

**Object Orientated  
Analysis & Design  
  
FOR PROPOSED**

**Hotel Management System**

**(April 2015)**

**Module:** CS4125 Systems Analysis & Design  
--------------------------------------------------------------------- **Authors:** Ross Tierney 12045284  
 Cormac Stone 12027243

**TABLE OF CONTENTS PAGE**

**SECTION 1: INTRODUCTION**

1.0 Marking Scheme

1.1 Narrative Description

1.2 Project Plan & Allocation of Roles

1.3 Software Life Cycle - Agile

**SECTION 2: REQUIREMENTS**

2.0 Functional Use Cases  
 2.0.1 Use Case Descriptions

2.1 Tactics to Support Non Quality Attributes

2.2 GUI Screenshots

**SECTION 3: ARCHITECTURE**

3.0 System Architecture ECB Pattern

3.1 System Architecture Entity Package

3.1.0 Singleton Pattern

3.2 System Architecture Control Package

3.3 System Architecture Entity Package

3.4 System Architecture Architectural Diagram  
 3.4.1 Class Diagram  
 3.4.2 Sequence State Chart for User Login  
 3.4.3 Added Value – Patterns  
 3.4.3.0 Observer Pattern  
 3.4.3.1 Factory Pattern  
 3.4.3.2 Decorator Pattern  
 3.4.3.3 State Pattern

3.5 Supporting Documentation

3.5.0 List of Candidate Objects  
 3.5.1 Early ECB Design  
3.5.2 Early Class Definitions  
 3.5.3 Early Entity Relationship Diagram with Cardinality  
 3.5.4 Early ECB Communications Diagram

**SECTION 4: APPENDIX**

4.0 References

4.1 Further Development

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **CS4125: Systems Analysis and Design. Semester II, 2014-2015 MARKING SCHEME for Team-Based Project:**  **Version 1 (19th February 2015 - Week 4)** | | | | | |  |
| Name:  Name | | | ID:  ID: | | |  |
|  | **Item** | **Detailed Description** | | **Marks**  **Allocated** | | **Marks Awarded** |
| Sub-total | Total |
|  | Presentation | * General Presentation * Adherence to guidelines i.e front cover sheet, blank marking scheme, table of contents | |  |  |  |
| 3 | Narrative | Narrative description of business scenario | |  |  |  |
| 4 | SLC | Discuss and justify SLC & risk mgt strategy? | |  |  |  |
| 5 | Project Plan | Plan specifying timeline, deliverables, and roles. | |  |  |  |
| 6 | Requirement | * Use case diagram(s) * Structured use case descriptions(s) * NFRs - quality attributes * Tactics to support quality attributes * Prototypes | |  |  |  |
| 7 | System  Architecture | System architecture diagram with interfaces | |  |  |  |
| 8 | Analysis | * Listing of Candidate classes * Sketch of a class diagram with generalisation, composition, multiplicity, dialog, control, entity, interfaces, pre and post conditions, etc. * Sketch of an Interaction diagram. * Entity relationship diagram with cardinality | |  |  |  |
| 9 | Code | * Compiles and runs * Object Oriented, interfaces, etc * MVC * Automated test cases * ADDED VALUE: (a) Architectural / Design pattern(s) (b) Concurrency (c) GUI | |  | P/F |  |
| 10 | Added Value | Two page discussion on added value | |  |  |  |
| 11 | Design  Blueprints | * Architectural diagram * Class diagram  Interaction diagram  State chart. * Description of patterns and approach to concurrency support. | |  |  |  |
| 12 | Critique | Evaluate the analysis & design artefacts. | |  |  |  |
| 14 | References |  | |  |  |  |
| 15 | Lab  Attendance | Attendance at labs (weeks 5-11) | |  |  |  |
|  | Interview Week 13 (Pass/Fail basis) | | |  |  |  |
|  | **Total** | | | | **50** |  |

**1.1 INTRODUCTION**

**Narrative Description**

**The Prince JJ Hotel – Limerick, Ireland**

Step into a world of ease and comfort, and experience  
the finest of Irish hospitality at the luxurious  
Prince JJ Hotel, located in the heart of historic Limerick within walking distance of the most  
extraordinarily breath-taking churches, castles and monuments.

Be treated to unrivalled comfort and discerning service, in a proud example of lovingly restored  
18th century paradise that offers beautifully  
appointed guestrooms and deluxe suites with  
private balconies providing stunning views  
of Limerick’s bustling cityscape.

Visit the highly acclaimed award winning, Scrumptious Ristorante, inspired by Limerick’s vibrant culture and history, offering sophisticated selections of delicious traditional and contemporary cuisine - perfect epicure for every occasion.

Overlooking the magnificent River Shannon, the hotel's luxurious 200 square meter swimming pool is an al-fresco masterpiece, nested in verdant gardens, and is maintained using Advanced Oxidation (AO) technologies, helping reduce the amount of required chlorine by 70%.

Additional amenities include high-speed Wi-Fi internet, air-conditioning, soundproofed rooms, TVs with international entertainment and news, in-room safe boxes, business desks with international power sockets, free hair dryers and ironing boards (on request), complimentary newspapers, and 24-hour concierge services, and much, much more.

Simply put, our goal is to bring you the very finest in hospitality and standards.

|  |
| --- |
| https://encrypted-tbn2.gstatic.com/images?q=tbn:ANd9GcRoMel7v54RxoclOal66jlgPVkL8xtr6zOZC-JqEKK7H2M9Um_A6w http://www.3dhousedownload.com/wp-content/uploads/2013/01/Luxury-hotel-interior.jpg  The Prince JJ Hotel, Limerick The Apex of Opulence |

**1.2 INTRODUCTION**

**Project Plan & Allocation of Roles**

|  |  |  |
| --- | --- | --- |
| **ROLE** | **DESCRIPTION** | **MEMBER** |
| GROUP | All team members | All team members |
| PROJECT LEADER | Project Management | Ross Tierney |
| SCRUM MASTER | Scrum team facilitator | Cormac Stone |
| STAKEHOLDER | University of Limerick | CSIS |
| PRODUCT OWNER | University Representative | JJ Collins |

**CONSTRAINTS:**

1. A team size of two members restricted the scope of this proposal.
2. This proposal was completed within a relatively short timeframe.
3. A limited project budget was available.

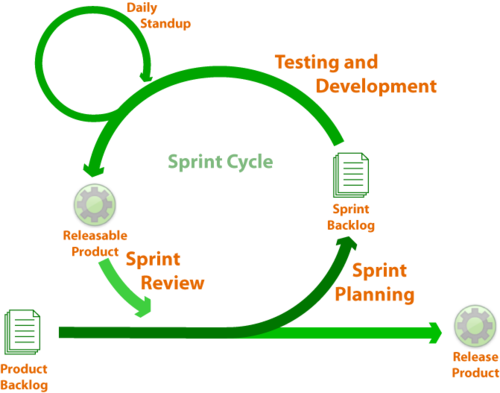
**Testing Constraints:**Automated JUnit Test Case Generation to automate the creation of comprehensive   
JUnit regression test cases was not implemented due to team size constraints.

**CONTROLS:**

1. Version control was implemented via <http://github.com/cs4125>
2. Group activities were co-ordinated via an online scrum board <http://scrumblr.ca/cs4125>
3. Agile project management principles were included as a team dynamic.
4. An activity log recording progress was updated after team meetings.

**1.3 INTRODUCTION**

**Software Lifecycle - Agile**

The Agile Model is defined via the “Manifesto for Agile Software Development” and it’s “Twelve Principles of Agile Software”, set in February 2001, when 17 software developers met at the Snowbird resort in Utah to discuss lightweight development methods.

**The manifesto dictates (direct extract):**

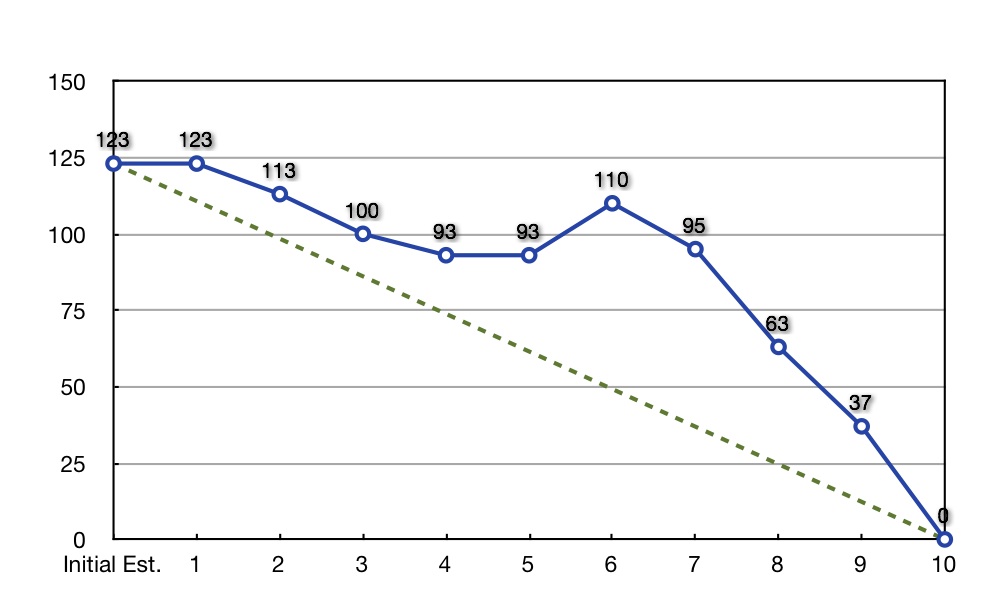
“We are uncovering better ways of developing software by doing it and helping others do it.  
Through this work we have come to value:

* Individuals and interactions over processes and tools.
* Working software over comprehensive documentation.
* Customer collaboration over contract negotiation.
* Responding to change over following a plan.

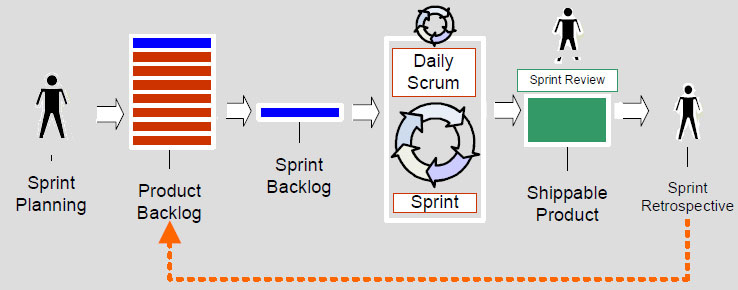
That is, while there is value in the items on the right, we value the items on the left more.”

**The 12 Principles of Agile as defined in 2001 are:**

1. Our highest priority is to satisfy the customer through early  
   and continuous delivery of valuable software.
2. Welcome changing requirements, even late in development.  
   Agile processes harness change for the customer's competitive advantage.
3. Deliver working software frequently, from a couple of weeks to a couple of months,  
   with a preference to the shorter timescale.
4. Business people and developers must work together daily throughout the project.
5. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
7. Working software is the primary measure of progress.
8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
9. Continuous attention to technical excellence and good design enhances agility.
10. Simplicity--the art of maximizing the amount of work not done--is essential.
11. The best architectures, requirements, and designs emerge from self-organizing teams.
12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behaviour accordingly.



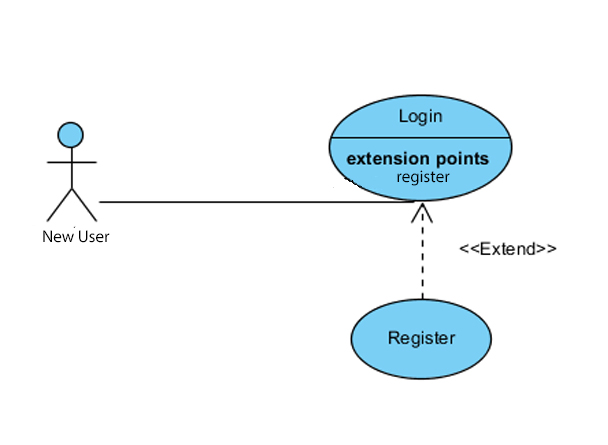
*Sample burndown chart showing story points vs sprint time in days*



*Overall Agile process*



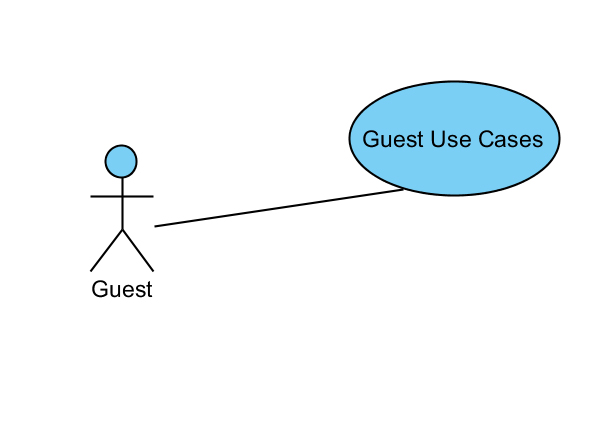
*Scrum board*

****2.0 REQUIREMENTS**

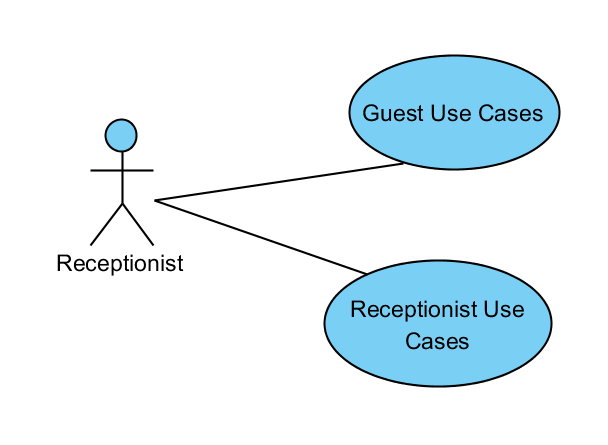
**Functional Use Cases**

Four potential Actors can use the System.

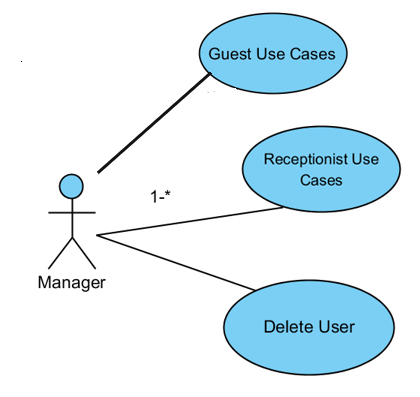
New Users have the most limited access.  
They can attempt to login, or register.



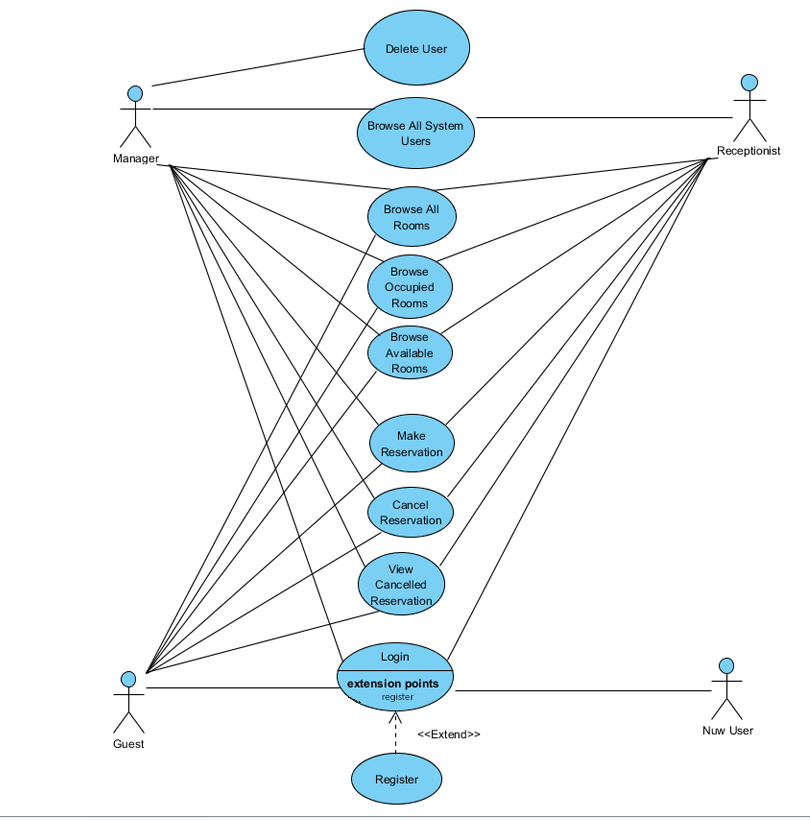
The three remaining Actors must all login.  
Of these, a Guest is the most limited.

**

A Receptionist has more functionality  
and inherits Guest functionality.

**

A Manager has maximum functionality  
and inherits both Guest and Receptionist  
functionalities.

A System Use Case diagram for all 4 Actors is illustrated below:

**2.0.1 Use Case Descriptions:**

|  |  |  |  |
| --- | --- | --- | --- |
| USE CASE 1: | Delete User | | |
| Goal in Context | Manager deletes a User account from the System. | | |
| **Scope & Level** | Hotel, summary. | | |
| **Preconditions** | The User exists on the System. | | |
| **Success End Conditions** | The User no longer exists on the System. | | |
| **Failed End Condition** | The User exists on the System. | | |
| **Primary, Secondary, Actors** | Only Managers delete all User types. | | |
| **Trigger** | Manager discretion. | | |
| **DESCRIPTION** | **Step** | | **Action** |
|  | 1 | | A decision to delete is authorised. |
|  | 2 | | A System deletion for User is requested. |
|  | 3 | | A System search is made for the User. |
|  | 4 | | A success/fail message is shown. |
|  | 5 | | A System search is made for the User. |
| **EXTENSIONS** | **Step** | | **Branching Action** |
|  | 4a | | User cannot be deleted. - call IT support. |
| **VARIATIONS** |  | | **Branching Action** |
|  | 1 | | User requests their own deletion. |
| RELATED INFORMATION | | 1. Make backup of User data | |
| Priority: | | Top. | |
| **Performance** | | 5 minutes required. | |
| **Frequency** | | 26/yr or upon every User deletion / Addition | |
| **Channel to actors** | | Closed | |
| **OPEN ISSUES** | | What if money is outstanding? | |
| **Due Date** | | Release 1.0 | |
| **…any other information…** | |  | |
| **Superordinates** | | Manage customer relationship | |
| **Subordinates** | | Maintain database | |

**Use Case 1: Delete User**Access: Manager.  
Description: Delete a User from the System.  
Non-Functional Requirements: Performance, safety, security, privacy, legal issues, backup.

**Use Case 2: Browse all System Users**Access: Manager, Receptionist.  
Description: Display details of all registered System Users.  
Non-Functional Requirements: Performance, security, privacy, legal issues, efficiency, currency.

**Use Case 3: Browse all Rooms**Access: Manager, Receptionist, Guest.  
Description: Display every room in the hotel.  
Non-Functional Requirements: Performance, usability, operability, efficiency.

**Use Case 4: Browse Occupied Rooms**Access: Manager, Receptionist, Guest.  
Description: Display a list of occupied rooms.  
Non-Functional Requirements: Performance, usability, operability, efficiency, currency.

**Use Case 5: Browse Available Rooms**Access: Manager, Receptionist, Guest.  
Description: Display a list of unoccupied rooms.  
Non-Functional Requirements: Performance, usability, operability, efficiency, currency.

**Use Case 6: Make Reservation**Access: Manager, Receptionist, Guest.  
Description: Reserve a room.  
Non-Functional Requirements: Performance, reliability, privacy, security, robustness.

**Use Case 7: Cancel Reservation**Access: Manager, Receptionist, Guest.  
Description: Cancel a reservation on a room.  
Non-Functional Requirements: Performance, privacy, security, robustness.

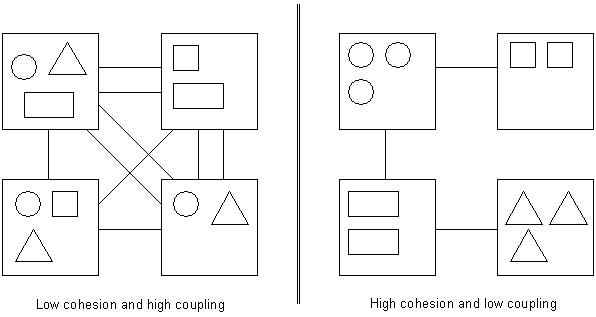
**Use Case 8: View Cancelled Reservations**Access: Manager, Receptionist, Guest.  
Description: Display a list of cancelled room reservations.  
Non-Functional Requirements: Performance, currency, operability.

**Use Case 9: Login**Access: Manager, Receptionist, Guest, New User.  
Description: Request access to the System.  
Non-Functional Requirements: Performance, reliability, security, privacy, usability.

**Use Case 10: Register**Access: New User.  
Description: Register as a User.  
Non-Functional Requirements: Performance, reliability, security, privacy, usability.

**2.1 REQUIREMENTS**

**Tactics to Support Non-Quality Attributes**



1The use of system package design patterns helped ensure maximised cohesion of code modules, while minimising coupling, within a sensible, functional, proven framework.

A discussion on the Entity Control Boundary (ECB) Pattern employed can be found in subsequent sections.

Examination of the class diagrams illustrated low coupling between packages, and between classes within Packages. Connections between different packages and classes are strictly controlled.

Further system design patterns are employed within the powerhouse which is the control package. The use of e.g. the observer pattern employed to observe Manager logins, adds to security.

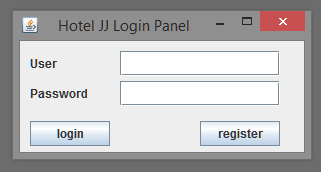
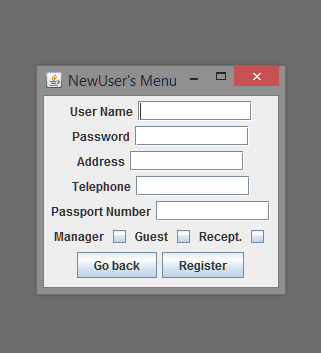
Maintainability is always an expensive concern. The proposed System employs pattern defined modularity with high code cohesion and low coupling in an attempt to address maintenance costs.

Confidentiality and privacy are protected by the use of the ECB data centric model, which separates users from data, placing the control package between them, to protect both data and clients.

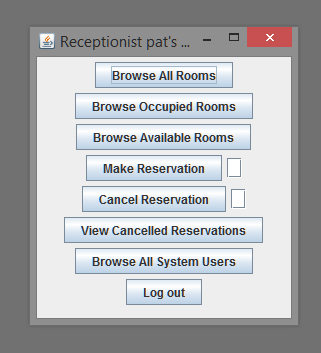
Finally, the use of the Agile software framework for developing the System will ensure constant re-iterations of System development with field-testing. Feedback from the field tests of earlier releases can be regularly integrated back into development ensuring higher levels of support for non-quality attributes.

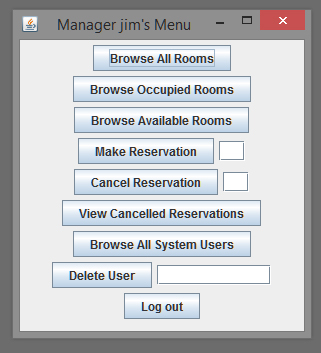
**2.2 REQUIREMENTS**

**GUI Screenshots**





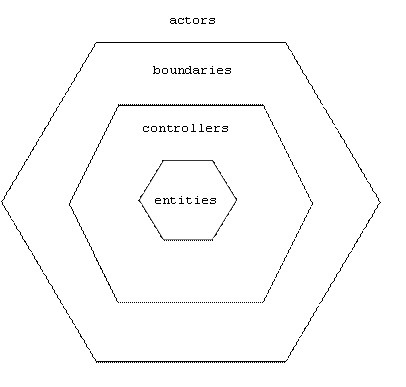




**3.0 ARCHITECTURE**

**System Architecture ECB Pattern**

The [Entity-Control-Boundary Pattern (ECB)](http://www.cs.sjsu.edu/~pearce/modules/patterns/enterprise/ecb/ecb.htm) is a variation of the [Model-View-Controller Pattern](http://www.cs.sjsu.edu/~pearce/modules/patterns/enterprise/presentation/mvc.htm).



Entity, Control, and Boundary are class stereotypes.

* **Entities** are objects representing system data -  
  Customers, Staff, Rooms, Reservations, etc.
* **Boundaries** are objects that interface  
   with actors – Login\_UI, NewUser\_UI, etc.
* **Controllers** are objects that mediate between  
  the separated boundaries and entities.

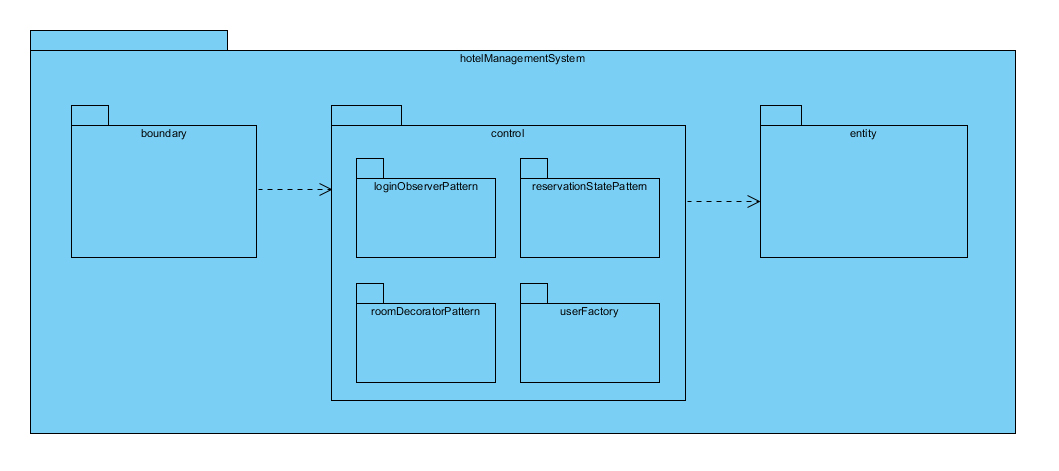
**Encapsulation:**

Actors only interact with boundary objects. Boundary objects only communicate with controller objects. Controller objects can communicate with both Entity objects and Boundary objects.  
Entity objects never communicate with Boundary objects.

**Dependencies:** Packages respect the following allowable set of ECB communications:

|  |  |  |  |
| --- | --- | --- | --- |
|  | Entity | Boundary | Control |
| Entity | X |  | x |
| Boundary |  |  | x |
| Control | x | x | x |

The hotel management System is divided into three packages, as defined by the ECB pattern:

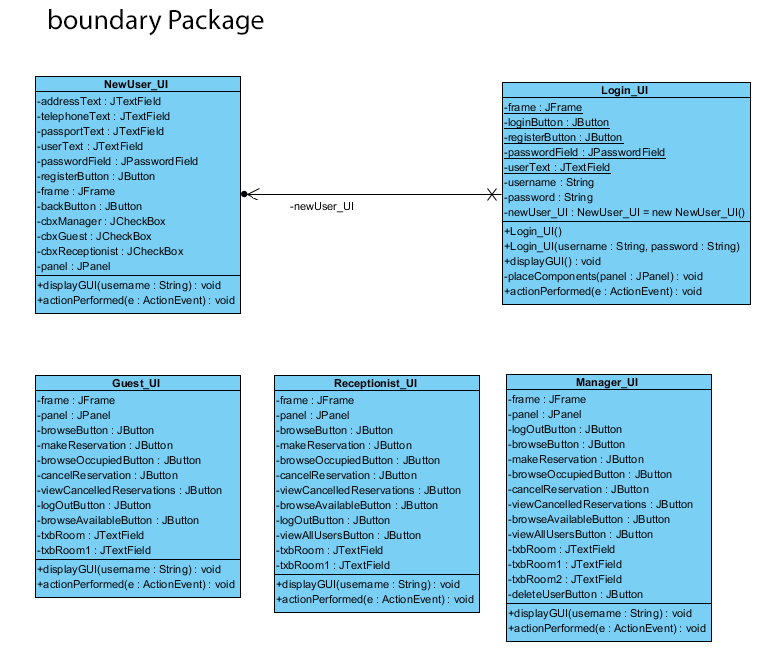


**3.1 ARCHITECTURE**

**System Architecture Boundary Package**

The boundary package represents an Actors gateway into the hotel management System  
and consists of a set of graphical user interfaces, as shown in “GUI Screenshots”.

As per the ECB pattern, a Boundary object never communicates with an Entity object and deals strictly with Controller objects only. The Boundary package contains separate GUI interface classes for each user-type.

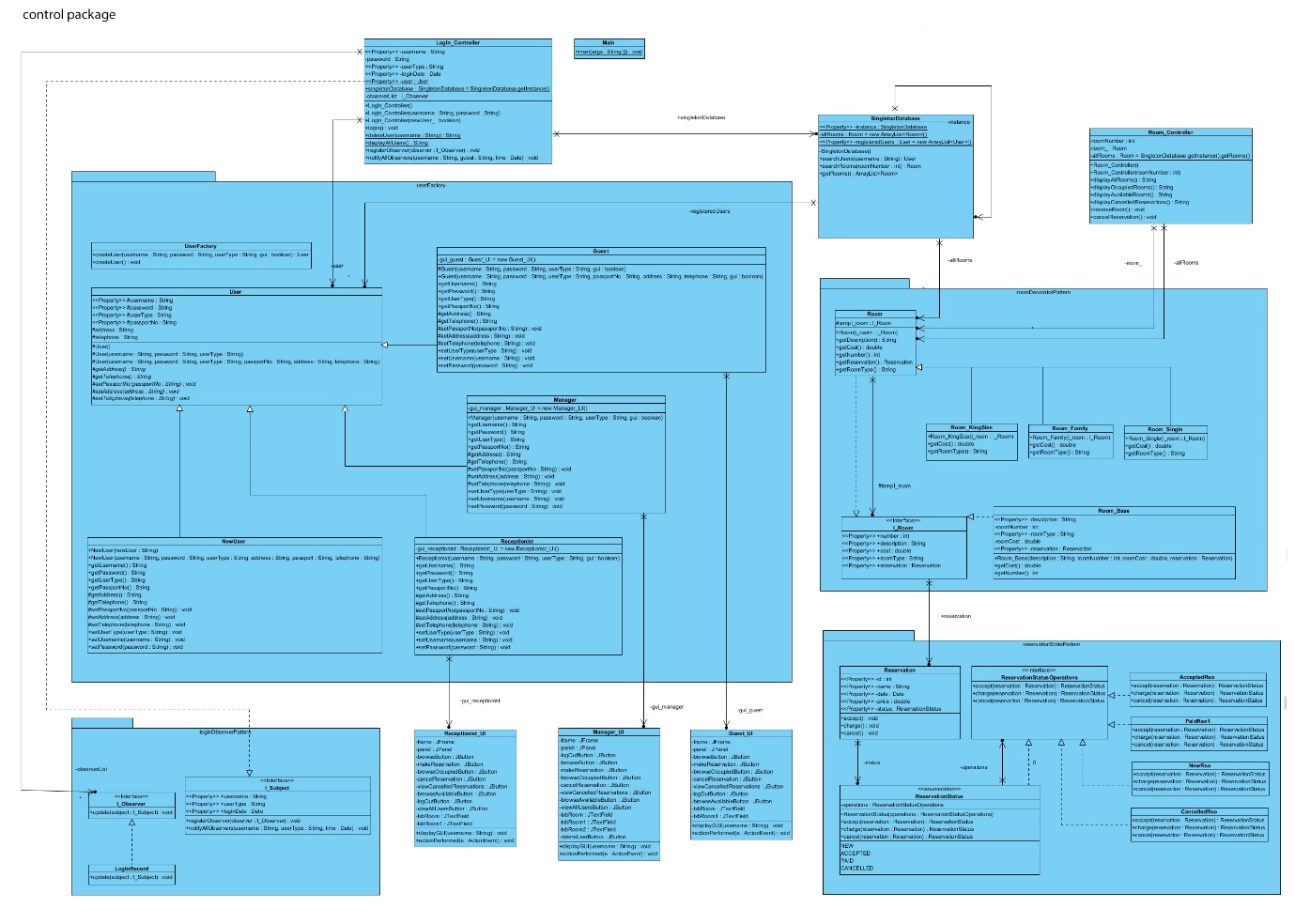


**3.2 ARCHITECTURE**

**System Architecture Control Package**

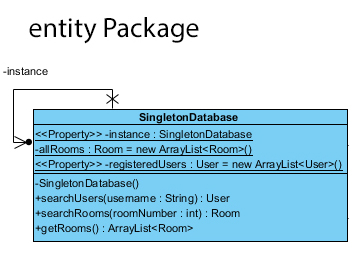
The Control package is the powerhouse of the hotel management system. If the Boundary package handles user interactions, and the Entity package handles data interactions, then their intermediary, the Control package, is where all functionality, aka our Use Cases, is implemented.

The sequence diagram above illustrates Boundary access into Control via Login\_Controller.



**3.3 REQUIREMENTS**

**System Architecture Entity Package**



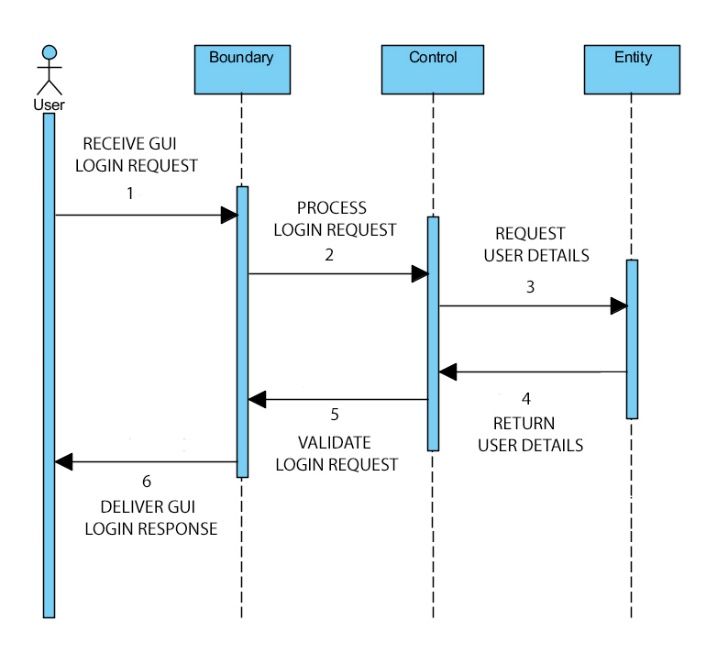
Within the ECB framework, the Entity package represents data-centricity, represented as the centre of the hexagonal illustration above, protected by its ECB communications dependencies set.

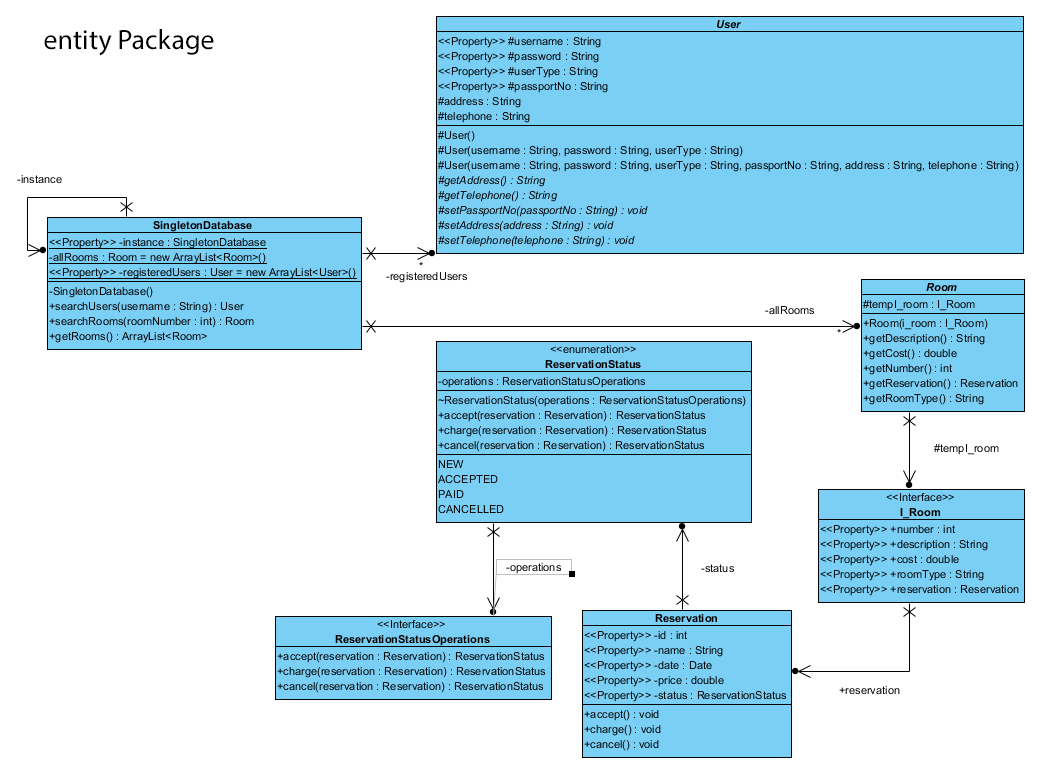
For this proposal, the Entity package is limited to one class which maintains System data. In practice, there would be a separate Entity sub class of our SingletonDatabase for each user-type to collate data for its appropriate User.

**3.3.0 Singleton Pattern: Entity Package**

The Entity package employs the Singleton design Pattern to ensure that only one instance of the Singleton database can ever exist, via the use of a private constructor linked to a unique single Class instance variable.

The main objective of this pattern within the context of this project is to ensure security, confidentiality and privacy when dealing with the critical core of System operations - data.

The diagram to the right illustrates how a Control object mediates between a Boundary object and an Entity object during a  
User login sequence.

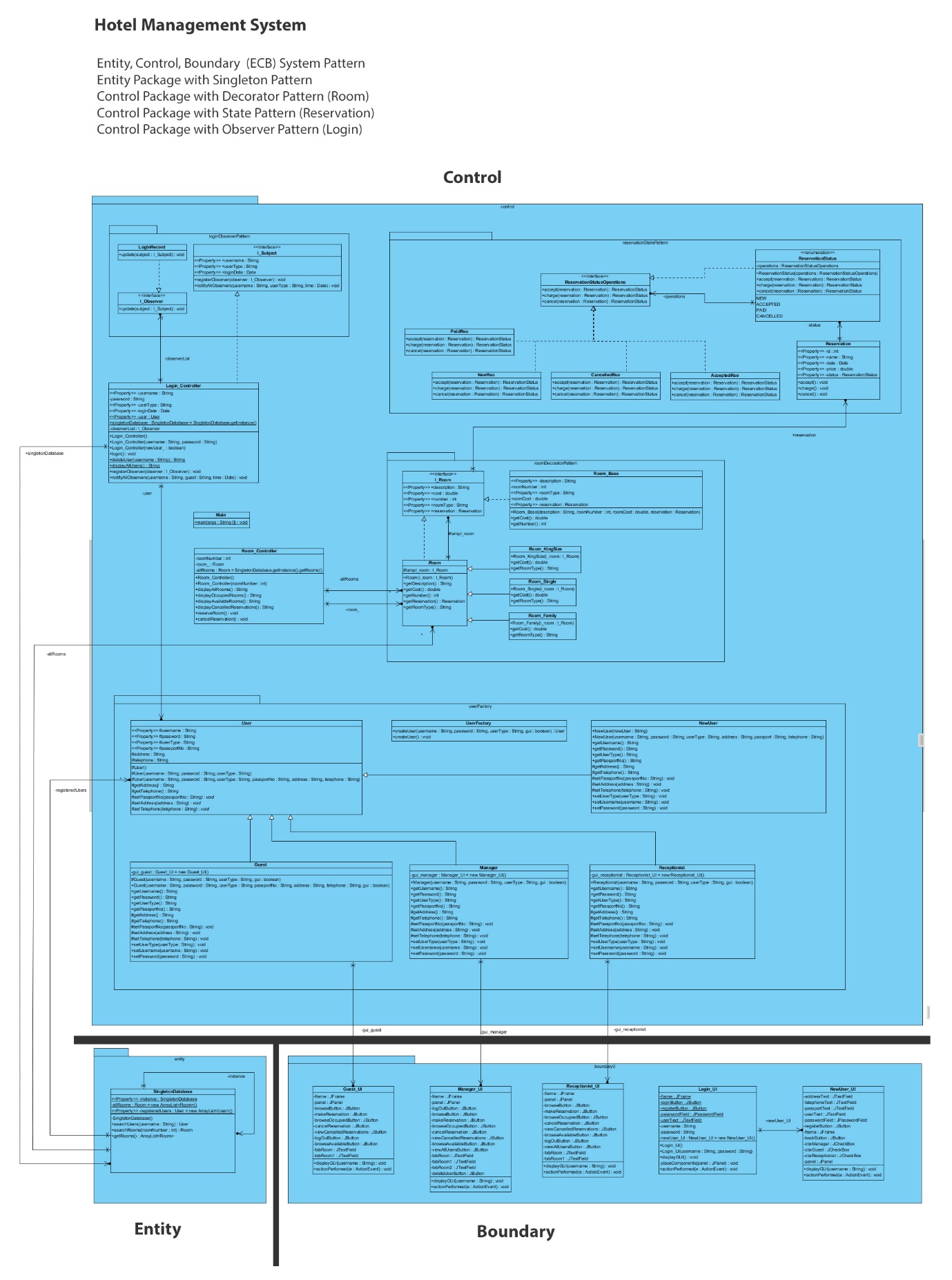
****

**3.4 ARCHITECTURE**

**Architectural Diagram**

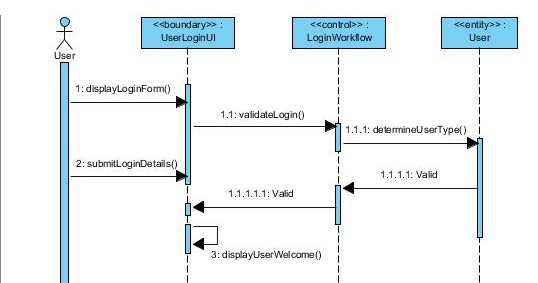


**3.4.1 Class Diagram**



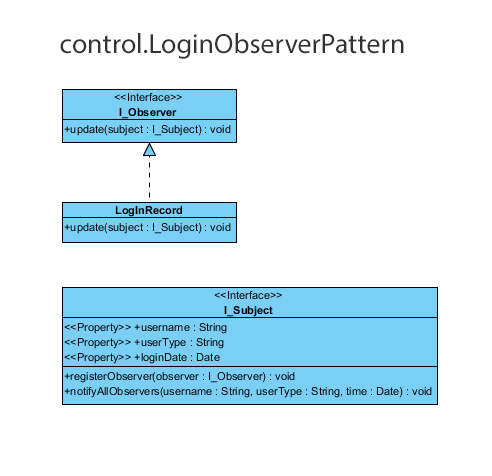
**3.4.2 Sequence State Chart for User Login:**

The diagram below illustrates the interaction between an unknown Actor and the Entity database via the Control package as executed during the login procedure.



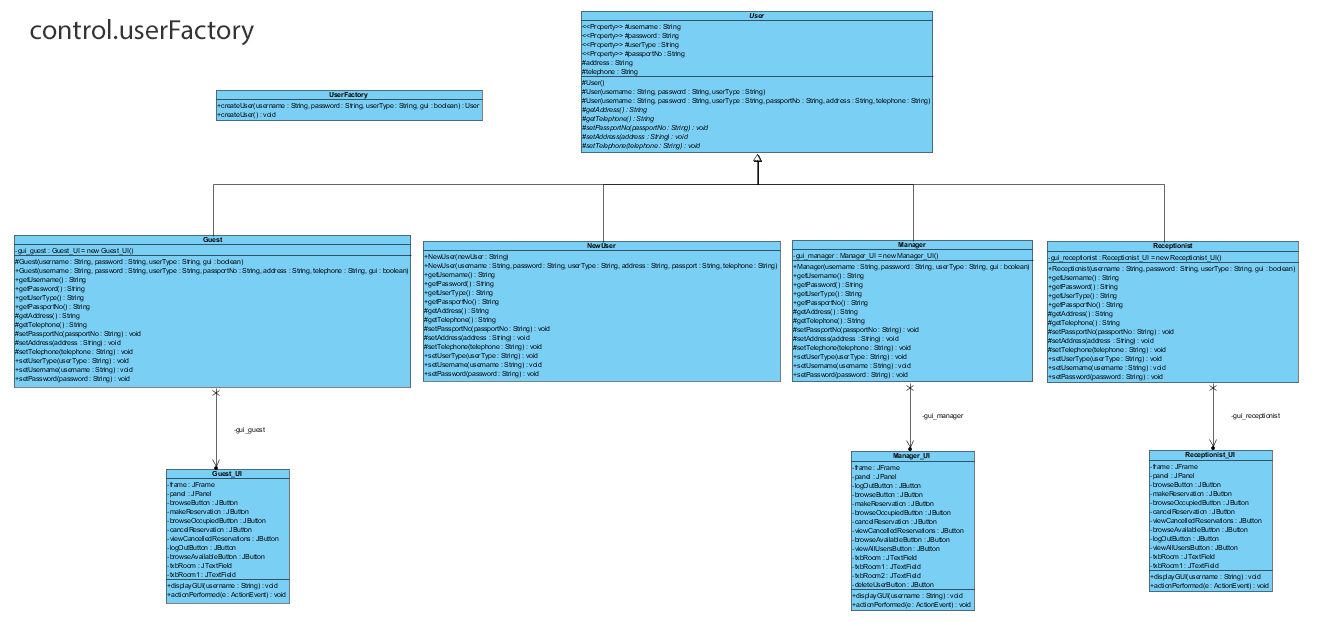
**3.4.3 Added Value:**

Added value was implemented through the use of the Factory, Observer, State, and Decorator Patterns within the Control package, in addition to the overall ECB System architecture, and the use of the Singleton pattern in the Entity package

**3.4.3.0 Observer Pattern – Control Package:**

Login\_Controller is being watched by loginObserverPattern Class LoginRecord. Whenever a Manager login occurs, Observer LoginRecord receives a notification from its Subject Login\_Controller and issues an update method that appends the login details to a text file.

**3.4.3.1 Factory Pattern – Control Package:**

****Upon successful login, a UserFactory object is created. Calling createUser on this object creates the appropriate new User object required by the login.The main advantage is that the factory pattern allows the use of polymorphism for the creation of objects, not just object use.

**3.4.3.2 Room Decorator Pattern – Control Package:**

Rooms are created as Room\_Base which have a base cost and which extends the abstract Room Class, which implements the I\_Room interface and has an I\_Room object as a parameter. Room type is considered an extra cost. Three room types are available – Single, Family and King Size which take a Base\_Room as a parameter. This pattern allows for wrapping extras onto existing base objects.

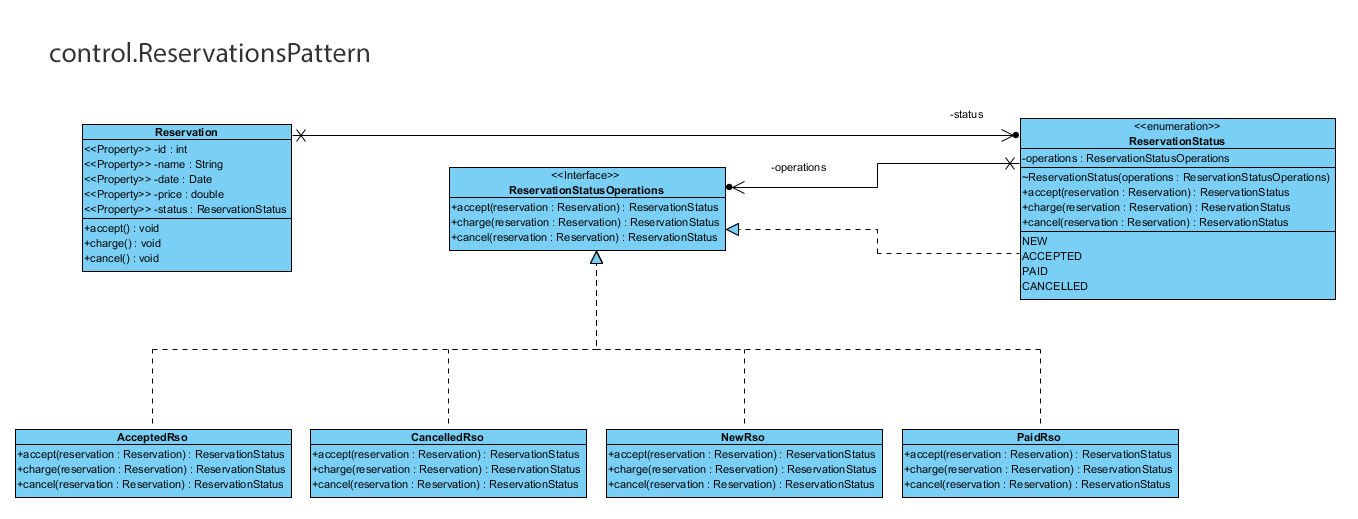
****Room\_KingSize four = **new** Room\_KingSize(**new** Room\_Base(**"Sleeps 5"**, 4, 30.00, **new** Reservation()))

**3.4.3.3 State Pattern - Control Package:**

The hotel management System creates four objects which represent various behavioural reservation states: AcceptedRso, CancelledRso, NewRso and PaidRso.

These four Classes implement a common interface and trade ReservationStatusOperations  
with a context class, ReservationStatus.

Every Reservation object has a ReservationStatus. This status can be of one of the four state Classes,  
Accepted, Cancelled, New or Paid.



**3.5 ARCHITECTURE**

**Supporting Documentation**

**3.5.0 List of Candidate Objects:** - derived by applying data driven design to use case descriptions.

*Room - what we are dealing in*

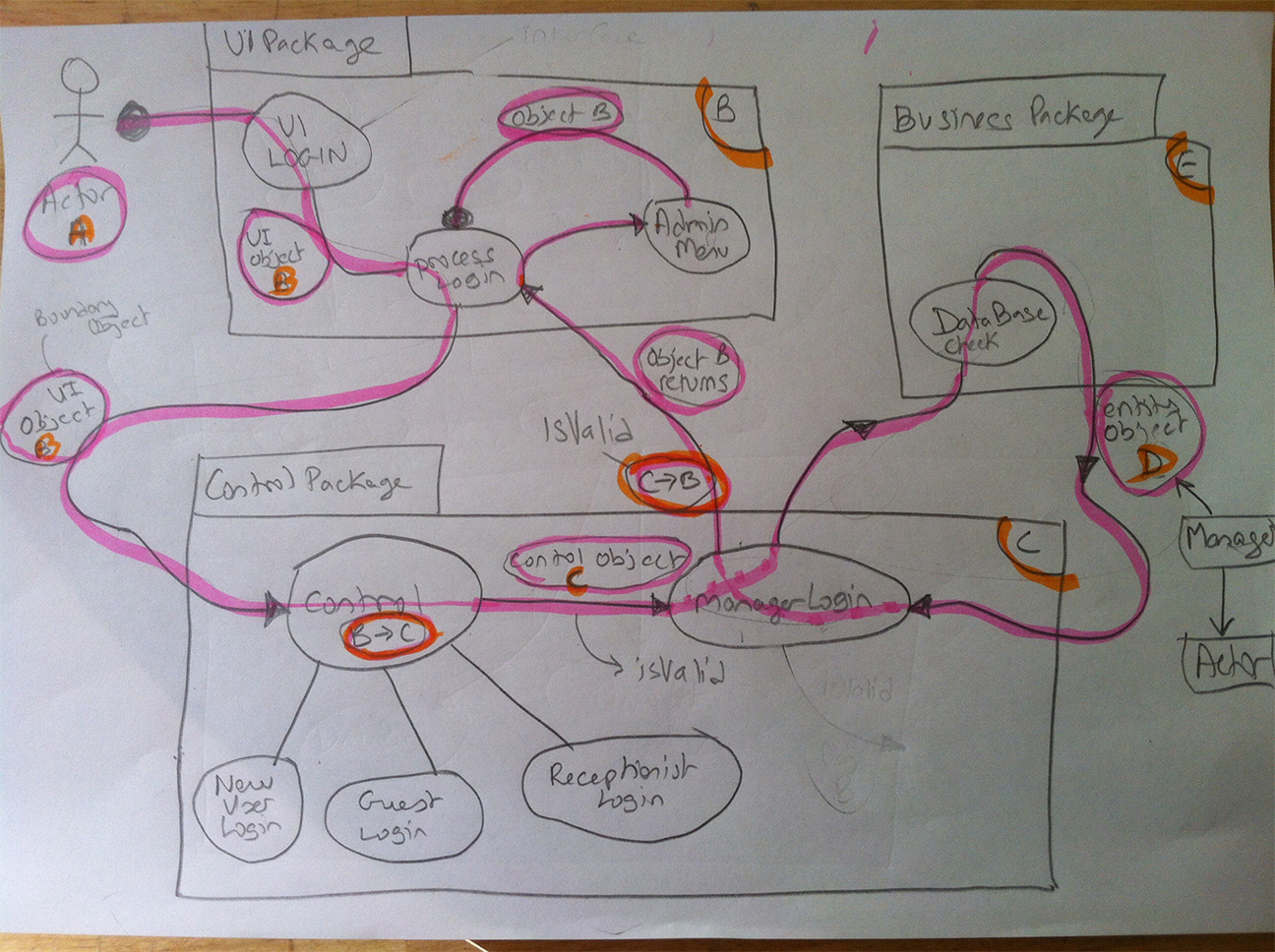
*Reservation - how we interact with customers*

*User, NewUser, Manager, Receptionist, Guest - who we process as*

*GUI\_ NewUser, GUI\_ Manager, GUI\_ Receptionist, GUI\_Guest - us as people*

*Database - our customers’ data*

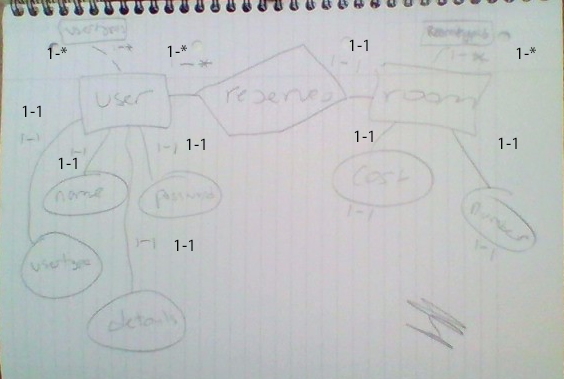
**3.5.1 Early ECB Design** *(supporting documentation – not to release standard)*



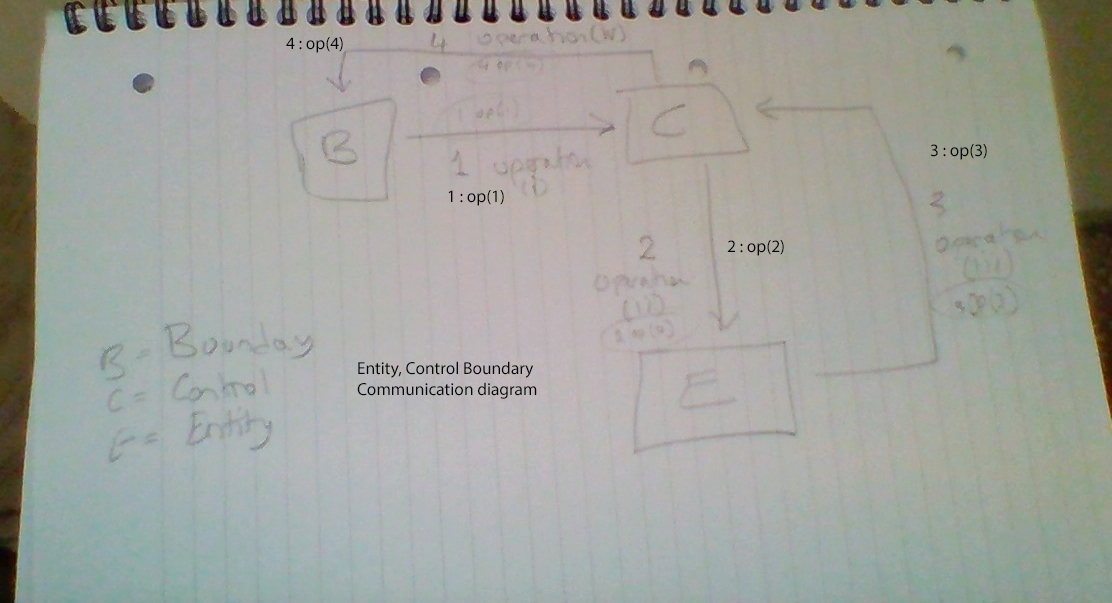
**3.5.2 Early Class Definitions:***(supporting documentation – not to release standard)*

****

**3.5.3 Early Entity Relationship Diagram with Cardinality:***(supporting documentation – not to release standard)*

****

**3.5.4 Early ECB Communications Diagram:***(supporting documentation – not to release standard)*

****

**4.0 APPENDIX**

**References**

Technical Report CMU/SEI-95-TR-021 ESC-TR-95-021  
Quality Attributes Mario Barbacci, Mark H. Klein, Thomas A. Longstaff, Charles B. Weinstock  
December 1995

<http://www.sei.cmu.edu/reports/95tr021.pdf>

Entity Boundary Control:  
<http://www.cs.sjsu.edu/~pearce/modules/patterns/enterprise/ecb/ecb.htm>

**4.1 APPENDIX**

**Further Implementation:**

Improvements to the implementation have been carried out since the publication of this document.

A summary of and code for these improvements is available from the authors.