January 29, 2002
Seminar Software Engineering

ImmoWarrior

Report

e-Immo

Fimberger Lucia
redwing@utanet.at

Nidetzky Marion
mnidetzk@cosy.sbg.ac.at

Rescher Robert
rrescher@cosy.sbg.sc.at

Strobl Elisabeth
estrobl@cosy.sbg.ac.at

Schobesberger Oskar
oschobes@cosy.sbg.ac.at

Academic Supervisor Mag. Dipl.-Ing. Schwaiger Roland

Department of Computer Science
University of Salzburg

Correspondence to:
Universität Salzburg
Institut für Computerwissenschaften und Systemanalyse
Jakob-Haringer-Straße 2
A-5020 Salzburg
Austria

Contents

Chapter 1

Introduction

1.1 Terms of Reference

A real-estate auction firm $e - Immo$ wants a new software for processing the collection of information and of the representation of information. This processing should be simple. The users of this system are the collector of information and the inquirer, who can be a person or another system. One demand is, that the system is simple to service, by meaning, it should not be necessary to install the collection- and representation-components local. Furtheron, $e - Immo$ should have the possibility to influence the processing of information for individuals.

1.2 Development Progress

1. System Requirements
2. CRC Cards
3. Use Cases
4. Essential User Interface Prototyping
5. Class Diagram
6. User Interface Flow Diagram

Chapter 2

Problem Description

2.1 System Requirements

- Collecting information (in our case real-estate).
- Possibility to enquire information.
- Real-estate collection.
- No local installation of components.
- User has to be identified.
- Different access rights.
- Processing of representation.
- Auction is online.
- User's administration.

Chapter 3

CRC Cards

3.1 Short Introduction into CRC

Founder are Beck and Cunningham (ca. 1989).
A $Class - Responsibility - Collaboration - Session$ concentrates on modelling a CRC-model. A CRC-model defines a class for each part of a problem that should be solved. A class consists of its responsibilities, which are called the *properties* and the *methods*. A property is a non-operatable attribut, it is always an instance of another data type, which can also be a complex type or a class. The class-to-class relation is called collaboration. A method is an operating routine that can be called.
CRC-models are made for abstracting the reality by grouping functional units. These functional units communicate with each other. Usually, CRC-models have hierachical structures. It's not usual that two classes communicate circularly with each other.
The members of a CRC-session are a project leader, some collaborators and perhaps a supervisor. The CRC concept seperates a problem into several classes, which has the advantage that developers do not need to know the functional structure of the whole, but of only one class. Finally, the classes commuicate with each other via defined interfaces, and it's equal to the calling class, how the rest of the called class is structured and implemented.

3.2 Each individual CRC Card

1. <u>Real-Estate</u>

   responsibilities — collaboration

   - knows its characteristics

2. <u>Real-Estate Collection</u>
   responsibilities — collaboration

   - knows the real-estates .... Real-Estate

- edits real-estates .... Recorder
- returns real-estates .... Enquirer

3. User

   responsibilities — collaboration

   - knows its characteristics

4. User Collection

   responsibilities — collaboration

   - knows users and their access rights .... User, Access Rights
   - edits users and their access rights
   - passes information about user and his rights .... System Interface

5. Access Rights

   responsibilities — collaboration

   - knows its characteristics

6. System Interface

   responsibilities — collaboration

   - offers login .... Representation Rrocessor
   - checks login data .... User Collection
   - enables request or record of data .... Representation Processor
   - transmits data .... Recorder, Enquirer

7. Representation Processor

   responsibilities — collaboration

   - receives preprocessed data .... Enquirer
   - passes query related real-estates .... Terminal
   - displays masks for login, input and output

8. Terminal

   responsibilities — collaboration

   - receives data .... Representation Processor

- submits data .... Representation Processor
- displays data

9. <u>Enquirer</u>

   responsibilities — collaboration

   - requests .... Real-Estate Collection
   - gets required information about real-estates .... Real-Estate Collection
   - returns information .... System Interface

10. <u>Recorder</u>

    responsibilities — collaboration

    - makes possibility for input available .... System Interface
    - passes completely collected real-estates .... Real-Estate Collection
    - accepts information about real-estates

Chapter 4

Use Cases

4.1 What is a Use Case

A *Use-Case* describes a possible application of the system. Therefore, we need to know what the system offers, and what the system can be done with. Furtheron, a use case can be subdived into another or a number of other use cases. This encapsulation is often need to distinguish several possibilities without copying a part of another use case (as we also could see when creating classes).

4.2 Each individual Use Case

First of all, the main use cases, which are to input and enquiry real-estates.

1. **Record Real Estate**
   Description: User can record (input) a new real-estate.
   Precondition: User is logged in and has the right to record real estates.
   Postcondition: A new real-estate is stored in the system.

   - System presents input-mask.
   - User inputs data and submits them.
   - System verifies data and asks the user to recheck it.
     Okay stores data.
     Cancel: Back to input and submit data.

   At any time:

   - Cancel: Presents reset choice mask.
   - Logout: Presents login mask.

2. **Enquiry Real Estates**
   Description: User can make a query for one or a group of real-estates.
   Precondition: User is logged in.
   Postcondition: User receives data refering to his security level.

- System presents enquiry mask.
- User chooses searching criteria from several possibilities.
- System searches real-estates refering to the query and the security level. The found real-estates are presented the user.

At any time:

- Cancel: Presents reset choice mask.
- Logout: Presents login mask.

These were the main use cases. As one can see, they contain parts which are equal in some cases. On this point, the use cases are subdivided into the sub-use-cases.

1. **Login**
   Description: Logs in a user. The login also defines the user's security level and access rights.
   Precondition: None.
   Postcondition: User is logged in.

   - System presents login mask.
   - User inputs his identification and password.
   - System verifies his input and logs him in.

   At any time:

   - Cancel: Presents reset login mask.

2. **Action Choice**
   Description: User chooses *Record* or *Enquiry*.
   Precondition: User is logged in and has the right to record data.
   Postcondition: System presents enquiry- or record-mask.

   - System presents this choice mask.
   - User chooses either *Record* or *Enquiry*.
   - System presents the mask in compliance to the user's choice.

   At any time:

   - Logout: Logs out the user and presents the login mask.

3. **Participation at the Auction**
   Description: User has the possibility to take part at the auction and to bid.
   Precondition: User is logged in, has the right security level, has found the enquired real-estate.
   Postcondition: User holds highest offer.

- System displays the details of the enquired real-estate and offers the user the possibility to take part at the auction.
- User bids.
- System asks the user to confirm his bid.

  User confirms: System actualizes score and refreshes the display.

  User cancels: System returns to the detailed view of the real-estate.

At any time:

- Cancel: Presents searching results.
- Logout: Presents login mask.

Chapter 5

Essential User Interface Prototyping

Here is a rough draft for two versions of the user interfaces. They meet the compliances of the use cases. The following six ”screen-images” are thought to be useful for an HTML version.

Login

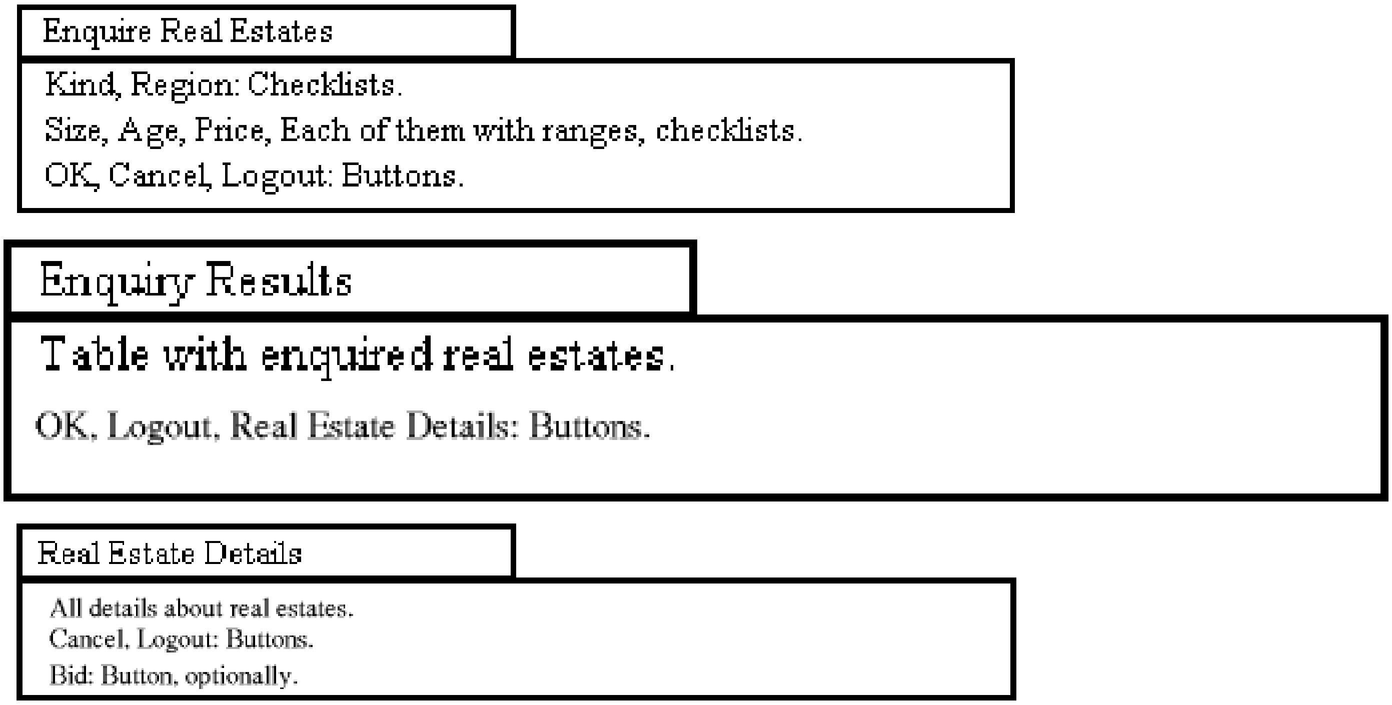
Name: Edit field.
Password: Edit field without displaying the input.
OK, Cancel: Buttons.

Action Choice

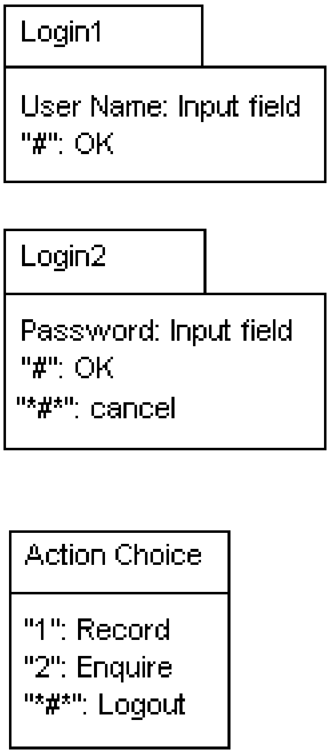
Action: Radiogroup containing: * Record, * Enquire.
OK, Logout: Buttons.

Record Real Estate

Name, Description, Address: Edit fields.
Size, Number of Roos, Age, Base Price, Steps: Numberic edit fields.
Kind: Combobox
User: Key.
Picture: Picture.
OK, Cancel, Logout: Buttons.



This was the user interface prototype for an HTML version. We created another protoype which should be suitable for using the system on a mobile phone. There are more screens, of course, because the size of a mobile phone display is expected to be far smaller.


Login1
User Name: Input field
"#": OK
Login2
Password: Input field
"#": OK
"*#*": cancel
Action Choice
"1": Record
"2": Enquire
"*#*": Logout

Enquire Real Estate 1 (Kind)

"1": Flat
"2": House
...

"*": Cancel
"*#*": Logout

ERE 2 | ERE 3 | ERE 4 | ERE 5 | ERE 6

Equivalent to ERE 1: Enquire Real Estate 2-6,
(for Region, Size, Age, Price)

Query Results 1

Table of results (1-5)
"1": Details
"2": Details
...

"0": Results (6-10), optionally
"*": Cancel
"*#*": Logout

optionally: Query Results 2-...
Equivalent to Results 1

Record Real Estate 1

Name: Edit field
"#": OK
"*": Cancel
"*#*: Logout

RRE 2 | RRE 3

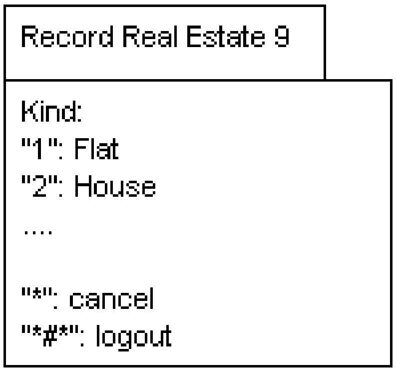
Record Real Estate 2-3:
Equivalent to Record Real Estate 1 are two other sreens
(for input of Description and Adress)

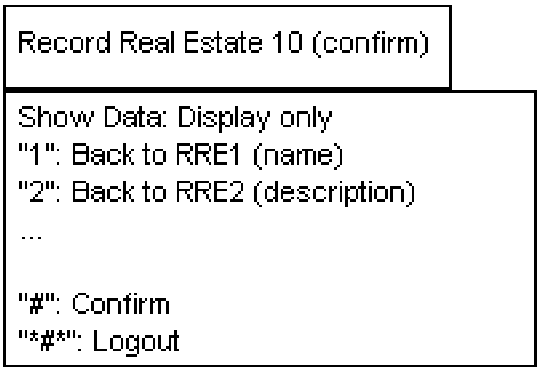
Record Real Estate 4

Size: Numberic Edit field
"#": OK
"*": cancel
"*#*": logout

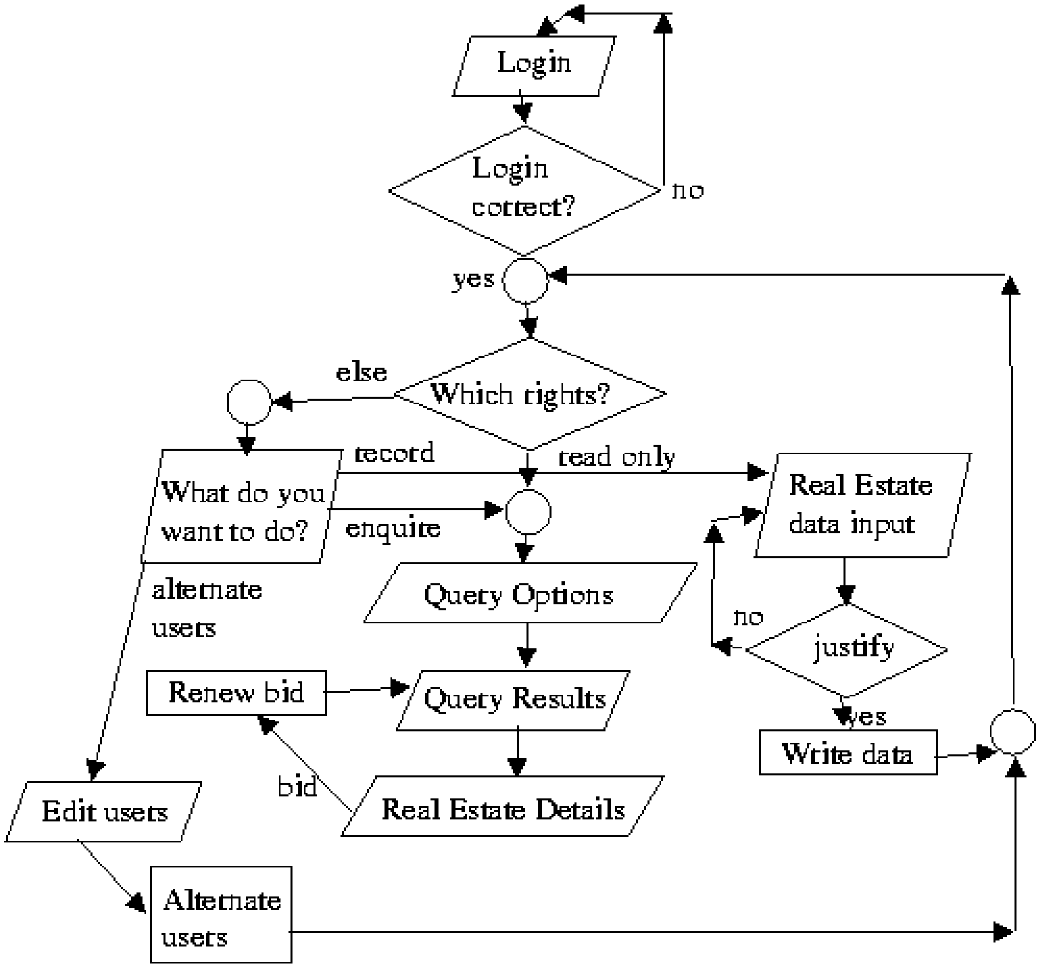
RRE 5 | RRE 6 | RRE 7 | RRE 8

Record Real Estate 5-8:
Equivalent to Record Real Estate4 are 4 more sreens
(for input of Number of Rooms, Age, Base Price, Bid Steps)

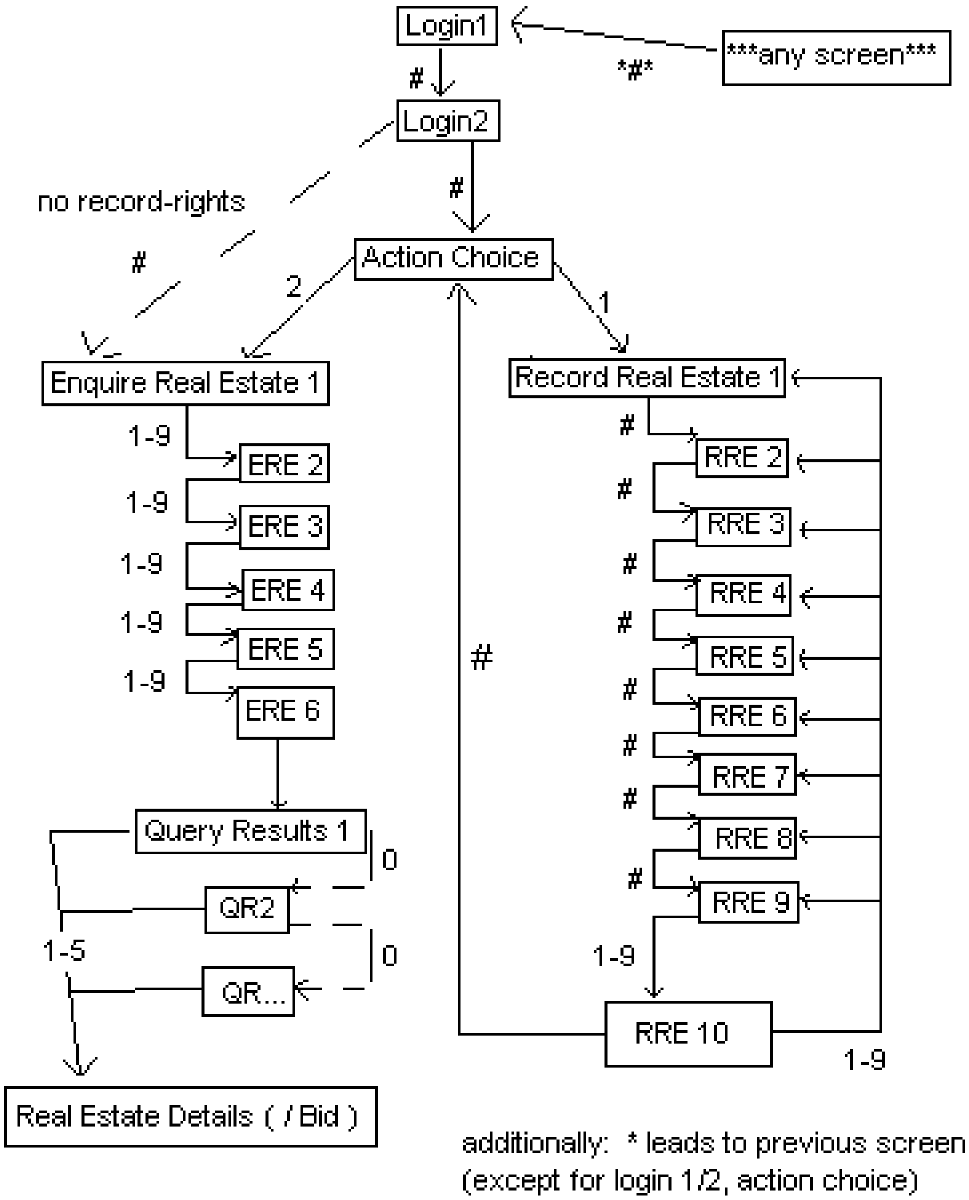




Now we know that the names of the masks, and we can put them into a flow diagram. First there is a flow diagram for HTML display.



Each input field (the ones with the slant side lines) is a mask from the user interface prototype, which also means that the numbers are equal. This diagram is drawn in the usual way we learned to draw flow diagrams. The followin flow diagram (for the mobile version) is quite different, drawn in another possible way to make user interface flow diagrams, which we found by the help of the internet.



Now we see, where the masks have to be put into.

Chapter 6

Class Diagram

6.1 What is a Class Diagram

The use of a class diagram is to document the static structure of the system, by meaning what classes are given and in what kind of relationship they have. The CRC cards, the essential user interface and the use cases are the basis for the class diagram.

6.2 Class Diagram

6.2.1 Description of the Class Diagram

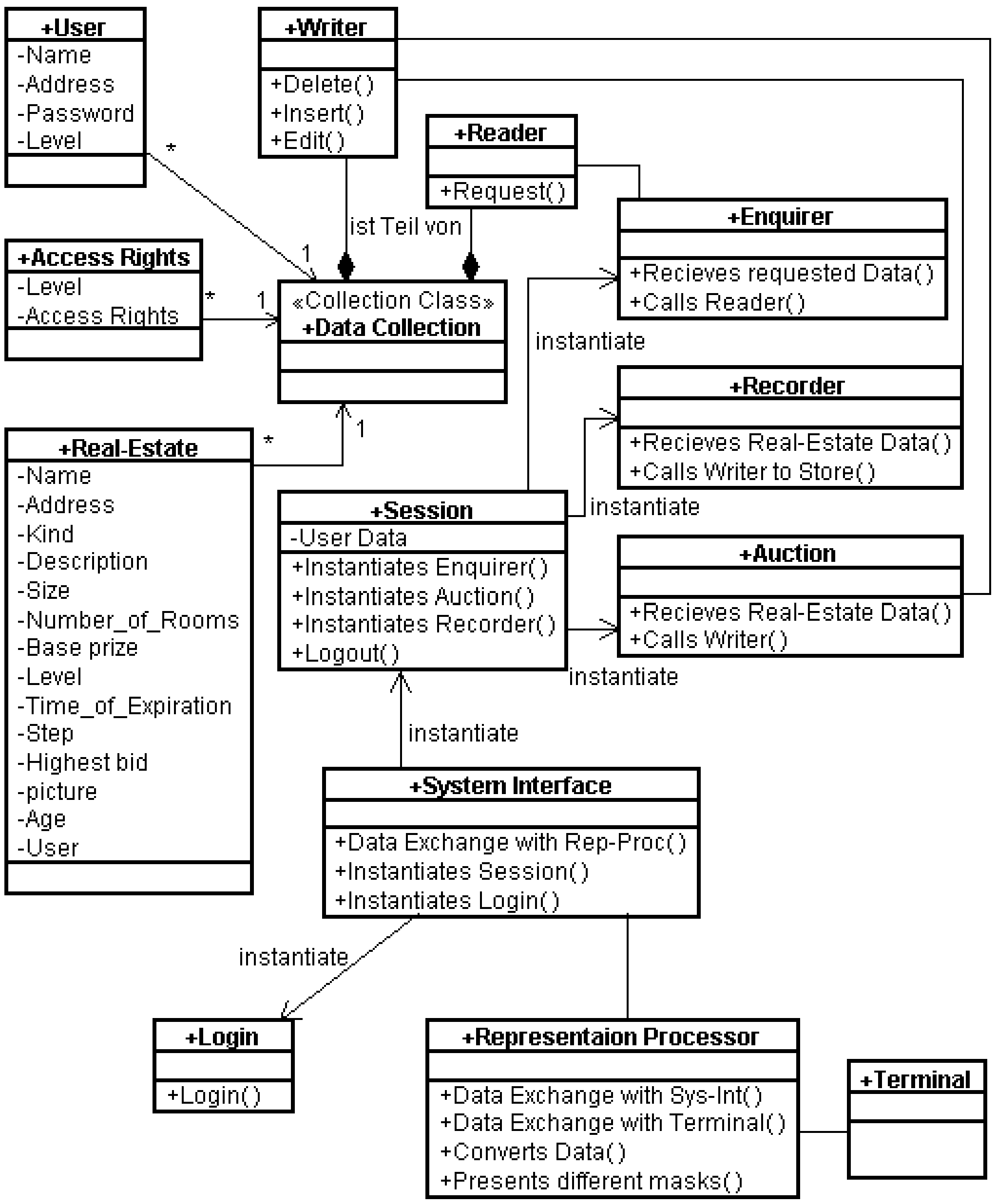
The Representation Processor knows from the System Interface the Session (e.g. number) and presents the login mask. The classes Login and Session, which is waiting for complet and correct login, are instantiated. The user writes the login, then the System Interface gets these information from the Representation Processor. Further on, the user submits the login and his data are returned to the System Interface, which passes the information to the Session. Then the Session opens.
The actions (Auction, Enquiry, record) the user can do depend on his level.

Record The Session passes the information to the System Interface and further to the Representation Processor to present the record mask. The recorder gets the informationindirectly from the Terminal ans calls the proper method of the Writer by justification.

Enquiry Like Record, calls the Recorder's method enquiry.

Auction Makes query. Then it opens a link for auction mask. On justification, Auction calls the Writer to alter the bid-state of the real-estate.

6.2.2 Diagram



+User
-Name
-Address
-Password
-Level
+Writer
+Delete()
+Insert()
+Edit()
+Reader
+Request()
+Access Rights
-Level
-Access Rights
ist Teil von
«Collection Class»
+Data Collection
+Enquirer
+Recieves requested Data()
+Calls Reader()
instantiate
+Recorder
+Recieves Real-Estate Data()
+Calls Writer to Store()
+Real-Estate
-Name
-Address
-Kind
-Description
-Size
-Number_of_Rooms
-Base prize
-Level
-Time_of_Expiration
-Step
-Highest bid
-picture
-Age
-User
+Session
-User Data
+Instantiates Enquirer()
+Instantiates Auction()
+Instantiates Recorder()
+Logout()
instantiate
+Auction
+Recieves Real-Estate Data()
+Calls Writer()
instantiate
instantiate
+System Interface
+Data Exchange with Rep-Proc()
+Instantiates Session()
+Instantiates Login()
instantiate
+Login
+Login()
+Representaion Processor
+Data Exchange with Sys-Int()
+Data Exchange with Terminal()
+Converts Data()
+Presents different masks()
+Terminal

Chapter 7

Control Diagrams

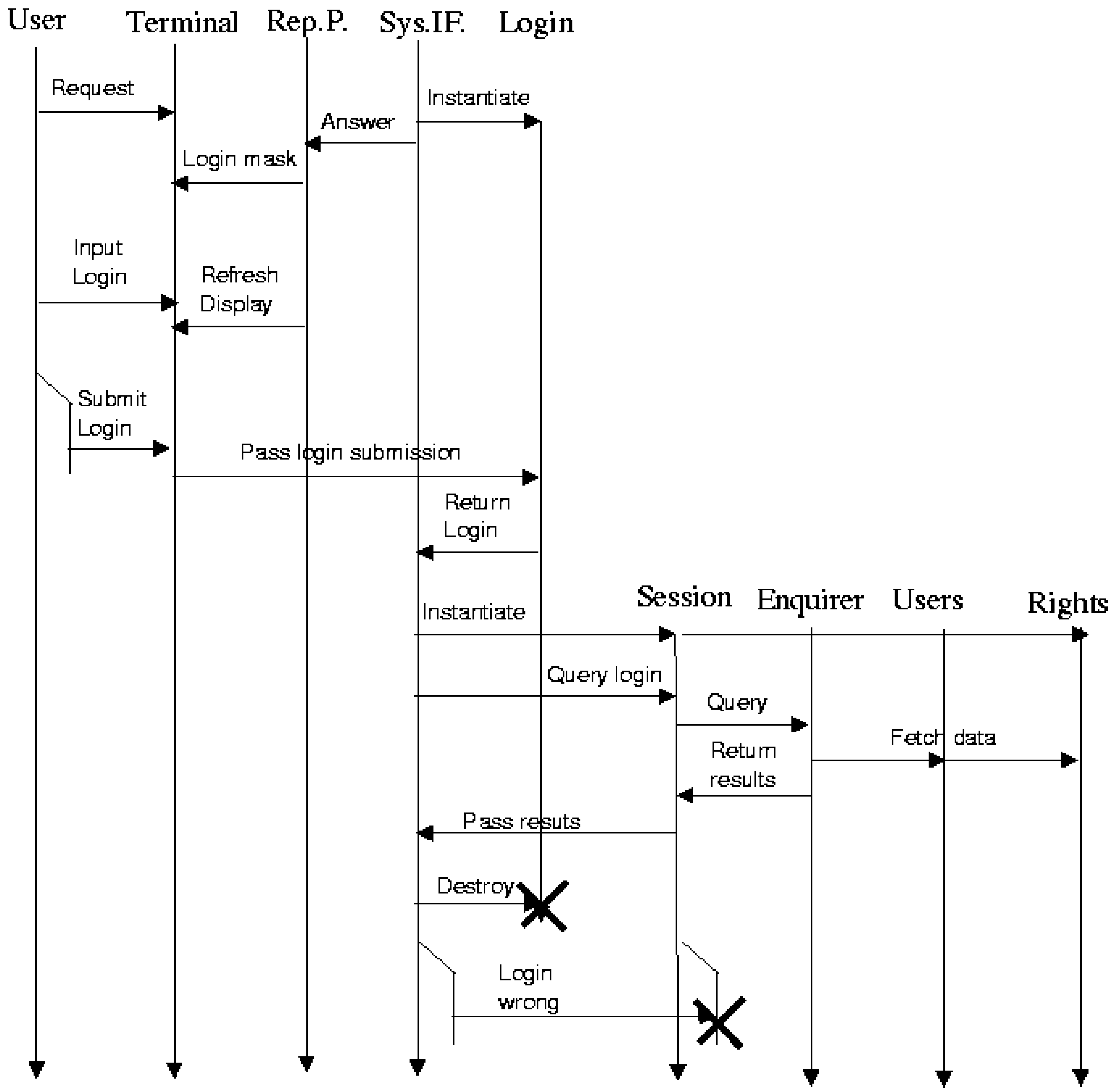
Here we have two kinds of diagrams that show the control flow of our application. On the one hand, there is the *Sequence Diagram* where we will see exactly the system calls. This means that the sequence diagram also will unveil how the implementation of the classes and their methods will be. In the *State Diagram* we will see only the shapes of what the system will do internally, the emphasis lies on the negotiations between user and system.

7.1 Sequence Diagram

First of all, we want to have a view on the sequence diagrams of our *e-Immo*-project.

7.1.1 Login

The login case looks a bit complicated, since there are the instantiations of the user classes. We have a permanent system interface, whilst login and session are instantiated for each user. The database has its interfaces instantiated in the declaration of the class *Session*. Each time, a user forces a session to be opened, the session is instantiated in the main method of the *System Interface* locally. We need to instantiate the session in any case, even though the login could fail. The reason for it is that the *Login* does not have instantiations of the database in its class declaration. Hence, it passes its request to the System Interface, where the session is instantiated. Via an aggregation call, the System Interface can access the database to fetch the data for login check. The check needs two entities of the database, namely *Users* and *Access Rights*; the fetch must be realized by a natural join. In any case, the instance of Login (also locally in the previously mentioned method of the System Interface) is destroyed. If login fails, also the Session will be destroyed; otherwise, the session can open.

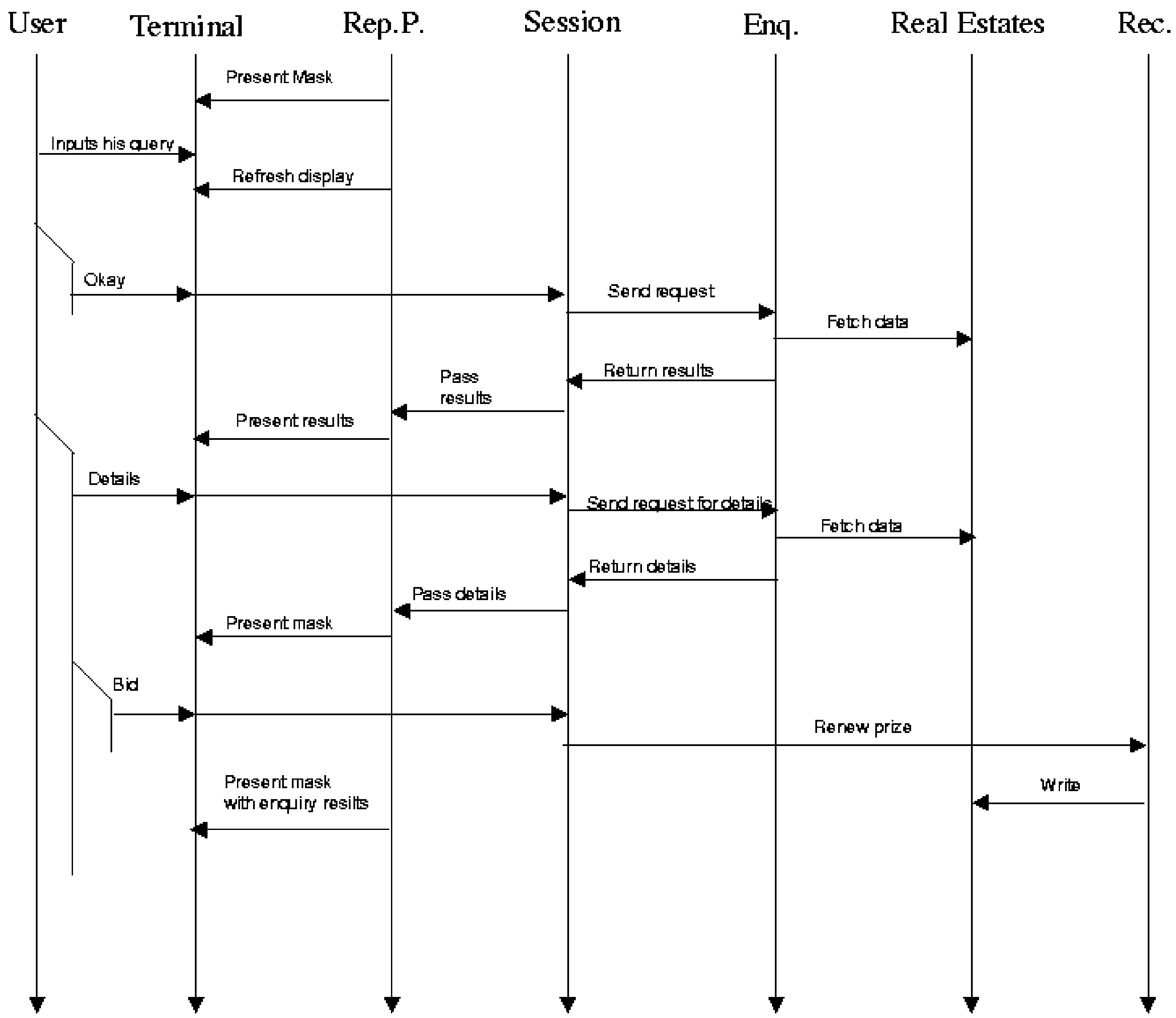


7.1.2 Enquire Real Estate and bid

If one has understood, how enquiring and bidding works, the most difficult work for understanding this system is done. In there, we have al we need in future cases: Read from and write to the database. In here, we have the only one entity, namely *Real Estates*. All the things that happen between user and Session is equal to the negotiations in Login between user and System Interface. We do not need the System Interface anymore, it must be eased of work, because it has to manage all the users that can be logged in simultaneously.

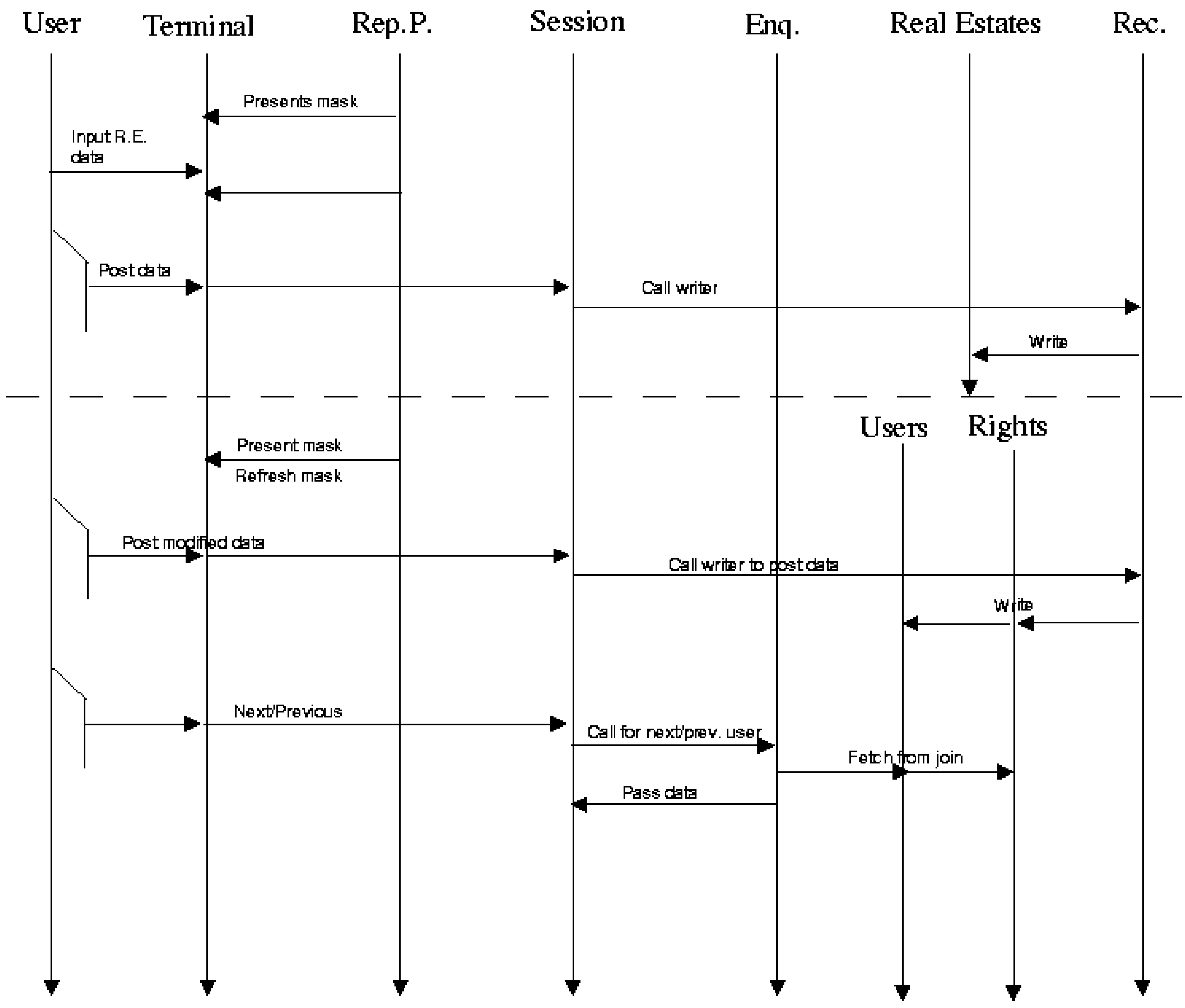
Any time, does not matter if submit query or require view of details, the signal is passed from the Terminal to the Session. The Session-object formulates the request for the database, which might be done in the language SQL, and calls the Enquirer's only method to fetch the data. The fetched data are returned to the Representation Processor. This must be done via the interface of the called method in the Session, otherwise we would break the principle of aggregation. By that time, I want to remark that every return must be done as the return value of a method due to this.

To bid for a real estate runs a bit different. We do not need a return value, and we have to call the recorder. We have a fix step width, so the user does not need to input a step.



7.1.3 Record Real Estate and User Administration

To record a real estate is not complicated. It is only a simple transaction. A bit more is to regard in the case of the *User Administration*. Here we have to call both, the reader and writer. We need the reader to browse forward and backward; and we need the writer to post the modified data. Here it is not necessary to show in which way we can alternate users — we can modify, add and delete a user. However, there is a cycle, namely present and refresh refers to that cycle. Each time, something has been done, the sequence needs to restart here.



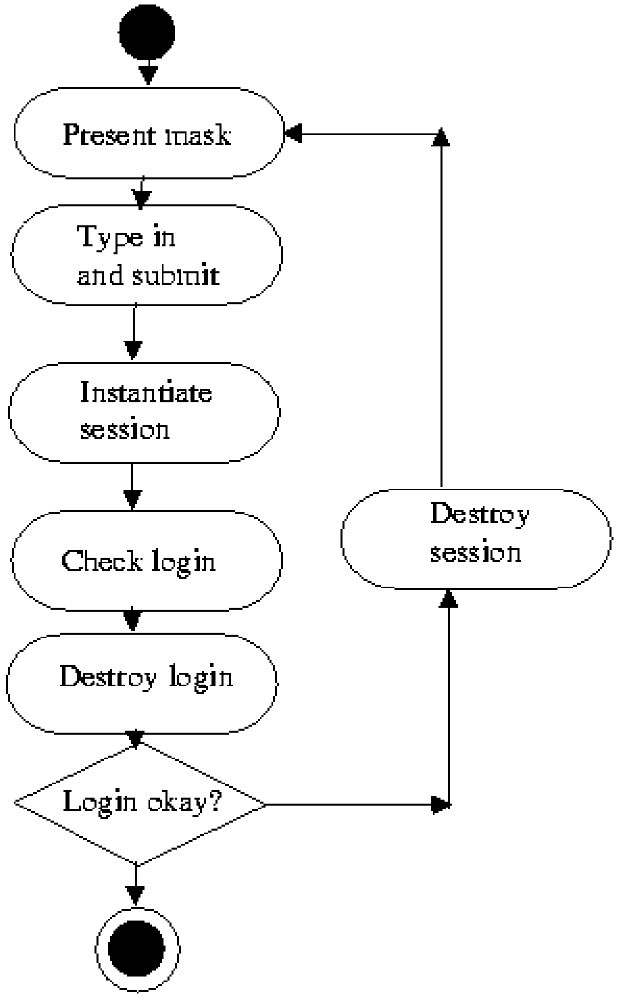
On the top, we see the Record case, on the bottom there is the User Administration.

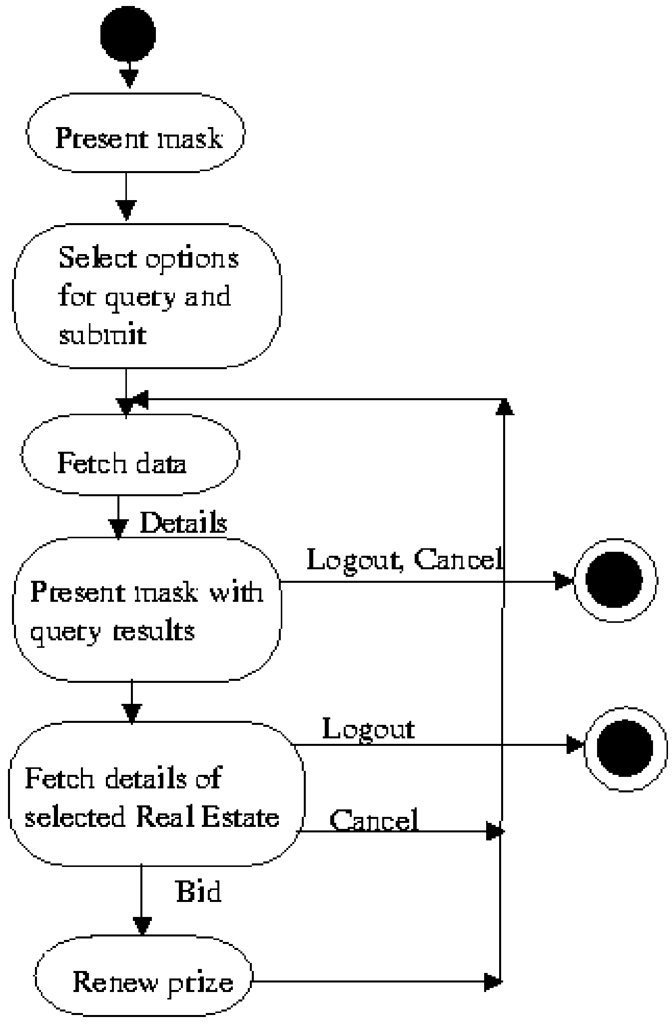
7.2 State Diagrams

Now we want to have a view on the state diagrams. Here it is not very interesting how the application processes the actions, but which states it runs through. They look like flow diagrams, but in flow diagrams the states are not important; whilst in state diagrams the actions done on the interfaces are not so important.

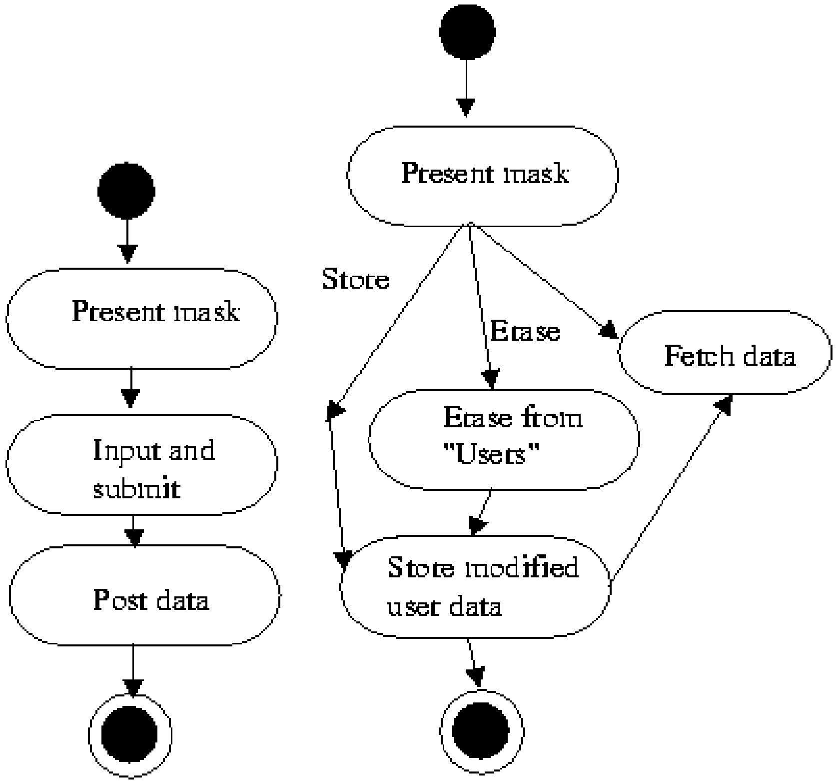
7.2.1 State Diagrams for *e-Immo*

Let us have a look on the state diagrams to the project.





The diagram for enquirying and bidding looks rather complicated. We have two exit points. Furtheron we have to regard that this is only a state diagram. Here it is not interesting if the user can bid at all, but what happens if he bids.



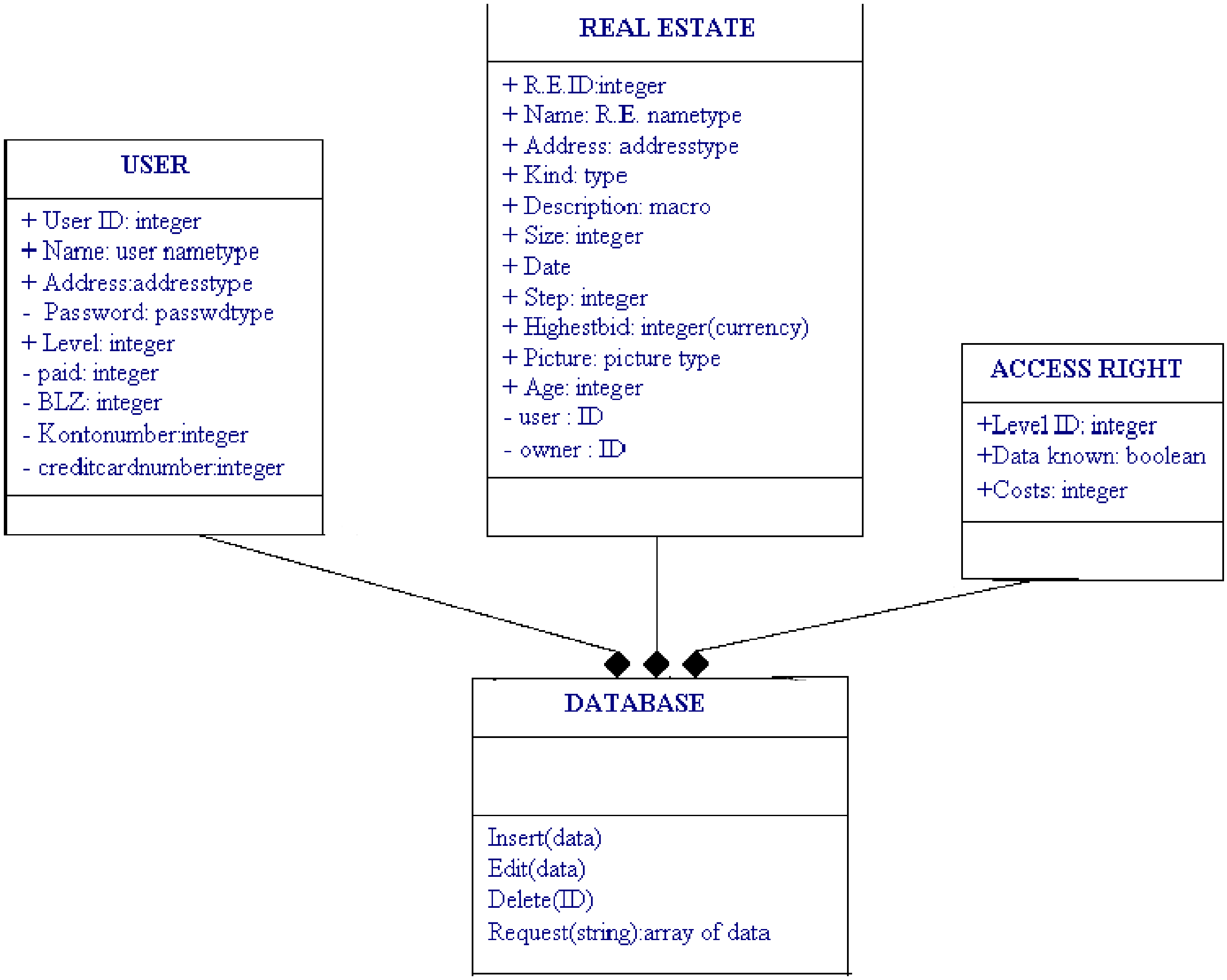
For user administration we have to distinguish what the adminstritator does, hence the branches.

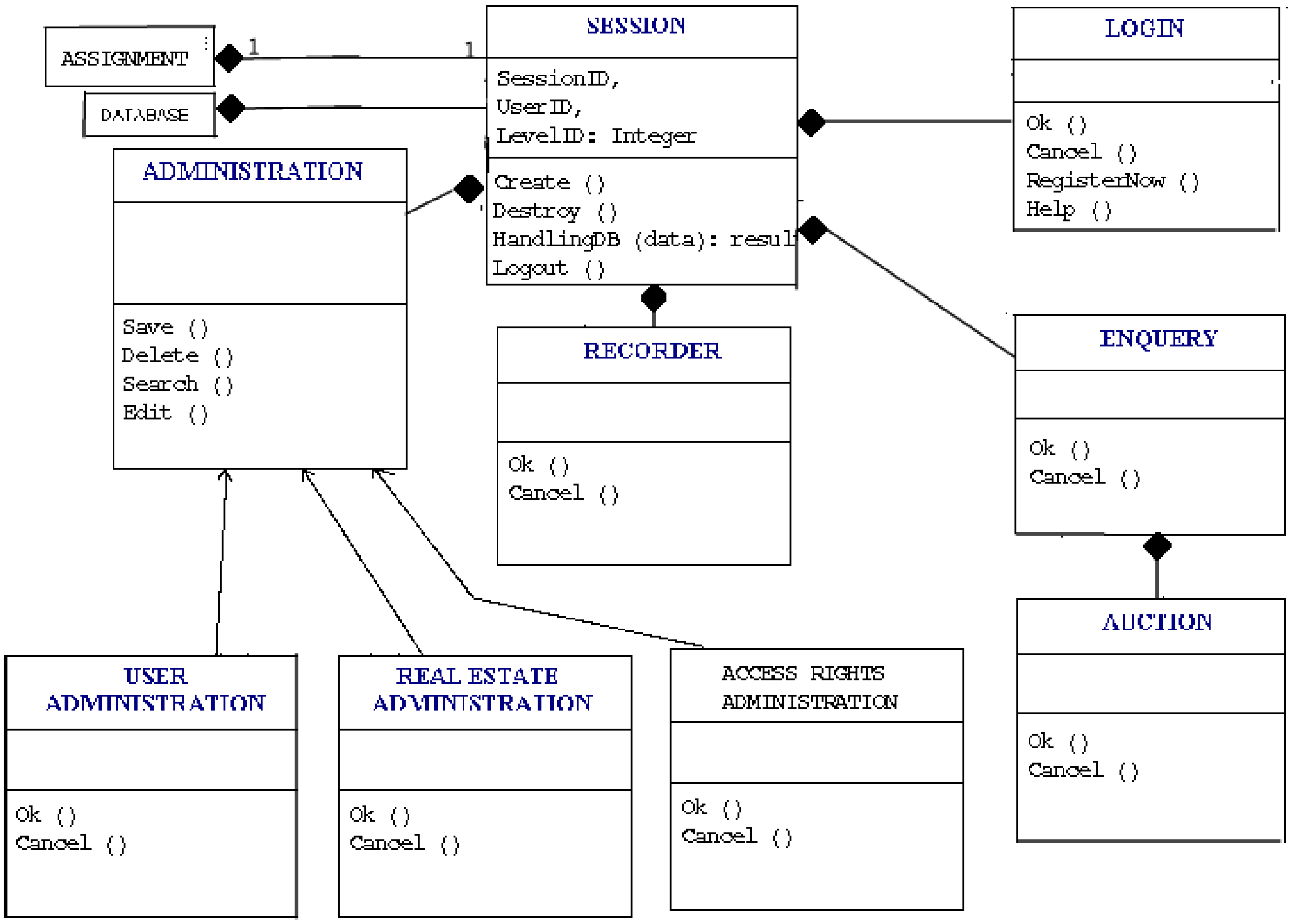
7.3 Designing the running System

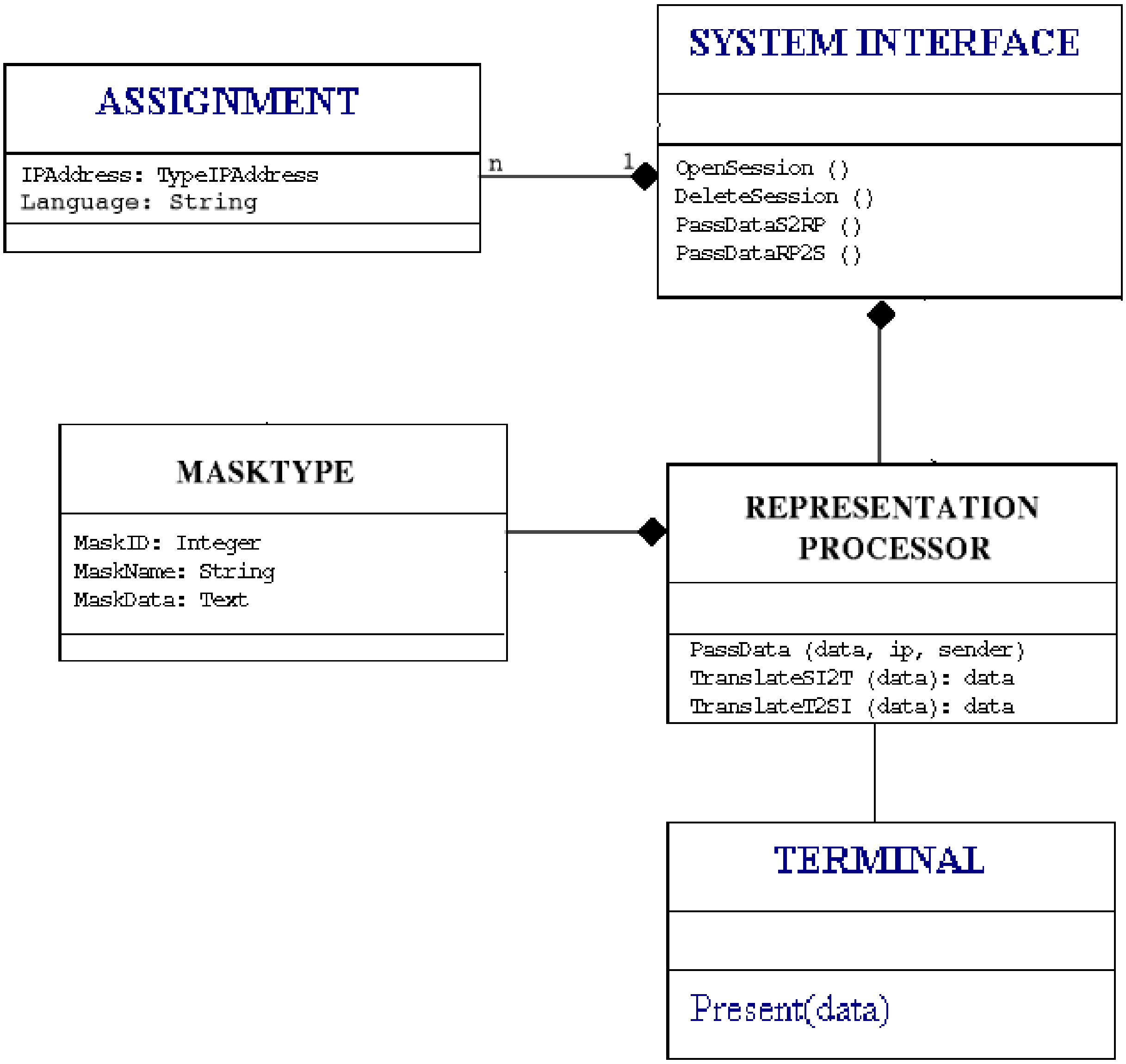
To design a running system requires some know-how. First of all, we need to know how the control elements are able to react. This means for example, how a button calls an event-controlled method. Event-controlled methods usually have void returns, since there is no calling to routine to which it can return a value. Furtheron, the system can be designed when it's clear in which direction and when and what for some classes should communicate with each other. The object knows its owner so the communication between both is no problem. Another point is that the implementation of a problem usually shows certains patterns that are used again and again, just in different ways; such patterns are called *Design-Patterns*. These are very helpful because then the program is easier to understand, when it has consistent ways of solutions to solve the certain subproblems.

7.3.1 Our Design

In *e-Immo*, one of these design-patterns is the solution of how to create and destruct an object of a class. They both have a special position. That means the creator exists and creates the object allthough it is a method of this object.


REAL ESTATE
+ R.E.ID:integer
+ Name: R.E. nametype
+ Address: addresstype
+ Kind: type
+ Description: macro
+ Size: integer
+ Date
+ Step: integer
+ Highestbid: integer(currency)
+ Picture: picture type
+ Age: integer
- user : ID
- owner : ID
USER
+ User ID: integer
+ Name: user nametype
+ Address:addresstype
- Password: passwdtype
+ Level: integer
- paid: integer
- BLZ: integer
- Kontonumber:integer
- creditcardnumber:integer
ACCESS RIGHT
+Level ID: integer
+Data known: boolean
+Costs: integer
DATABASE
Insert(data)
Edit(data)
Delete(ID)
Request(string):array of data


SESSION
SessionID,
UserID,
LevelID: Integer
Create ()
Destroy ()
HandlingDB (data): result
Logout ()
ASSIGNMENT
1
1
DATABASE
LOGIN
Ok ()
Cancel ()
RegisterNow ()
Help ()
ADMINISTRATION
Save ()
Delete ()
Search ()
Edit ()
RECORDER
Ok ()
Cancel ()
ENQUERY
Ok ()
Cancel ()
AUCTION
Ok ()
Cancel ()
USER ADMINISTRATION
Ok ()
Cancel ()
REAL ESTATE ADMINISTRATION
Ok ()
Cancel ()
ACCESS RIGHTS ADMINISTRATION
Ok ()
Cancel ()



Here are the classes again, with their attributes and methods — and why Only the methods will be explained, since they use the attributes somehow.

Terminal present(data): To present data which are translated by the Representation Processor on the screen.

Representation Processor translatesi2t(data): To translate data from the format specifications of the System Interface into HTML or any other output format. This method is called by *passdata*. The converted data are passed back to *passdata*.

translatet2si(data): To translate data from terminal into a system-required format. Functionallity same as *translatesi2t*.

passdata(data, IP, sender): Gets data with a boolean sender which defines if the sender is either the system interface or the terminal, so the method knows where to send the data after editing. Also the job of passdata is to put the masks (the data of masks) at disposal. This means it gets a mask requirement (the mask-ID), searches for the mask data and passes it to the terminal.

System Interface openSession(): Calls the Session's constructor. The sessions are instantiated in the attributes of the System Interface. There is an array of sessions (not explicite mentioned because the UML-Notation 1..* between the classes *assignement* and *system interface* means exactly the same).

deleteSession(): Calls the Session's destructor.

passdatas2rp(data, ID): Receives data from a session and looks in the tabele of assignment (also not explicitely declared because of UML-Notation) for the IP-address which is assigned to the session and passes the data and the address to the representation processor.

passdatarp2s(data, IP): Receives data from the representation processor, looks in the table of assignement for the correct session and passes the data.

Session create(): Is the constructor for the Session. It opens the Session for a user.

destroy(): Is the destructor for the Session. It destroys the Session with all of its components.

handlingDB(): Converts a string into an SQL-query and passes it to the database. Then it receives the answer, if necessary converts it and passes it back to the caller.

Login help(): Opens a help mask for the user.

cancel():Knows how to deal with all the events caused by *Cancel*-buttons.

ok(): Functionality like cancel but for each ok-button in the routine.

registernow: Gives the user the possibility to register itself immediately to get more information.

Enquirer cancel(): Knows how to deal with all the events caused by *Cancel*-buttons.

OK(): Deals with all the events concerning the enquirer caused by *OK*-buttons.

Auction OK(): Deals with all the events concerning the auction caused by *OK*-buttons.

cancel():Knows how to deal with all the events caused by *Cancel*-buttons.

Recorder OK(): Deals with all the events concerning the Recorder caused by *OK*-buttons.

cancel():Knows how to deal with all the events caused by *Cancel*-buttons.

Administration, User, Real Estate and Access Rights Administration OK(): Deals with all the events concerning the administration caused by *OK*-buttons.

cancel():Knows how to deal with all the events caused by *Cancel*-buttons.

7.4 Design and Deployment Diagram

7.4.1 Architecture

We changed our system. First we said, that the Login opens simultaneously with the Session, but now our system first creates the Session, and the Session creates and calls the Login.

We found out five subsystems, namely:

1. DATABASE with its components: Real Estate, User, Access Right.
2. SESSION with its components: Recorder, Inquiry, Auction, Login, Administration.
3. SYSTEM INTERFACE with its component: Assignment.
4. REPRESENTATION PROCESSOR.
5. TERMINAL.

7.4.2 Use Cases can help

Now we want to explain how our System works refering to Use Cases.

LOGIN The user is on the company's homepage and clicks on the button LOGIN. This Information passes the Representation Processor to the System Interface, it looks up the IP-Address in the *Assignment*-table and passes it to the correct Session's instance. The Session opens the Login, the Login passes the information about its MaskID via Session, System Interface to the Representation Processor which presents the mask on the Terminal. The user enters his password and his username and submits. This information passes the Representation Processor again as before. The System Interface knows to which Session this information belongs to, and passes it to the correct instance of the Session-class. The Session sends the information further to the Login, which checks the data via its owner's method *handlingDB*. The Database returns the fitting record - there can be only one or none found. It contains all the user's information and rights. The Login will be destroyed, and the Session gets the UserID and the Level, if it is wrong the Login is called again.

CHOICE MASK The Session submits the maskID for the choiceMask according to the Level to the Representation Processor via the System Interface, which knows from which Session to which Terminal the information is to send. The Representation Processor presents the mask, and the user chooses one topic. The information is passed by the Representation Processor to the System Interface, which submits the information to the correct Session. The Session creates either Enquirer or Recorder.

RECORDER If the user has choosen to record, the Session calls the constructor of the Recorder. The Recorder sends the information about its mask via the Session, and the System Interface to the Representation Processor, which sends the mask to the Terminal. So the user can type in data. When he submits the data, they go via the Representation Processor, which translate the language the System Interface will understand. It knows to which Session the data belong to and sends them. The data come via the Session's method *handlingDB* to the database. Here the data are fetched and returned to the callee, which is the Ok of the Recorder.

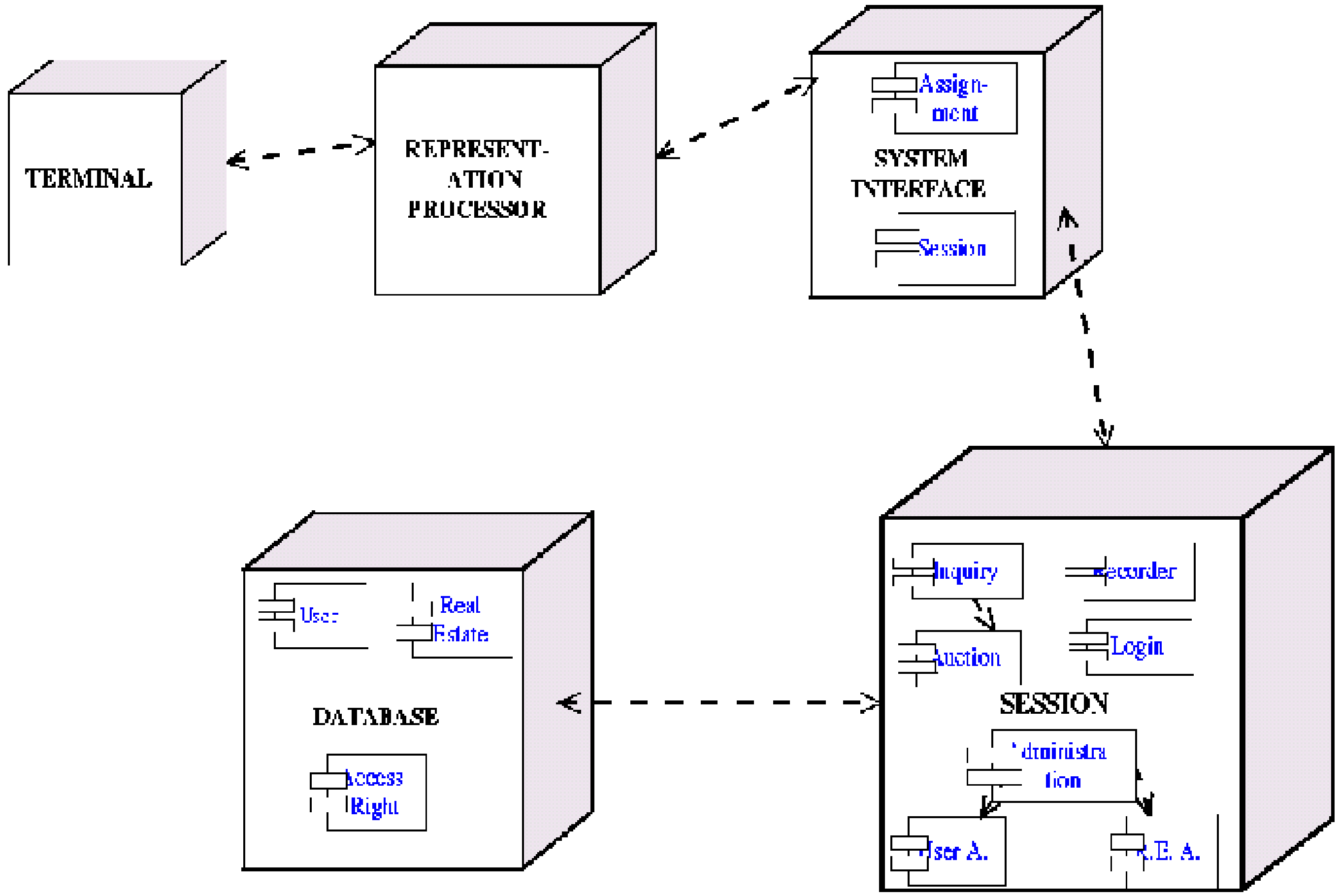
ENQUIRER The Enquirer will be created by the Session, and the user gets the mask the same way like the recorder mask. The user types in the data and submits it. Now he can choose data and click Okay. This informatin reaches the Enquirer the same way again. It checks the input and sends it to the Database which makes a query. The information will be sent back to the Enquirer which tells the Representation Processor to translate the information and to present it on the Terminal. The user can choose one real estate, and this information passed by the Representation Processor, which translates it, comes to the System Interface, which sends it to the correct Session, and makes a query again. The information is presented on the terminal at the same way, and the user can click at the button Auction.

AUCTION The Inquirer starts the Auction's constructor, and the auction requests if the user really wants to bid. If he confirm, the auction gets the information, and calls the Database to renew the prize.

ADMINSTRATION The Administration is also started by the Session, and also the mask is presented on the same way. But first we must request how to search for the specific data.

CANCEL, LOGOUT, OK Every time the user clicks Cancel, Logout or Ok, the System knows how to handle this information. Each control element calls the correct routine which has to check from which control element this signal came (or we need to write one routine for each of the buttons).
If the user logs out, a call of the session's logout is forced. This call is caught by the system interface which then will destroy the Session's instance by calling its destructor.

Deployment Diagram



Chapter 8

Abstract

As an abstract, we want to show the system architecture. Well, first of all we have to ask, what the system should be for — hence, what are the requirements to the application. We have an action firm via the Internet which implies some kind of login that distiguishes between administration and other users. Furtheron, we need to regard that the visual components are not part of our system, they need to be formulated as HTML or WML. The system needs to be able to deal with a number of simultaneous logins. The action firm handles real estates. These real estates are held in a database.

The next question was how to realize the the user interfaces. We distinguish between

1. Login.
2. Enquiring real estates.
3. Offering real estates.
4. Participate the auction.
5. Administrate.

We solved this by putting the point *participate the auction* to the detail view of *enquiring real estates*. The rest is clear. We have a login, and after having logged in successfully, you will come to an action choice (what do you want to do?). This choice will not appear if you have the right to enquire only.
In the administration, you can edit all the users with their rights. Also the administration of the real estates is an own mask.

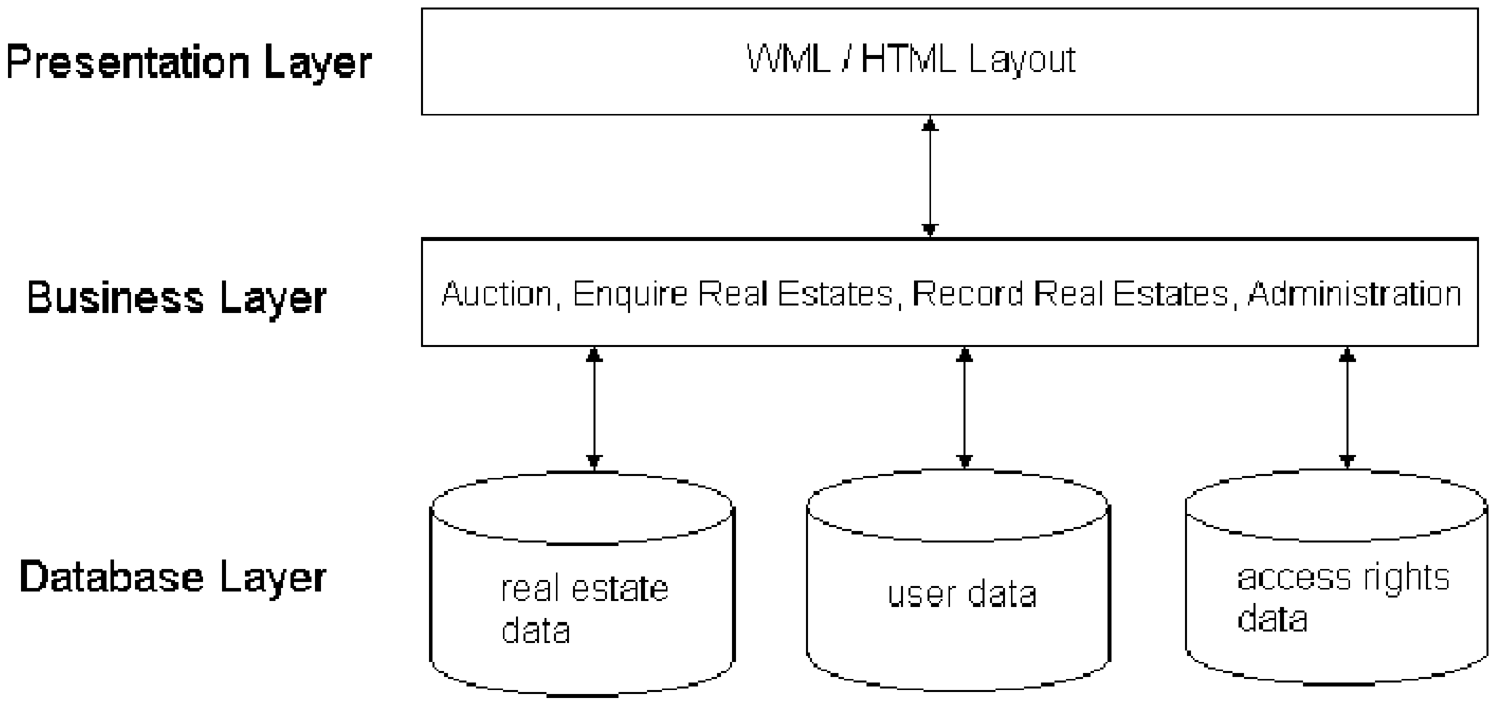
Our system is a layer model — containing three layers:

1. Presentation.
2. Business.
3. Database.

Most compilcated part of the system is the presentation layer, since it has to translate between HTML, WML and a self convented language. Furhteron, it is open to the external network client. It communicates with the Business-layer via well-defined routines. The Business-layer is our interface to the user. Regard that there is one part within that is fix (instantiated only once), and one part that is instantiated for each user. The physical database, of course, exists only once. On the interface to the application (internal and external file accession), we have the database class to access the physical database.

On this point, it became even more clear, that our application needs to be event-controlled. Otherwise, all the simultaneous logins would conflict with each other, and waiting-loops will make the system quite inert.

Layered architecture - functional view



Appendix A

Protocols

A.1 2001-31-October

1. Person responsible for protocol
   Rescher Robert Harald

2. Participating people
   Mag. Dipl. Ing. Schwaiger Roland, Rescher Robert, Strobl Elisabeth, Nidetzky Marion, Fimberger Lucia, Schobesberger Oskar

3. Topics talked about
   What is a CRC-Session?
   Presentation of our CRC cards.

4. New assignments
   We need new classes to refine our application.

5. Open questions
   Is the auction online?
   Is there any restriction?
   Should the user be identified?

First of all, the application was built. Therefore, we declared a class named **managing system**, which was our main operating application unit. This unit encapsulates the login managing system as well as the read- and -write-methods to access the database class. The **database** class is the highest level in the hierarchy. It consists of the data pool and the methods to post and enquiry data. The database encapsulates the $real-estates$, $users$, $login-role$ and $user-rights$. Here are the contents of the classes:

- **real-estates**: This class contains all the necessary entities of the real-estates to be stored.

- **users**: This class contains all the user information.

- **login-role**: This class contains the access rights.

- **user-rights**: This class joins the user class with the login-role class.

These classes are part of the database, so they do not need functional routines. For logging in, there will be responsible another class that contains the functional routines, and another one for the user interface.
Finally, we constructed a user interface for managing the classes of the database, since the managing system does not distinguish between a user or a further instance of itself when communicating. So, there is a class **user interface**, which calls the methods of the managing system, and is also responsible for the display and input/output to communicate with a human user. The user interface is best described with a *user interface prototype.*

A.2 2001-07-November

1. Person responsible for protocol
   Rescher Robert Harald

2. Participating people
   Mag. Dipl. Ing. Schwaiger Roland, Rescher Robert, Strobl Elisabeth, Nidetzky Marion, Fimberger Lucia, Schobesberger Oskar

3. Topics talked about
   Presentation of refined CRC cards.

4. New assignments
   Auction is online.
   User interface prototype.
   Restriction on complete object via onion-model.
   Other companies can input an offer.
   Perhaps a special offer: Display offers close to logged in user's area.

The next step will be to refine the user interface.
Therefore, we renamed the managing system to **system interface**, since it is better to understand with this terminology.
We have to distinguish at least these levels in our problem: The database level containing the real-estates and the one with the entities for user logins. Both these class groups (four class) can be joined by one class which holds the methods for editing and inquirying the data, or one class for the real-estates and one for the classes that contain the users login information. The second possibility can be used when there is a second system installed in the company that does not distinguish between collaborators and other users.
In combination with the login rights are not only the access rights, but also a set of real-estates that are returned when inquirying. So, the class **real-estates** has a property level. The level assigned to the user defined in the user collection must be at least as great as the level defined in the single instances of the real-estates to make them visible to the user. This filtering

is done by the **representation processor**, a class that passes the real-estates valuing them by the **user collection**.
Since another system can communicate with it, until this point there are only passed data, never a display instruction. A class called **representation processor** formulates the data to a user mask, and passes the inputs done by the user to the higher classes.
A further requirement was that there is no local terminal program to be installed. So, the **representation processor** formulates a common format, such as *hypertext*. In the model appears a class called **terminal** which is not part of our application; it is for example a web-browser that understands hypertext.

A.3 2001-14-November

1. Person responsible for protocol
   Rescher Robert Harald

2. Participating people
   Mag. Dipl. Ing. Schwaiger Roland, Rescher Robert, Strobl Elisabeth, Nidetzky Marion, Fimberger Lucia

3. Topics talked about
   Presentation of the use cases for recording and enqurying data. Furhteron, a draft of the user interface prototype was mentioned.

4. New assignments
   Making an auction.

A.4 2001-21-November

1. Person responsible for protocol
   Fimberger Lucia

2. Participating people
   Mag. Dipl. Ing. Schwaiger Roland, Rescher Robert, Fimberger Lucia, Schobesberger Oskar

3. Topics talked about
   Presentation of the Use Case "Auction" and User Interface Prototyping "Auction". Presentation of Class diagram: class references, inheritation and instantiation.

4. New assignments
   We should avoid inheritation.
   Multiple user, solved by instantiation
   New classes: Auction, Session, Login.
   User Interface Flow Diagram for WML and HTML and proof of functionality.

A.5 2001-5-December

1. Person responsible for protocol
   Fimberger Lucia

2. Participating people
   Mag. Dipl. Ing. Schwaiger Roland, Rescher Robert, Fimberger Lucia, Schobesberger Oskar

3. Topics talked about
   Presentation of the User Interface Flow Diagram and its masks.

4. New assignments
   Sequence Diagram, State Diagram, WML and HTML pages.

A.6 2001-12-December

1. People responsible for protocol
   Rescher Robert, Strobl Elisabeth, Schobesberger Oskar, Fimberger Lucia

2. Participating people
   Mag. Dipl. Ing. Schwaiger Roland, Rescher Robert, Fimberger Lucia, Schobesberger Oskar

3. New assignments
   Designing the running system.

A.7 2002-16-January

1. People responsible for protocol
   Nidetzky Marion, Rescher Robert

2. Participating people
   Mag. Dipl. Ing. Schwaiger Roland, Rescher Robert, Fimberger Lucia, Schobesberger Oskar

3. Topics talked about
   Presentation of the entire project.

A.8 2002-23-January

1. People responsible for protocol
   Fimberger Lucia, Nidetzky Marion, Rescher Robert

2. Participating people
   Mag. Dipl. Ing. Schwaiger Roland, Rescher Robert, Fimberger Lucia, Schobesberger Oskar, Strobl Elisabeth, Nidetzky Marion

3. Topics talked about
   Presentation of the entire project (infront of the other groups). The project has been completely re-structured since the week before, and then it worked perfectly.