# 1. Short Description of Project

1. Title: **Railway Information System**
2. Name of team members: Carmen Casaru, Moisei Andrei, Tinta Mihaita
3. Text with description-outline of the characteristics of the software

The board for the National Railway Company decides to implement a new information system to offer access to the timetable of the passenger trains (CFR).

The users can interact with the system through special terminals which are available in the stations.

The user selects his departure station and his arrival station plus a number of options:

* the user may want the train to pass through a certain station along the way
* the user may choose the type of train (Personal, Accelerat, Rapid, Intercity)
* the user may choose an interval for the departure hour and an interval for the arrival
* hour
* the user may input a maximum price for the purchase
* the user may choose first or second class
* the user can manifest his desire for a seat at the window or forward facing

Based on these preferences, a list of possible trains is displayed. If no match is found, a composed entry may be shown (for example we do not have a train from Brasov to Constanta, instead we take a train from Brasov to Bucuresti and then from Bucuresti to Constanta).

The price is computed based on the type of the train, the class and the number of kilometers of the voyage.

There can also be some special trains, which travel only during some months of the year, which have special prices.

Once the user has made a choice, he can pay directly at the terminal using his credit card.

The system can also be accessed by the cashiers in which case the customer presents his choices to the cashier who uses the system. In this case the customer can benefit from discounts (students, soldiers, retirees) by presenting the documents to the cashier. He can also pay by cash.

In both cases a ticket is printed and the purchase is registered in the system. The ticket contains the train, seat number, and, for discounted tickets, the CNP of the customer.

The system also allows an administrator to enter train routes details or modify the available ones.

1. Brief on main functional and non-functional requirements

**Functional requirements:**

1. System shows list of trains

a). show availability

b). show after the type (Personal, Accelerat, Rapid, Intercity)

c). show the class of the train (first or second)

d). show position of the seat (window or forward facing)

e). show after the price

2. System should process new registrations of customers

a). auto-complete fields

b). register an associated credit card of the customer

3. System should process purchase of tickets

a). check availability

b). process a composed entry

c). process the route to pass through a certain station given by the customer

d). apply discount

e). compute total cost

f). update the registry

g). store transactions with credit cards

4. System should process new updates of routes

a). add delays to routes

b). add route

c). modify route details

d). delete route

e). modify train details

f). change price/discount

**Non-functional requirements:**

1. Security – The system stores the train routes details on a secure server. Direct access is given only to terminals and cashiers. The system checks the IDCard of the customer to be valid.

2. Reliability – 5 days/week, 12 hours/day

3. Usability – the system can be easily used by high-school graduates, after two hours of briefing

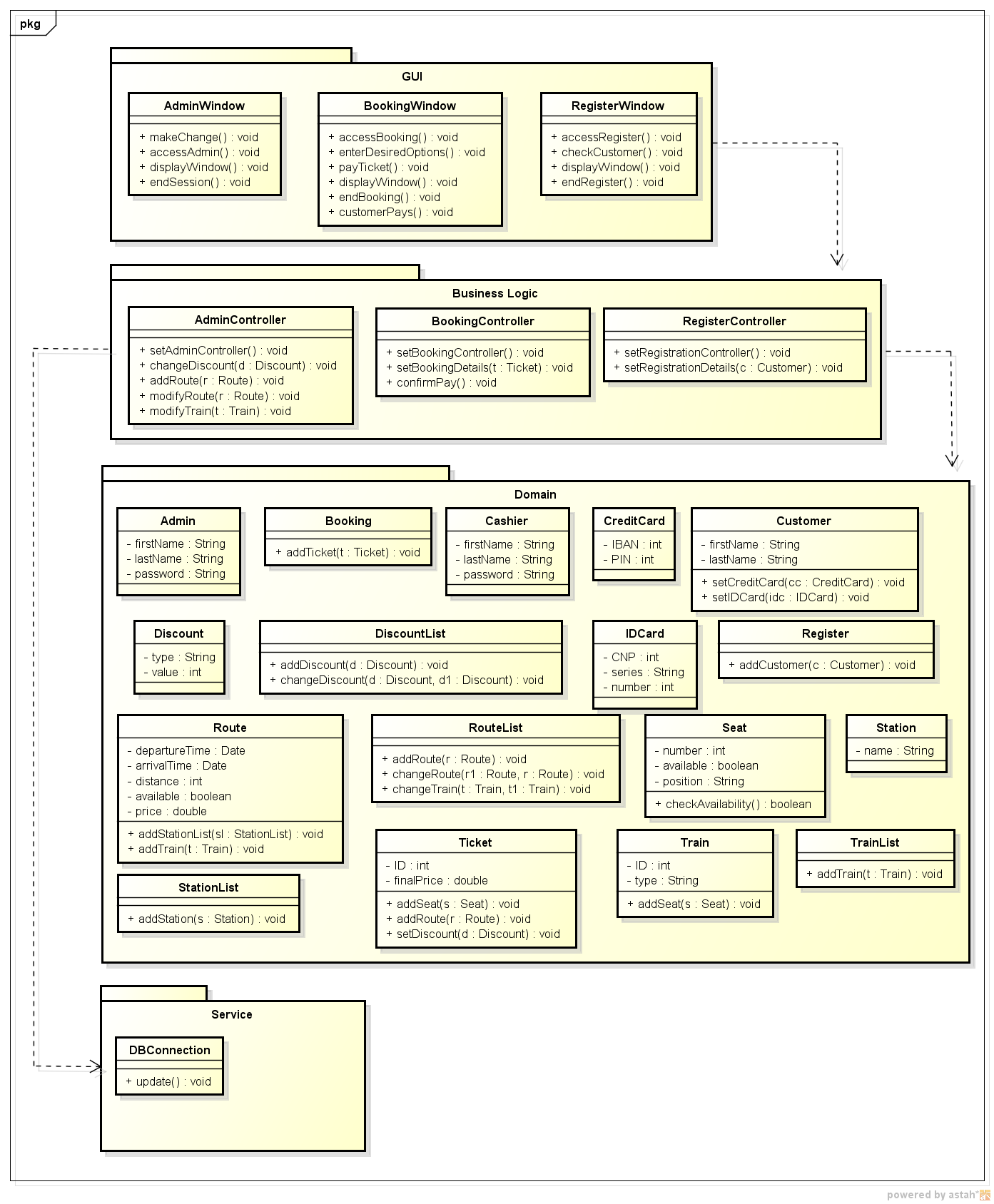
4. Capacity – the system can store 1.000 routes, 500 trains and 300 users

1. Short description of the main components using some UML notations.

We are planning on using 3 components:

* a Graphical User Interface component
* a Domain component and
* a Database Management component

Below is a diagram of the application layers. A first layer contains all the graphical elements and associated to it is a business logic layer which is in charge of controlling the graphical elements and synchronizing them with the domain information. It is also in charge with the retrieval and data storage triggered by the action performed on the graphical elements.

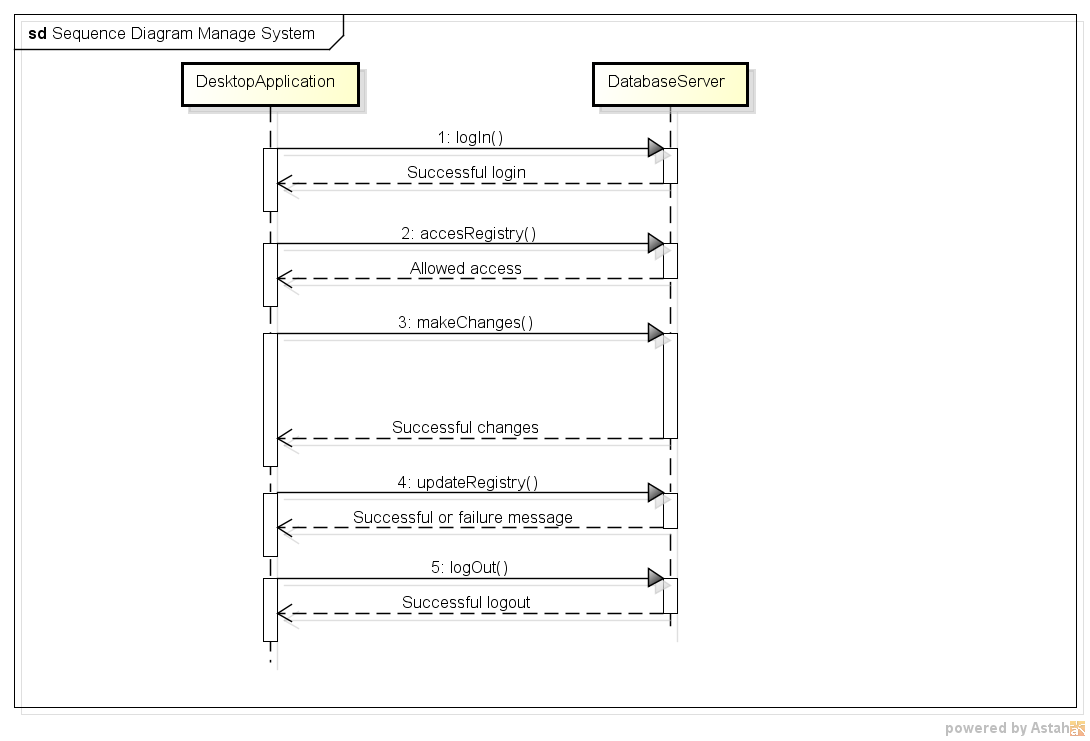


# 2. Formal verification of component(s) design

**1). Buy a ticket from terminal**

**2). Buy a ticket from cashier**

**3). Manage the system**



**Promela code:**

#define match = 1

/\* Definition of types \*/

mtype {

login

successfulLogin,

accesRegistry,

allowedAccess,

makeChanges,

successfulChanges,

updateRegistry,

successfulMessage,

logout,

successfulLogout

}

/\* -- Global definitions

--------------------------------------------------------------- \*/

#define iSize 100 /\*Size of the internal event channel buffers(>=1) \*/

/\* --- External event channels --- \*/

chan DADS = [iSize] of {mtype} /\* Channel for sending-receiving messages \*/

proctype DesktopApplication(){

Start:atomic{ /\*All operations are executed as atomic transition \*/

printf("\n DesktopApplication: Start\n");

DADS!login; /\* sending of messages \*/

DADS?successfulLogin; /\* reception of messages \*/

goto Next1;

}

Next1:atomic{ /\*All operations are executed as atomic transition \*/

printf("\n DesktopApplication: Next1\n");

DADS!accesRegistry; /\* sending of messages \*/

DADS?allowedAccess; /\* reception of messages \*/

goto Next2;

}

Next2:atomic{ /\*All operations are executed as atomic transition \*/

printf("\n DesktopApplication: Next2\n");

DADS!makeChanges; /\* sending of messages \*/

DADS?successfulChanges; /\* reception of messages \*/

goto Next3; /\* if we want a cycle \*/

}

Next3:atomic{ /\*All operations are executed as atomic transition \*/

printf("\n DesktopApplication: Next3\n");

DADS!updateRegistry; /\* sending of messages \*/

DADS?successfulMessage; /\* reception of messages \*/

goto Next4; /\* if we want a cycle \*/

}

Next4:atomic{ /\*All operations are executed as atomic transition \*/

printf("\n DesktopApplication: Next4\n");

DADS!logout; /\* sending of messages \*/

DADS?successfulLogout; /\* reception of messages \*/

goto Start; /\* if we want a cycle \*/

}

}

proctype DatabaseServer(){

Start:atomic{ /\*All operations are executed as atomic transition \*/

printf("\n DatabaseServer: Start\n");

DADS?login; /\* reception of messages \*/

DADS!successfulLogin; /\* sending of messages \*/

goto Next1;

}

Next1:atomic{ /\*All operations are executed as atomic transition \*/

printf("\n DatabaseServer: Next1\n");

DADS?accesRegistry; /\* reception of messages \*/

DADS!allowedAccess; /\* sending of messages \*/

goto Next2;

}

Next2:atomic{ /\*All operations are executed as atomic transition \*/

printf("\n DatabaseServer: Next2\n");

DADS?makeChanges; /\* reception of messages \*/

DADS!successfulChanges; /\* sending of messages \*/

goto Next3;

}

Next3:atomic{ /\*All operations are executed as atomic transition \*/

printf("\n DatabaseServer: Next3\n");

DADS?updateRegistry; /\* reception of messages \*/

DADS!successfulMessage; /\* sending of messages \*/

goto Next4;

}

Next4:atomic{ /\*All operations are executed as atomic transition \*/

printf("\n DatabaseServer: Next4\n");

DADS?logout; /\* reception of messages \*/

DADS!successfulLogout; /\* sending of messages \*/

goto Start;

}

}

/\* --- Initialization --------------------------------\*/

init{

run DesktopApplication();

run DatabaseServer();

}