**Discworld Ankh Morpork**

*Software Architecture Documentation*

# 1 - Introduction

**1.1****Purpose**

This document provides a comprehensive architectural overview of the Discworld Ankh Morpork board game, using a number of different architectural views to depict different aspects of the game. It is intended to capture and convey the significant architectural decisions which have been made on the system.

**1.2****Scope**

This Software Architecture Document provides an architectural overview of the Discworld Ankh Morpork. The Discworld Ankh Morpork is being developed by Group 7 – Advance Programming Practices in Concordia University to play the game on computer by 2 to 4 people.

This Document has been generated directly from the Discworld Ankh Morpork Analysis & Design Model implemented in \_\_\_\_\_\_\_\_\_\_\_\_\_\_.

**1.3****Definitions, Acronyms and Abbreviations**

Need to write.

**1.4****References**

Applicable references are:

Need to write.

# 2 - Architectural Representation

This document presents the architecture as a series of views; use case view, logical view, process view and deployment view. There is no separate implementation view described in this document. These are views on an underlying Unified Modeling Language (UML) model developed using -------------------------------.

# 3.  Goals and Features

There are some key requirements and system constraints that have a significant bearing on the architecture. They are:

* Ability to create and populate the data structures required to model the state of a game.
* Program must initialize all data structures appropriately for the start of a new game, following all of the instructions given on page 3 of the [rule book](http://www.fantasyflightgames.com/ffg_content/kingdoms/support/KN20_Kingdoms_Rulebook_sm2.pdf).
* Ability to save the current game state to a file, in a format of your choice (note that the user must be able to specify the name of the file to save to)
* Ability to load the game state from a file (note that the user must be able to specify the name of the file to load from)
* Ability to display the game state in text mode
* Ability to model an entire game, with 2-4 human players (using only the green-bordered Player cards for now; the brown-bordered cards are an optional feature for this build)
* Your code for this build must incorporate at least one design pattern other than Singleton (i.e. if you include Singleton, the requirement is to incorporate two patterns).

# 4.  Use-Case View

A description of the use-case view of the software architecture. The Use Case View is important input to the selection of the set of scenarios and/or use cases that are the focus of iteration. It describes the set of scenarios and/or use cases that represent some significant, central functionality. It also describes the set of scenarios and/or use cases that have a substantial architectural coverage (that exercise many architectural elements) or that stress or illustrate a specific, delicate point of the architecture.

The Ankh Morpork use cases are:

- Load an existing Game

- Start a New Game

- Assign Assets to each Players (minion, money etc..)

- Select A Player Card

- Action According to Player Card

- Winner Decision

These use cases are initiated by the student or the player actors.

## 4.1 Use Cases Diagram

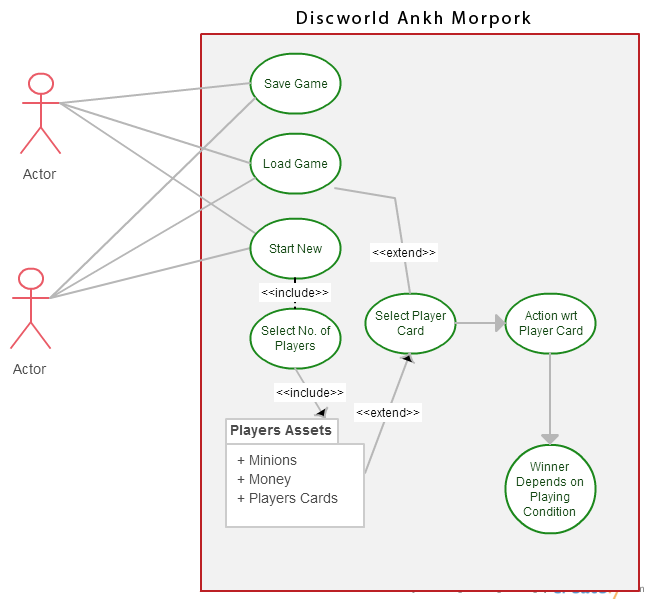


Diagram Name: Ankh Morkpork Use-Cases

### 4.1.1 Save Game

This use case allows a player to save the game if they need to leave the game at any stage. The action collects the information of all the players, their existing assets and objects on the board to save in file. Player has to choose a file and file is saved in json format.

### 4.1.2 Load An Existing Game

This use case allows a player to select an existing game saved previous to resume from the same position where it was left.

### 4.1.3 Start A New Game

It allows you to select if you have to start a new game and further moved to next step.

### 4.1.4 Select Players

This action allows you to select number of you players you want to play. Number of players could be any between 2 to 4.

### 4.1.5 Players Assets

It’s a kind of auto process where each players is assigned a color and according to that color player get 10$, minions, 5 players cards.

### 4.1.6 Select Player

On a player turn player choose a player card from his / her cards and does the action according after completing the action of all cards player can get more cards to maintain 5 cards with him / her.

### 4.1.7 Action W.R.T Player Card

Players do the action while using their playing cards that could be one or more action with respect to the card chooses by the player. There are various action such as Placing the minion, removing the building, Assassination, getting money from other player etc….

### 4.1.8 Winner Decision

With respect to personality card possessed by a plyer winner conditions are applied that make the decision of the winner.

# 5.  Logical View

A description of the logical view of the architecture. Describes the most important classes, their organization in service packages and subsystems, and the organization of these subsystems into layers. Also describes the most important use-case realizations, for example, the dynamic aspects of the architecture. Class diagrams may be included to illustrate the relationships between architecturally significant classes, subsystems, packages and layers.

The logical view of the game comprised of the 2 main packages: User Interface, and Playing Conditions.

The User Interface Package contains classes for each of the action that the actors use to communicate with the System. Boundary classes exist to support Selecting players, assigning objects, resume the game, taking action with respect to card instructions, maintaining of other players info, adding or removing objects, maintaining personal game info, completing turn, and viewing cards.

The Playing Condition includes classes for the players to play according to the rules and regulation define in the rule book.

**6.****Process View**

A description of the process view of the architecture. Describes the processes involved in the system's execution, their interactions and configurations. Also describes the allocation of objects and classes to tasks.

The Process Model illustrates the ankh morpork classes organized as executable processes. Processes exist to support start new game, players functions, changing player turn, and access to board areas to placement or removal of object.

**6.1****Processes**

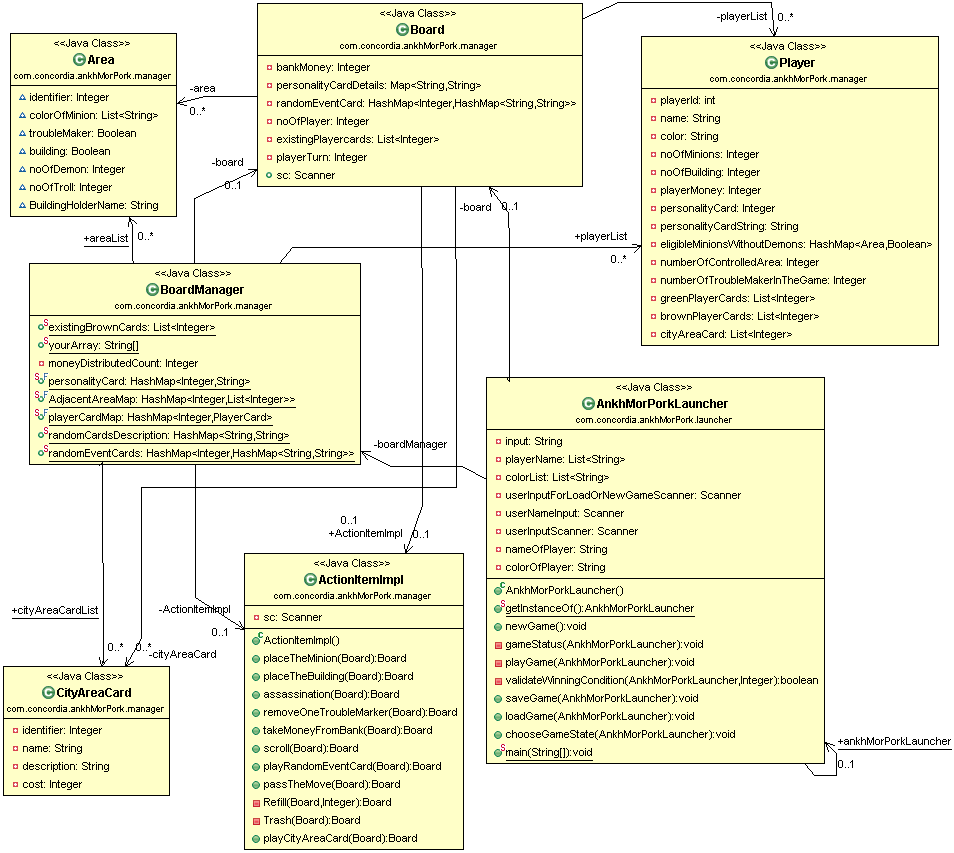


Diagram Name: Class Diagram

*6.1.1**CourseCatalogSystemAccess*

This process manages access to the legacy Course Catalog System. It can be shared by multiple users registering for courses. This allows for a cache of recently retrieved courses and offerings to improve performance.

The separate threads within the CourseCatalog process, CourseCache and OfferingCache are used to asynchronously retrieve items from the legacy system.

Analysis Mechanisms:

- Legacy Interface

Requirements Traceability:

- Design Constraints: The system shall integrate with existing legacy system (course catalog database).

*6.1.2**CourseCatalog*

The unabbridged catalog of all courses and course offerings offered by the university including those from previous semesters.

This class acts as an adapter (see the Gamma pattern). It works to makes sure the CourseCatalogSystem can be accessed through the ICourseCatalog interface to the subsystem.

*6.1.3**CourseRegistrationProcess*

There is one instance of this process for each student that is currently registering for courses.

*6.1.4**RegistrationController*

This supports the use case allowing a student to register for courses in the current semester. The student can also modify or delete course selections if changes are made within the add/drop period at the beginning of the semester.

Analysis Mechanisms:

- Distribution

*6.1.5**StudentApplication*

Manages the student functionality, including user interface processing and coordination with the business processes.

There is one instance of this process for each student that is currently registering for courses.

*6.1.6**MainStudentForm*

Controls the interface of the Student application. Controls the family of forms that the Student uses.

*6.1.7**BillingSystemAccess*

This process communicates with the external Billing  System to initiate student billing.

*6.1.8**CloseRegistrationProcess*

The Close Registration process is initiated at the end of the registration time period. This process communicates with the process controlling access to the Billing  System.

*6.1.9**BillingSystem*

The Billing System supports the submitting of student bills for the courses registered for by the student for the current semester.

Analysis Mechanisms:

- Legacy Interface

*6.1.10**CloseRegistrationController*

The Close Registration Controller controls access to the Billing  System.

Analysis Mechanisms:

- Distribution

**7.****Deployment View**

A description of the deployment view of the architecture Describes the various physical nodes for the most typical platform configurations. Also describes the allocation of tasks (from the Process View) to the physical nodes.

This section is organized by physical network configuration; each such configuration is illustrated by a deployment diagram, followed by a mapping of processes to each processor.

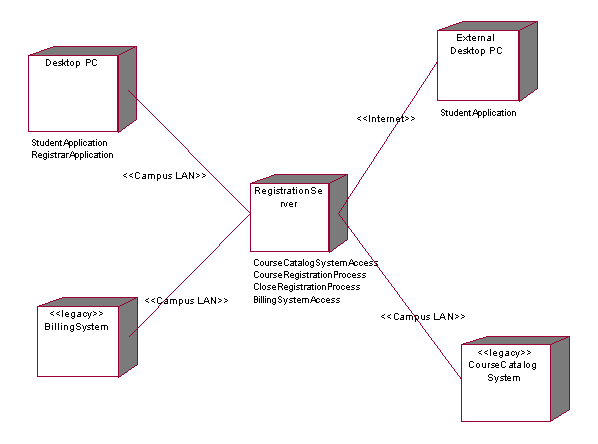


Diagram Name: Deployment View

**7.1****External Desktop PC**

Students register for courses using external desktop PCs which are connected to the College Server via internet dial up.

**7.2****Desktop PC**

Students register for courses via local Desktop PCs that are connected directly to the College Server via LAN. These local PCs are also used by professors to select course and submit student grades. The Registrar uses these local PCs to maintain student and professor information.

**7.3****Registration Server**

The Registration Server is the main campus UNIX Server. All faculty and students have access to the Server through the campus LAN.

**7.4****Course Catalog**

The Course Catalog System is a legacy system that contains the complete course catalog. Access to it is available via the College Server and LAN.

**7.5****Billing System**

The Billing System (also called the Finance System) is a legacy system that generates the student bills each semester.

**8.****Size and Performance**

The chosen software architecture supports the key sizing and timing requirements, as stipulated in the Supplementary Specification [15]:

* + 1. The system shall support up to 2000 simultaneous users against the central database at any given time, and up to 500 simultaneous users against the local servers at any one time.
    2. The system shall provide access to the legacy course catalog database with no more than a 10 second latency.
    3. The system must be able to complete 80% of all transactions within 2 minutes.
    4. The client portion shall require less than 20 MB disk space and 32 MB RAM.

The selected architecture supports the sizing and timing requirements through the implementation of a client-server architecture. The client portion is implemented on local campus PCs or remote dial up PCs. The components have been designed to ensure that minimal disk and memory requirements are needed on the PC client portion.

**9.****Quality [http://www.ecs.csun.edu/~rlingard/COMP684/Example2SoftArch_files/top.gif](http://www.ecs.csun.edu/~rlingard/COMP684/Example2SoftArch.htm#Toc)**

The software architecture supports the quality requirements, as stipulated in the Supplementary Specification [15]:

* + 1. The desktop user-interface shall be Windows 95/98 compliant.
    2. The user interface of the C-Registration System shall be designed for ease-of-use and shall be appropriate for a computer-literate user community with no additional training on the System.
    3. Each feature of the C-Registration System shall have built-in online help for the user. Online Help shall include step by step instructions on using the System. Online Help shall include definitions for terms and acronymns.
    4. The C-Registration System shall be available 24 hours a day, 7 days a week. There shall be no more than 4% down time.
    5. Mean Time Between Failures shall exceed 300 hours.
    6. Upgrades to the PC client portion of C-Registration shall be downloadable from the UNIX Server over the internet. This feature enables students to have easy access to system upgrades.