November 21, 2001

Seminar Software Engineering

ImmoWarrior

Report

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

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 is Beck 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. Real-Estate

   responsibilities collaboration

   - knows its characteristics

2. Real-Estate Collection

   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. Enquirer

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.

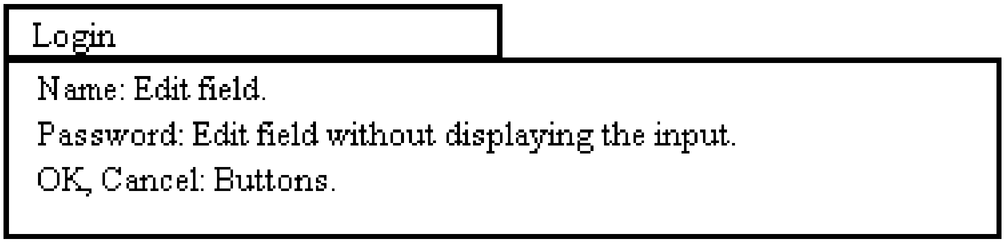
At any time:

- Cancel: Presents detailed view of the real-estate.
- Logout: Presents login mask.

Chapter 5

Essential User Interface Prototyping

Here is a rough draft for the user interfaces. They meet the compliances of the use cases.

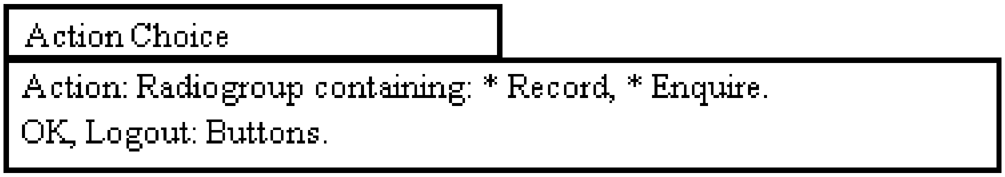


Login

Name: Edit field.

Password: Edit field without displaying the input.

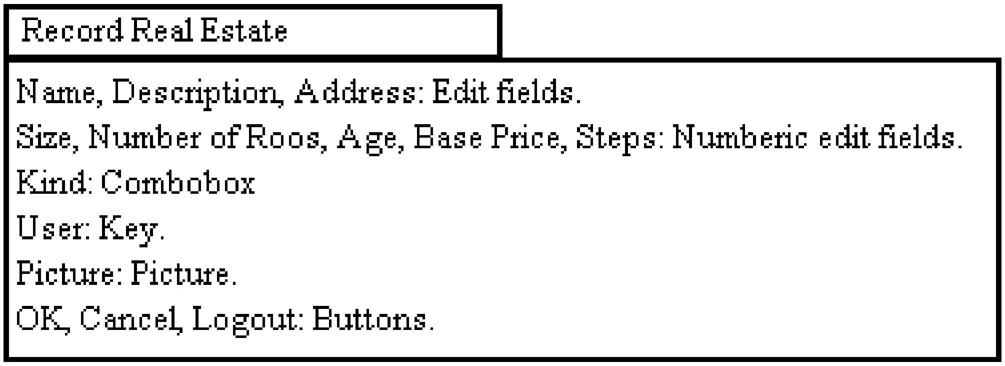
OK, Cancel: Buttons.



Action Choice

Action: Radiogroup containing: * Record, * Enquire.

OK, Logout: Buttons.



Record Real Estate

Name, Description, Address: Edit fields.

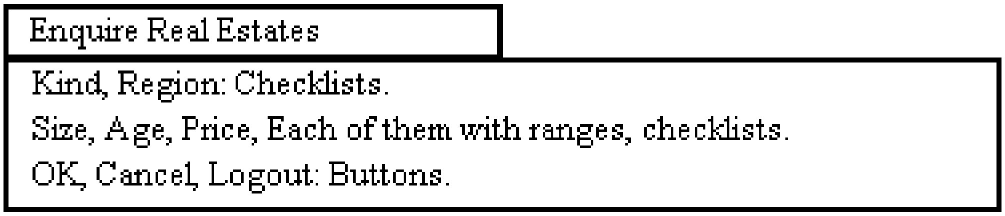
Size, Number of Roos, Age, Base Price, Steps: Numberic edit fields.

Kind: Combobox

User: Key.

Picture: Picture.

OK, Cancel, Logout: Buttons.



Enquire Real Estates

Kind, Region: Checklists.

Size, Age, Price, Each of them with ranges, checklists.

OK, Cancel, Logout: Buttons.

Enquiry Results

Table with enquired real estates.
OK, Logout, participate Auction, Real Estate Details: Buttons

Real Estate Details

All details about the displayed real estate.
Cancel, Logout, Bid: Buttons.

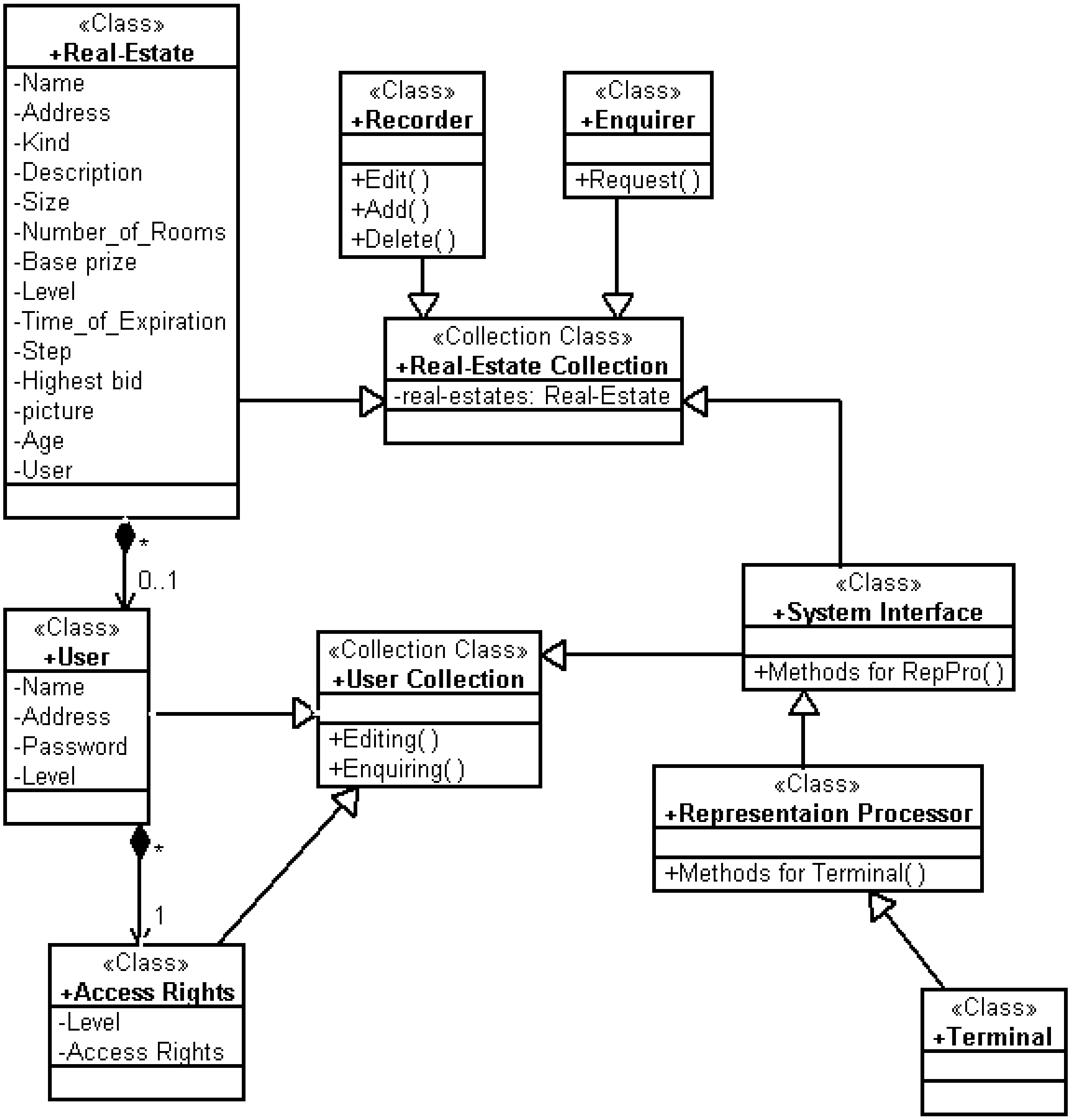
Chapter 6

Class Diagram

6.1 What is 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



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.