Software Design Description (SDD)

Table of Contents

1. Introduction

1.1 Purpose

The purpose of this document is to create a set of requirements that define essential development components for a management system used in tracking reservations and payments for a small bed-and-breakfast establishment. This document is specifically for use during implementation of the system.

1.2 Scope

The management software will be named Bed and Breakfast Management System, or BBMS. The BBMS is designed to act as an enabler for a small business. In this respect, the scope of the application and its intended functionality should be kept small and specific to the problem set. By tracing vacancies from a calendar view, determining vacancies exist and are either guaranteed until a specific date or first payment, and direct input of a guests' information, the effective management of the bed and breakfast can be made more efficient with a simple workflow.

1.3 Definitions

For the scope of this document, the following definitions and lexicon should be noted.

- *Workflow* – The sequence through a series of steps necessary to accomplish a task
- *Guarantee* – A specific contract between the owners of the establishment and the prospective guest that the room will be available.
- *Reservation* – A room that has been guaranteed to a guest for a specific date range.
- *Service* – Something that performs business logic.
- *User interface* – The graphical element that lets users interface with the system.
- *Domain Object* – Something that holds information for a specific function.
- *Concurrent Process* – Something that continuously runs.

2. References

References for this document include the following:

- IEEE Std. 1016-1998: IEEE Recommended Practice for Software Design Descriptions

3. Decomposition Description
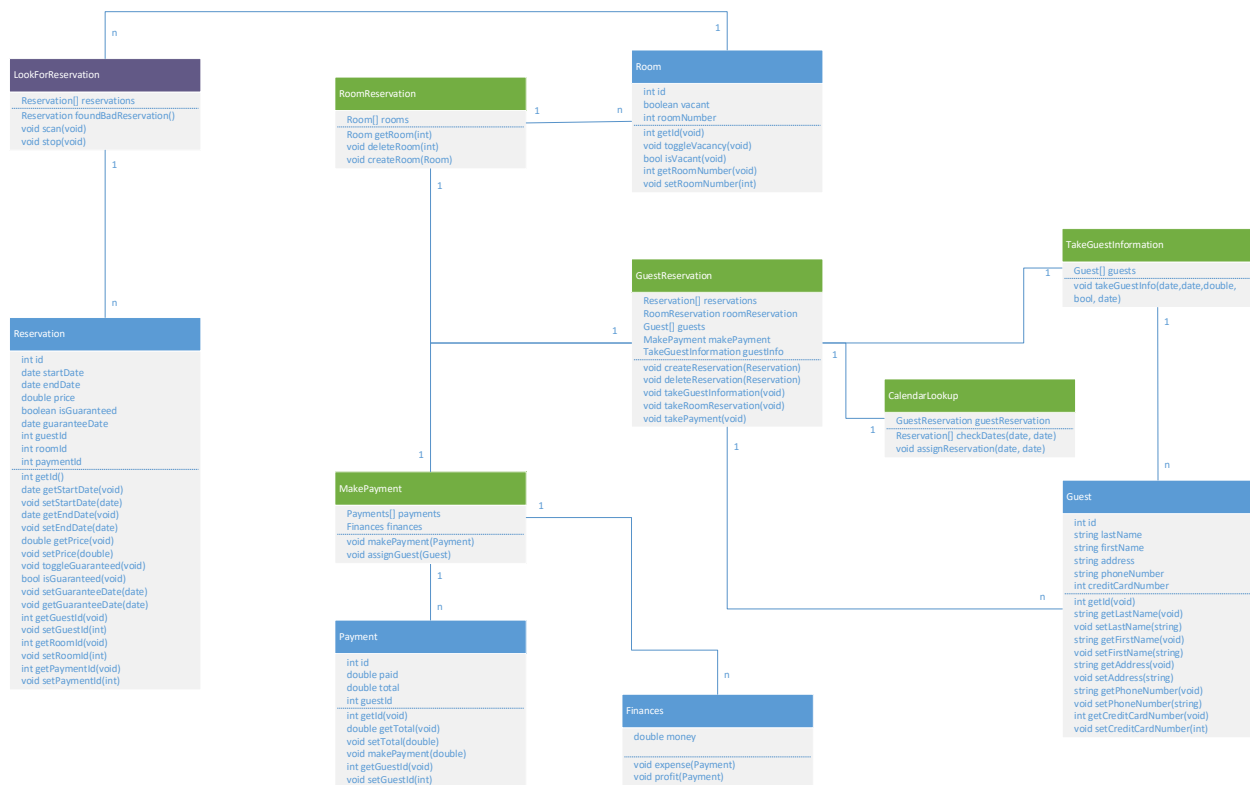
Figure 1. Overall Entity Relationship Design Diagram

Figure 1 gives the complete entity relationship diagram (ERD) in context for the entire BBMS. This ERD shows the relationship between the domain objects (in blue), their services (in green), and the concurrent process used to remove reservations with a past-due guarantee date.

In this section, we will describe the entities and their relationships. They are divided into modules, concurrent processes, and data.

3.1 Module Decomposition

The BBMS is composed of four different components, which provide specific functionality in the system. These are described at a high level in Figure 2. The overall design is composed of 4 primary services:

1. User Interface
2. Services
3. Domain Objects
4. Database

In the following representation and design specification, numbers 1-3 will be covered, as database persistence is covered via each appropriate service. The diagram includes a database reference generically to summarize persistence activities.

The complete architectural context diagram illustrates the human-computer interface for the user of the BBMS through the user interface, and interaction between domain objects and the user interface via the service layer. This overall design allows for data to be abstracted from the views through functional business logic in the services component, reducing overall complexity.
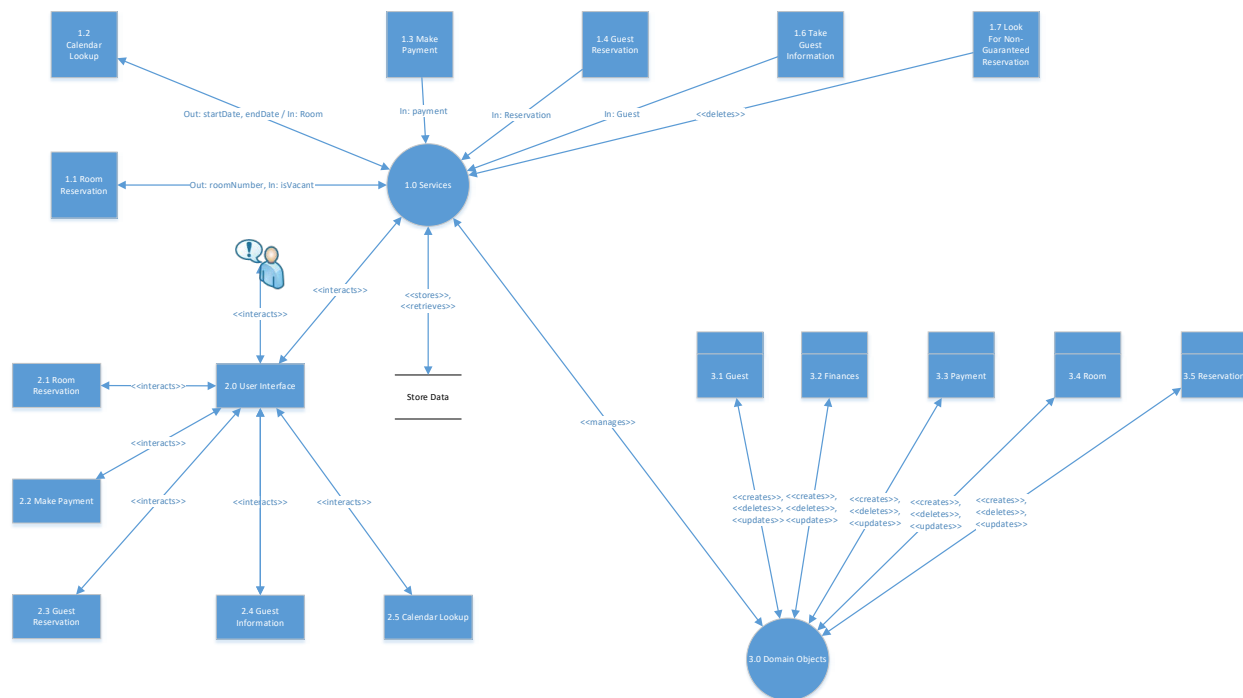

Figure 2. High Level Architectural Context Diagram

3.1.1 User Interface

This module is for users of the system, such as managers of a bed and breakfast. They are able to use the system via this interface exclusively. Access to this user interface does not require any external interface, other than via the keyboard and mouse, and is a single-user set of views.
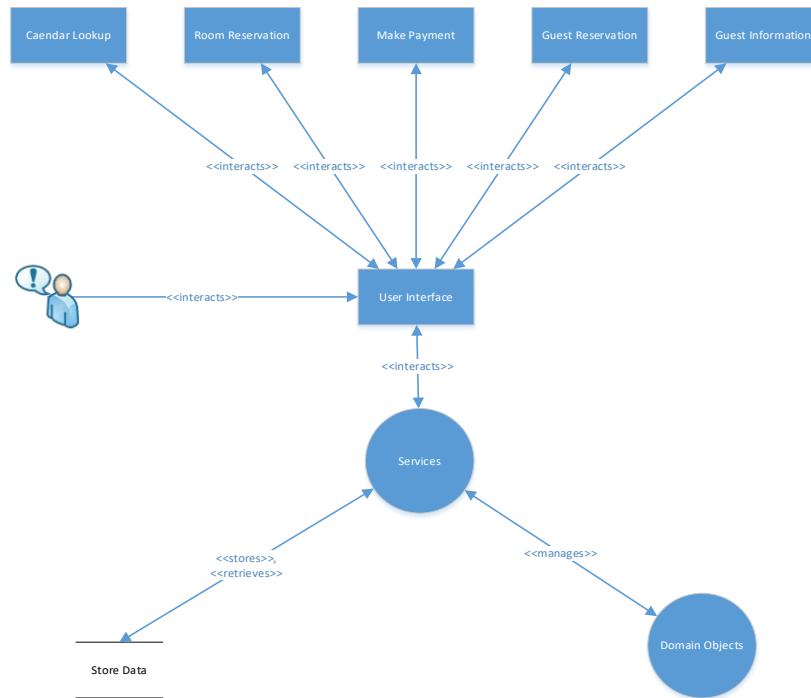

Figure 3. Architectural Context Diagram – User Interface

3.1.1.1 Calendar Lookup
Shows a calendar layout and information about room vacancies and room reservations. The specific design for the calendar lookup can be seen in section 5.1.

3.1.1.2 Room Reservation
Shows information about a room, and data entry elements to reserve a room. The specific design for the room reservation can be seen in section 5.2.

3.1.1.3 Make Payment
Shows information about payments specific to a guest. The specific design for making a payment can be seen in section 5.3.

3.1.1.4 Guest Reservation
Shows information about a guest's reservation in relation to a specific room and date. The specific design for the guest reservation can be seen in section 5.4.

3.1.1.5 Guest Information
Shows information about who a guest is, contact information, and payment information. The specific design for the guest information can be seen in section 5.5.

3.1.2 Services

This module is used to house the business logic for processes that the BBMS needs to conduct where domain objects are involved, such as looking up a calendar date or saving a payment. The service module provides output to the user interface, and input from the user interface is used to manage domain objects where appropriate. Persistence of the domain objects to the database is also managed by the service component.
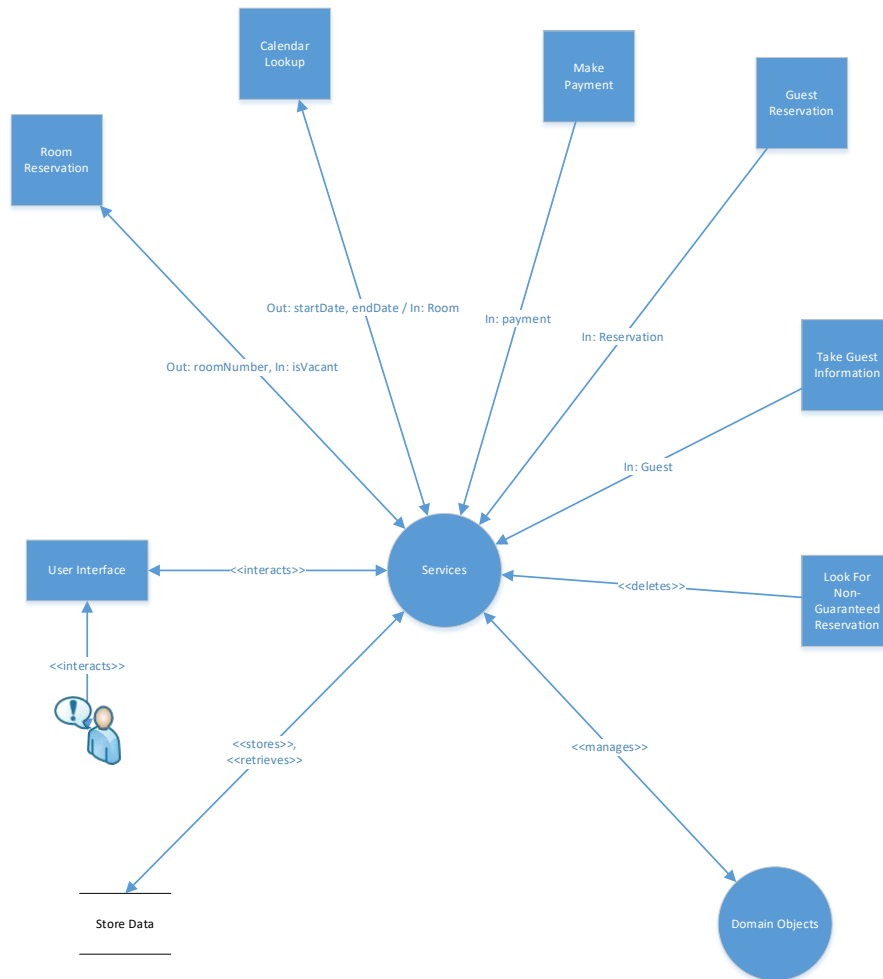

Figure 4. Architectural Context Diagram – Services

3.1.2.1 RoomReservation Entity

This service entity component handles the reservation of rooms, and management of the Room object.

3.1.2.2 CalendarLookup Entity

This service entity component handles the business logic to search for rooms in a date range, and return a room that is vacant.

3.1.1.3 MakePayment Entity

This service entity component handles the business logic for receiving a payment, and adjusting a Finances balance, as well as association to a room reservation and guest.

3.1.1.4 GuestReservation Entity

This service entity assigns guest information to a room that has been determined to be vacant, and holds other metadata, such as guarantee dates and associated payment status.

3.1.1.5 TakeGuestInformation Entity

This service entity assigns guest information directly, such as phone numbers, name, and address.

3.2 Concurrent Process Decomposition

This module is used to describe any service that acts independent of the user. This includes processes that create, manage, update, or destroy any domain object within the system.
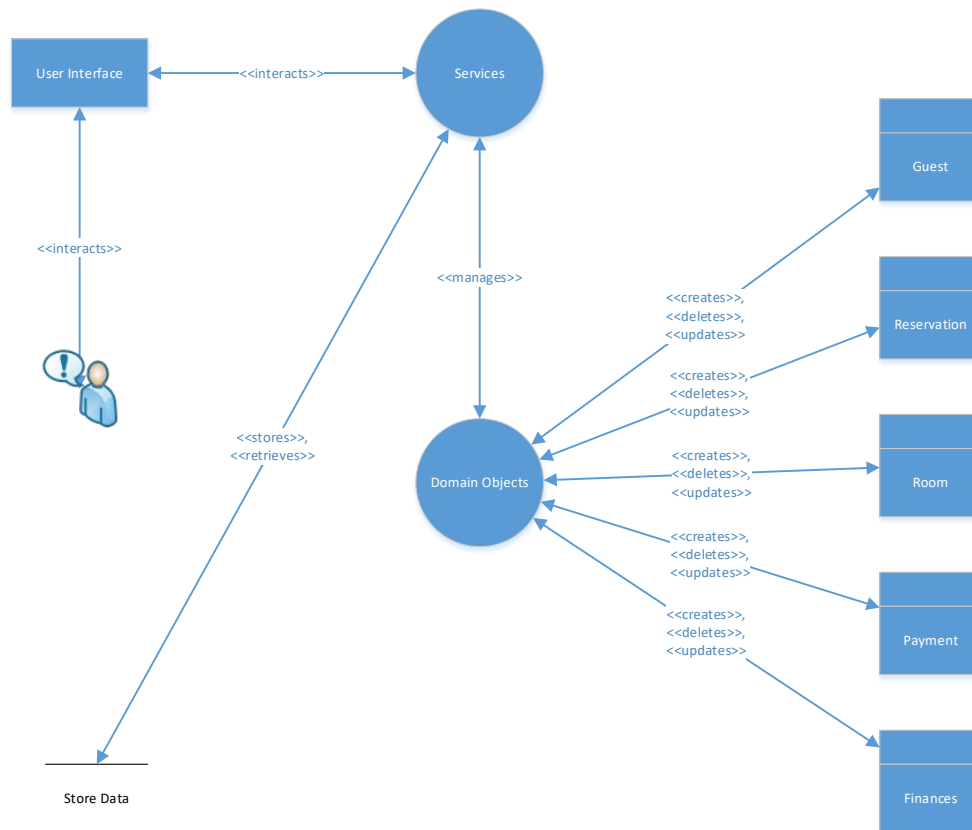
Figure 5. Architectural Context Diagram – Domain Objects

3.2.1 LookForReservation Entity

This service entity scans all Reservation objects looking for those with a guarantee date that matches the current date. If one is found the reservation is deleted to ensure vacancies for other guests.

3.3 Data Decomposition

This module covers all entities that hold data about a particular function in the BBMS system. These objects are referred to as domain objects.

3.3.1 Guest Entity

This entity holds all information related to a guest that has made a reservation and will be staying at the bed and breakfast. This information includes metadata such as first and last name, address, credit card number, and so on.

3.3.2 Room Entity

This entity holds all information about a room in the establishment. This information is used to correlate with reservations.

3.3.3 Reservation Entity

This entity holds all information about an active reservation, to include metadata such as start date, end date, guest making the reservation, and so forth. This entity is the general correlation point for other domain objects within the BBMS.

3.3.4 Payment Entity

This entity holds information about a payment that correlates directly from a guest to a room being reserved.

3.3.5 Finances Entity

This entity holds information about all payments made to the establishment through the BBMS.
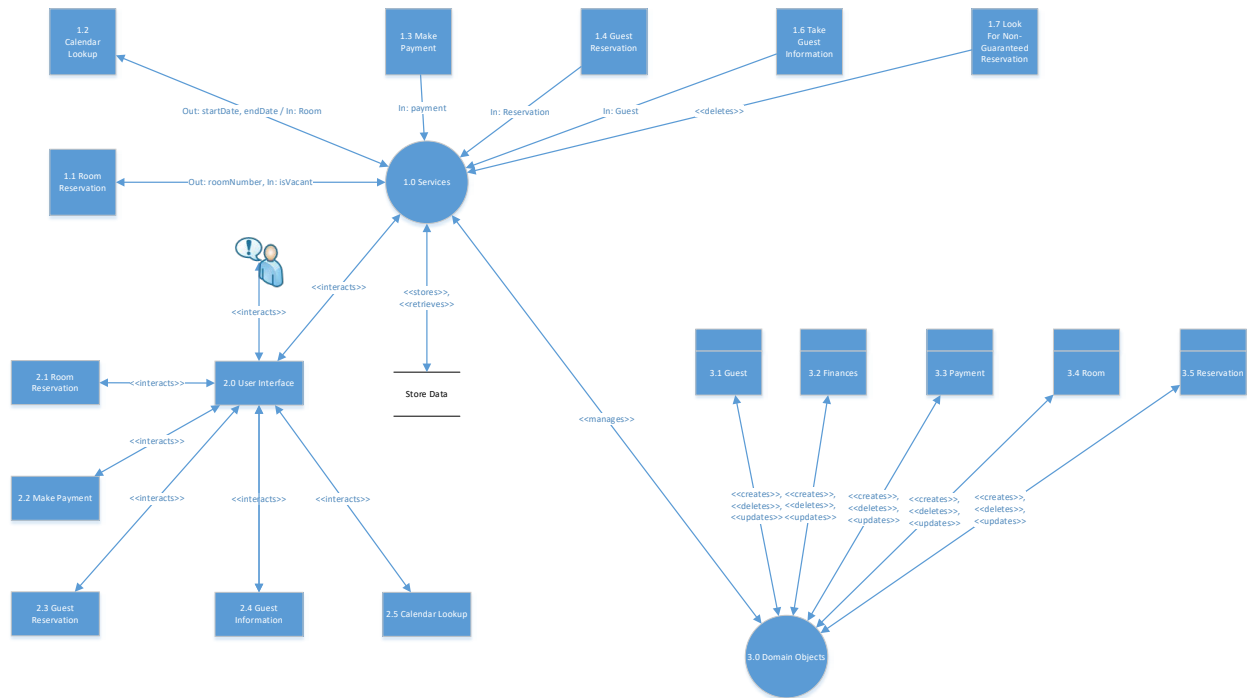
4. Dependency Description

Figure 6. Level 1 Data Flow Diagram

Service modules save, delete and update the domain object directly via the database module entries. The user interface module interacts with the service module. The service module controls and manages the domain model objects. This list of dependencies can be seen in a level 1 DFD in figure 6. Additionally, the list of dependencies for modules, services, and data are explained in detail below.

4.1 Inter-module Dependencies

For the BBMS software, there are 4 core modules. These include the user interface, services, domain objects, and database. For the scope of this document, the database module is called directly by the service modules as appropriate, and will not be extracted as a core module for description. In Figure 7, these are highlighted as dependencies, with the added interaction of the user to the user interface.
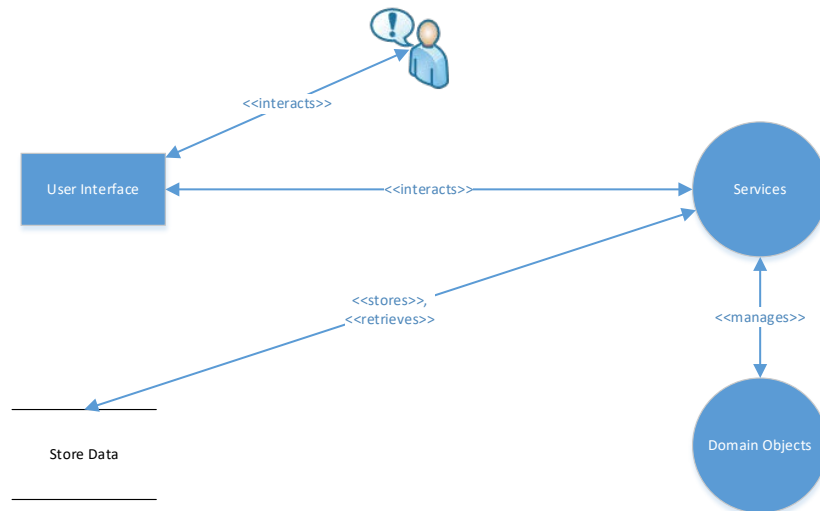

Figure 7. Inter-module Dependency Diagram

1. The User Interface module has a dependency on the service module. Since the user interface is tasked with displaying information from the BBMS, this dependency allows the business logic from the services module to be represented in the views directly at the user interface. In addition, the user has a dependency on the user interface to represent and interact with those views.

2. The Services module has dependencies on both the user interface module to represent a view, as well as the domain objects module to store information. Since business logic is represented at the service module, but information is stored within the domain objects, this dependency is critical to represent the state of actions and information within the system.

3. The Domain Objects module has a dependency on the services module to process information with business logic, as explained in dependency #2.

4.2 Inter-process Dependencies

There are six process dependencies within the BBMS, as represented in Figure 9. The use of a domain object within the inter-process dependency diagram is essential to describe the relationship between the RoomReservation service and a Room domain object, and how that relates to the LookForReservation concurrent service.
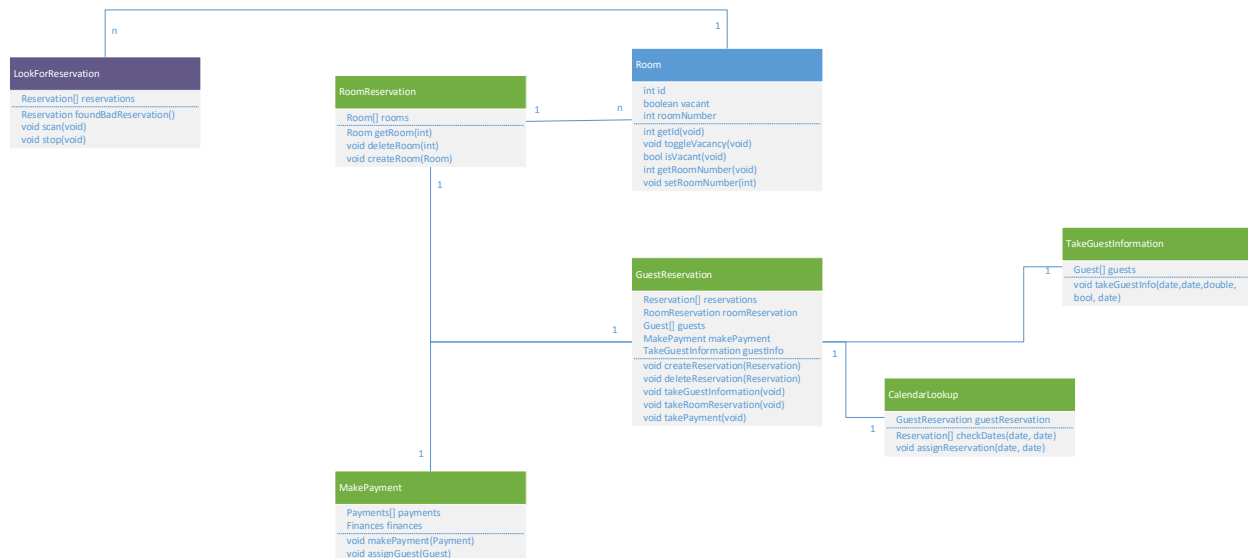

Figure 9. Inter-process Dependency Diagram

1. GuestReservation has a dependency on several other services, to include:

   a. RoomReservation – This defines the room information for a guest reservation.

   b. CalendarLookup – This relationship is defined by the link between a set of dates, and the subsequent guest reservation at the establishment.

   c. TakeGuestInformation – This dependency is used to drive information for a Guest object, which directly corresponds to a reservation.

   d. MakePayment – The relationship is defined as the payment by a guest which is linked to a room reservation

2. The LookForReservation process is loosely related to the RoomReservation service. Since this process is defined as a concurrent service, it scans reservations continuously to remove them in the event that the guarantee date has been exceeded.

4.3 Data Dependencies

Data dependencies in the BBMS are loosely linked through process dependencies to services. This is done to abstract business logic away from both the user interface module, as well as the domain object model. Due to this fact, there is no direct correlation between any of the domain objects. Instead, the processes/services share the dependencies to assist in business logic needs and abstraction between modules.
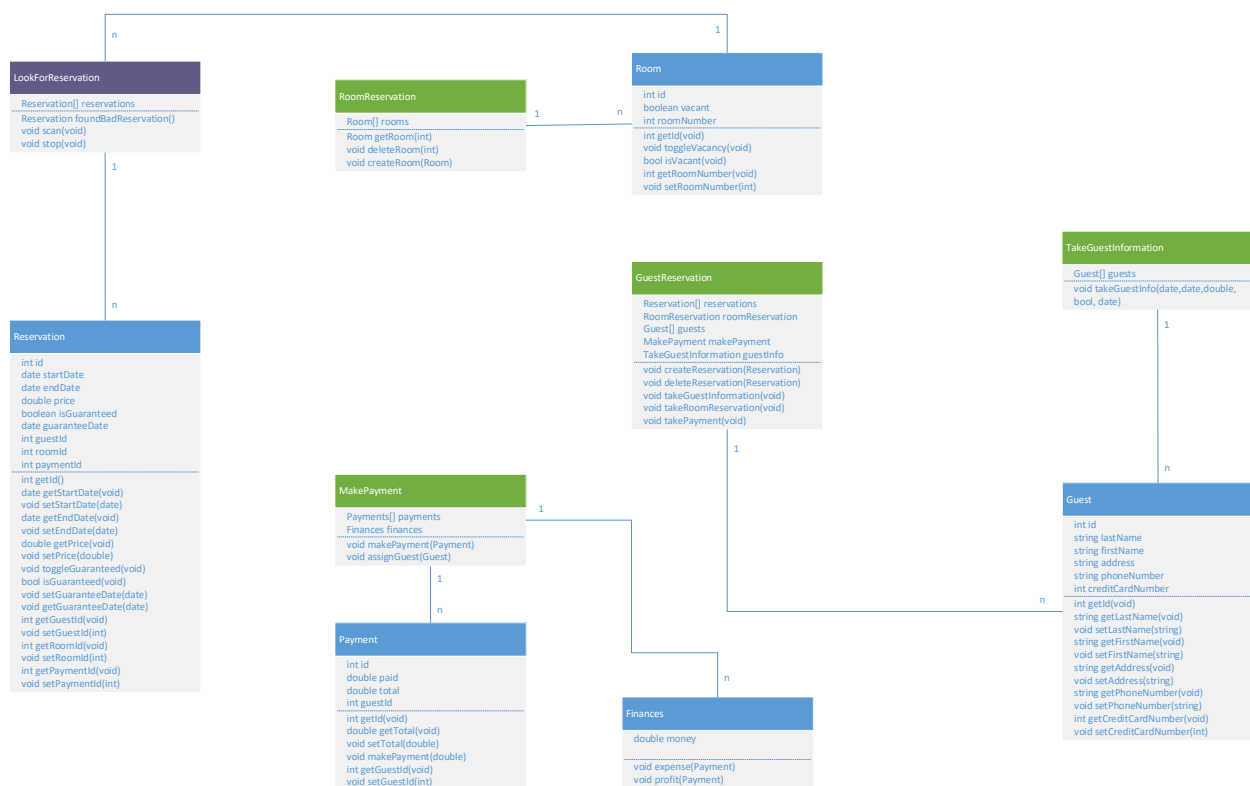

Figure 10. Data Dependency Diagram

5. Interface Description

5.1 User Interface Description

The user interface provides the user with the ability to interact with the BBMS system, and as a result create records, reservations, and associations that culminate in the successful stay of a user in the establishment. The figure below highlights the overall workflow from a human-computer interaction perspective, and how a daily user of the BBMS would interact with the system to create a reservation for a guest.
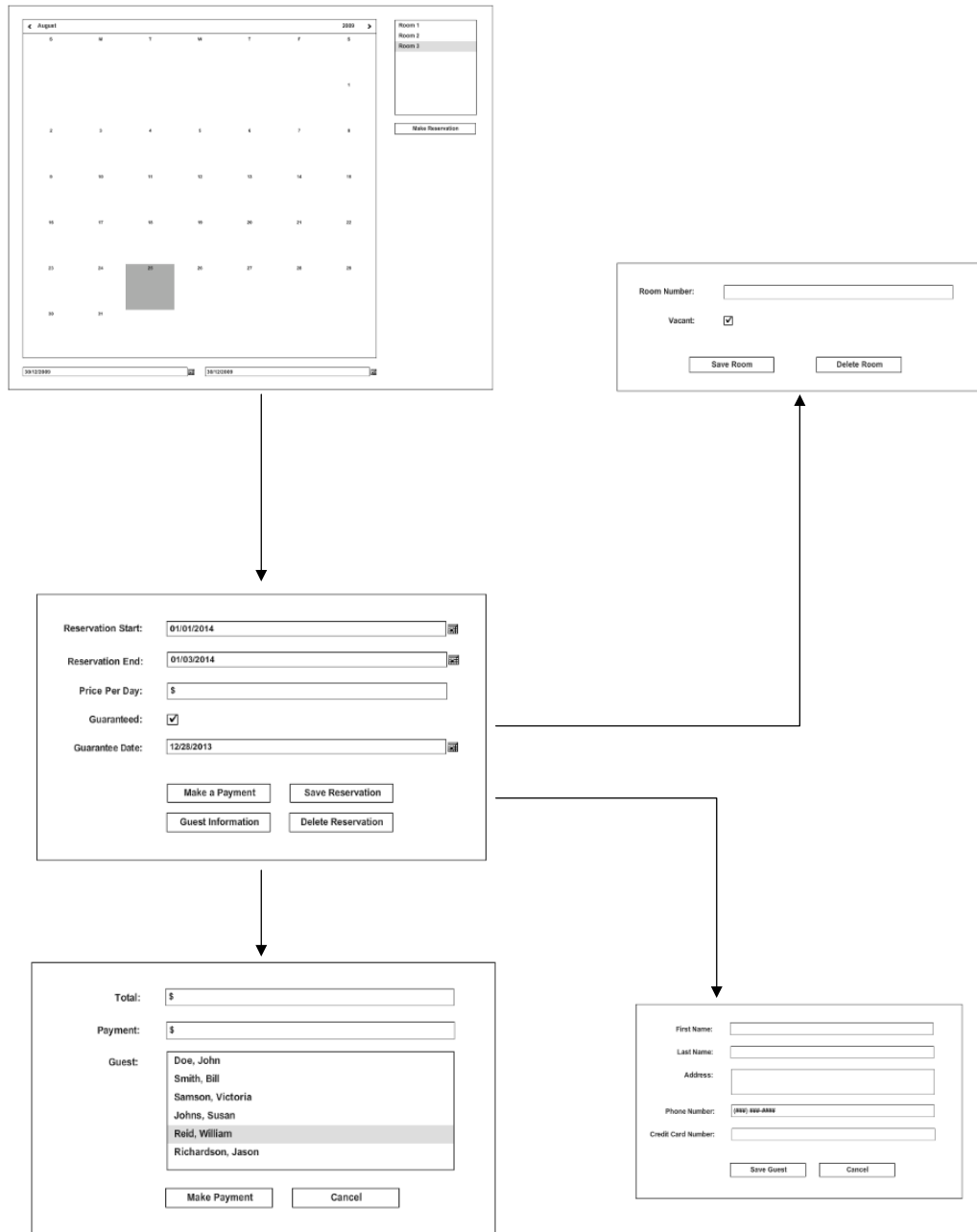

Figure 11. User Interface Workflow

5.1 Calendar Lookup

Calendar lookup is the primary interface for handling calls for the bread and breakfast manager. This screen provides three capabilities:

1. Calendar – This shows days of a month, or in a date range.
2. Room Vacancy Status – Shows room vacancies for a given date.
3. Make Reservation – Takes the date and room information for creation of a room reservation.

The calendar lookup user interface element provides the user with the ability to start the workflow when communicating with a guest. By finding a date range that is of interest to the guest, and verifying acceptable room vacancies within that range, the user of the BBMS can then start the process to guarantee a room for the guest, and create a reservation.
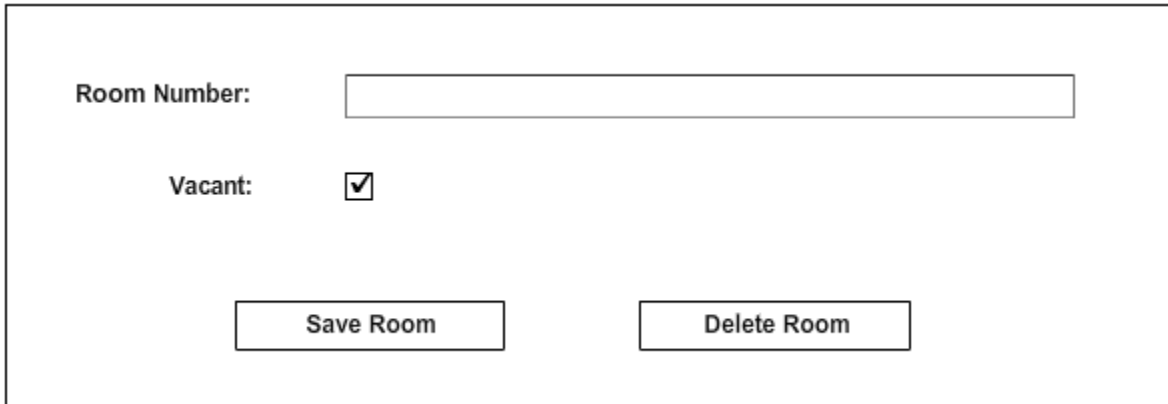
Figure 12. Calendar Lookup User Interface

5.2 Room Reservation

The room reservation form is designed to create and update vacancy statuses of rooms within the bed and breakfast establishment. This portion of the user interface is very simple, and only gives information specific to the room for a number and vacancy status. Two buttons are provided to save and delete the rooms based on the choices made by the user.
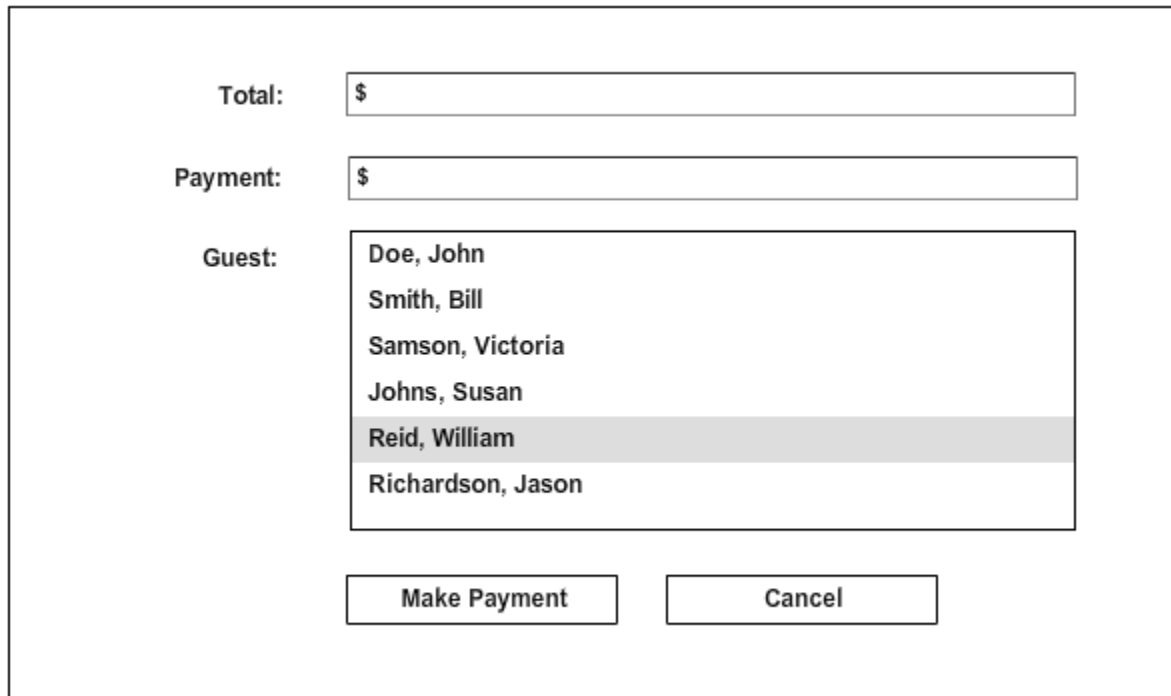
Figure 13. Room Reservation User Interface

5.3 Make Payment

The payment element of the user interface is designed to encapsulate all elements of a guest's payment towards their reservation. To accomplish this, the total amount of the stay is provided, along with a current payment. The guest list is provided to associate the payment correctly with the credit card provided by the guest when their information has been taken. Finally, the buttons allow for the user to either process the payment, or cancel the transaction completely.

Figure 14. Make Payment User Interface

5.4 Guest Reservation

The guest reservation allows a manager to input information specific to a guest's intended stay at the bread and breakfast, to include start and end dates, price per day, guaranteed status and date, as well as options to input information related to the reservation.
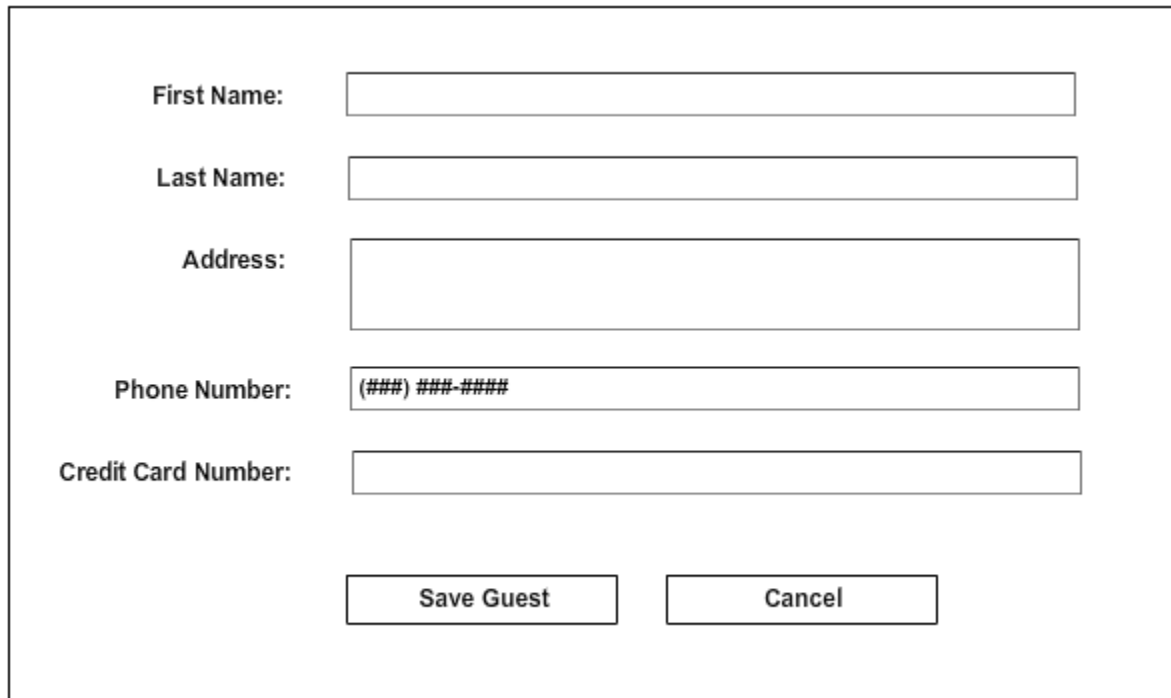
These buttons, shown at the bottom of the user interface, allow the manager to make a payment associated with the reservation, input information about the guest who will be staying, and options to save and delete the reservation as necessary.

Figure 15. Guest Reservation User Interface

5.5 Guest Information

In order to correctly guarantee and reserve a room for a guest, the user must be able to record their personal information. This element of the user interface provides data entry for their name, address, phone number, and associated credit card number for all payment transactions. Finally, the button options allow the user to either save the guests information, or cancel in case the phone call does not result in a successful stay by the guest (for example, not interested prior to the end of the reservation).

Figure 16. Guest Information User Interface

5.2 Service Interface Description

Service interfaces (domain object, service, and concurrent service) relationships can be found in the Entity Relationship Diagram in section 3 (Decomposition Description). Interface descriptions can be found with detailed descriptions and class models in section 6.1 (Module Detailed Design) and section 6.2 (Data Detailed Design)

6. Detailed Design

6.1 Module Detailed Design

6.1.1 RoomReservation Entity

The RoomReservation entity is a service that contains a collection of Room domain objects, serving as its business logic management. Figure 17 represents the class diagram for RoomReservation.

Three methods are present in this entity:

1. Room getRoom(int) – Returns the Room object that corresponds to the room number (int).

2. void deleteRoom(int) – Deletes the Room object from the collection that corresponds to the room number (int).

3. void createRoom(Room) – Adds a Room object to the managed collection.

One attribute is present in this entity:

1. Room[] rooms – Represents a collection of Room objects.

RoomReservation
Room[] rooms
Room getRoom(int) void deleteRoom(int) void createRoom(Room)
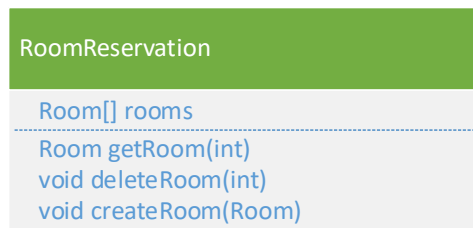
Figure 17. RoomReservation Class

6.1.2 CalendarLookup Entity

The CalendarLookup entity is a service that contains a single object, GuestReservation, and is used to define the relationship between a calendar date range, and an available reservation spot. Figure 18 represents the class diagram for CalendarLookup.

Two methods are present in this entity:

1. Reservation[] checkDates(date, date) – Returns a Reservation object that has been validated between the start date and the end date.

2. void assignReservation(date, date) – Assigns a reservation block between a start date and an end date in the calendar days.

One attribute is present in this entity:

1. GuestReservation guestReservation – This is an instance of the GuestReservation class, which is used to set up information about the guest, room, and payment for the reservation.
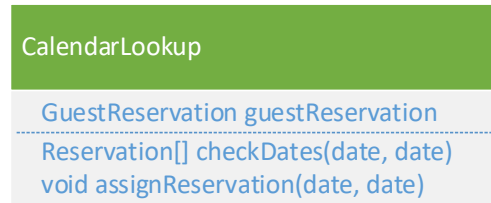
Figure 18. CalendarLookup Class

6.1.3 MakePayment Entity

The MakePayment entity is a services that contains a collection of Payment objects, a Finances object for total money tracking, and methods to make a payment and assign a guest. Figure 19 represents the class diagram for MakePayment.

Two methods are present in this entity:

1. void makePayment(Payment) – Takes a payment object and parses it to add to the total financial value of the establishment.

2. void assignGuest(Guest) – Takes a guest object and assigns it to the payment in relationship to the reservation.

Two attributes are present in this entry:

1. Payments[] payments – Represents a collection of payment objects which have been made and processes.

2. Finances finances – Represents the utility to track all financial dollar amounts for the establishment.
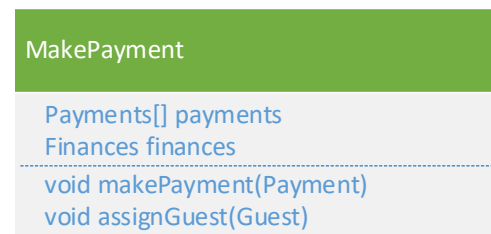
Figure 19. MakePayment Class

6.1.4 GuestReservation Entity

The GuestReservation entity is a service that represents the bulk of the business logic for the BBMS. This service contains instances of objects which manage most of the domain objects within the system. Figure 20 represents the class diagram for GuestReservation.

Five methods are present in this entity:

1. void createReservation(Reservation) – Takes a Reservation object and adds it to the overall collection.

2. void deleteReservation(Reservation) – Removes the Reservation object from the managed collection.

3. void takeGuestInformation(void) – Executes the service management for the TakeGuestInformation service entity.

4. void takeRoomReservation(void) - Executes the service management for the RoomReservation service entity.

5. void takePayment(void) – Executes the service management for the MakePayment service entity.

Five attributes are present in this entry:

1. Reservation[] reservation – Managed collection of Reservation domain objects.

2. RoomReservation roomReservation – Instance of the RoomReservation service entity.

3. Guest[] guests – Managed collection of Guest domain objects.

4. MakePayment makePayment – Instance of the MakePayment service entity.

5. TakeGuestInformation guestInfo – Instance of the TakeGuestInformation service entity.
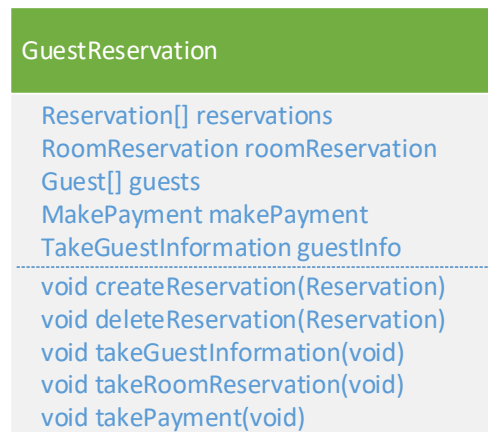
Figure 20. GuestReservation Class

6.1.5 TakeGuestInformation Entity

The TakeGuestInformation entity is a service that deals with information about a specific guest. This service retrieves information, and creates a Guest domain object from its result. Figure 21 represents the class diagram for TakeGuestInformation.

One method is present in this entity:

1. void takeGuestInfo(date,date,double,bool,date) – Takes relevant guest information, such as start date, end date, price, guarantee, and guarantee date, and creates a Guest object.

One attribute is present in this entity:

1. Guest[] guests – A managed collection of Guest domain objects.

TakeGuestInformation
Guest[] guests
void takeGuestInfo(date,date,double, bool, date)
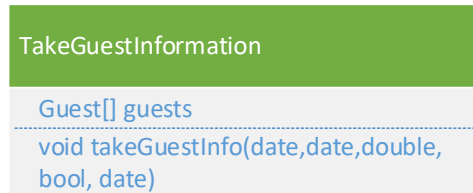
Figure 21. TakeGuestInformation Class

6.1.6 LookForReservation Entity

The LookForReservation entity is a concurrent service that is used to scan the collection of Reservations, and remove those with expired guaranteed dates. This service is different from the others in that it is constantly scanning the collection, and runs in the background without other interaction. Figure 22 represents the class diagram for LookForReservation.

Three methods are present in this entity:

1. Reservation foundBadReservation(void) – Polling function call that returns a bad Reservation with an expired guarantee date.

2. void scan(void) – Starts the scanning/polling of all Reservation objects.

3. void stop(void) – Stops the scanning/polling of all Reservation objects.

One attribute is present in this entity:

1. Reservation[] reservations – Managed collection of Reservation domain objects.

LookForReservation
Reservation[] reservations
Reservation foundBadReservation() void scan(void) void stop(void)

Figure 22. LookForReservation Class

6.2 Data Detailed Design

6.2.1 Guest Entity

The Guest entity is a domain object that is used to represent information about a guest. This domain object stores information about a guest, and is primarily composed of attributes and accessor methods. Figure 23 represents the class diagram for Guest.

Twelve methods are present in this entity:

1. int getId(void) – Retrieves the unique identifier for a Guest object.

2. string getLastName(void) – Retrieves the last name of a guest.

3. void setLastName(string) – Sets the guest's last name based on the passed variable.

4. string getFirstName(void) – Retrieves the first name of a guest.

5. void setFirstName(string) – Sets the guest's first name based on the passed variable.

6. string getAddress(void) – Retrieves the address of a guest.

7. void setAddress(string) – Sets the guest's address based on the passed variable.

8. string getPhoneNumber(void) – Retrieves a string representation of a guest's phone number.

9. void setPhoneNumber(string) – Sets the guest's phone number based on the passed variable.

10. int getCreditCardNumber(void) – Retrieves the guest's credit card number.

11. void setCreditCardNumber(int) Sets the guest's credit card number based on the passed variable.

Six attributes are present in this entity:

1. int id – Unique identifier for a guest.

2. string lastName – Last name of a guest.

3. string firstName – First name of a guest.

4. string address – Address of a guest.

5. string phoneNumber – Phone number of a guest.

6. int creditCardNumber – Credit card number associated with a guest.

Guest
int id
string lastName
string firstName
string address
string phoneNumber
int creditCardNumber
int getId(void)
string getLastName(void)
void setLastName(string)
string getFirstName(void)
void setFirstName(string)
string getAddress(void)
void setAddress(string)
string getPhoneNumber(void)
void setPhoneNumber(string)
int getCreditCardNumber(void)
void setCreditCardNumber(int)

Figure 23. Guest Class

6.2.2 Payment Entity

The Payment entity is a domain object that is used to represent payment information for a reservation. This domain object stores information about a payment, and is primarily composed of attributes and accessor methods. Figure 24 represents the class diagram for Payment.

Six methods are present in this entity:

1. int getId(void) – Retrieves the unique identifier for a payment.

2. double getTotal(void) – Retrieves the total amount that the payment requires to be paid in full for a reservation.

3. void setTotal(double) – Sets the total amount that the payment requires to be paid in full for a reservation.

4. void makePayment(double) – The amount being paid toward the full amount.

5. int getGuestId(void) – Returns the unique identifier for the correlated guest.

6. void setGuestId(int) – Sets the unique identifier for the correlated guest.

Four attributes are present in this entity:

1. int id – Unique identifier for a payment.

2. double paid – Amount paid towards the total.

3. double total – Total amount required to be paid in full for a reservation.

4. int guestId – Unique identifier for the correlated guest.

Payment
int id
double paid
double total
int guestId
int getId(void)
double getTotal(void)
void setTotal(double)
void makePayment(double)
int getGuestId(void)
void setGuestId(int)
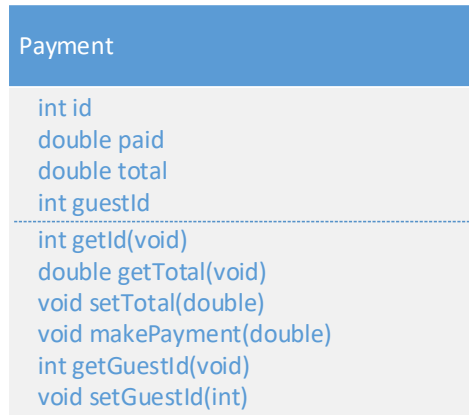
Figure 24. Payment Class

6.2.3 Reservation Entity

The Reservation entity is a domain object that is used to represent information for a reservation. This domain object stores information about a reservation, and is primarily composed of attributes and accessor methods. Figure 25 represents the class diagram for Reservation.

Seventeen methods are present in this entity:

1. int getId(void) – Retrieves the unique identifier for a reservation.

2. date getStartDate(void) – Retrieves date for the start of the reservation.

3. void setStartDate(date) – Sets the date for the start of the reservation.

4. date getEndDate(void) – Retrieves date for the end of the reservation.

5. void setStartDate(date) – Sets the date for the end of the reservation.

6. double getPrice(void) – Retrieves the price per day of a reservation.

7. void setPrice(double) – Sets the price per day of a reservation.

8. void toggleGuarantee(void) – Toggles the Boolean state of the reservation having a guarantee.

9. bool isGuaranteed(void) – Returns the Boolean state of the reservation having a guarantee.

10. date getGuaranteeDate(void) – Retrieves the date for the reservation guarantee expiry.

11. void setGuaranteeDate(date) – Sets the date for the reservation guarantee expiry.

12. int getGuestId(void) – Returns the unique identifier for the correlated guest.

13. void setGuestId(int) – Sets the unique identifier for the correlated guest.

14. int getRoomId(void) – Returns the unique identifier for the correlated room.

15. void setRoomId(int) – Sets the unique identifier for the correlated room.

16. int getPaymentId(void) – Returns the unique identifier for the correlated payment.

17. void setPaymentId(int) – Sets the unique identifier for the correlated payment.

Nine attributes are present in this entity:

1. int id – Unique identifier for a reservation.

2. date startDate – Start date for the reservation.

3. date endDate – End date for the reservation.

4. double price – The price per day of a reservation.

5. boolean isGuaranteed – Boolean status to determine if reservation has a guarantee.

6. int guestId – Unique identifier for a correlated guest.

7. int roomId – Unique identifier for a correlated room.

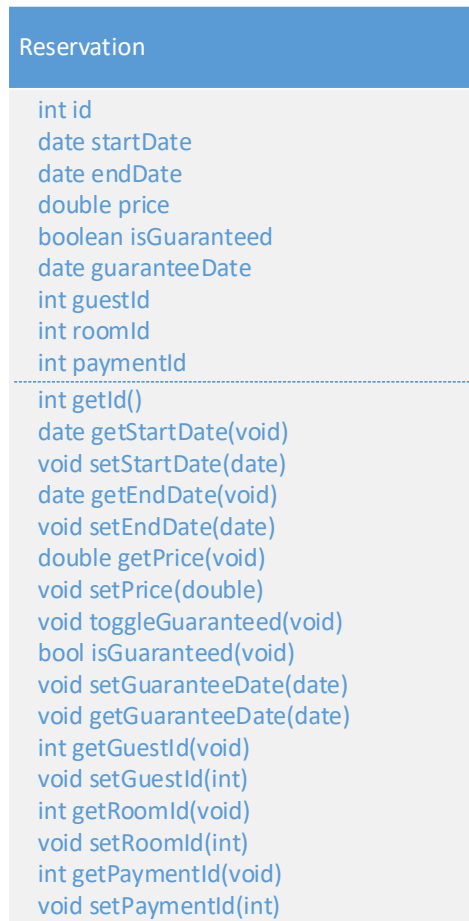8. int paymentId – Unique identifier for a correlated payment.

```
┌─────────────────────────────────┐
│ Reservation                     │
├─────────────────────────────────┤
│   int id                        │
│   date startDate                │
│   date endDate                  │
│   double price                  │
│   boolean isGuaranteed          │
│   date guaranteeDate            │
│   int guestId                   │
│   int roomId                    │
│   int paymentId                 │
├ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ┤
│   int getId()                   │
│   date getStartDate(void)       │
│   void setStartDate(date)       │
│   date getEndDate(void)         │
│   void setEndDate(date)         │
│   double getPrice(void)         │
│   void setPrice(double)         │
│   void toggleGuaranteed(void)   │
│   bool isGuaranteed(void)       │
│   void setGuaranteeDate(date)   │
│   void getGuaranteeDate(date)   │
│   int getGuestId(void)          │
│   void setGuestId(int)          │
│   int getRoomId(void)           │
│   void setRoomId(int)           │
│   int getPaymentId(void)        │
│   void setPaymentId(int)        │
└─────────────────────────────────┘
```

Figure 25. Payment Class

6.2.4 Room Entity

The Room entity is a domain object that is used to represent information for a room. This domain object stores information about a room, and is primarily composed of attributes and accessor methods. Figure 26 represents the class diagram for Room.

Five methods are present in this entity:

1. int getId(void) – Retrieves the unique identifier for a room.

2. void toggleVacancy(void) – Toggles the Boolean state of the room being vacant.

3. bool isVacant(void) – Returns the Boolean state of the room being vacant.

4. int getRoomNumber(void) – Retrieves the number of the associated room.

5. void setRoomNumber(int) – Retrieves the number of the associated room.

Three attributes are present in this entity:

1. int id – Unique identifier for a room.

2. boolean vacant – Boolean status to determine if room is vacant.

3. int roomNumber – Number associated with the room.

Room
int id boolean vacant int roomNumber
int getId(void) void toggleVacancy(void) bool isVacant(void) int getRoomNumber(void) void setRoomNumber(int)

Figure 26. Room Class

6.2.5 Finances Entity

The Finances entity is a domain object that is used to represent financial information for the totality of the establishment. This domain object stores information about a total finances, and is primarily composed of attributes and accessor methods. Figure 27 represents the class diagram for Finances.

Two methods are present in this entity:

1. void expense(Payment) – Calculates the expenditure of funds based on a Payment (negative).

2. void profit(Payment) – Calculates the addition of finds based on a Payment (positive).

Two attributes are present in this entity:

1. double money – Total money available to the establishment.

2. Payment[] payments – Managed collection of Payment domain objects.

Finances
double money Payment[] payments
void expense(Payment) void profit(Payment)

Figure 27. Finances Class