

Horse Racing Simulation System

Prepared by

Ting Hin Chau
PTMsc Csc
Department of Computer Science
The Chinese University of Hong Kong

Supervised by
Professor Michael R. Lyu
Department of Computer Science
The Chinese University of Hong Kong

2004-04-28

| | |
|--|----|
| Abstract | 1 |
| 1 Introduction..... | 1 |
| 2 System Overview | 3 |
| 2.1 Object Request Broker (VisiBroker for Java 4.0)..... | 3 |
| 2.2 Database Server (Oracle 8i)..... | 3 |
| 2.3 Server Objects..... | 3 |
| 2.4 Client Objects..... | 4 |
| 3 Functional specification | 6 |
| 3.1 Jockey Club..... | 6 |
| 3.2 Stable..... | 28 |
| 3.3 Gambler..... | 34 |
| 4 System Design | 43 |
| 4.1 Component Interaction..... | 43 |
| 4.2 Objects | 44 |
| 4.3 Database..... | 59 |
| 5 User Manual..... | 65 |
| 5.1 Compilation..... | 65 |
| 5.2 Installation..... | 65 |
| 5.3 Uninstallation..... | 65 |
| 5.4 Using the System | 65 |
| 6 Conclusion | 66 |
| 7 References..... | 67 |

Abstract

This horse racing simulation system demonstrates how all the processes involved in horse racing can be automated by a distributed system using the Common Object Request Broker Architecture (CORBA). The processes refer to the interactions among the three main identities in this system, namely the gamblers, the stables and the Jockey Club. A distributed system environment can offer better scalability, transparency and failure handling in this case.

1 Introduction

When people think about horse racing, people think about money. We are talking about billions of money on any given horseracing day. In the past, people need to go to the racecourse or off-course betting branches (OCB) of Jockey Club to place their bet and, if fortunate enough, take the money they win. This is not at all very convenient. With the advancement of technology, people should be given more channels to do transactions with the Jockey Club. One way is to do it electronically, and most desirably, through the Internet, which has become one of the most important ways of communication over the past few years. This not only can be advantageous to the gamblers, but also the Jockey Club as it can trim down its operating costs with these automated procedures. Given the large amount of money and the frequency of transactions involved, the chance of making an error is much bigger if everything has to be done manually. In a similar way, the dividends can be credited into the gambler's accounts instead of giving out in the form of cash. Similarly, the interactions between the stable administrators and the Jockey Club can be computerized.

To implement these services as a whole, the identities have to overcome the heterogeneity among them and be able to seamlessly interact with one another. The CORBA (Siegel, 2000) implementation would be one of the better solutions to this challenge, as it can allow different objects to be written in different programming

languages and to communicate across different hardware platforms on different machines.

The next section will be an overview of the system. It provides details about the main components in this system and depicts their relationship. Section 3 will be the functional specification of the system. The functionalities of the 3 main identities, namely the Jockey Club, the stables and the gamblers will be explained. Section 4 will be formal description of the objects in the system, the database design and the user interface. The last section will give a conclusion.

2 System Overview

The system would be built under a distributed environment. The heterogeneity will be masked by CORBA implementation. Client objects and the servers can lie on different machines from within the local network or outside the local network. They communicate through invocation of the methods defined in the interfaces of one another.

2.1 Object Request Broker (VisiBroker for Java 4.0)

The Object Request Broker enables CORBA objects to communicate with each other. The ORB connects objects requesting services to the objects providing them by performing object lookup and instantiation. VisiBroker for Java 4.0 will be used in this project. It provides a complete CORBA 2.3 runtime that supports development environment for building, deploying, and managing distributed Java applications (Farley, 1998) to achieve openness, flexibility, and inter-operability. Objects built with VisiBroker for Java can be accessed by Web-based applications that communicate using OMG's Internet Inter-ORB Protocol (IIOP) standard for communication between distributed objects through the Internet or through local intranets. All machines containing the client Java applets and the server objects have to be installed with VisiBroker for Java 4.0 to enable communication.

2.2 Database Server (Oracle 8i)

The database server provides a persistent storage of data. Information of horses, stables, gamblers, races, as well as the bets will be stored in the database. Apart from providing storage, the database manager built into Oracle 8i will also be responsible for maintaining data integrity during concurrent accesses.

2.3 Server Objects

The server objects provide all the implementation of the services requested by the clients. They also interact with the database server to store and retrieve

transaction information. The functionalities of each server object are exposed to clients through the interface it is implementing. As CORBA implementation will be adopted in this project, the interface will be defined as standard CORBA/IDL (IDL is the abbreviation of Interface Definition Language). The server objects will be implemented in Java in this project. The server objects, however, will not communicate with the client objects. Instead, as stated in above sections, they communicate through the CORBA Object Request Broker.

2.4 Client Objects

The client objects are Java applets that request services from the server objects. All the communication will go through the Object Request Broker, which will map the server names to remote objects. The client objects will not directly interact with the database. For this system, the clients exist from within the Intranet for performance and demonstration purposes. The clients are divided into three categories. The first category is race administration clients from Jockey Club. They are responsible for functions carried out by the staff of the Jockey Club. The functions include race schedule setting, allowing bets, starting races and paying out dividends. The second category is stable administration clients. The functions they carry out include horse management, and race registration with the Jockey Club for their horses. The last category is the gamblers. With the client application, the gamblers can perform betting account management transactions, queries on horses and races, as well as betting on a horse of a race.

The following diagram (Figure 1) depicts graphically the interaction between the various components described above.

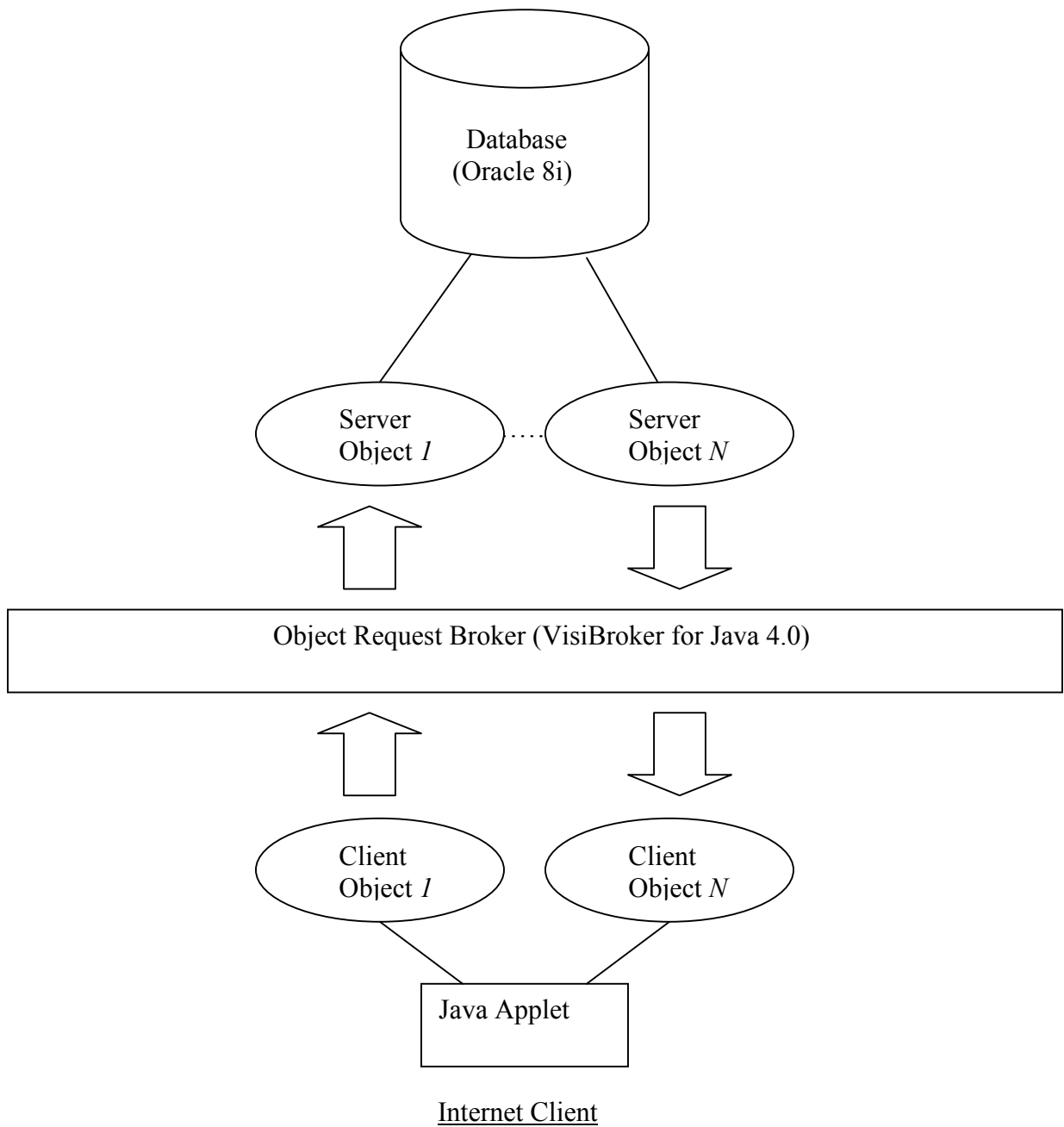


Figure 1: The interaction among components.

3 Functional specification

The section will describe the functionalities of each of the 3 identities, Jockey Club, stables and gamblers. Moreover, the business model of the Jockey Club will be explained together with the assumptions of this project.

3.1 Jockey Club

It is responsible for provision of racing games by interacting with both stables and the gamblers. It administers the schedule of each race, accepts bets, starts a race and pays out dividends after the race is over. In this project, the result will be simulated based on simple Monte Carlo algorithm using historical ranks of the horses.

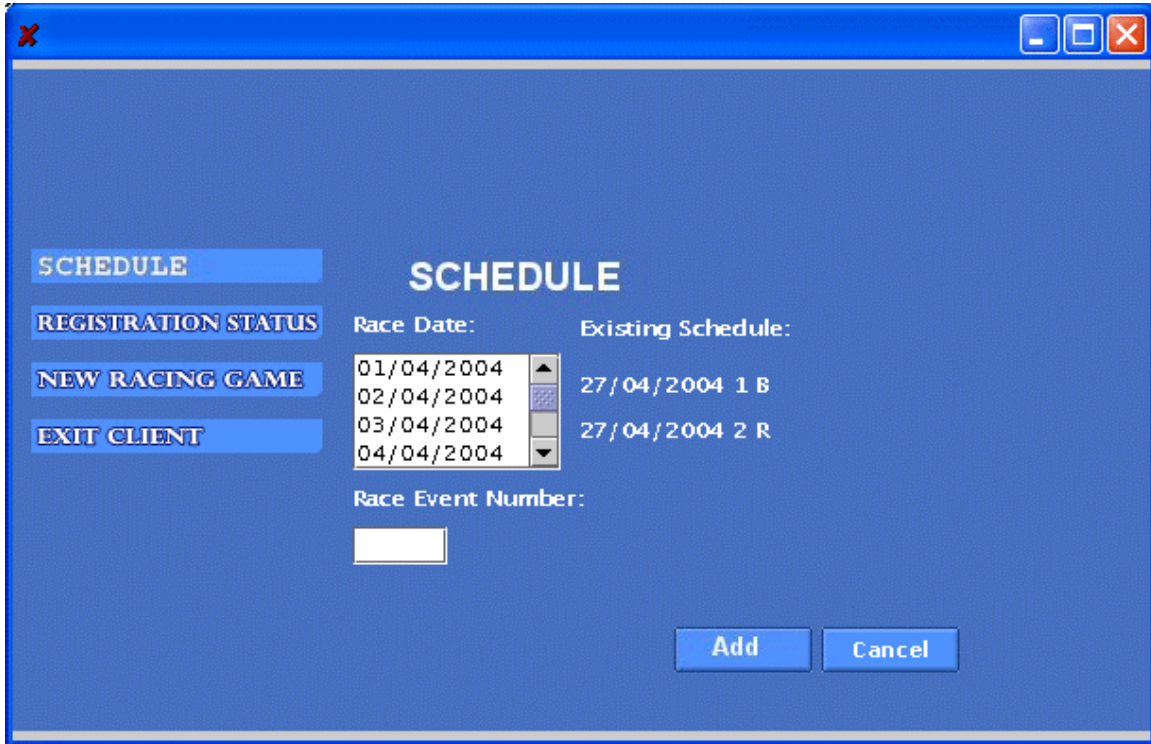
3.1.1 Major responsibilities and functionalities

1) Formulating Horseracing Schedule

This function will be requested by the race administration client application. The Jockey Club first decides on which dates horseracing will be taking place. Apart from the dates, it also determines how many races there will be on each horseracing day. In an attempt to reduce the system resource requirement of this project, the race administrator can only set the schedule day by day with the date equals to the current date. When the race administrator wants to set a date as horseracing day, the horseracing date and the race number have to be inputted. Each race has to be inputted separately. Figure 2 shows the screen for setting race schedule. Existing schedule will also be shown on this page. Existing horseracing date, race event number and its respective status will be shown. The 4 statuses of a race are 'R', which stands for horse registration period, 'B', which stands for

betting period, 'D', which stands for race over, and 'P', which mean all dividends have been paid out.

Figure 2: The screen for setting race schedule. (Race Administration Client)



In reality, under the rules set when the Jockey Club was founded, the horseracing dates are confined to Wednesdays, Saturdays, Sundays or other public holidays of Hong Kong Special Administrative Region (HKSAR). However, this limitation is relaxed in this project.

2) Stable Account Registration

This function will be requested by the stable administration client application. It allows stables to register itself with the Jockey Club to make it eligible for registering their horses and jockeys for a race. A stable identity number and a password will be assigned to

each registered stable. Upon registration, the Jockey Club can query information about the horses and jockeys of the stable.

3) Stable Account Removal

This function will be requested by the stable administration client application. It allows stables to cancel its registration with the Jockey Club. This would make them ineligible for registering their horses and jockeys for a race. Upon successful account cancellation, the gamblers can no longer query information about the horses of the stable.

4) Information Query by Stable

This function will be requested by the stable administration client application. It allows stables to ask information about the horseracing schedule.

5) Processing Horse Registration for a Race by Stable

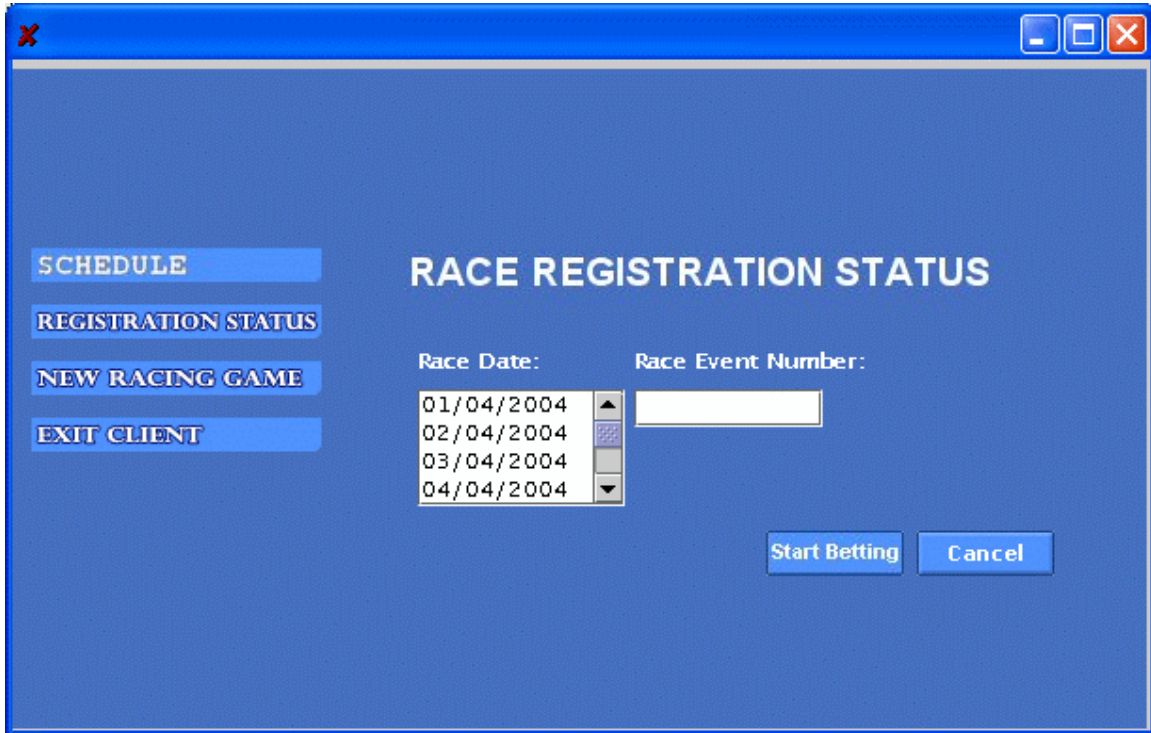
This function will be requested by the stable administration client application. It allows them to register their horses for a race on a valid horseracing day. No horse can register twice for any race, as it is not possible to do so. The maximum number of horses in a race is 14. The Jockey Club will assign the horses to different lanes sequentially, with the first one registered assigned to lane 1.

6) Changing a race status from registration to betting mode

This function will be requested by the race administration client application. Only races with 4 or more starters are valid and will be open to public for placing bets. In this project, registration has to be done on the same day as the horseracing day. Once, the mode changes to betting mode, no more horse can be registered for

that particular race. Figure 3 shows the screen for changing the status of a race from registration to betting mode.

Figure 3: The screen for changing a race from registration to betting mode. (Race Administration Client)



7) Betting Account Registration

This function will be requested by the gambler client application. It allows gamblers to register with the Jockey Club and thereby opening a betting account. To successfully open an account, gamblers are required to provide a banking account as the source of money for the betting account. In this project, there will be no checking on the validity of the banking account and its balance. Therefore, all transfer from banking account to betting account will be treated as successful if the input amount is a valid number. An identity number and a password will be assigned to each registered

gambler. The name of the gambler will also be stored and displayed.

8) Betting Account Removal

This function will be requested by the gambler client application. It allows gamblers to cancel its registration with the Jockey Club. This would make them ineligible for betting on games through this system.

9) Gambler Profile Update

This function will be requested by the gambler client application. It allows gamblers to update their registered name, registered bank account and their password.

10) Accepting Money Deposit

This function will be requested by the gambler client application. It can handle money deposit from the bank account associated with the gamblers' account to the betting account.

11) Race Query by Gamblers

This function will be requested by the gambler client application. It allows gamblers to ask information about a racehorse of a particular race. The information includes the stable ID, horse ID, the win odds and the place odds range will be shown.

12) Accepting bets from Gamblers

This function will be requested by the gambler client application. It accepts bets from gamblers. When a race is in betting mode, gamblers can bet on it. As a result of this operation, money will be deducted from the account of the gambler. There will be two types of games in this project. The first one is called 'Win' and the

second one is called 'Place'. These two games will be elaborated in the next section 3.1.2.

13) Calculating Odds

Every time a new bet is being placed on a particular horse, the Jockey Club will recalculate the odds in the same pool of all the horses participating in the same race. A detailed description of the calculation will be described in the next section 3.1.2.

14) Starting a Race

This function will be requested by the race administration client application. The Jockey Club is responsible to cut off all betting and start a race. In this project, there will not be a complete simulation of the whole racing process, where horses overtake one another over the course of the race. After the Jockey Club starts a race, the result will then be generated based on Monte Carlo Simulation using historical ranks of each participating horse. A detailed description of the simulation will be described in the next section 3.1.2. Figure 4 shows the screen for starting a race. Figure 5 shows the screen for the results of a race. Only the top 3 winners will be shown.

Figure 4: The screen for starting a race. (Race Administration Client)

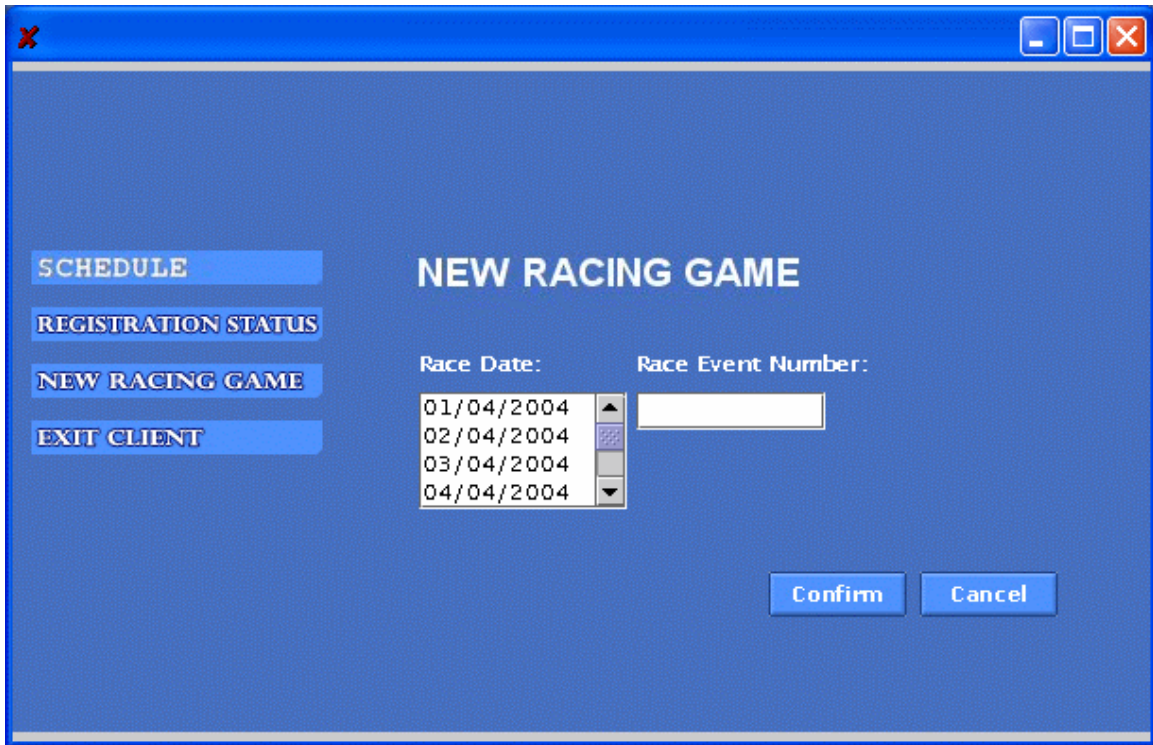
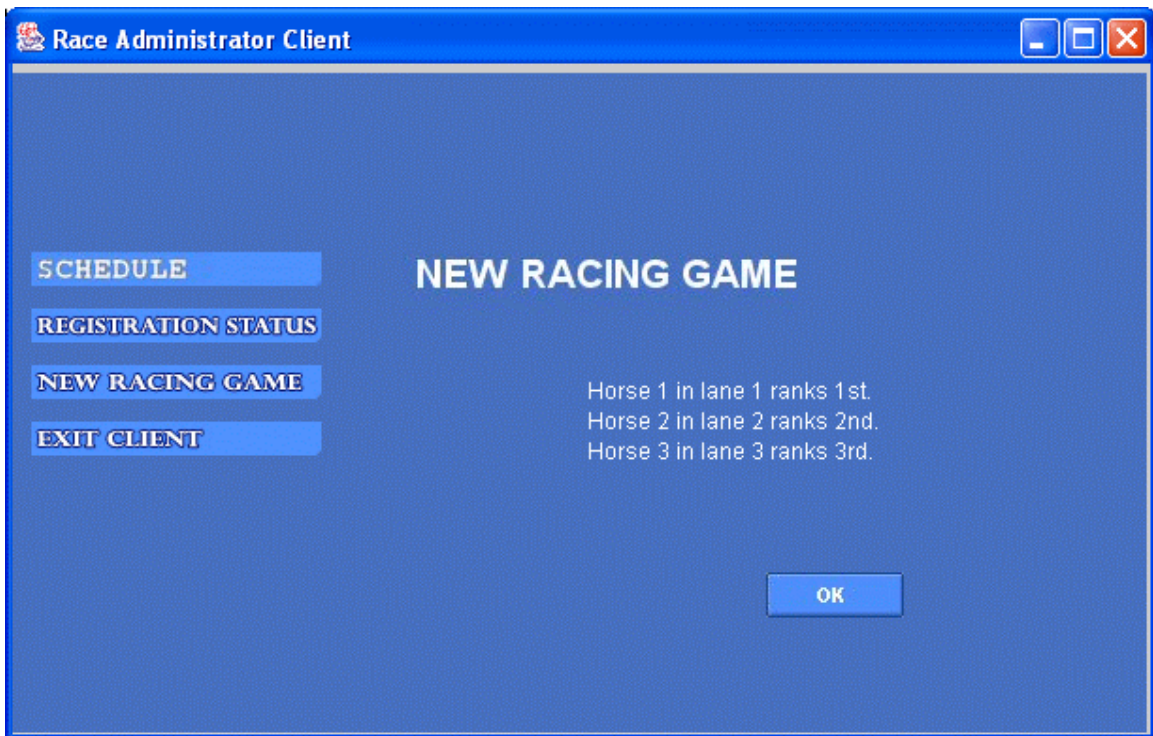


Figure 5: The screen for the results of a race. (Race Administration Client)



15) Saving the Results into a Database

The results of a race will be saved into a database by the Jockey Club after each race. No explicit request is needed. These results will be used for paying out dividends, as well as inputs for future race simulation.

16) Paying out Dividends to Gamblers

Concerning the interaction with the gamblers, it pays out dividends to gamblers if their bets win after each race. As a result of this operation, money will be credited to the betting account of the gambler. The exact amount of dividends payable to gamblers depends on the final odds of each horse under various betting types.

3.1.2 Types of Bet

1) Win

With “Win”, a gambler picks one horse and if it turns out to be the winner (1st), the gambler will be entitled to receive dividend, otherwise, the gambler loses.

2) Place

With “Place”, the gambler picks one horse and if it is the winner (1st), the first runner-up (2nd), or second runner-up (3rd), the gambler will be entitled to receive dividend. Otherwise, the gambler loses.

3.1.3 Calculation of Odds

The calculation assumes that the Jockey Club makes no profit out of the pool. Please note that all odds will be initialized to 99.

1) Win

As stated in (3.1.2.1), only gamblers that have placed bets on the eventual winner (1st) will win and all other gamblers in this pool lose money. From the standpoint of the Jockey Club, whenever a bet b_w is placed on a horse, the Jockey Club will reduce its odds to make sure that it will break even.

Suppose b_w is the amount of bet on horse w , $\sum b_i$ is the total amount of bets and d_w is the win odds of the horse.

$$\text{Dividend payout} = \text{Amount of bets on winner} * \text{odds of winner}$$

$$\text{Revenue} = \text{Total amount of bets}$$

$$\text{Net gain} = 0$$

$$- b_w * d_w + \sum b_i = 0$$

$$d_w = \sum b_i / b_w \quad \text{----- (1)}$$

Every time a new bet is being placed on any horse, the odds of all horses changes according to equation (1).

To illustrate, suppose there are 4 horses, and let b_1, b_2, b_3, b_4 be the bets on the 4 horses respectively and let d_1, d_2, d_3, d_4 be their respective odds.

Four scenarios:

$$1) -b_1 * d_1 + b_1 + b_2 + b_3 + b_4 = 0$$

$$2) -b_2 * d_2 + b_1 + b_2 + b_3 + b_4 = 0$$

$$3) -b_3 * d_3 + b_1 + b_2 + b_3 + b_4 = 0$$

$$4) -b_4 * d_4 + b_1 + b_2 + b_3 + b_4 = 0$$

Suppose $b_1 = 100, b_2 = 200, b_3 = 400$ and $b_4 = 500$, then

$$d_1 = (100 + 200 + 400 + 500) / 100 = 12$$

$$d_2 = (100 + 200 + 400 + 500) / 200 = 6$$

$$d_3 = (100 + 200 + 400 + 500) / 400 = 3$$

$$d_4 = (100 + 200 + 400 + 500) / 500 = 2.4$$

2) Place

With “Place”, the gambler picks one horse and if it is the winner (1st), the first runner-up (2nd), or second runner-up (3rd) in a race, the gambler will be entitled to receive dividend. Otherwise, the gambler loses. Again, we assume the Jockey Club does not make any profit out of the pool.

Suppose d_{w1} is the place odds of winner $w1$, $\sum b_i$ is the total amount, b_{w1} is the amount of bet on winner $w1$, b_{w2} is the amount of bet on winner $w2$ and b_{w3} is the amount of bet on winner $w3$.

$$\begin{aligned} \text{Dividend payout} &= \text{Amount of bets on winners} \\ &\quad * \text{their respective odds} \end{aligned}$$

$$\text{Revenue} = \text{Total amount of bets}$$

$$\text{Net gain} = 0$$

The equation for place odds is given by:

$$d_{w1} = (\sum b_i - b_{w1} - b_{w2} - b_{w3}) / (3 * b_{w1}) + 1$$

$$d_{w2} = (\sum b_i - b_{w1} - b_{w2} - b_{w3}) / (3 * b_{w2}) + 1$$

$$d_{w3} = (\sum b_i - b_{w1} - b_{w2} - b_{w3}) / (3 * b_{w3}) + 1$$

As the place odds of a horse depends on the amounts of bets of the other 2 winners, before the result of a race is known, there are many possible place odds of a horse. Instead of calculating all those possible place odds every time a gambler places a bet and showing all those possible odds, the Jockey Club will use a range to describe the place odds. Given a total amount of place bets on all the horses, the place odds of a horse will be largest possible if the amounts of bets of the two other horses are the smallest among all the other place bets. Similarly, if the place odds of a horse will be the smallest possible if the amounts of bets of the two other horses are the largest among all the other place bets. To illustrate, again, suppose there are 4 horses, and let b_1, b_2, b_3, b_4 be the bets on the 4 horses respectively and let d_1, d_2, d_3, d_4 be their respective odds. Also, we assume $b_1 < b_2 < b_3 < b_4$

The smallest possible value of d_1 is when horse 3 and horse 4 are the two other winners, as b_3, b_4 are the largest among all other place bets. The largest possible value of d_1 is when horse 2 and horse 3 are the two other winners, as b_2, b_3 are the smallest among all other place bets.

Suppose $b_1 = 100, b_2 = 200, b_3 = 400$ and $b_4 = 500$, then

$$\begin{aligned} d_{w1\text{smallest}} &= (\sum b_i - b_{w1} - b_3 - b_4) / (3 * b_{w1}) + 1 \\ &= (b_1 + b_2 + b_3 + b_4 - b_1 - b_3 - b_4) / (3 * b_1) + 1 \\ &= 200/300 + 1 = 1.66 \end{aligned}$$

(All odds are rounded down to 2 decimal places.)

$$d_{w1\text{largest}} = (\sum b_i - b_{w1} - b_{w2} - b_{w3}) / (3 * b_{w1}) + 1$$

$$\begin{aligned}
 &= (b_1 + b_2 + b_3 + b_4 - b_1 - b_2 - b_3) / (3 * b_1) + 1 \\
 &= 500/300 + 1 = 2.66
 \end{aligned}$$

Therefore, before the race results come out, the place odds of horse 1 will be specified by the range 1.66 – 2.66.

$$\begin{aligned}
 d_{w2smallest} &= (\sum b_i - b_{w2} - b_3 - b_4) / (3 * b_{w2}) + 1 \\
 &= (b_1 + b_2 + b_3 + b_4 - b_2 - b_3 - b_4) / (3 * b_2) + 1 \\
 &= 100/600 + 1 = 1.16
 \end{aligned}$$

$$\begin{aligned}
 d_{w2largest} &= (\sum b_i - b_{w1} - b_{w2} - b_{w3}) / (3 * b_{w2}) + 1 \\
 &= (b_1 + b_2 + b_3 + b_4 - b_1 - b_2 - b_3) / (3 * b_2) + 1 \\
 &= 500/600 + 1 = 1.83
 \end{aligned}$$

Therefore, before the race results come out, the place odds of horse 2 will be specified by the range 1.16 – 1.83.

The respective range for place odds of horse 3 and horse 4 can be calculated in similarly.

3.1.4 Horseracing Simulation Model

This simulation model (Ross, 1989) is responsible for generating results for each race. Monte Carlo simulation (Bickel and Doksum, 1977) will be employed to perform the above task. In Monte Carlo simulation, samples can be drawn from a given distribution (Chung, 1998) using a random number and the distribution concerned as input. It is used to imitate a random sampling from the input distribution (Hogg and Craig, 1970). In this case, the distribution will be the historical statistics of a horse (Wagner, 1969). These statistics can be relieved from the official website of the Jockey Club (<http://www.hongkongjockeyclub.com>).

Instead of applying a theoretically derived parametric distribution like Exponential distribution or Lognormal distribution (Conover and Whitten, 1980) to the statistics of the horses, nothing will be assumed about the type of distributions (Hastings, 1975). General non-parametric distribution will be used, which means that the mathematics of a distribution will be defined by its range, the number of classes and the frequency of each class. In this case, the data we have are previous rankings of a horse in various races. The range of the rankings will be from 1 to 14 and each integer from 1 to 14 represents a distinctive class. From this raw frequency distribution, we generate the probability distribution (Bratley, Fox, and Schrage, 1987) of this horse. (Figure 6) $P(X = x)$ is the probability P that the variable ranking (X) of the horse is equals to a certain value x .

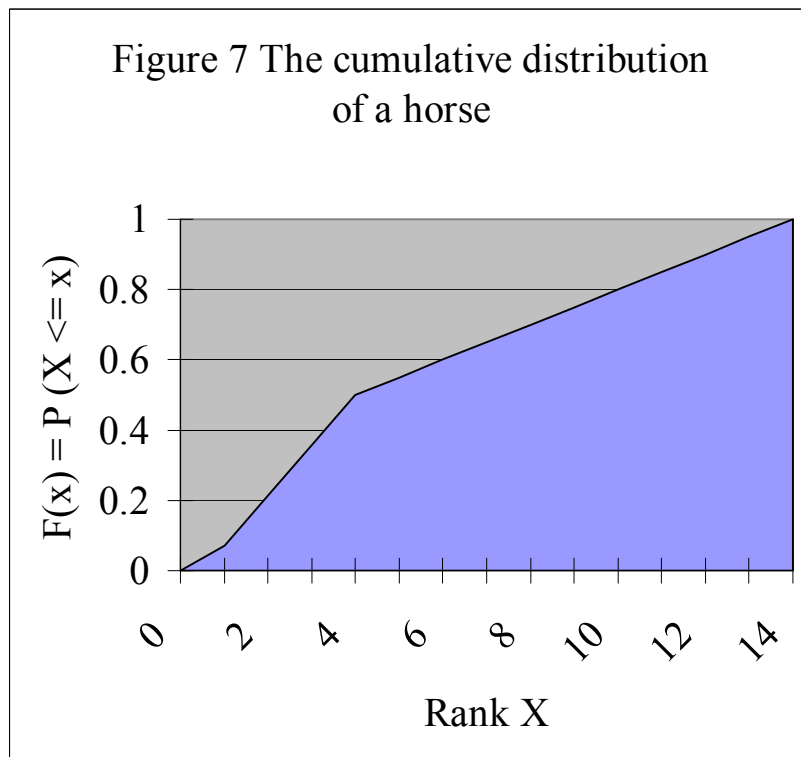
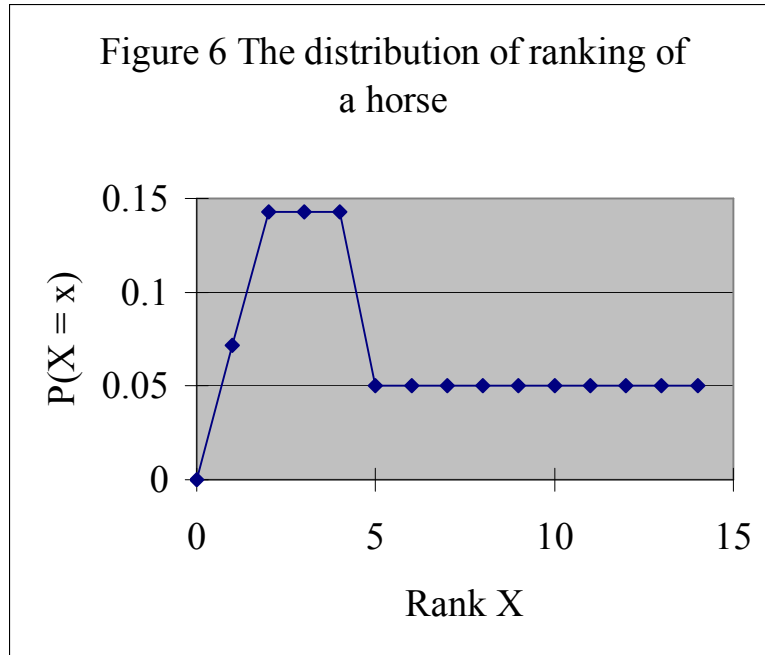


Figure 7 shows the cumulative distribution (Kendall, Stuart, and Ord, 1987) derived from the probability distribution in Figure 6. $F(x)$ gives the

probability P that the variable ranking X is less than or equal to x . Mathematically, the relationship can be described as $F(x) = P(X \leq x)$. (Vose, 1966) For example, $F(5)$ refers to the probability of the ranking of a horse to be less than 5 or equal to 5. In this project, rankings less than 5 or equal to 5 include 1, 2, 3, 4 and 5. It does not elaborate into the idea of winning and losing.

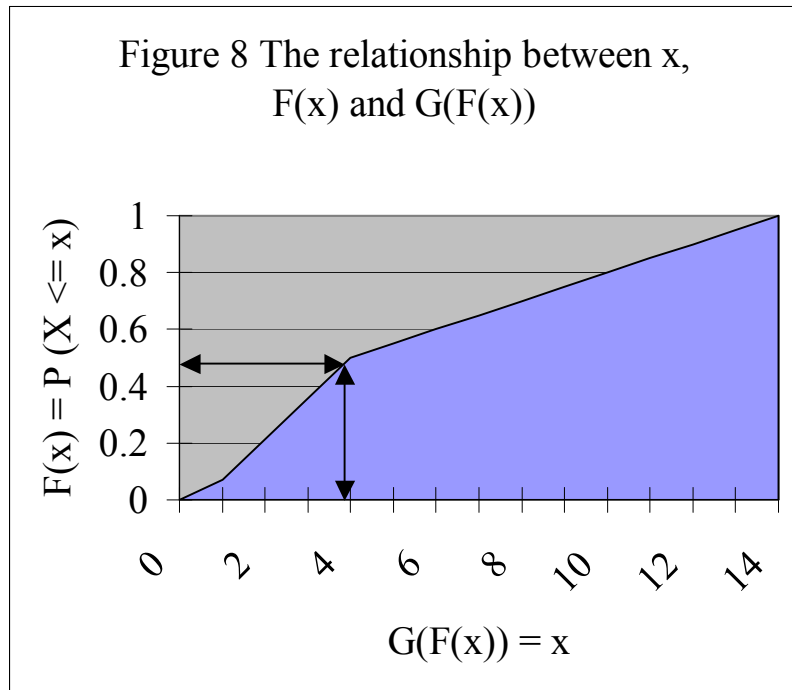
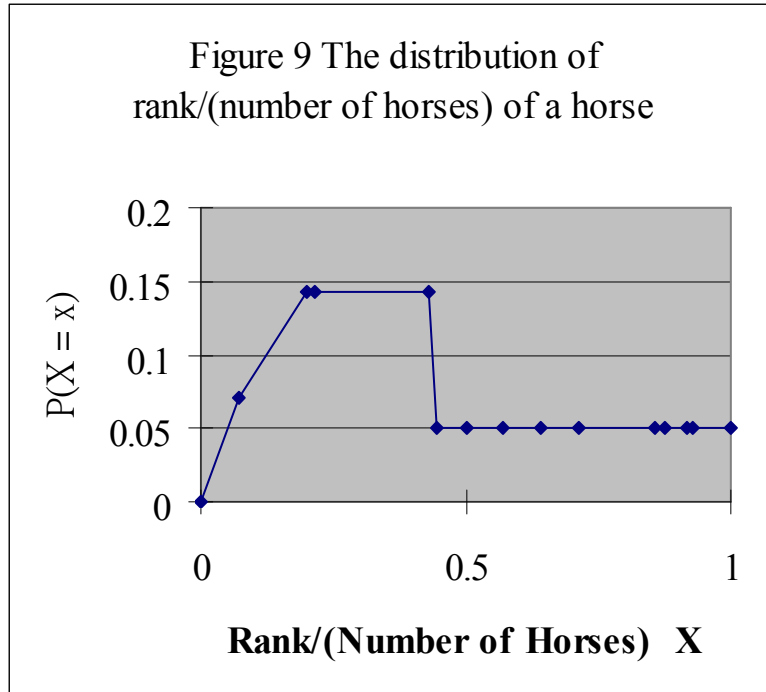


Figure 8 in fact represents the same data sets as Figure 7. It simply highlights the relationship between x , $F(x)$ and $G(F(x))$. (Young, 1992) As stated above, given x , the value of $F(x)$ can be found. $G(F(x))$ is the inverse function where $G(F(x)) = x$. This function is crucial in Monte Carlo Simulation (Law and Kelton, 1991) because if a number from 0 to 1 is randomly generated and fed into the function $G()$, value of ranking X can be randomly sampled based on its distribution as a result. It is worth to note that the value of X does not have to be an integer, as it does not represent the final ranking. Every horse in the same race will have its own ranking distribution. Each will generate a random value X_i . The value of

Xs will be compared and sorted. The horse with the smallest value of ranking is the winner. This is the final ranking of the horse. The one with the largest value of X will be assigned to the last position of the race. In this way, the result of a race can be generated. As opposed to the discrete distribution of rankings of a horse, the intermediate 'ranking' X sampled has a continuous distribution. The final ranking, which will be obtained by comparing the values of Xs, will be discrete in nature, meaning that it can only take one of a set of identifiable values. In this case, the value ranges from 1 to the number of starters in that race and the maximum is 14.

However, the above model presents a problem given the rules of the Jockey Club. As there can be 4 to 14 horses in a race, the rank of a race can be misleading. After all, it should be easier to rank 4th in a race with 4 horses than a race with 14 horses! Therefore, instead of adopting the model described above directly, some modifications have to be made. Another other factor, which is the total number of horses participated in a race, has to be taken into account. The input distribution will be changed from rank to rank divided by the total number of horses. For instance, if a horse ranks 4th in a race with 8 horses, the number will be 4 divided by 8 which is 2. It can be seen that if a horse ranks 7th in a race with 14 horses, the number will be again 2. This number depicts the relative position of a horse instead of absolute position of a horse in a race. After this transformation, a frequency distribution of rank/(number of horses) of each horse will be obtained. The rest of the steps are just the same as described before. Figure 9 shows the distribution of rank/(number of horses).



In the process of randomly sampling from the distribution of a horse, the model being used in this project makes the following assumptions:

- 1) The distribution of a horse is independent of other horses. The distributions are uncorrelated.
- 2) The ranking of a horse in a race only depends on historical data and the random number generated. Factors including location of racecourse, lane, and jockey are insignificant in determining estimating the 'future' rankings of a horse. The first two assumptions allow each horse to generate an intermediate value for X without the knowledge of their opponents. The simulation procedure only depends on its historical data and the random number generated.
- 3) The randomness of the above sampling method means that it will over-sample and under-sample from various parts of the distribution and cannot be relied upon to replicate the input distribution's shape unless a very large number of iterations are performed. As this project is concerned about horseracing, which allows for huge upsets, this randomness is assumed to be acceptable.
- 4) The data from the 10 most recent races a horse participated in will be used as the input distribution frequency. If the horse has not participated in 10 races, the missing values of rank/(number of horses) will be randomly generated within the range 0.45 to 0.55. This implies that if there are insufficient data about the horse, the horse will be perceived to perform around average in those missing races.

To illustrate the above model, consider there is a race with 4 horses with the following input data. R refers to rank, N refers to number of horses in that race and P refers to relative position of the horse and equals R/N. Please note that horse 3 only has participated in 9 races before and horse 4 has only participated in 8 races before. Missing values are randomly generated within the range 0.45 to 0.55.

Table 1: Historical Ranks of Participating Horses

| Race | Horse 1 | | | Horse 2 | | | Horse 3 | | | Horse 4 | | |
|------|---------|----|---------|---------|----|---------|---------|-----|---------|---------|-----|---------|
| | R | N | R/N | R | N | R/N | R | N | R/N | R | N | R/N |
| 1 | 2 | 8 | 0.25 | 4 | 10 | 0.4 | 3 | 10 | 0.3 | 5 | 10 | 0.5 |
| 2 | 3 | 10 | 0.3 | 6 | 12 | 0.5 | 2 | 8 | 0.25 | 6 | 11 | 0.54545 |
| 3 | 1 | 14 | 0.07143 | 5 | 10 | 0.5 | 3 | 12 | 0.25 | 6 | 12 | 0.5 |
| 4 | 1 | 9 | 0.11111 | 4 | 8 | 0.5 | 4 | 11 | 0.36364 | 8 | 14 | 0.57143 |
| 5 | 4 | 14 | 0.28571 | 4 | 10 | 0.4 | 3 | 9 | 0.33333 | 12 | 12 | 1 |
| 6 | 3 | 12 | 0.25 | 7 | 14 | 0.5 | 1 | 7 | 0.14286 | 13 | 14 | 0.92857 |
| 7 | 2 | 8 | 0.25 | 8 | 12 | 0.66667 | 1 | 8 | 0.125 | 8 | 14 | 0.57143 |
| 8 | 1 | 10 | 0.1 | 8 | 14 | 0.57143 | 5 | 12 | 0.41667 | 6 | 10 | 0.6 |
| 9 | 1 | 10 | 0.1 | 6 | 10 | 0.6 | 4 | 10 | 0.4 | N/A | N/A | 0.5355 |
| 10 | 2 | 8 | 0.25 | 5 | 8 | 0.625 | N/A | N/A | 0.4805 | N/A | N/A | 0.5010 |

As the total number of data sets input for each horse is 10, the probability of occurring for each R/N value is 0.1. Also, Table 1 can be reduced to Table 2 because only the values of R/N will be used for calculation. For calculation purposes, the R/N values are sorted in ascending order.

Table 2: Cumulative Probability Distribution of (R/N) of Participating Horses

| Cumulative Probability (R/N) | Horse 1 R/N | Horse 2 R/N | Horse 3 R/N | Horse 4 R/N |
|------------------------------|-------------|-------------|-------------|-------------|
| 0 | 0 | 0 | 0 | 0 |
| 0.1 | 0.071429 | 0.4 | 0.125 | 0.5 |
| 0.2 | 0.1 | 0.4 | 0.142857 | 0.5 |
| 0.3 | 0.1 | 0.5 | 0.25 | 0.501 |
| 0.4 | 0.111111 | 0.5 | 0.25 | 0.5355 |
| 0.5 | 0.25 | 0.5 | 0.3 | 0.545455 |
| 0.6 | 0.25 | 0.5 | 0.333333 | 0.571429 |
| 0.7 | 0.25 | 0.571429 | 0.363636 | 0.571429 |

| | | | | |
|-----|----------|----------|----------|----------|
| 0.8 | 0.25 | 0.6 | 0.4 | 0.6 |
| 0.9 | 0.285714 | 0.625 | 0.416667 | 0.928571 |
| 1 | 0.3 | 0.666667 | 0.4805 | 1 |

After the cumulative distribution for (R/N) for each of the participating horses has been drawn up, a random number from 0 to 1 will be generated for each of the horse as the other input for Monte Carlo Simulation. This value corresponds to the value of F(X) in Figure 8. With this random number, the corresponding R/N value will be calculated. The R/N values of each horse will then be compared with one another and horse with a lower R/N value beats out horse with a higher R/N value. In case two horses have the same score, their position will be randomly determined.

For instance, random numbers 0.5379783655654367, 0.21096296234313094, 0.02229991816731558 and 0.4920612612595172 are generated for the above four horses respectively. For horse 1, 0.5379783655654367 is in the range 0.5 to 0.6 of the cumulative probability. Two points on the cumulative distribution (x1, y1) and (x2, y2) corresponding to (0.25, 0.5) and (0.25, 0.6) will be used to calculate the final score of a horse, based on the random number generated. The corresponding R/N value can be calculated using simple linear equation. Also suppose rand1, rand2, rand3 and rand4 are the random numbers generated for horse 1, horse 2, horse 3 and horse 4 respectively.

As mentioned, for horse 1, point 1 (x1, y1) is (0.4, 0.2) and point 2 (x2, y2) is (0.5, 0.3)

R/N of horse 1

$$\begin{aligned}
 &= (\text{rand1}-y1)*((x2-x1)/(y2-y1)) + x1 \\
 &= (0.5379783655654367 - 0.2) * (0.25 - 0.25)/(0.6-0.5) + 0.25 \\
 &= 0.25
 \end{aligned}$$

For horse 2, point 1 (x1, y1) is (0.4, 0.2) and point 2 (x2, y2) is (0.5, 0.3).

R/N of horse 2

$$\begin{aligned} &= (\text{rand2}-y1)*((x2-x1)/(y2-y1)) + x1 \\ &= (0.21096296234313094 - 0.2)* (0.5 - 0.4)/(0.3-0.2) + 0.4 \\ &= 0.410962962 \end{aligned}$$

For horse 3, point 1 (x1, y1) is (0, 0) and point 2 (x2, y2) is (0.125, 0.1).

R/N of horse 3

$$\begin{aligned} &= (\text{rand2}-y1)*((x2-x1)/(y2-y1)) + x1 \\ &= (0.02229991816731558 - 0)* (0.125 - 0)/(0.1-0) + 0 \\ &= 0.027874898 \end{aligned}$$

For horse 4, point 1 (x1, y1) is (0.5355, 0.4) and point 2 (x2, y2) is (0.545455, 0.5).

R/N of horse 3

$$\begin{aligned} &= (\text{rand2}-y1)*((x2-x1)/(y2-y1)) + x1 \\ &= (0.4920612612595172 - 0.4)* (0.545455 - 0.5355)/(0.5-0.4) + 0.5355 \\ &= 0.544664699 \end{aligned}$$

After all the R/N values have been computed, they will be compared with one another to get the final rank of a horse in a race. From the above results, horse 3 will rank 1st, followed by horse 1 and then horse 2. Horse 4 will be the last one.

3.2 Stable

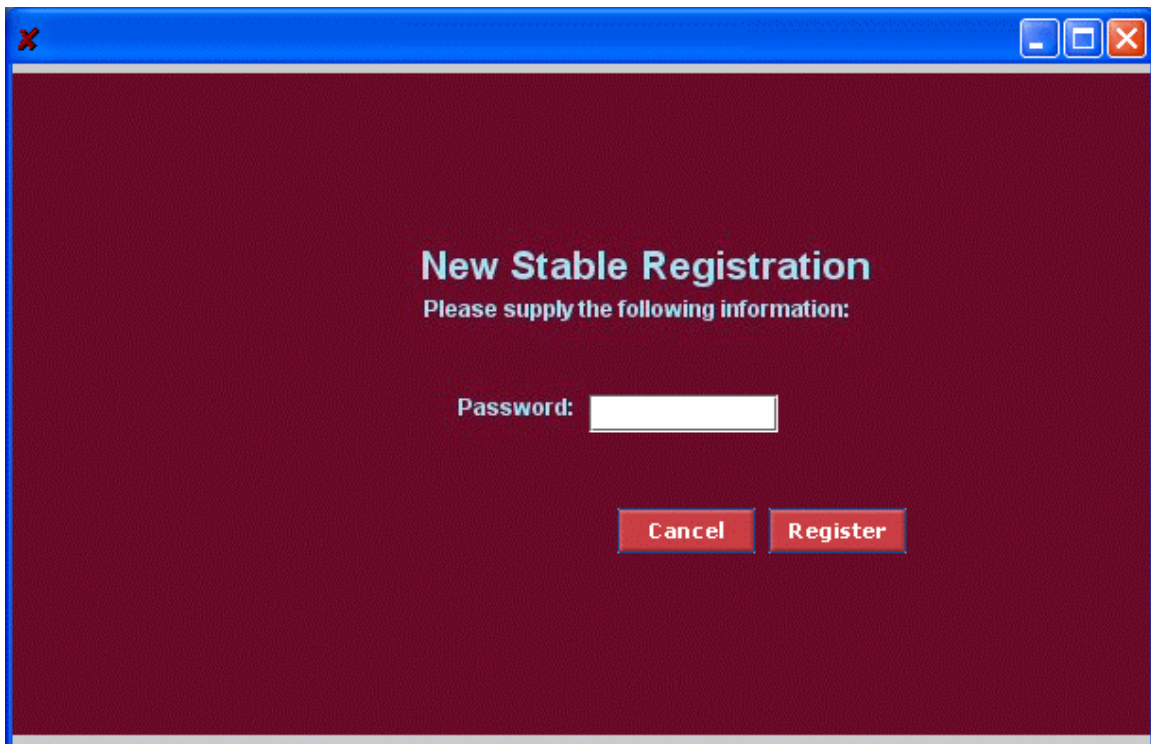
It is an entity where horses and jockeys belong. It maintains their information. It is responsible for accommodation of horses and registering them together with jockeys with the Jockey Club for a race. It is also responsible for withdrawing horses and jockeys from a race. Before registering with the Jockey Club, a stable has to open a stable account with the Jockey Club. Gamblers can query information about the horses of the stable.

3.2.1 Major functionalities and responsibilities

1) Stable Account Registration

This function will be requested by the stable administration client application. It allows stables to register itself with the Jockey Club to make it eligible for registering their horses for a race. A stable identity number will be assigned to each registered stable. Figure 10 shows the screen for stable account registration.

Figure 10: The screen for stable account registration. (Stable Administration Client)



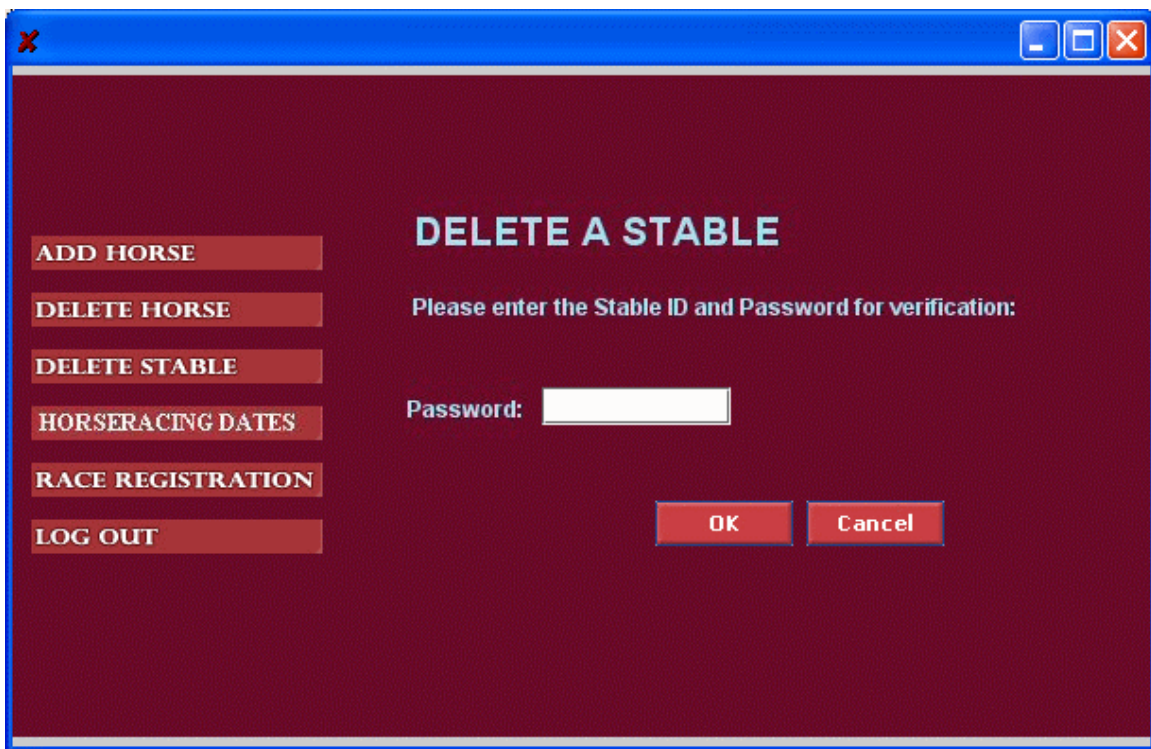
New Stable Registration
Please supply the following information:

Password:

2) Stable Account Removal

This function will be requested by the stable administration client application. It allows stables to cancel its registration with the Jockey Club. This would make them ineligible for registering their horses for a race. Upon successful account cancellation, the gamblers can no longer query information about the horses of the stable. Figure 11 shows the screen for stable account cancellation.

Figure 11: The screen for stable account cancellation. (Stable Administration Client)



3) Adding a horse to the stable

This function will be requested by the stable administration client application. It allows stables to add a horse to the stable. The gamblers can then query information about the horses of the stable. Figure 12 shows the screen for adding a horse.

Figure 12: The screen for adding a horse to a stable. (Stable Administration Client)

ADD A HORSE

Please enter the information for the horse to be added:

Horse Name:

Birthday:

Weight:

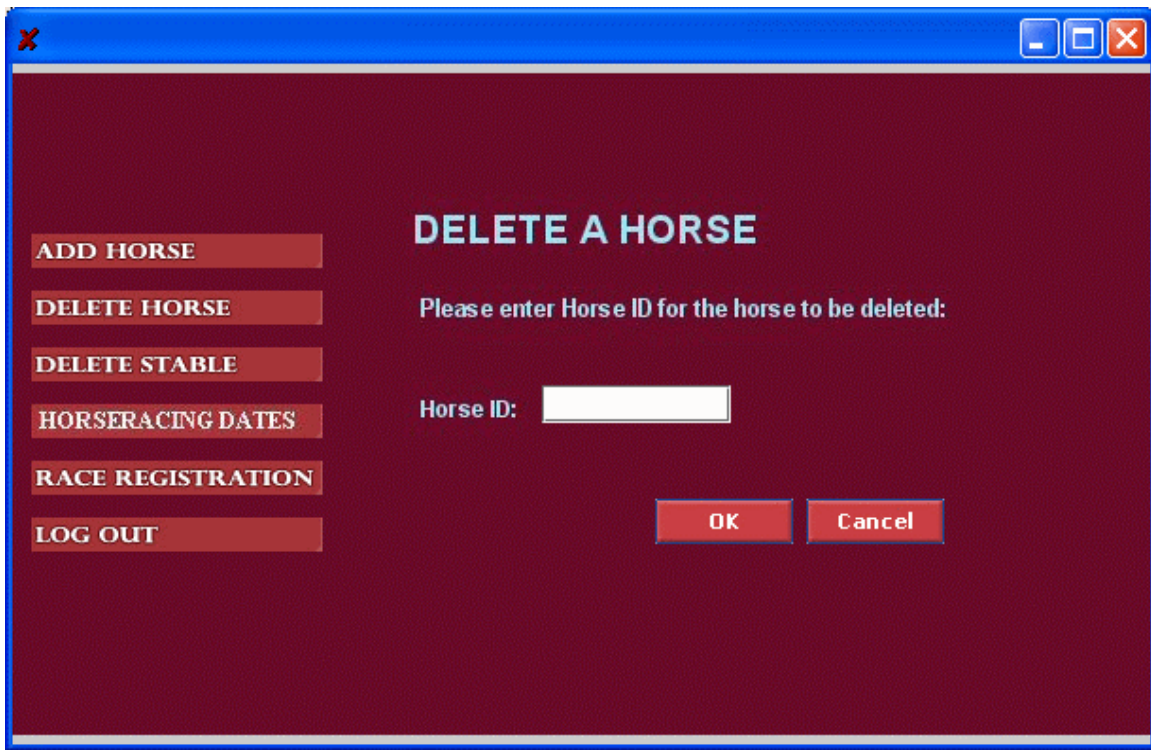
OK Cancel

4) Removing a horse from the stable

This function will be requested by the stable administration client application. It allows stables to remove a horse to the stable. Upon deletion, the gamblers can no longer query information about that horse.

Figure 13 shows the screen for removing a horse.

Figure 13: The screen for removing a horse from a stable. (Stable Administration Client)



5) Horse Query by Gamblers

This function will be requested by the gambler client application. It allows gamblers to ask information about a horse of a stable. The information includes the name, the birthday, and the weight of a horse.

6) Information Query on Current Race Schedule

This function will be requested by the stable administration client application. Stable can ask information from the Jockey Club about the horseracing schedule. If there exists any races on the current date, the races together with their respective statuses will be shown. The 4 statuses of a race are 'R', which stands for horse registration period, 'B', which stands for betting period, 'D', which stands for race over, and 'P', which mean all dividends have been paid out. Figure 14 shows the screen for querying race schedule.

Figure 14: The screen for checking schedule on current date. (Stable Administration Client)



7) Horse Registration for a Race

This function will be requested by the stable administration client application. Stables can register their horses for a race on a valid horseracing day. No horse can register twice for any race, as it is not possible to do so. Figure 15 shows the screen for horse registration for a race.

Figure 15: The screen for horse registration for a race. (Stable Administration Client)

RACE REGISTRATION

ADD HORSE
DELETE HORSE
DELETE STABLE
HORSERACING DATES
RACE REGISTRATION
LOG OUT

Race Date: 01/04/2004
Stable Password:
Horse ID:
Race Event Number:

OK Cancel

3.3 Gambler

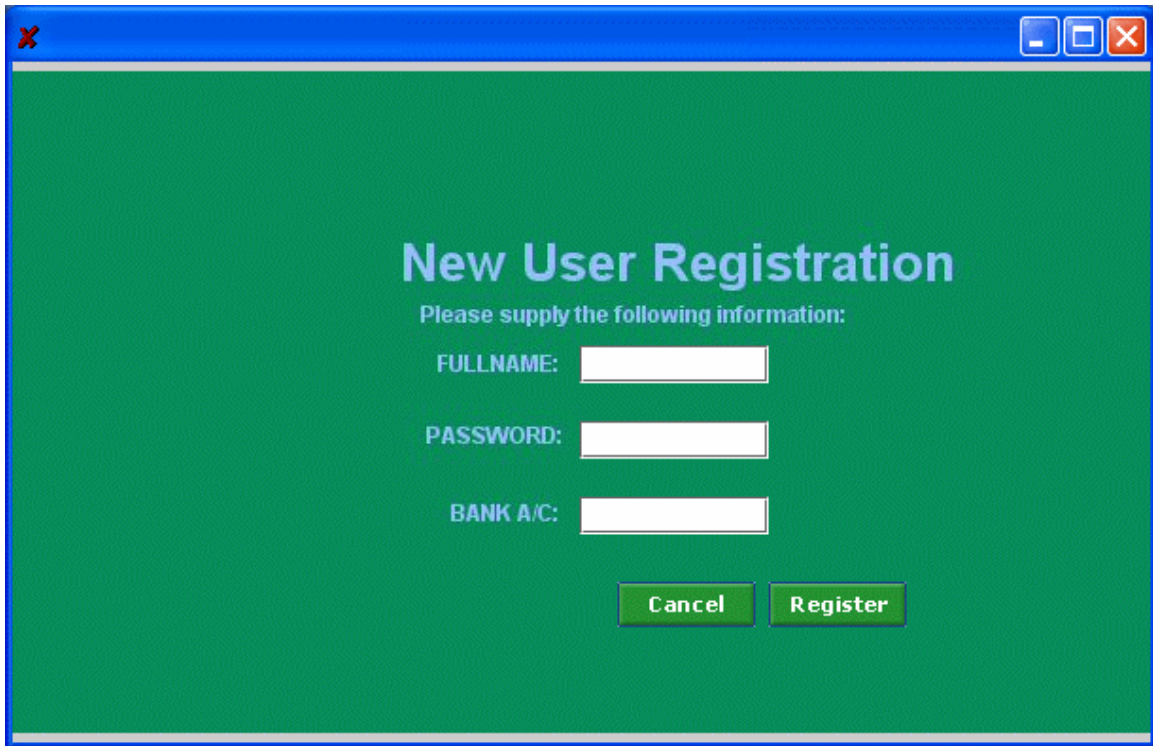
It is an entity that places bets on horses. It can query various kinds of information from the Jockey Club. It can also query information about the horses of a stable. Before betting on races, gamblers are required to register themselves with the Jockey Club to open a betting account, which will be associated with a bank account of the gambler where he/she can deposit money to the betting account. The gamblers can then place bet on a race. If the bet wins, dividends will be transferred from the Jockey Club to the betting account directly. The gamblers can also cancel their betting account with the Jockey Club.

3.3.1 Major responsibilities and functionalities

- 1) **Betting Account Registration**

This function will be requested by the gambler client application. Gamblers are required to register with the Jockey Club before they can place bet through this system. To successfully open an account, gamblers need to provide a banking account as the source of money for the betting account. An identity number will be assigned to each registered gambler. Figure 16 shows the screen for betting account registration.

Figure 16: The screen for betting account registration. (Gambler Client)



New User Registration

Please supply the following information:

FULLNAME:

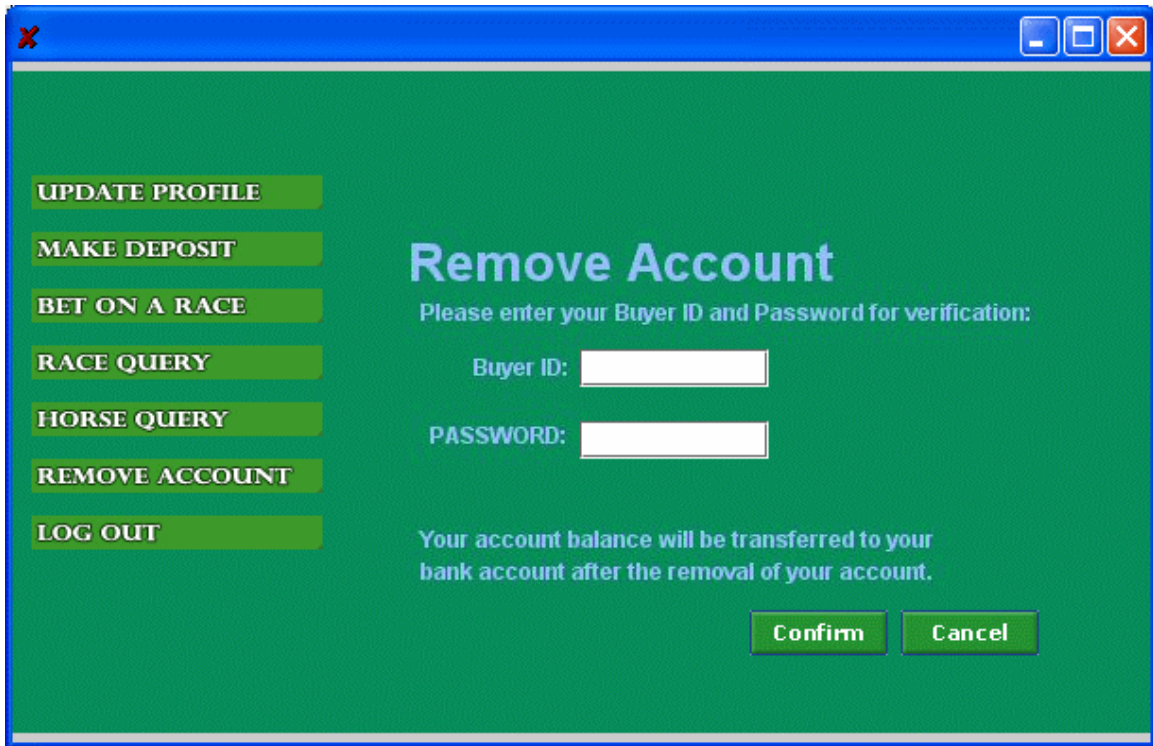
PASSWORD:

BANK A/C:

2) Betting Account Removal

This function will be requested by the gambler client application. Gamblers can cancel its registration with the Jockey Club. This would make them ineligible for betting on games through this system. Figure 17 shows the screen for betting account cancellation.

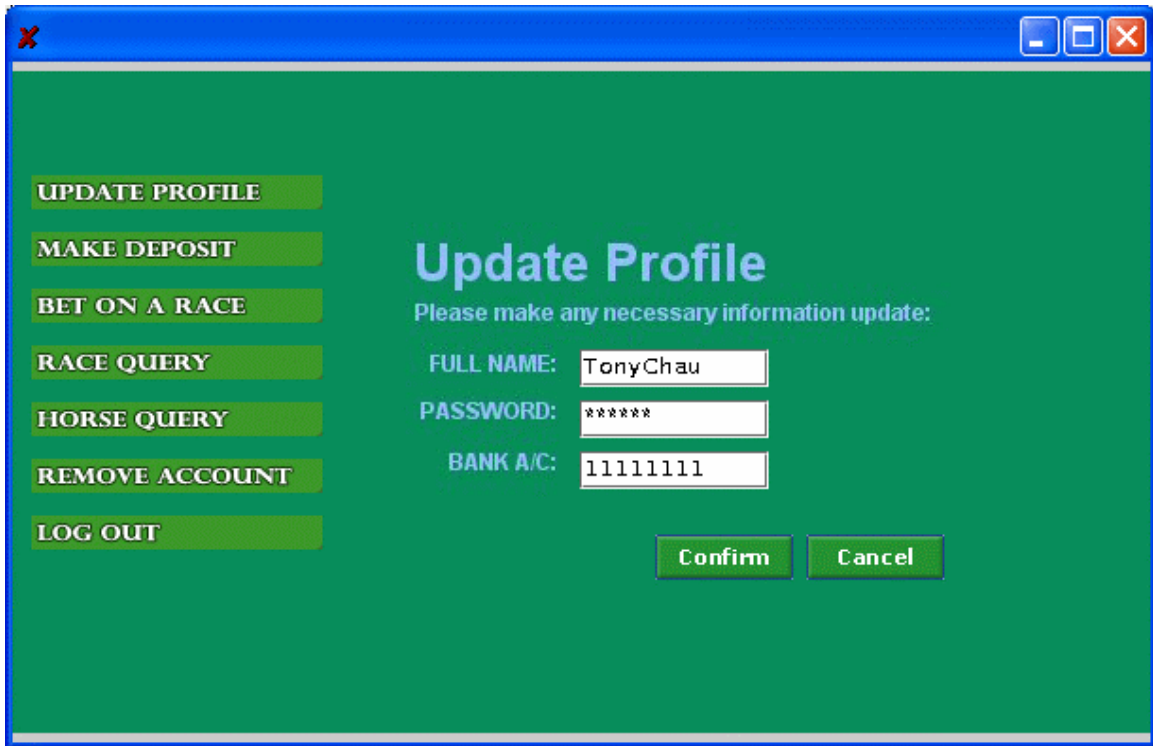
Figure 17: The screen for betting account cancellation. (Gambler Client)



3) Update Gambler Profile

This function will be requested by the gambler client application. Gamblers can update the profile associated with the betting account include the name, ID and password of the gambler. Figure 18 shows the screen for updating profile.

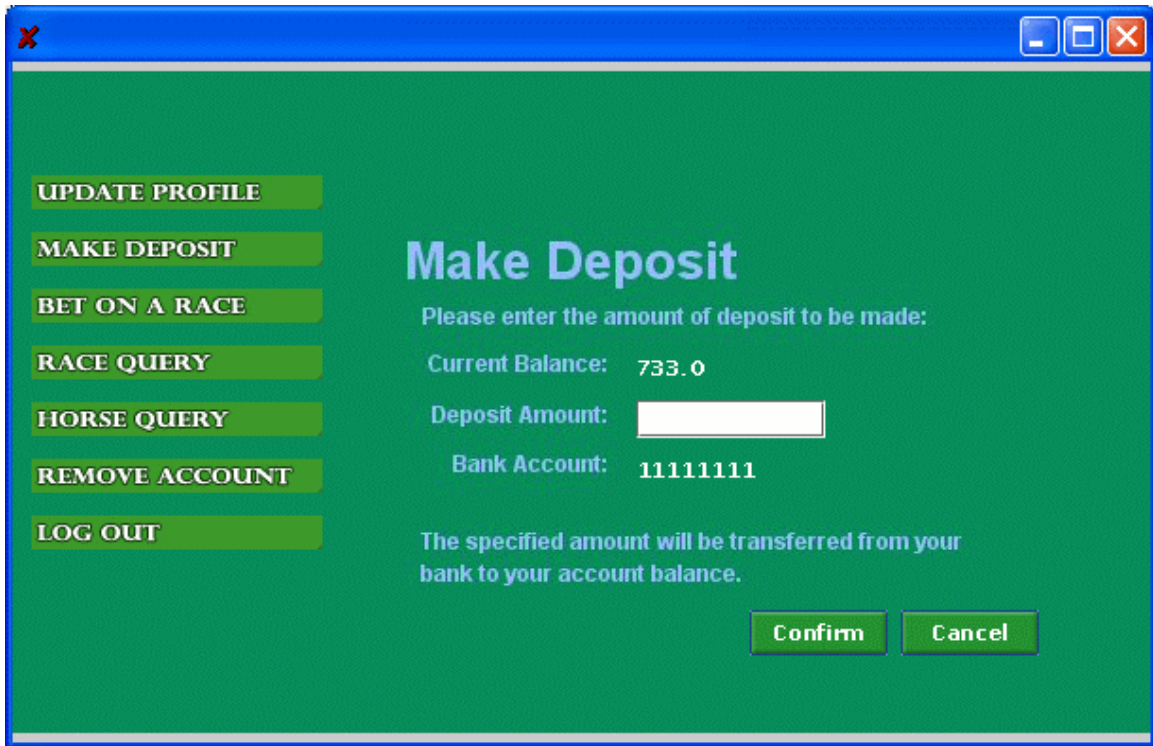
Figure 18: The screen for updating profile. (Gambler Client)



4) Depositing Money to Betting Account

This function will be requested by the gambler client application. Gamblers can deposit money from the bank account associated with the gamblers' account to the betting account. Please note that there will be no validation on the balance of the bank account. Figure 19 shows the screen for money deposit.

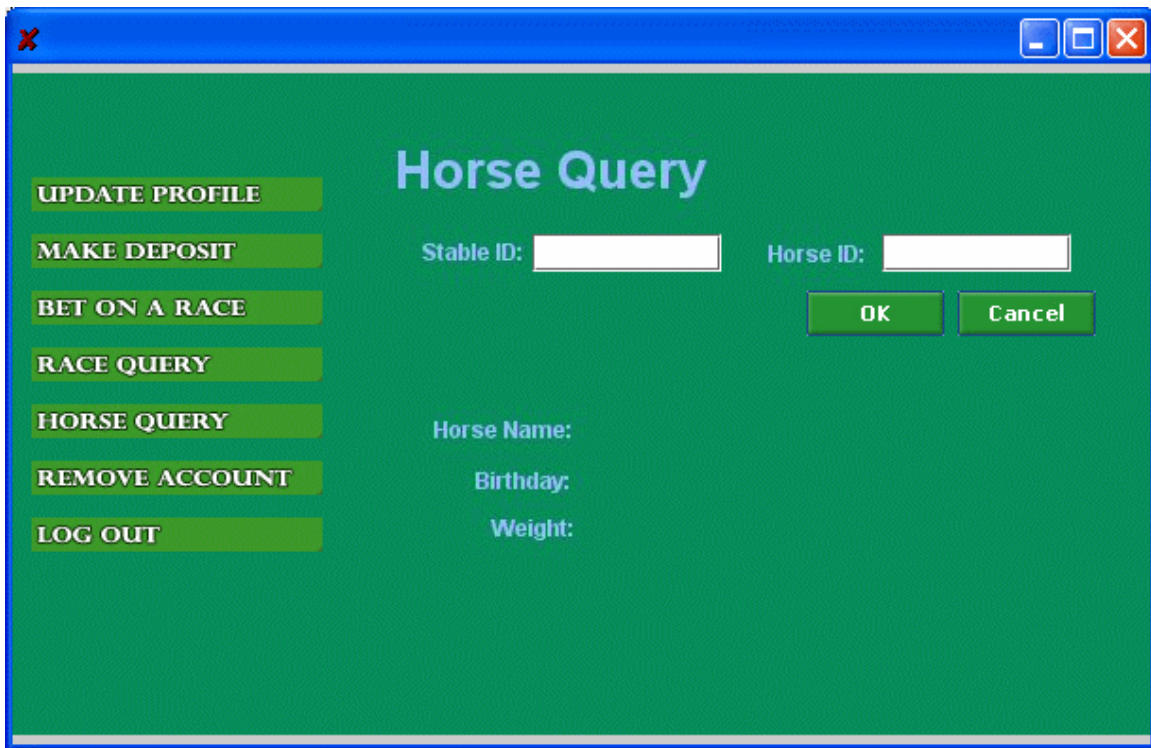
Figure 19: The screen for money deposit. (Gambler Client)



5) Horse Query

This function will be requested by the gambler client application. Gamblers can ask information about horses of a stable. Figure 20 shows the screen for horse query.

Figure 20: The screen for horse query. (Gambler Client)



6) Race Query

This function will be requested by the gambler client application. Gamblers can ask information about horses of a stable. In the first step, gamblers input the race event number. The lanes with horse registered will have a "Show Details" button shown. Details including stable ID, horse ID, win odds and place odds range of the horse in that lane will be displayed if the user clicks on the "Show Details" button. Figure 21 –Figure 23 show the screens for horse query.

Figure 21: The screen for race query – step 1. (Gambler Client)

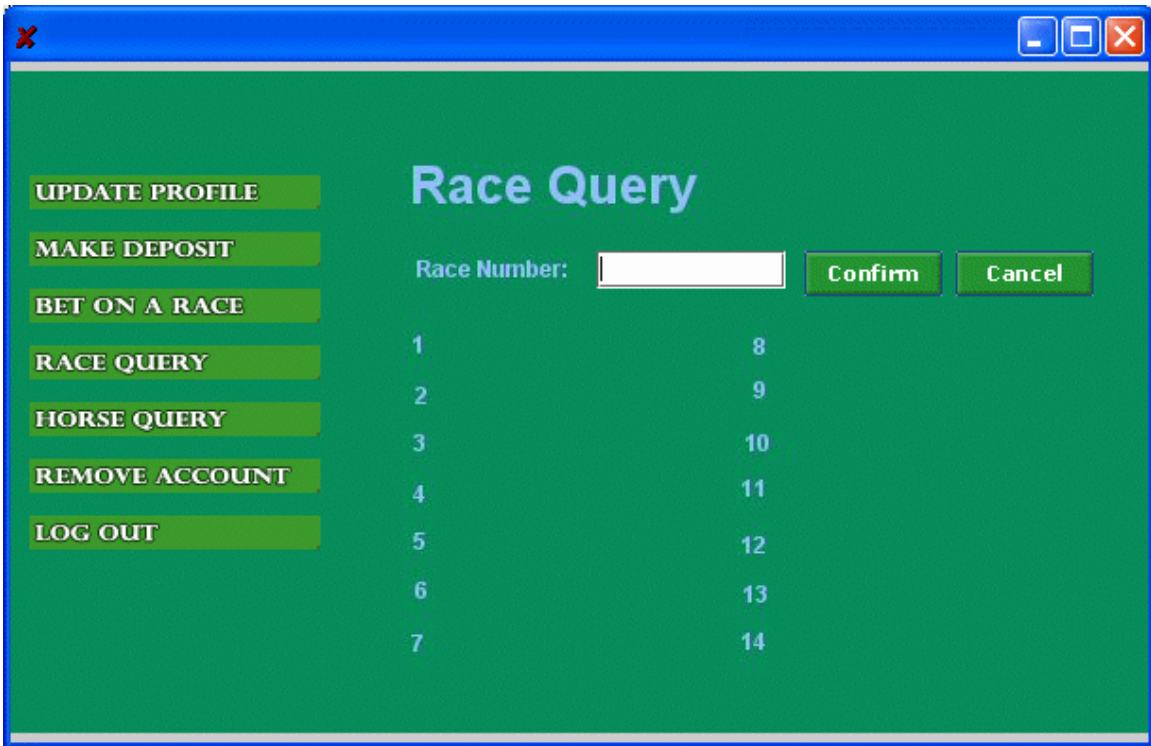


Figure 22: The screen for race query – step 2. (Gambler Client)

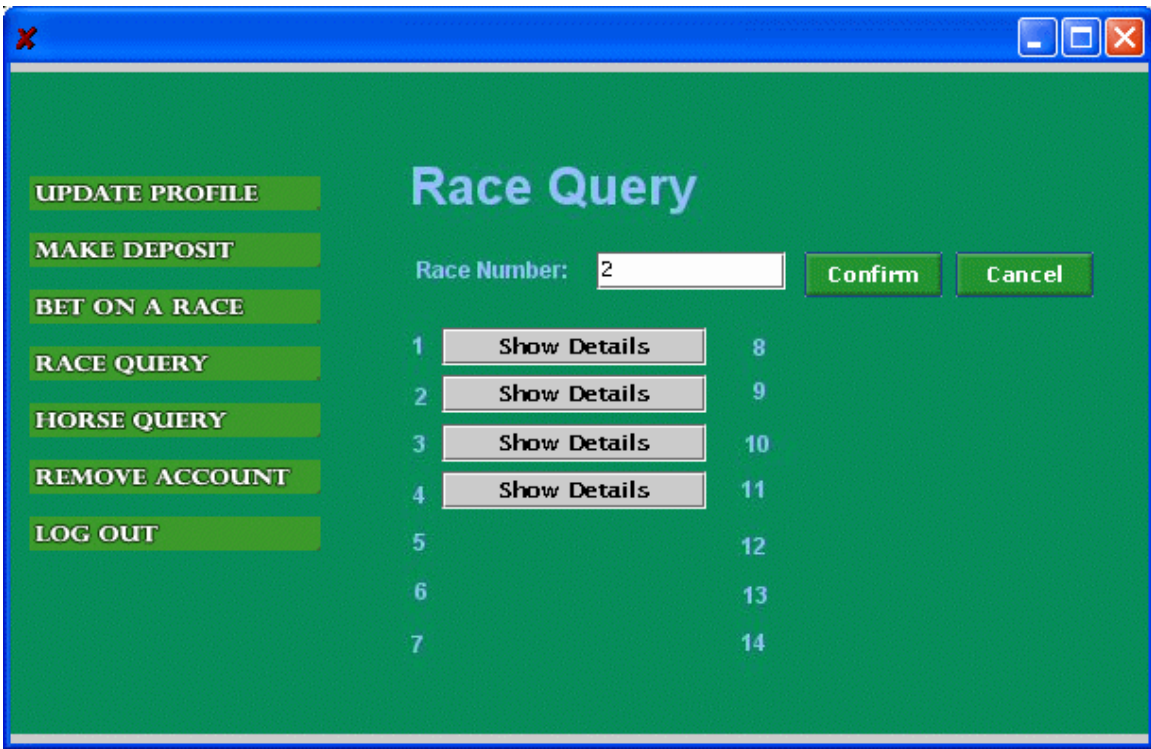
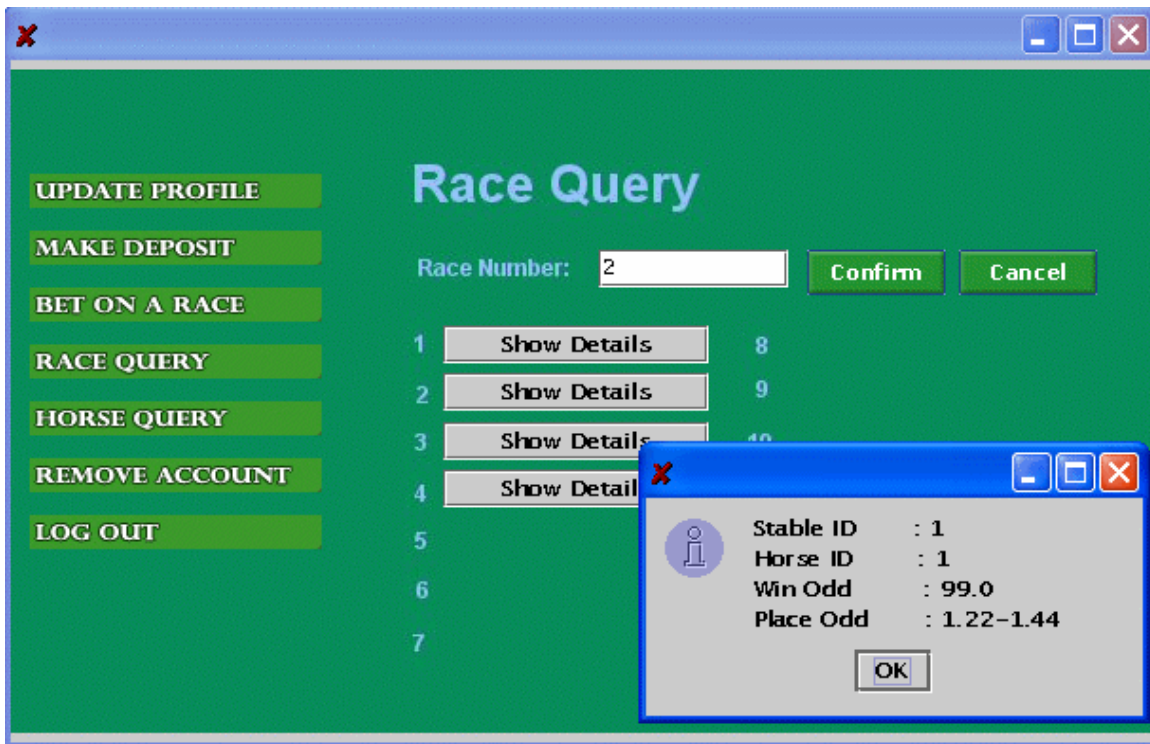


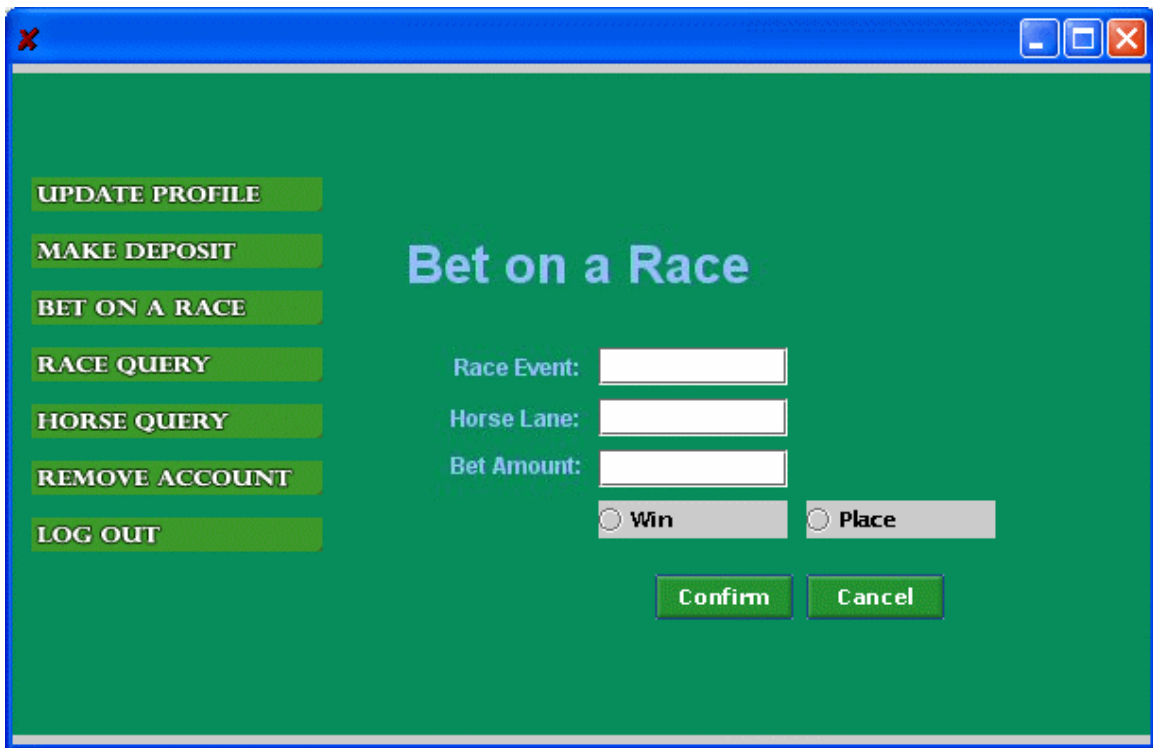
Figure 23: The screen for race query – step 3. (Gambler Client)



7) Placing Bets

This function will be requested by the gambler client application. If there is any race on the current date, gamblers with valid accounts can place bets on the races where the status is “betting mode”. As a result of this operation, money will be deducted from the betting account of the gambler. The two types of games in this project are ‘Win’ and ‘Place’, as described in section 3.1.2. Figure 24 shows the screens for placing bets.

Figure 24: The screen for placing bets. (Gambler Client)



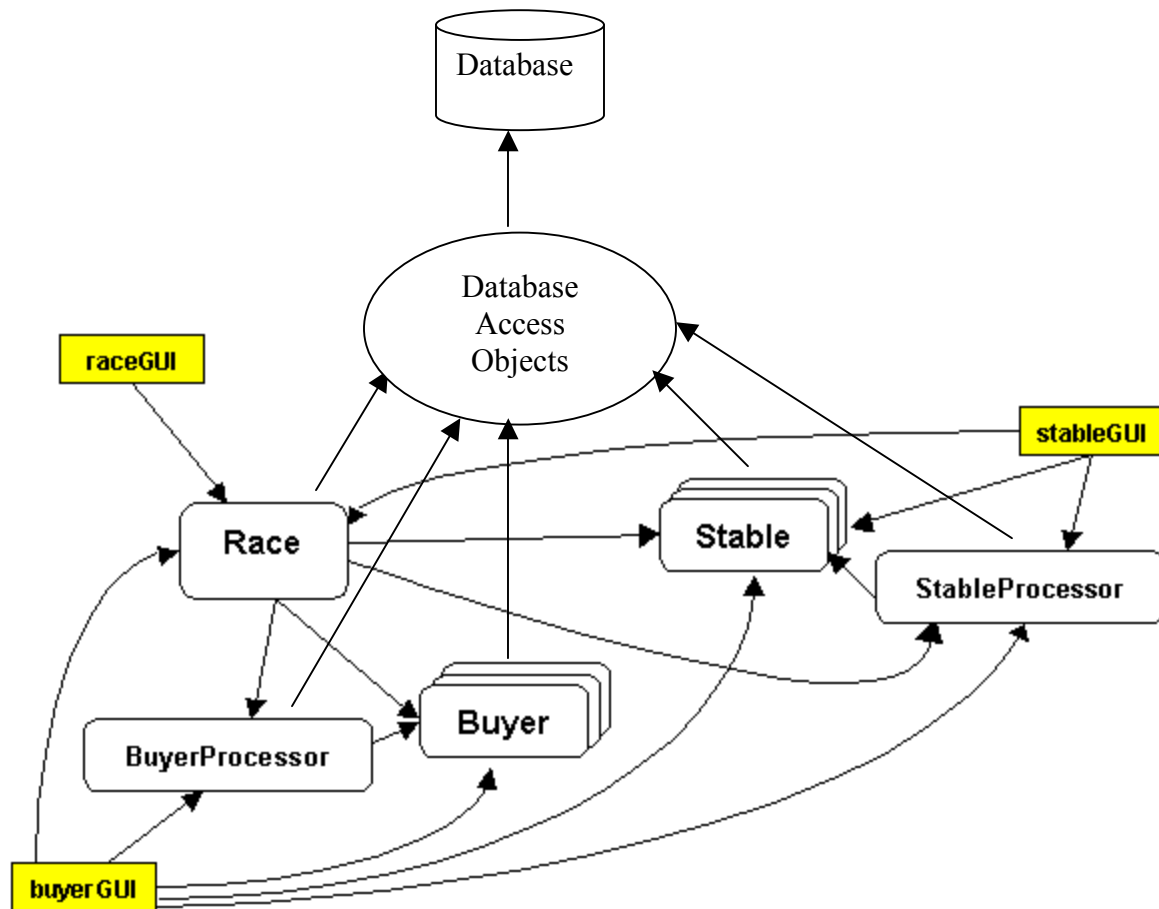
4 System Design

This section will discuss the design of the system. The interaction between different components and their respective architecture will be described in detail. The design of database will be discussed first. Then the server objects and client objects will be covered. In particular, the interfaces of the server objects implementing the functionalities of each of the 3 identities, Jockey Club, stables and gamblers will be described. Their interfaces will be put down in the form of CORBA/IDL, as this project uses the CORBA (Visibroker for Java) implementation. In the following, 'Client' should be considered as client objects under the context of CORBA. 'Client' does not refer to gambler.

4.1 Component Interaction

Figure 25 depicts the interaction between the components of the system. Except for the interaction between the server objects and database access objects, all interaction between objects goes through the Object Request Broker. The three GUI applications, namely RaceAdminClient, StableClient and BuyerClient, do not directly communicate with the database to do transactions. Rather, they request services from the server objects, which in turn perform the transactions on behalf of the clients. The interfaces of the server objects are exposed through CORBA/IDL. These server objects then save the state of transaction to the database via the database access objects. The use of database access objects is to hide database specific details from the server objects.

Figure 25: Interaction between components



4.2 Objects

A race administrator client will serve as a proxy for the Jockey Club. It can be used to formulate horseracing schedule. Stables and gamblers can query information about the horseracing schedule. Stables can also register their horses with the Jockey Club for a race. The race administrator then decides on when to end the registration period of a race and allow gamblers to bet on a race. As it accepts bets from gamblers, it updates the win odds and place odds of each horse based on the amount of bet on each horse and the total amount of bet in the pool. Gamblers can query information about the horses participating in a race. Another

important function that the race administrator can carry out is to start a race. Once the race is started, the simulation stated in section 3.1.4 will kick off to generate the result for the race. In turn, the race results will be saved into a database. Lastly, it pays out the dividends to gamblers. All the functions of the Jockey Club are implemented by the Race server object. The RaceAdminClient object provides a GUI for administrators of Jockey Club to request the above functions to be executed.

A stable administrator client will be used to manage a stable. Unlike the Jockey Club, there can be over 1 stable operating concurrently. A stable consists of a number of horses and jockeys. A stable administrator client is allowed to register a stable with the Jockey Club, as well as canceling the stable account. Registering with the Jockey Club allows it to have the information of its horses available for query by gamblers. Of course, it can update the information. One important function that the stable administrator can perform is to register a horse to join a race. The StableClient object provides a GUI for administrators of stables to request the above functions to be executed. These functions will be performed by the Stable object. The StableProcessor object plays the role of managing all the Stable objects.

A buyer client will be used to perform various operations for the gambler. Multiple buyer clients can be used by a number of gamblers concurrently. It enables a person to register with the Jockey Club and open a betting account, which should be associated with a bank account for settlement. The gambler will then obtain a pair of gambler identification number and password to logon the system. The gambler can then query information about a horse of a stable, check the odds of the starters and place a bet on races. Account balance query, money transfer between the betting account and the bank account also go through this gambler client. The Buyer object is the server object responsible for serving the requests of the gamblers from the GUI object BuyerClient. The BuyerProcessor object plays the role of managing all the Buyer objects.

4.1.1 Interface Race

A Race object does not refer to a single racing game. Here, it provides the whole platform to perform most of the operations for the Jockey Club, stables and gamblers. However, at any time, there will only be one Race Object, which deals with the current racing date. In this project, the Race object only handles races on the current date. This limitation simplifies hierarchy of the system. Figure 26 shows the defined types of Race object.

Figure 26: The idl of Race interface

```
interface Race {
    typedef sequence<unsigned long> ULongArr;
    typedef sequence<unsigned long> ULongArr4;
    typedef sequence<string> stringArr;

    struct RaceSchedule {
        string RaceDate;
        unsigned long RaceEventNumber;
        string TxnStatus;
    };

    struct RaceHorse {
        unsigned long StableID;
        unsigned long HorseID;
        unsigned long HLane;
        float Odds;
        ULongArr Buyers;
        ULongArr Amount;

        float PlaceHighOdds;
        float PlaceLowOdds;
        float PlaceOdds;
        ULongArr PlaceBuyers;
        ULongArr PlaceAmount;
        float Score;
        unsigned long Rank;
    };

    struct HorseRank {
        string RaceDate;
        unsigned long RaceEventNumber;
        unsigned long HorseID;
        unsigned long Rank;
        unsigned long HorseTotal;
    };

    typedef sequence<RaceHorse> RaceHorseArr;

    typedef sequence<RaceSchedule> RaceScheduleArr;

    struct RaceEvent {
        RaceHorseArr RaceHorseArray;
    };
};
```

```
typedef sequence<RaceEvent> RaceEventArr;
attribute string          CurrentRaceDate;
attribute RaceEventArr    RaceNumber;
readonly attribute RaceScheduleArr RaceSch;
attribute ULongArr4       RaceResult;
attribute unsigned long   CurrentRaceEvent;
readonly attribute unsigned long CanStart;

exception HorseAlreadyRegistered { };
exception InvalidRaceEventNumber { };
exception InvalidHorseNumber { };
exception InvalidStableID { };
exception InvalidStablePassword { };
exception InvalidHorseID { };
exception FullRaceHorse { };
exception DuplicatedHorseNumber { };
exception NotEnoughMoney { };

exception PlaceBetNotAvailable { };
exception RegisterHorseNotAvailable { };
exception GetRaceResultNotAvailable { };
exception InvalidInput { };
exception RaceAlreadyStarted { };
exception InvalidRaceDate { };
exception RaceSimulationError { };

void insertRaceDates (in string raceDate,
                    in unsigned long raceEventNumber)
                    raises(InvalidInput);

void setRace (in string raceDate,
             in unsigned long raceEventNumber);

void updateRaceStatus (in string raceDate,
                    in unsigned long raceEventNumber,
                    in string txnStatus)
                    raises(InvalidRaceDate,
                    InvalidRaceEventNumber);
```

```
void buyerBet (in string raceDate,
               in unsigned long raceEventNumber,
               in unsigned long hlane,
               in string    betType,
               in unsigned long betAmount,
               in unsigned long buyerID)
    raises(InvalidRaceEventNumber,
           InvalidHorseNumber,
           NotEnoughMoney,
           PlaceBetNotAvailable,
           RaceAlreadyStarted,
           InvalidInput,
           InvalidRaceDate);

void registerRace (in string raceDate,
                  in unsigned long raceEventNumber,
                  in unsigned long stableID,
                  in string stablePassword,
                  in unsigned long horseID
                  )
    raises(HorseAlreadyRegistered,
           InvalidRaceEventNumber,
           InvalidStableID,
           InvalidStablePassword,
           InvalidHorseID,
           FullRaceHorse,
           InvalidInput,
           RegisterHorseNotAvailable,
           RaceAlreadyStarted,
           InvalidRaceDate);

stringArr getRaceResult (in string raceDate,
                         in unsigned long raceEventNumber
                         )
    raises (RaceSimulationError,
           GetRaceResultNotAvailable,
           RaceAlreadyStarted,
           InvalidRaceDate,
           InvalidRaceEventNumber);
```

```
RaceHorse queryRaceHorse (in unsigned long raceEventNumber,  
                          in unsigned long hlane)  
  raises (InvalidRaceEventNumber,  
         InvalidHorseNumber,  
         InvalidRaceDate);  
  
  long canStartBetting(in string raceDate,  
                      in unsigned long raceEventNumber)  
    raises (InvalidRaceDate,  
           InvalidRaceEventNumber);  
};
```

Figure 27: The relationship between Race, RaceEvent and RaceHorses.

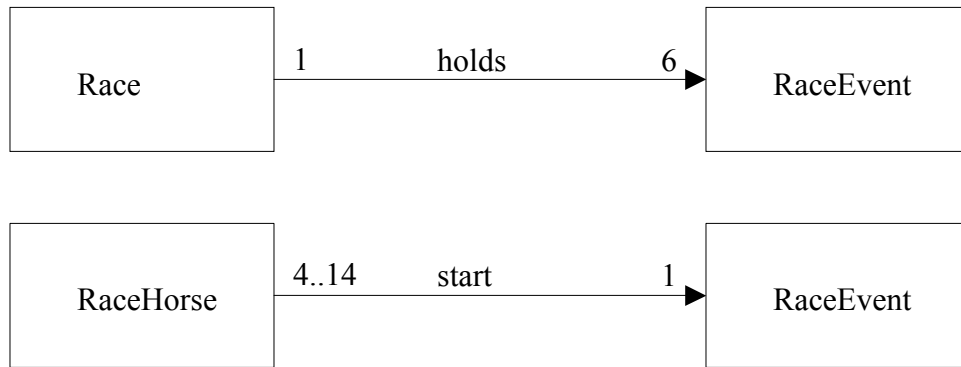
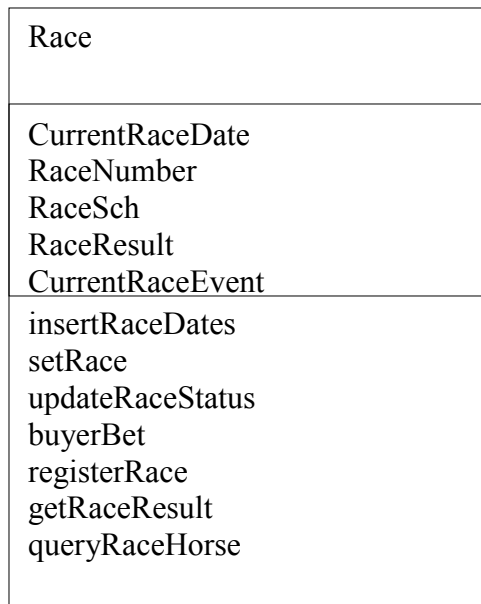


Figure 28: Simple class diagram of Race**Explanation:**

The Race object keeps tracks of the racing event schedule, racing game result, and also the win bet amount and place bet amount of various gamblers. The operation insertRaceDates is used by race administrator to input race schedule. The operation setRace is used to set the values of current race date and current race event number. The operation updateRaceStatus is used to change the status of each race. For stance, it can be used to change a race from “registration mode” to “betting mode”, thus allowing a race to proceed properly. The operation BuyerBet is used to record the amount that the gamblers bet on the races. The amount of bets on each horse affects the odds for the all horses. Theoretically, the odds should be updated every time a new bet is being placed on a horse. The operation registerRace is used by stable administrators to schedule the horses in their stables for a certain race. The identity number of the stable (StableID), the identity number of the racehorse (HorseID) will be associated with the racehorse. The operation getRaceResult is used by race administrator to start a race, which will automatically generate the result by the simulation process described in section 3.1.4. The results

will automatically be saved in the database by this operation. Dividends will also be paid out by this method. The operation queryRaceHorse is used by gamblers to query information about the participating horses.

4.1.2 Interface Stable

The Stable object basically serves to contain horses. Figure 29 shows the defined types of Stable object.

Figure 29: The idl of Stable interface

```
interface Stable {

    struct Horse {
        string          HorseName;
        unsigned long   StableID;
        unsigned long   HorseID;
        string          Birthday;
        unsigned long   HorseWeight;
        unsigned long   RacesWon;
    };

    typedef sequence<Horse> HorseArr;
    typedef sequence<string> stringArr;
    readonly attribute HorseArr          HorseArray;
        attribute string          Trainer;
        attribute unsigned long StableID;
    readonly attribute unsigned long HorseTotal;
        attribute string          StablePassword;

    exception InvalidBirthday { };
    exception NoSuchHorseID { };
    exception FullHorse { };
    exception HorseIDCrash { };

    void addHorse (in string name,
                  in unsigned long horseID,
                  in string birthday,
                  in unsigned long horseWeight,
                  in unsigned long racesWon)
        raises(InvalidBirthday, FullHorse, HorseIDCrash);

    void deleteHorse (in unsigned long horseID)
        raises(NoSuchHorseID);
```

```
Horse queryHorse (in unsigned long horseID)
    raises(NoSuchHorseID);

stringArr getRaceSchedules();

};
```

Explanation:

The operation `addHorse` is used to add a horse into a particular stable, while the operation `deleteHorse` is used to delete a horse from a particular stable. The operation `queryHorse` allows gamblers to retrieve the profile of a specified horse. Finally, the operation `getRaceSchedules` allows stable administrators to check the schedule of horse races.

4.1.3 Interface StableProcessor

The StableProcessor is used to manage all stables. It is also responsible for registering a stable with the Jockey Club, and if necessary, cancel the stable account with the Jockey Club. Figure 30 shows the defined types of StableProcessor object.

Figure 30: The idl of StableProcessor interface

```
interface StableProcessor {
    typedef sequence<Stable> StableArr;

    readonly    attribute StableArr          StableArray;
    readonly    attribute unsigned long      StableTotal;
    readonly    attribute unsigned long      HorseIDNumber;

    exception NoSuchStableID { };
    exception StableNotEmpty { };
    exception NoSuchSourceStableID { };
    exception NoSuchTargetStableID { };
    exception NoSuchHorseID { };
    exception FullStable { };
    exception PasswordNotMatch { };
    exception SourceStablePasswordNotMatch { };
    exception TargetStablePasswordNotMatch { };
    exception TargetStableFullHorse { };

    unsigned long createStable (
        in string stablePassword)
        raises(FullStable);

    void removeStable (in unsigned long stableID,
        in string stablePassword)
        raises(NoSuchStableID, StableNotEmpty, PasswordNotMatch);

    Stable queryStable (in unsigned long stableID)
        raises(NoSuchStableID);
};
```

Explanation:

The operation `createStable` is for registering the stable with the Jockey Club. On the contrary, the operation `removeStable` is for canceling the account with the Jockey Club. The operation `queryStable` is used to retrieve the information about a stable.

4.1.4 Interface Buyer

The Buyer object stores information including the betting account balance and the profile of the gambler. Figure 31 shows the details of this interface.

Figure 31: The idl of Buyer interface

```
interface Buyer {  
    attribute unsigned long    BuyerID;  
    attribute string          BuyerPassword;  
    attribute string          BuyerName;  
    attribute float           Balance;  
    attribute string          AccountNumber;  
  
    exception NotEnoughMoneyInAccount { };  
  
    void deposit(in float amount);  
  
    void withdraw(in unsigned long amount)  
        raises(NotEnoughMoneyInAccount);  
  
    void edit(in string buyerName,  
             in string buyerPassword,  
             in string accountNumber);  
  
};
```

Explanation:

One Buyer object stores the information of a gambler. The operation deposit is used to transfer money from the bank account associated with this BuyerID to the betting account. The operation withdraw is used to transfer money away from the betting account. The operation edit is used to update the personal information of the gambler.

4.1.5 Interface BuyerProcessor

The BuyerProcessor is mainly used to manage all Buyer objects and betting account opening and cancellation. Figure 32 shows the defined types of BuyerProcessor object.

Figure 32: The idl of BuyerProcessor interface

```
interface BuyerProcessor {
    typedef sequence<Buyer> BuyerArr;

    readonly    attribute BuyerArr          BuyerArray;
    readonly    attribute unsigned long     BuyerTotal;

    exception NoSuchBuyerID { };
    exception PasswordNotMatch { };
    exception FullBuyer{ };

    unsigned long createBuyer (in string name,
                               in string password,
                               in string accountNumber)
        raises(FullBuyer);

    void removeBuyer (in unsigned long buyerID,
                     in string buyerPassword)
        raises(NoSuchBuyerID,PasswordNotMatch);

    Buyer queryBuyer (in unsigned long buyerID)
        raises(NoSuchBuyerID);
};
```

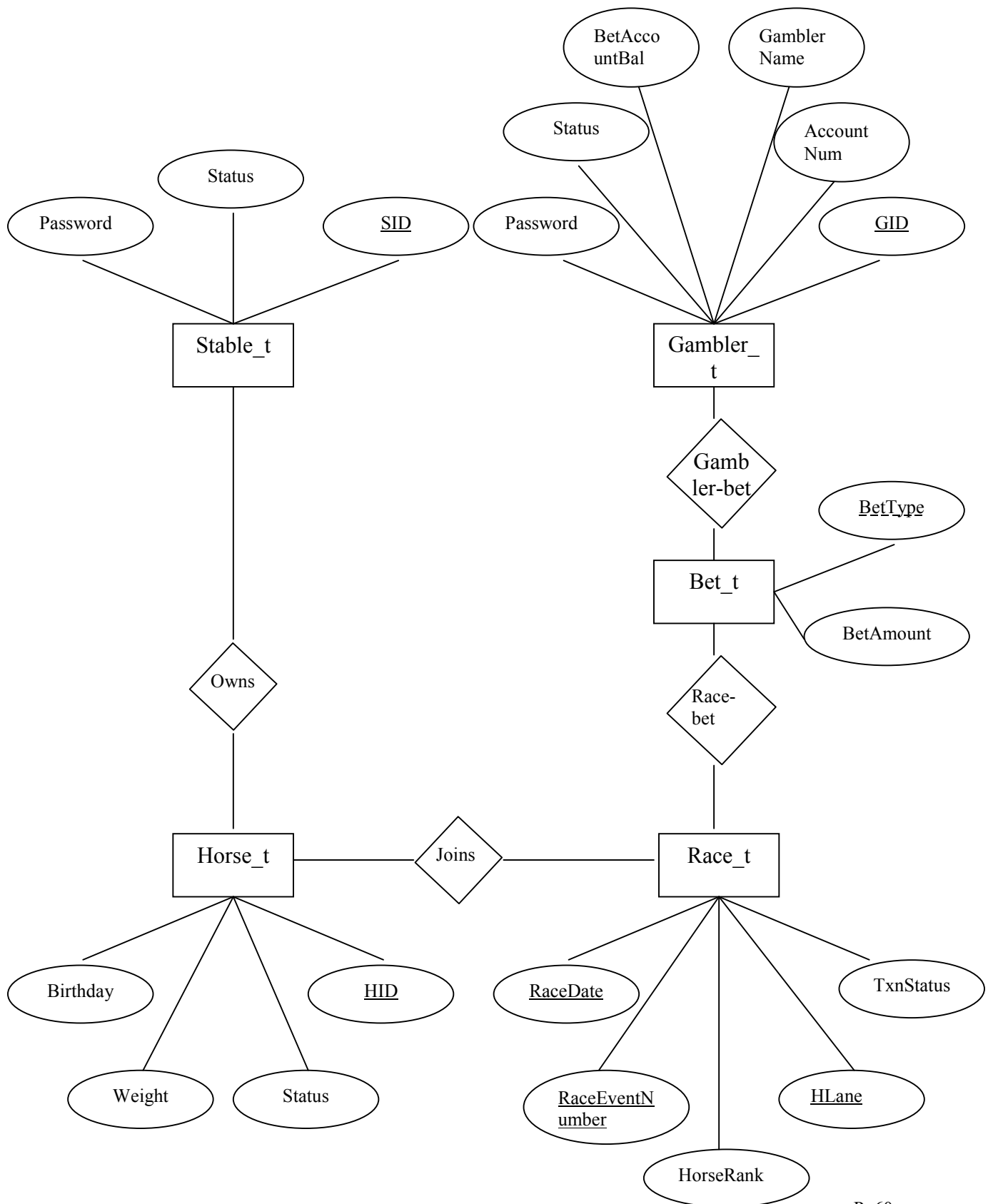
Explanation:

The operation createBuyer registers a gambler with the Jockey Club while the operation removeBuyer removes the gambler's betting account. The operation queryBuyer is used for retrieving the information of a particular gambler by giving a BuyerID.

4.3 Database

This section describes design of the database. E-R diagram and the schema of the tables will be discussed. The E-R diagram depicts the relationship between the entities. From this E-R diagram, the schema will be defined and it will be reduced to tables.

Figure 33: E-R Diagram



The above E-R diagram can be reduced to the following tables.

4.2.1 STABLE_T (SID, Password, Status)

SID refers to the stable id and status refers to the account status. If it is 'Y', the stable account is active. If it is 'N', it is cancelled.

4.2.2 HORSE_T (HID , SID, HorseName, Birthday, Weight, Status)

HID refers to the horse id and status refers to the status of the horse. If it is 'Y', the horse is active. If it is 'N', it is removed.

4.2.3 GAMBLER_T (GID, Password, GamblerName, AccountNum, BetAccountBal, Status)

GID refers to the gambler id and status refers to the status of the betting account. If it is 'Y', the betting account is active. If it is 'N', it is cancelled.

4.2.4 RACE_T (RaceDate, RaceEventNumber, HLane, HID, HorseRank, TxnStatus)

HLane refers to a particular lane of the race. TxnStatus refers to the transaction status of a race. The 4 statuses of a race are 'R', which stands for horse registration period, 'B', which stands for betting period, 'D', which stands for race over, and 'P', which mean all dividends have been paid out.

4.2.5 BET_T (RaceDate, RaceEventNumber, HLane, GID, BetType, BetAmount, TxnStatus)

BetType refers to the type of betting. The 2 types of betting are 'Win' and 'Place'. A detailed description is given in section 3.1.2.

The above table structure can be proved to be complied with third normal form.

The column type of each table will be given below.

STABLE_T

| Column | Type | Description |
|----------|----------|-----------------------|
| SID | CHAR (8) | Stable ID |
| Password | CHAR (8) | Stable Login Password |
| Status | CHAR (1) | Stable Account Status |

HORSE_T

| Column | Type | Description |
|-----------|-----------|-------------------|
| HID | CHAR (8) | Horse ID |
| SID | CHAR (8) | Stable ID |
| HorseName | CHAR (40) | Horse Name |
| Birthday | CHAR(8) | Birthday of Horse |
| Weight | CHAR(3) | Weight of Horse |
| Status | CHAR (1) | Horse Status |

GAMBLER_T

| Column | Type | Description |
|---------------|--------------|-------------------------|
| GID | CHAR (8) | Gambler ID |
| Password | CHAR (8) | Gambler Login Password |
| GamblerName | CHAR(40) | Name of Gambler |
| AccountNum | VARCHAR (14) | Bank Account Number |
| BetAccountBal | CHAR(13) | Betting Account Balance |
| Status | CHAR (1) | Betting Account Status |

RACE_T

| Column | Type | Description |
|--------------|----------|-------------------------|
| RaceDate | CHAR (8) | Date of Event |
| TotRaceEvent | CHAR (2) | Total number of races |
| HLane | CHAR (2) | Lane number of the race |

| | | |
|-----------|----------|--|
| HID | CHAR (*) | Horse ID |
| HorseRank | CHAR (2) | Rank of the horse on this lane |
| TxnStatus | CHAR(1) | Game status R – allow register B – No register, allow bets D – Result generated P – Dividends paid |

BET_T

| Column | Type | Description |
|-----------------|-----------|--|
| RaceDate | CHAR (8) | Date of Event |
| RaceEventNumber | CHAR (2) | Race Number |
| HLane | CHAR (2) | Lane number of race |
| GID | CHAR (8) | Gambler ID |
| BetType | CHAR (1) | Type of Bet W – Win P - Place |
| BetAmount | CHAR (11) | Amount of Bet from this gambler on a Horse |
| TxnStatus | CHAR(1) | Game status - no use currently |

Database access object will interact with the above tables to save and retrieve data. There are 21 database access objects and 1 class responsible for getting connection from database. Figure 34 shows the whole list of database related classes.

Figure 34: Database Access Objects

| Object Name | Usage |
|--------------------|---|
| DbConManager | Get database connection |
| CreateDB | Create database tables |
| DbDeleteStable | Remove stable |
| DbUpdateRace | Update race details |
| DbDeleteHorse | Remove horse |
| DbCreateHorse | Add horse |
| DbCreateStable | Create stable account |
| DbDeleteBuyer | Remove betting account |
| DbInsertRaceDates | Set race schedule |
| DbUpdateRaceStatus | Update the transaction status of a race |
| DbInsertResult | Insert race result |
| DbUpdateBuyerBet | Update the status of a bet – no use currently |
| DbGetResult | Retrieve historic ranks of a horse |
| DbGetHorse | Retrieve horses |
| DbGetStable | Retrieve stables |
| DbGetBuyer | Retrieve gamblers |
| DbGetRaceDates | Retrieve race schedule |
| DbRegisterHorse | Save horse registration |
| DbCreateBuyer | Create betting account |
| DbUpdateBuyer | Update gambler's profile |
| DbBuyerBet | Place a buyer's bet |

5 User Manual

5.1 Compilation

After extracting all the files into a directory, type “make” to compile the whole system. Then all the Java classes will be generated.

5.2 Installation

To install the server program, type “make install” after compilation. Then the server will be set up.

5.3 Uninstallation

To clean up everything, type “make clean”. Then all the Java classes together with the current database will be removed. But once you have uninstalled the system, all the data in the database will be lost.

5.4 Using the System

The system contains 3 interfaces – the buyer, the stable administrator and the race administrator. After setting up the server, type “./start_buyer” to start the buyer interface. Type “./start_stable” to start the stable administrator interface. Type “./start_race” to start the race administrator interface.

6 Conclusion

This horse racing simulation project demonstrates how all the processes involved in horse racing can be automated by a distributed system using the Common Object Request Broker Architecture (CORBA) and how the results of horseracing can be simulated through simple Monte Carlo Simulation.

Theoretically, CORBA implementation allows objects to be implemented using different languages under different platforms. This can be highly beneficial to system integration and flexibility. Though only Java programming language was used in this project, it demonstrates how objects can communicate under CORBA based on operations defined in CORBA/IDL. However, as the popularity of CORBA has yet to drive it to become industry standard, CORBA implementation may be thus confined to internal system integration and can hardly be spread among the community to a large extent.

Regarding the Monte Carlo Simulation to generate race results, there are a few areas that can be improved. First, factors such as racecourse location, race distance can be taken into account to further refine the model. Second, instead of running through the model once to generate the results, each horse can go through the model a few times to reduce the effect of the randomness of the model. An alternative to simulation is modeling, which set the rank of a horse as the dependent variable and other factors as independent variables.

7 References

- Bickel, P. J. and Doksum, K. A.: *Mathematical Statistics: Basic Ideas and Selected Topics*, Holden-Day, San Francisco (1977).
- Bratley, P., Fox, B. L., and Schrage, L. E.: *A Guide to Simulation*, 2d ed., Springer-Verlag, New York (1987)
- Conover, A. C., and Whitten, B. J.: *Estimation in the Three Parameter Lognormal Distribution*, *J. Am. Statist.* 2d ed., John Wiley, New York (1980)
- Chung, K. L.: *A Course in Probability Theory*, 2d ed., Academic Press, New York (1974).
- Farley, J: *Java Distributed Computing*, 1st ed, O'Reilly, CA (1998)
- Hastings, N. A. J., and Peacock, J. B.: *Statistical Distributions*, John Wiley, New York (1975).
- Hogg, R. V. and Craig, A.F.: *Introduction to Mathematical Statistics*, 3d ed., Macmillan, New York (1970).
- Kendall, M. G., Stuart, A. and Ord, J. K.: *The Advanced Theory of Statistics*, Vol. 1, 5th ed., Oxford University Press, New York (1987).
- Law, A. M. and Kelton, W. D.: *Simulation Modeling & Analysis*, McGraw-Hill, 267 – 297 (1991)
- Ross, S. M.: *Introduction to Probability Models*, 4th ed., Academic Press, San Diego (1989)
- Siegel, J.: *CORBA 3 Fundamentals and Programming*, 2nd ed, John Wiley, New York (2000)
- Vose, D: *Quantitative Risk Analysis*, John Wiley, New York, 39 – 56 (1996)
- Vose, D: *Quantitative Risk Analysis*, John Wiley, New York, 103 – 135 (1996)
- Wagner, H. M.: *Principles of Operations Research*, Prentice-Hall, Englewood Cliffs, N.J. (1969)
- Young, Donovan: *Modern Engineering Economy*, Wiley, 411 – 415 (1992)