaller windows are the secondary windows of the large mainwindow [11]

Figure 3.7: An example of a pull–down–menu integrated in a menu bar

Chapter 4

A short overview of wxPython

4.1 Abstract

Primarily the plan was to implement the graphical user interface with the Mozilla Framework. But during the ongoing project we had to recognize that the present possibilities of the chosen toolkit did not fit the essential requirements of the application .

As alternative we changed to wxPython, a Python wrapper for the wxWidgets GUI Library [15].

One attribute of wxPython is it's open source code so that it is free for anyone to use. The available sourcecode can be seen and modified by anyone and anyone can contribute to the open source project.

Furthermore wxPython is a cross—platform toolkit. This feature makes it possible to create programs running on multiple platforms without any modification. Currently it supports the following platforms: 32-bit Microsoft Windows, most Unix or unix-like systems, and Macintosh OS X [15].

4.2 wxWidgets

wxWidgets is a widget toolkit and as easy-to-use API useful for developing GUIs for applications on multiple platforms that also utilize the native platform's controls and utilities. If the application is linked with the appropriate library for the used platform and compiler, it will

adopt the appropriate look and feel of the platform. Therefore, by using these native widgets, the application gains kind of a recognition factor for what the user knows from the platform he is using [16] [17].

The wxWidgets project was founded in 1992 by Julian Smart under the name wxWindows, aiming to create applications portable between Unix and Windows. Since then it was ported to various platforms supporting graphical user interfaces [17].

Apart from the already mentioned GUI functionality, the cross-platform ability, and the open source property these are some other features offered by wxWidgets:

- Help and documentation — it runs a suitable documentation for different formats; with the free add-on documentation utility the developer can integrate help and manuals to their own application and it offers a wide range of samples
- Wide range of widgets — apart from usual controls and windows wxWidgets offers a variety of more advanced classes like the wxMDIParentFrame and wxMDIChildFrame allowing child frames to be viewed inside a main frame.
- Debugging facilities — which offers a memory-checking facility to detect, report and run memory leaks in a debug mode
- Database functionality — it offers a set of ODBC classes or you can even use other librariers as the Xbase library, a free dBase clone
- Drag and drop is supported by most ports
- The availability of wrappers for several scripting languages, causing increasing popularity as GUI scripting tool

For the future can be said that as industry reconsiders dependencies on expensive and closed-source products wxWidgets and wxPython will continue to be developed and improved as instruments to build useful and aesthetic tools for users [17].

4.3 Python

Python is a dynamic object–oriented and interactive programming language which offers a strong support for integration with other languages and tools and offers extensive standard

libraries. It can be run e.g. on Windows, Linux/Unix, Mac OS X, and OS/2. Python is also distributed as an open source license.

Python is said to

- be very clear and having a readable syntax
- have good introspection capabilities
- have an intuitive object orientation
- have an natural expression of procedural code
- offer a full modularity, supporting hierarchical packages
- offer an exception–based error handling
- have a great variety of standard libraries and third party modules

The python project is forced by the Python Software Foundation which holds the licenses and is responsible for future enhancements of the programming language and to keep it available to the public [15].

It has been verified by the develpors of VISIMS, the project at hand, that a given problem can be solved 2–3 times faster with Python than with C++ by an experienced developer.

Chapter 5

The application elaboration according to the user interface life cycle

5.1 The project

The goal of this project was to develop a graphical user interface for VISIMS a software project for the Institute of Chemical Technologies and Analytics at the Technical University Vienna. The aim of the VISIMS project is an application for visualisation, classification, and quantitation of 3D SIMS data.

Analog to the life cycle for User Interface Development **??** several steps have been taken and were redesigned in an iterative proceeding.

5.2 System Analysis

5.2.1 User profile

The TOF–SIMS gear itself is highly technic focused. As there is a workstation connected with the gear, data received from a sample will be directly executed in a complex way. The software being developed starts with its analysis based on these measured data.

The average user of the gear is a high–end scientist from the institute and therefore he is a technically experienced user, used to handle complex tasks. It can be excluded that young unexperienced studendts will work with the application.

5.2.2 Needs Analysis

Although the TOF–SIMS gear offers an integrated solution with an analysing unit, these analysis possess some failings: treating the data is not very comfortable, standards like drag&drop of pictures are very restricted and there are only limited possibilities of 3D-analysis, which is be the mainfocus to implement this application.

5.2.3 Task Analysis

To get an overview of the tasks the user interface has to offer, I made interviews with the recent users of the TOF–SIMS gear. Subsequent will be a list of tasks that have been requested to be implemented.

- the tool assumes a 3D-data range from the original measurement, whereupon peaks will be already selected in the origin dataset
- it has to be possible to chose a specific measurement from the raw data
- when data are imported the tool opens them with a "standard" visualisation
- based on these visualisation it must be possible to step into and to make deep changes
- up to 4-5 visualisation models must be possible to reported, whereas parameters must be modifiable
- it must be possible to save this set of parameters to be available for further analysis
- it is necessary to optimise and focus these parameters
- it shall be possible to cut out cubes of another cube and to present the result within different styles
- displaying several masses at once
- combining data analysed in different visualisation methods within one picture
- it is not allowed to mix masses of different measurements
- implementing a preview window where the user can directly see what happens whenever a parameter is changed
- it mus be possible to analyse multiple measurements simultaneously

- drag&drop of masses into the analysing area
- export visualisations via copy&paste
- safe templates
- displaying a logview
- offering the possibility of multi variant analysis
- execute cluster analysis within an own window
- extracting from 3D visualisations 2D visualisations
- providing information toolbar, where the user can chose further information for the visualisation window (e.g. date of last proceeding)

5.3 Use cases

Based on the information of the interviews I started to analyse, sort, and group the tasks to form use cases. In the following you will find a sample of necessary user cases.

These use cases are a result of iterative discussions about the aim of the application.

Copy and Paste of graphics

Item	*Description*
Description	the visualisation can be exported via 'copy & paste' into other standard applications by clicking on the picture, right mouse button, chosing copy of the menu, selecting the target application, and clicking the right mouse button paste (or even with strg&c and strg&v)
Output	a picture (bmp) of the visualisation in the target standard application
Actor	User
Trigger	the user needs a picture of the visualisation with all its information on it to show it in other standard applications like Microsoft Office Products, e.g. preparing a presentation of the measuring results
Input	a significant visualisation of a measurement with all needed information in it
Precondition	the user defines all further information to be seen in the visualisation window
Postcondition	—
Essential steps	—

Export of Data (Parameter)

Item	*Description*
Description	the result of a visualisation has to be exportable, e.g. as txt- or csv-file, to be applicable in other standard programs as MS Powerpoint or MS Excel. The exact range of exportable data has to be further specified
Output	a txt- or csv-file (standard importable files) or a bitmap of a visualisation for import and further treatments in other applications
Actor	user
Trigger	the need to process the data in standard applications like Microsoft Office Products, e.g. to prepare a presentation of the measuring results
Input	specified data of the resulting visualisation (have to be further detailled)
Precondition	a significant visualisation was finished
Postcondition	—
Essential steps	provide the user different exporting options (csv, txt, bmp)

Preview Window (Change of parameters and selected masses)

Item	*Description*
Description	provide a preview for setting paramters and/or selecting masses that the user can see the changes made
Output	a preview of the effects on the visualisation resulting from the changes of parameters and/or mass selections
Actor	user
Trigger	trigger: the need to optimize the visualisation for a certain purpose
Input	changed parameters and or mass selecetion
Precondition	a visualisation was already selected, the input of parameters and masses is valid
Postcondition	—
Essential steps	—

Saving parameters

Item	*Description*
Description	the parameters specified for a certain visualisation can be saved to be available for further visualisations
Output	a reuseable set of parameters for further (similar) visualisations
Actor	user
Trigger	necessity of reuse of parameters
Input	set of parameters defined from the user for visualisation
Precondition	all parameters are specified for the selected visualisation model
Postcondition	—
Essential steps	—

Parameter setting

Item	*Description*
Description	the user defines the parameter needed for the specified visualisation
Output	a set of parameter necessary for the visualisation
Actor	user
Trigger	user selcets a visualisation model for the chosen data
Input	selected data of measurement selection
Precondition	a selected set of data, user specifies visualisation model
Postcondition	—
Essential steps	defining the necessary parameters for each visualisation model

Standard visualisation

Item	*Description*
Description	after choosing a set of data the system displays the data in a predefined standard visualisation.
Output	a standard visualisation of the data for further treatment
Actor	system
Trigger	—
Input	selected data of measurement selection
Precondition	a selected set of data
Postcondition	—
Essential steps	defining the mostly used standard visualisation

Measurement selection

Item	*Description*
Description	the user chooses a measurement out of a range of existing measurements for further treatment.
Output	data are available for further processing
Actor	system
Trigger	—
Input	original data from the measurement tool
Precondition	a selected set of data
Postcondition	—
Essential steps	defining the mostly used standard visualisation

Chapter 6

The user interface

In the following section you will get an overview of the GUI drafts with their advantages and disadvantages and why or how we stepped forward in developing the user interface.

The first two drafts figure 6.1 and 6.2are quite similar. The main window contains all analysis. An advantage of this style is, that the user will not generate a "window"-overflow on his screen but on the other hand this advantage implicates the disadvantage that the user cannot arrange different pictures parallel to see some differences at a glance.

The prototyping process was a very fast process — probably depending on the technical experience of the users — that the final way to solve the problem with a primary window as basis and analysis will be made in secondary windows was clear within few iterations 6.3.

The figures 6.4, 6.5, and **??**show the main frame in an early stage of development.

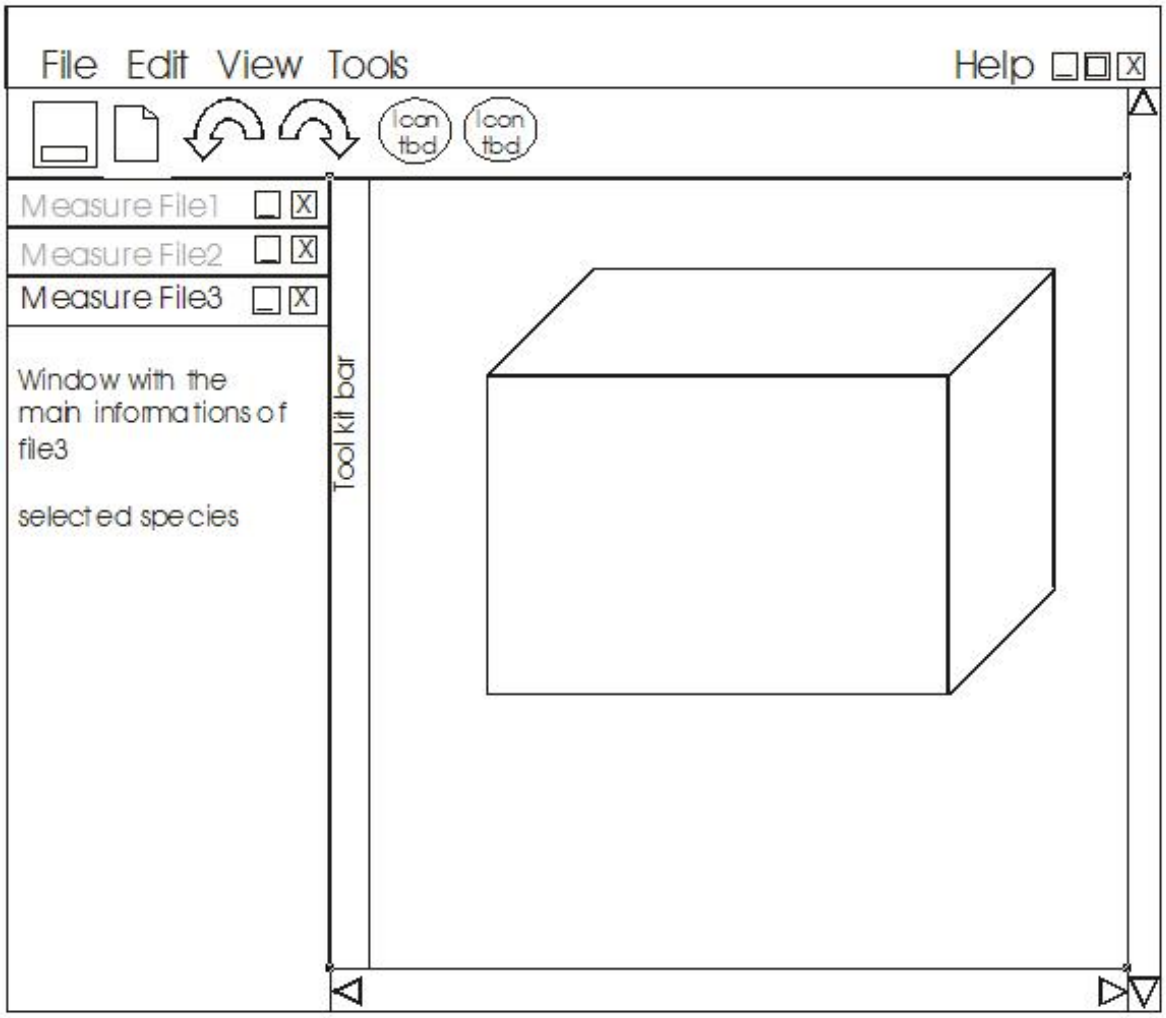

Figure 6.1: The first draft

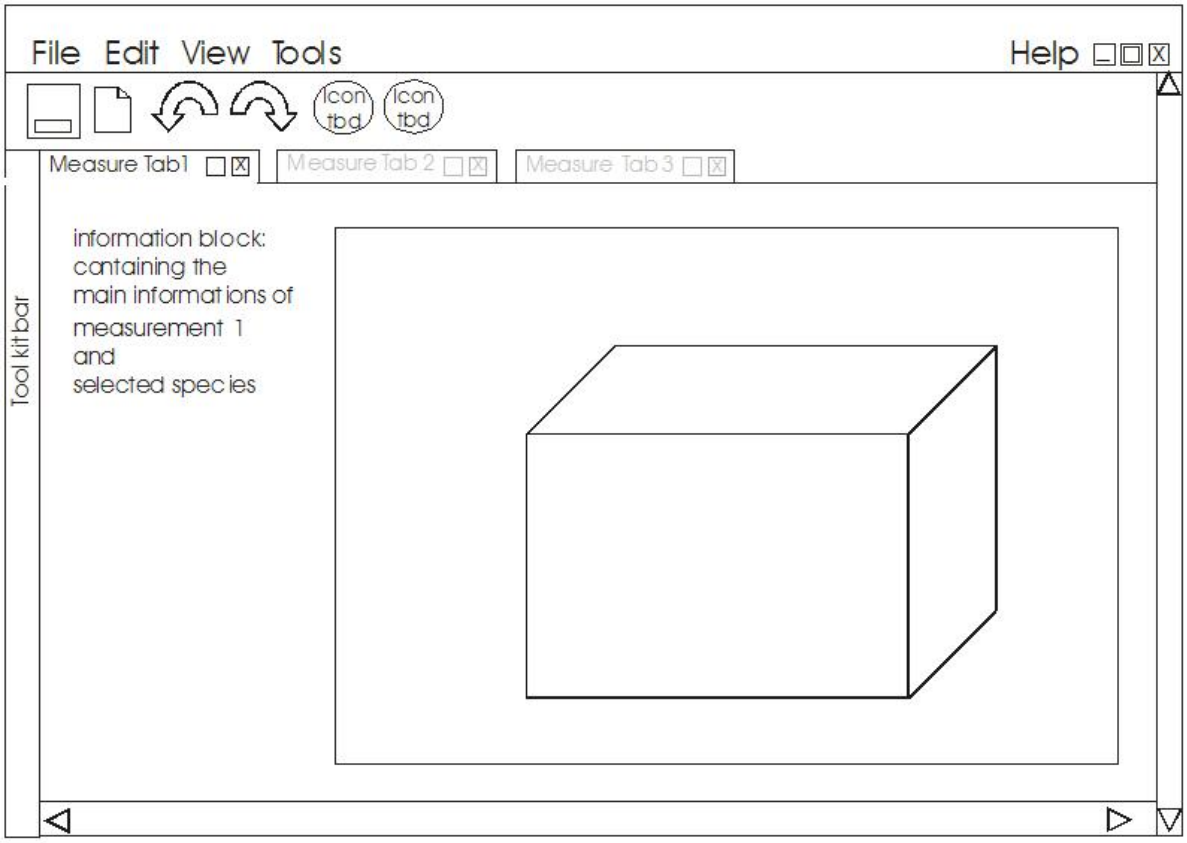

Figure 6.2: The next draft of a gui design

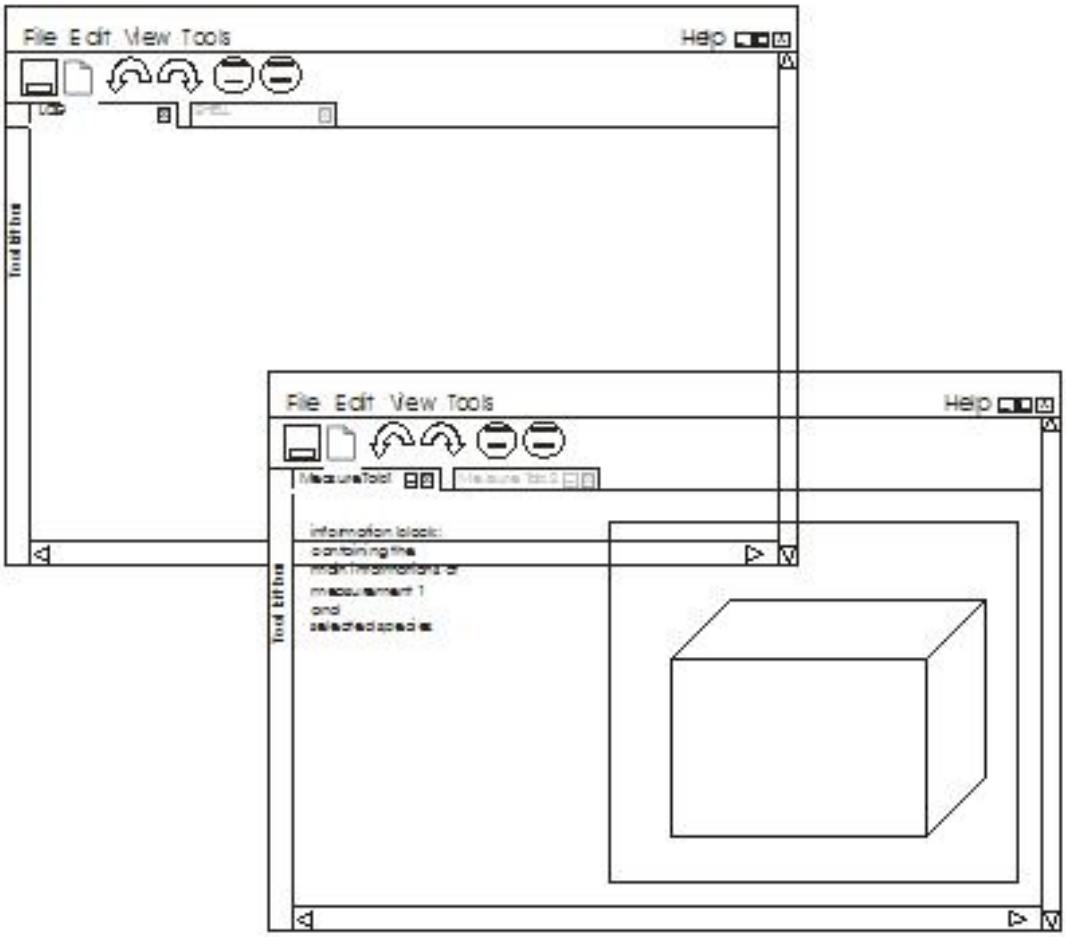

Figure 6.3: The final draft

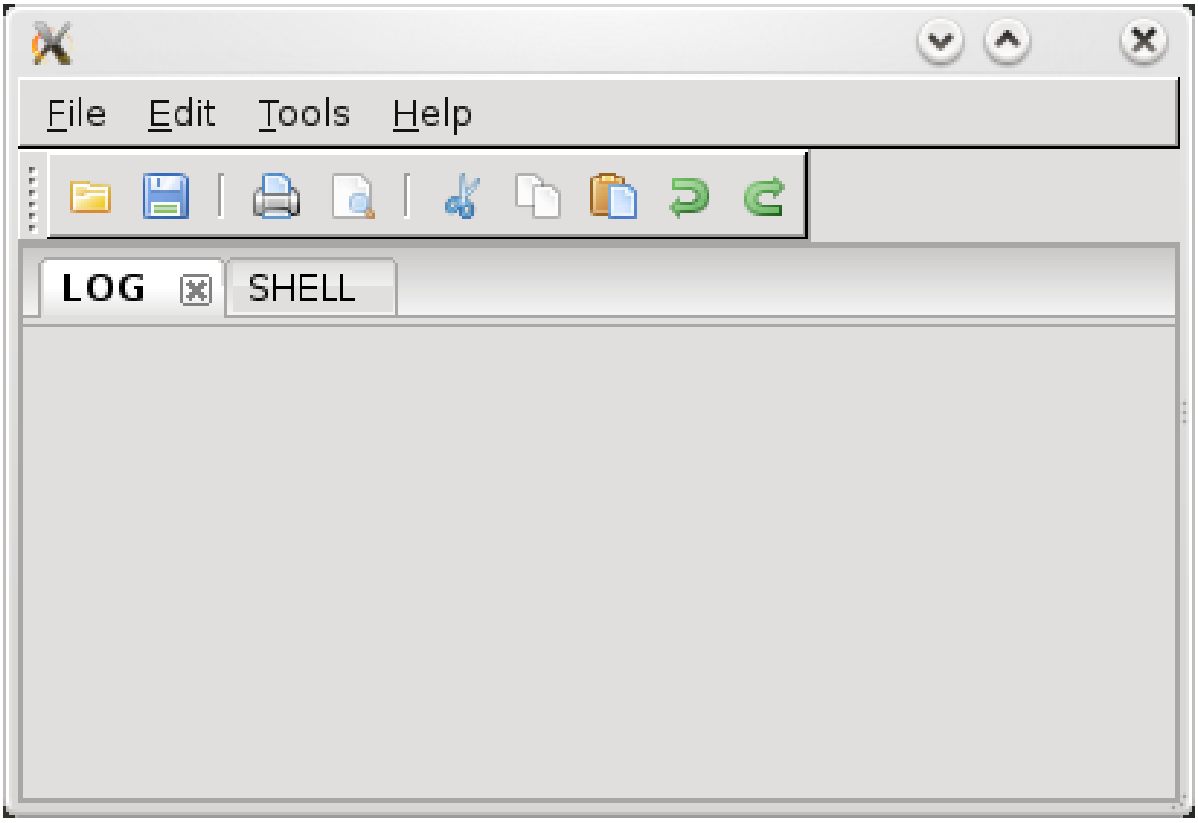

Figure 6.4: The main-window of the application

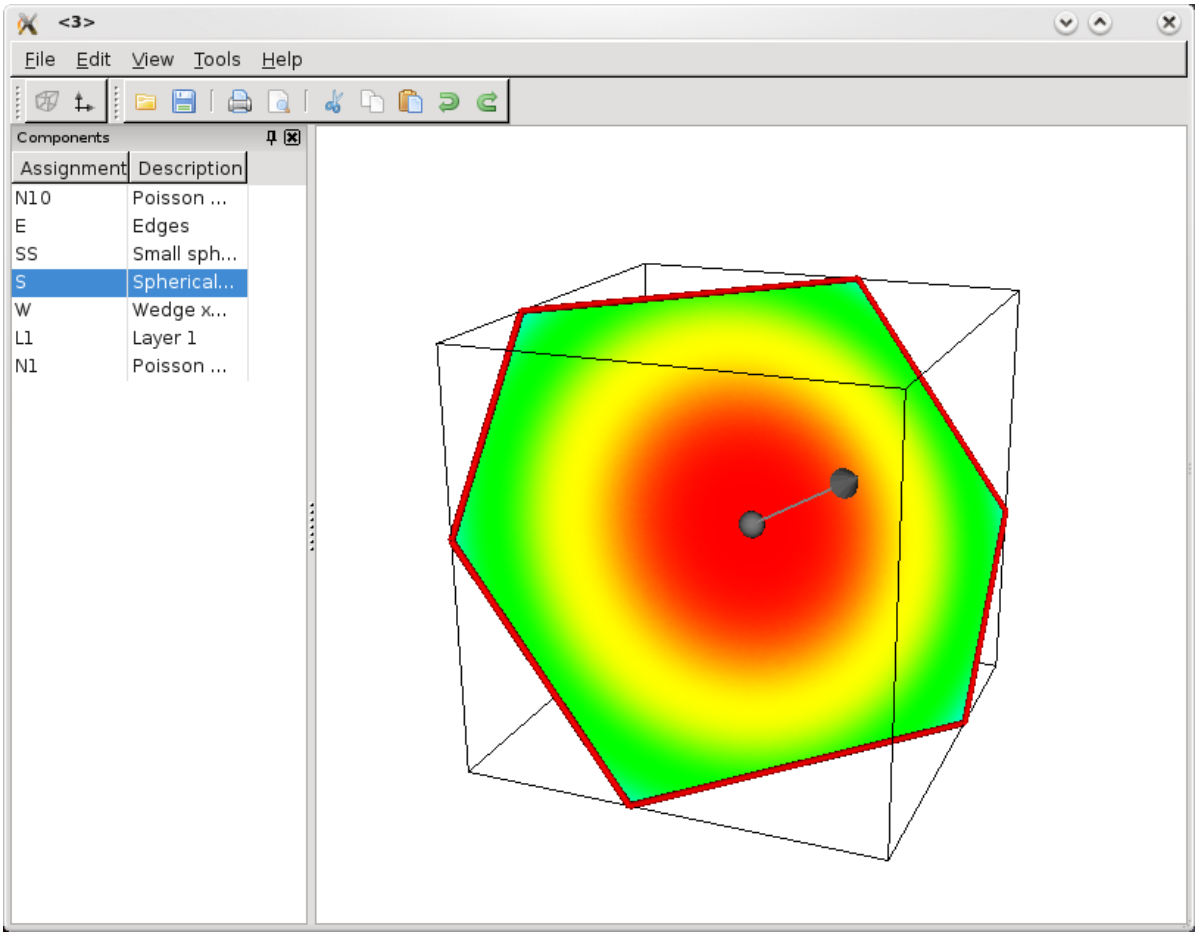

Figure 6.5: The planned render-window

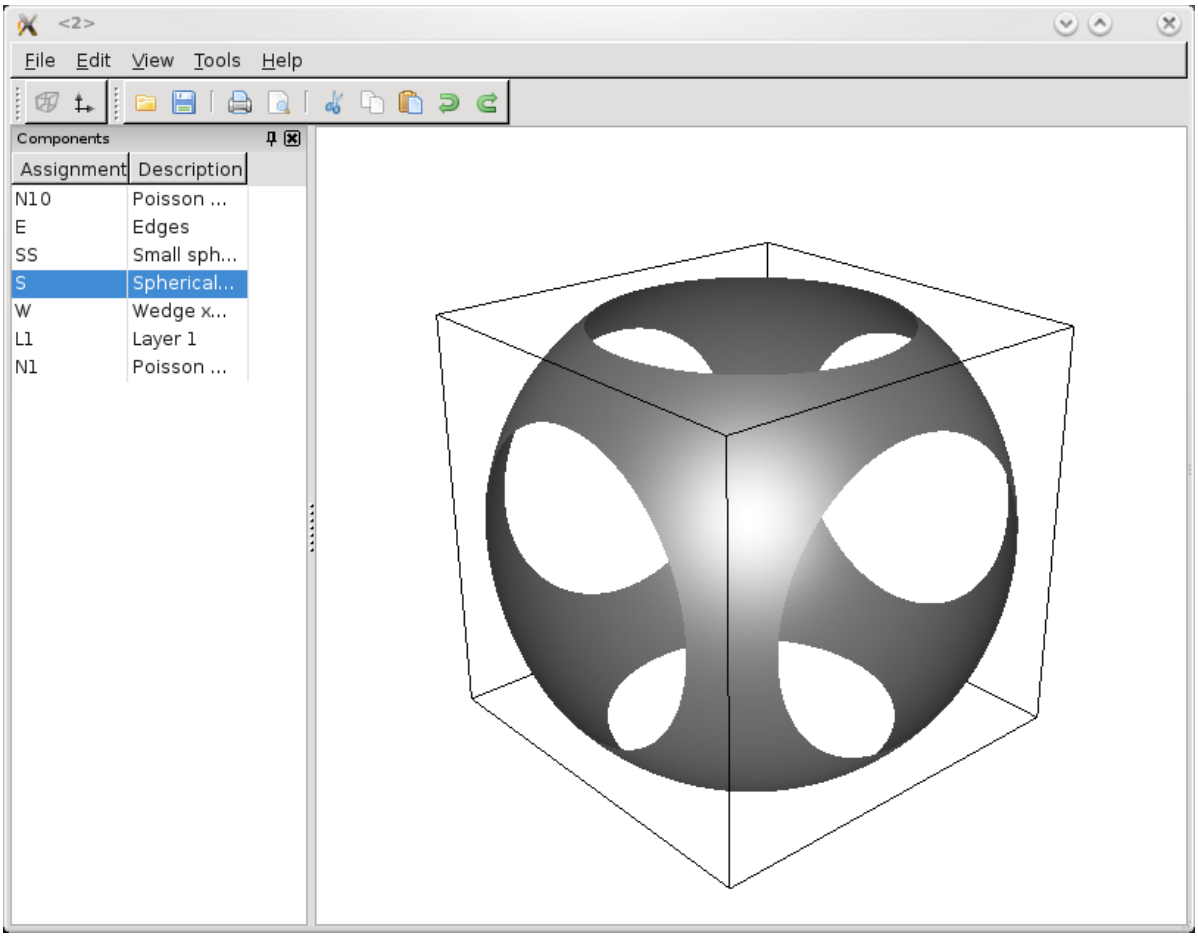

Figure 6.6: The render-window with an Isosurface

Chapter 7

Bibliography

[1] Wilbert O. Galitz. *The Essential Guide to User Interface Design*. Wiley Computer Publishing, second edition edition, 2002.

[2] Deborah Hix and H. Rex Hartson. *Developing User Interfaces*. John Wiley & Sons, Inc., 1993.

[3] D-48149 Münster ION-TOF GmbH. Secondary ion mass spectrometry. Website. Available online at http://www.iontof.com/technique-sims-IONTOF-TOF-SIMS-TIME-OF-FLIGHT-SURFACE-ANALYSIS.htm; visited November 2008.

[4] D-48149 Münster ION-TOF GmbH. Staying ahead. Website. Available online at http://www.iontof.com/applications-IONTOF-TOF-SIMS-TIME-OF-FLIGHT-SURFACE-ANALYSIS.htm; visited November 2008.

[5] D-48149 Münster ION-TOF GmbH. Staying ahead. Website. Available online at http://www.iontof.com/technique-depthprofiling-IONTOF-TOF-SIMS-TIME-OF-FLIGHT-SURFACE-ANALY
htm; visited November 2008.

[6] D-48149 Münster ION-TOF GmbH. Staying ahead. Website. Available online at http://www.iontof.com/technique-retrospectiveanalysis-IONTOF-TOF-SIMS-TIME-OF-FLIGHT-SURFAC
htm; visited November 2008.

[7] D-48149 Münster ION-TOF GmbH. Time-of-flight mass spectrometer. Website. Available online at http://www.iontof.com/technique-timeofflight-IONTOF-TOF-SIMS-TIME-OF-FLIGHT-SURFACE-ANALYSI htm; visited November 2008.

[8] Clayton Lewis and John Rieman. Task–centered user interface design. shareware notice. Available online at ftp.cs.colorado.edu; download May 1993.

[9] Klaus Nowikow. Visualisation of three-dimensional secondary ion mass spectrometry data, 2000.

[10] US-Northfield MN 55057 Science Education Resource Center, Carleton College. Time-of-flight secondary ion mass spectroscopy (tof-sims). Website. Available online at http://serc.carleton.edu/research_education/geochemsheets/techniques/ToFSIMS.html; visited November 2008.

[11] 2001 Sun Microsystems, Inc. Property windows. Website. Available online at java.sun.com/products/jlf/at/book/Windows6.html; visited November 2008.

[12] last update 1998 Talin. A summary of principles for user-interface design. Website. Available online at http://www.sylvantech.com/~talin/projects/ui_design.html; visited November 2008.

[13] Manfred Tscheligi. User interface design. Skriptum zur Vorlesung, 1991.

[14] Ivo Wessel. *GUI–Design*. Hanser, 2., aktualisierte und bearbeitete auflage edition, 2002.

[15] wxPython Community. What is wxpython? Website. Available online at http://www.wxpython.org/what.php; visited November 2008.

[16] wxPython Community. wxpython 2.6 migration guide. Website. Available online at http://www.wxpython.org/migrationguide.php; visited November 2008.

[17] The wxWidgets Team. What is wxwidgets? Website. Available online at http://www.wxwidgets.org/; visited November 2008.