2014

Software Requirements Specification (SRS)

CMIS330

IAN NELSON

DEVELOPED FOR **JENEO K. DURHAM**

Table of Contents

1. Introduction

1.1 Purpose

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

1.1.1 Scope

The management software will be named Bed and Breakfast Management System, or BBMS.

1.1.1.1 Required Features

- Search for vacancies in a date range
- Input a guests' information
- Assign a guest to a reservation
- Assign a reservation to a date range
- Take payments for a reservation
- Show total payments for all reservations
- Guarantee a room until a specific date
- Check that payment is made by guarantee date
- Drop reservation if payment is not made by guarantee date
- Save a reservation
- Drop a reservation
- Save customer information
- Delete customer information

1.1.1.2 Out-of-Scope Functionality

- Act as an independent calendar application
- Act as a financial calculator for all aspects of the business
- Store information beyond guests, room vacancies, reservations, and payments

1.1.2 Objectives

This software 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.

1.2 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.

1.3 References

References for this document include the following:

- IEEE Std. 1233-98: IEEE Guide for Developing System Requirements Specification
- IEEE Std. 1058-1998: IEEE Standard for Software Project Management Plans
- IEEE Std. 1016-1998: IEEE Recommended Practice for Software Design Descriptions

1.4 Overview

This document contains the following artifacts:

- Architectural context diagram (ACD)
- Level 0 Data Flow Diagram (DFD)
- Level 1 Data Flow Diagram
- Entity Relationship Diagram
- Activity Diagram
- Use Case Diagram
- State Diagram

2. Overall Description

2.1 Product perspective

The BBMS is an isolated system making use of file-system storage and embedded logic. Due to the nature of its isolation, dependencies on other applications, infrastructure or components is extremely limited. This helps in the portability of the application, as well as speed of access to the user.

An overall architectural context diagram for the BBMS can be located in Appendix A.

2.1.1 System interfaces

Limited interfaces exist within the BBMS to reduce complexity and aid in the need for a simple and efficient reservation workflow.

2.1.1.1 User interfaces

The user interface is designed to be a simple and textual menu system making use of only the console. More complicated user interfaces, through products such as Microsoft Windows and X-Windows on Linux are unnecessary and over-complicate the requirements set for workflow. Additionally, the console-based menu system provides portability in Windows/DOS environments as well as Linux workstations should the need arise for a change in infrastructure. In addition, the use of a console-based menu provides increase speed in data entry when dealing with guests calling via telephone. Entry of a guest's information and relevant reservation data (approximately 50 words) should be accomplished in 90 seconds.

2.1.1.2 Hardware interface;

Due to the simple design of the software, the hardware interfaces are extremely limited by requirement. A basic Windows or Linux workstation is sufficient, provided a screen, keyboard and operating system exist and are configured.

2.1.1.3 Software interface

Software interfaces for the BBMS are not applicable (N/A).

2.1.1.4 Communications interfaces

Communication interfaces for the BBMS are not applicable (N/A).

2.1.2 Memory constraints

Hard drive storage requirements are minimal, and will require less than 10MB of available space. Non-persistent memory requirements are also minimal, and will require less than 50MB of RAM at run-time.

2.1.2 Operations

2.1.2.1 User-initiated operations

Most functionality of the BBMS will be achieved through interactive data entry within the menu system.

2.1.2.2 Periods of interactive operations and periods of unattended operations

The BBMS will conduct scheduled or polling tasks to verify reservation dates that are unpaid do not exceed the current date. This meets the only requirement for unattended operations.

2.1.2.3 Data processing support functions

Data processing functions are limited to the user-interactive data entry for reservation management.

2.1.2.4 Backup and recovery operations

Backup and recovery operations are limited to the needs of the customer, and are dependent on file-system-centric recovery tools. Since data for the BBMS is not accessing external databases, and is instead stored on the local file system, the user of the BBMS is responsible for any data loss risk.

2.1.3 Site adaptation requirements

Site adaptation requirements for the BBMS are not applicable (N/A).

2.2 Product functions

Product functionality can be decomposed into the following areas:

1. Calendar-Based Reservation Lookups
2. Reservation Entry
3. Reservation Removal
4. Non-paid Reservation Guarantee Checks
5. Guest Information Entry
6. Payment Processing

Additional information regarding the functional decomposition or requirements can be seen in Appendix B/C (Data flow Diagrams) or in Appendix D (Entity Relationship Diagram)

2.3 User characteristics

The primary users of the BBMS are managers of small bed and breakfast establishments. These managers, or owners, provide a level of service to customers based on a small number of rooms, and reservation of those rooms. In addition, basic financial information must also be accounted for to ensure the efficient monitoring of both profit and expense within the establishment. More specific information regarding user characteristics can be seen in Appendix F (Use Case / Scenario Diagrams).

2.4 Constraints

Based on the requirements for small businesses, and efficient workflow and financial tracking, constraints on the BBMS are minimal to none.

2.5 Assumptions and dependencies

In order for the BBMS to operate, these basic assumptions are made:

1. The business provides a workstation onto which the software can be installed
2. The business provides a telephone services to communicate with guests, and provide the entry point for data into the system.
3. The business provides adequate infrastructure (power, etc.) for the software to be used.

3. Specific Requirements

3.1 External interface requirements

These are requirements specific to the interaction between the software, user, and other components of the entirety of the system.

3.1.1 User interfaces

The software should have the following user interface options:

1. Main menu – Displays choices to see guests, calendar, reservations, and finances
2. Calendar – Shows a weekly calendar of reservations and vacancies, and allows the user to change the date range
3. Reservations – Shows a list of reservations, and the ability to add, change, or remove a reservation
4. Finances – Shows a list of payments, expenses, and the total net gain or loss of profit.

3.1.2 Hardware interfaces

As defined in 2.1.1.2, hardware interface requirements for the BBMS are not applicable (N/A).

3.1.3 Software interfaces

As defined in 2.1.1.3, software interface requirements for the BBMS are not applicable (N/A).

3.1.4 Communications interfaces

As defined in 2.1.1.4, communication interface requirements for the BBMS are not applicable (N/A).

3.2 Classes/Objects

The following represents a decomposition of attributes and methods associated with each class of the BBMS. An Entity Relationship Diagram (ERD) summarizing these relationships can be seen in Appendix D.

3.2.1 Guest

This object is used to represent information about someone staying in the establishment. The Guest object is composed of six unique attributes, and eleven methods.

3.2.1.1 Attributes (direct or inherited)

3.2.1.1.1 id

This is an integer ID that represents a unique record for a guest.

3.2.1.1.2 lastName

This is a string that represents the last name of a guest.

3.2.1.1.3 firstName

This is a string that represents the first name of a guest.

3.2.1.1.4 address

This is a string that represents the address of a guest.

3.2.1.1.5 phoneNumber

This is a string that represents the phone number of a guest.

3.2.1.1.4 creditCardNumber

This is an integer that represents the credit card number used for payment by a guest.

3.2.1.2 Functions (services, methods, direct or inherited)

3.2.1.2.1 int getId()

Returns the unique ID of the guest.

3.2.1.2.2 string getLastName()

Returns the last name of the guest.

3.2.1.2.3 setLastName(string)

Sets the last name of the guest.

3.2.1.2.4 string getFirstName()

Returns the first name of the guest.

3.2.1.2.5 setFirstName(string)

Sets the first name of the guest.

3.2.1.2.6 string getAddress()

Returns the address of the guest.

3.2.1.2.7 setAddress (string)

Sets the address of the guest.

3.2.1.2.8 string getPhoneNumber()

Returns the phone number of the guest.

3.2.1.2.9 setPhoneNumber(string)

Sets the phone number of the guest.

3.2.1.2.10 int getCreditCardNumber()

Returns the credit card number of the guest.

3.2.1.2.11 setCreditCardNumber()

Sets the credit card number of the guest.

3.2.2 Reservation

The Reservation object is designed as the backbone of the BBMS, and stores reservation data such as dates, but also ID references to a guest, reserved room, and payment. The Reservation object is composed of nine unique attributes, and seventeen methods.

3.2.2.1 Attributes (direct or inherited)

3.2.2.1.1 id

This is an integer ID that represents the unique record of a Reservation.

3.2.2.1.2 startDate

This is a date object that represents the start of a reservation.

3.2.2.1.3 endDate

This is a date object that represents the end of a reservation.

3.2.2.1.4 price

Represents the dollar amount (as type double) of the price per day of the room.

3.2.2.1.5 isGuaranteed

Represents a Boolean toggle for the guaranteed status of a reservation. This has a relationship with a payment, which guarantees a room's reservation status.

3.2.2.1.6 guaranteeDate

This is a date object that represents the date that the guarantee is good for. Can be set to the end of the reservation if guarantee is paid, or to some date prior to the reservation if unpaid.

3.2.2.1.7 guestId

Reference to a unique Guest ID. One-to-one relationship between Guest and Reservation.

3.2.2.1.8 roomId

Reference to a unique Room ID. One-to-one relationship between Room and Reservation.

3.2.2.1.9 paymentId

Reference to a unique Payment ID. One-to-one relationship between Payment and Reservation.

3.2.2.2 Functions (services, methods, direct or inherited)

3.2.2.2.1 int getId()

Returns the unique integer ID representing a single Reservation.

3.2.2.2.2 date getStartDate()

Returns the date specific to the start of the Reservation.

3.2.2.2.3 setStartDate(date)

Sets the date specific to the start of the Reservation.

3.2.2.2.4 date getEndDate()

Returns the date specific to the end of the Reservation.

3.2.2.2.5 setEndDate(date)

Sets the date specific to the end of the Reservation.

3.2.2.2.6 double getPrice()

Returns the daily price of the Reservation.

3.2.2.2.7 setPrice(double)

Sets the daily price of the Reservation.

3.2.2.2.8 boolean toggleGuaranteed()

Toggles the guaranteed status of the Reservation, and returns the current value.

3.2.2.2.9 boolean isGuaranteed()

Returns the guaranteed status of the Reservation.

3.2.2.2.10 date getGuaranteeDate()

Returns the date specific to the guarantee of the Reservation.

3.2.2.2.11 setGuaranteeDate(date)
Sets the date specific to the guarantee of the Reservation.

3.2.2.2.12 int getGuestId()
Returns the unique Guest ID associated with the reservation.

3.2.2.2.13 setGuestId(int)
Sets the unique Guest ID associated with the reservation.

3.2.2.2.14 int getRoomId()
Returns the unique Room ID associated with the reservation.

3.2.2.2.15 setRoomId(int)
Sets the unique Room ID associated with the reservation.

3.2.2.2.16 int getPaymentId()
Returns the unique Payment ID associated with the reservation.

3.2.2.2.17 setPaymentId(int)
Sets the unique Payment ID associated with the reservation.

3.2.3 Room
This object represents a room in the establishment. The Room object is composed of three unique attributes, and five methods.

3.2.3.1 Attributes (direct or inherited)
3.2.3.1.1 id
This is an integer ID that represents the unique record of a Room.

3.2.3.1.1 vacant
Boolean flag specifying if the room is vacant (true) or not (false).

3.2.3.1.1 roomNumber
Integer value of the room number.

3.2.3.2 Functions (services, methods, direct or inherited)
3.2.3.2.1 int getId()
Returns the unique integer ID representing a single Room.

3.2.3.2.2 boolean toggleVacancy()
Toggles the vacancy status of a Room, and returns the current value.

3.2.3.2.3 boolean isVacant()
Returns the current vacancy status of a Room.

3.2.3.2.4 int getRoomNumber()
Returns the room number associated with the Room.

3.2.3.2.5 setRoomId(int)
Sets the room number associated with the Room.

3.2.4 Payment

This object is used to represent the positive financial transaction from a guest to the establishment for a room. The Payment object is composed of four unique attributes, and six methods.

3.2.4.1 Attributes (direct or inherited)

3.2.4.1.1 id

This is an integer ID that represents the unique record of a Payment.

3.2.4.1.2 paid

Total amount paid towards the Payment (total).

3.2.4.1.3 total

Total amount due for the Payment.

3.2.4.1.4 guestId

Reference to a unique Guest ID. One-to-one relationship between Guest and Payment.

3.2.4.2 Functions (services, methods, direct or inherited)

3.2.4.2.1 int getId()

Returns the unique integer ID representing a single Payment.

3.2.4.2.1 double getTotal()

Returns the total amount due.

3.2.4.2.1 setTotal(double)

Sets the total amount due.

3.2.4.2.1 makePayment(Double)

Makes a payment and increments against the paid variable.

3.2.4.2.1 int getGuestId()

Returns the unique Guest ID associated with the payment.

3.2.4.2.1 setGuestId(int)

Sets the unique Guest ID associated with the payment.

3.2.5 Finances

This object is designed as the representation for positive and negative financial transactions. The Finances object is composed of two unique attributes, and two methods.

3.2.5.1 Attributes (direct or inherited)

3.2.5.1.1 money

Amount of money represented from all payments.

3.2.5.1.2 payments

Collection of Payment objects that represent the sum of money.

3.2.5.2 Functions (services, methods, direct or inherited)

3.2.5.2.1 expense(Payment)

Increments the total money based on a payment.

3.2.5.2.2 profit(double)

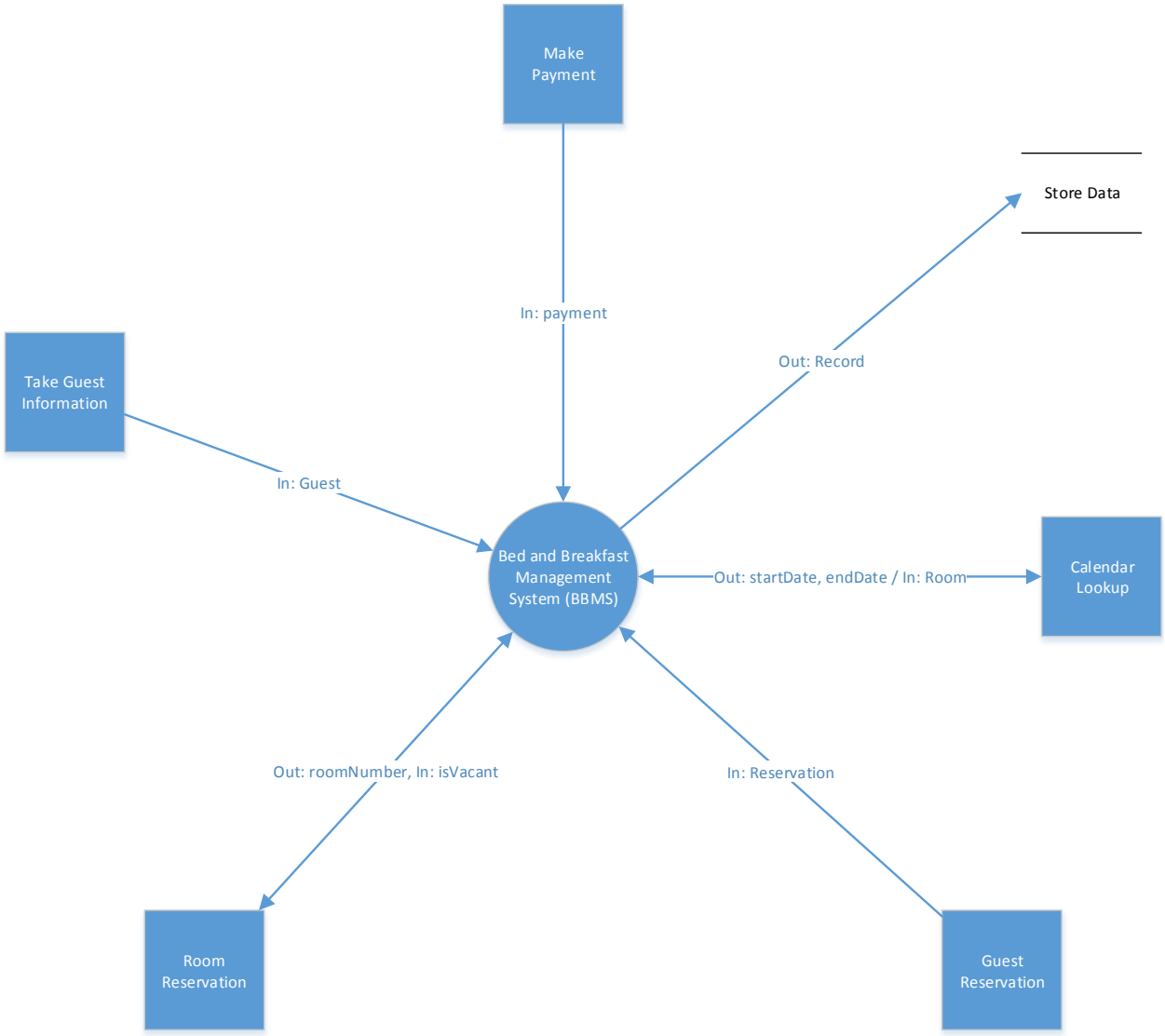
Decrements the total money.

3.3 Design constraints

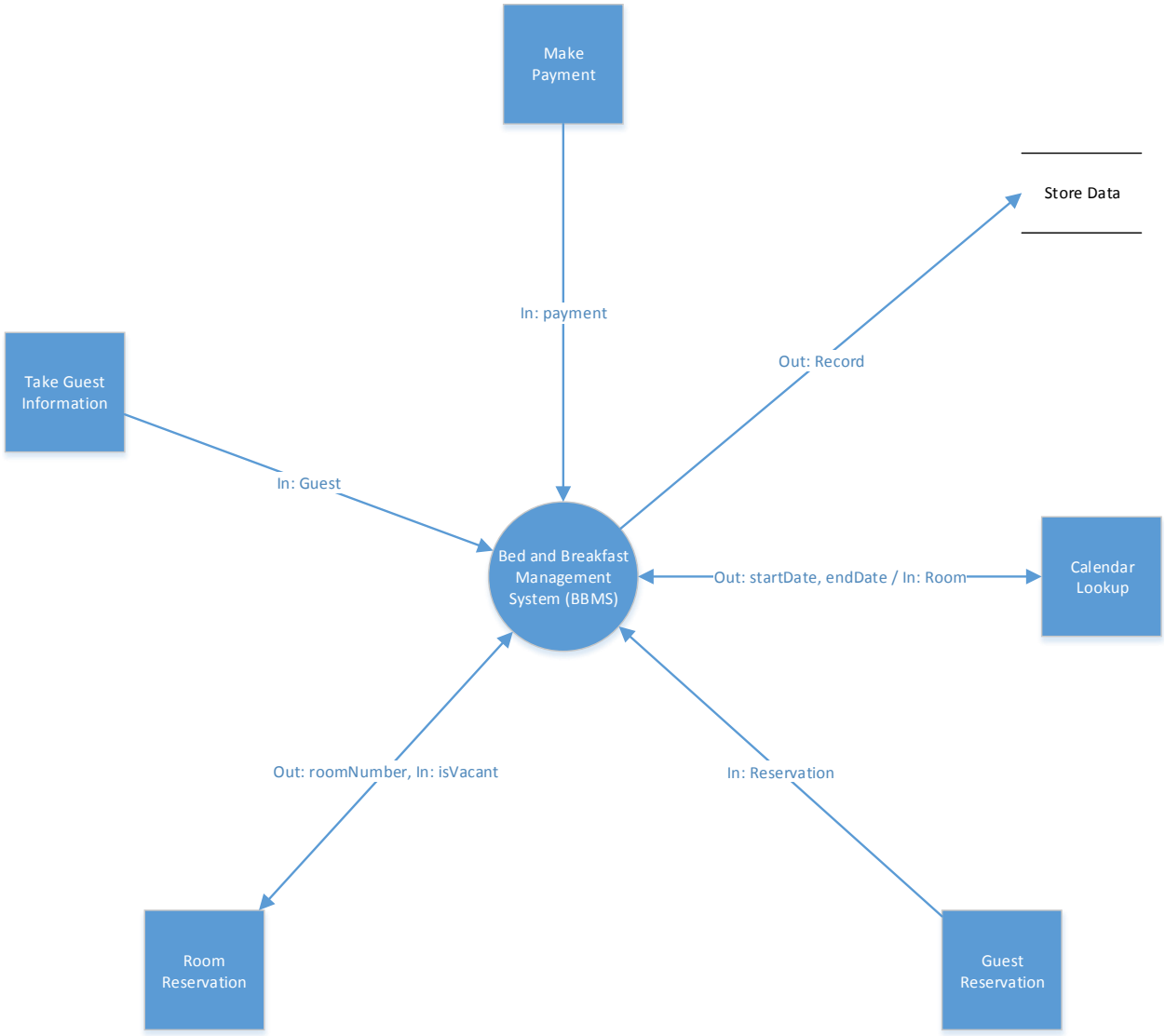
There are no known design constraints for this system based on the requirements.

Appendixes

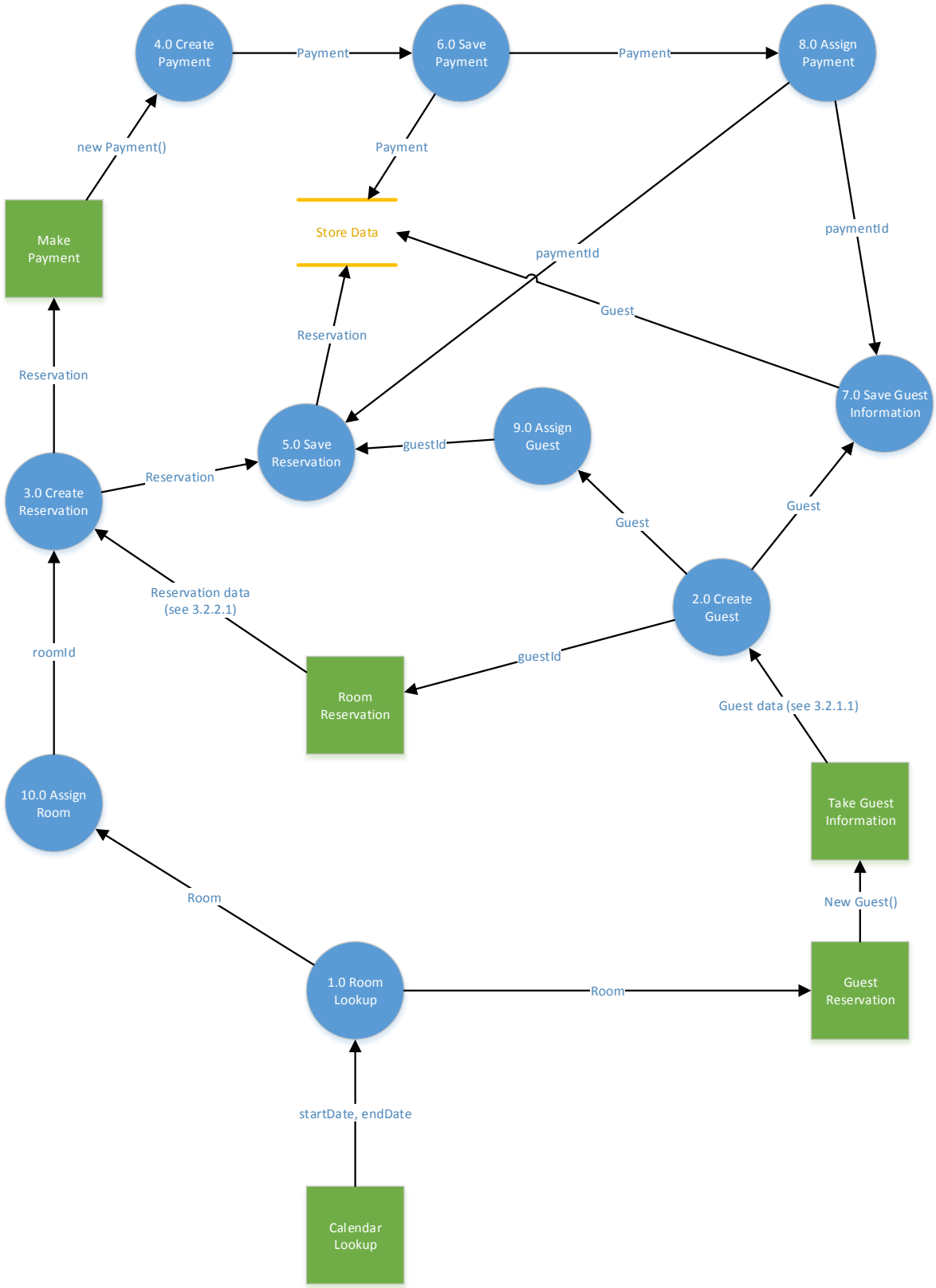
Appendix A: Architectural Context Diagram (ACD)



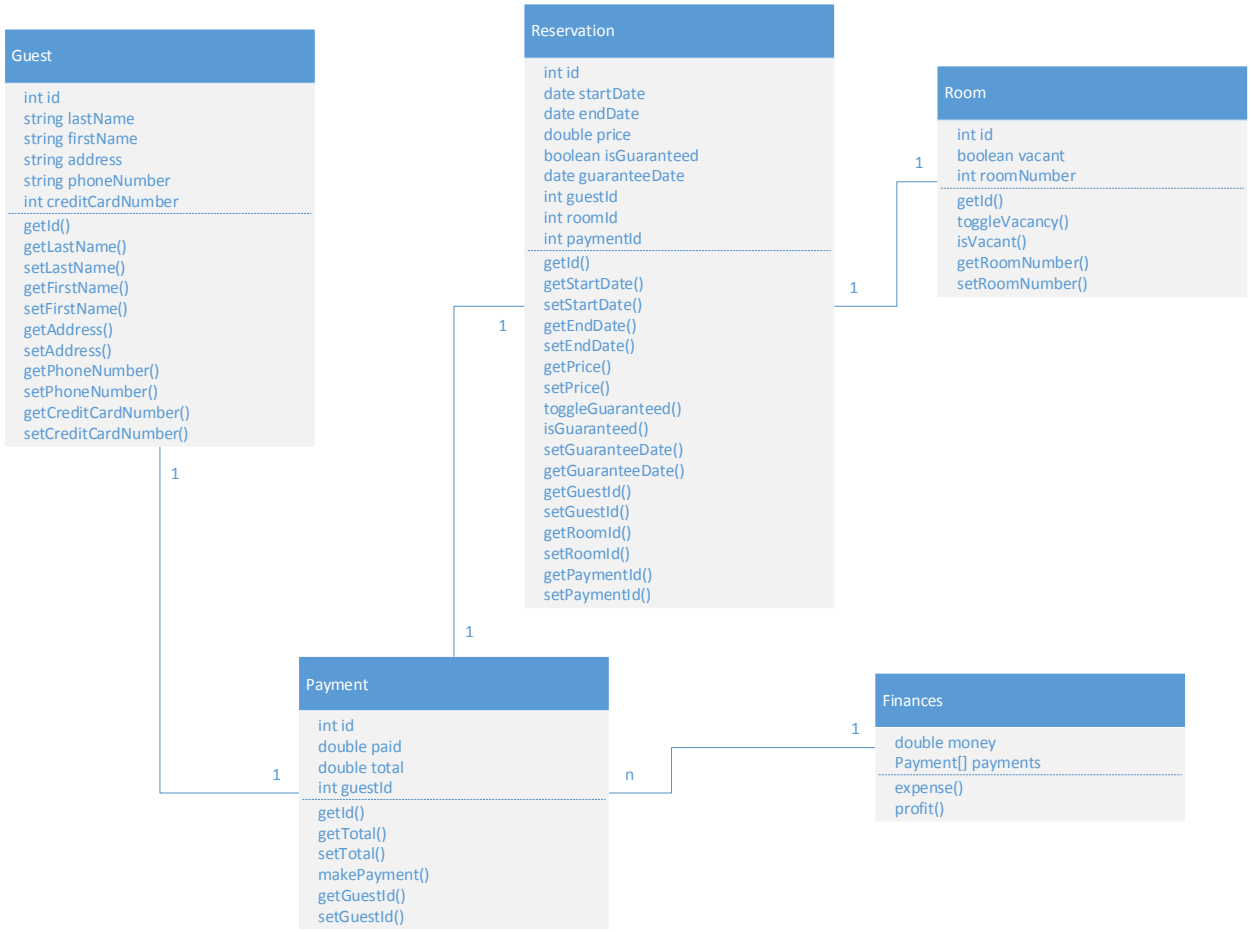
Appendix B: Level 0 Data Flow Diagram (DFD)



Appendix C: Level 1 Data Flow Diagram (DFD)



Appendix D: Entity Relationship Diagram (ERD)



Appendix F: Use Case / Scenario Diagrams

Use Case: Manager searches for room vacancy

Manager checks for a room vacancy based on a phone call from an interested guest.

Pre-condition: The software should be running and accessible to the manager.

Post-condition: The system should display a menu option to display the calendar.

Actor Profile: Manager is talking to a guest, who is looking to reserve a room for a series of days.

Sequence of Events:

1. Manager receives phone call from interested guest
2. Manager asks for dates associated with the reservation
3. Manager selects calendar from the menu
4. Manager enters date range supplied from the user
5. Manager verifies vacancy status for the date range

Scenarios

User Scenario 1: Start the calendar

1. User selects the calendar menu option
2. Calendar is displayed

User Scenario 2: Date range is entered

1. Start date is entered into the calendar menu
2. End date is entered into the calendar menu
3. Empty rooms from the start date to the end date are displayed

Use Case: Manager reserves a room

Pre-Condition: The software should be running and displaying vacancy information for a certain date range, and valid vacancy exists for the date range.

Post-Condition: The system should be ready to enter a reservation.

Actor Profile: Manager is talking to a guest who is ready to reserve a room. Manager has also selected a date of guarantee for the guest.

Sequence of Events:

1. Manager receives confirmation from the guest to reserve a room.
2. Manager enters guest information
3. Manager confirms reservation

Scenarios

User Scenario 1: User information is entered

1. Manager selects menu option to reserve a room
2. Manager enters guest last name
3. Manager enters guest first name
4. Manager enters guest phone number

5. Manager enters guest address
6. Manager enters guest credit card
7. Manager enters expected room price per day
8. Manager enters start date of reservation
9. Manager enters end date of reservation

User Scenario 2: Room is guaranteed with credit card for single day

1. Option to pay for single day is selected
2. Payment is processed for 1 day times room price per day
3. Reservation is flagged as guaranteed
4. Guarantee date is set to reservation end date

User Scenario 3: Room is guaranteed with credit card for entire stay

1. Option to pay for entire stay is selected
2. Payment is processed for number of days times price per day
3. Reservation is flagged as guaranteed
4. Guarantee date is set to reservation end date

User Scenario 4: Room is reserved without credit card

1. Option to reserve without pay is selected
2. Payment is not processed
3. Reservation is flagged as not guaranteed
4. Guarantee date is set to manager-specified date

Use Case: Room without paid guarantee has exceeded specified date

Pre-Condition: The software should be running hold information about a room that has exceed the non-paid guarantee date

Post-Condition: The system should be polling all reservations with the current date.

Actor Profile: System is conducting background tasks to verify guarantee dates.

Sequence of Events:

1. System is scanning all current reservations
2. System encounters a guarantee date that exceeds current date
3. System removes reservation

Scenarios

User Scenario 1: System encounters valid date

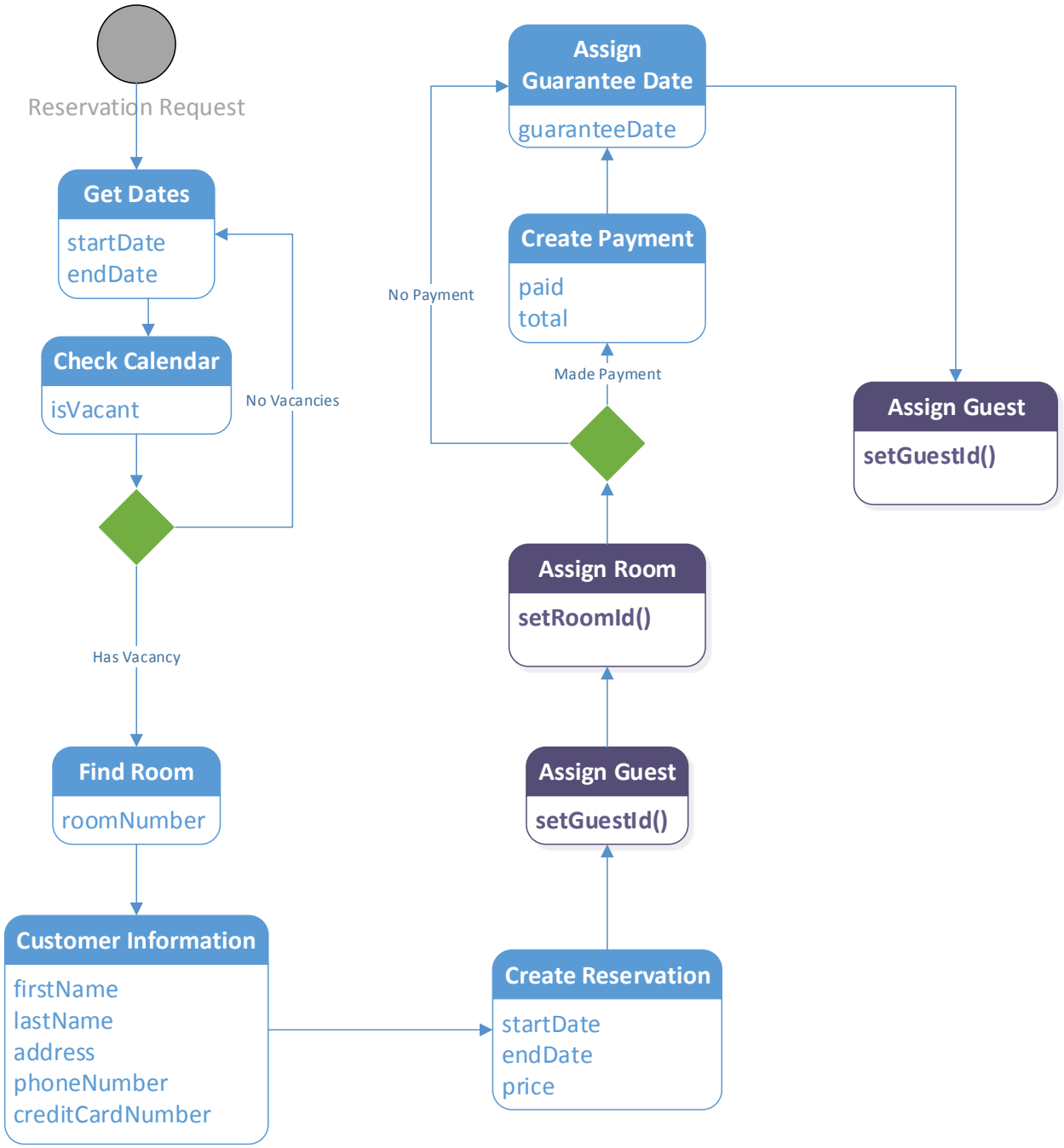
1. Guarantee date is checked from a reservation
2. Guarantee date is after current date
3. Reservation is skipped

User Scenario 2: System encounters invalid date

1. Guarantee date is checked from a reservation
2. Guarantee date is before current date
3. Reservation is removed from the collection

Appendix G: State Transition Diagrams

State Diagram 1: Reservation Request



State Diagram 2: Checking Guarantee Dates