# 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

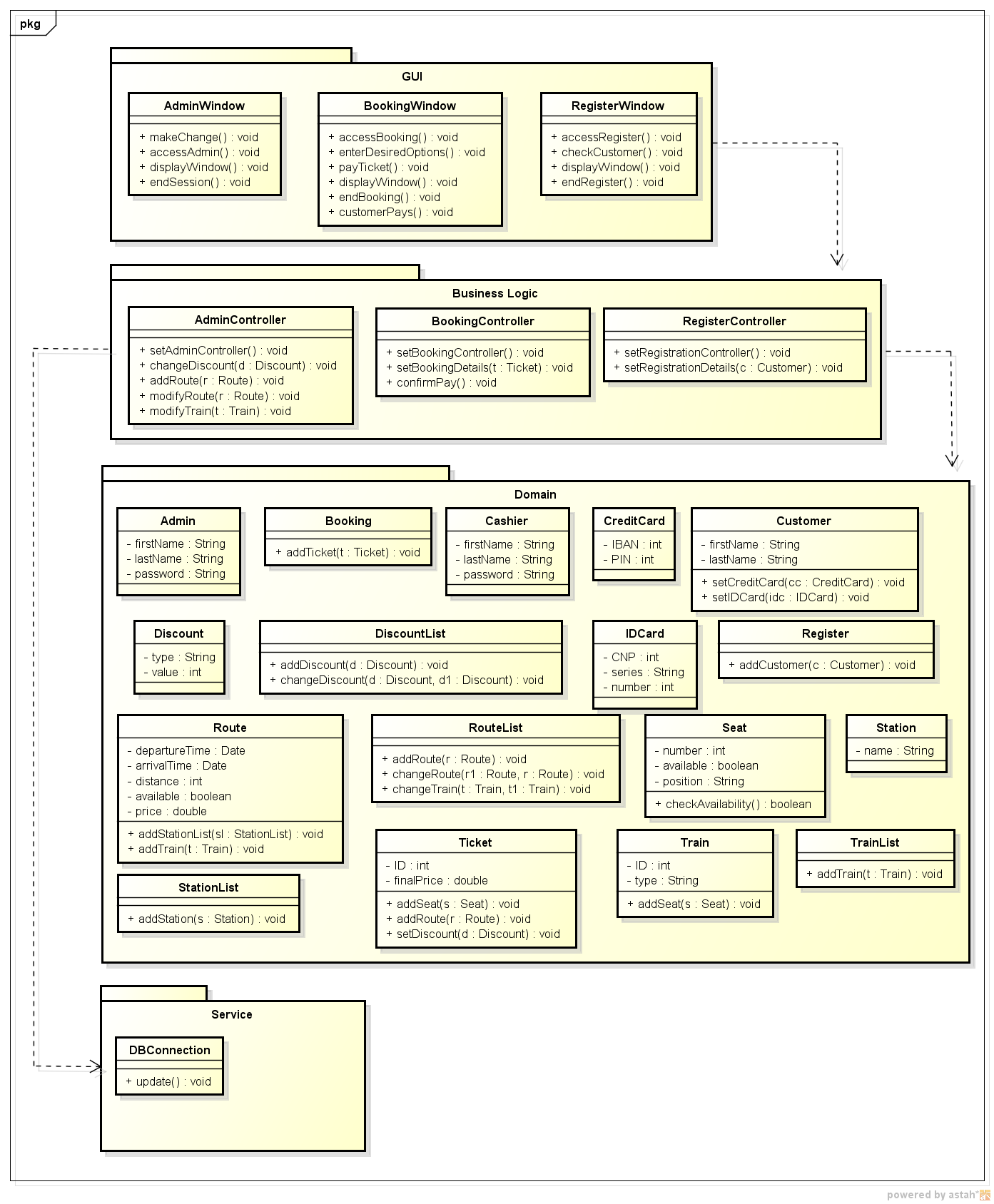
4. Capacity – the system can store 1.000 routes, 500 trains and 300 users

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.

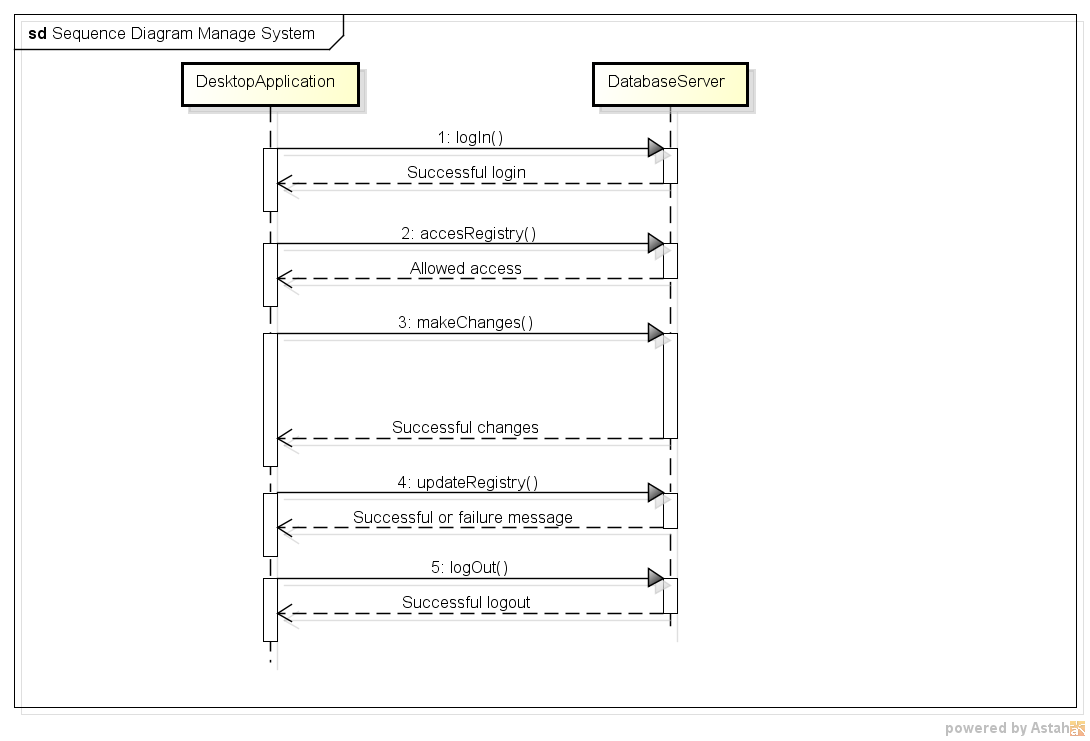


# 2. Formal verification of component(s) design

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

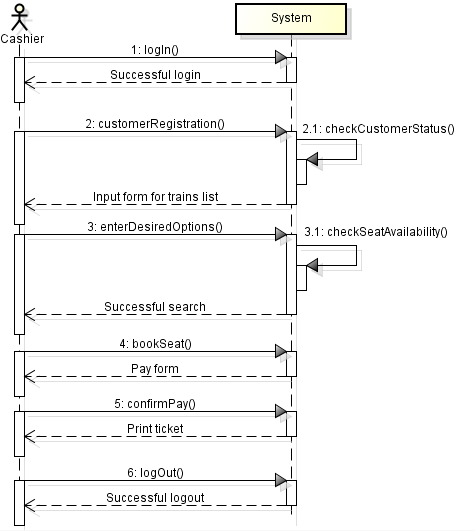
init{

run DesktopApplication();

run DatabaseServer();

}

2. Cashier



/\*\*

\* This program is designed in order to illustrate and verify the communication between

\* the database server and the rest application for a "Sequence Diagram Cashier Ticket use case scenarion

\* that the application offers

\*

\* Author: Mihaita Tinta

\*

\*

\* flags : -s -r -n, -t -p

\*/

/\* Definition of messages on which the communication is based \*/

mtype{

/\* These messages form the handshake packets needed to establish the connection

between a terminal and the database server \*/

Connection\_Request,

Connection\_Established\_Reply,

/\* This state is required by the database server letting the terminal know that it is ready

to receive and process requests \*/

Waiting\_For\_Request,

/\* Login \*/

Login\_Request,

Login\_Response,

/\* Customer registration\*/

Customer\_Registration\_Request,

Customer\_Registration\_Response,

/\* Search train route \*/

Search\_Train\_Request,

Search\_Train\_Response,

/\* Requesting a ticket \*/

Terminal\_Ticket\_Request,

Terminal\_Ticket\_Results\_Reply,

/\* Pay desired ticket \*/

Pay\_Ticket\_Request,

Pay\_Ticket\_Reply,

/\* Pay chash \*/

Pay\_Cash\_Request,

Pay\_Cash\_Reply,

/\* Pay with credit card \*/

Pay\_With\_Credit\_Card\_Request,

Pay\_With\_Credit\_Card\_Reply\_Reply,

/\* Logout \*/

Logout\_Request,

Logout\_Response,

/\* These messages comprise the packets needed to end a connection between a terminal and

the database server \*/

Connection\_End\_Request,

Connection\_End\_Established\_Reply,

};

/\* The number of terminals that are available and can be connected to the database server \*/

#define terminalNumber 3

/\* A global synchronous channel used for communication between the terminal and the database

server \*/

chan ssocket = [0] of { mtype, chan, chan };

/\* The database server \*/

active proctype DataBaseServer()

{

/\* Used to hold a uniqe ID for each terminal with which the database server establishes a connection \*/

int terminalID = 0;

/\* Input and output channels. Can store max 1 message per channel \*/

chan ChannelIn[terminalNumber] = [1] of { mtype};

chan ChannelOut[terminalNumber] = [1] of { mtype};

do

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Establish Connection \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Waiting for a connection request from a terminal \*/

:: ssocket ? Connection\_Request(ChannelIn[terminalID], ChannelOut[terminalID])->

/\* Establish connection. Send reply to terminal \*/

ssocket ! Connection\_Established\_Reply(ChannelIn[terminalID], ChannelOut[terminalID]) ->

/\* Inform terminal that the database server is ready to receive and process requests \*/

ssocket ! Waiting\_For\_Request(ChannelIn[terminalID], ChannelOut[terminalID]) ->

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Login User \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Terminal login \*/

ssocket ? Login\_Request(ChannelIn[terminalID], ChannelOut[terminalID]) ->

/\* successfully login \*/

ssocket ! Login\_Response(ChannelIn[terminalID], ChannelOut[terminalID])->

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Customer Registration Request \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Terminal login \*/

ssocket ? Customer\_Registration\_Request(ChannelIn[terminalID], ChannelOut[terminalID]) ->

/\* successfully login \*/

ssocket ! Customer\_Registration\_Response(ChannelIn[terminalID], ChannelOut[terminalID])->

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Search Train Request \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Terminal login \*/

ssocket ? Search\_Train\_Request(ChannelIn[terminalID], ChannelOut[terminalID]) ->

/\* successfully login \*/

ssocket ! Search\_Train\_Response(ChannelIn[terminalID], ChannelOut[terminalID])->

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Buy Ticket \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Terminal request for all available train tickets \*/

ssocket ? Terminal\_Ticket\_Request(ChannelIn[terminalID], ChannelOut[terminalID]) ->

/\* Send the list of all available tickets to the terminal \*/

ssocket ! Terminal\_Ticket\_Results\_Reply(ChannelIn[terminalID], ChannelOut[terminalID])->

/\* The client wants to pay for the desired ticket \*/

ssocket ? Pay\_Ticket\_Request(ChannelIn[terminalID], ChannelOut[terminalID])->

/\* Inform the terminal the the payment protocol has been initiated and that the payment

can commence \*/

ssocket ! Pay\_Cash\_Reply(ChannelIn[terminalID], ChannelOut[terminalID])->

if

/\* Client has chosen to pay cash \*/

::ssocket ? Pay\_Cash\_Request(ChannelIn[terminalID], ChannelOut[terminalID]) ->

/\*Confirm cash payment \*/

ssocket ! Pay\_Cash\_Reply(ChannelIn[terminalID], ChannelOut[terminalID]);

/\* Client has chose to pay using a credit card \*/

::ssocket ? Pay\_With\_Credit\_Card\_Request(ChannelIn[terminalID], ChannelOut[terminalID]) ->

/\*Confirm credit card payment \*/

ssocket ! Pay\_With\_Credit\_Card\_Reply\_Reply(ChannelIn[terminalID], ChannelOut[terminalID]);

fi;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* End Connection \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* The terminal has initiated the disconnect sequence \*/

ssocket ? Connection\_End\_Request(ChannelIn[terminalID], ChannelOut[terminalID]) ->

/\* Confirm connection termination \*/

ssocket ! Connection\_End\_Established\_Reply(ChannelIn[terminalID], ChannelOut[terminalID]) ->

/\* Increase Id corresponding to next terminal connection \*/

terminalID++;

/\* Upon reaching the maximum number of terminal connections, stop \*/

::terminalID == terminalNumber ->

break;

od;

}

/\* The terminal \*/

active [terminalNumber] proctype Terminal()

{

/\* Input and output channels. Can store max 1 message per channel \*/

chan TerminalChanIn = [1] of { mtype};

chan TerminalChanOut = [1] of { mtype};

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Establish Connection \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Send a connection request to the database server \*/

ssocket ! Connection\_Request( TerminalChanOut, TerminalChanIn) ->

/\* Wait for acknowlegment from dabase server, letting us know that the connection has been established

successfully \*/

ssocket ? Connection\_Established\_Reply(TerminalChanOut, TerminalChanIn) ->

/\* Wait for message from database server, letting us know that it is ready to receive requests

and process them \*/

ssocket ? Waiting\_For\_Request(TerminalChanOut, TerminalChanIn) ->

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Login User \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

ssocket ! Login\_Request(TerminalChanOut, TerminalChanIn) ->

/\* Wait for successfully login \*/

ssocket ? Login\_Response(TerminalChanOut, TerminalChanIn)->

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Customer Registration Request \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

ssocket ! Customer\_Registration\_Request(TerminalChanOut, TerminalChanIn) ->

/\* Wait for successfully login \*/

ssocket ? Customer\_Registration\_Response(TerminalChanOut, TerminalChanIn)->

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Search Train Request \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

ssocket ! Search\_Train\_Request(TerminalChanOut, TerminalChanIn) ->

/\* Wait for successfully login \*/

ssocket ? Search\_Train\_Response(TerminalChanOut, TerminalChanIn)->

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Buy Ticket \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Look up all available tickets \*/

ssocket ! Terminal\_Ticket\_Request(TerminalChanOut, TerminalChanIn) ->

/\* Wait for a list of all available tickets from the database server \*/

ssocket ? Terminal\_Ticket\_Results\_Reply(TerminalChanOut, TerminalChanIn)->

/\* Pay for desired ticket \*/

ssocket ! Pay\_Ticket\_Request(TerminalChanOut, TerminalChanIn)->

/\* Wait for confirmation from the database server that the payments protocol has been initiated

successfully and that the payment operation can commence \*/

ssocket ? Pay\_Cash\_Reply(TerminalChanOut, TerminalChanIn)->

if

/\* Choose to pay with cash \*/

::ssocket ! Pay\_Cash\_Request(TerminalChanOut, TerminalChanIn);

/\* Choose to pay with a credit card \*/

::ssocket ! Pay\_With\_Credit\_Card\_Request(TerminalChanOut, TerminalChanIn);

fi;

if

/\* Wait for cash payment confirmation \*/

::ssocket ? Pay\_Cash\_Reply(TerminalChanOut, TerminalChanIn);

/\* Wait for credit card payment confirmation \*/

::ssocket ? Pay\_With\_Credit\_Card\_Reply\_Reply(TerminalChanOut, TerminalChanIn);

fi;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* End Connection \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Send Connection\_End\_Request \*/

ssocket ! Connection\_End\_Request(TerminalChanOut, TerminalChanIn)->

/\* Wait for acknowledgement form database server \*/

ssocket ? Connection\_End\_Established\_Reply(TerminalChanOut, TerminalChanIn);

}