PS Software Engineering
January 30, 2002

ImmoWarrior

Report

Andrea Geierspichler, Roman Gross, Christian Koidl, Michael Pober, Gunnar Ruhs
andreageierspichler@hotmail.com

Academic Supervisor Mag. DI Dr. Roland Schwaiger

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

The Project

To learn the how-to's of "Software Engineering" we simulate a real life situation in the pro-seminar. A client, represented by the seminar-leader, comes to our software-development-firm and gives us an assignment.

The initial information is: A company that auctions real estate wants us to develop a software-solution for their process of collecting and representing information. (Code: "Immowarrior). Potential users of this system are: the ones, who gather and enter the information and the ones, who want to read it. Those readers can either be persons, or other systems. This leads to two different scenarios:

- Immowarrior sends the information to an other computer system by using XML
- Immowarrior displays the information for a person - what requires us to design an appropriate user-interface.

Specially required by the client is, that the software is to be easily maintainable and that no one has to install client-software to be able to use Immowarrior.
For the first phase of the project, we ought to concentrate on the basic-functions of the software, but we should keep the possibility in mind to add expansions, developed in co-operation with our group.

This is the requirement profile for our project. The given deadline is the end of January 2002

Chapter 2

A CRC Session

2.1 What is a CRC-session?

The CRC-session was invented by Ward Cunningham and Kent Beck in 1989 and then popularised by Rebecca Wirfs-Brock.CRC stands for Class, Responsibility and Collaborator.It is an informal technique for object oriented analysis and design, used for visualising the structure and behaviour of object oriented systems. Basically standard index cards are used to represent classes, responsibilities and colaborabtions with other classes, they are collected on the frontside. On the backside descriptions of the classes and attributes are witten down. A class is an abstraction of something from the problem domain. Responsibilities are tasks that can be ascribed to particular classes and a collaborator is another class that you require to talk to in order to carry out your own responsibilities. The goal is to define and to structure the problem domain and to represent the problem in an object oriented way.

2.2 Organising a CRC-session

The number of people is important, too few and it becomes difficult to mentally change gear, too many and you are not all involved and the productivity is cut by more disagreements. Five is the ideal number. If there are more people, one solution is to have the extra people be present strictly as observers. The group should be composed of developers, domain experts and an object oriented faciliator. Try to avoid having an excessivly dominant participant - no bosses either. From the system requirements decide on a simple set of scenarios that you want to try out.

2.3 Starting the session

The first step is to extract classes from the problem domain. One useful tool is to find all the nouns and verbs. The nouns are a good key to what classes are in a system and the verbs show what their responsibilities are. With this information you can start a brainstorming session. Now analyse which classes are important for the project. You do not

need to find all classes and responsibilities. The scenario will make them more obvious later on. In a CRC-session subclasses and superclasses are also defined. This can be done at any time it becomes obvious. Attributes of the class do not really have to be defined right now. This will be necessary when you get to the design phase.

2.4 Defining Use-Cases

Each person should take the responsibility for being a class. Select a scenario, identify and allocate new classes and responsibilities as the scenario unfoldes. Each person has to focus on their own class. The list of classes will grow and then shrink as the group filters out the good ones. Afterwards try out some speacial scenarious like exceptions. Later on, when getting to the design phase consider folowing things: Target environment, language choice of supporting software components, performance requirements.

2.5 Perform Use-Case Scenario Testing

1. **Call out a new scenario.** In this phase of the CRC-session the scenario is discussed and described in basic
2. **Determine which card should handle the responsibility.**
3. **Update the card whenever necessary.** If a card needs to be updated one of two situations has arisen: The responsibility needs to be added to an existing card or a new card needs to be created.
4. **Describe the processing logic.** Now the bussines logic for the responsibilities should be described step by step and the rules for the system should be recorded.
5. **Act out the collaboration between the different classes.**

2.6 Advantages of CRC-Modelling

1. The analysis is done by people who understand the problem domain (experts)
2. The future-users are actively involved in defining the model for the program.
3. CRC-modelling can easily be explained to a group of people.
4. The CRC-session leads directly into class-diagramming.

2.7 Things you have to keep in mind

- Start with simple scenarios.
- Take the time to select meaningful class names.

- Take the time to write a description of the class.
- If in doubt, act it out!
- Layout the cards on the table to get a feeling for system structure.
- Be prepared to be flexible.

Chapter 3

Our CRC Cards

This is the latest version of the cards we have developed during the course.

Classname: **Admin**	
Responsibilities	Collaborators
access user interface view free offers log in configurate account create/modify account search for real estate view real estate info place/edit/delete real estate log off	user interface

Classname: **EditorAndUser**	
Responsibilities	Collaborators
access user interface view free offers log in configurate account search for real estate view real estate info place/edit/delete real estate log off	user interface

Classname: **Editor**	
Responsibilities	Collaborators
access user interface view free offers log in configurate account place/edit/delete real estate log off	user interface

Classname: **User**	
Responsibilities	Collaborators
access user interface view free offers log in configurate account search for real estate view real estate info log off	user interface

Classname: **Guest**	
Responsibilities	Collaborators
access user interface view free offers request new read account	user interface

Classname: **User Interface**	
Responsibilities	Collaborators
read input forward data show info start session	guest user editor editoranduser admin session

Classname: **Session**	
Responsibilities	Collaborators
create/modify/delete account initiate sending of account info forwards data to user interface initiate search send real estate info close session access info storage request account info	user interface account send account info search for info send real estate info info storage

Classname: **Account**	
Responsibilities	Collaborators
access info storage modify account info forward account info	session info storage

Classname: **Information Storage**	
Responsibilities	Collaborators
store account info store real estate info forward data	session account search for info

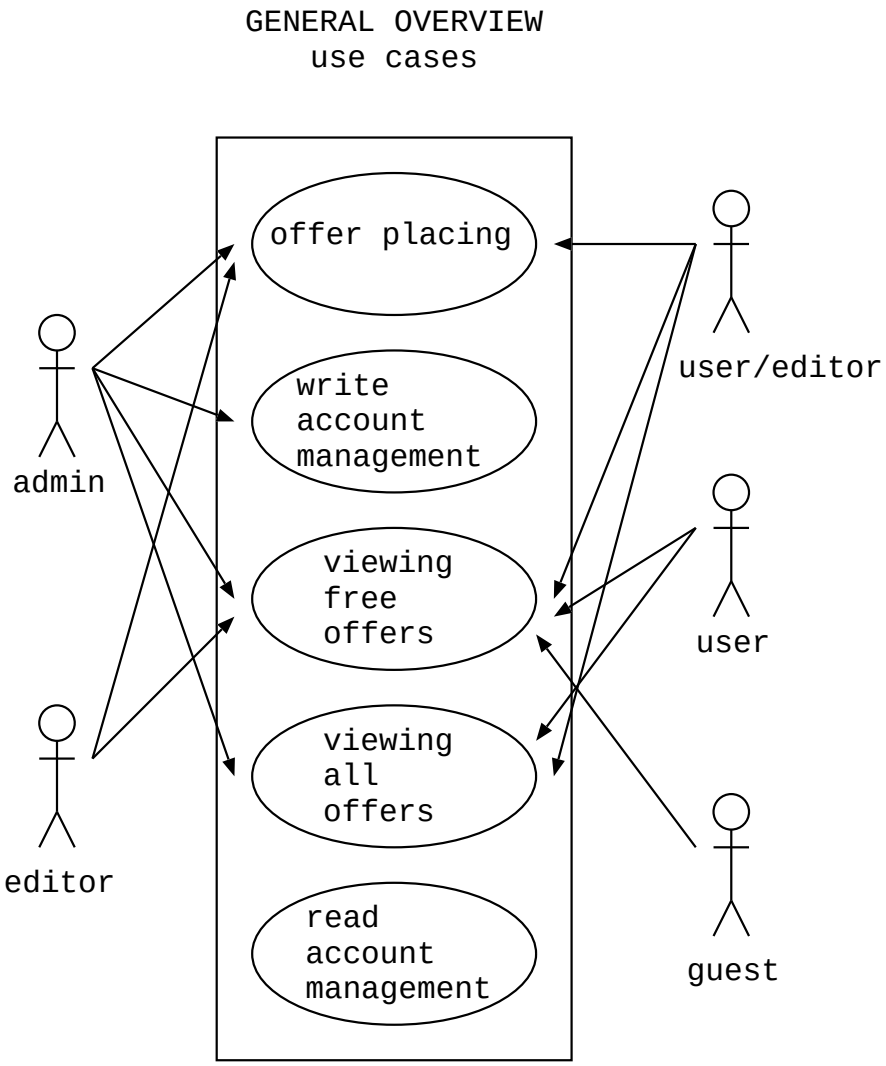
Classname: **Search For Info**	
Responsibilities	Collaborators
perform search forward search results	info storage session

Classname: **Send Account Info**	
Responsibilities	Collaborators
send username to e-mail/cell phone send password to e-mail/cell phone send account privileges to e-mail/cell phone	account session

Classname: **Send Real Estate Info**	
Responsibilities	Collaborators
send info to e-mail/cell phone look up account info	session account

Chapter 4

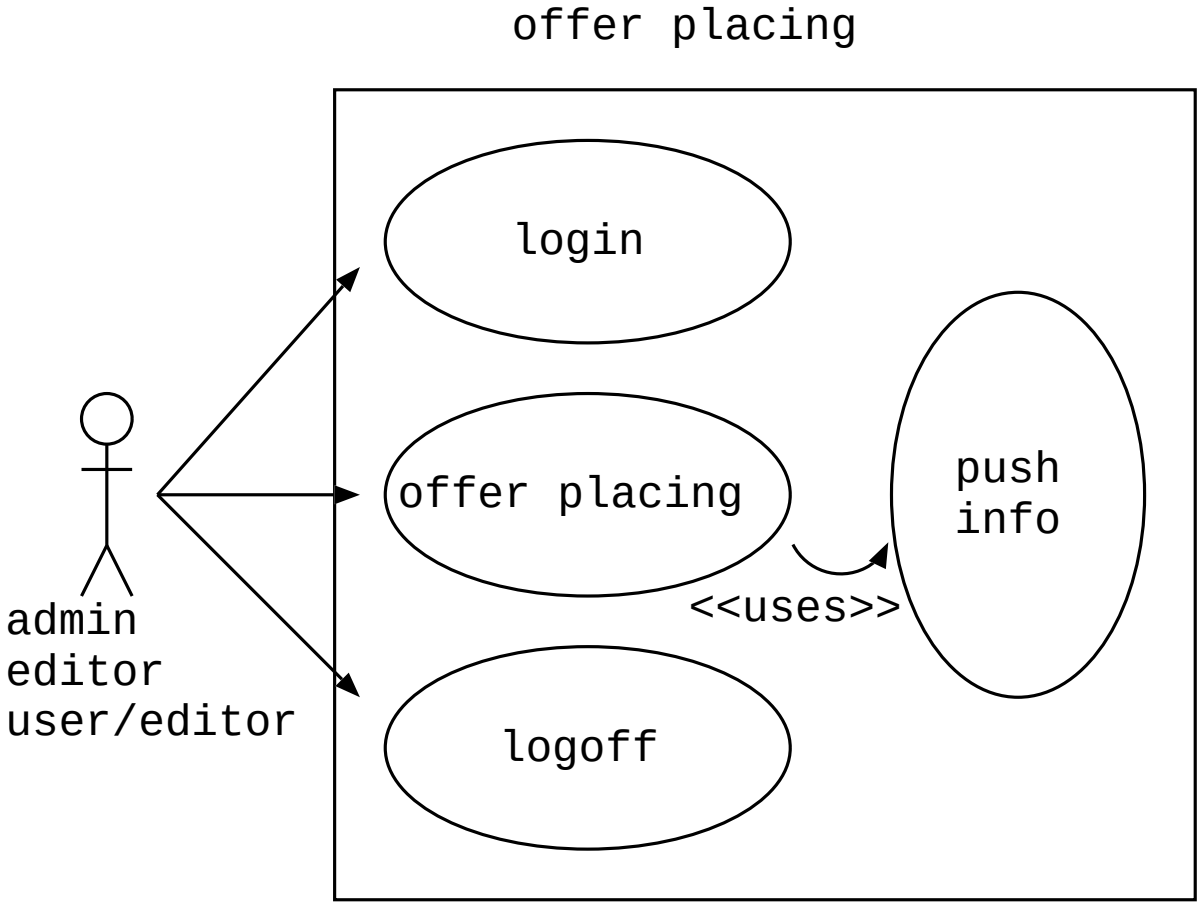
Essential Use Cases



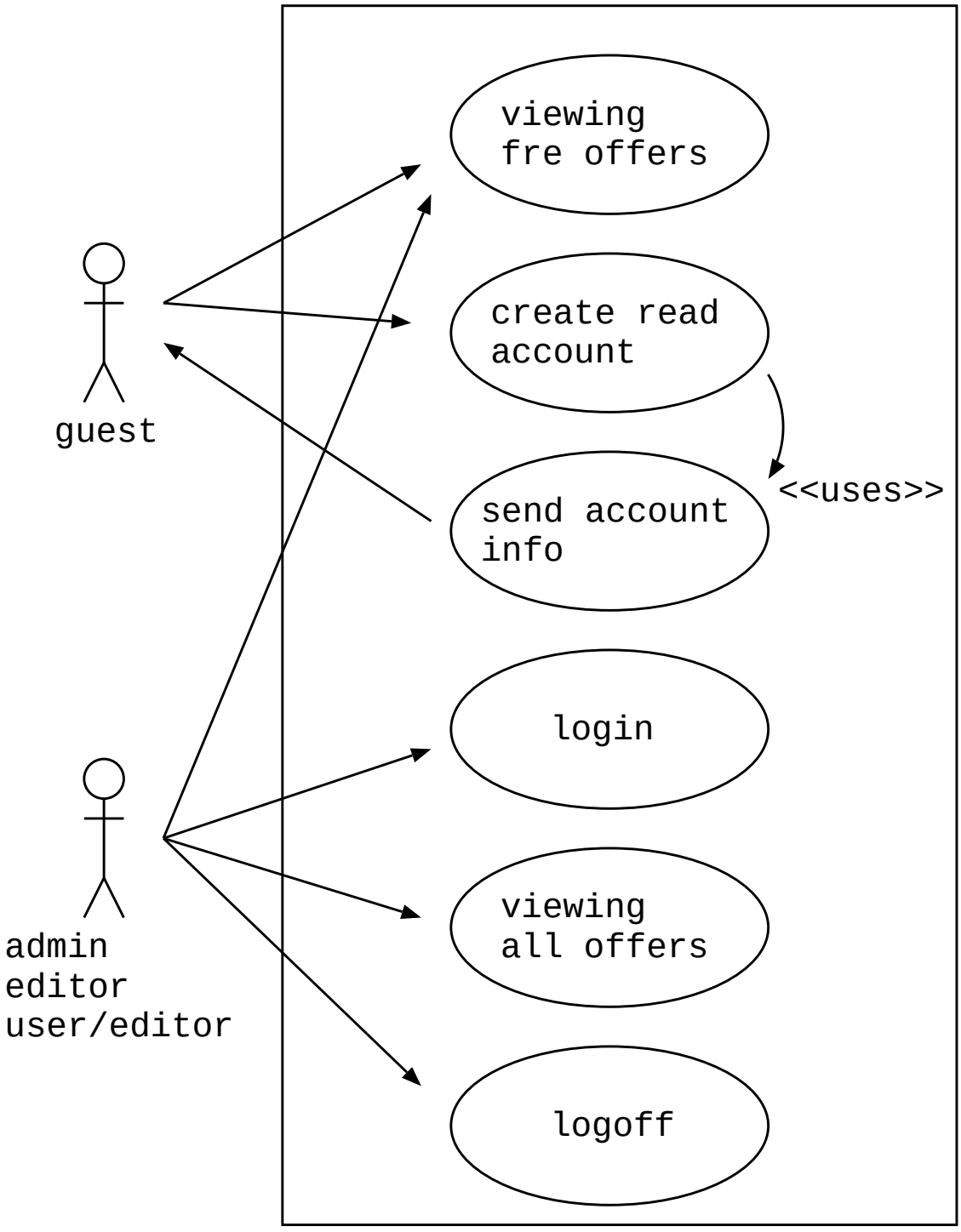
This general overview shows the main use cases and actors of ImmoWarrior. Five different actors can interact with the system:

- **guest**: The guest may view the free offers and request a read account.
- **user**: The user may view all offers and has all the guest's privileges.

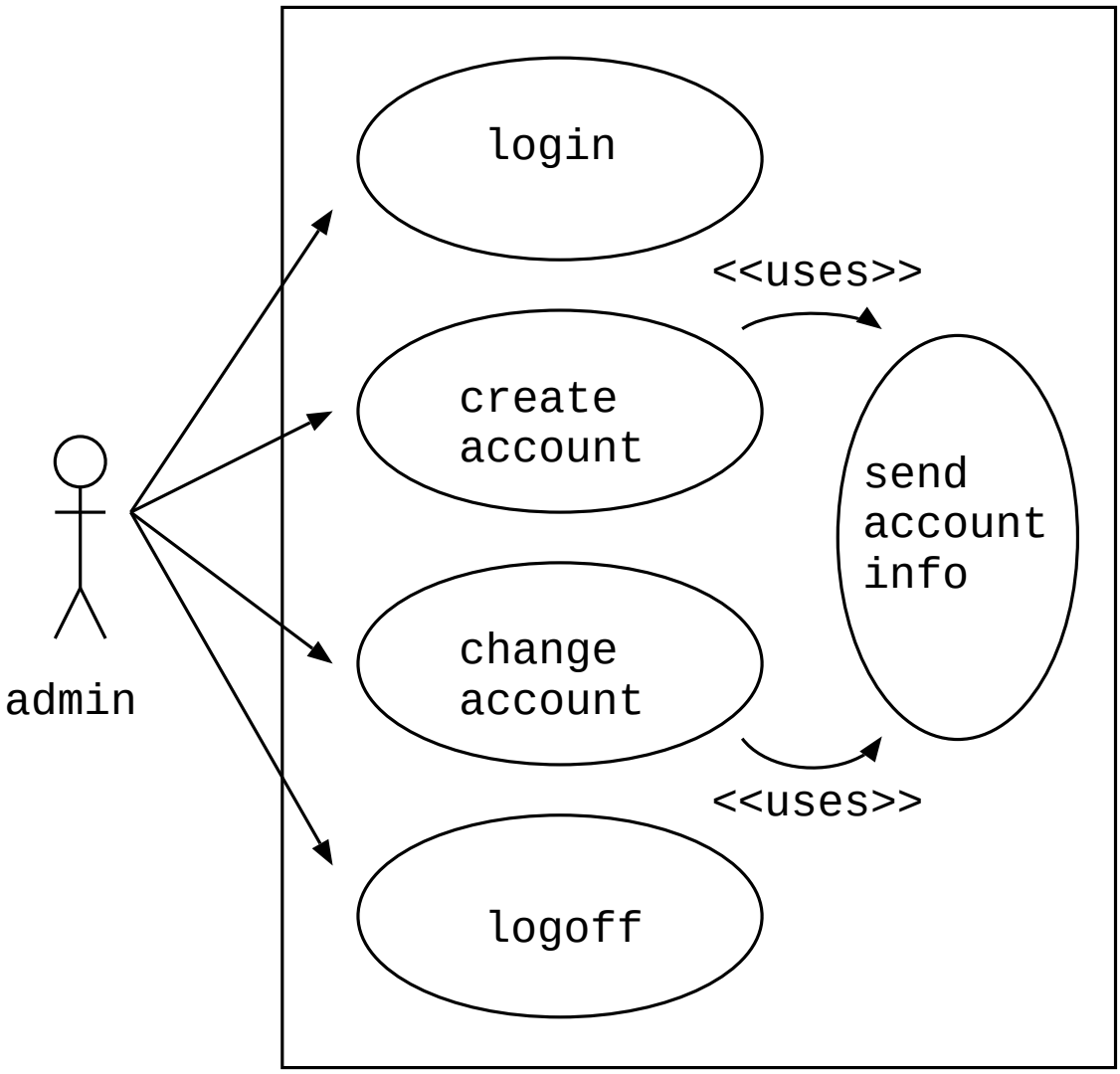
- **editor**: The editor may place real estate offers in the system and has all the guest's privileges.
- **user/editor**: The user/editor has the same privileges as the user and additionally the editor's privileges.
- **admin**: The admin has all the privileges of the other actors and additionally the write account management privilege.

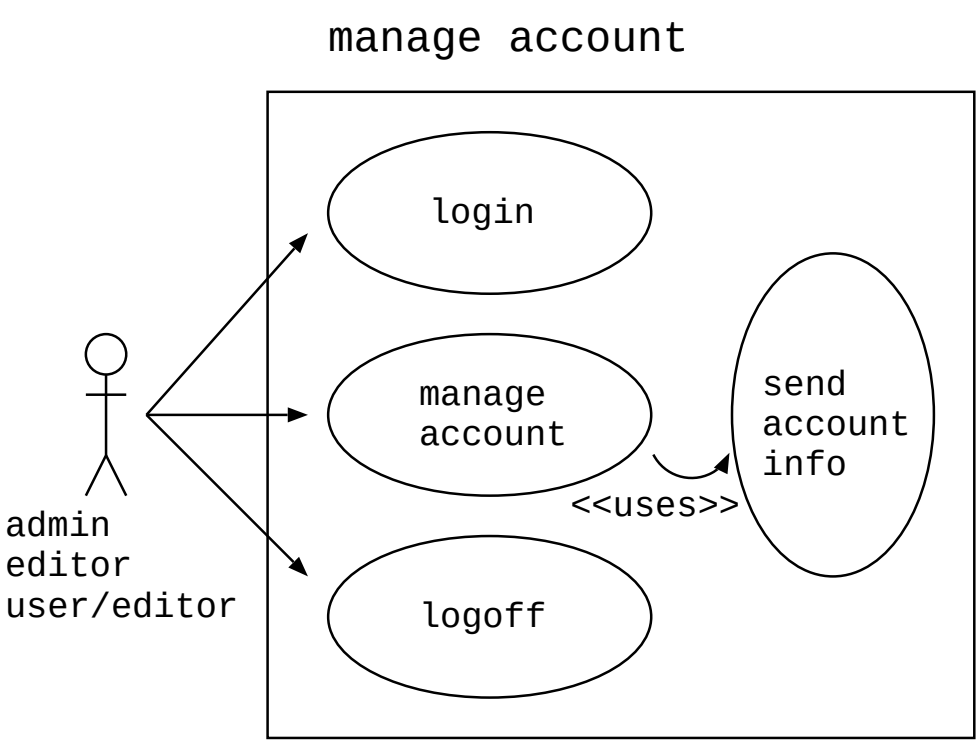


view free offers / create read account / view all offers



create/change account

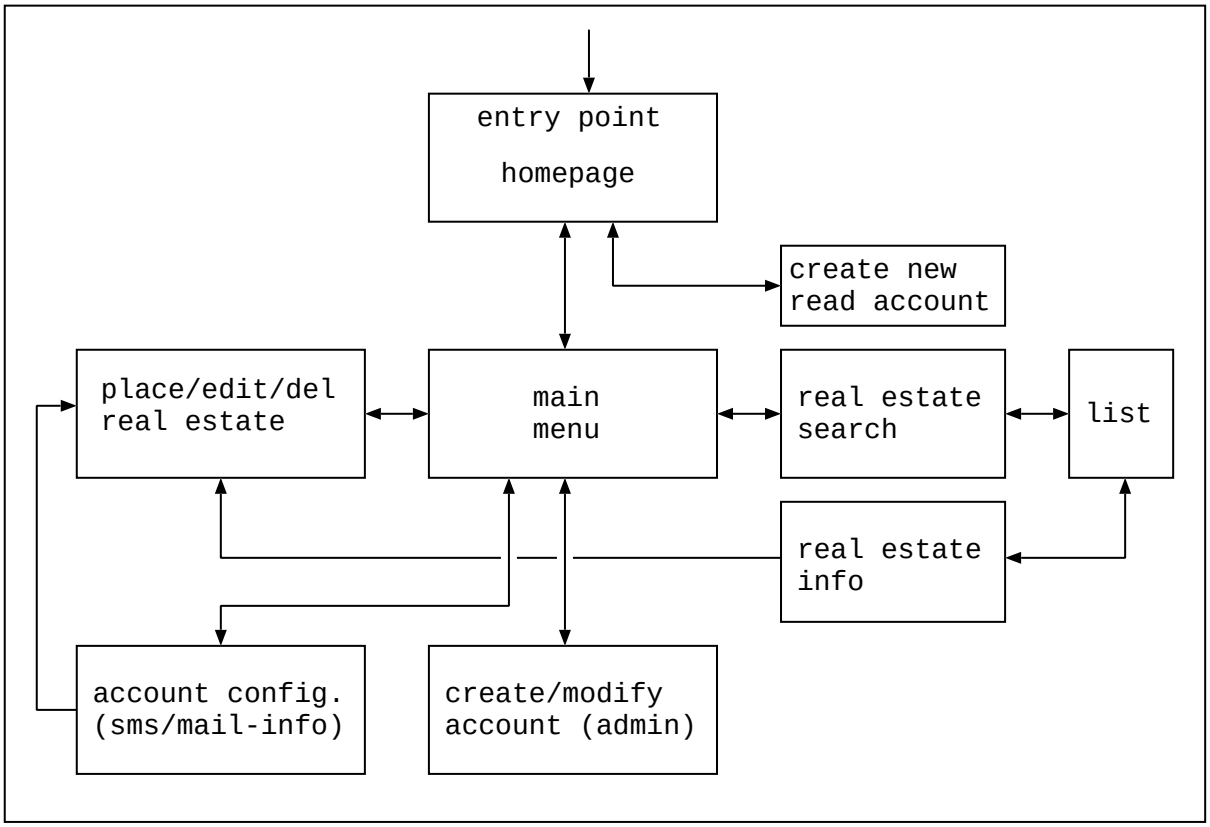



manage account
login
manage
account
send
account
info
<<uses>>
admin
editor
user/editor
logoff

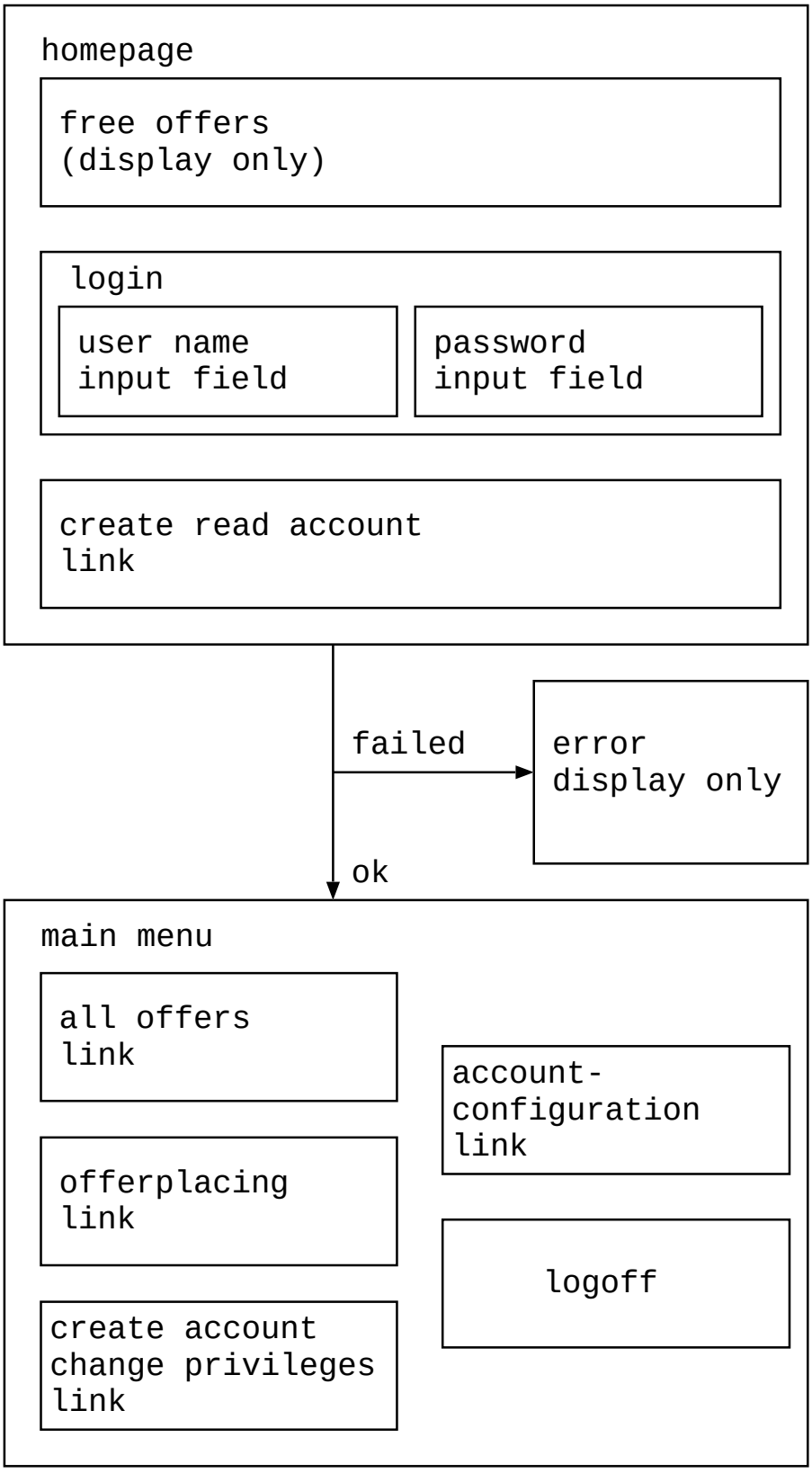
Chapter 5

Essential User Interface

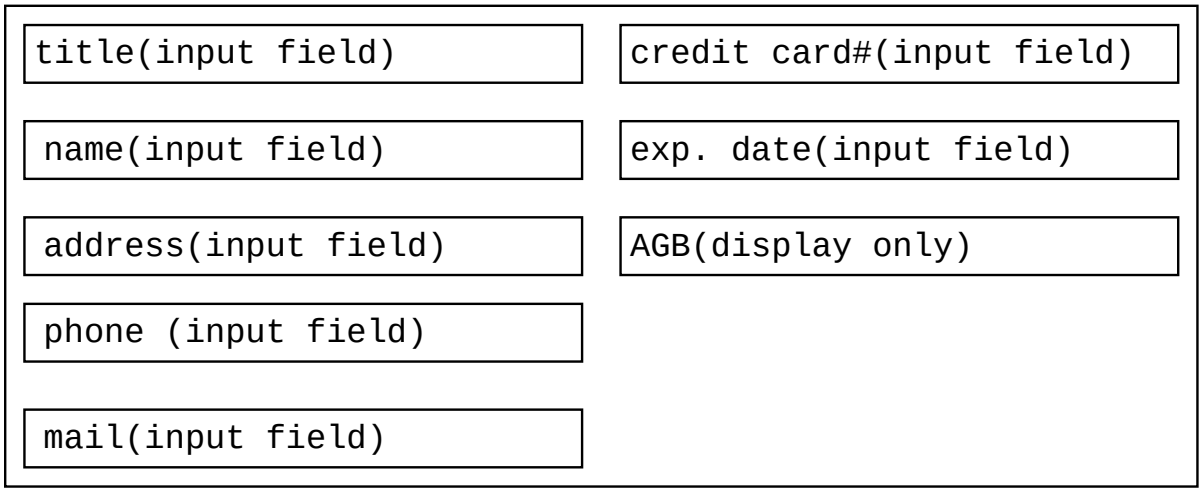
OVERVIEW: User Interface



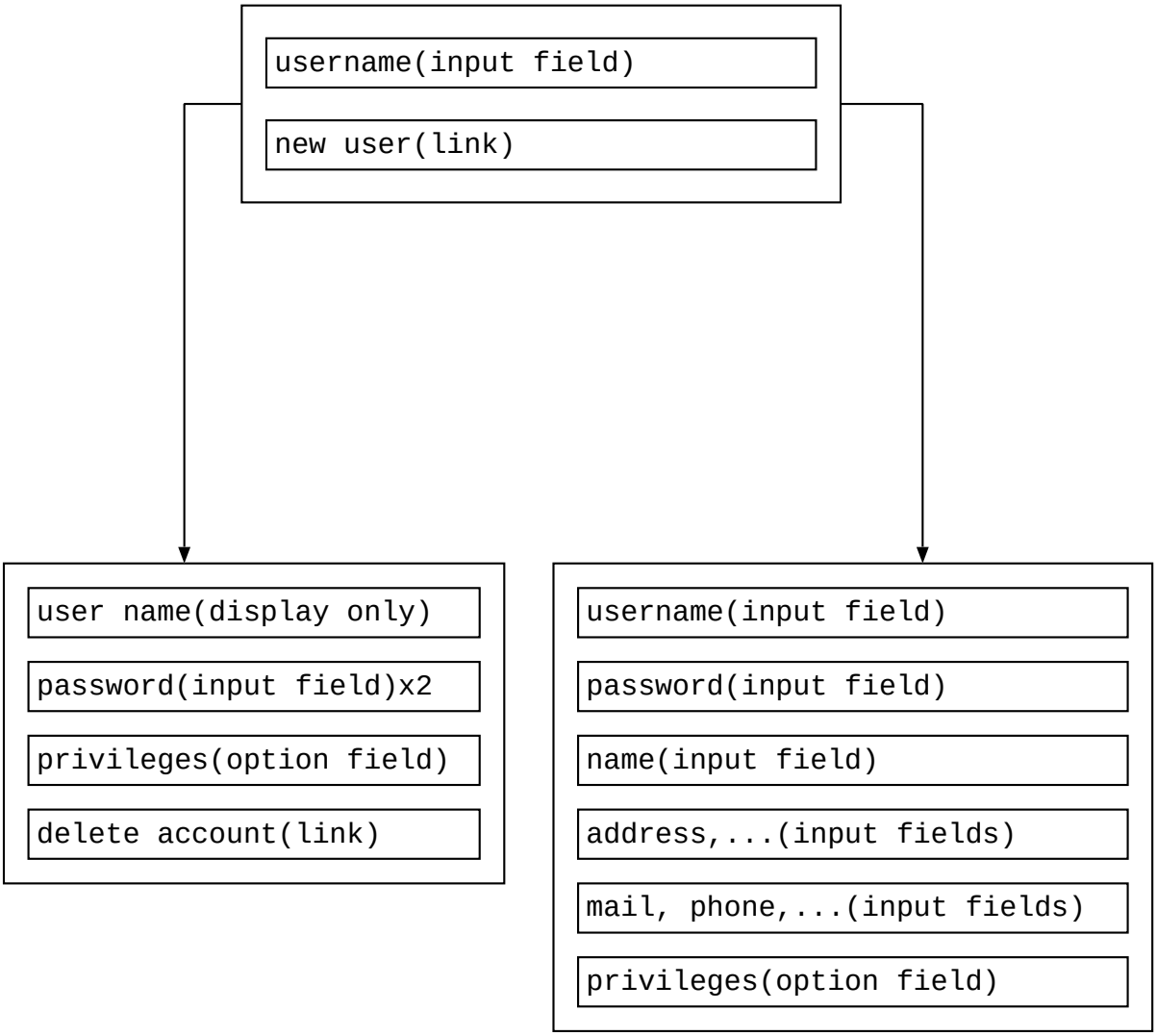
homepage/main menu



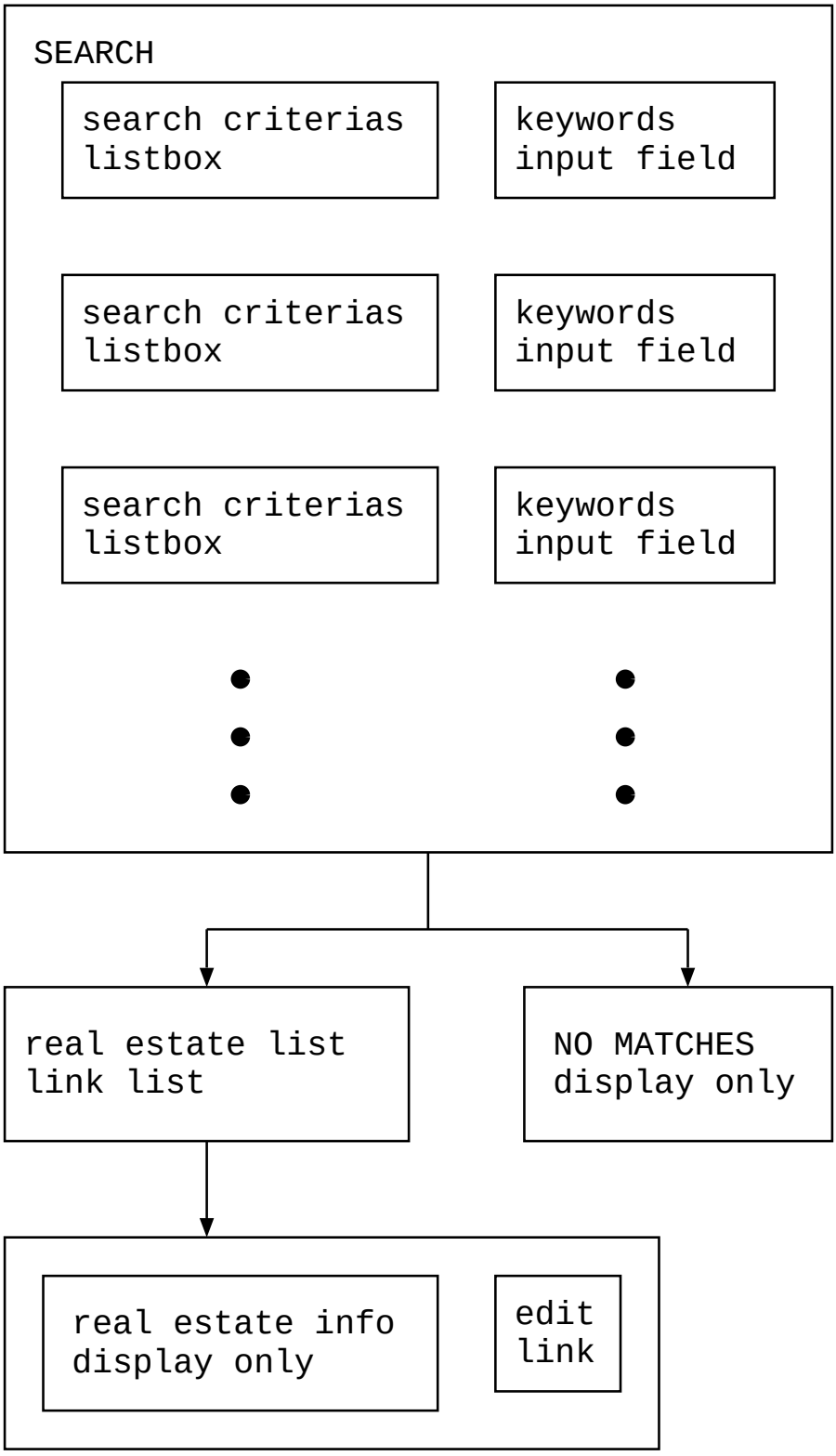
create read account



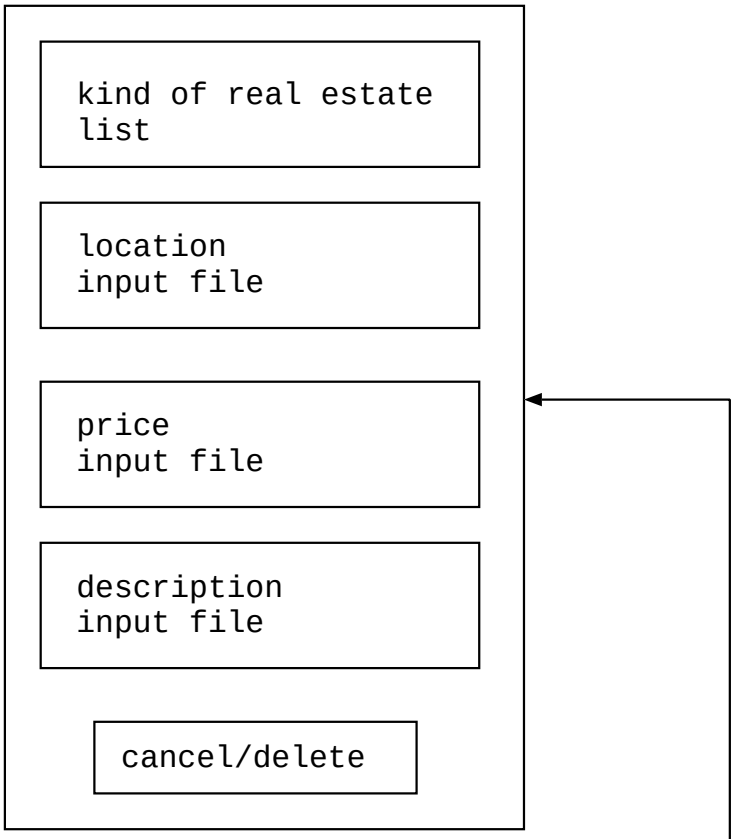
create account/change privileges



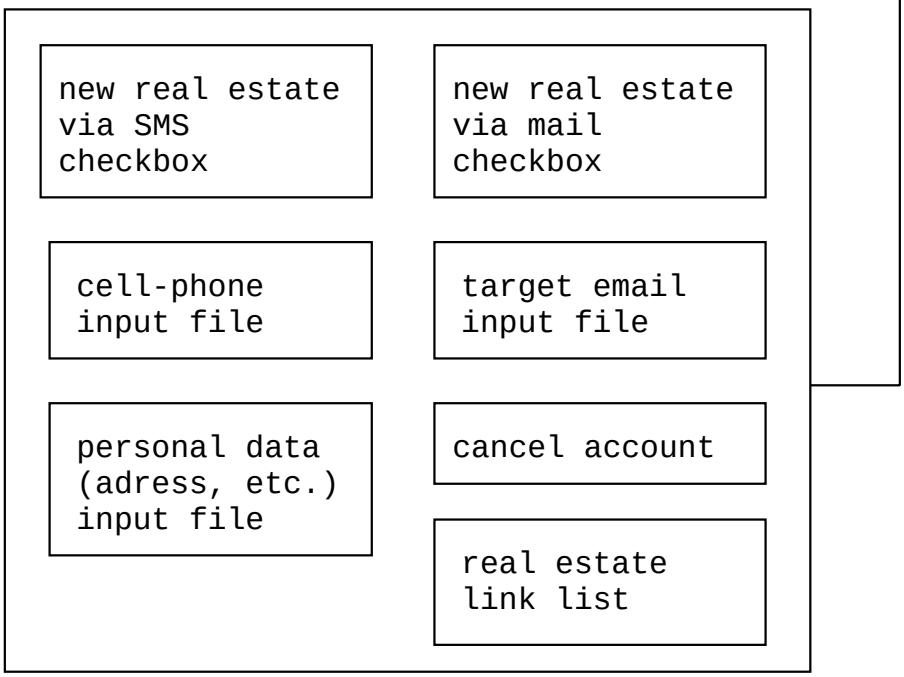
ALL OFFERS


SEARCH
search criterias listbox
keywords input field
search criterias listbox
keywords input field
search criterias listbox
keywords input field
real estate list link list
NO MATCHES display only
real estate info display only
edit link

OFFER PLACING



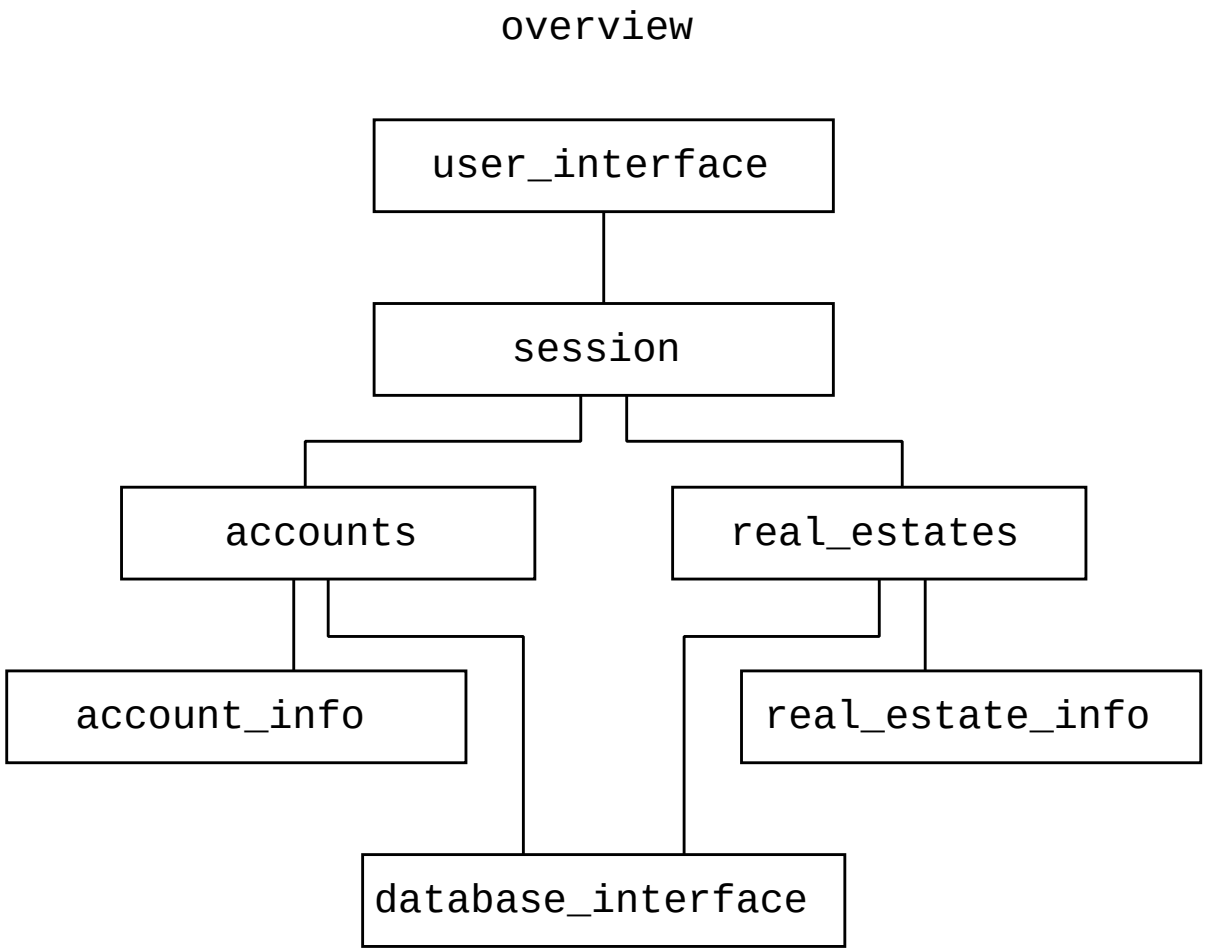
ACCOUNT CONFIGURATION

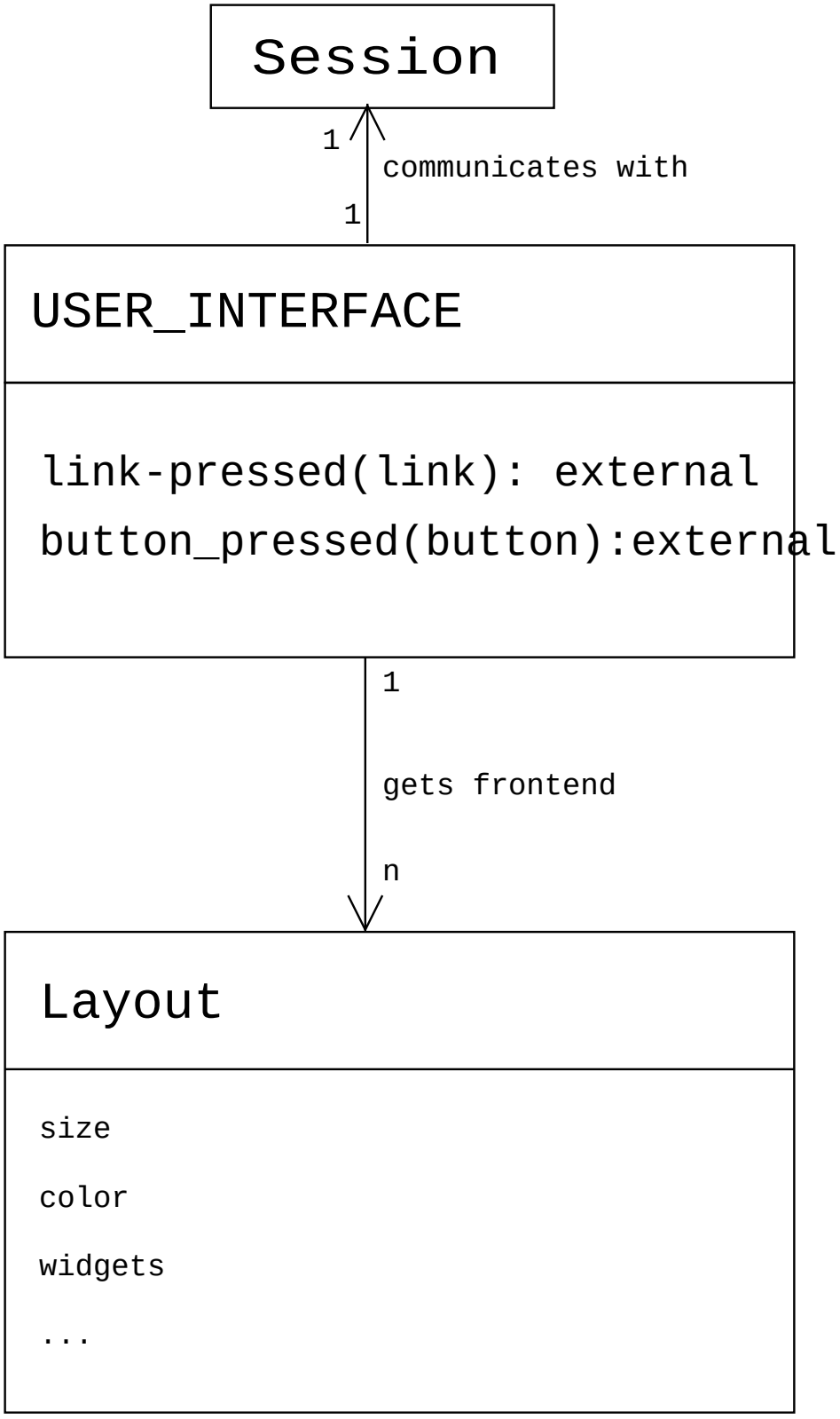


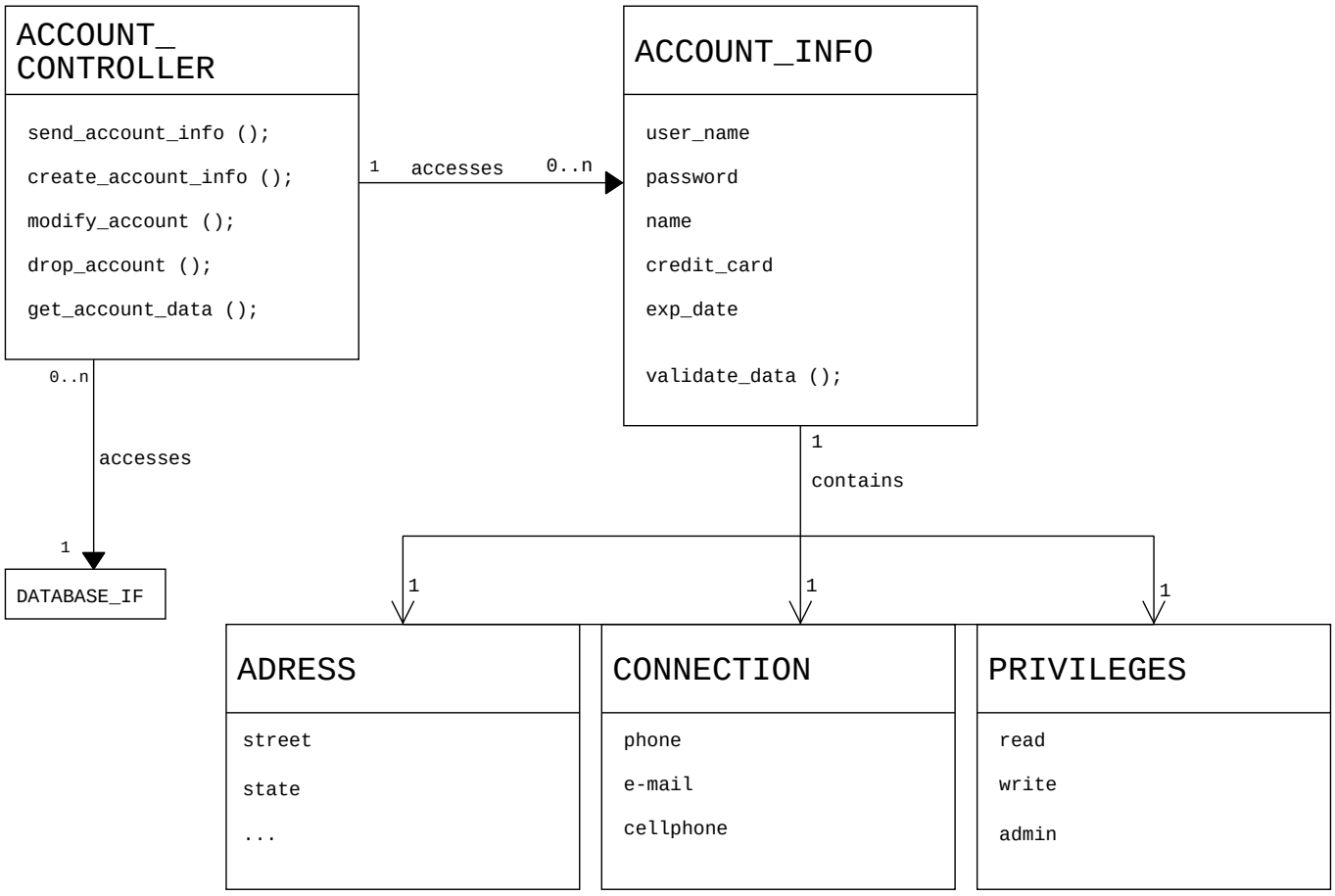
Chapter 6

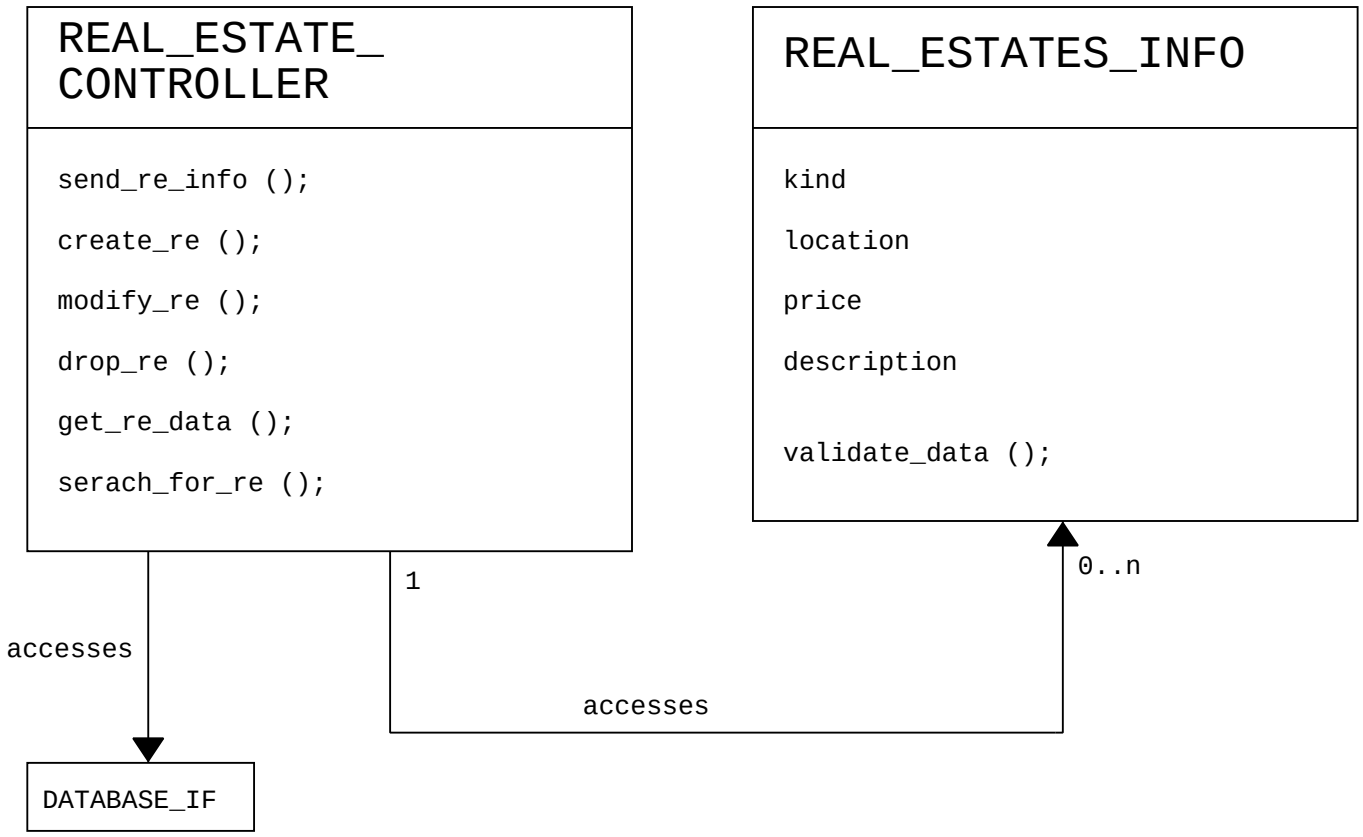
Class Diagrams

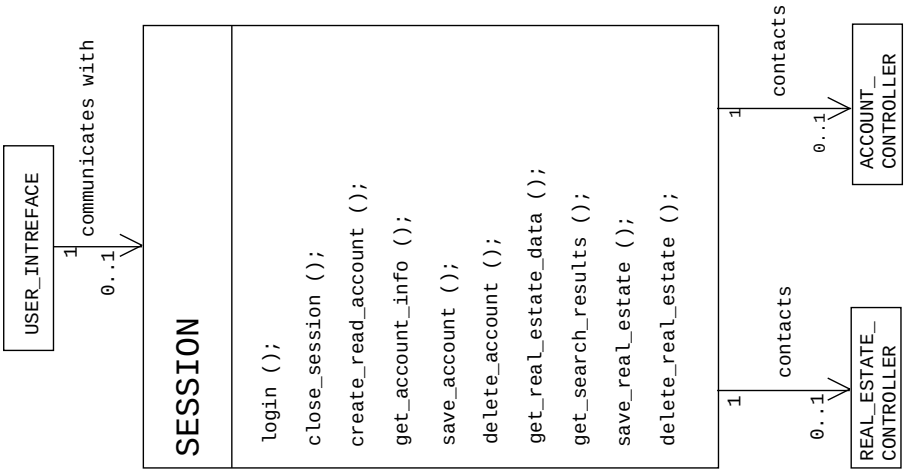
In the previous chapters we dealt with the conceptual model of our system. Now we are going to start with the object oriented analysis. We have to transform the CRC-Model into object-oriented class-diagrams. Therefore the CRC-Cards become Classes, a new class for each Card. A class contains attributes and essential methods. Furthermore there are different relations between the classes: one to one, on to many, many to many (many may also mean zero). Inheritance is also a point of interest. Thus attributes and methods of one class are forwarded to another one - the child-class -, which can also have additional attributes and methods.

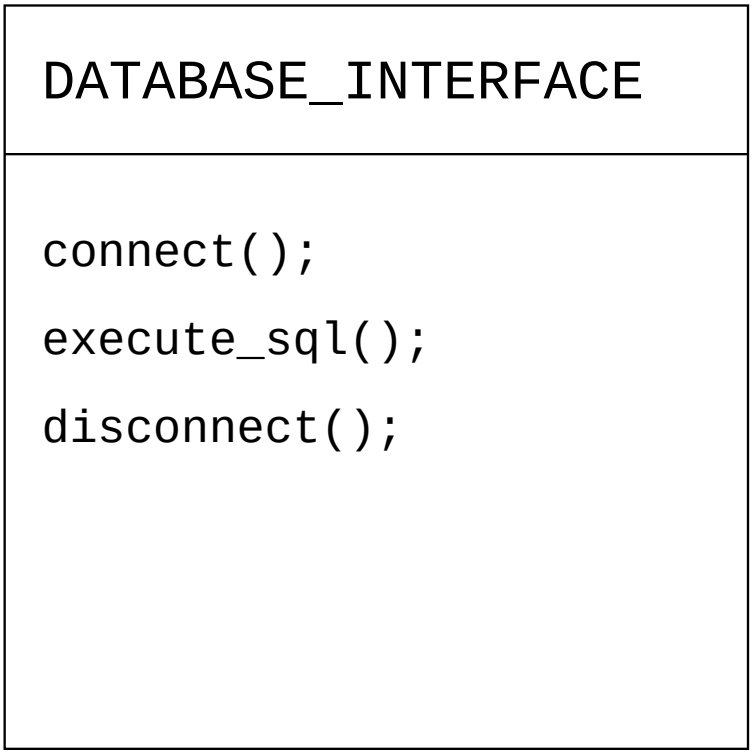



Session
1
communicates with
1
USER_INTERFACE
link-pressed(link): external
button_pressed(button):external
1
gets frontend
n
Layout
size
color
widgets
...


ACCOUNT_
CONTROLLER
send_account_info ();
create_account_info ();
modify_account ();
drop_account ();
get_account_data ();
1
accesses
0..n
ACCOUNT_INFO
user_name
password
name
credit_card
exp_date
validate_data ();
0..n
accesses
1
DATABASE_IF
1
contains
1
1
1
ADRESS
street
state
...
CONNECTION
phone
e-mail
cellphone
PRIVILEGES
read
write
admin


REAL_ESTATE_
CONTROLLER
send_re_info ();
create_re ();
modify_re ();
drop_re ();
get_re_data ();
serach_for_re ();
REAL_ESTATES_INFO
kind
location
price
description
validate_data ();
1
0..n
accesses
accesses
DATABASE_IF


USER_INTREFACE
communicates with
1
0..1
SESSION
login ();
close_session ();
create_read_account ();
get_account_info ();
save_account ();
delete_account ();
get_real_estate_data ();
get_search_results ();
save_real_estate ();
delete_real_estate ();
contacts
1
0..1
ACCOUNT_
CONTROLLER
contacts
1
0..1
REAL_ESTATE_
CONTROLLER


DATABASE_INTERFACE
connect();
execute_sql();
disconnect();

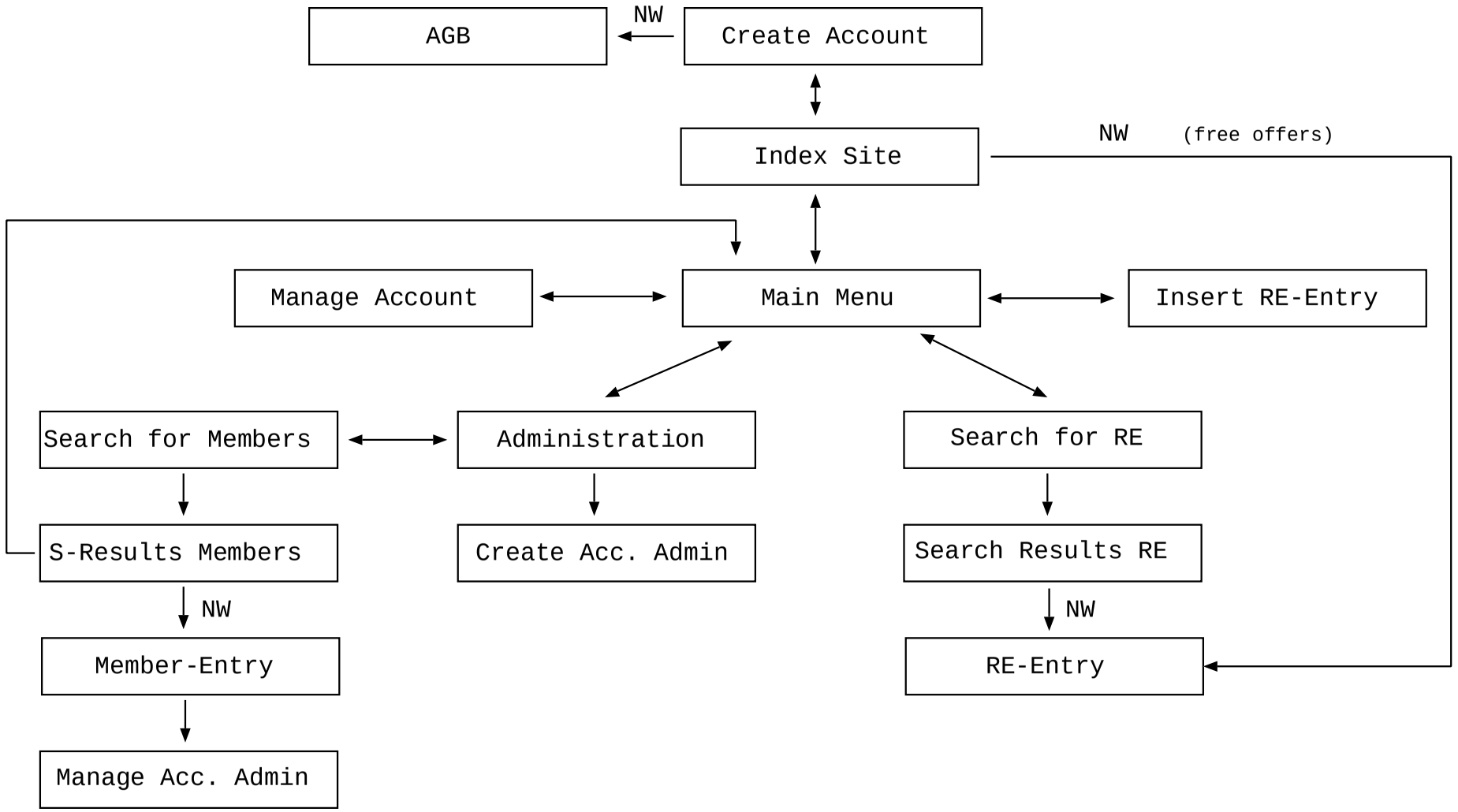
Chapter 7

User-Interface Prototyping

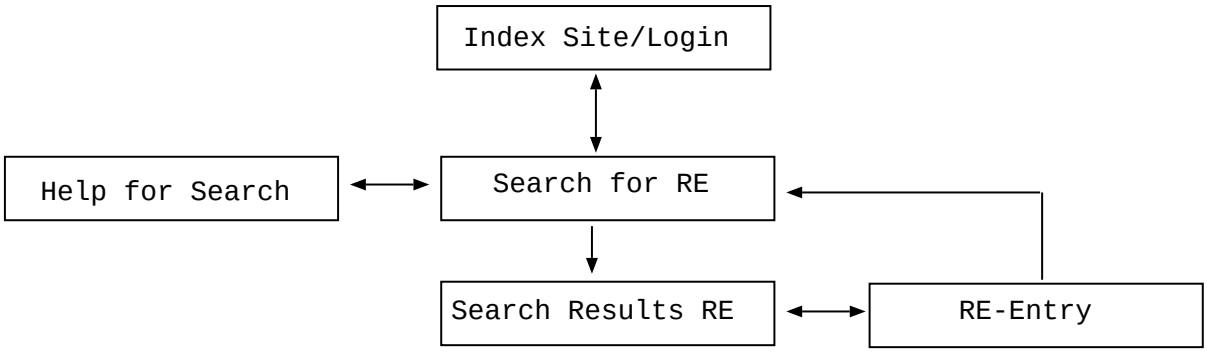
On the basis of the user-interface design we had developed during our previous session with the customer we created a corresponding prototype using HTML (Hypertext Markup Language) and WML (Wireless Markup Language). The purpose of user-interface prototyping is to test out in reality if the conceptual model works. Therefore we have to check if for every use-case a corresponding set of user-interface-sites exist. Another important point is to verify all the sites are interlinked and no unexpected dead ends exist.
The flow-diagram of the HTML and WML versions can be found on the succeeding pages. The prototypes of the Immowarrior user-interface can be found at the following locations:

- HTML version: http://www.cosy.sbg.ac.at/ TILDE mpober/se1/se_html/
- WML version: http://immowarrior.wap3.de/

User-Interface Prototype - HTML Version



User-Interface Prototype - WML Version



Chapter 8

Sequence Diagram

8.1 What is a Squence Diagram?

A sequence diagram shows an interaction arranged in time sequence. In particular, it shows the objects participating in the interaction by their "lifelines". It does not show the associations among the objects. The sequence diagram represents an Interaction, which is a set of messages exchanged among objects within a collaboration to effect a desired operation or result.
A sequence diagram has two dimensions: the vertical dimension represents time, the horizontal dimension represents different objects. Normally time proceeds down the page. Usually only time sequences are important but in real-time applications, the time axis could be an actual metric. There is no significance to the horizontal ordering of the objects.

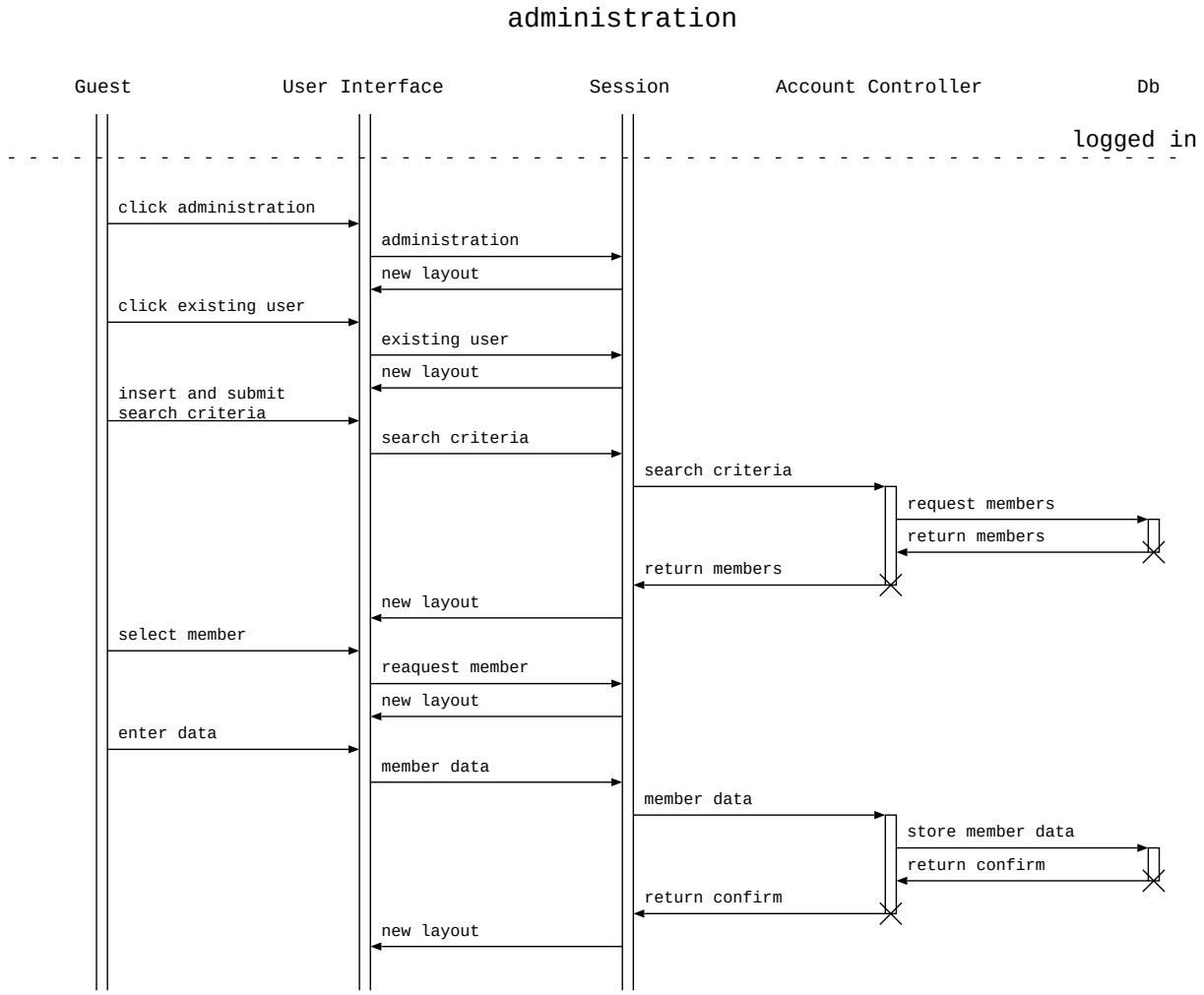
8.2 Modeling a Sequence Diagram

An objects in a sequence diagram is represented by a rectangle. The objects role is shown as a vertical dashed line called the "lifeline". This lifeline represents the existence of the object at a particular time. If the object is created or destroyed during the period of time shown on the diagram, then its lifeline starts or stops at the appropriate point; otherwise it goes from the top to the bottom of the diagram.
A message is a communication between objects that conveys information with the expectation that action will ensue. The receipt of a message is a kind of event. A message is shown as a horizontal solid arrow from the lifeline of one object to the lifeline of another object. In case of a message from an object to itself, the arrow may start and finish on the same object symbol. The arrow is labeled with the name of the message and its argument values. The arrow may also be labeled with a sequence number to show the sequence of the message in the overall interaction. Sequence numbers are useful on the diagrams for identifying concurrent threads of control.

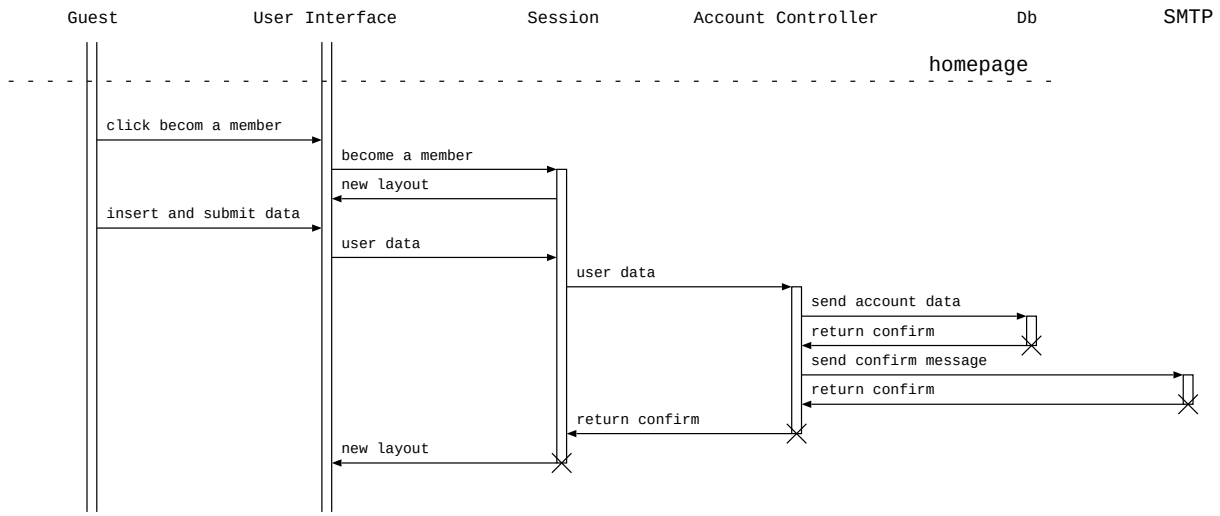
8.3 When to use a Sequence Diagram

A good design can have lots of small methodes in different classes. Because of this it can be difficult to figure out the overall sequence of behavior. This diagram is simple and visually logical, so it easy to see the sequence of the flow of control. The diagram also clearly shows concurrent processes and activations.

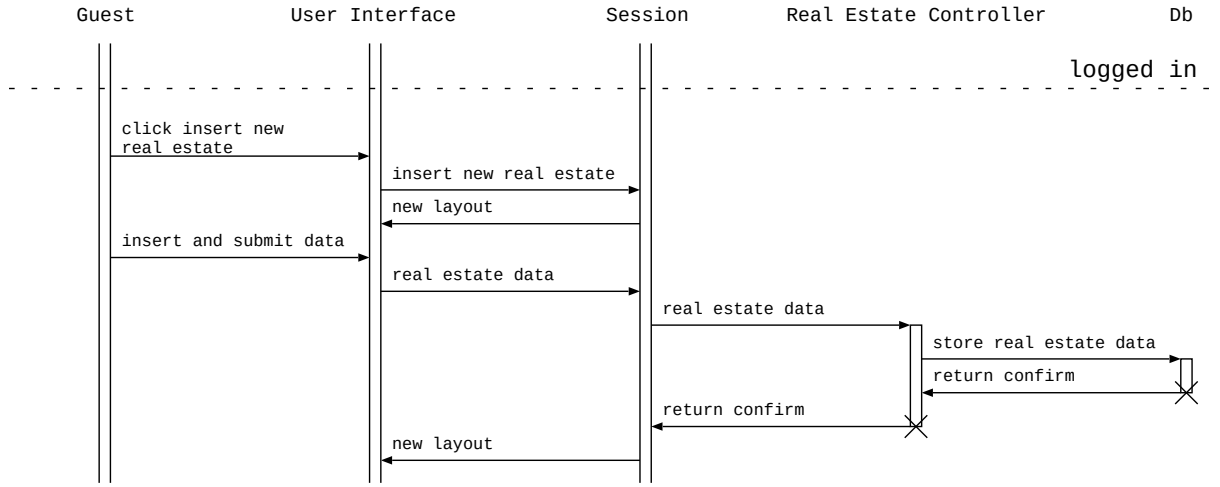
8.4 Our Sequence Diagrams



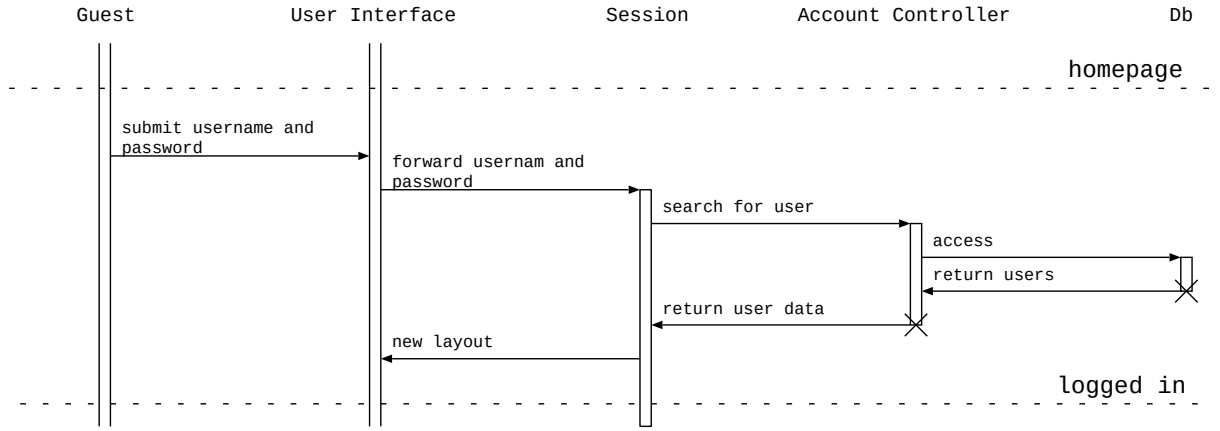
become a member



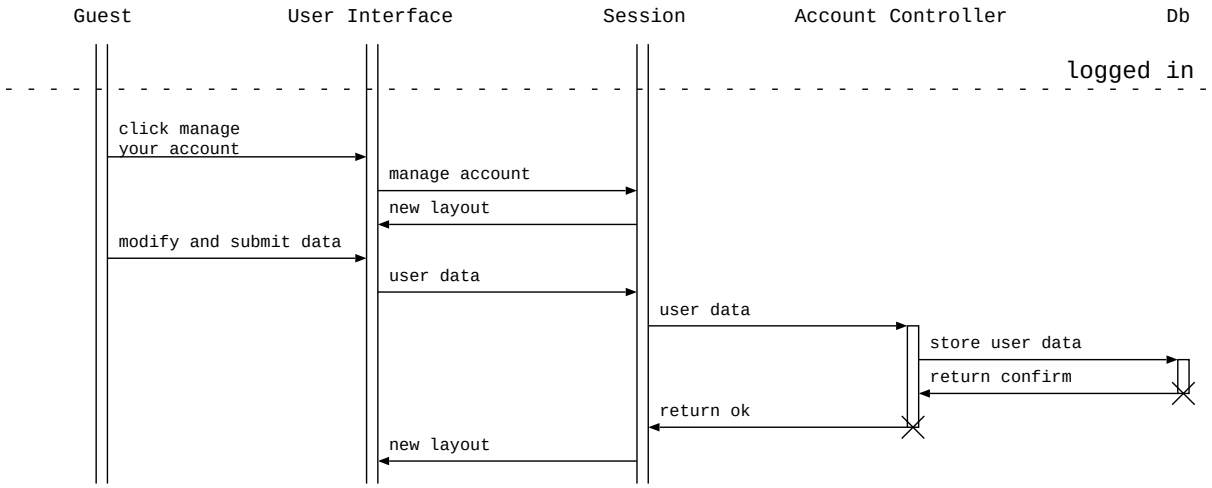
insert new eral estate



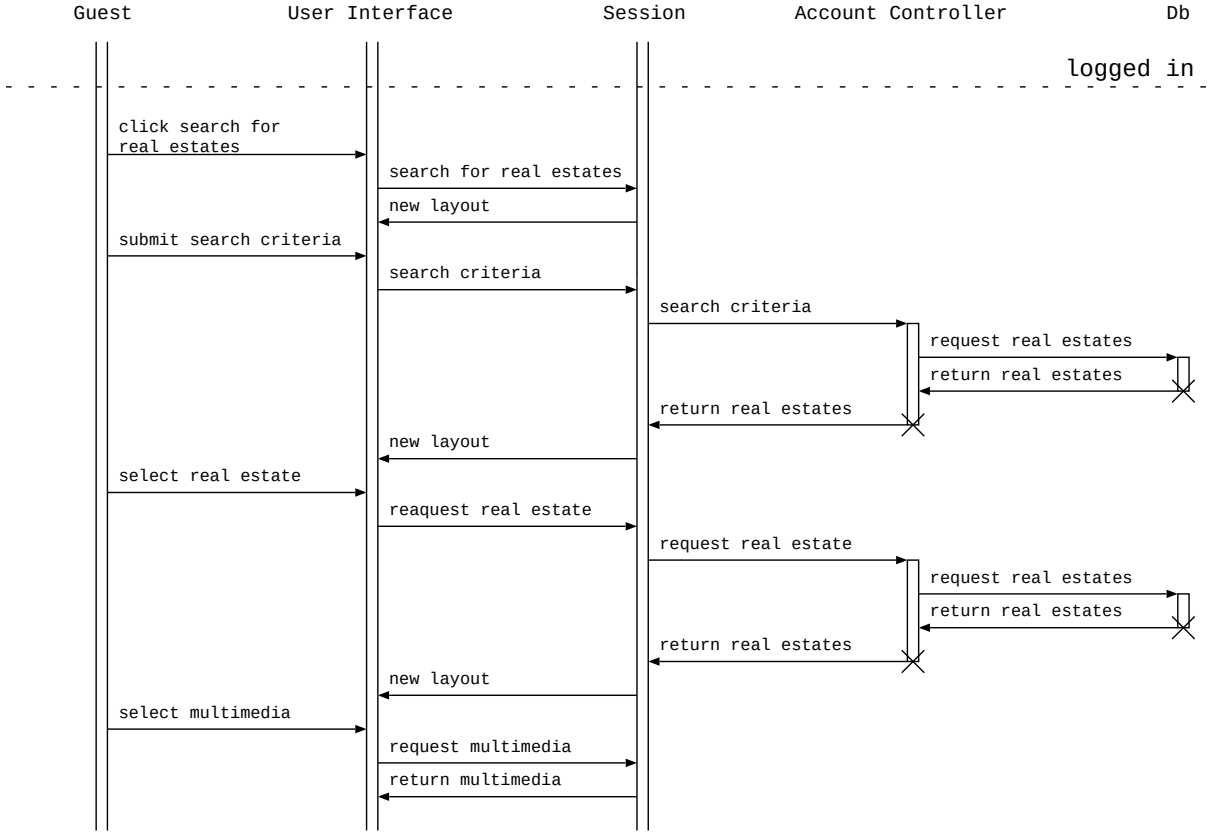
login



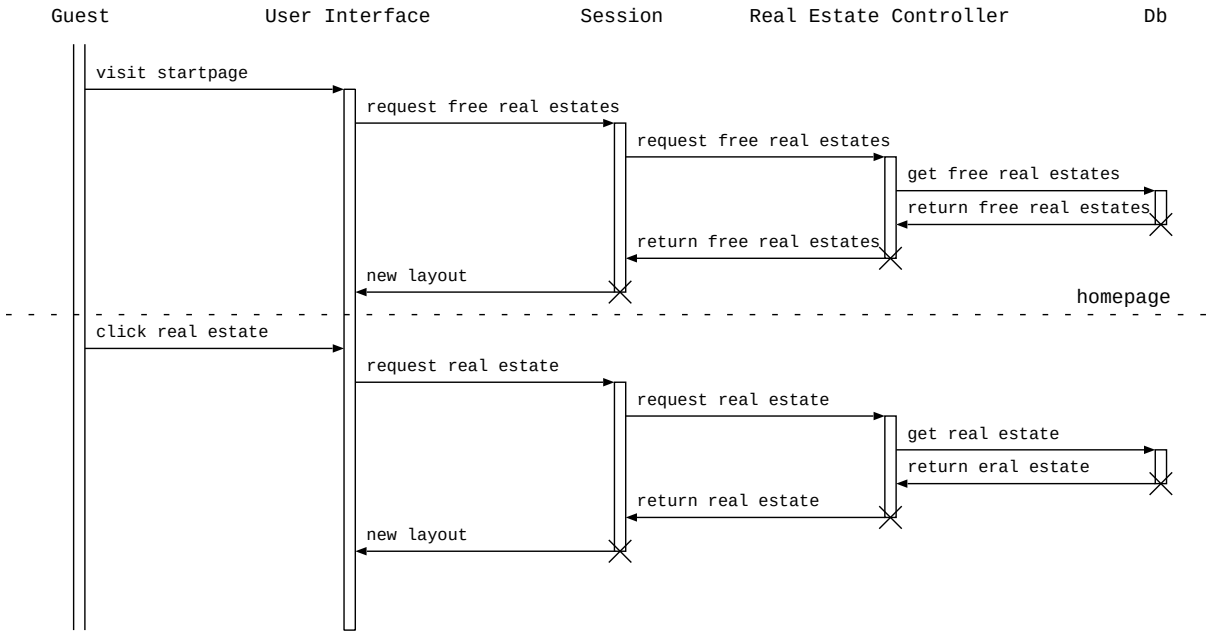
manage your account



serach for real estates



visit startpage/view free offers



Chapter 9

Activity Diagram

9.1 What is an Activity Diagram?

Activity diagrams describe the workflow behavior of a system. The diagram illustrates the dynamic nature of a system by modeling the flow of control from activity to activity. An activity represents an operation on some class in the system that results in a change in the state of the system. An activity is trigged by one or more events and activity may result in one or more events that may trigger other activity or processes.
Typically an activity diagram is attached to the implementation of an use case. The purpose of this diagram is to focus on flows driven by internal processing (as opposed to external events)

9.2 Modeling an Activity Diagram

Basic elemtents for creating an anctivity diagram:

Action State: Action states represent the noninterruptible actions of objects. Typically an action state is represented by an rectangle with convex arcs as sides.

Action Flow: Action flow arrows illustrate the relationships among action states.

Object Flow: Object flow refers to the creation and modification of objects by activities. An object flow arrow from an action to an object means that the action creates or influences the object. An object flow arrow from an object to an action indicates that the action state uses the object.

Initial State: A filled circle followed by an arrow represents the initial action state.

Final State: An arrow pointing to a filled circle nested inside another circle represents the final action state.

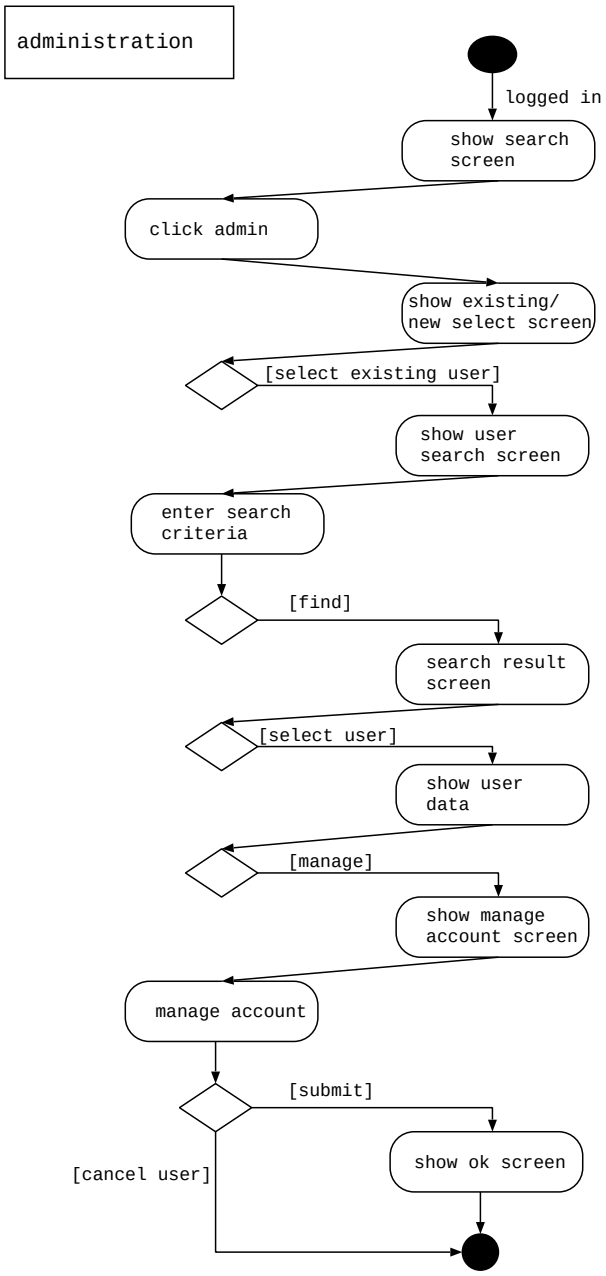
Branching: A diamond represents a decision with alternate paths. The outgoing alternates should be labeled with a condition or guard expression. You can also label one of the paths "else."

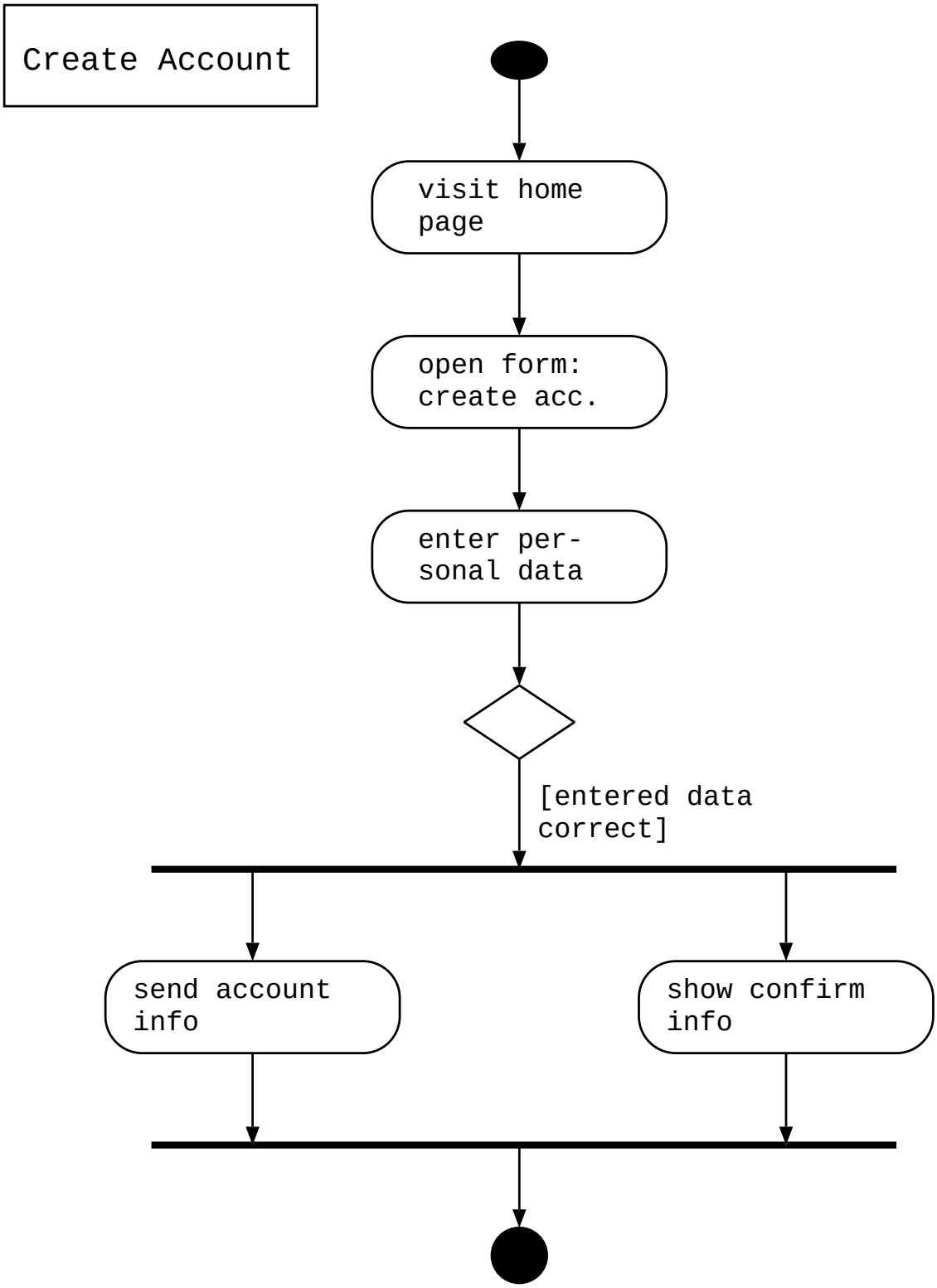
Swimlanes: The activity diagram may be divided visually into "swimlanes" each separated from neighboring swimlanes by vertical solid lines on both sides. Swimlanes are used to group related activities into one column.

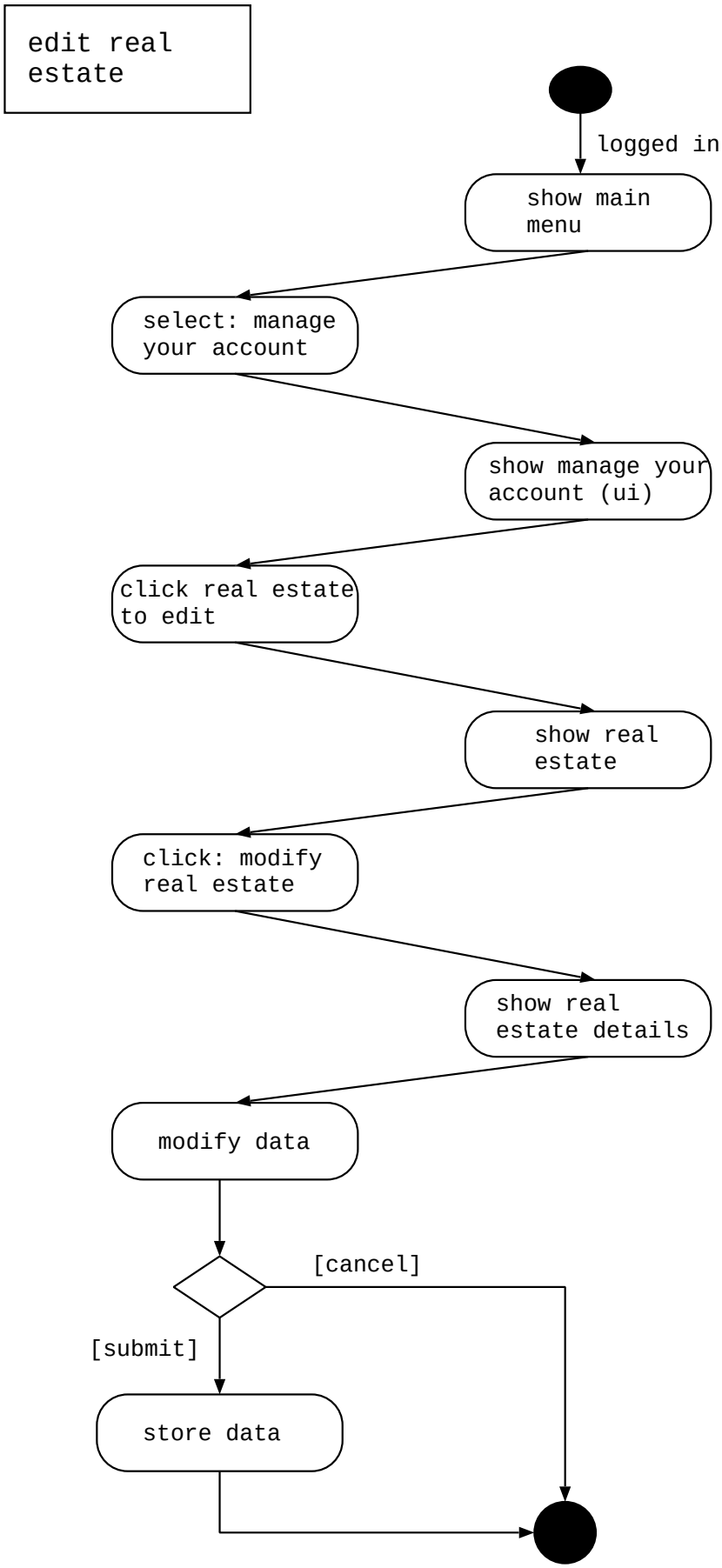
9.3 When to use an Activity Diagram

Activity diagrams are used for documenting existing process, analyzing new Process Concepts, identifying and finding reengineering opportunities. They are mostly used to show parallel behavior between the different events and activities. Activity diagrams are also used to document decisions and iterations. From the software engineering perspective they can be used to analyze requirements of the use cases befor modeling and assigning methods.

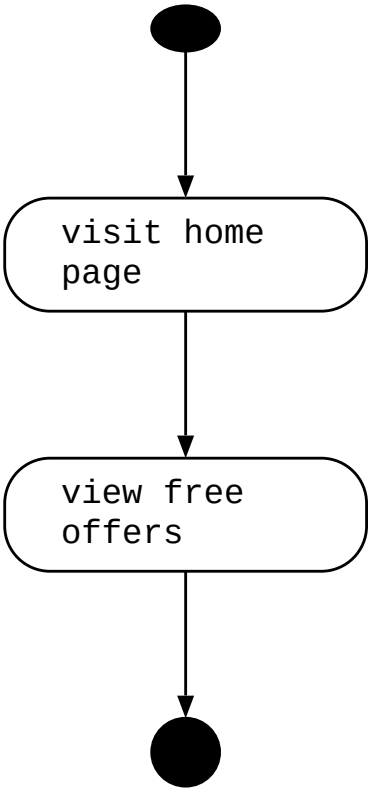
9.4 Our Activity Diagrams

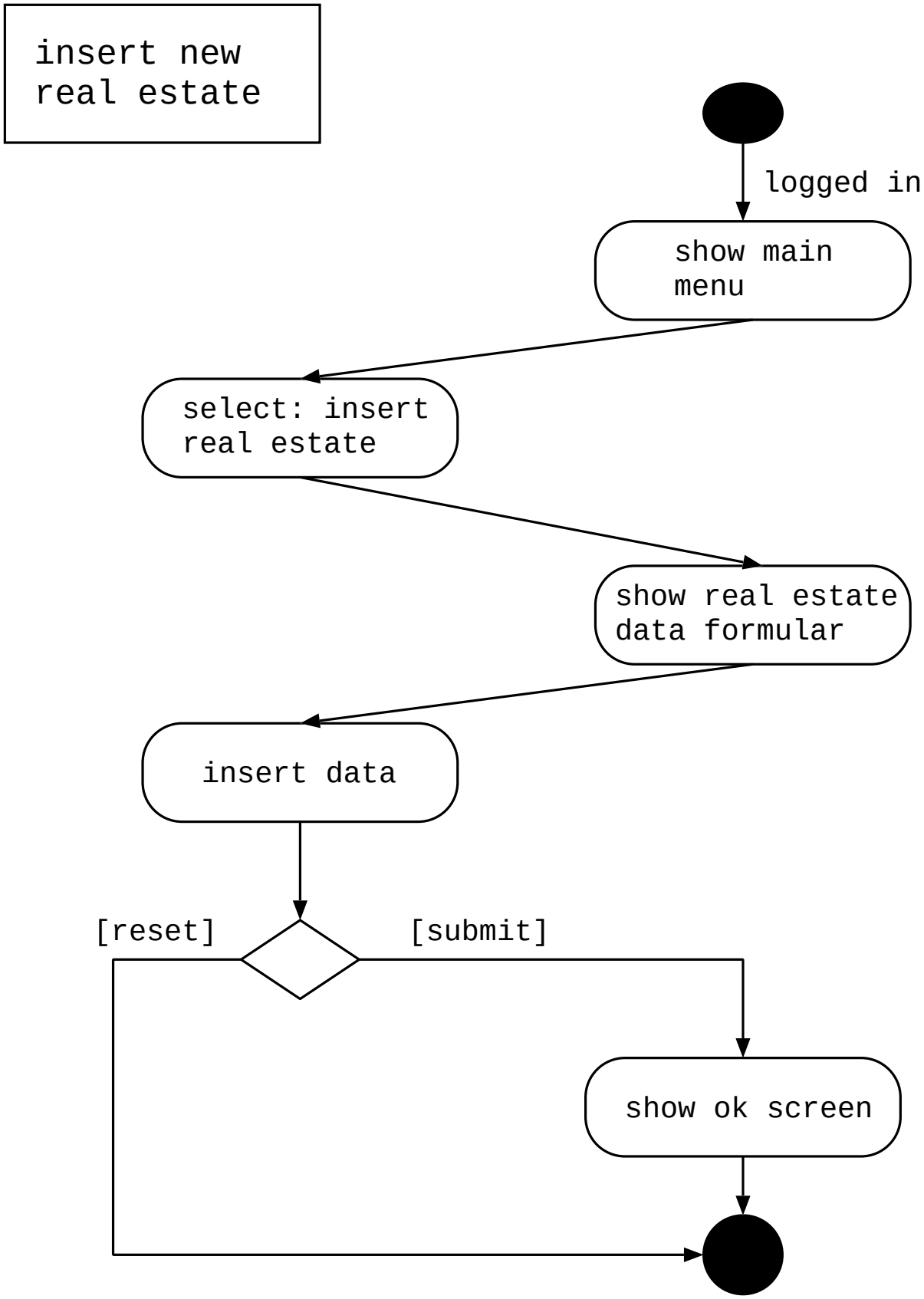


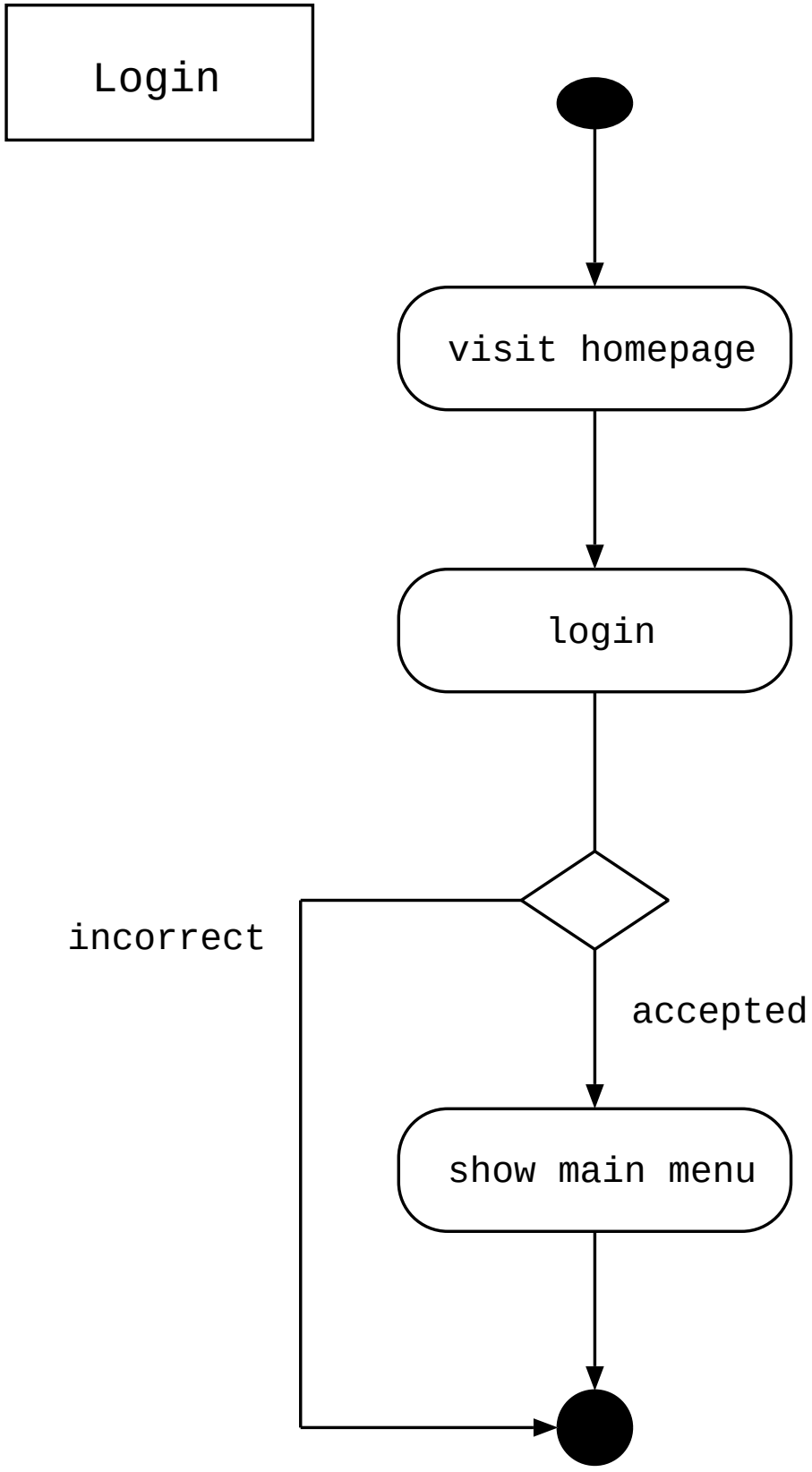

Create Account
visit home page
open form: create acc.
enter per-sonal data
[entered data correct]
send account info
show confirm info

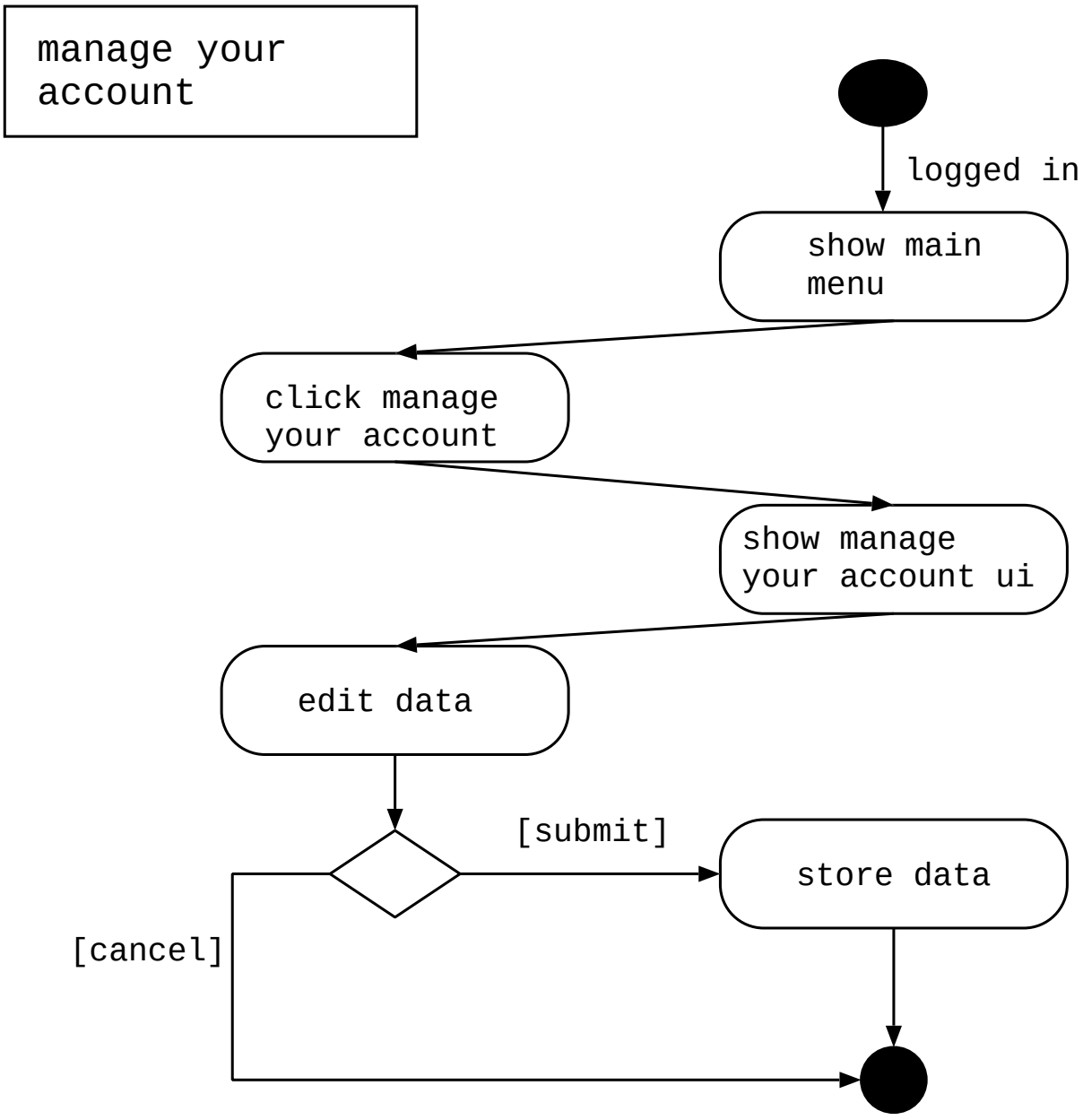

edit real estate
logged in
show main menu
select: manage your account
show manage your account (ui)
click real estate to edit
show real estate
click: modify real estate
show real estate details
modify data
[cancel]
[submit]
store data

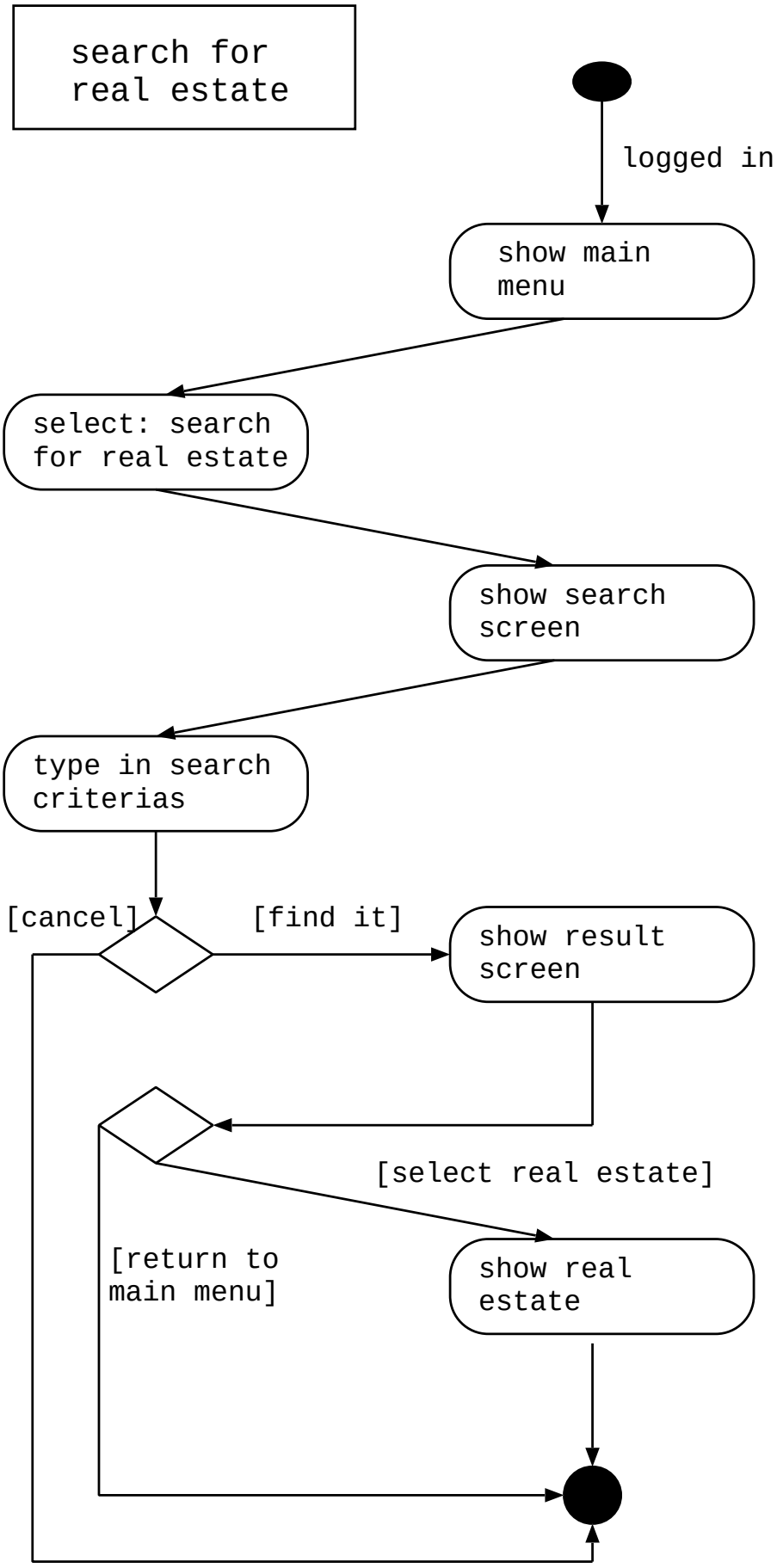
viewing free offers

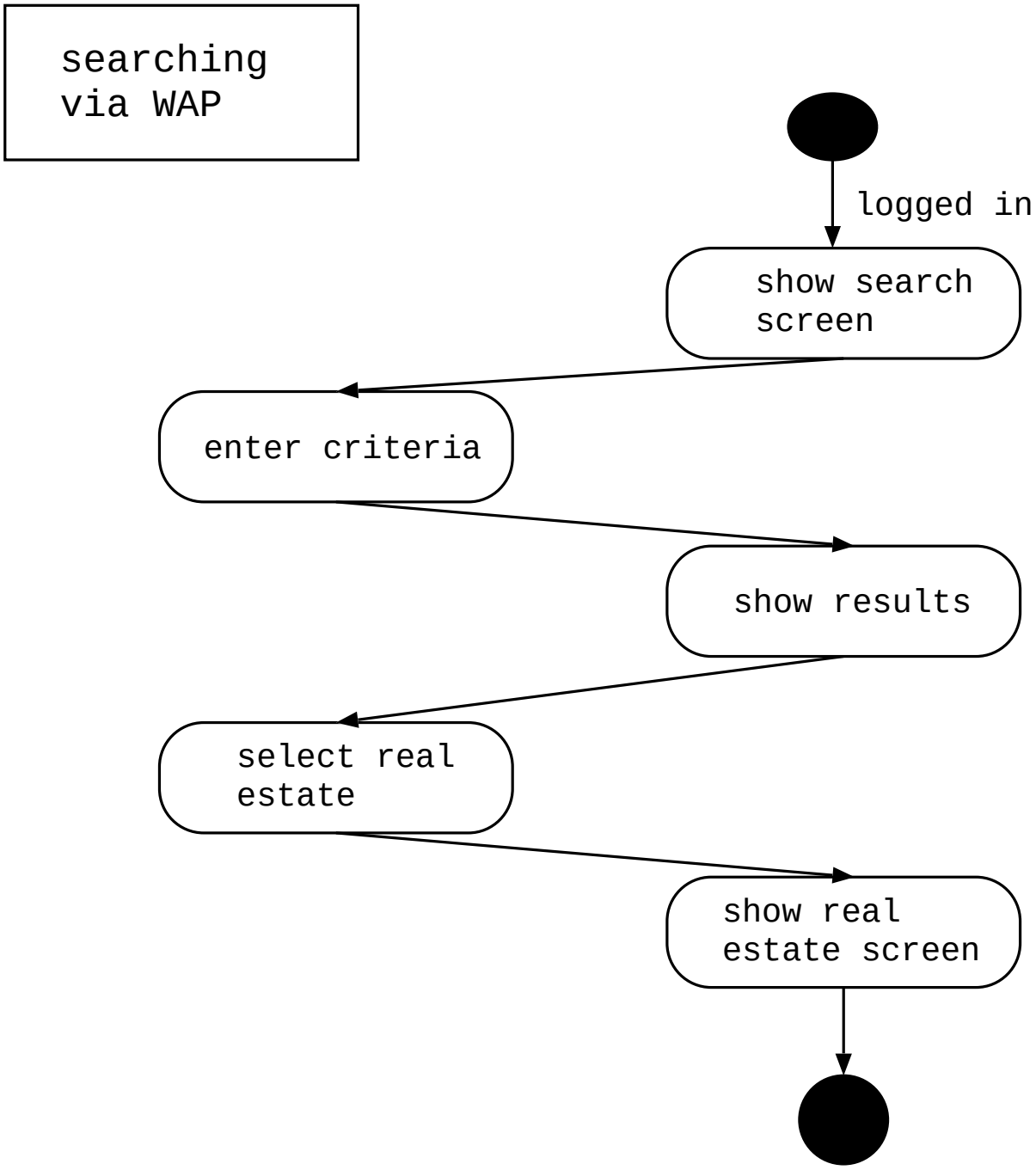



insert new real estate
logged in
show main menu
select: insert real estate
show real estate data formular
insert data
[reset]
[submit]
show ok screen


Login
visit homepage
login
incorrect
accepted
show main menu


manage your account
logged in
show main menu
click manage your account
show manage your account ui
edit data
[submit]
store data
[cancel]


search for real estate
logged in
show main menu
select: search for real estate
show search screen
type in search criterias
[cancel]
[find it]
show result screen
[select real estate]
[return to main menu]
show real estate


searching via WAP
logged in
show search screen
enter criteria
show results
select real estate
show real estate screen

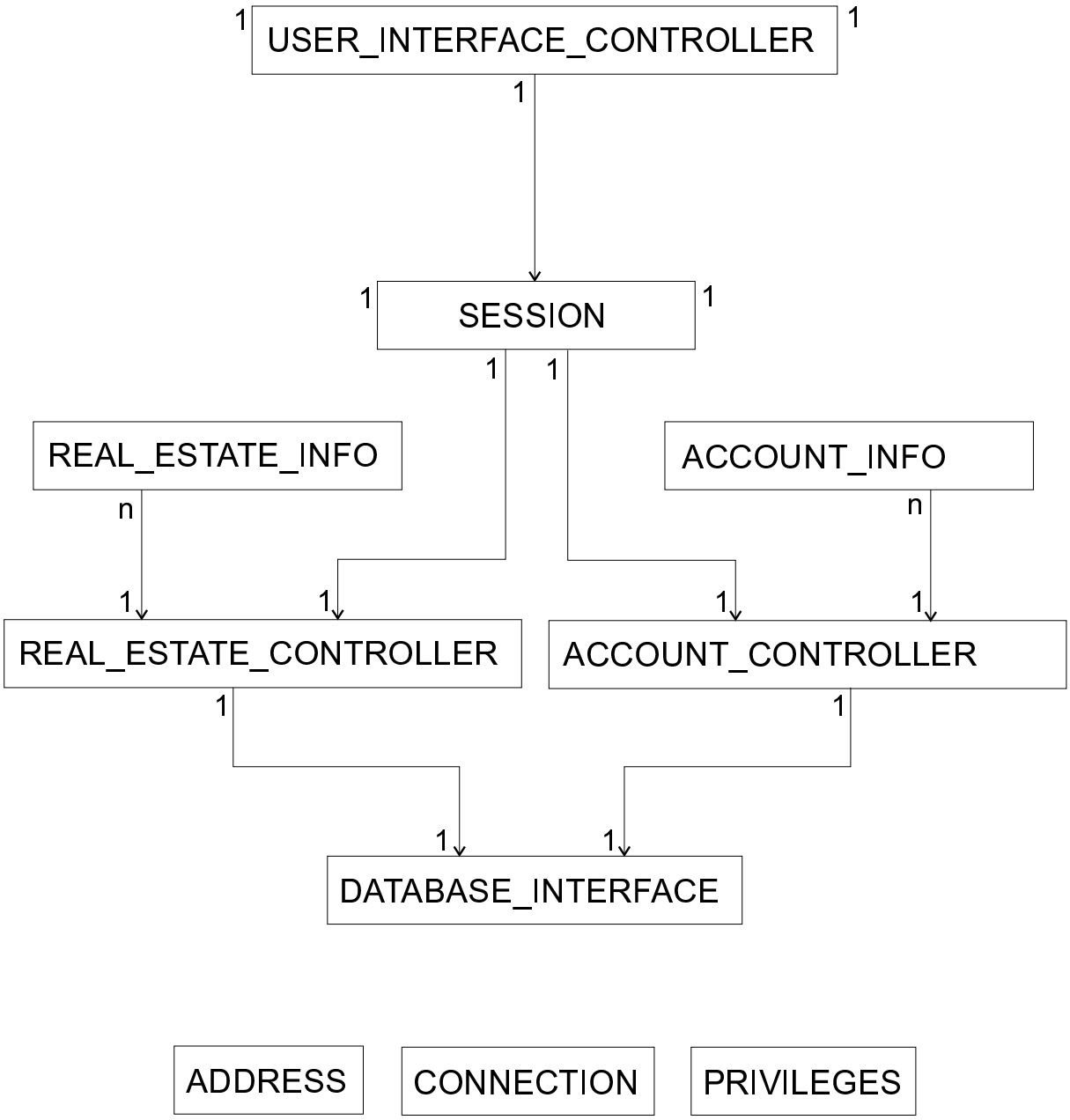
Chapter 10

Design Class Diagram

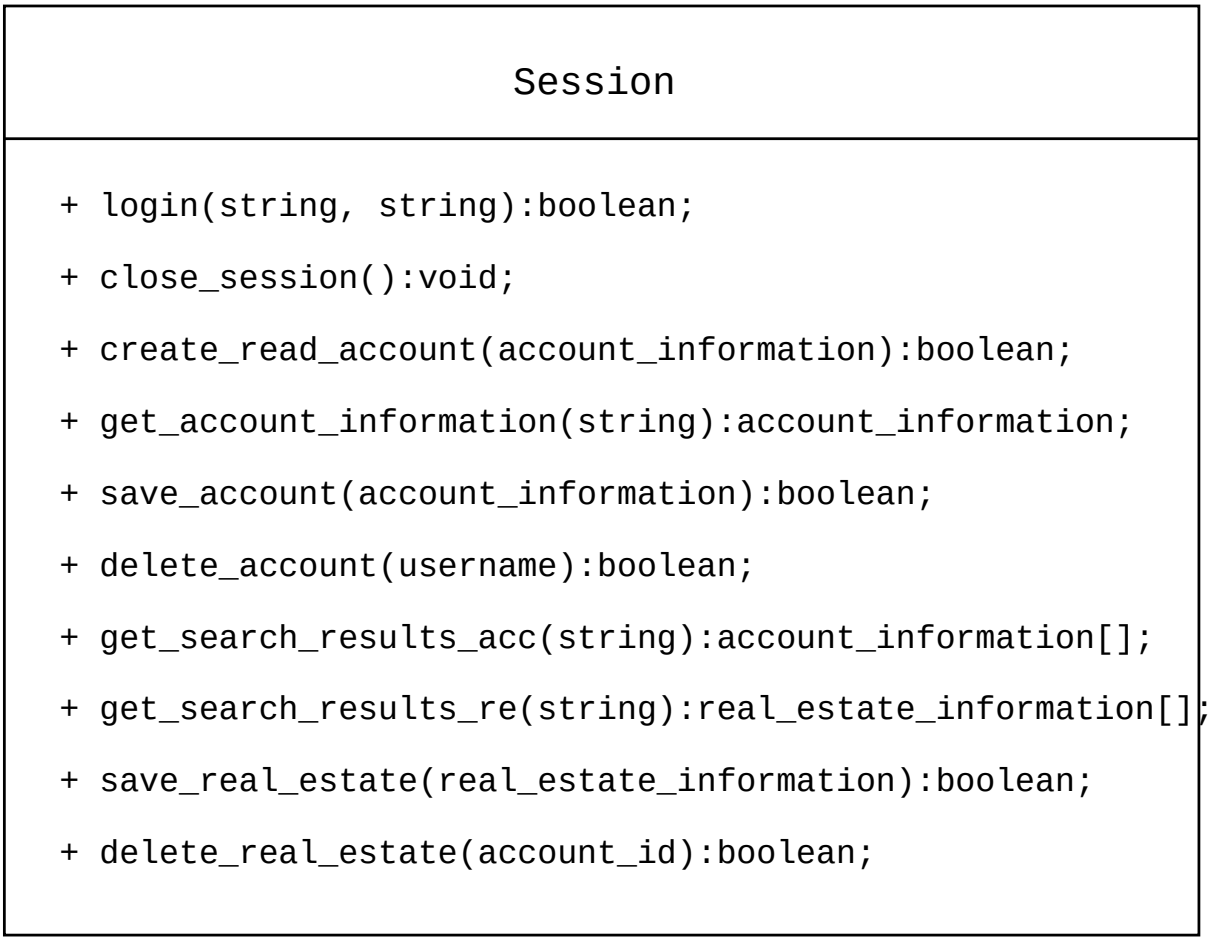
10.1 What is a Design Class Diagram

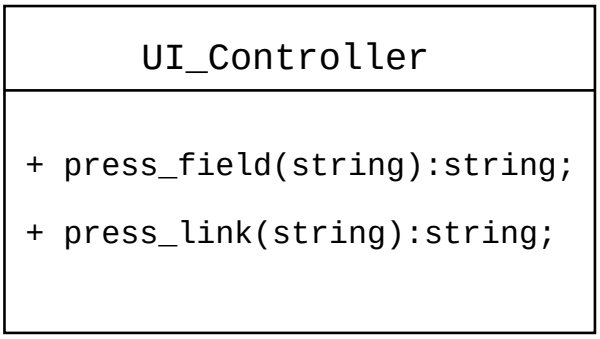
Activity and Sequence Diagrams are the end of the analysis phase. Now we switch from analysis to design phase. The first thing to do is to create a design class diagram.
The goal of a Design Class Diagramm is to model the static appearance of the system. Unlike the analysis in which we concentrated on the problem domain, we focus on the solution domain. Changes are now caused by different implementation technologies. In this diagram the classes and their attributes and methods are shown. Additionally the visibilities (public +, protected #, private -), names, paramters and return values of methods as well as visibilities, names and types of attributes are listed.

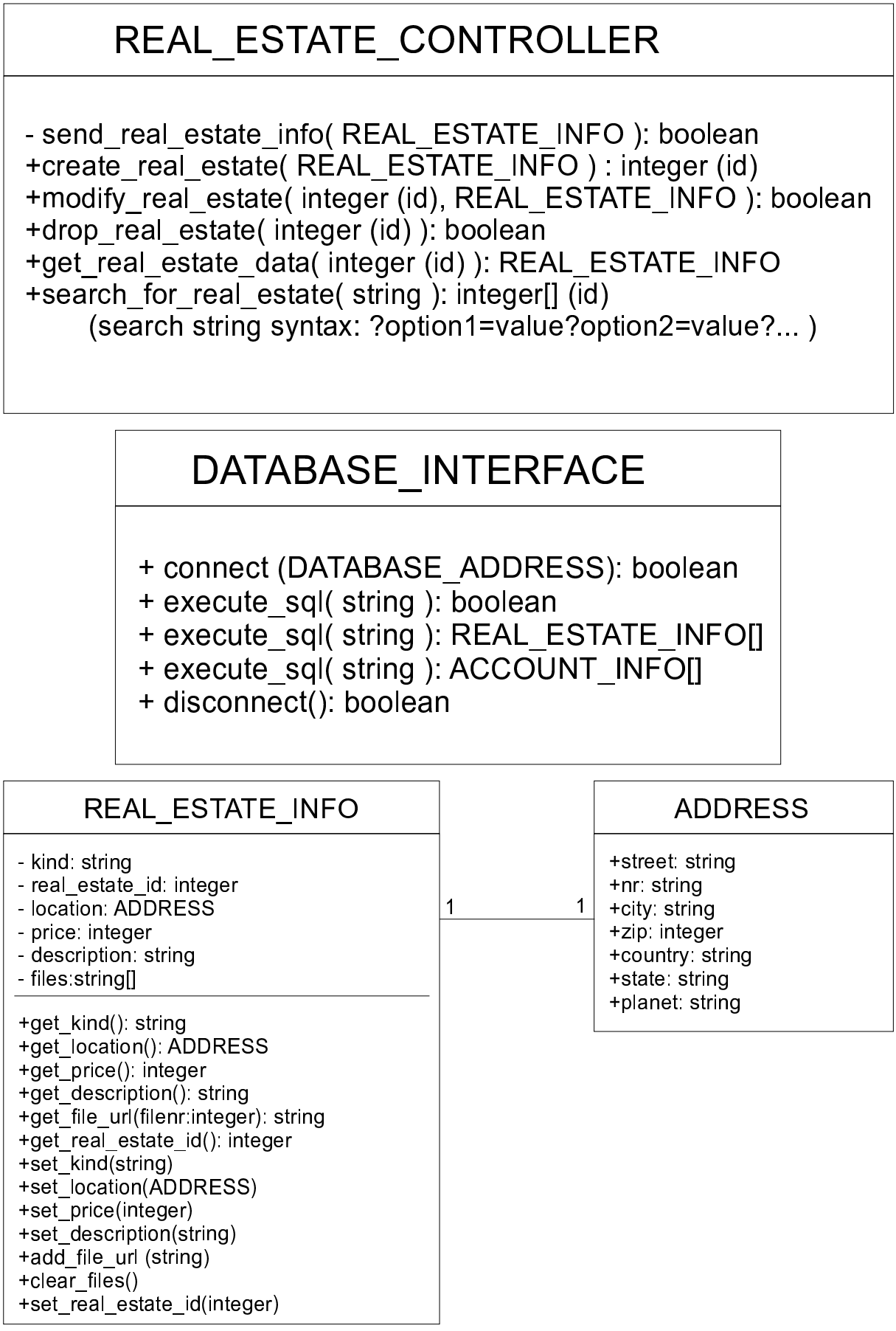
10.2 Our Design Class Diagram




ACCOUNT_INFO
-account_id : integer
-user_name : string
-password : string
-privileges : PRIVILEGES
-family_name : string
-surname : string
-address : ADDRESS
-connection_info : CONNECTION
-credit_card : string
-expiry_date : string
+set_account_id(account_id: integer)
+set_user_name(user_name: string)
+set_password(password: string)
+set_privileges(privileges: PRIVILEGES)
+set_family_name(family_name: string)
+set_surname(surname: string=
+set_address(address: ADDRESS)
+set_connection_info(connection_info: CONNECTION)
+set_credit_card(credit_card: string)
+set_expiry_date(expiry_date: string)
+get_account_id():integer
+get_user_name():string
+get_password():string
+get_privileges():PRIVILEGES
+get_family_name():string
+get_surname():string
+get_address():ADDRESS
+get_connection_info():CONNECTION
+get_credit_card():string
+get_expiry_date():string
CONNECTION
-cell_phone : string
-email_address : string
-phone : string
+set_cell_phone(cell_phone_number: string)
+set_email_adress(email_adress: string)
+set_phone(phone: string)
+get_cell_phone():string
+get_email_address():string
+get_phone():string
PRIVILEGES
-prileges : integer
+set_privileges(privileges: integer)
+get_privileges():integer
ACCOUNT_CONTROLLER
-send_account_info(account_id: integer)
+create_account(new_account: ACCOUNT_INFO):integer
+modify_account(account_id: integer, account: ACCOUNT_INFO)
+drop_account(account_id: integer):boolean
+get_account_data(account_id: integer):ACCOUNT_INFO
+search_for_account(search_criterias: string):account_id_array


Session
+ login(string, string):boolean;
+ close_session():void;
+ create_read_account(account_information):boolean;
+ get_account_information(string):account_information;
+ save_account(account_information):boolean;
+ delete_account(username):boolean;
+ get_search_results_acc(string):account_information[];
+ get_search_results_re(string):real_estate_information[];
+ save_real_estate(real_estate_information):boolean;
+ delete_real_estate(account_id):boolean;


UI_Controller
+ press_field(string):string;
+ press_link(string):string;


REAL_ESTATE_CONTROLLER
- send_real_estate_info(REAL_ESTATE_INFO): boolean
+create_real_estate(REAL_ESTATE_INFO) : integer (id)
+modify_real_estate(integer (id), REAL_ESTATE_INFO): boolean
+drop_real_estate(integer (id)): boolean
+get_real_estate_data(integer (id)): REAL_ESTATE_INFO
+search_for_real_estate(string): integer[] (id)
(search string syntax: ?option1=value?option2=value?...)
DATABASE_INTERFACE
+ connect (DATABASE_ADDRESS): boolean
+ execute_sql(string): boolean
+ execute_sql(string): REAL_ESTATE_INFO[]
+ execute_sql(string): ACCOUNT_INFO[]
+ disconnect(): boolean
REAL_ESTATE_INFO
- kind: string
- real_estate_id: integer
- location: ADDRESS
- price: integer
- description: string
- files:string[]
+get_kind(): string
+get_location(): ADDRESS
+get_price(): integer
+get_description(): string
+get_file_url(filenr:integer): string
+get_real_estate_id(): integer
+set_kind(string)
+set_location(ADDRESS)
+set_price(integer)
+set_description(string)
+add_file_url (string)
+clear_files()
+set_real_estate_id(integer)
1
1
ADDRESS
+street: string
+nr: string
+city: string
+zip: integer
+country: string
+state: string
+planet: string

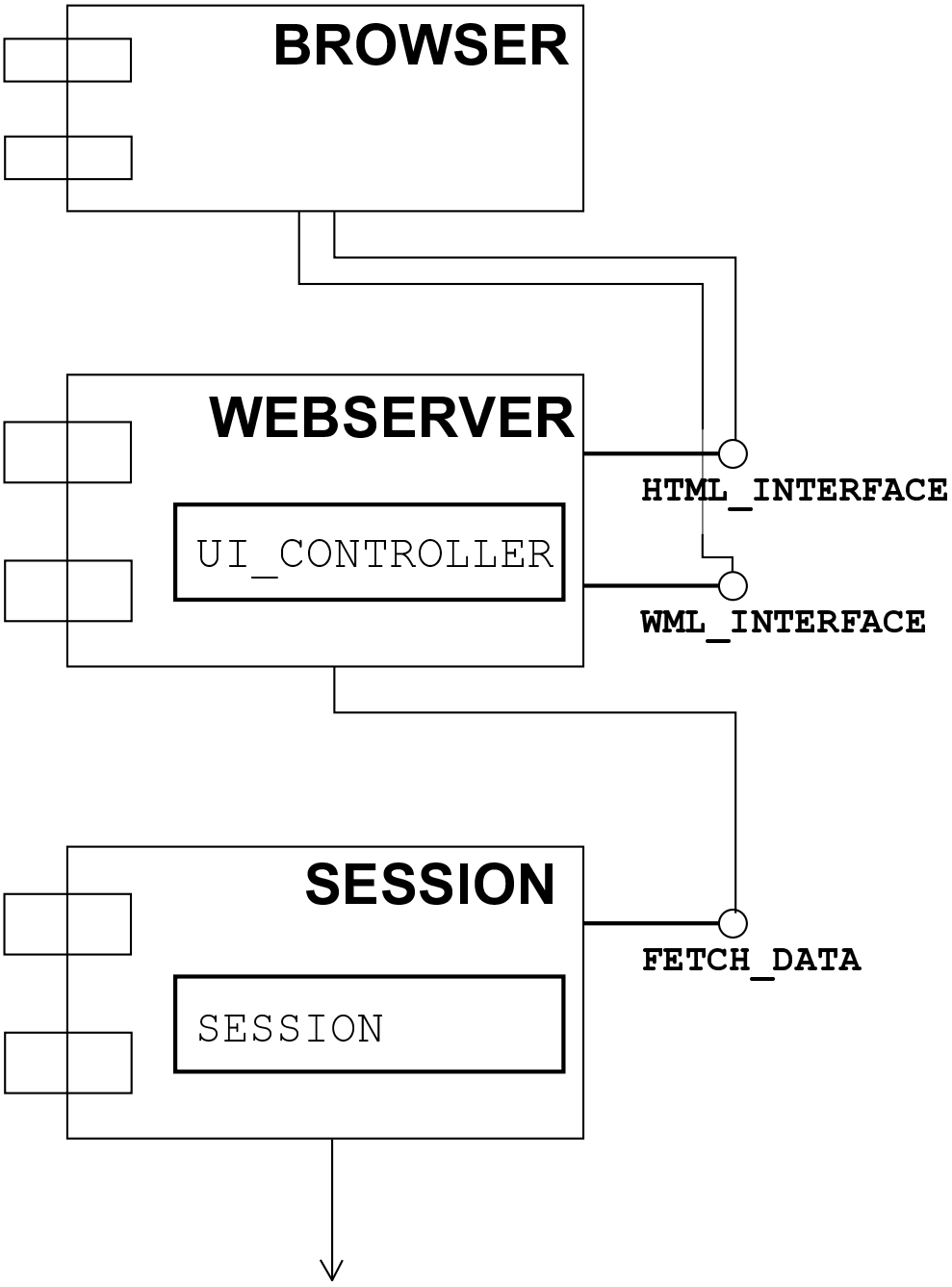
Chapter 11

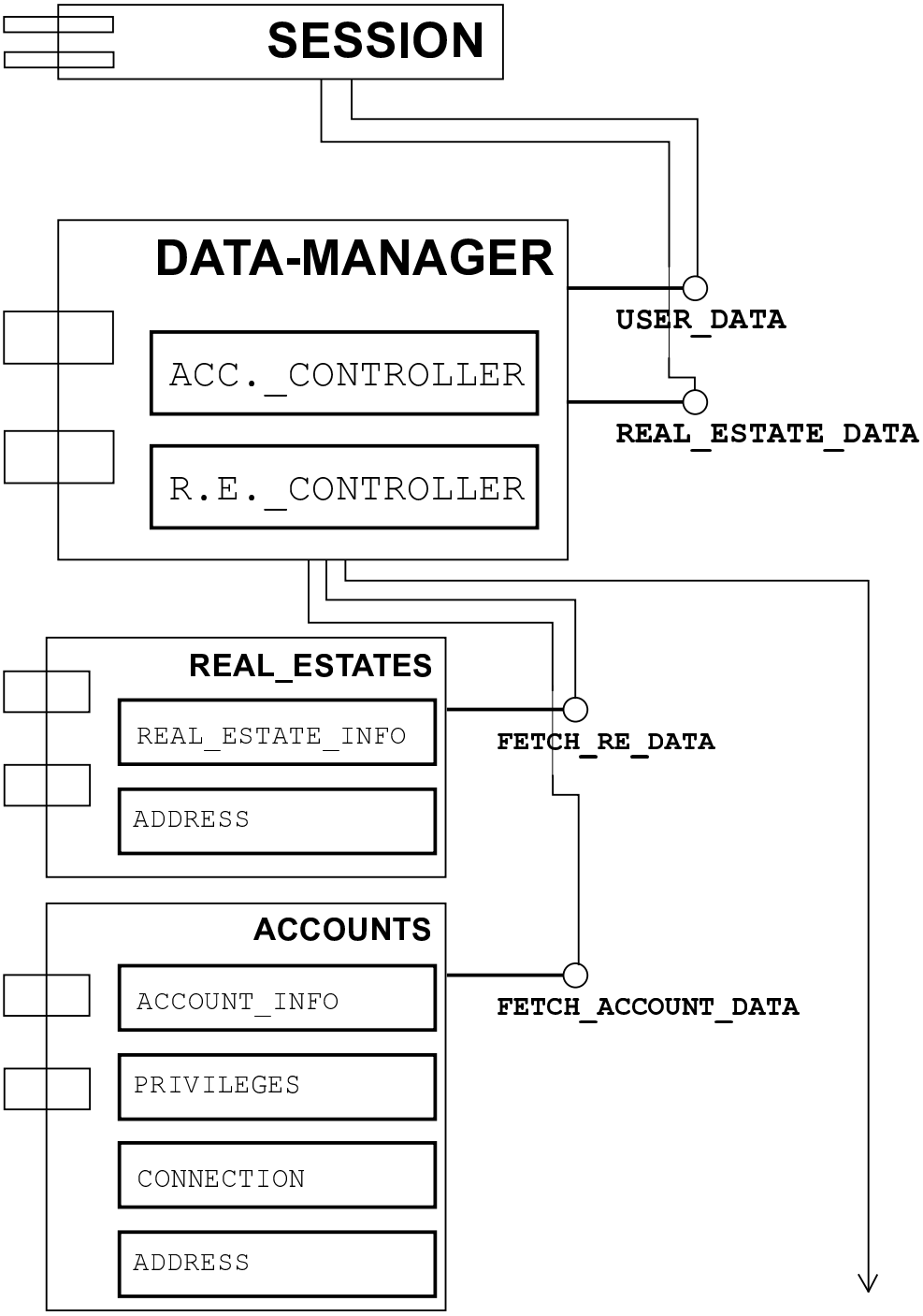
Component Diagram

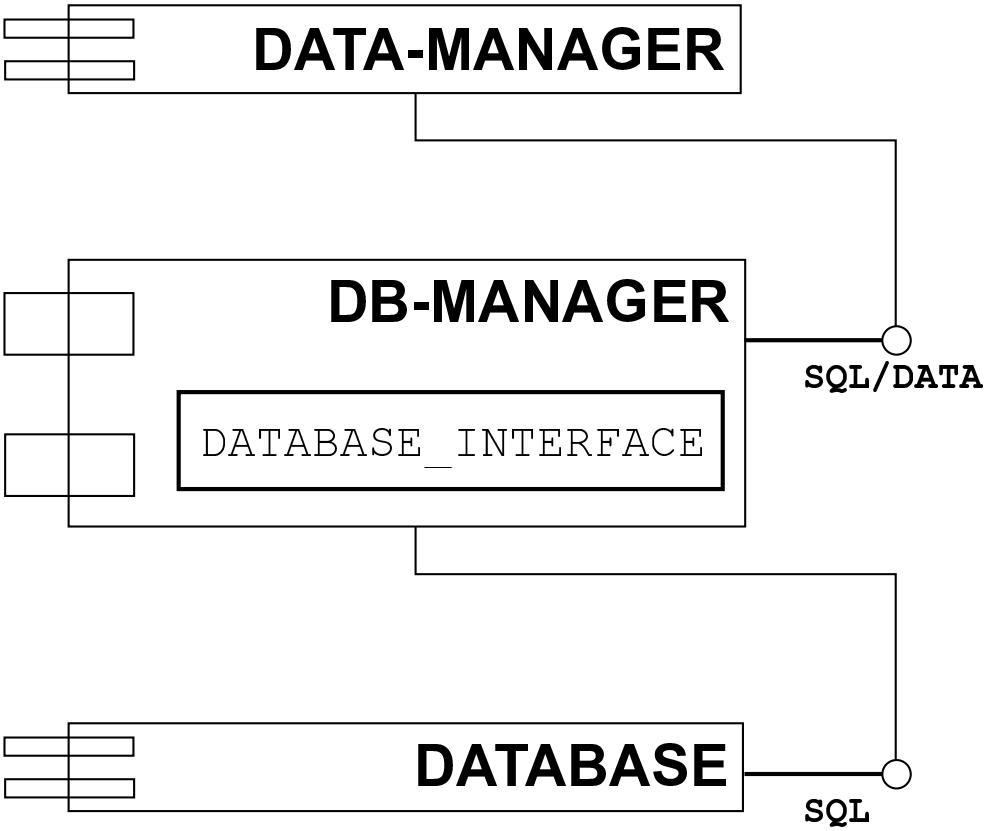
11.1 What is a Component Diagram

Component Diagrams show software components and the dependencies between them. The components can exist during compilation, linking or execution. Component Diagrams show the components just as types, not as instances. Component instances are visualized in the deployment diagram. Components are modeled as rectangles with two smaller rectangles jutting out from the left hand side and the name of the component written in the rectangle. Components implement one or more interfaces, modeled using a line with a circle at the end and the name of the interface beside the symbol. Components have dependencies on the interfaces of other components, modeled using the standard UML dependency notation, as seen in our component diagrams.

11.2 Our Component Diagrams




SESSION
DATA-MANAGER
ACC._CONTROLLER
R.E._CONTROLLER
USER_DATA
REAL_ESTATE_DATA
REAL_ESTATES
REAL_ESTATE_INFO
ADDRESS
FETCH_RE_DATA
ACCOUNTS
ACCOUNT_INFO
PRIVILEGES
CONNECTION
ADDRESS
FETCH_ACCOUNT_DATA


DATA-MANAGER
DB-MANAGER
DATABASE_INTERFACE
SQL/DATA
DATABASE
SQL

Chapter 12

Deployment Diagram

12.1 What is a Deployment Diagram?

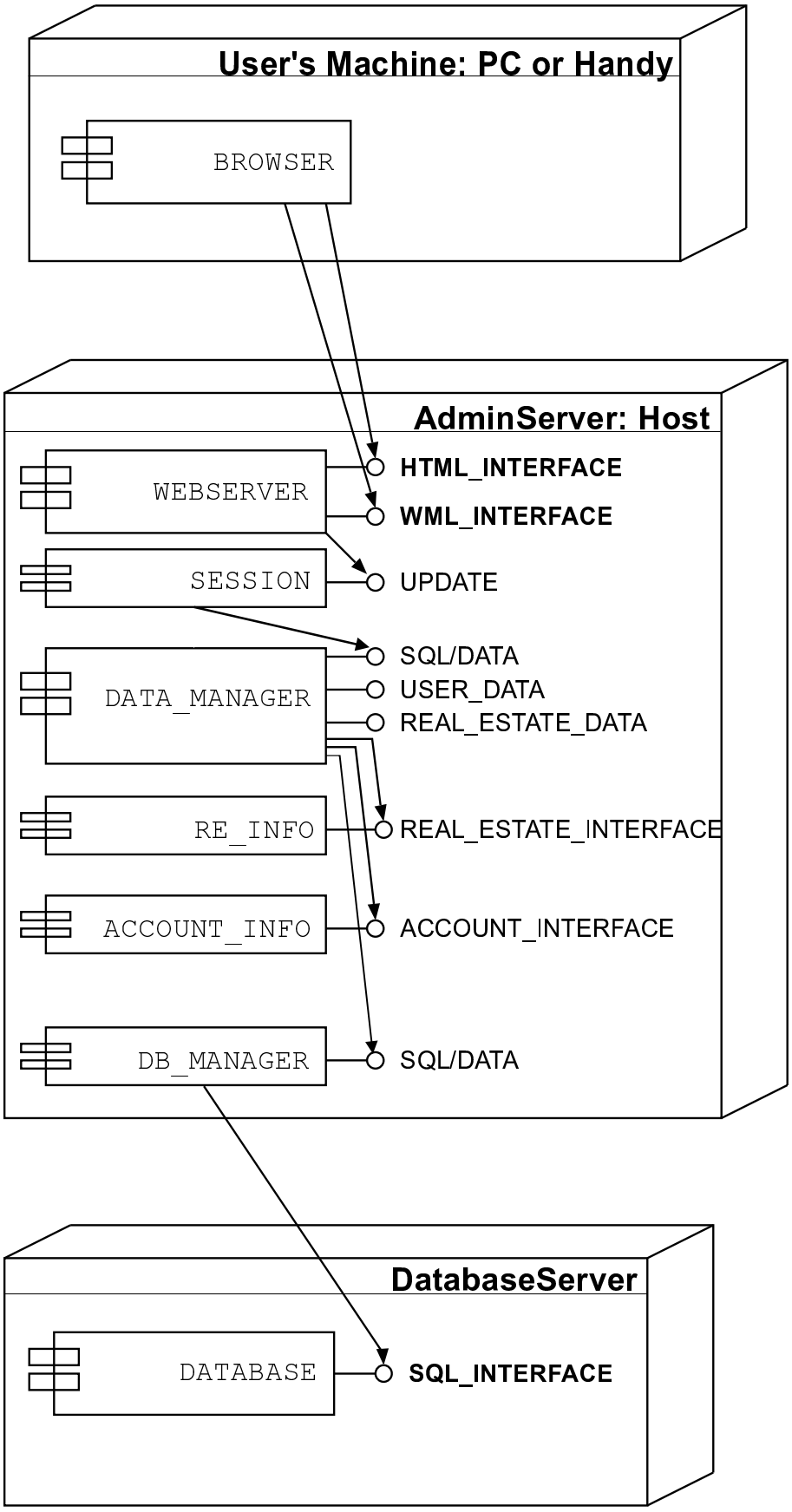
It shows the configuration of software-components, objects and processes at runtime. A deployment diagram is an important illustration of the topology of the system (hardware and software combined). By inspecting a single node (i.e. hardware unit) of the diagram, one can see, what components of the software system are running on it, and of which logical units (classes, objects,...) these component consist.

12.2 Modeling a Deployment Diagram

This diagram is a graph of nodes, connected by associations showing the communication between them. Nodes may contain representations of components, meaning, that the corresponding software-components are running on this specific hardware-unit. The components are interlinked with others by dependencies (dashed arrows in the diagram) and possibly via interfaces. Stereotypes can be used to describe the kind of dependance (e.g. ≪supports≫, ≪uses≫,...).

12.3 Our Deployment Diagrams

As you can see in the following diagram our system is based on a **layered architecture**:


User's Machine: PC or Handy
BROWSER
AdminServer: Host
WEBSERVER
HTML_INTERFACE
WML_INTERFACE
SESSION
UPDATE
DATA_MANAGER
SQL/DATA
USER_DATA
REAL_ESTATE_DATA
RE_INFO
REAL_ESTATE_INTERFACE
ACCOUNT_INFO
ACCOUNT_INTERFACE
DB_MANAGER
SQL/DATA
DatabaseServer
DATABASE
SQL_INTERFACE

Appendix A

Protocolls

A.1 1^{st} Lesson - October, 25^{th}, 2001

Basical information about the course and division into groups.

A.2 2^{nd} Lesson - October, 31^{st}, 2001

First meeting with the client and CRC-session.

1. Groupleader: Christian Koidl
2. Participating Groupmembers: Andrea Geierspichler, Gunnar Ruhs, Roman Gross, Michael Pober
3. Topics talked about: Analysis of the problem and basic class-definition
4. Results:
   (a) The client wants the possibility to obtain relevant information on offers via the internet or the mobile phone.
   (b) The information should be sent to the customer either automatically by the system (push) or on individual request (pull).
   (c) Clients as banks, authorities and private persons get only one account.
   (d) The system should provide an access-control-system.
   (e) Private persons receive their username/password by e-mail.
   (f) Private persons have to pay provision for publishing their information
   (g) There should be the possibility to include different multimedia-elements (pictures, movies, sounds,...).
5. List of possible classes: Session, User Interface, Account, Permission, Information Storage, Real Estate, Editor, Private Person, Prospective Buyer
6. New assignments: Refining the problem-statement. Discuss and act out possible use-cases. Think about the user-interface.

7. Open questions:

   (a) Should permanent accounts be available for privat persons?

   (b) Do private persons have to pay a fee although the object is not sold?

   (c) Is it necessary to have an account to request information?

A.3 3^{rd} Lesson - November, 7^{th}, 2001

1. Groupleader: Andrea Geierspichler

2. Participating Groupmembers: Christian Koidl, Gunnar Ruhs, Roman Gross, Michael Pober

3. Topics talked about: Essential Use-Case-Definition, Essential User-Interface-Modeling

4. Use-Cases discussed:

   (a) Offer-Placing
   Actions: Login - Offer-Placing - Logoff
   Actors: Editor
   Details: The editor contacts the user-interface. The interface forwards the login-inputs to the accout, that checks in the information-storage, if the account exists. When positiv it sends the data to the privilege. The Privilege gets the user-permissions from the information-storage and returns it to the account. Then the account passes the results to the user-interface. If the user is accepted, then the interface starts a session and displays the next screen. Now the user chooses the offer-placing-menu. Here he enters his real-estate data. The interface reads the information and forwards it to the session, that stores it in the information-storage. The storage sends a success message to the user-interface over the session. The interface displays the success message and returns to the menu-screen. The user hits the logout-button and the interface closes the session and shows the login-screen again.

   (b) Create Account
   Actions: Viewing free offers - Create Account - Send Account-Information
   Actor: User
   Details: The User visits the Homepage. The homepage starts a session, that querries the information-storage for the free offers. The storage returns this informations via the session to the homepage and the session is closed. Now the user wants to see more details, so he enters the "create account" menu. Here a session is started and the user is offered a form, where he enters his personal information. The session gets this data and forwards it to the information-storage. Then it activates send-account-information. From here the user gets informed about his username and password. Finally the session querries for the free offers, returns them to the homepage and closes. The free-offers-screen is displayed again.

5. Essential user interface:
   Homepage, subscribe screen, subscription successful, subscription error, login screen, access denied, form for placing offers, infoscreen successful, infoscreen error,

6. List of possible classes: Session, User Interface, Account, Privilege, Information Storage, Real Estate, Editor, Private Person, User, Send Account Information

7. New assignments: Present the essential use-cases and the essential user-interface using slides.

8. Open questions:

   (a) How is a user able to manage his account?

   (b) Is "Privileges" necessary?

A.4 4^{th} Lesson - November, 14^{th}, 2001

1. Groupleader: Gunnar Ruhs

2. Participating Groupmembers: Christian Koidl, Andrea Geierspichler, Roman Gross, Michael Pober

3. Topics talked about: Presentation of the Essential Use-Cases and a basic design for the User-Interface. (For details, please refer to chapter 4)

4. Results:

   (a) The different actors are to be disigned hierachically. Going out from the Guest-Actor, the other persons can be constructed by inheritance.

   (b) We have to add a new class "System" to our list. This represents a foreign system, that tries to interact with our software.

5. List of possible classes: Session, User Interface, Account, Information Storage, Admin, Editor, EditorAndUser, User, Guest, Send Account Info, Search for Info, Push Real-Estate-Info, System

6. New assignments: Present the class-strucure of the project with slides.

7. Open questions: so far: none

A.5 5^{th} Lesson - November, 21^{th}, 2001

1. Groupleader: Roman Gross

2. Participating Groupmembers: Christian Koidl, Andrea Geierspichler, Gunnar Ruhs, Michael Pober

3. Topics talked about: Presentation of the Classdiagrams

4. Results:

(a) Cardinalities have to be inserted into the class-diagrams

(b) The attributes "Privileges" and "Address" are to modeled as own classes

(c) The layout for the user-interface can be put into a class, so that all the different sites can be objects of "layout"

5. New assignments: Build a user-interface prototype for mobile phone using WML (Wireless Markup Language) and for the internet using HTML.

6. Open questions: so far: none

A.6 6^{th} Lesson - December, 5^{th}, 2001

1. Groupleader: Michael Pober

2. Participating Groupmembers: Christian Koidl, Andrea Geierspichler, Gunnar Ruhs, Roman Gross

3. Topics talked about: Presentation of the User-Interface-Prototype (WML and HTML)

4. Results:

(a) The possibility to check the user's resolution and then show an optimized user interface for his special requirements should be given.

(b) An extendable permission-system should be considered.

(c) A user should be able to view his own placed real estates, even he just has write-permissions.

(d) If a visitor wants to search for real estates via WAP, his entered search criterias should be listed in each card.

5. New assignments: Create sequence diagram and activity diagram

6. Open questions: so far: none

A.7 7^{th} Lesson - December, 12^{th}, 2001

1. Groupleader: Christian Koidl

2. Participating Groupmembers: Michael Pober, Andrea Geierspichler, Gunnar Ruhs, Roman Gross

3. Topics talked about: Presentation of the activity diagrams and sequence diagrams

4. Results:

(a) The flow diagram has to be created from the users point of view only.

(b) Descriptions of the events have to be added to the left side of the sequence diagrams.

5. Open questions: so far: none

A.8 8^{th} Lesson - January, 9^{th}, 2002

1. This meeting did not take place so we will talk about the topics planned for that meeting in the next week.

A.9 9^{th} Lesson - January, 16^{th}, 2002

1. Groupleader: Andrea Geierspichler
2. Participating Groupmembers: Michael Pober, Roman Gross, Christian Koidl, Gunnar Ruhs
3. Topcs talked about: Presentation of out Component Diagrams and Deployment Diagrams
4. Results:
   (a) One thing that had to be changed in the Deployment Diagram: The direct connection between the UI_Controller and the Real_Estate_Info/Account_Info as well as the connection between Session and Real_Estate_Info/Account_Info is not necessary.
   (b) For security reasons the attributes in the ADDRESS-Class have to be private and the corresponding methods have to be inserted.
   (c) Our Search-Methods were working with search strings not unlike those used in CGI but we should use hash tables instead.
   (d) We have to extend our CONNECTION-Class. It should theoretically be possible to enter an indefinite number of contact information. Therefore we have to create a "list" for the phone-numbers, etc.
   (e) We should to split up the PRIVILEGE-Class into a list of classes, one for every single privilege. Thereby the system is extendable more easily.