HOTEL RESERVATION SYSTEM

Phase II

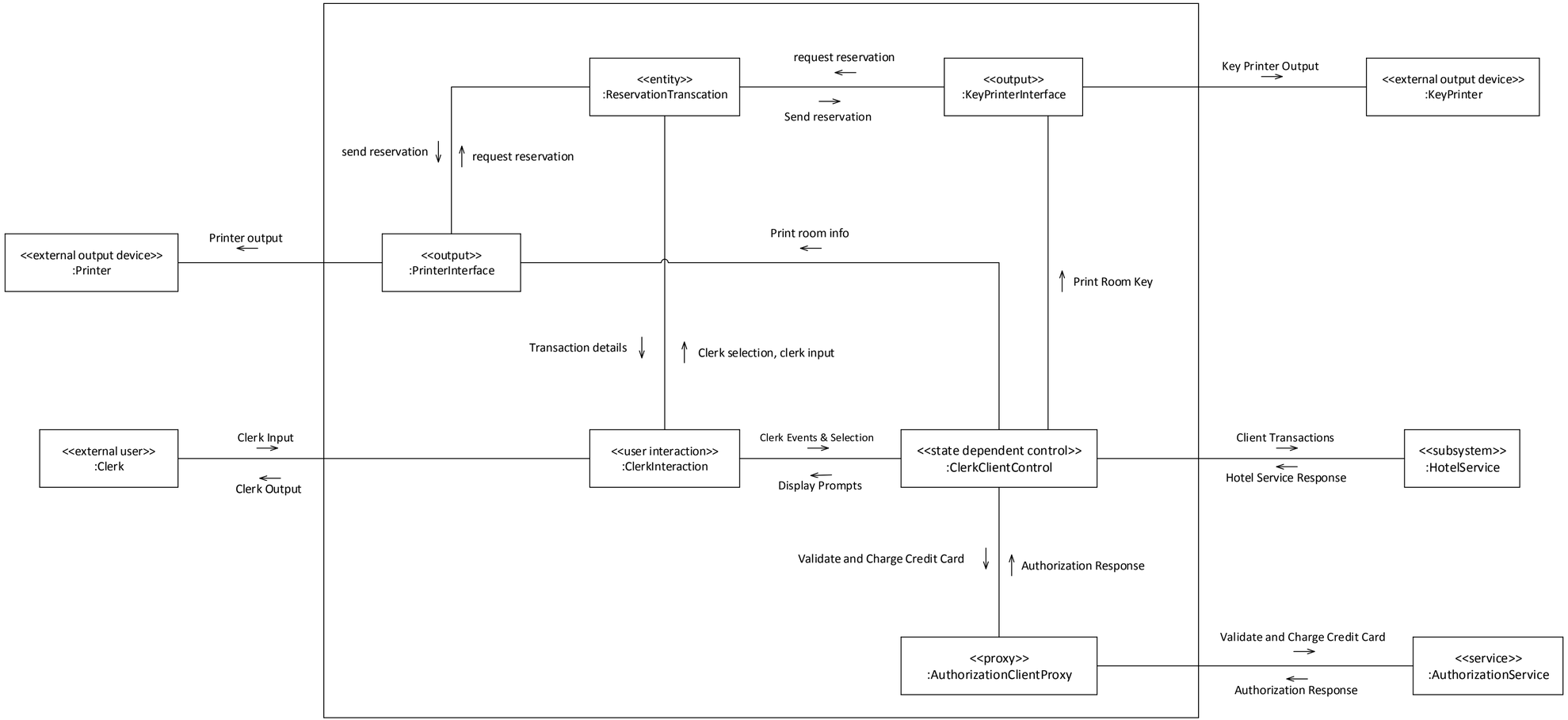
SWE 621 – Spring 2013 Term Project

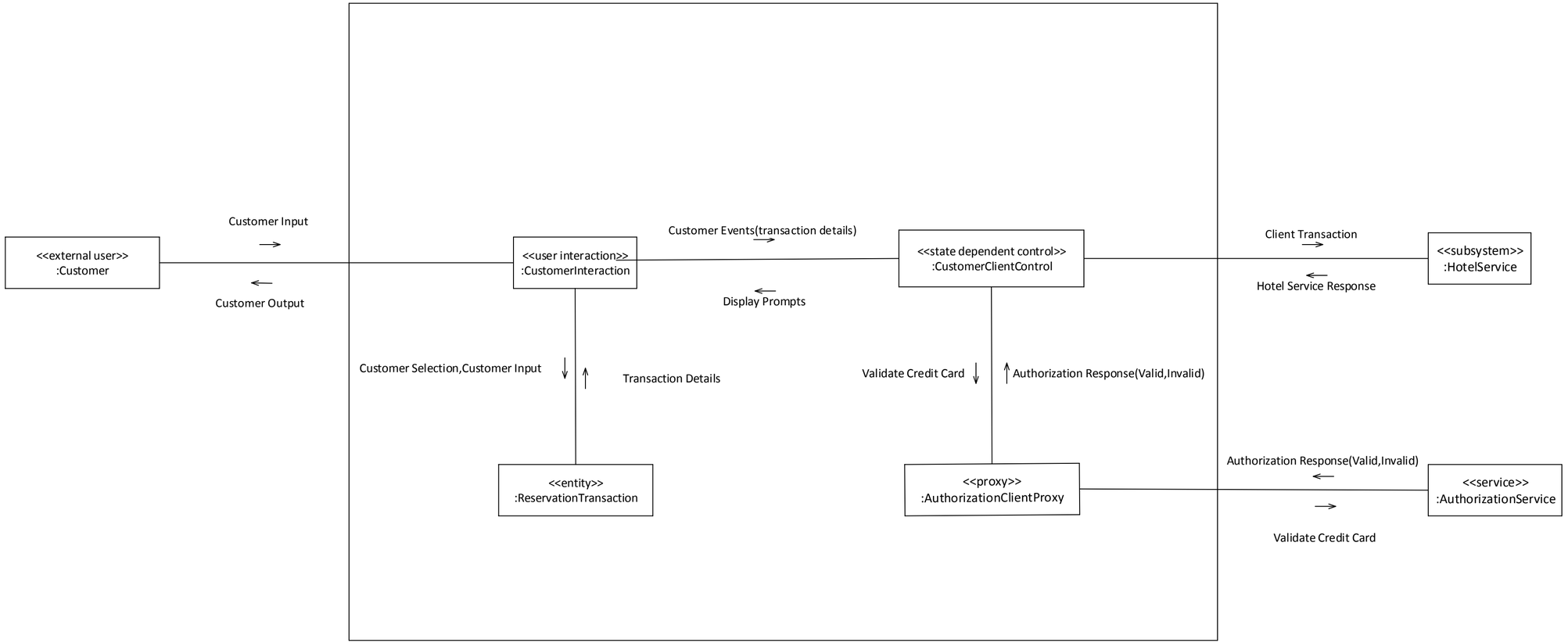
Project Team
Adarsh Ghanta
Matthias Mohammod
Shaeq Khan
Vividh S. Viswanatha

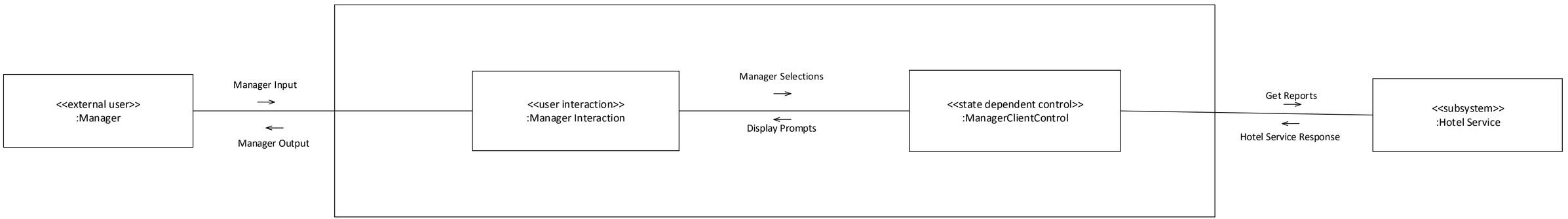
Table of Contents

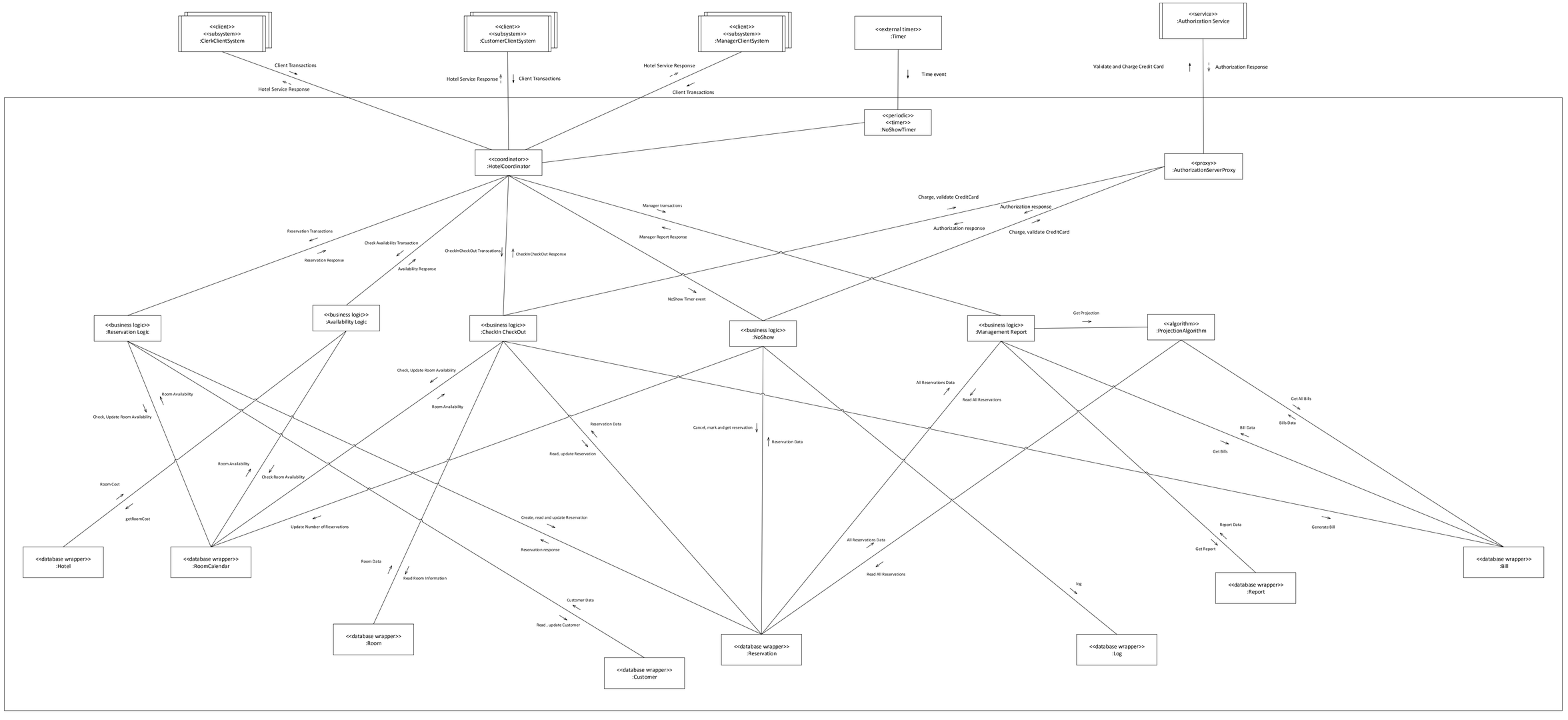
d) *Define the information hiding classes in the system. Define the operations of each class. Define the class interface specifications for the information hiding classes in the system.*

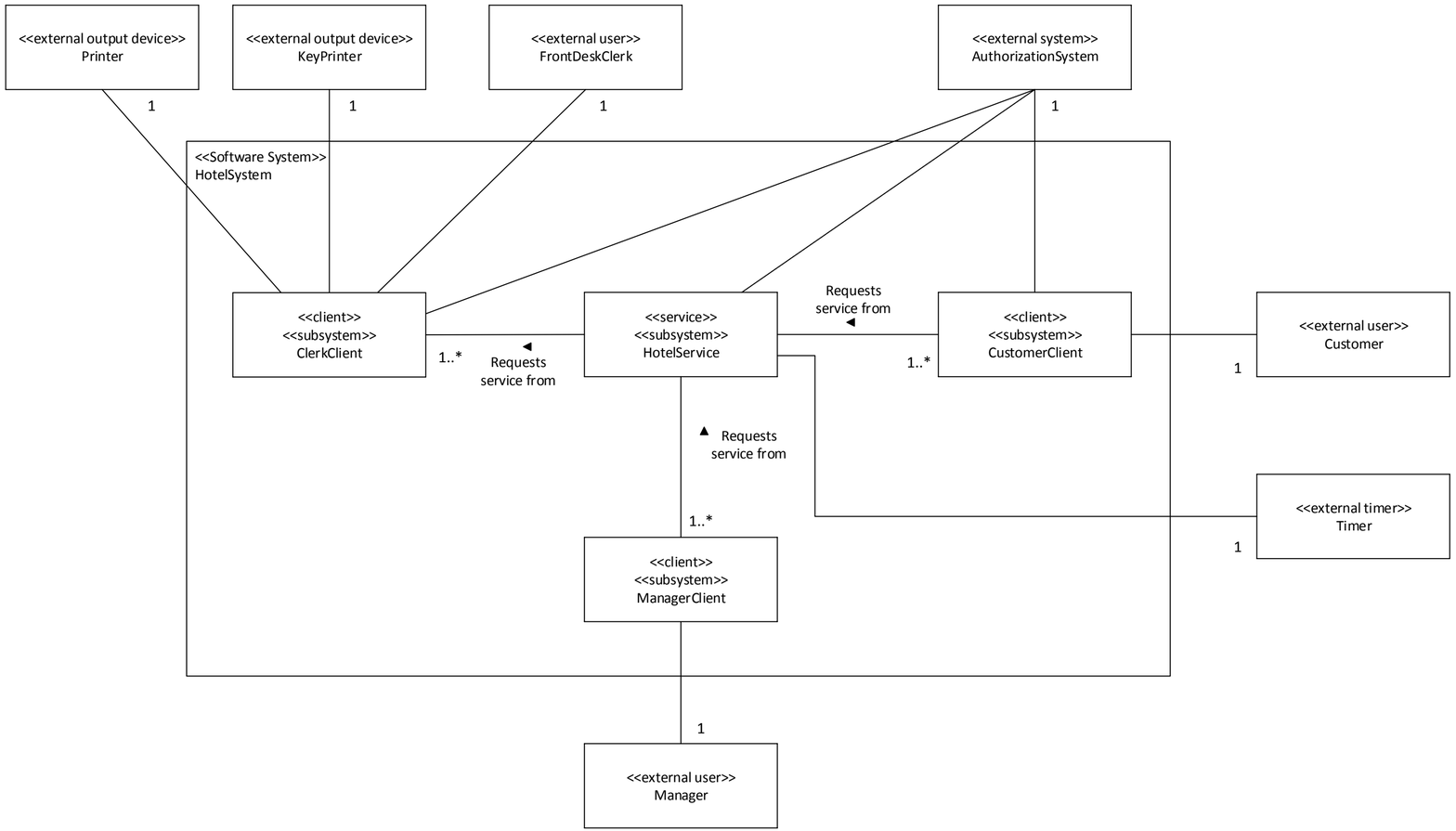
e) *Develop a task interface specification and a task behavior specification for each concurrent task in the system, showing how each task responds to the inputs it receives.*

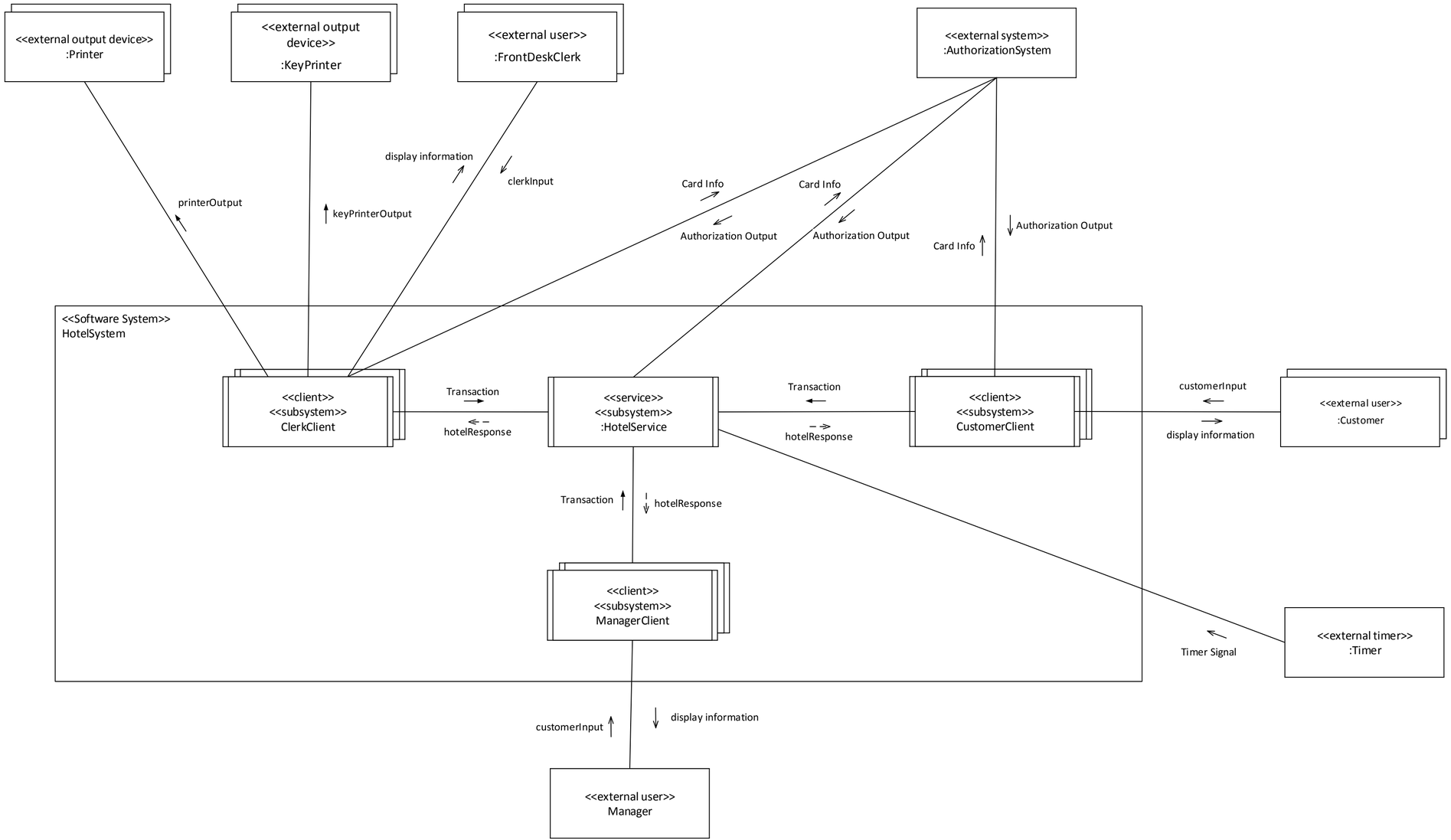

<<entity>>
:ReservationTranscation
request reservation
Send reservation
<<output>>
:KeyPrinterInterface
Key Printer Output
<<external output device>>
:KeyPrinter
send reservation
request reservation
<<external output device>>
:Printer
Printer output
<<output>>
:PrinterInterface
Print room info
Print Room Key
Transaction details
Clerk selection, clerk input
<<external user>>
:Clerk
Clerk Input
Clerk Output
<<user interaction>>
:ClerkInteraction
Clerk Events & Selection
Display Prompts
<<state dependent control>>
:ClerkClientControl
Client Transactions
Hotel Service Response
<<subsystem>>
:HotelService
Validate and Charge Credit Card
Authorization Response
<<proxy>>
:AuthorizationClientProxy
Validate and Charge Credit Card
Authorization Response
<<service>>
:AuthorizationService

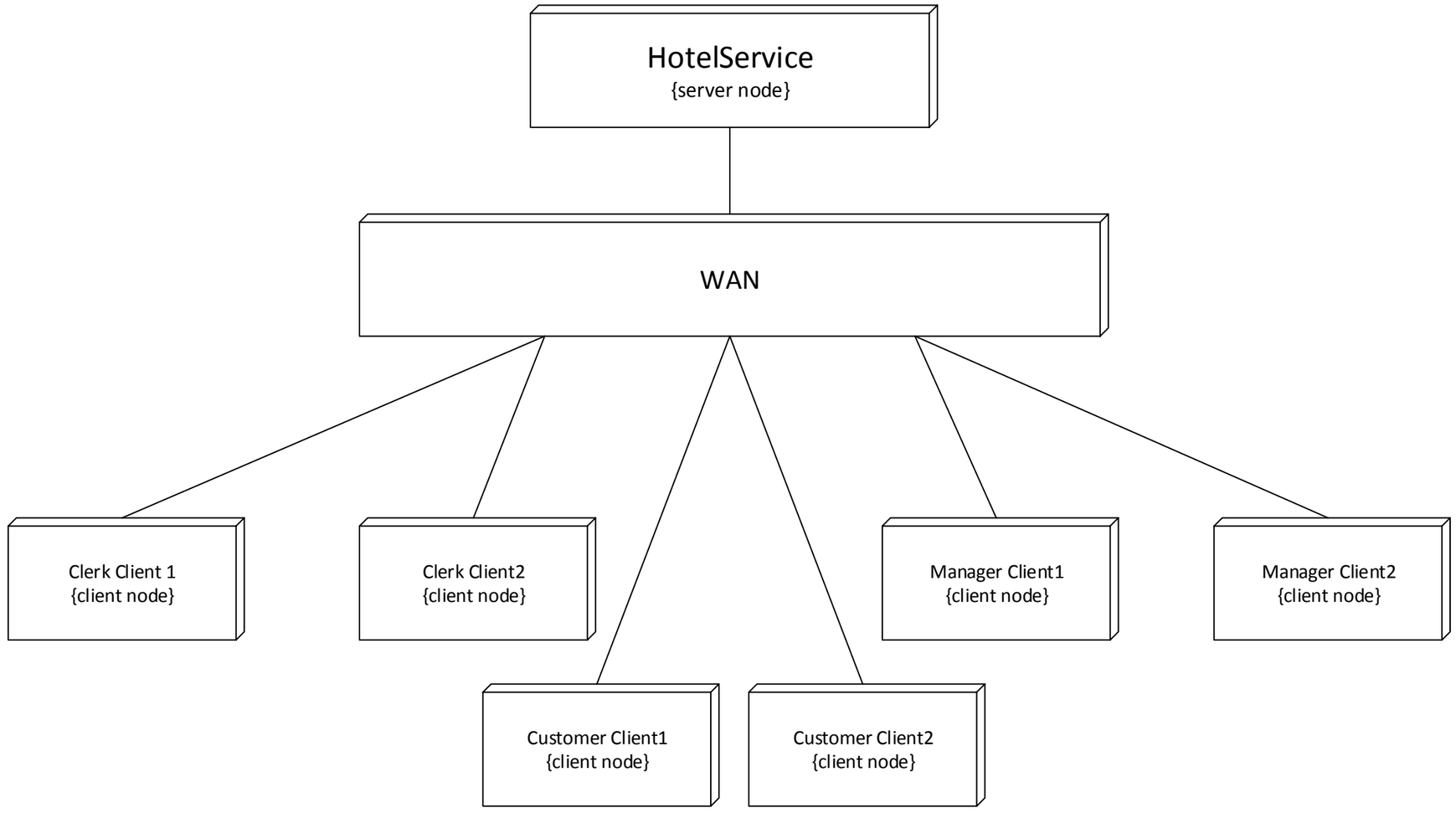

<<external user>>
:Customer
Customer Input
Customer Output
<<user interaction>>
:CustomerInteraction
Customer Events(transaction details)
Display Prompts
<<state dependent control>>
:CustomerClientControl
Client Transaction
Hotel Service Response
<<subsystem>>
:HotelService
Customer Selection,Customer Input
Transaction Details
Validate Credit Card
Authorization Response(Valid,Invalid)
<<entity>>
:ReservationTransaction
<<proxy>>
:AuthorizationClientProxy
Authorization Response(Valid,Invalid)
Validate Credit Card
<<service>>
:AuthorizationService

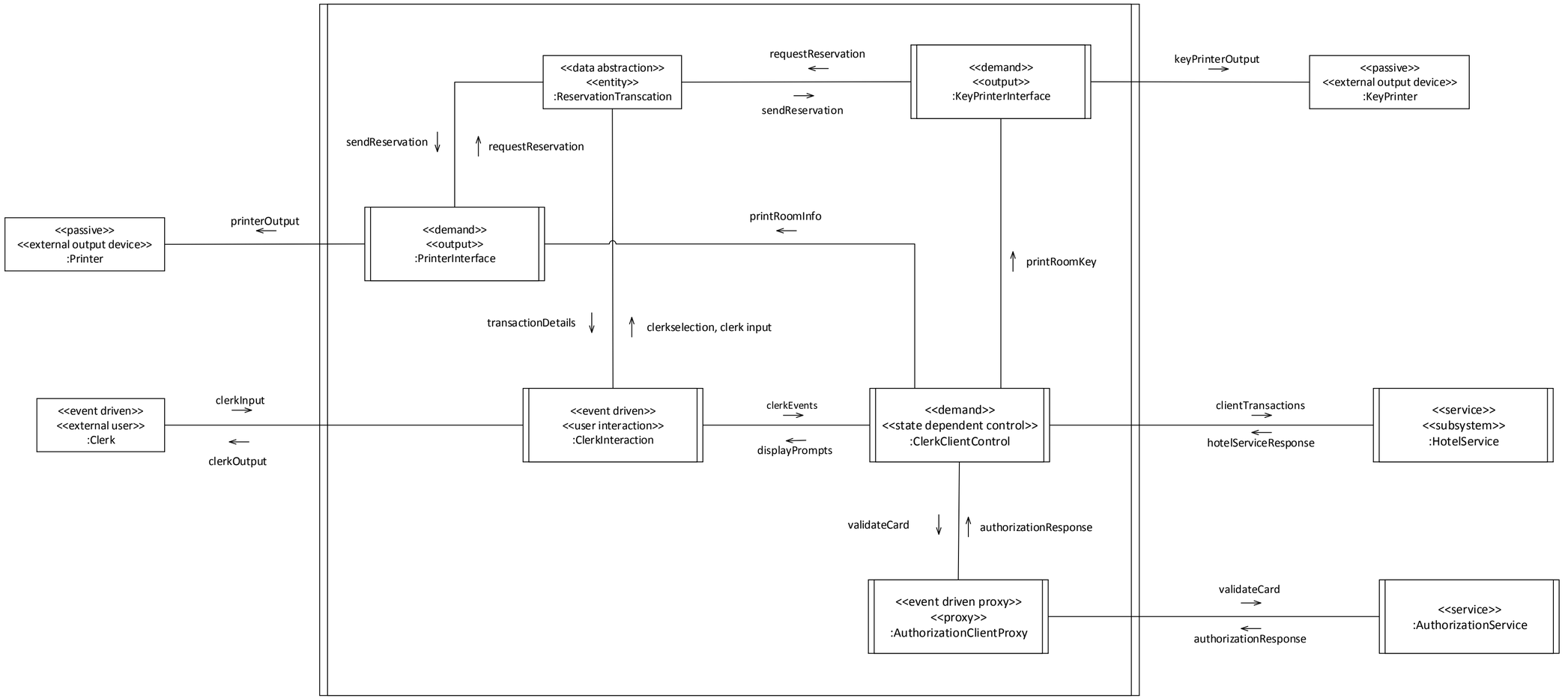

<<external user>>
:Manager
Manager Input
Manager Output
<<user interaction>>
:Manager Interaction
Manager Selections
Display Prompts
<<state dependent control>>
:ManagerClientControl
Get Reports
Hotel Service Response
<<subsystem>>
:Hotel Service

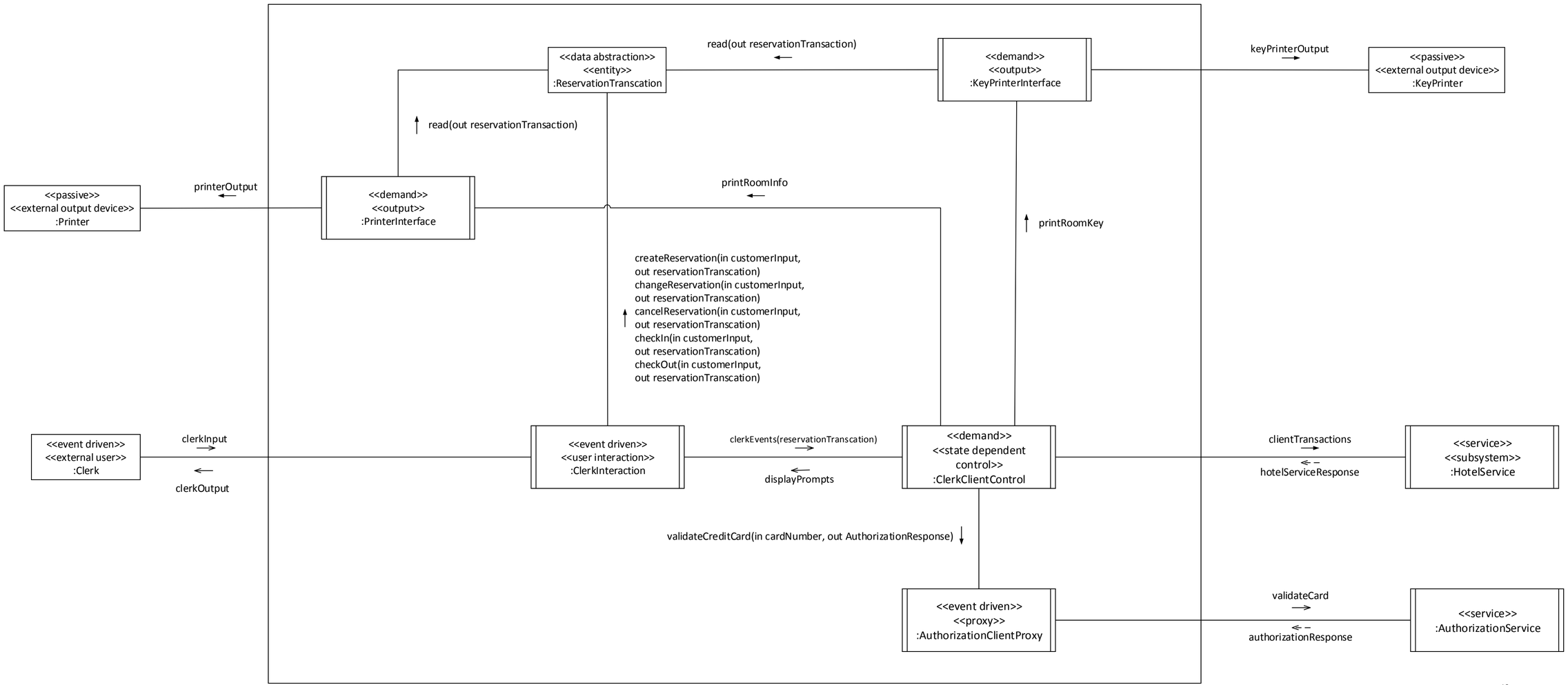

<<client>>
<<subsystem>>
:ClerkClientSystem
<<client>>
<<subsystem>>
:CustomerClientSystem
<<client>>
<<subsystem>>
:ManagerClientSystem
<<external timer>>
:Timer
<<service>>
:Authorization Service
Client Transactions
Hotel Service Response
Hotel Service Response
Client Transactions
Hotel Service Response
Client Transactions
Time event
Validate and Charge Credit Card
Authorization Response
<<periodic>>
<<timer>>
:NoShowTimer
<<coordinator>>
:HotelCoordinator
<<proxy>>
:AuthorizationServerProxy
Manager transactions
Manager Report Response
Charge, validate CreditCard
Authorization response
Authorization response
Charge, validate CreditCard
Reservation Transactions
Reservation Response
Check Availability Transaction
Availability Response
CheckinCheckOut Transcations
CheckinCheckOut Response
NoShow Timer event
<<business logic>>
:Reservation Logic
<<business logic>>
:Availability Logic
<<business logic>>
:CheckIn CheckOut
<<business logic>>
:NoShow
<<business logic>>
:Management Report
Get Projection
<<algorithm>>
:ProjectionAlgorithm
Check, Update Room Availability
Room Availability
Check, Update Room Availability
Room Availability
Reservation Data
Read, update Reservation
Cancel, mark and get reservation
Reservation Data
All Reservations Data
Read All Reservations
Get All Bills
Bills Data
Bill Data
Get Bills
Room Availability
Check Room Availability
Room Cost
getRoomCost
Update Number of Reservations
Create, read and update Reservation
Reservation response
All Reservations Data
Read All Reservations
Report Data
Get Report
Generate Bill
<<database wrapper>>
:Hotel
<<database wrapper>>
:RoomCalendar
<<database wrapper>>
:Bill
Room Data
Read Room Information
log
<<database wrapper>>
:Report
Customer Data
Read , update Customer
<<database wrapper>>
:Room
<<database wrapper>>
:Reservation
<<database wrapper>>
:Log
<<database wrapper>>
:Customer

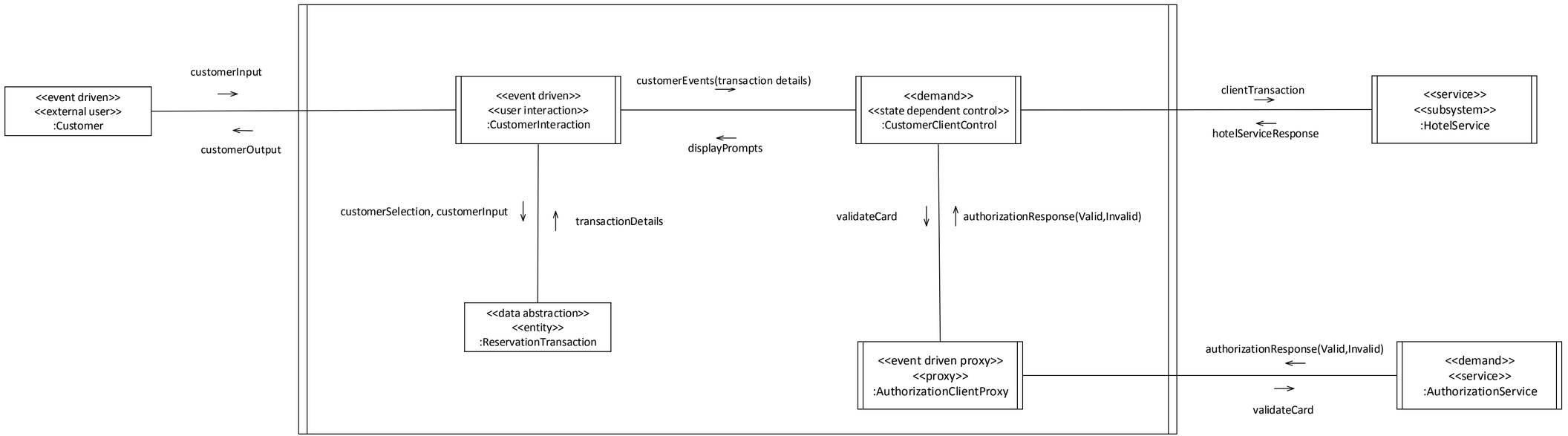

<<external output device>>
Printer
<<external output device>>
KeyPrinter
<<external user>>
FrontDeskClerk
<<external system>>
AuthorizationSystem
1
1
1
1
<<Software System>>
HotelSystem
<<client>>
<<subsystem>>
ClerkClient
<<service>>
<<subsystem>>
HotelService
<<client>>
<<subsystem>>
CustomerClient
<<external user>>
Customer
1..*
Requests
service from
Requests
service from
1..*
1
Requests
service from
<<external timer>>
Timer
1..*
1
<<client>>
<<subsystem>>
ManagerClient
1
<<external user>>
Manager

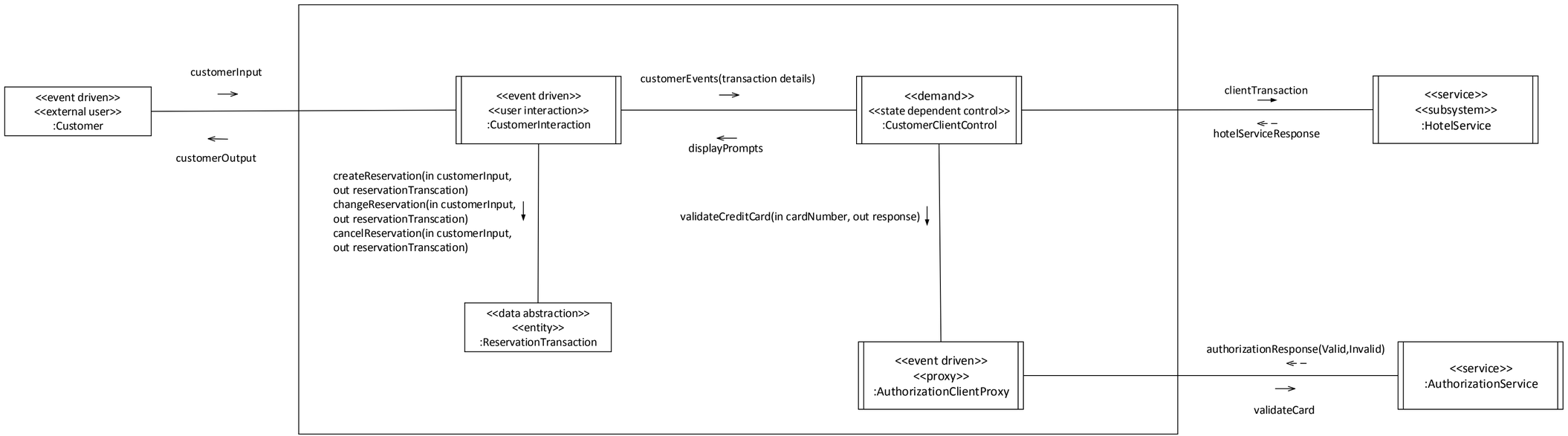

<<external output device>>
:Printer
<<external output device>>
:KeyPrinter
<<external user>>
:FrontDeskClerk
<<external system>>
:AuthorizationSystem
printerOutput
keyPrinterOutput
display information
clerkInput
Card Info
Authorization Output
Card Info
Authorization Output
Card Info
Authorization Output
<<Software System>>
HotelSystem
<<client>>
<<subsystem>>
ClerkClient
Transaction
hotelResponse
<<service>>
<<subsystem>>
:HotelService
Transaction
hotelResponse
<<client>>
<<subsystem>>
CustomerClient
customerInput
display information
<<external user>>
:Customer
Transaction
hotelResponse
<<client>>
<<subsystem>>
ManagerClient
<<external timer>>
:Timer
Timer Signal
customerInput
display information
<<external user>>
Manager

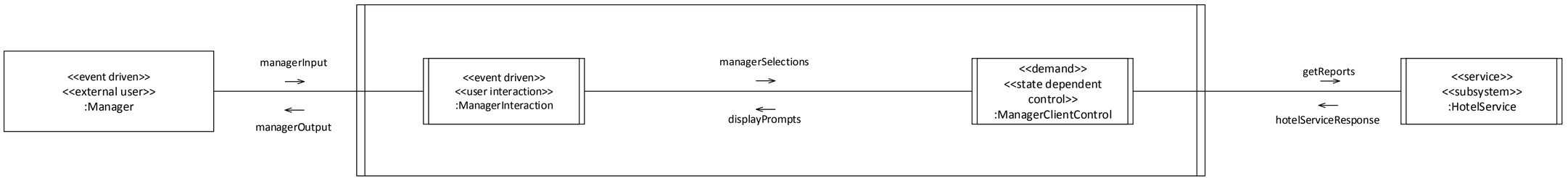

HotelService
{server node}
WAN
Clerk Client 1
{client node}
Clerk Client2
{client node}
Customer Client1
{client node}
Customer Client2
{client node}
Manager Client1
{client node}
Manager Client2
{client node}

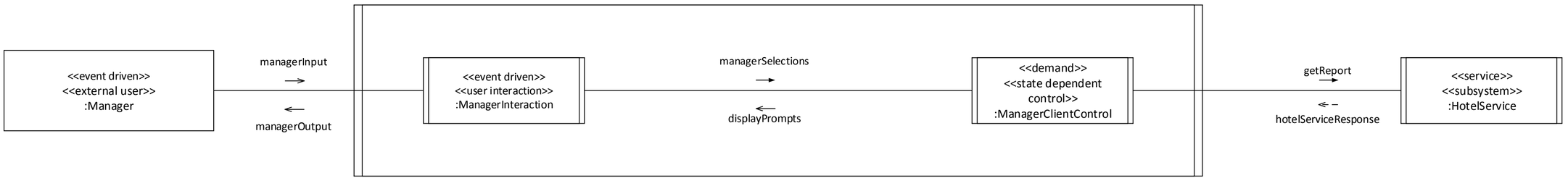

<<data abstraction>>
<<entity>>
:ReservationTranscation
requestReservation
sendReservation
<<demand>>
<<output>>
:KeyPrinterInterface
keyPrinterOutput
<<passive>>
<<external output device>>
:KeyPrinter
sendReservation
requestReservation
<<passive>>
<<external output device>>
:Printer
printerOutput
<<demand>>
<<output>>
:PrinterInterface
printRoomInfo
printRoomKey
transactionDetails
clerkselection, clerk input
<<event driven>>
<<external user>>
:Clerk
clerkInput
clerkOutput
<<event driven>>
<<user interaction>>
:ClerkInteraction
clerkEvents
displayPrompts
<<demand>>
<<state dependent control>>
:ClerkClientControl
clientTransactions
hotelServiceResponse
<<service>>
<<subsystem>>
:HotelService
validateCard
authorizationResponse
<<event driven proxy>>
<<proxy>>
:AuthorizationClientProxy
validateCard
authorizationResponse
<<service>>
:AuthorizationService

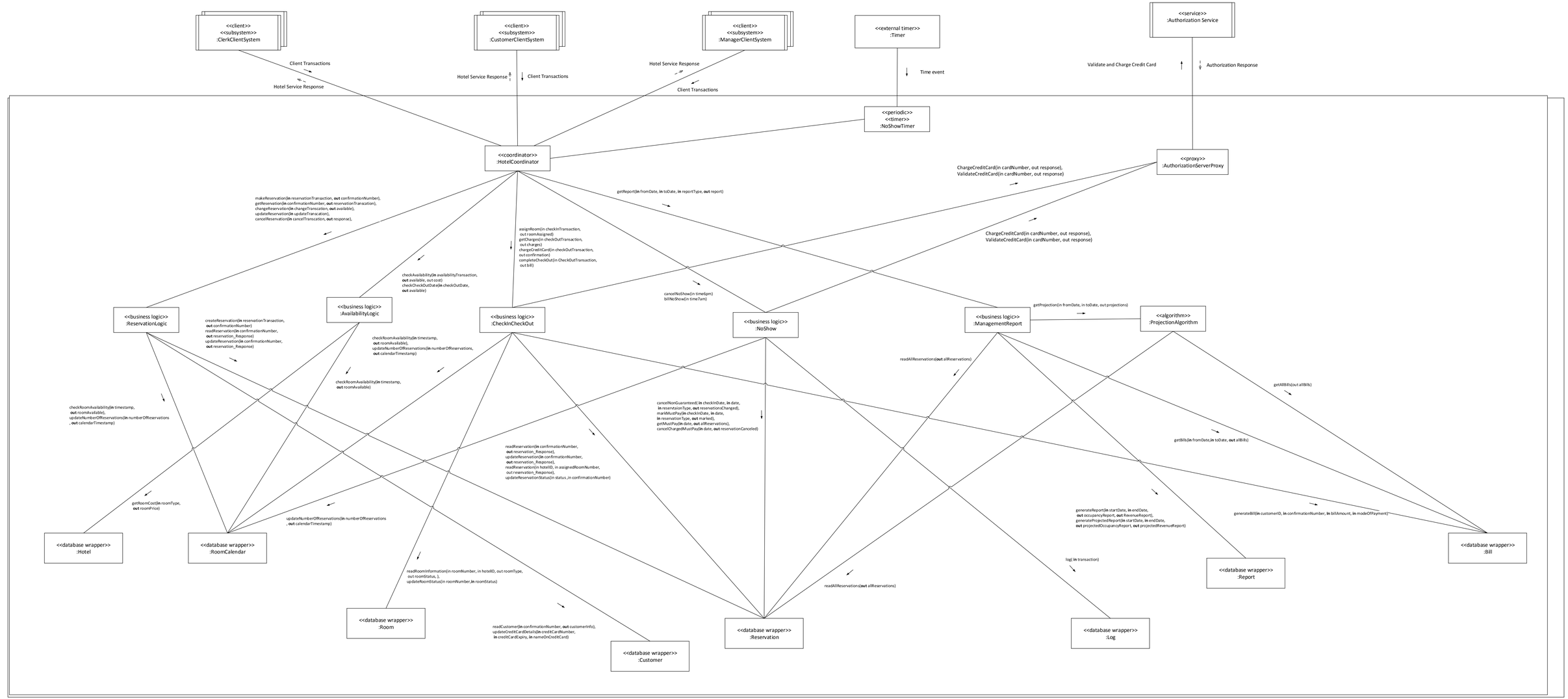

<<data abstraction>>
<<entity>>
:ReservationTranscation
read(out reservationTransaction)
<<demand>>
<<output>>
:KeyPrinterInterface
keyPrinterOutput
<<passive>>
<<external output device>>
:KeyPrinter
read(out reservationTransaction)
<<passive>>
<<external output device>>
:Printer
printerOutput
<<demand>>
<<output>>
:PrinterInterface
printRoomInfo
printRoomKey
createReservation(in customerInput,
out reservationTranscation)
changeReservation(in customerInput,
out reservationTranscation)
cancelReservation(in customerInput,
out reservationTranscation)
checkIn(in customerInput,
out reservationTranscation)
checkOut(in customerInput,
out reservationTranscation)
<<event driven>>
<<external user>>
:Clerk
clerkInput
clerkOutput
<<event driven>>
<<user interaction>>
:ClerkInteraction
clerkEvents(reservationTranscation)
displayPrompts
<<demand>>
<<state dependent
control>>
:ClerkClientControl
clientTransactions
hotelServiceResponse
<<service>>
<<subsystem>>
:HotelService
validateCreditCard(in cardNumber, out AuthorizationResponse)
<<event driven>>
<<proxy>>
:AuthorizationClientProxy
validateCard
authorizationResponse
<<service>>
:AuthorizationService


<<event driven>>
<<external user>>
:Customer
customerInput
customerOutput
<<event driven>>
<<user interaction>>
:CustomerInteraction
customerEvents(transaction details)
displayPrompts
<<demand>>
<<state dependent control>>
:CustomerClientControl
clientTransaction
hotelServiceResponse
<<service>>
<<subsystem>>
:HotelService
customerSelection, customerInput
transactionDetails
validateCard
authorizationResponse(Valid,Invalid)
<<data abstraction>>
<<entity>>
:ReservationTransaction
<<event driven proxy>>
<<proxy>>
:AuthorizationClientProxy
authorizationResponse(Valid,Invalid)
validateCard
<<demand>>
<<service>>
:AuthorizationService


<<event driven>>
<<external user>>
:Customer
customerInput
customerOutput
<<event driven>>
<<user interaction>>
:CustomerInteraction
customerEvents(transaction details)
displayPrompts
<<demand>>
<<state dependent control>>
:CustomerClientControl
clientTransaction
hotelServiceResponse
<<service>>
<<subsystem>>
:HotelService
createReservation(in customerInput,
out reservationTranscation)
changeReservation(in customerInput,
out reservationTranscation)
cancelReservation(in customerInput,
out reservationTranscation)
validateCreditCard(in cardNumber, out response)
<<data abstraction>>
<<entity>>
:ReservationTransaction
<<event driven>>
<<proxy>>
:AuthorizationClientProxy
authorizationResponse(Valid,Invalid)
validateCard
<<service>>
:AuthorizationService


<<event driven>>
<<external user>>
:Manager
managerInput
managerOutput
<<event driven>>
<<user interaction>>
:ManagerInteraction
managerSelections
displayPrompts
<<demand>>
<<state dependent
control>>
:ManagerClientControl
getReports
hotelServiceResponse
<<service>>
<<subsystem>>
:HotelService


<<event driven>>
<<external user>>
:Manager
managerInput
managerOutput
<<event driven>>
<<user interaction>>
:ManagerInteraction
managerSelections
displayPrompts
<<demand>>
<<state dependent
control>>
:ManagerClientControl
getReport
hotelServiceResponse
<<service>>
<<subsystem>>
:HotelService


<<client>>
<<subsystem>>
:ClerkClientSystem
<<client>>
<<subsystem>>
:CustomerClientSystem
<<client>>
<<subsystem>>
:ManagerClientSystem
<<external timer>>
:Timer
<<service>>
:Authorization Service
Client Transactions
Hotel Service Response
Hotel Service Response
Client Transactions
Hotel Service Response
Client Transactions
Time event
Validate and Charge Credit Card
Authorization Response
<<periodic>>
<<timer>>
:NoShowTimer
<<coordinator>>
:HotelCoordinator
<<proxy>>
:AuthorizationServerProxy
ChargeCreditCard(in cardNumber, out response),
ValidateCreditCard(in cardNumber, out response)
ChargeCreditCard(in cardNumber, out response),
ValidateCreditCard(in cardNumber, out response)
makeReservation(in reservationTransaction, out confirmationNumber),
getReservation(in confirmationNumber, out reservationTransaction),
changeReservation(in changeTransaction, out available),
updateReservation(in updateTransaction),
cancelReservation(in cancelTransaction, out response),
getReport(in fromDate, in toDate, in reportType, out report)
assignRoom(in checkInTransaction,
out roomAssigned)
getCharges(in checkOutTransaction,
out charges)
chargeCreditCard(in checkOutTransaction,
out confirmation)
completeCheckOut(in CheckOutTransaction,
out bill)
checkAvailability(in availabilityTransaction,
out available, out cost)
checkCheckOutDate(in checkOutDate,
out available)
cancelNoShow(in time6pm)
billNoShow(in time7am)
<<business logic>>
:ReservationLogic
<<business logic>>
:AvailabilityLogic
<<business logic>>
:CheckInCheckOut
<<business logic>>
:NoShow
<<business logic>>
:ManagementReport
getProjection(in fromDate, in toDate, out projections)
<<algorithm>>
:ProjectionAlgorithm
createReservation(in reservationTransaction,
out confirmationNumber)
readReservation(in confirmationNumber,
out reservation_Response)
updateReservation(in confirmationNumber,
out reservation_Response)
checkRoomAvailability(in timestamp,
out roomAvailable),
updateNumberOfReservations(in numberOfReservations,
out calendarTimestamp)
checkRoomAvailability(in timestamp,
out roomAvailable)
readAllReservations(out allReservations)
getAllBills(out allBills)
checkRoomAvailability(in timestamp,
out roomAvailable),
updateNumberOfReservations(in numberOfReservations
, out calendarTimestamp)
cancelNonGuaranteed(in checkInDate , in date,
in reservationType, out reservationsChanged),
markMustPay(in checkInDate, in date,
in reservationType, out marked),
getMustPay(in date, out allReservations),
cancelChargedMustPay(in date, out reservationCanceled)
getBills(in fromDate,in toDate, out allBills)
readReservation(in confirmationNumber,
out reservation_Response),
updateReservation(in confirmationNumber,
out reservation_Response),
readReservation(in hotelID, in assignedRoomNumber,
out reservation_Response),
updateReservationStatus(in status ,in confirmationNumber)
getRoomCost(in roomType,
out roomPrice)
updateNumberOfReservations(in numberOfReservations
, out calendarTimestamp)
generateReport(in startDate, in endDate,
out occupancyReport, out RevenueReport),
generateProjectedReport(in startDate, in endDate,
out projectedOccupancyReport, out projectedRevenueReport)
generateBill(in customerID, in confirmationNumber, in billAmount, in modeOfPayment)
<<database wrapper>>
:Hotel
<<database wrapper>>
:RoomCalendar
<<database wrapper>>
:Bill
log(in transaction)
<<database wrapper>>
:Report
readRoomInformation(in roomNumber, in hotelID, out roomType,
out roomStatus,),
updateRoomStatus(in roomNumber,in roomStatus)
readAllReservations(out allReservations)
<<database wrapper>>
:Room
readCustomer(in confirmationNumber, out customerInfo),
updateCreditCardDetails(in creditCardNumber,
in creditCardExpiry, in nameOnCreditCard)
<<database wrapper>>
:Reservation
<<database wrapper>>
:Log
<<database wrapper>>
:Customer

<<database wrapper>>
Hotel

+create(**in** hotelChainName, **in** hotelName, **in** street,**in** city,**in** state,**in** zipcode)
+updateTotalRooms (**in** totalRooms)
+updateSingleRooms(**in** singleRooms)
+updateDoubleRooms(**in** doubleRooms)
+updateSingleRoomPrice(**in** singleRoomPrice)
+updateDoubleRoomPrice(**in** doubleRoomPrice)
+updateGroupBookingPercentage(**in** groupBookingPercentage)
+delete(**in** hotelId)
+getRoomCost(**in** roomType, **out** roomPrice)

<<database wrapper>>
Room

+createRoom(**in** roomNumber,**in** hotelID,**in** roomType)
+updateRoomStatus(**in** roomNumber,**in** roomStatus)
+readRoomInformation(**out** roomType,**out** roomStatus,**in** roomNumber,**in** hotelID)
+delete(**in** roomNumber, **in** hotelID)

<<database wrapper>>
Customer

+createCustomer(**in** firstName,**in** lastname,**in** contactNumber,**in** email)
+updateConfirmationNumber(**in** confirmationNumber)
+updateCreditCardDetails(**in** creditCardNumber,**in** creditCardExpiry,**in** nameOnCreditCard)
+readCustomer(**in** confirmationNumber,**out** customerInfo)
+delete(customerID)
+readCreditCard(in customerID, out CreditCardNumber, out creditCardExpiry, out NameOnCreditCard)

<<database wrapper>>
Room Calendar

+createCalendar(**in** hotelID, **in** totalRooms)
+updateNumberofReservations(**in** numberOfReservations, **in** calendarTimeStamp)
+updateNumberOfRoomsOccupied(**in** numberOfRoomsOccupied)
+updateNumberOfRoomsUnderMaintanence(**in** numberofRoomsUnderMaintenance)
+readRoomsAvailable(**out** numberOfRoomsAvailable)
+checkRoomAvailability(**in** timestamp,**out** roomAvailable)

<<database wrapper>>
Reservation

allReservations : Reservations

+createReservation(**in** reservationTransaction,**out** confirmationNumber)
+updateAssignedRoomNumber(**in** assignedRoomNumber)
+updateCheckInDate(**in** checkInDate)
+updateCheckOutDate(**in** checkOutDate)
+updateReservationStatus(**in** status,**in** confirmationNumber)
+readReservation(**in** confirmationNumber,**out** reservation_Response)
+updateReservation(**in** confirmationNumber, **out** reservation_Response)
+readReservation(**in** hotelID,**in** assignedRoomNumber, **out** reservation_response)
+cancelNonGuaranteed(**in** checkInDate,**in** date,**in** reservationType,**out** reservationsChanged)
+markMustPay(**in** checkIndate,**in** date,**in** reservationType,**out** marked)
+getMustPay(**in** date,**out** allReservations)
+cancelChargedMustPay(**in** date,**out** reservationCancelled)
+readReservations(**in** fromDate,**in** toDate,**out** allReservations)
+readAllReservations(**out** allReservations)

<< database wrapper>>
Log

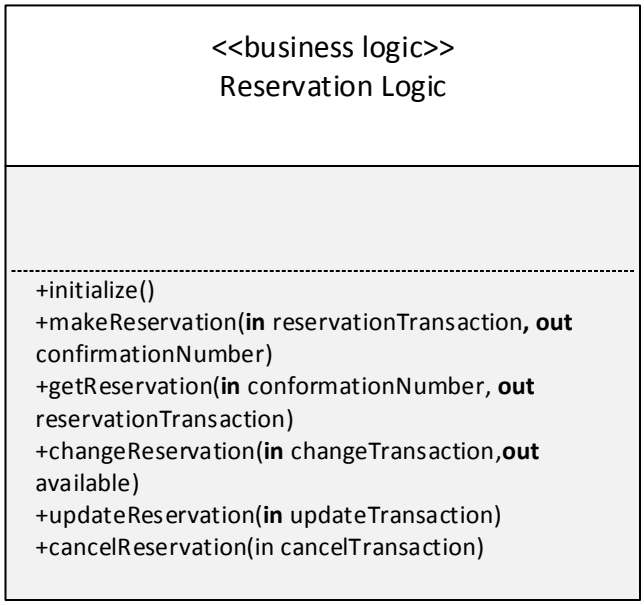
+ read(**out** transaction)
+ log(**in** transaction)

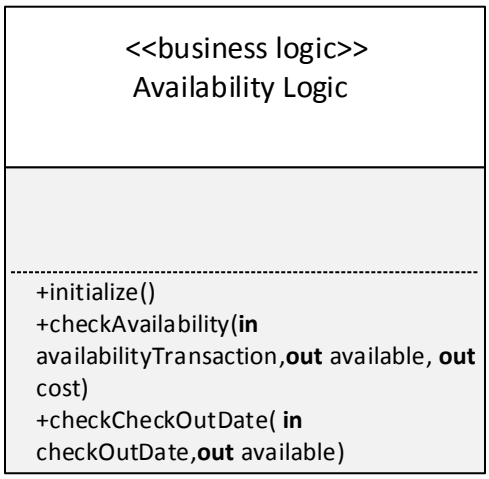
<<database wrapper>>
Report

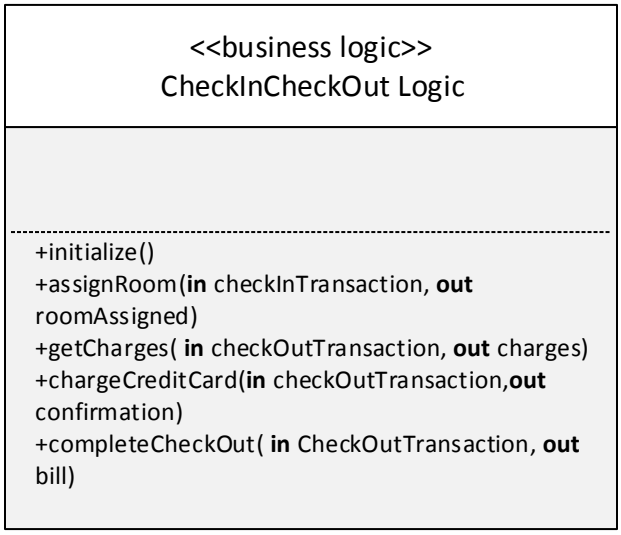
+ generateReport(**in** startDate, **in** endDate, **out** occupancyReport, **out** RevenueReport)
+ generateProjectedReport(**in** startDate, **in** endDate, **out** projectedOccupancyReport, **out** projectedRevenueReport)

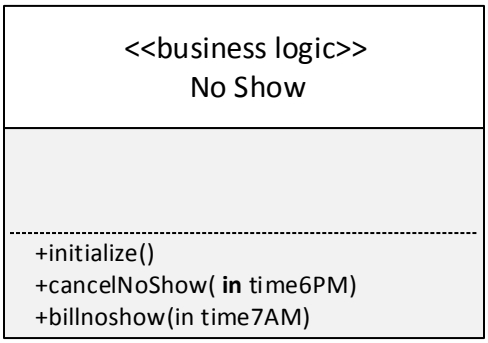
<<database wrapper>>
Bill

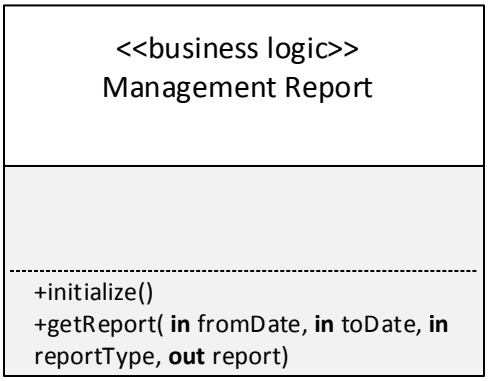
+ generateBill(**in** customerID, **in** confirmationNumber,**in** billAmount, **in** modeOfPayment)
+ updateBillAmount(**in** billAmount)
+ updateModeOfPayment(**in** modeOfPayment)
+ getBill(**in** customerID, **in** confirmationNumber, **out** billAmount, **out** modeOfPayment)
+ getBills(**in** fromDate,**in** toDate, **ou**t allBills)
+ getAllBills(**out** allBills)

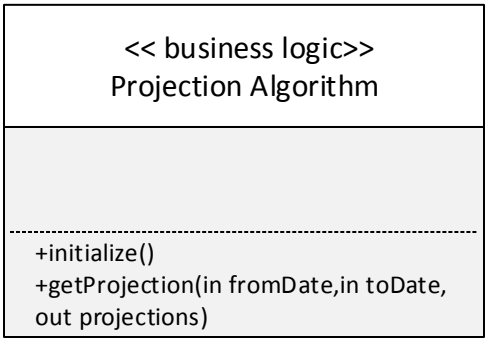

<<business logic>>
Reservation Logic
+initialize()
+makeReservation(in reservationTransaction, out confirmationNumber)
+getReservation(in conformationNumber, out reservationTransaction)
+changeReservation(in changeTransaction,out available)
+updateReservation(in updateTransaction)
+cancelReservation(in cancelTransaction)

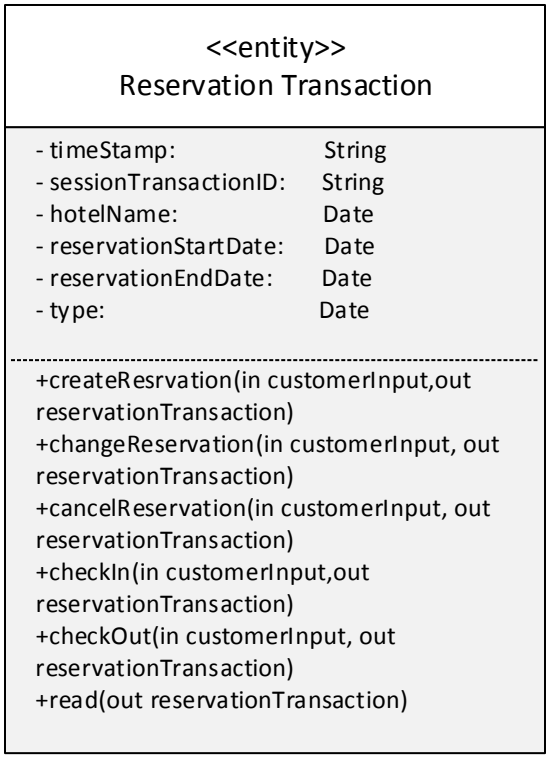

<<business logic>>
Availability Logic
+initialize()
+checkAvailability(in availabilityTransaction,out available, out cost)
+checkCheckOutDate(in checkOutDate,out available)


<<business logic>>
CheckInCheckOut Logic
+initialize()
+assignRoom(in checkInTransaction, out roomAssigned)
+getCharges(in checkOutTransaction, out charges)
+chargeCreditCard(in checkOutTransaction,out confirmation)
+completeCheckOut(in CheckOutTransaction, out bill)


<<business logic>>
No Show
+initialize()
+cancelNoShow(in time6PM)
+billnoshow(in time7AM)


<<business logic>>
Management Report
+initialize()
+getReport(in fromDate, in toDate, in reportType, out report)


<< business logic>>
Projection Algorithm
+initialize()
+getProjection(in fromDate,in toDate, out projections)


<<entity>>
Reservation Transaction
- timeStamp: String
- sessionTransactionID: String
- hotelName: Date
- reservationStartDate: Date
- reservationEndDate: Date
- type: Date
+createResrvation(in customerInput,out reservationTransaction)
+changeReservation(in customerInput, out reservationTransaction)
+cancelReservation(in customerInput, out reservationTransaction)
+checkIn(in customerInput,out reservationTransaction)
+checkOut(in customerInput, out reservationTransaction)
+read(out reservationTransaction)

Database Wrapper Class Interface Specifications

1.

Information Hiding Class : Hotel
Information Hidden : Encapsulates hotel attributes and hides details of how to interface to database management system.
Class Structuring Criterion :Database Wrapper Class
Assumptions : None.
Anticipated Changes : None.
Superclass : None.
Inherited Attributes : None.
Inherited Operations : None.
Attributes : None.
Operations Provided :
1. create(**in** hotelChainName:String, **in** hotelName:String, **in** street:String,**in** city:String,**in** state:String,**in** zipcode:int)

Function: Creates a new hotel object with the given attributes.
Precondition: None
Postcondition: A new hotel entity is created with the specified values.
Input Parameters: hotelChainName- the name you want to give, hotelName - the name of the hotel under which you want to add this new chain, street- street name, city- name of the city where your hotel is located,state- state name, zipcode- zip of the location.
Operations Used:None.

2.updateTotalRooms (**in** totalRooms:int)

Function: Updates the number of rooms available.
Precondition: None
Postcondition: The total number of rooms available in the hotel are changed after this method is executed.
Input Parameters: totalRooms- this number is taken and those many rooms are available..
Operations Used:None.

3. updateSingleRooms(**in** singleRooms:int)

Function: Updates the number of single rooms available.
Precondition: None
Postcondition: The total number of single rooms available in the hotel are changed after this method is executed.
Input Parameters: singleRooms- this number is taken and those many singlerooms are made available..
Operations Used:None.

4. updateDoubleRooms(**in** doubleRooms:int)

Function: Updates the number of double rooms available.
Precondition: None
Postcondition: The total number of double rooms available in the hotel are changed after this method is executed.
Input Parameters: doubleRooms- this number is taken and those many double rooms are made available..
Operations Used:None.

5. updateSingleRoomPrice(**in** singleRoomPrice:double)

Function: Updates the price of single room.
Precondition: None
Postcondition: A new price is set for each single room.
Input Parameters: singleRoomPrice- this number is taken and this amount is charged for each single room.
Operations Used:None.

6. updateDoubleRoomPrice(**in** doubleRoomPrice:double)

Function: Updates the price of double room.
Precondition: None
Postcondition: A new price is set for each double room.
Input Parameters: doubleRoomPrice- this number is taken and this amount is charged for each double room.
Operations Used:None.

7. updateGroupBookingPercentage(**in** groupBookingPercentage:double)

Function: Updates the percentage of discount applied when group booking is done.
Precondition: None
Postcondition: A new percentage is set for group booking.
Input Parameters: groupBookingPercentage- this new percentage value is taken into consideration.
Operations Used:None.

8. delete(**in** hotelId)

Function: Deletes the hotel entity with the given hotelID from the system.
Precondition: There should be a hotel existing with that hotelID in the system
Postcondition: The hotel with give hotelID gets deleted from the system.
Input Parameters: hotelID- this value is taken and the matching hotel is deleted from the system.
Operations Used:None.

9. getRoomCost(**in** roomType, **out** roomPrice:double)

Function: Takes the room type as input and returns its price as output.
Precondition: The given room type should exist in the system.
Postcondition: The price of the input room type is displayed on the standard O/P.
Input Parameters: roomType- this value is taken to display its cost.
Output Parameters: roomPrice- the cost of each room.
Operations Used:None

2.

Information Hiding Class : Room
Information Hidden : Encapsulates room attributes and hides details of how to interface to database management system.
Class Structuring Criterion :Database Wrapper Class
Assumptions : None.
Anticipated Changes : None.

Superclass : None.
Inherited Attributes : None.
Inherited Operations : None.
Attributes : None.
Operations Provided :

1. createRoom(in roomNumber:int,in hotelID:String,in roomType:String)

Function: Takes the roomNumber,hotelID and roomType and creates a room entity with given specifications
Precondition: None.
Postcondition: A new room type is created.
Input Parameters: roomNumber- a new unique room number is given to the room, hotelID- the Unique ID of the hotel where this new room has to be created is given, roomType- this value is taken to display its cost.
Operations Used:None

2. updateRoomStatus(in roomNumber:int,in roomStatus:String)

Function: This method updates the room status, ex. occupied, under maintenance etc.
Precondition: The specified room number should exist.
Postcondition: The status of the room gets changed after the successful execution of the method .
Input Parameters: roomStatus- this value is assigned and displayed as the status of the room,roomNumber- The room number whose status should be changed.
Operations Used:None

3. readRoomInformation(out roomType:String,out roomStatus:String,in roomNumber:int,in hotelID:String)

Function: This method displays roomNumber and hotelID by taking in roomType and roomStatus as inputs.
Precondition: The specified room number should exist.
Postcondition: The roomNumber and HotelID are displayed .
Invariant: Values of the room remain unchanged.

Output Parameters: roomType- this is the type of room, roomStatus- status of the room.
Input Parameters: roomNumber,hotelID
Operations Used:None

4. delete(in roomNumber:int, in hotelID:String)

Function: This method deletes the room with specified number in the mentioned hotel.
Precondition: The specified room number should exist.
Postcondition: The roomNumber gets erased from the system .
Input Parameters: roomNumber- this is the number of the room, hotelID- the unique value given to each room.
Operations Used:None

3.

Information Hiding Class : Customer
Information Hidden : Encapsulates customer attributes and hides details of how to interface to database management system.
Class Structuring Criterion :Database Wrapper Class
Assumptions : None.
Anticipated Changes : None.
Superclass : None.
Inherited Attributes : None.
Inherited Operations : None.
Attributes : None.
Operations Provided :

1. createCustomer(in firstName:String,in lastname:String,in contactNumber:int,in email:String)

Function: This method creates a new customer entity with the given attributes.
Precondition: None.
Postcondition: A new customer object is created .

Input Parameters: firstName- the first name of the customer, lastName- last name of the customer, contactNumber- phone number of the customer should be entered, email- email address of the customer.
Operations Used:None

2. updateConfirmationNumber(in confirmationNumber:int)

Function: takes in a conformationNumber and generates a new conformation Number.
Precondition: None.
Postcondition: The conformationNumber is updated.
Input Parameters: conformationNumber- the existing conformation number given at the time of booking.
Operations Used:None

3. updateCreditCardDetails(in creditCardNumber:int,in creditCardExpiry:date,in nameOnCreditCard:String)

Function: Updates the credit card details with the provided ones.
Precondition: None.
Postcondition: New credit card details are stored in the system.
Input Parameters: creditCardNumber- number present on the top of the credit card,creditCardExpiry- the credit card expiry date, nameOnCeditCard- card holders name.
Operations Used:None

4. readCustomer(in confirmationNumber:int,out customerInfo:Customer)

Function: Displays the customer info on receiving the confirmation number.
Precondition: The specified conformation number should exist.
Postcondition: The customer details are displayed .
Input Parameters: conformationNumber- the number given to the customer at the time of booking.
Output Parameters: customerInfo- the customer details such as name, credit card details etc.
Operations Used:None

5. delete(customerID)

Function: Deletes the customer entity with the given customerID
Precondition: The specified customer should exist.
Postcondition: The specified customer entity is deleted.
Input Parameters: customerID- the unique ID associated with the customer.
Operations Used:None

6. readCreditCard(in customerID, out CreditCardNumber, out creditCardExpiry, out NameOnCreditCard)

Function: Displays credit card information of the customer upon receiving the customerID.
Precondition: The specified customer should exist.
Postcondition: The specified customers card details are displayed .
Input Parameters: customerID- the unique ID associated with the customer.
Output Parameters:credit card number,credit card expiry, name on credit card.
Operations Used:None

14.

Information Hiding Class : Room Calendar
Information Hidden : Encapsulates room calendar attributes and hides details of how to interface to database management system.
Class Structuring Criterion :Database Wrapper Class
Assumptions : None.
Anticipated Changes : None.
Superclass : None.
Inherited Attributes : None.
Inherited Operations : None.
Attributes : None.
Operations Provided :

1. createCalendar(in hotelID:String, in totalRooms:int)

Function: Creates the calendar entity with the given hotelID and totalRooms.
Precondition: The specified hotelID should exist.

Postcondition: The specified customer entity is deleted.
Input Parameters: customerID- the unique ID associated with the customer.
Operations Used:None

2. updateNumberofReservations(in numberOfReservations:int, in calendarTimeStamp:date)

Function: Updates the number of reservations made .
Precondition: None.
Postcondition: The number of reservations made are updated in the system.
Input Parameters: numberOfReservations- a number representing the reservations made, calendarTimeStamp- the time and date at that point of time.
Operations Used:None

3. updateNumberOfRoomsOccupied(in numberOfRoomsOccupied:int)
Function: Updates the number of rooms occupied .
Precondition: None.
Postcondition: The number of available rooms is updated.
Input Parameters: numberOfRoomsOccupied- a number representing the no.of occupied rooms.
Operations Used:None

4. updateNumberOfRoomsUnderMaintanence(in numberofRoomsUnderMaintenance:int)

Function: Updates the number of rooms under maintenance .
Precondition: None.
Postcondition: The number of rooms under maintenance is updated.
Input Parameters: numberOfRoomsUnderMaintenance- a number representing the no.of rooms under maintenance.
Operations Used:None

5. readRoomsAvailable(out numberOfRoomsAvailable:int)

Function: Displays the no. of rooms available on the standard I/O.
Precondition: None.
Postcondition: The number of rooms under maintenance is updated.

Input Parameters: None.
Output Parameters: numberOfRoomsAvailable-Displays the number of rooms available.
Operations Used:None

6. checkRoomAvailability(in timestamp:date,out roomAvailable:boolean)

Function: Displays the room availability.
Precondition: None.
Postcondition: The availability of a particular room is displayed.
Input Parameters: timeStamp- current timeStamp.
Output Parameters: roomAvailable-Displays whether the room is available or not.
Operations Used:None

5.

Information Hiding Class : Reservation
Information Hidden : Encapsulates reservation attributes and hides details of how to interface to database management system.
Class Structuring Criterion :Database Wrapper Class
Assumptions : None.
Anticipated Changes : None.
Superclass : None.
Inherited Attributes : None.
Inherited Operations : None.
Attributes : None.
Operations Provided :

1. createReservation(in reservationTransaction,out confirmationNumber:int)

Function: Creates a new reservation transaction.
Precondition: None.
Postcondition: A new reservation entity is created.
Input Parameters: reservationTransaction- Reservations details.
Output Parameters: confirmationNumber-A unique identification number that associates a particular reservation with a customer.

Operations Used:None

2. updateAssignedRoomNumber(in assignedRoomNumber)

Function: Alters already assigned room number.
Precondition: None.
Postcondition: A new roomNumber is generated.
Input Parameters: assignedRoomNumber- The number with which the room is identified.
Operations Used:None

3. updateCheckInDate(in checkInDate)

Function: Updates check in date.
Precondition: None.
Postcondition: A new check in date is generated.
Input Parameters: checkInDate- The date when customer lodges the room.
Operations Used:None

4. updateCheckOutDate(in checkOutDate)

Function: Updates check out date.
Precondition: None.
Postcondition: A new check out date is generated.
Input Parameters: checkOutDate- The date when customer leaves the room.
Operations Used:None

5. updateReservationStatus(in status,in confirmationNumber)

Function: Updates reservation status.
Precondition:A reservation must have been already made.
Postcondition: Reservation details are updated.
Input Parameters: status- The current state of the reservation,conformationNumber- a unique number which can pull the reservation details pertaining to a reservation transaction.
Operations Used:None

6. readReservation(**in** confirmationNumber,**out** reservation_Response)

Function: Displays reservation details upon receiving conformationNumber.
Precondition:A reservation must be existing with the given conformationNumber.
Postcondition: Reservation details are displayed .
Input Parameters: conformationNumber- unique number which pulls the reservation details.
Output Parameters: reservation_Response- reservation details displayed.
Operations Used:None

7. cancelNonGuaranteed(**in** checkInDate:date,**in** date:date,**in** reservationType:String,**out** reservationsChanged:String)

Function: Automatically cancels the reservations which are not guaranteed, after specified time .
Precondition:A non-guaranteed reservation must have already been made.
Postcondition: The changed reservation is diplayed, the cancelled room is made available for re-reservation .
Input Parameters: checkInDate- the date specified by the customer when he is going to come,date- present day's date,reservationType- Category of reservation.
Output Parameters: reservationChanged- modified reservation.
Operations Used:None

8. markMustPay(in checkIndate,in date,in reservationType,out marked)

Function: Identifies all the reservations that are to be charged that particular day.
Precondition: None.
Postcondition: Reservations are identified, based on their check-in date .
Input Parameters: checkInDate- the date chosen by the customer to checkin, date- present day's date,reservationType- specifies the category of reservation.
Output Parameters: marked- all the reservations that have to be charged.
Operations Used:None

9. getMustPay(in date,out allReservations)

Function: Pulls all reservations that have to make a payment on that particular calendar day .

Precondition: None.
Postcondition: All the have to pay reservations are identified.
Input Parameters: date- present day's date.
Output Parameters: allRservations- all the reservations that have to be charged.
Operations Used:None

10. cancelChargedMustPay(in date,out reservationCancelled)

Function: Cleans up the system by deleting all the marked and charged reservations .
Precondition: None.
Postcondition: Only those reservations exist which have pending payments.
Input Parameters: date- present day's date.
Output Parameters: reservationCancelled- all the reservations cancelled.
Operations Used:None
11. readReservations(in fromDate,in toDate,out allReservations)

Function: Retrieves the reservations made between the specified dates .
Precondition: None.
Postcondition: All the reservations made the mentioned period are displayed.
Input Parameters: fromDate-desired day's date, toDate- end date of the period,
Output Parameters: allReservations- all the reservations made in the mentioned period.
Operations Used:None

12. readAllReservations(out allReservations)

Function: Retrieves the reservations made between the specified dates .
Precondition: None.
Postcondition: All the reservations made the mentioned period are displayed.
Output Parameters: allReservations- all the reservations made in the mentioned period.
Operations Used:None

6.

Information Hiding Class : Log

Information Hidden : Encapsulates Log attributes and hides details of how to interface to database management system.
Class Structuring Criterion :Database Wrapper Class
Assumptions : None.
Anticipated Changes : None.
Superclass : None.
Inherited Attributes : None.
Inherited Operations : None.
Attributes : None.
Operations Provided :

1. read(out transaction)

Function: Reads the transactions from log and displays all of them.
Precondition: None.
Postcondition: All the reservations made the mentioned period are displayed.
Output Parameters: allReservations- all the reservations made in the mentioned period.
Operations Used:None

2. log(in transaction)

Function: Stores the given transaction in the log .
Precondition: None.
Postcondition: The log is saved on the database for future reference.
Input Parameters: transaction- details pertaining to a reservation.
Operations Used:None

7.

Information Hiding Class : Report
Information Hidden : Encapsulates Report attributes and hides details of how to interface to database management system.
Class Structuring Criterion :Database Wrapper Class
Assumptions : None.
Anticipated Changes : None.

Superclass : None.
Inherited Attributes : None.
Inherited Operations : None.
Attributes : None.
Operations Provided :

1. generateReport(in startDate, in endDate, out occupancyReport, out RevenueReport)

Function: Generates a report on the number of rooms occupied and revenue generated from the occupation, within the specified period .
Precondition: None.
Postcondition: A report is generated.
Input Parameters: startDate- the beginning date from which the system should look for occupancy, endDate- the date until which the system should generate the report.
Output Parameters: occupancyReport- document displaying the occupancy rate of the hotel, revenueReport- document displaying the revenue generated from the occupied rooms.
Operations Used:None

2. generateProjectedReport(in startDate, in endDate, out projectedOccupancyReport, out projectedRevenueReport)

Function: Generates reports based on the execution of projection algorithm.
Precondition: None.
Postcondition: The generated report indicates the expected occupancy and expected revenue from that.
Input Parameters: startDate- the beginning date from which the system should look for occupancy, endDate- the date until which the system should generate the report.
Output Parameters: projectedOccupancyReport- document displaying the projected occupancy rate of the hotel, projectedRevenueReport- document displaying the projected revenue generated from the occupied rooms.
Operations Used:None

8.

Information Hiding Class : Bill
Information Hidden : Encapsulates Bill attributes and hides details of how to interface to database management system.
Class Structuring Criterion :Database Wrapper Class
Assumptions : None.
Anticipated Changes : None.
Superclass : None.
Inherited Attributes : None.
Inherited Operations : None.
Attributes : None.
Operations Provided :

1. generateBill(in customerID, in confirmationNumber,in billAmount, in modeOfPayment)

Function: Creates a bill by taking in customerID, confirmationNumber, billAmount and mode of payment.
Precondition: None.
Postcondition: A bill is generated from the inputs.
Input Parameters: customerID- The unique number with which a customer is identified, conformationNumber- the unique number with which each reservation is identified,billAmount- the amount for which the customer should be charged for,modeOfPayment- type of payment option the customer has chosen.
Output Parameters:None.
Operations Used:None

2. getBill(in customerID, in confirmationNumber, out billAmount, out modeOfPayment)

Function: Pulls out the bill when we provide customerID, confirmationNumber.
Precondition: None.
Postcondition: The bill is diplayed.
Input Parameters: customerID- The unique number with which a customer is identified, conformationNumber- the unique number with which each reservation is identified.

Output Parameters:billAmount- the amount for which the customer has been charged,modeOfPayment- kind of payment option the customer has chosen.
Operations Used:None

3. getBills(in fromDate,in toDate, out allBills)

Function: Pulls out all the bills generated between the from date and the to date.
Precondition: None.
Postcondition: The bills are displayed.
Input Parameters: fromDate- The date from which the system should start looking for bills,toDate-the date until when the system should look for bills.
Output Parameters:allBills- The bills that have been generated in the mentioned period.
Operations Used:None
4. getAllBills(out allBills)

Function: Pulls all the bills from the inception of the system.
Precondition: None.
Postcondition: The bills are displayed.
Input Parameters: None.
Output Parameters:allBills- The bills that have been generated from the beginning .
Operations Used:None

Business Logic Class Interface Specifications

1.

Information Hiding Class : Reservation Logic
Information Hidden :Encapsulates reservation specific business rules.
Class Structuring Criterion :Business Logic Class
Assumptions : None.
Anticipated Changes : None.
Superclass : None.
Inherited Attributes : None.
Inherited Operations : None.
Attributes : None.
Operations Provided :

1. makeReservation(in reservationTransaction:reservationTransaction, out confirmationNumber:int)

Function: Makes a reservation for the customer by using the details from client side reservation Transaction, and generates a confirmation number.
Precondition: None.
Postcondition: Confirmation Number generated.
Input Parameters: reservationTransaction- client side details entered by customer/clerk.
Output Parameters:confirmationNumber- a unique number which can identify the reservation number.
Operations Used:None

2. getReservation(in conformationNumber:int, out reservationTransaction:reservationTransaction)

Function: Upon receiving the conformation number , returns the reservation transaction.
Precondition: The specified conformation number should exist.
Postcondition: reservation transaction displayed .
Input Parameters: confirmationNumber- a unique number which can identify the reservation number.

Output Parameters: reservationTransaction- client side details entered by customer/clerk.
Operations Used:None

3. changeReservation(in changeTransaction:reservationTransaction,out available:String)

Function: Makes changes to the reservation and specifies whether the altered room specifications are available.
Precondition: None.
Postcondition: reservation transaction altered and displayed .
Input Parameters: changeTransaction- new transaction information entered at client side.
Output Parameters: available- shows whether the new changes are allowed or not.
Operations Used:None

4. updateReservation(in updateTransaction:reservationTransaction)

Function: Updates the reservation details with the new input.
Precondition: None.
Postcondition: Reservation Details Updated .
Input Parameters: updateTransaction- altered transaction details.
Output Parameters: None.
Operations Used:None

5. cancelReservation(in cancelTransaction:reservationTransaction)

Function: Cancels the reservation details for the given transaction received from client side.
Precondition: The transaction to be cancelled should exist.
Postcondition: Reservation Details Cancelled .
Input Parameters: cancelTransaction- transaction details of the reservation to be cancelled.
Output Parameters: None.

2.

Information Hiding Class : Availability Logic
Information Hidden :Encapsulates occupancy and availability specific business rules.
Class Structuring Criterion :Business Logic Class
Assumptions : None.
Anticipated Changes : None.
Superclass : None.
Inherited Attributes : None.
Inherited Operations : None.
Attributes : None.
Operations Provided :

1. checkAvailability(in availabilityTransaction:reservationTransaction,out available:boolean, out cost:float)

Function: Confirms the availability of a particular selection transaction received from the client side.
Precondition: None.
Postcondition: Sends back available and cost if the operation is successful .
Input Parameters: availabilityTransaction- transaction details of the reservation to be checked at the server side.
Output Parameters: available- message showing whether that particular selection is available or not, cost-price for that selection.

2. checkCheckOutDate(in checkOutDate:date,out available:boolean)

Function: Confirms the availability of a particular check out date.
Precondition: None.
Postcondition: Sends back available if the room is available for those many days .
Input Parameters: checOutDate- the date on which the customer wants to vacate the room.
Output Parameters: available- message showing whether that particular selection is available or not.

3.

Information Hiding Class : CheckInCheckOut Logic
Information Hidden :Encapsulates checkIn and checkOut specific business rules.
Class Structuring Criterion :Business Logic Class
Assumptions : None.
Anticipated Changes : None.
Superclass : None.
Inherited Attributes : None.
Inherited Operations : None.
Attributes : None.
Operations Provided :

1. assignRoom(in checkInTransaction:reservationTransaction, out roomAssigned:boolean)

Function: Assigns room to a customer with the given transaction details .
Precondition: None.
Postcondition: Room Assigned and that room made unavailable until he checks out .
Input Parameters: checkInTransaction- transaction details of the reservation coming from the client.
Output Parameters: roomAssigned- message showing that the room has been assigned.

2. getCharges(in checkOutTransaction:reservationTransaction, out charges:float)
Function: Calculates the amount to be charged based on the check out date .
Precondition: None.
Postcondition: The charges are computed and forwarded to the client side.
Input Parameters: checkOutTransaction- checkOut transaction details of the reservation coming from the client.
Output Parameters: charges- message showing the amount to be paid by the customer.

3. chargeCreditCard(in checkOutTransaction:reservationTransaction,out confirmation:int)

Function: Charges the customer from his credit card, based on the amount which he needs to pay calculated from his check out date.
Precondition: None.
Postcondition: After successful execution amount comes into hotels account from customers credit card.
Input Parameters: checkOutTransaction- checkOut transaction details of the reservation coming from the client.
Output Parameters: conformation- message showing that the payment transaction has been successful .

4. completeCheckOut(in CheckOutTransaction:reservationTransaction, out bill:bill)

Function: Completes the transaction by generating a bill by taking checkOutTransaction into account.
Precondition: None.
Postcondition: Generates a bill.
Input Parameters: checkOutTransaction- checkOut transaction details of the reservation coming from the client.
Output Parameters: bill- a PDF format document containing summary of the entire customers stay .

4.

Information Hiding Class : NoShow Logic
Information Hidden :Contains NoShow specific business rules.
Class Structuring Criterion :Business Logic Class
Assumptions : None.
Anticipated Changes : None.
Superclass : None.
Inherited Attributes : None.
Inherited Operations : None.
Attributes : None.
Operations Provided :

1. cancelNoShow(in time6PM)

Function: Automatically cancels reservations that are not guaranteed and the customer does not show up.
Precondition: None.
Postcondition: Removes the hold put on that room and makes is available again.
Input Parameters: time6PM- timer input coming from external timer.
Output Parameters: None .

2. billnoshow(in time7AM:timestamp)

Function: Automatically bills all the guaranteed reservations when the time is 7AM and the customer has not turned up.
Precondition: None.
Postcondition: The billing process initiated.
Input Parameters: time7AM- timer input coming from external timer.
Output Parameters: None .

5.

Information Hiding Class : Management Report Logic
Information Hidden :Encapsulates manager reporting specific business rules.
Class Structuring Criterion :Business Logic Class
Assumptions : None.
Anticipated Changes : None.
Superclass : None.
Inherited Attributes : None.
Inherited Operations : None.
Attributes : None.
Operations Provided :

1. getReport(in fromDate:date, in toDate:date, in reportType:String, out report:report)

Function: Generates report depending on the inputs received from the user.
Precondition: None.
Postcondition: A report is generated in the specified report format and in between the mentioned period.

Input Parameters: fromDate- the date from which the system should start looking for transaction to generate reports, toDate- the date until which the system should look for, reportType- mentiones the kind of report the manager is looking for ex. revenue, room occupancy .
Output Parameters: report- the document displaying the information the user is looking for .

6.

Information Hiding Class : Projection Algorithm Logic
Information Hidden :Encapsulates Algorithmic business rules which can predict the future occupancy and revenue.
Class Structuring Criterion :Business Logic Class
Assumptions : None.
Anticipated Changes : None.
Superclass : None.
Inherited Attributes : None.
Inherited Operations : None.
Attributes : None.
Operations Provided :

1. getProjection(in fromDate:date,in toDate:date, out projections:report)

Function: Generates report depending on the calculations of the algorithm .
Precondition: None.
Postcondition: A report is generated in the specified report format and in between the mentioned period.
Input Parameters: fromDate- the date from which the system should compute the algorithm to generate reports, toDate- the date until which the system should compute the algorithm for generating reports, reportType- mentions the kind of report the manager is looking for ex. revenue, room occupancy .
Output Parameters: report- the document displaying the information the user is looking for .

Data Abstraction Class Interface Specifications

1.

Information Hiding Class : ReservationTransaction
Information Hidden : Encapsulates reservation attributes and their current values.
Class Structuring Criterion : Data Abstraction Class
Assumptions : None.
Anticipated Changes : None.
Superclass : None.
Inherited Operations : None.
Attributes : timeStamp:Date , sessionTransactionID:String, hotelName:String, reservationStartDate:date,reservationEndDate:date,type:String
Operations :

1. createReservation(in customerInput, out reservationTranscation)

Function: Creates a new reservation transaction.
Precondition: None.
Postcondition: A new reservation entity is created.
Input Parameters: customerInput- Details entered by customer from keyboard.
Output Parameters: reservationTransaction- New reservation entity created.
Operations Used:None

2. changeReservation(in customerInput,out reservationTranscation)

Function: Changes the reservation transaction, by using the new customer input.
Precondition: None.
Postcondition: A new reservation entity is created.
Input Parameters: customerInput- Details entered by customer from keyboard.
Output Parameters: reservationTransaction- New reservation entity created.
Operations Used:None

3. cancelReservation(in customerInput,out reservationTranscation)

Function: Cancels the reservation transaction.
Precondition: None.
Postcondition: Reservation is canceled and the entity is deleted.

Input Parameters: customerInput- Details entered by customer from keyboard.
Output Parameters: reservationTransaction- New reservation entity created.
Operations Used:None

4. checkIn(in customerInput,out reservationTranscation)

Function: Checks in a customer into the hotel.
Precondition: None.
Postcondition: The room has been checked in and the reservation transaction is updated.
Input Parameters: customerInput- Details entered by customer from keyboard.
Output Parameters: reservationTransaction- New reservation entity created.
Operations Used:None

5. checkOut(in customerInput,out reservationTranscation)

Function: Checks out a customer out of the hotel.
Precondition: None.
Postcondition: The room has been checked out and the reservation transaction is updated.
Input Parameters: customerInput- Details entered by customer from keyboard.
Output Parameters: reservationTransaction- New reservation entity created.
Operations Used:None

6. read(out reservationTransaction)

Function: Reads the reservation entity that is currently residing on the client machine.
Precondition: None.
Postcondition: The reservationTransactiond details are displayed.
Input Parameters: None.
Output Parameters: reservationTransaction- Reservation Entity displayed.
Operations Used:None

TIS_TBS- Customer Interaction

Name: CustomerInteraction
Information hidden: Details for processing input from customer and output processed information to the customer using standard I/O.
Structuring criteria: role criterion: user interaction; concurrency criterion: event driven
Assumptions: only one Customer Interaction is handled at one time.
Anticipated Changes: Possible additional selections and output will need to be provided.
Task interface:

Task inputs:
Event input: Customer selection interrupts to indicate that a selection has been made.
External input: CustomerInput through the standard input device such as keyboard (Customer info, room info).
Asynchronous message communication:

- Display prompts

Task outputs:
External output: CustomerOutput
Asynchronous message communication:

- DisplayPrompt

Passive objects accessed: ReservationTransaction
Errors detected: Invalid date interval

loop
-- Wait for external interrupt from customer client
wait (selectionEvent);
read reservation information and customer choice;
if customerChoice = make reservation
then --
-- Write Reservation data into the ReservationTransaction object;
ReservationTransaction.createReservation(customerInput)
elseif customerChoice = change reservation
then
-- Write Reservation data into the ReservationTransaction object;
ReservationTransaction.changeReservation(customerInput)
elseif customerChoice = cancel reservation
then
-- Write Reservation data into the ReservationTransaction object;
ReservationTransaction.cancelReservation(customerInput)

-- Send transaction object created message to CustomerClientControl

```
send(CustomerClientControlMessageQ, reservationTransaction, customerChoice)
--wait for message from CustomerClientControl
receive(CustomerInteractionMessageBuffer, displayPrompt)
--send message to Clerk
send(customerOutput)
end loop;
```

TIS_TBS - Customer Client Control

Name: CustomerClientControl
Information hidden: Details of how CustomerClientControl handles events.
Structuring criteria: role criterion: control; concurrency criterion: demand driven
Assumptions:
Anticipated Changes: Possible addition of further Customer selections (Options).
Task interface:

Task inputs:
Asynchronous message communication:

- Customer events

Task outputs:
External output: Client Transactions

Synchronous message communication:
Message:

- ValidateCreditCard

Input parameters: cardNumber
Reply: response

Asynchronous message communication:

- Display Prompts

Errors detected: Invalid cardNumber, invalid events

loop
-- Messages from all senders are received on Message Queue
receive(CustomerClientControlMessageQ, reservationTransaction, customerChoice)
-- Extract the event name and any message parameters
-- Given the incoming event, lookup state transition table;
-- change state if required; return action to be performed;
newEvent = message.event
outstandingEvent = true;
while outstandingEvent **do**
CustomerClientStateMachine.processEvent (**in** newEvent, **out** action);
outstandingEvent = false;
-- Execute action(s) as given on CustomerClientControl statechart

case action **of**

Create Reservation: --create a reservation request to the Hotel Service
send(promptMessageQueue, displayWait);
send(Hotel Service, **in** clientTransaction, **out** hotelServiceResponse);
newEvent = hotelServiceResponse; outstandingEvent = true;

Change Reservation: --change a reservation request to the Hotel Service
send(promptMessageQueue, displayWait);
send(Hotel Service, **in** clientTransaction, **out** hotelServiceResponse);
newEvent = hotelServiceResponse; outstandingEvent = true;

Cancel Reservation: --cancel a reservation request to the Hotel Service
send(promptMessageQueue, displayWait);
send(Hotel Service, **in** clientTransaction, **out** hotelServiceResponse);
newEvent = hotelServiceResponse; outstandingEvent = true;

Display Prompt: --display information or updates to the Clerk
send(clerkInteractionMessageBuffer, displayPrompt);
outstandingEvent = true;

send(validateCreditCard, in cardNumber, out response);
newEvent = response; outstandingEvent = true;

end case;
end while;
end loop;

TIS_TBS - Customer Client System

Name: CustomerClient
Information hidden: Details of how CustomerClient processes customer selections and hotel service responses
Structuring criteria: role criterion: client; concurrency criterion: demand driven
Assumptions: only one request is made at one time.
Anticipated Changes: Possible addition of further requests.
Task interface:
Task inputs:
Synchronous message communication with reply:
Message replies as described below
Task outputs:
Synchronous message communication with reply:

Messages:

- makeReservation
  Input parameters: reservationTransaction
  Reply: confirmationNumber
- getReservation
  Input parameters: confirmationNumber
  Reply: reservationTransaction
- changeReservation
  Input parameters: changeTransaction
  Reply: available
- updateReservation
  Input parameters: updateTransaction
- cancelReservation
  Input parameters: cancelTransaction
- checkAvailability
  Input parameters: avaialabilityTransaction
  Reply: available, cost
- checkCheckOutDate
  Input parameters: checkOutDate
  Reply: available

Errors detected: Unrecognized response

loop
receive (HotelService, Response) from Hotel Service Response Queue;
Extract response name and response parameters from response;
case Response of
CheckAvailabilityResponse:

-- Room(s) Available
display (CustomerInteraction, roomsAvailable,totalCost);
-- Room(s) not Available
displayPrompt (CustomerInteraction, roomsNotAvailable);

case Response of
MakeReservationResponse:
-- Reservation Successful
display (CustomerInteraction, confirmationNumber);
-- Reservation not Successful
displayPrompt (CustomerInteraction, reservationNotsuccessful);

case Response of
CancelReservationResponse (non-guaranted or guaranted before deadline):
-- Valid Confirmation Number
display (CustomerInteraction, reservationCanceled);
-- Invalid Confirmation Number
prompt (CustomerInteraction, reservationNotFound);

case Response of
CancelReservationResponse (guaranted after deadline):
-- Valid Confirmation Number
display (CustomerInteraction, reservationCanceled, customerCharged);
-- Invalid Confirmation Number
prompt (CustomerInteraction, reservationNotFound);

case Response of
ChangeReservationResponse:
-- Valid Confirmation Number
display (CustomerInteraction, reservationInformation);
-- Invalid Confirmation Number
prompt (CustomerInteraction, reservationNotFound);

case Response of
MakeReservationResponse:
-- Valid Confirmation Number, Change Reservation Successful
display (CustomerInteraction, reservationChangedInformation);
-- Valid Confirmation Number, Change Reservation Not Successful
prompt (CustomerInteraction, reservationNotSuccessful);
-- Invalid Confirmation Number, Change Reservation Not Successful
prompt (CustomerInteraction, reservationNotFound);

```
end case;
end loop;
```

TIS_TBS - Clerk Interaction

Name: ClerkInteraction
Information hidden: Details for processing input from clerk and output processed information to the clerk using standard I/O.
Structuring criteria: role criterion: user interaction; concurrency criterion: event driven
Assumptions: one Clerk Interaction is handled at a time.
Anticipated Changes:
Task interface:

Task inputs:
Event input: Clerk selection interrupts to indicate that a selection has been made.
External input: ClerkInput through the standard input device such as keyboard (Customer info, room info).
Asynchronous message communication:

Task outputs:
External output: ClerkOutput
Asynchronous message communication:

- DisplayPrompt

Passive objects accessed: ReservationTransaction
Errors detected: Invalid date interval

loop
-- Wait for external interrupt from clerk client
wait (selectionEvent);
read reservation information and clerk choice;
if clerkChoice = make reservation
then --
-- Write Reservation data into the ReservationTransaction object;
ReservationTransaction.createReservation(customerInput)
elseif clerkChoice = change reservation
then
-- Write Reservation data into the ReservationTransaction object;
ReservationTransaction.changeReservation(customerInput)
elseif clerkChoice = cancel reservation
then
-- Write Reservation data into the ReservationTransaction object;
ReservationTransaction.cancelReservation(customerInput)
elseif clerkChoice = check in
then
-- Write Reservation data into the ReservationTransaction object;

ReservationTransaction.checkInReservation(customerInput)
elseif clerkChoice = check out
then
-- Write Reservation data into the ReservationTransaction object;
ReservationTransaction.checkOutReservation(customerInput)
else -- selection was not recognized so display “not implemented”;
Exit;
end if;
-- Send transaction object created message to ClerkClientControl
send(ClerkClientControlMessageQ, reservationTransaction, clerkChoice)
--wait for message from ClerkClientControl
receive(clerkInteractionMessageBuffer, displayPrompt)
--send message to Clerk
send(clerkOutput)
end loop;

TIS_TBS - Clerk Client Control

Name: ClerkClientControl
Information hidden: Details of how ClerkClientControl handles events.
Structuring criteria: role criterion: control; concurrency criterion: demand driven
Assumptions:
Anticipated Changes: Possible addition of further Clerk selections (Options).
Task interface:

Task inputs:
Asynchronous message communication:

- Clerk events

Task outputs:
External output: Clerk Transactions

Synchronous message communication:
Message:

- ValidateCreditCard

  Input parameters: cardNumber
  Reply: response

- Print Room Key
- Print room info

Asynchronous message communication:

- Display Prompts

Errors detected: Invalid cardNumber, invalid events

loop
-- Messages from all senders are received on Message Queue
receive(ClerkClientControlMessageQ, reservationTransaction, clerkChoice)
-- Extract the event name and any message parameters
-- Given the incoming event, lookup state transition table;
-- change state if required; return action to be performed;
newEvent = message.event
outstandingEvent = true;
while outstandingEvent **do**
ClerkClientStateMachine.processEvent (**in** newEvent, **out** action);
outstandingEvent = false;
-- Execute action(s) as given on ClerkClientControl statechart

case action **of**

Create Reservation: --create a reservation request to the Hotel Service
send(promptMessageQueue, displayWait);
send(Hotel Service, **in** clientTransaction, **out** hotelServiceResponse);
newEvent = hotelServiceResponse; outstandingEvent = true;

Change Reservation: --change a reservation request to the Hotel Service
send(promptMessageQueue, displayWait);
send(Hotel Service, **in** clientTransaction, **out** hotelServiceResponse);
newEvent = hotelServiceResponse; outstandingEvent = true;

Cancel Reservation: --cancel a reservation request to the Hotel Service
send(promptMessageQueue, displayWait);
send(Hotel Service, **in** clientTransaction, **out** hotelServiceResponse);
newEvent = hotelServiceResponse; outstandingEvent = true;

Check In: --Check In request to the Hotel Service
send(promptMessageQueue, displayWait);
send(Hotel Service, **in** clientTransaction, **out** hotelServiceResponse);
newEvent = hotelServiceResponse; outstandingEvent = true;

Check Out: --Check Out request to the Hotel Service
send(promptMessageQueue, displayWait);
send(Hotel Service, **in** clientTransaction, **out** hotelServiceResponse);
newEvent = hotelServiceResponse; outstandingEvent = true;

Print Room Info: --Print the room information
send(PrinterInterface);
outstandingEvent = true;

Print Room Key: --Print the room key
send(KeyPrinterInterface);
outstandingEvent = true;

Display Prompt: --display information or updates to the Clerk
send(clerkInteractionMessageBuffer, displayPrompt);
outstandingEvent = true;

Validate Credit Card: --validate the Customer Credit Card for the Reservation
send(validateCreditCard, in cardNumber, out response);
newEvent = response; outstandingEvent = true;

```
end case;
end while;
end loop;
```

TIS_TBS - Clerk Client System

Name: ClerkClient
Information hidden: Details of how ClerkClient processes clerk selections and hotel service responses
Structuring criteria: role criterion: client; concurrency criterion: demand driven
Assumptions: only one request is made at one time.
Anticipated Changes: Possible addition of further requests.
Task interface:
Task inputs:
Synchronous message communication with reply:
Message replies as described below
Task outputs:
Synchronous message communication with reply:

Messages:

- makeReservation
  Input parameters: reservationTransaction
  Reply: confirmationNumber
- getReservation
  Input parameters: confirmationNumber
  Reply: reservationTransaction
- changeReservation
  Input parameters: changeTransaction
  Reply: available
- updateReservation
  Input parameters: updateTransaction
- cancelReservation
  Input parameters: cancelTransaction
- checkAvailability
  Input parameters: avaialabilityTransaction
  Reply: available, cost
- checkCheckOutDate
  Input parameters: checkOutDate
  Reply: available
- assignRoom
  Input parameters: checkInTransaction
  Reply: roomAssigned
- getCharges
  Input parameters: checkOutTransaction
  Reply: charges
- chargeCreditCard
  Input parameters: checkOutTransaction
  Reply: confirmation
- completeCheckOut

Input parameters: checkOutTransaction
Reply: bill

Errors detected: Unrecognized response

loop
receive (HotelService, Response) from Hotel Service Response Queue;
Extract response name and response parameters from response;
case Response of
CheckAvailabilityResponse:
-- Room(s) Available
display (ClerkInteraction, roomsAvailable,totalCost);
-- Room(s) not Available
displayPrompt (CustomerInteraction, roomsNotAvailable);

case Response of
MakeReservationResponse:
-- Reservation Successful
display (ClerkInteraction, confirmationNumber);
-- Reservation not Successful
displayPrompt (CustomerInteraction, reservationNotsuccessful);

case Response of
CancelReservationResponse (non-guaranted or guaranted before deadline):
-- Valid Confirmation Number
display (ClerkInteraction, reservationCanceled);
-- Invalid Confirmation Number
prompt (ClerkInteraction, reservationNotFound);

case Response of
CancelReservationResponse (guaranted after deadline):
-- Valid Confirmation Number
display (ClerkInteraction, reservationCanceled, customerCharged);
-- Invalid Confirmation Number
prompt (ClerkInteraction, reservationNotFound);

case Response of
ChangeReservationResponse:
-- Valid Confirmation Number
display (ClerkInteraction, reservationInformation);

-- Invalid Confirmation Number
prompt (ClerkInteraction, reservationNotFound);

case Response of
MakeReservationResponse:
-- Valid Confirmation Number, Change Reservation Successful
display (ClerkInteraction, reservationChangedInformation);
-- Valid Confirmation Number, Change Reservation Not Successful
prompt (ClerkInteraction, reservationNotSuccessful);
-- Invalid Confirmation Number, Change Reservation Not Successful
prompt (ClerkInteraction, reservationNotFound);

case Response of
CheckInReservationResponse:
-- Valid Confirmation Number
display (ClerkInteraction, reservationInformation);
-- Invalid Confirmation Number
prompt (ClerkInteraction, reservationNotFound);

case Response of
CheckInReservationResponse:
-- Valid Credit Card
display (ClerkInteraction, assignedRoomNumber);
print(keyPrinterInterface,printKey);
print(printerInterface,roomInformation);
-- Invalid Credit Card
prompt (ClerkInteraction, invalidCreditCard);

case Response of
ChangeCheckOutDateResponse:
-- Rooms Available
display (ClerkInteraction, newCheckOutDate);
-- Rooms Not Available
prompt (ClerkInteraction, roomsNotAvailable);

case Response of
CheckOutReservationResponse:
-- Valid Room Number
display (ClerkInteraction, reservationInformation);
-- Invalid Room Number
prompt (ClerkInteraction, reservationNotFound);

```
case Response of
PaymentResponse:
-- Payment Successful
print (printerInterface, receipt);
-- Payment Not Successful, Payment Method Credit Card
prompt (ClerkInteraction, chargeNotAuthorized);
-- Payment Not Successful, Payment Cash
prompt (ClerkInteraction, InsufficientAmount);

end case;
end loop;
```

TIS_TBS - Manager Interaction

Name: ManagerInteraction
Information hidden: Details of retrieving reports from Hotel Service database and displaying information to the Manager.
Structuring criteria: role criterion: user interaction; concurrency criterion: event driven
Assumptions: only one Manager Interaction is handled at one time.
Anticipated Changes: Possible additional selections will need to be provided.
Task interface:
Task inputs:
Event input: Manager selection interrupt to indicate that a selection has been made.
External input: managerInput through the keypad (report type and date intervals).
Task outputs:
External output: prompt, report
Asynchronous message communication:
- displayPrompt
- displayReport
- promptDisplayed
- reportDisplayed.

Passive objects accessed: none
Errors detected: Invalid date interval

loop
-- Wait for external interrupt from manager client
wait (selectionEvent);
get manager selection;
if selection is recognized
then --
-- send manager selection message to ManagerClient Control;
send (ManagerClientControlMessageQ, selected);
-- Wait for message from ManagerClientControl;
receive (managerInteractionMessageBuffer, message);
if message = prompt
then
prompt for new date intervals;
-- Send prompt displayed message to ManagerClientControl;
Send (ManagerClientControlMessageQ, prompted);
elseif message = display
then
display reports;
-- Send report displayed message to ManagerClientControl;
send (ManagerClientControlMessageQ, reportDisplayed);

```
else error condition;
end if;
else -- selection was not recognized so display “not implemented”;
Exit;
end if;
end loop;
```

TIS_TBS - Manager Client Control

Name: ManagerClientControl
Information hidden: Details of how ManagerClientControl handles events.
Structuring criteria: role criterion: control; concurrency criterion: demand driven
Assumptions:
Anticipated Changes: Possible addition of further Manager options.
Task interface:
Task inputs:
Asynchronous message communication:
Messages:
. generateReport
Input parameters: startDate, endDate, reportType
External input: managerInput through the keypad (report type and date intervals).
Synchronous message communication without reply:
- displayPrompt
- displayReport
Task outputs:
External output: prompt, report
Asynchronous message communication:
- promptDisplayed
- reportDisplayed.
Errors detected: Invalid date interval

loop
-- Messages from all senders are received on Message Queue
Receive (ManagerClientControlMessageQ, message);
-- Extract the event name and any message parameters
-- Given the incoming event, lookup state transition table;
-- change state if required; return action to be performed;
newEvent = message.event
outstandingEvent = true;
while outstandingEvent **do**
ManagerClientStateMachine.processEvent (**in** newEvent, **out** action);
outstandingEvent = false;
-- Execute action(s) as given on ManagerClientControl statechart
case action **of**
GetReport:--get manager reports from Hotel Service;
send (Hotel Service, **in** getReport, **in** fromdate **, in** toDate **,out** getReportResponse);
newEvent = getReportResponse; outstandingEvent = true;

Display Prompt: -- Display selection menu for manager;

send (promptMessageQueue,displayPrompt);
ManagerClientTransaction.updateManagerSelection (valid);

Invalid Selection Action: -- Display InvalidSelection prompt;
send (promptMessageQueue, displayInvalidSelectionPrompt);
ManagerClientTransaction.updateSelectionStatus (invalid);
...
end case;

TIS_TBS - Manager Client System

Name: ManagerClient
Information hidden: Details of how ManagerClient processes manager selections and hotel service responses
Structuring criteria: role criterion: client; concurrency criterion: demand driven
Assumptions: only one request is made at one time.
Anticipated Changes: Possible addition of further requests.
Task interface:
Task inputs:
Synchronous message communication with reply:
.message replies from hotel service
. getReports response
Task outputs:
Synchronous message communication with reply:
Messages:
. getReport
Output parameters: startDate, endDate, reportType

Errors detected: Unrecognized response

loop
receive (HotelService, Response) from Hotel Service Response Queue;
Extract response name and response parameters from response;
case Response of
GetReportResponse:
-- Selection was valid
respond (ManagerInteraction, displayReport);
-- Selection was invalid
respond (ManagerInteraction, displayPrompt);
end case;
end loop;

TIS_TBS - Hotel Service

Name: HotelService
Information hidden: Details of how HotelService processes client Transactions.
Structuring criteria: role criterion: service; concurrency criterion: demand driven
Assumptions: Transactions are processed sequentially
Anticipated Changes: Possible addition of further transactions; possible change from sequential service to concurrent service processing.
Task interface:

Task inputs:
Synchronous message communication with reply:
Messages:

- checkAvailability

  Input parameters: reservationTranscation
  Reply: availabilityResponse

- makeReservation

  Input parameters: reservationTranscation
  Reply: reservationResponse

- getReservation

  Input parameters: reservationTranscation
  Reply: reservationResponse

- changeReservation

  Input parameters: reservationTranscation
  Reply: reservationResponse

- updateReservation

  Input parameters: reservationTranscation
  Reply: reservationResponse

- cancelReservation

  Input parameters: reservationTranscation
  Reply: reservationResponse

- assignRoom

Input parameters: reservationTranscation
Reply: CheckInCheckOutResponse

- getCharges

  Input parameters: reservationTranscation
  Reply: CheckInCheckOutResponse

- chargeCreditCard

  Input parameters: reservationTranscation
  Reply: CheckInCheckOutResponse

- completeCheckOut

  Input parameters: reservationTranscation
  Reply: CheckInCheckOutResponse

- getReport

  Input parameters: reservationTranscation
  Reply: ManagmentReportResponse

Task outputs:

Message replies as described previously.
Errors detected: Unrecognized response

loop
receive (ClientMessageQ, message) from Hotel Service Message Queue;
Extract message name and message parameters from the message;
Case Message of
checkAvailability:

-- Check that number rooms entered and dates are valid for reservation;
AvailabilityLogic.checkAvailability
(**in** availabilityTransaction, **out** available, out cost);
-- If rooms area available Response are available and return cost for the
Reservation;
-- otherwise availability response is unavailable;
reply(CustomerClient, ClerkClient, available, cost);

makeReservation:

```
        -- Check that number rooms entered and dates are valid for reservation;
        -- If all checks are successful, then create a reservation.
        ReservationLogic. createReservation
        (in reservationTransaction, out confirmationNumber);
        reply(CustomerClient, ClerkClient, confirmationNumber);
getReservation:
        -- Check confirmation number or the room number is present in
        resevationTranscation sent;
        ReservationLogic.getReservation
        (in confirmationNumber, out reservationTransaction);
        -- If Reservation Found then reservation response is the
        -- ReservationTransaction object;
        -- otherwise the response is reservation not found.
        reply(client, reservationTransaction);
changeReservation:
        -- Check if change reservation transaction contains all the required details to change
        reservation ;
        ReservationLogic.changeReservation;
        (in changeTranscation, out available);
        -- If reservation is available for the new changes changeReservation response will be
        available;
        --otherwise unavailable;
        reply(client, available);
updateReservation:
        -- Check if change reservation transaction contains all the required details;
        ReservationLogic.updateReservation
                (in updateTranscation);
cancelReservation:
        -- Check if change reservation transaction contains all the required details;
        ReservationLogic.cancelReservation(in cancelTranscation, out cancelResponse);
        -- On successful cancelation cancelReservation response done;
        reply(client, cancelResponse);
assignRoom:
        -- Check if change reservation transaction contains all the required details;
        CheckInCheckOut.assignRoom
        (in checkInTransaction, out roomAssigned);
        -- after assigning the room successfully send back a response roomAssigned;
        reply(client, roomAssigned);
getCharges:
```

-- Check if change checkOuttransaction contains all the required details;
CheckInCheckOut. getCharges
(in checkOutTransaction, out charges)
--System gets the charges and send responses back to the client with the charges;
reply(client, charges);

chargeCreditCard:
-- Check if checkOuttransaction contains all the required details;
CheckInCheckOut .chargeCreditCard
(in checkOutTransaction, out confirmation);
--Gets the credit card number from the checkOutTransaction and charges through
AuthorizationServerProxy;
-- If payment is done successfully chargeCreditCard sends a successful response;
-- Otherwise unsuccessful response;
reply (client, confirmation);

completeCheckOut:
-- Check if checkOuttransaction contains all the required details;
CheckInCheckOut .completeCheckOut
(in CheckOutTransaction, out bill);
-- Completes the Check out for Reservation in the checkoutTransaction;
-- send the bill for the reservation as response.
reply (Client, bill);

getReport:
-- Check if date and report type entered is valid;
CheckInCheckOut.getReport(**in** fromDate, **in** toDate, **in** reportType, **out** report);
-- This generates a report and send back the generated report as response;
reply(report);

TIS_TBS - Authorization Client Proxy

Name: AuthorizationClientProxy
Information hidden: Details of how AuthorizationClientProxy handles events.
Structuring criteria: role criterion: control; concurrency criterion: proxy
Assumptions:
Anticipated Changes:
Task interface:

Task inputs:
Synchronous message communication:
Message:

- ValidateCreditCard

  Input parameters: cardNumber
  Reply: response

Task outputs:
External output: AuthorizationResponse
Errors detected: Invalid cardNumber, invalid events

loop
receive (ClerkClientMessageQueue, message) from Authorization Proxy Queue;
Extract message name and message parameters from message;
case Message **of**
ValidateCard:
-- Get checked that card is valid and not expired
AuthorizationClientProxy.ValidateCard(**in** cardNumber, **out** authorizationResponse);
-- If successful, validation response is valid
reply(clerkClient, authorizationResponse);

case Message **of**
ChargeCreditCard:
-- Get amount to be charged, and get credit card charged
AuthorizationProxy.ChargeCreditCard(**in** cardNumber, **in** amount, **out** authorizationResponse);
-- If successful, charging is authorized and complete
reply(clerkClient, paymentResponse);

end case;
end loop;

TIS_TBS - Authorization Server Proxy

Name: AuthorizationServerProxy
Information hidden: Details of how AuthorizationServerProxy handles events.
Structuring criteria: role criterion: control; concurrency criterion: proxy
Assumptions:
Anticipated Changes:
Task interface:

Task inputs:
Synchronous message communication:
Message:

- ChargeCreditCard

  Input parameters: cardNumber
  Reply: response

- ValidateCreditCard

  Input parameters: cardNumber
  Reply: response

Task outputs:
External output: AuthorizationResponse
Errors detected: Invalid cardNumber, invalid events

loop
receive (ClerkClientMessageQueue, message) from Hotel Service Queue;
Extract message name and message parameters from message;
case Message **of**
ValidateCreditCard:
-- Get checked that card is valid and not expired
AuthorizationServerProxy.ValidateCreditCard(**in** cardNumber, **out** authorizationResponse);
-- If successful, validation response is valid
reply(hotelService, authorizationResponse);

case Message **of**
ChargeCreditCard:
-- Get amount to be charged, and get credit card charged
AuthorizationServerProxy.ChargeCreditCard(**in** cardNumber, **in** amount, **out** authorizationResponse);
-- If successful, charging is authorized and complete
reply(hotelService, paymentResponse);

```
end case;
end loop;
```

TIS_TBS - Key Printer Interface

Name: KeyPrinterInterface
Information hidden: Details of processing output to Key Printer
Structuring criteria: role criterion: output; concurrency criterion: demand driven
Assumptions: only one request is made at one time.
Anticipated Changes: Possible additional output to be printed.
Task interface:

Task inputs:
Event input: ClerkClientControl messages the KeyPrinterInterface to print room key.
Internal input: ClerkClient Control
Synchronous message communication without reply:

- Print Room Key

Synchronous message communication with reply:

- Read
  Reply: reservationTranscation

Task outputs:

Synchronous message communication without reply:

- KeyPrinter Output

Errors detected: Unrecognized response

loop
-- Wait for external interrupt from clerk client control
wait (selectionEvent);
Read reservation information from Reservation Transaction object
ReservationTransaction.read(out ReservationTransaction)
Send the information to the Printer
send(KeyPrinter, ReservationTransaction)
end loop;

TIS_TBS - Printer Interface

Name: PrinterInterface
Information hidden: Details of processing output to the Printer
Structuring criteria: role criterion: output; concurrency criterion: demand driven
Assumptions: only one request is made at one time.
Anticipated Changes:
Task interface:

Task inputs:
Event input: ClerkClientControl messages the PrinterInterface to print room info.
Internal input: ClerkClientControl
Synchronous message communication without reply:

- Print Room Info

Synchronous message communication with reply:

- Read
  Reply: reservationTranscation

Task outputs:

Synchronous message communication without reply:

- Printer Output

Passive objects accessed: ReservationTransaction

Errors detected: Unrecognized response

loop
-- Wait for external interrupt from clerk client control
wait (selectionEvent);
Read reservation information from Reservation Transaction object
ReservationTransaction.read(out ReservationTransaction)
Send the information to the Printer
send(Printer, ReservationTransaction)
end loop;