An Online Reservation System

by

Arpan Shah

Submitted to the Department of Electrical
Engineering and Computer Science in
Partial Fulfillment of the Requirements for the
Degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

Massachusetts Institute of Technology

February 2000

Signature of Author .....
Department of Electrical Engineering and Computer Science
January 30, 2000

Certified by .....
Dr. Amar Gupta
Co-Director, Productivity From Information Technology (PROFIT) Initiative
Thesis Supervisor

Accepted by .....
Arthur C. Smith
Chairman, Department Committee on Graduate Students

AN ONLINE RESERVATION SYSTEM

by

ARPAN SHAH

Submitted to the Department of Electrical of Electrical Engineering
and Computer Science on February 2, 2000 in partial
fulfillment of the requirements for the
Master of Engineering in Electrical Engineering and Computer Science

Abstract

This thesis paper discusses the design and development of an Online Reservation System (ORS) for restaurants. It begins by examining the needs and objectives of such a system. It then discusses existing reservation and non-reservation computer systems that are relevant to the design of the ORS. The design of the system is then presented, analyzing the different software components that make up the system. It also discusses data structures, protocols, and algorithms that the system uses. After the design, the implementation of the design is discussed. Following, is the test analysis section which discusses whether the ORS met its objectives. Performance is also evaluated near the end of the paper along with possible extensions of the system.

Thesis Supervisor: Dr. Amar Gupta

Title: Co-Director, Productivity From Information Technology (PROFIT) Initiative
Sloan School of Management

Acknowledgements

I would like to thank Dr. Gupta for the support and advice he has given me in the last one year. His encouragement and guidance were extremely helpful in the design and development of the product.

Also, I would like to thank my family, Ashok, Bharati, and Arpa Shah, for all their encouragement and support in the last four years.

Lastly, I would like to extend a thank you to Ravindra Sarma who helped me with the graphical user interface.

Table of Contents

Chapter 1

Introduction

In the last one hundred years, mankind has made incredible technological progress. The television, the airplane, space travel and of course, the computer have all been innovated this past century. We live in a society where capitalism leads to innovative ideas, and where innovative ideas lead to a higher standard of living.

It is the end of the twentieth century, and it is the age of the Internet. Everyone has gone "dot com" crazy. With so few barriers to entry, anyone can put up a website: a teenager, high-school teacher, a housewife, or a multi-million dollar company. The Internet is a faceless medium that everyone can have access to; one that everyone can take part in.

The Internet, besides just being a network of websites, has enabled numerous companies to do business differently. The brick-and-mortar retail stores are now able to sell their merchandise to customers over the Internet by creating virtual store-fronts. Barnes and Noble, for example, sells books to consumers through barnesandnoble.com. The Internet has not only helped businesses succeed, it has also allowed new companies to emerge. In fact, companies that do not even exist physically in the "real world" are making money by selling over the Internet. Companies that do not even have a physical store are competing with brick-and-mortar stores neck-to-neck in the Internet space. One example of such a company is Amazon.com. Amazon competes with arch-rival Barnes and Noble, despite its lack of a physical presence in the "real world."

Companies that have been quick to obtain a web presence without having modified their business model are likely to lose to their competition who have. With 43% of all Americans

online[1], it is important for businesses to channel their products and services directly to the consumer. A perfect example of such a strategy is e-commerce. E-commerce, electronic commerce, is simply commerce done over the Internet. Some companies make all of their revenue using this business model. However, the Internet is not just about making money through e-commerce; it is also about making businesses grow and improving the lifestyles of consumers. Many times, it is about advertising to a larger audience; other times it is about catering to the specific needs of customers.

This thesis document discusses the specification, design, and implementation of a reservation system that helps restaurants move to the web and build their online presence in the Internet economy. The online reservation system enables restaurants to expand their customer base while allowing customers to make reservations in a real-time environment. Before research and implementation of this product began there were virtually no restaurants that had this capability and no company that offered this technology. More recently, however, a few companies have sprung up that offer this as a service to restaurants - savvydiner.com, foodline.com and opentable.com. The online reservation system described in this paper is extremely object oriented in nature and is designed keeping issues such as scalability, portability, connectivity and practicality in mind. The needs of the maitre d'[2] and the business model of the restaurant were also key factors in designing the system. It is implemented using off the shelf software and hardware, unlike other systems that are customized for each restaurant.

[1] General Internet usage figures from CyberAtlas, at internet.com, Inc. (July 1999)
[2] maitre d' is the acceptable English short form of the French word maître d'hôte.

Chapter 2

Current Restaurant Reservation System

Today, almost all of the restaurants in the United States and in the rest of the world make reservations by hand. Maitre d's usually have a "reservation book" that they make reservations in – either blank or dated.

After having spoken to maitre d's of different types of restaurants, it is also clear that most do not have a particular way of scheduling their customers. For instance, if customer A calls for a reservation at 9pm, most maitre d's go to the page of the date of interest, check the time slots around 9pm, and make a "guess" as to whether or not there is going to be availability. Many times, these "guesses" or "hunches" are wrong. As a result, customers with reservations often wait for a table, which in turn leads to customer dissatisfaction. Occasionally, the opposite occurs where maitre d's make poor estimates which lead to loss of business.

In order to make a reservation today, customers have to call the restaurant to make a reservation. Unfortunately, this practice has many drawbacks, some of which include the following:

i) Customers can only make reservations during the restaurant's operating hours. This prevents customers from making reservations outside business hours.

ii) Customers must make reservations over the phone. This requires the customer to know the telephone number of the particular restaurant. Since the average restaurant only has one telephone line, it may be difficult for a customer to get through to the restaurant. Also, the restaurant could be a long distance telephone call for a customer.

iii) The customer is unaware of all of his/her dining options. For instance, presently, most people choose to dine at a particular restaurant either because they have heard of the restaurant through word of mouth, due to personal experience, or because of advertising. However, with the large and growing number of restaurants, especially cities like New York City and San Francisco, the customer may be missing out on a great dining experience at a restaurant he/she has not heard about.

iv) The customer does not know about the different reservations options that he/she has. For instance, he/she can not talk with the maitre d' for extended periods of time. He/she does not have the luxury of knowing the different reservation schedules or the arrangement of the tables.

v) There is no quick way of knowing what the restaurant serves. Many restaurants do not have an Internet presence, and consequently users do not have the luxury to browse their menus and learn about their specials and pricing.

This thesis paper will describe the Online Reservation System (ORS) in great detail. At first, the objectives and features of the ORS will be discussed, then a general overview will be presented, and finally the specifics of the ORS will be detailed examining the different components of the ORS.

Chapter 3

Objectives

The Online Reservation System addresses all the problems stated in the previous section. It is a distributed N-client computer system. The computer system consists of a central component and N-clients (the N-clients being the N restaurants). Specifics of the design are discussed later in the paper.

3.1 The New with the Old

The ORS is a system that enhances the current restaurant reservation process. However, in order to make the restaurant reservation process powerful, restaurants must be willing to make some changes to their current way of making reservations. Ideally, all customers would make reservations via the Web. If this were true, the ORS would be easier to design and implement since there would only be "one point of reservation" - there would only be one place from which customers could make reservations. Unfortunately, this is not a valid assumption, and consequently, many intricacies and complications arise. One cannot assume that customers will stop calling restaurants; so it is impossible to replace the current reservation system with a completely new system. ORS is designed in such a way that customers still have the option of calling; in software terms, it is backward compatible.

In order to accommodate this essential feature, it is important to create an application for the client, which in this case is the restaurant. (For the remainder of the thesis paper, "client" will refer to a restaurant and "user" will refer to its customer). A client application is needed so that the maitre d' can enter reservations made over the phone into the system. They must enter the

phone reservations into the ORS or the system will not be aware of which tables are reserved and which ones are available.

3.2 Connectivity

Connectivity is an important issue that must be addressed. As mentioned above, it is clear that the restaurant has to enter reservations made by phone into the system. However, does the restaurant have to be connected to the Internet to make reservations? Questions such as, "what happens if the restaurant disconnects from the Internet?" have to be answered since one cannot assume that all restaurants are connected at all times. The reason for this is two fold: connectivity is unreliable with a modem or DSL line and staying connected to the internet twenty four hours a day can turn out to be quite expensive for small businesses.

If the restaurant is connected to the Internet at all times, then customers can make reservations over the web and customers can call the restaurant without any problems. This is clear since the restaurant system (the restaurant system is referred to as RezzSys) knows exactly what reservations have been made over the web. However, if the connection is broken, then what happens is not clear. The ORS is no longer able to make reservations via the web since it does not have a complete view of all the reservations made.

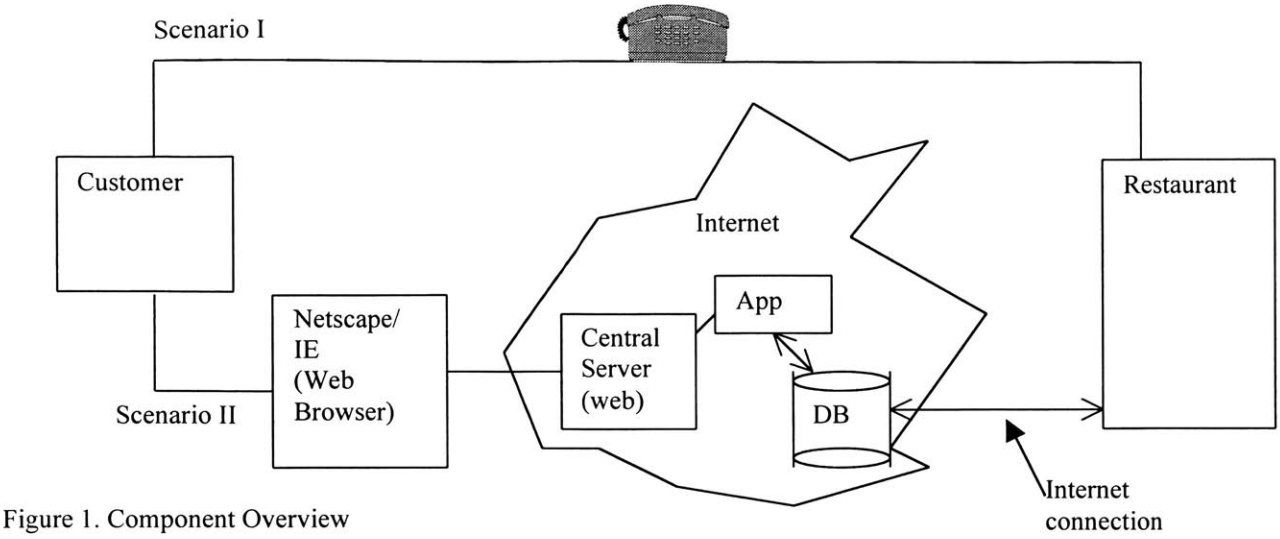

Figure 1. Component Overview

In any implementation, it is necessary for the central server to have access to the reservations. In the ideal situation the client and the central server should both have complete, consistent data. Therefore all reservations made via the Internet and all reservations made over the phone must be recorded somewhere where the central server and client can access them: some IP address they both can access over the Internet. If the Internet connection at a restaurant breaks down, the ORS faces a problem: How does the client update the data? How does the central server know about the new reservations made while the client is disconnected? The ORS should be able to handle this situation effectively. Clearly, connectivity is an issue that must be addressed when designing the ORS. There are different approaches that one can take to this problem. In this paper, the ORS system is designed to allow restaurants to make telephone reservations while temporarily preventing reservations to be made over the web.

3.3 Hard Disk Crash

The chances of an individual hard disk crashing and getting damaged permanently is small. However, the chances that at least one restaurant's hard disk permanently "crashes" from a large sample is non-negligible. Assume that the probability of a hard disk permanently crashing in a given time interval T (say one year) is p, where p is very small (assume it is .001). Also assume that there are a total of N people and that the probability of a permanent hard disk crash is an independent event.

Pr[1 computer crashes] = p
Pr[1 computer does NOT crash] = 1 - p
Pr[N computers do NOT crash] = $(1 - p)^N$
Pr[At least one computer crashes] = $1 - (1 - p)^N$

The number of computers that need to be part of the experiment so that the probability of at least one computer crashing permanently is ½ is:

$1 - (1 - p)^N = ½$
$N = -\ln 2 / \ln(1 - p)$
➔ assuming p = .001, N = 692

Taking all these assumptions into account, if there are at least 692 computers in the system, that is, if there are 692 restaurants who provide online reservations, at least one of those restaurants will experience permanent hard disk failure with probability greater than ½. This concludes that from a business point of view, one must take into account the possibility of a hard disk failure when designing the system. Thus, it is important for the ORS to have backup data just in case one copy is corrupted.

3.4 Reservation Algorithm

Designing a reservation algorithm does not seem like a complicated problem. Unfortunately, it is very complex. Restaurant reservation is analogous to many scheduling problems, which have been proven to be extremely difficult to solve in polynomial time. However, what makes restaurant reservation particularly difficult are the following factors:

i) The system does not have knowledge of all the reservations. It does not know of what future reservations will be made and/or requested. This is because the system is real-time. If the system was not in real time and instead performed batch scheduling, the problem would be significantly easier since all the information would be available.

ii) Many times, particular tables are reserved for important people: celebrities, for example. These tables cannot be reserved by a software agent. In other words, only the maitre d' should be allowed to assign certain tables. Also, even if two tables hold the same number of people, one may be preferred over the other. For example, table A may still be preferred over B because A has a better view for instance.

iii) Smoking and non-smoking tables. Though many restaurants in the United States are becoming non-smoking restaurants, there are still many restaurants that have non-smoking and smoking sections.

iv) The maitre d' may want to override a reservation made by the system and swap reservations around. For example, he/she may move customer C's reservation from table 2 to table 8 because of a crack in table 2.

The ORS addresses issues ii - iv. Unfortunately, there is nothing the ORS can do about the lack of information; it cannot possibly know more about the future. However, the ORS'

reservation algorithm takes a greedy approach to come up with a solution that is efficient and near optimal.

3.5 Performance

It is important for the ORS to be efficient. While including all the necessary features, one must keep in mind the efficiency of the system. For example, if the client application made a database transaction for every single operation, the system would be slowed down, especially if the database did not reside locally (that is, on the client machine). At the same time, the client cannot afford to cache every single piece of data since that would take an unrealistic amount of memory. Therefore, there must be some compromise made between the two. Also, the system should be efficient for the customer who makes a reservation from the Internet. Similarly, the central server should not query the database lying on the client every time a customer wants to get information - reservation or otherwise.

3.6 Customer and Restaurant Profiling

In order for a restaurant to offer the best services to their customers, it is essential for it to know its customers. This is guaranteed to improve business and customer relations. The ORS is designed to support customer profiling. When customers call the restaurant for reservations, the ORS checks to see whether a customer profile of the customer exists. If a profile exists, the maitre d' immediately knows the relevant information of the customer and his/her preferences. For example, he knows whether the customer prefers a smoking/non-smoking table, his favorite table, food preferences of the customer, and his first and last name. Similarly, if a customer makes a reservation via the web, the central server knows all this information and thus is able to

make a well-informed reservation for the customer. Needless to say, the ORS will not have the customer's information the first time he/she uses the system (either calling or via the Internet), so the customer will have to enter data the first time a customer uses the system.

The ORS not only provides a mechanism for maitre d's to know more about their clients, but it also provides a way for customers to learn about different restaurants. They can learn about their different options via the Web. By choosing the type of food they like, their location preference, and the price range, the ORS can provide the customers with many dining options. This is possible because the ORS has knowledge of all its different clients. This is an excellent way for restaurants to expand their business to new customers.

Chapter 4

Relevant Computer Systems

This section discusses some of the existing computer systems and technology that have helped in the design of the Online Reservation System discussed in this paper. At first, existing reservation systems across different industries are explored. Later in the section, non-reservation computer systems with desirable characteristics and properties that are relevant to reservation systems are analyzed.

4.1 Existing Reservation Systems

Many companies across several industries have developed and adopted reservation systems. These reservation systems have proven to be extremely helpful and have, in many cases, led to increases in revenues and market share. Corporations that have been quick to adopt this new technology have benefited tremendously, and over time competitors have been forced to use a reservation system to stay in business. With the advancement of the Internet, real-time, online reservation systems are becoming more and more popular. The ability for customers to make reservations over the web has attracted many client driven large companies.

Most existing reservation systems are large, expensive computer systems that have been designed and developed for large companies. Until recently, only big corporations needed reservation systems since most small businesses could operate using a notepad or a preexisting applications like Excel. Hence, the most common reservation systems are those of major airlines, rental car companies, and movie theaters. These systems have similarities as well as significant differences from the ORS that is discussed in this thesis.

Points of Entry

A key difference between large existing systems and the ORS is the number of points of entry (POEs). In a large system, there are potentially thousands of POEs to the same system. These POEs are usually operators (who work for the company directly). For example, when a person calls an airline to make a reservation, they speak with an operator who

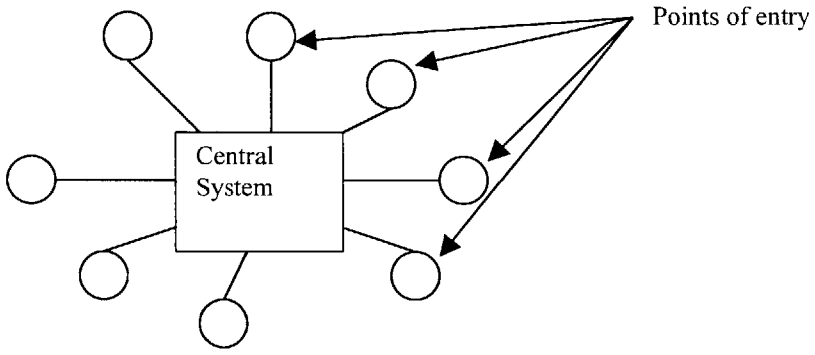

Figure 4.1 Typical Company Reservation System Layout

is a point of entry into the airline reservation system. More sophisticated reservation systems have other virtual extensions to their network. An airline company, for example, can offer travel agencies connectivity into their network from their offices. Now, these agencies also become POEs. The reason why there are so many POEs in large systems is because of the volume of transactions and the popularity of the service and network. Consequently, more operators and extensions are required to handle the customer requests.

With the ORS, a restaurant reservation system, there are exactly N points of entry, where N is the number of restaurants connected to the network. Even with a web component, the ORS has N POEs since reservation requests are pushed to the appropriate local system, as is discussed

later in the paper. As can be observed in figure 4.2, the ORS is a network of many local restaurant systems.

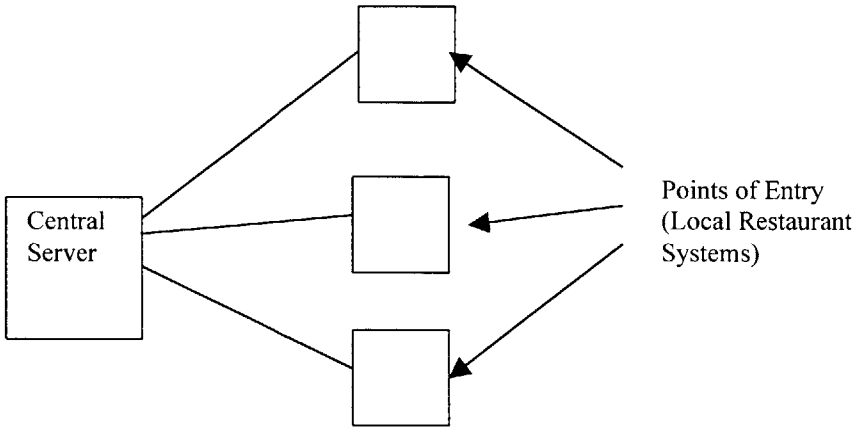

Figure 4.2 ORS System Layout

Having a large number of entry points has its advantages and disadvantages. One of the advantages of having so many, as depicted in figure 4.1, is that if one of them malfunctions (goes down or gets disconnected from the network), it does not affect the system. People merely go through another operator. Due to the layout of the system, special attention does not have to be given to connectivity, whereas, connectivity is a very important issue in the ORS as discussed later. One of the disadvantages of having many entry points is that concurrency is difficult to deal with – this is also discussed in detail later in the section.

Centralized vs. Decentralized Systems

As can be seen in figure 4.1, airline reservation systems and other large systems generally tend to be centralized systems. This does not mean that all the data lies in one location. In this

case, centralized means that the system resides in a controlled network environment – typically an Intranet. By using the latest server technology, nearly 100% reliability can be guaranteed. Of course, the system itself can be spread across many different machines and servers. In most cases, systems are spread geographically, as well as processor-wise, across many different servers. The primary reason to have distributed data and processing is to enhance performance, which is critical in large reservation systems.

The ORS, as denoted in figure 4.2, is a distributed network. It is comprised of many different restaurant systems that do not lie in a controlled environment. It cannot be assumed that a restaurant, for example, is going to be connected to the Internet at all times. This is one of the complications of the layout of the ORS. However, because of its distributed nature, each local system is a virtual processor of the system, and consequently, performance requirements are more easy to meet.

Connectivity

As discussed earlier, connectivity is an issue that must be addressed in the design of the ORS. In larger, centralized systems, it is unrealistic for a client to be able to function without being connected to the system. In order for a client (computer terminal in most cases) to operate when disconnected from the network, it would have to be able to know the most current information. Since all the clients are making changes to the same data, this makes it impossible for disconnected terminals to continue to operate. Therefore, even if the data were stored locally on the client machine, it would not be up-to-date and the terminal would not be able to make a reservation. As a result, when a computer terminal of a larger system (airport terminal, for example), goes down or gets disconnected from the central system, the employee operating the

terminal simply uses another terminal to make reservations. In this case, it is not a great complication for a terminal to go down, since there are many other terminals that can continue to make reservations. However, in a not properly designed restaurant system, if a client is disconnected, lack of connectivity can potentially prevent all reservations for that restaurant from being made.

Performance vs. Concurrency

In every computer system, trade-offs must be made between performance and concurrency issues. In large real-time reservation systems, this trade-off is very important because performance and consistency of data are both critical for most companies.

Fortunately, the ORS is a network of local restaurant systems. Each restaurant system is independent from another. Since reservation requests can be "pushed" across the network to the concerned local restaurant system, the load can be distributed amongst the different restaurant systems. Also, because of the relatively low number of transactions – on the busiest day, assume that the average restaurant has 200 reservation requests (this equates to approximately .007 reservations/second assuming an 8 hour period) – concurrency problems can be solved completely without compromising performance (a detailed discussion of the ORS' performance can be found in Chapter 9).

In the case of larger reservation systems, where the volume of transactions is thousands per second, dealing with concurrency and yet maintaining performance standards becomes a formidable task. Therefore, many systems compromise concurrency for performance and in some cases this results in inconsistent data.

Heterogeneity

Another important difference between the existing systems and the proposed restaurant reservation system is that the ORS has different restaurants (businesses) as its clients. This brings about many concerns that are not present in reservation systems designed for a particular company. Each client, in the case of the restaurant reservation system, has its own reservation constraints, graphical user interface, and data. In the case of an airline reservation system, for example, all the clients (terminals) have the same constraints, interface, and data. Also, another key difference between the ORS and other reservation systems is the complexity of the reservation algorithm. When tickets are sold, there is a predetermined non-variable time slot for each event. However, when making a reservation at a restaurant, a system has to be built to "guess" how long each table will be occupied for.

Budget Restrictions

Large hotels, car rental companies, and airlines can spend hundreds of millions of dollars to implement and maintain their reservation systems. Unfortunately, since restaurants are small businesses, they can not afford to spend as much as these large corporations. The ORS is specifically designed to allow restaurants to have a local reservation system, Internet presence and allow customers to make reservations via the Internet. Ideally, each restaurant system would be connected to the Internet with a T1 line with 100% connectivity to the Internet; each restaurant system would run on expensive servers. However, if a costly solution were implemented for the ORS, practically speaking, the price that would have to be charged for each restaurant to cover costs would be incredibly high. Consequently, the ORS is restricted and cannot afford the expensive hardware that large companies can.

4.1.1 ReserVec

ReserVec was the name given to the computerized reservation system developed by Trans-Canada Airlines in the 1960s. It was designed to handle over 60,000 transactions per day. The development of ReserVec eventually led to Canada's first built transistorized computer: the Gemini[3].

Design

The ReserVec system consisted of the central registry, the remote reservations offices, and the communication systems between the two (the general layout was similar to that depicted by figure 4.1, where the offices are the points of entry). The central registry of the ReserVec computer system was located in Toronto. It contained twin computers, their shared memory systems, computer couplers, paper-tape readers, and teletypewriters[4].

The Gemini computer consisted of two identical computers that ran on the same software. They were individually known as the Castor and Pollux. Together, they shared the load of the reservation system. Each computer had its own arithmetic unit, program control unit, core memory, and paper-tape reader. They, however, shared drum and magnetic-tape memory. This shared external memory had the "master inventory" which contained the entire Trans-Canada Airlines flight schedule for over 350 consecutive days. The drum memory contained the inventory for approximately the next 10 days' flights and for those that were on the brink of full capacity. Drum memory served as a form of cache for quick access and lookup.

---

[3] *Wings Across Time, The Story of Air Canada*, Griffin House, 1978.
[4] *The Electronic Reservations System for Trans-Canada Air Lines*, Univ. of Toronto Press

The ReserVec system effectively dealt with concurrency. If one computer was operating on a particular drum-memory location, the other computer was locked out of that memory location until the first was finished. The engineers accomplished this feat through a communications link between the computers via registers. If Pollux, without loss of generality, was accessing memory location m, the relevant registers would hold m so that Castor would know that Pollux was accessing m[5].

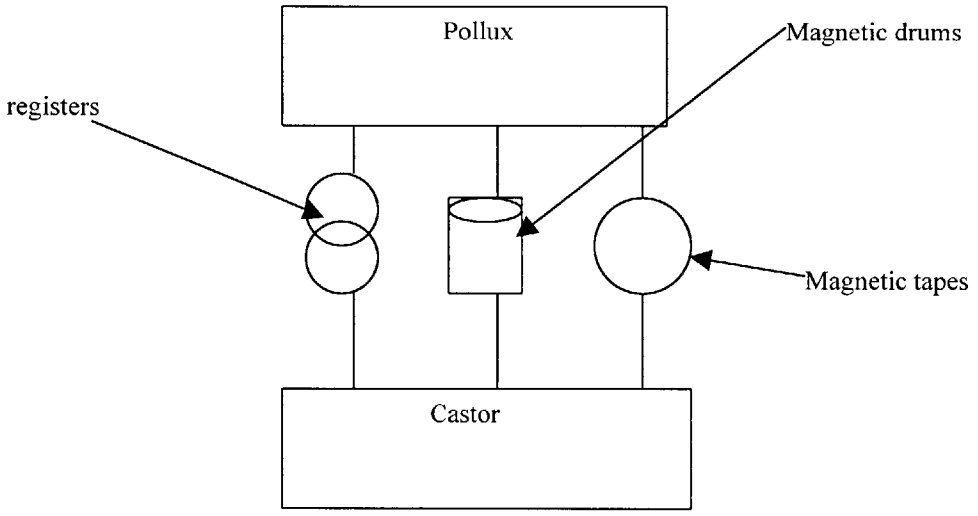

Figure 4.3 Gemini Architecture

On each reservation request, the Gemini computer system would determine whether the relevant flight's information was stored on drum or magnetic-tape memory (it would check if it was in cache). If the flight information did not reside in drum memory, the reservation request was recorded on the "pending store" tape. It was held there until the master inventory was updated. The reasoning behind this was that chances were very good that the reservation request

[5] *What is Inventory Updating?*, TCAL internal monthly newsletter.

could be carried out since the flight was scheduled to depart after ten days. Also, the overhead of checking the magnetic-tapes for one reservation was too high.

Transactions

Reservation requests were made from remote reservation offices with transactor cards. A transactor card was a standard 7.5 x 3.5 inches punch-card. Cards were printed in a format such that it could be used for several kinds of transactions. In the ReserVec system, thirteen different types of cards that were capable of a total of 47 different operational functions were initially used. Sales agents made requests by filling in small circles that were needed for the transaction at hand. On the average, eighteen circles would be marked for a transaction. When the agent was finished making the appropriate markings, the card was inserted into a transactor terminal, the information was passed to the central registry, and the computer punched its reply in the right-hand edge: SOLS, SELL, ACPTD (wait-listing requests), and FCST (forecast of a flight)[6].

The central registry received its input from trunk-line data buffers and two paper-tape readers. The trunk-line data buffers connected the Gemini computer system and the communication lines that originated from the reservation offices[7].

The ReserVec computer system was a large-scale reservation system that was centralized (the central registry at the center) with reservation offices as its points of entry. It made a tremendous impact to Trans-Canada Airlines.

---

[6] *How to Make a ReserVec Reservation,* TCAL internal monthly newsletter
[7] *Electronic Seat Reservation System,* Process Control and Automation

4.1.2 SABRE and Confirm Computer Systems

American Airlines (AA) is a company that has gained immensely from their information (which includes reservation) system. AA's information system is run from a central system known as the SABRE (Semi-Automated Business Research Environment) computer reservation system. American Airlines' Sabre customer reservation systems network company, which was recently spun-off at a multibillion-dollar valuation, is an excellent example of a reservation system that transformed an industry. For many years, Sabre generated more profit for AA's shareholders than the airline itself. Sabre Inc.'s primary goal is to redefine the way business is done in every travel agency and/or company on and off the Internet.

Development

In the 1930's, American Airlines used the Request and Reply system as their reservation system. This system was based on centralized control of seat inventories maintained at a flight's initial point of departure. It required a sales agent to communicate with inventory control before a seat could be confirmed to a passenger. Since a passenger had to call a sales agent who in turn had to call inventory control to make a reservation, the passenger was forced to wait for a return phone call from the agent to confirm a reservation. This system, though it tackles the issue of concurrency, is by no means scalable.

Later in the same decade, to improve performance of the reservation system, AA's Boston reservations office determined that up to the point where the flight was almost sold out, sales agents could sell space freely and only report the sale. This significantly improved performance since the process of checking was eliminated along with the transmission of a second phone call. Inventory control would monitor sales, and when the number of available

seats decreased to a certain point, it would broadcast a "stop-sale" message to all its agents. This indicated to the agents that they must go back to the Request and Reply system discussed earlier[8].

Availability display boards had been installed in reservation offices to further minimize message traffic. By simply glancing at the board, agents could figure out seat availability. However, as passenger volumes grew, these offices became larger and more congested[9].

In the next couple decades, AA developed the Boston Reservisor, the Magnetronic Reservisor, and other modified versions of the systems. Even though each system was an improvement, AA's early reservation systems suffered from problems of processing speed and data inconsistency.

Near the end of the 1950's, AA began working with IBM to come up with a real-time computerized reservation system that was capable of handling a high volume of reservation transactions. This relationship proved to be very useful, and is still one of the main reasons why the SABRE network is able to handle such a high volume of data (this is discussed a little later). AA concentrated on the application programs while IBM delivered the necessary hardware and control program.

The processors at the heart of the early SABRE system were IBM 7090s duplexed for fail-safe redundancy. Another important hardware component for the system was random-access disk storage. This resulted in a 1,000 times improvement in performance from sequential magnetic-tape storing. The initial SABRE system's secondary storage was divided among six IBM 1998 drum files capable of holding 7.4 million characters and 16 IBM 1301 disk files with

---

[8] *The Sabre System: A Presentation*, Oct.1, 1964
[9] *American Airlines Sabre*, Boston: HBS, Case No. EA-C78

a capacity of 710 million characters. Seat inventories were recorded on the drums while passenger-name records were stored on disks[10].

IBM was responsible for the system time of 3 seconds during peak-volume periods. However, in April 1963, the response time was 30 seconds due to the addition of the New York office. In response, IBM doubled the main memory of the 7079s to 64K 36-bit words[11].

Current Hardware

With over 5,000 transactions per second during peak hours and over one billion transactions weekly, the hardware used by SABRE is crucial to its performance. The SABRE global distribution system (GDS) is responsible for more than 25% of the world's airline reservations and eighty million passengers daily.

Today, the SABRE system is composed of clustered S/390 processors and a real-time operating environment called Transaction Processing Facility. They provide power, scalability, and reliability and enable SABRE to be available 99.999% of the time. Of course, needless to say, this solution costs hundreds of millions of dollars – an unimaginable sum of money for a small business (Source: IBM Release).

New Technology – Porting to Java

The Sabre Group has recently teamed up with Sun Microsystems to port many of their solutions to Java. Sabre has agreed to resell Sun network computers to its customers and port its reservation management software to Java. In return, Sun has agreed to provide Java products and services to Sabre. Sabre currently states that it will adopt the Sun NC as its platform of

---

[10] *IBM's Early Computers,* MIT Press, 1961
[11] *American Airlines 'Sabre' Electronic Reservations System,* Western Joint Computer Conference, 1961

choice for its Qik-Access family of products. Qik-Access is almost an airline industry standard, handling reservations for 40 airlines and 75,000 travel agency desktops. Currently, Sabre and Sun are writing Qik-Access in Java. Sabre is making this move to Java because Java is a good choice of programming language for cross-platform computer network systems[12].

New Frontiers

Currently, the points of entry to the SABRE system are predominantly airline computer terminals, travel agency desktops, and the Internet. However, the Sabre group plans to add another new kind of point of entry by partnering with Nokia and IBM – the mobile phone. By using the Wireless Application Protocol (WAP), a new open industry standard for the mobile Internet, they plan to allow travelers to make reservations and changes over a mobile phone. Each of these three companies has a specific strength that enables them to add value to the project. The Sabre Group is developing the software that sends information and allows changes to be made; IBM is providing application development tools and specialized hardware which translates the information from the SABRE system into a condensed form that can be sent to the mobile phone; Nokia is providing the servers as well as the Nokia phones that will receive this information (refer to figure 4.4 for the architecture)[13].

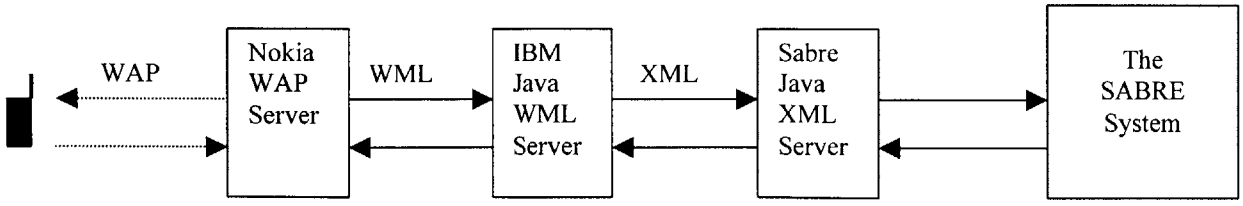

Figure 4.4 SABRE Wireless Architecture

[12] *Informationweek Online, June 3*
[13] *IBM Java News, January 2000*

Confirm Computer System

AMRIS (AMR Information Systems), besides successfully designing and developing the SABRE system, was also active in researching other computer systems. In March 1998, AMR Information Services and other large hotel and travel companies in the US, decided to create a computer system whose information and services would be available to all clients. This system was named Confirm. It was conceptually a distributed heterogeneous system composed of large homogeneous systems; somewhat similar to the ORS discussed in this paper. It was planned that Confirm would be a full function system capable of handling cross-reservations (car rentals and hotel reservations), building customer profiles, and other services. The development team included 500 people, 200 of which were programmers.

Confirm was going to be launched in 1992, but after completing the beta version of the system, many major problems were found in the system. Due to these problems and other reasons, Confirm was cancelled in mid-1992.

4.1.3 Pegasys and Shangri-La

Besides the airline industry, another industry that relies heavily on real-time reservation systems is the hotel industry. An example of a successful reservation system is the one used by Asia-based Shangri-La Hotels and Resorts. With the help of THISCO™ (The Hotel Industry Switch Company) and Dallas-based Pegasus Systems, Inc., Shangri-La has been able to connect over 35 hotels and resorts in Southeast Asia and China, enabling electronic reservations by travel agencies all over the world.

The reservation system Shangri-La has in place is different from standard airline reservation systems. The entire system architecture is different, but each individual node is

similar to an airline reservation system. That is to say, each hotel system, though similar to other hotels, has different data, room types, et cetera. Also, each hotel "looks" at a different data set. Consequently, the load is distributed amongst the different hotel systems, and performance and concurrency are not major concerns[14].

As mentioned earlier, each individual hotel system is similar to the system depicted in figure 4.1. The different hotel operators and points of entry connect to its central system. However, because hotel systems are not bombarded with nearly as many transactions as typical airline reservation systems, performance requirements are much easier to meet. Also, because of

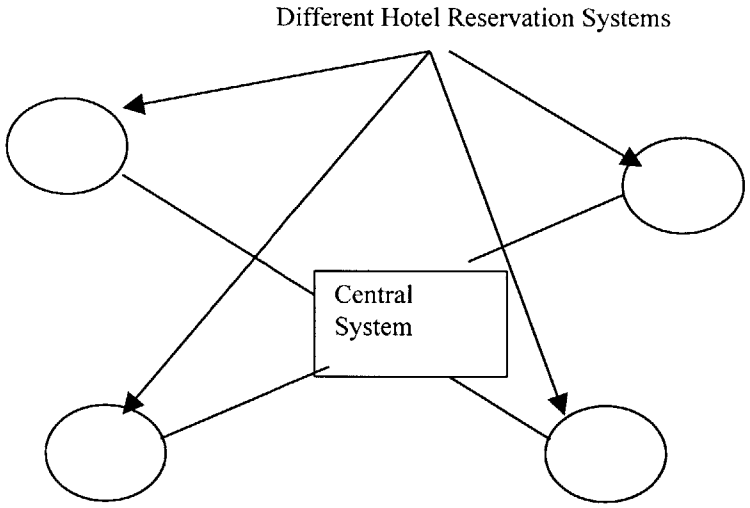

Figure 4.5 Large Hotel Reservation System

the costly hardware in place at the different hotel reservation systems as well as the numerous entry points in each individual system, hotel reservation systems do not have to address the issue of connectivity. Connectivity is something they have control over, unlike in the case in the ORS.

[14] Pegasus Systems, Inc. Immediate Release for October 6, 1997.

4.1.4 Ticket Master Systems (TMS)

Tickets in the sports and entertainment industries can often be purchased via the Internet and over the phone. Businesses that offer this service generally charge the customer an extra surcharge for the service. Tickets are then collected from automatic dispensing units at the venue or from a person at a special pickup/reservation counter.

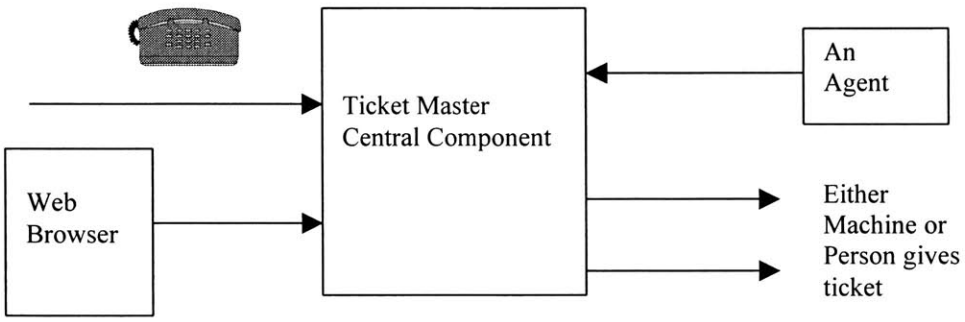

Figure 4.6 A General Ticket Master System Schematic

Figure 4.6 shows the general layout of a Ticket Master System. Like a restaurant reservation system, there are multiple ways in which a ticket can be booked: over the Internet, via telephone, and by physically going to an agent. These are different points of entry. Because there is more than one point of entry, TMS's must be designed so that they can handle concurrency issues during the reservation process. Two customers should never be assigned to the same seat. Similarly, in a restaurant, two customers should never be assigned to the same table in overlapping time intervals.

It can also be observed from the figure, that there are multiple ways in which a ticket (reservation record) can be retrieved. One must be careful while analyzing the system. Ticket retrieval is not entirely analogous to a schedule lookup – there is a subtle, yet very important

distinction. When retrieving a schedule, handling concurrency is not important. A schedule is a transitory state of the system at any particular time. However, during ticket retrieval, concurrency is important. It is possible that a person might attempt to retrieve two tickets with the same reservation from two different machines at the same time. Due to possible race conditions, that can occur if concurrency is not handled. This may sound like an unlikely situation, but race conditions can create havoc from time to time.

4.1.5 .COM Reservation Portals

Sabre and Microsoft have created successful reservation portals www.travelocity.com and www.expedia.com respectively. According to Forrester Research, over 24 million US households research trips and Travelocity and Expedia are well positioned to capture large shares of this growing market. Expedia and Travelocity allow the user to reserve flights, hotels, cars, and cruises.

Expedia and Travelocity provide an excellent way of making reservation via the Internet. This also allows the consumer to view other material before making his/her decision. Expedia and Travelocity. They serve as an interface to larger reservation systems; for example, Travelocity connects to the SABRE computer system for airline reservations.

Even though Expedia and Travelocity offer excellent services, they are limited to larger companies who have reservation systems in place. Customers are not able to make reservations at smaller businesses such as small motels and local restaurants. The ORS is an affordable reservation system for restaurant owners that allows customers to make reservations over the Internet through a similar portal.

4.2 Other Relevant Systems and Technology

Besides looking at existing reservation systems, it is important to analyze other computer systems, solutions and technologies with properties that are desirable in the Online Reservation System at discussion. Different computer systems exhibit different properties. By observing specific properties, different design features can be modified and reused in the ORS.

4.2.1 Cisco DistributedDirector

Cisco Systems Inc. specializes in network solutions. They provide many different products and services, such as routers, switches, telephony, et cetera, to large and small businesses. They also specialize in load-balancing and distribution solutions.

Cisco Systems DistributedDirector software is a software that provides scalable Internet traffic load distribution between dispersed servers. The design of this sofware is useful in developing the restaurant distributed system.

Figure 4.7 below shows how a Cisco DistributedDirector helps balance Internet traffic load across multiple servers. Suppose A, B, and C are identical websites located at different geographical location (different IP addresses) that map to www.companyxyz.com. In the scenario depicted by figure 4.7, the Cisco DistrubedDirector[15] accepts HTTP connections for www.companyxyz.com. Then, after figuring out which servers map to the domain name (A, B, C in this case), it finds the "best server" at that particular time for that particular client, in this case server C. It then sends the HTTP status code "302 Temporarily Moved" with the new URL for the "best" server to the client. The user is then transparently connected to the web server specified by this URL.

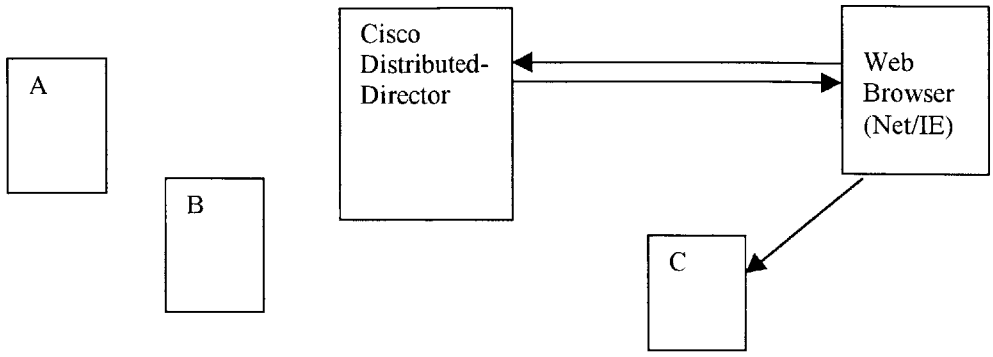

Figure 4.7 Cisco DistributedDirector – HTTP Redirect Mode

In a distributed N-client restaurant reservation system, a similar component is needed to "push" the reservation request to the appropriate restaurant system. However, the layout and functionality of the server is different. In the case of the restaurant reservation system, the browser does not connect to the central server directly. Instead, a connection object spawned by the web server, discussed later in the paper, pushes the request to the "central server", which in turn pushes it to the appropriate client system.

4.2.2 Server-side Scripting

Server-side scripting has become a common practice in Internet application development. Almost all web developers use some sort of server-side scripting. CGI (Common Gateway Interface) and Java servlets are examples of server-side scripting. A web browser invokes these scripts by making an HTTP connection to a particular URL; the web browser is performing a Remote Procedure Call (RPC). There are several advantages to server-side scripting and RPC.

[15] A description of the product can be found at http://www.cisco.com

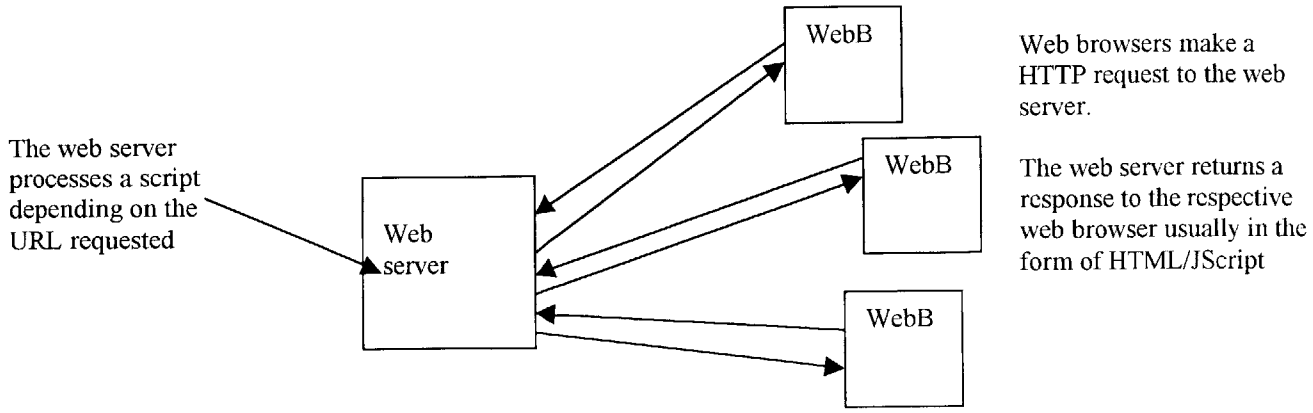

Figure 4.8 Many-to-one Server-side Processing

One of the advantages is that any change that needs to be made only needs to be made at one location. Suppose there was a bug in a program. If the program were located centrally, it would simply have to be changed in one location. Another advantage of server-side processing is that the software only has to be tested on one platform and one system. All possible scenarios do not have to be tested. In the case of client-processing, different software must be created for different platforms.

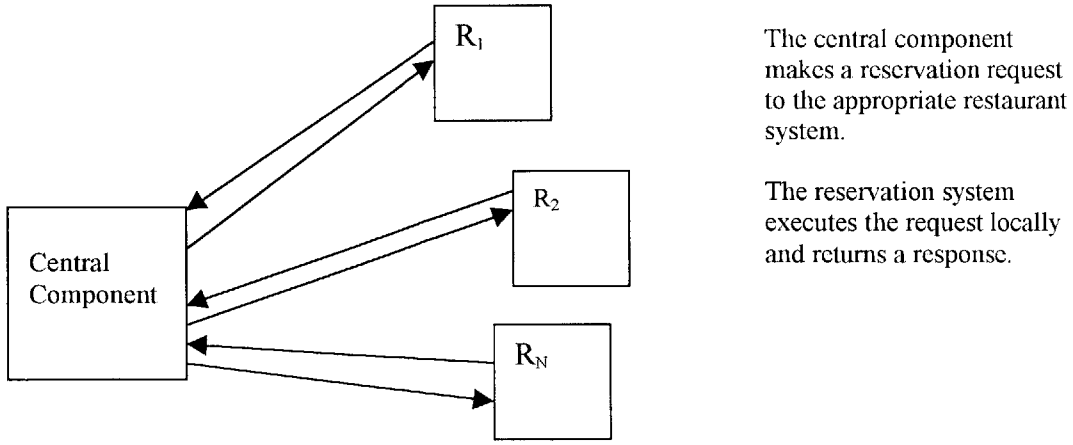

Figure 4.9 One-to-many Server-side Processing (Restaurant System)

Along with its advantages, there are disadvantages of server-side execution. By executing everything on the server-side, one can slow the system down significantly. This may hold true for a web server, but fortunately, the reverse is true for the restaurant reservation system. As discussed later in the paper, by pushing the request to another server, the system is less stressed. This is because a funneling effect takes place. The central component pushes requests (performs a RPC) to different restaurant systems as can been seen in figure 4.9. Therefore, unlike the many-to-one scenario in the web server case, it is a one-to-many situation.

4.3 Current Restaurant Reservation Systems

Currently, most restaurants do not have an Internet presence and have no way for customers to make reservations online. The few restaurants that do allow customers to make reservations have limitations on their reservation system.

The most common implementation of a restaurant "reservation system" is a form of email. Customers request a time slot at a particular restaurant and the information is emailed to a group of centrally located individuals. They then contact the restaurant and attempt to make the reservation for the customer. This is highly inefficient and does not address many of the problems in the conventional reservation method. In this business model, the restaurant pays a monthly fee and/or commission to the company that directs the reservations.

Another new and creative idea has been implemented to create specialized reservation hardware for restaurants. Unfortunately, the company has been very discreet about their product to prevent competitors from copying their technology. Hence, not much is known about their

system. However, because it is specialized hardware, making modifications and/or additions to such a system would be a difficult task.

Chapter 5

The Design

As mentioned earlier, the ORS is a distributed N-client computer system. The computer system consists of a central server and N-clients (the N-clients being the N restaurants).

The ORS is designed to fulfill all the objectives stated in the paper. The diagram below is a high level view of the ORS computer system.

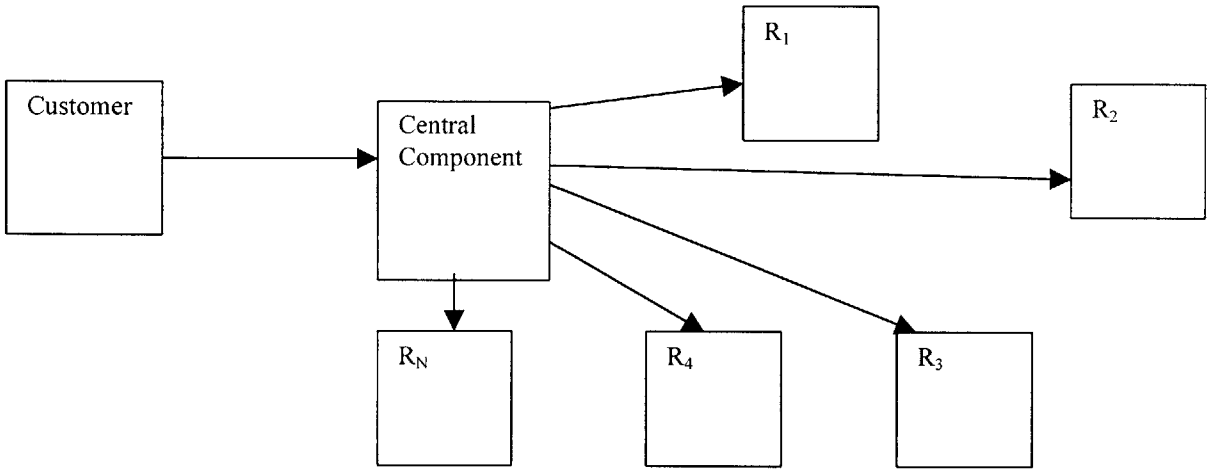

Figure 5.1 N-Client Distributed System

$R_1,\ldots,R_N$ = Restaurants

5.1 Central Component

The central component consists of the web server, central server, and the central database. It is responsible for handling the reservations made over the Internet.

The customer, using a browser such as Netscape or Internet Explorer, connects to the web server. There, he posts a request to the web server, which in turn invokes a Java servlet. This Java servlet is then responsible for handling the customer request. There are two main types of customer requests :

i) Simple Query

A RezzSys Simple Query (RSQ) simply queries the central database. This type of request does not change the state of the data in any way, but merely returns particular information depending on the query. Some examples of RSQs are querying for restaurant information (static information on the type of restaurant, location, et cetera) and checking for seating availability at a particular restaurant and time. These requests do not require any interaction with the restaurant component, but require interaction with the central database (as depicted by the dashed lines in Figure 5.2)

ii) Reservation Request

A RezzSys Reservation Request (RRR) is more complicated than a simple query. When a customer wants to make a reservation over the web, the web server invokes a Java servlet, which in turn, "talks" to the RezzSys Central Server (RCS). The servlet delegates responsibility to the RCS which in turn delegates responsibility to the restaurant component.

Below is a schematic of the RezzSys Central Component (RCC).

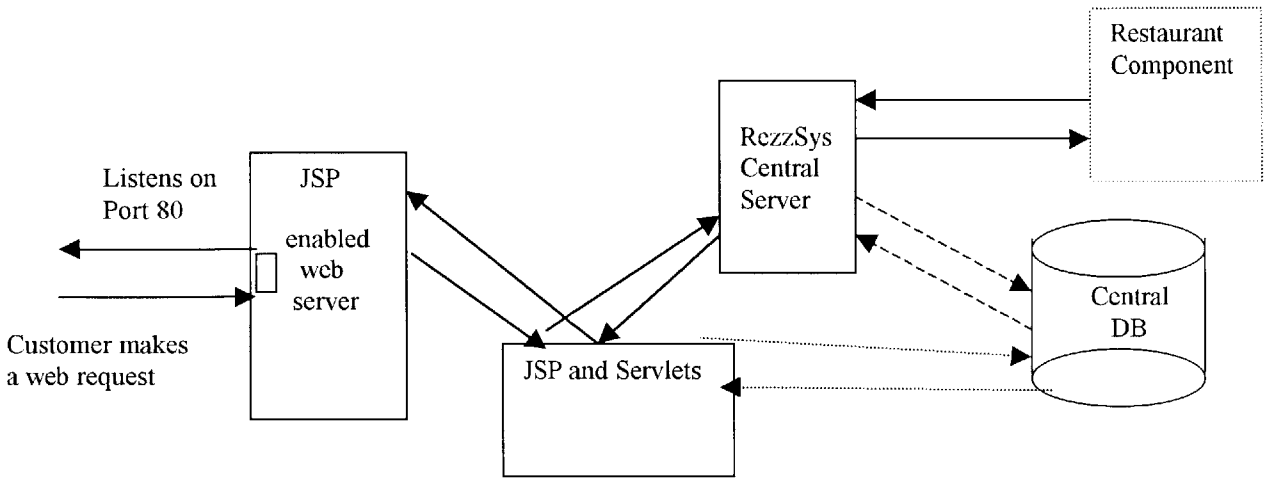

Figure 5.2 RezzSys Central Component (RCC)

5.1.1 Web Server and Servlets

The web server is responsible for making the web site available to the customers. It allows the customers to access the web site and make reservations. They can also browse through other information such as reviews, hot spots, et cetera. The actual web server used in the system is irrelevant as long as it is multi-threaded and supports Java servlets. That is to say, it can handle multiple requests at the same time.

Servlets serve as the request handlers of the web server. When a customer presses "Submit" on the HTML form, the servlet is invoked by the web server and the input parameters are passed to it. The servlet then does one of two things:

i) If the request is a RSQ, the servlet simply queries the central database making a database connection and carrying out a SQL command.

ii) If the request is a RRR, the servlet makes a connection to the central server and asks it to execute the reservation.

5.1.2 Central Database (RCDB)

The term "central database" is misleading as all the data does not lie on the same machine. In fact, the central database is a distributed database across different servers, across different physical machines. The reason for this distribution is to enhance performance; the more processors each data component has, the faster the retrieval time.

The central database has several properties:

i) It is always connected to the Internet (this assumption is valid since ASPs (application service providers) guarantee connectivity with a probability greater than 99%).

ii) The central database has general data on all the restaurants in the system. This includes information such as type, geography, atmosphere, menu, et cetera. This information is not replicated on the restaurant local databases.

iii) The RCDB contains, besides restaurant general information, reservation information about each and every restaurant. This is to insure that if a hard disk failure were to occur, a back-up copy would be available.

iv) The reservation information of each restaurant may or may not be identical to the reservation information that can be found on the particular restaurant system. The assumption is that the latest copy of data is on the restaurant side. For example, the local database residing on restaurant A's system has the most up-to-date reservations and data. Due to time lags and disconnectivity, it is possible that the restaurant A data on the

central database is not equal to the restaurant A data on restaurant A's local database at a given time T.

The central database is a vital component of the Online Reservation System for a number of reasons. Firstly, it provides a solution to the "hard disk crash" scenario possibility. Even though each of the local restaurant systems has a copy of its own specific data, it is important to have another copy present in case any of the restaurants experience a hard disk crash - something that is intolerable in a real, business, practical environment. In case of a hard disk failure, restaurants can call a centralized location for hard copies of their reservations. Eventually, when the computer system is up and running at the restaurant, the data can be efficiently copied from the central database to the restaurant local database by executing a program.

Secondly, the RCDB is important because it provides a compromised solution to the issue of connectivity. Suppose, for instance, restaurant A is not connected to the Internet. That is, the RCS does not have a record of restaurant A's IP address and port number. In this situation, if there were not a second copy of the data available, the user would not be allowed to carry out any requests. RSQ's, for example, would not be able to get carried out. By having the RCDB, the servlets can simply query the RCDB. The reason why the RCDB is a compromised solution and not a complete solution is because customers, even with the RCDB, cannot make or delete reservations when the restaurant system is not connected to the Internet.

Even though there are two copies of data a reservation cannot be made over the web at restaurant A when restaurant A's reservation system is not connected to the Internet. The reason is quite clear. System A could have made and deleted reservations to its own local system (local

database) when the system was disconnected. Because the central server does not know what reservations have been made and deleted, it cannot make any assumptions.

Besides data back-up and connectivity, the RCDB improves performance and allows miscellaneous information to be stored that is not relevant to restaurants. Performance is improved simply because the data transfer time is reduced. Because the servlets are running on a machine close to the RCDB, the number of hops to and from the database is very low. If the web server and the central database, for instance, resided on the same machine, then the number of hops would be zero since the database call would be local. On the contrary, if RSQ's resulted in querying specific restaurant databases, this would result, on average, in a greater number of hops related to, but not entirely based on, the physical distance between the restaurant and the central server. Also, not only is the mere number of hops a direct contributing factor to performance in terms of "distance", but the greater the number of hops, the more likely the lower the data transfer rate.

The central database holds a lot of crucial information. Some of the information it holds is information specific to the central database and is not stored on the restaurant side. The table that can only be found in the central database is the *restaurants* table.

```
create table restaurants (
        rid             number(9) not null primary key,
        name            varchar(50) not null,
        contact_name    varchar(60) not null,
        email           varchar(100),
        address         varchar(2000),
        food_type       varchar(20) check (food_type in ('italian', 'indian', 'chinese',
                        'thai', 'ethiopian', 'korean', 'american', 'malay', 'singaporean',
                        'french', 'afghani', 'japanese')),
        price_range     varchar(20) check (class_type in ('low', 'middle', 'high')),
        city            varchar(30),
        state           varchar(30),
```

```
    zipcode         varchar(10),
    country         varchar(30),
    location        varchar(50),
    size            varchar(10) check (size in ('small', 'medium', 'large')),
    atmosphere      varchar(50),
    start_time      varchar(10),
    end_time        varchar(10),
    interval        varchar(10),
    twopages        char(1) check (twopages in ('y', 'n')),
    start_time1     varchar(10),
    end_time1       varchar(10),
    start_time2     varchar(10),
    end_time2       varchar(10),
    duration1       varchar(10),
    duration2       varchar(10),
    smoking         char(1) check (smoking in ('y', 'n')),
    reserve         char(1) check (reserve in ('y', 'n'))
);
```

SQL Statement to create the "restaurants" table

5.1.3 Central Server

The central server is a multi-threaded server that is a very important component of the entire online reservation system. It is responsible for load balancing, making remote procedure calls (RPC) to the different restaurant systems, and making updates to the central database.

The components that the central server must interact with are the servlets. The servlets forward the reservation web requests to the central server. The RezzSys central server (RCS) carries out these requests by asking the restaurant system to "make the reservation" and check whether the restaurant reservation can be made. Once the restaurant reservation system confirms or rejects the reservation, the RCS returns a commit/rejection to the servlet.

In order for the RCS to make remote procedure calls to the restaurant system, it must be aware of which restaurant systems are up and which systems are down. It also must be aware of

which IP address the system is running on and what port the system is "listening" on. The RCS stores the restaurant IDs and their corresponding IP address and port in a table in memory.

Rest ID	IP, port
10	12.32.13.199

Figure 5.4 IP Client Hashtable used by Central Server

Processes 1-3 below describe how the RCS keeps the table of restaurant ID's and connection information up-to-date.

Process 1

```
while (true)
   for i = 1 to table.size
        Open connection to table[i].IP, table[i].port
        if connection cannot be established
             Delete_from_table(table[I].restID)
        else
             Ask if system is up and running properly
             Wait until message received or timeout
             If message_received != "ok"
                 Delete_from_table(table[I].restID)
   end for loop
   sleep(100)
 end while
```

Process 2

```
while (true)
   wait for an outside connection
      when an outside connection is made spawn Process_3(connection)
end while
```

```
Process 3(connection)
    add(connection.id, connection.ip, connection.port) to table
    send acknowledgement back to restaurant
```

"Single Point of Reservation"

As mentioned earlier, the RCS is also responsible for making remote procedure calls to the restaurant system; it makes reservations by invoking methods on the restaurant side. The RCS can only return a confirmation back to the servlet when the restaurant system "checks and adds" the reservation. Processes 4 and 5 below describe how the RCS accomplishes this task.

```
Process 4
while (true)
  wait for a servlet to make a connection
      when a servlet makes a connection spawn Process_5(connection)
end while
```

```
Process 5(connection)
    If table.get_IP_Port(connection.rid) exists
        Open a connection2 to table[connection.rid].IP, table[connection.rid].port
        Pass connection.request to connect2
        Wait for response from connection2
            If connection2.response = "ok"
                Return "ok" to connection (servlet)
            Else return "not_ok" to connection (servlet)
      Else return "system not up" to connection (servlet)
```

Pushing the responsibility of checking and making a reservation to the restaurant side is a very important design feature. One could have designed the system such that the RCS updated the central database and then pushed the information to the client side. Another approach would have been to have the RCS update the restaurant side database directly without having to go through the restaurant system. However, both those approaches can have bad side effects.

The approach taken in the design of the Online Reservation System ensures that there is only one point of reservation; only the restaurant side makes the decision of reservation. Having only one point of reservation has numerous advantages. Synchronization becomes significantly easier and system performance is enhanced.

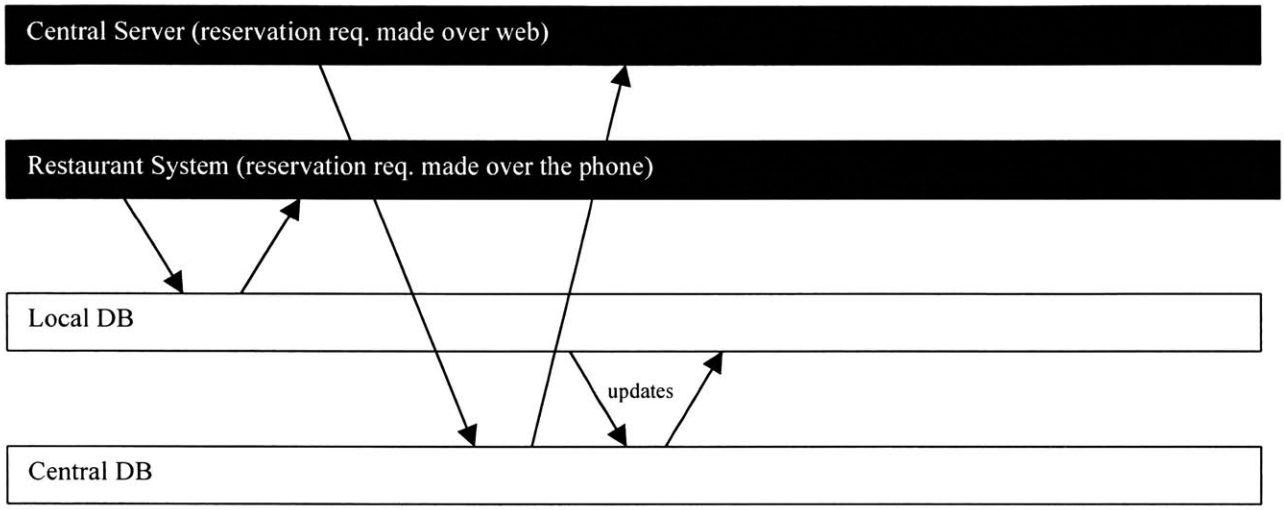

Figure 5.5 More than One "Point of Reservation"

Figure 5.5 shows what problems would arise if there was not a single point of reservation. The above scenario depicts what would happen if a person A tried to make a reservation from the web while a person B was calling the restaurant to make a reservation. The diagram assumes that only person is making a reservation via the Web at any given time.
In order for a reservation to be made, there are two steps that need to be performed:

i) Check whether the reservation can be made

ii) Make the reservation

Now, assume person A makes a reservation for a particular table, say 10, from 1pm - 2pm. Now assume that person B is making the same exact reservation except he is physically calling the restaurant. In the case of person A, the central server checks to see whether the reservation can be made. Assume that the table is available at that particular time. So, the central server adds the reservation record to the database. However, before the local database on the restaurant side is updated, person B's reservation was also checked and updated. Now, there exist two identical reservations for the same table. This is a constraint violation.

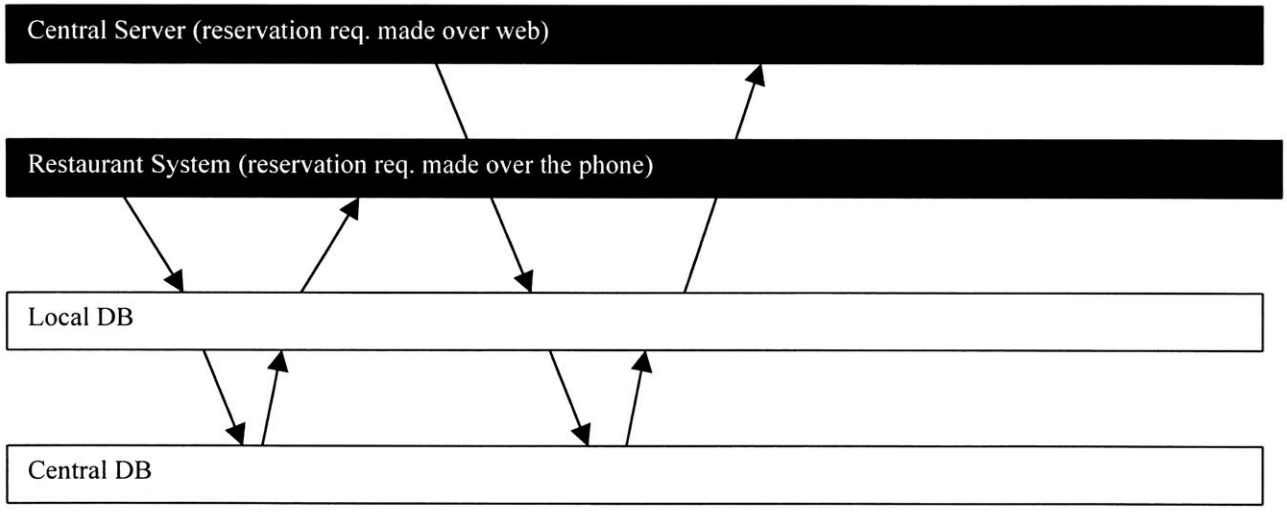

Figure 5.6 Single Point of Reservation

In figure 5.6, however, by having one "point of reservation" the problem is eliminated. When person A makes a reservation over the web, since there is only one point of reservation, there is no conflict.

5.2 Restaurant Component

The restaurant component consists of a restaurant local server, local database, and the stand-alone application that runs on a computer in the restaurant. In fact, all the software components reside on a computer in the restaurant. The stand-alone application can be further broken down to the graphical user interface (GUI) and the "back-end system". The back-end system is composed of a system driver and other software objects.

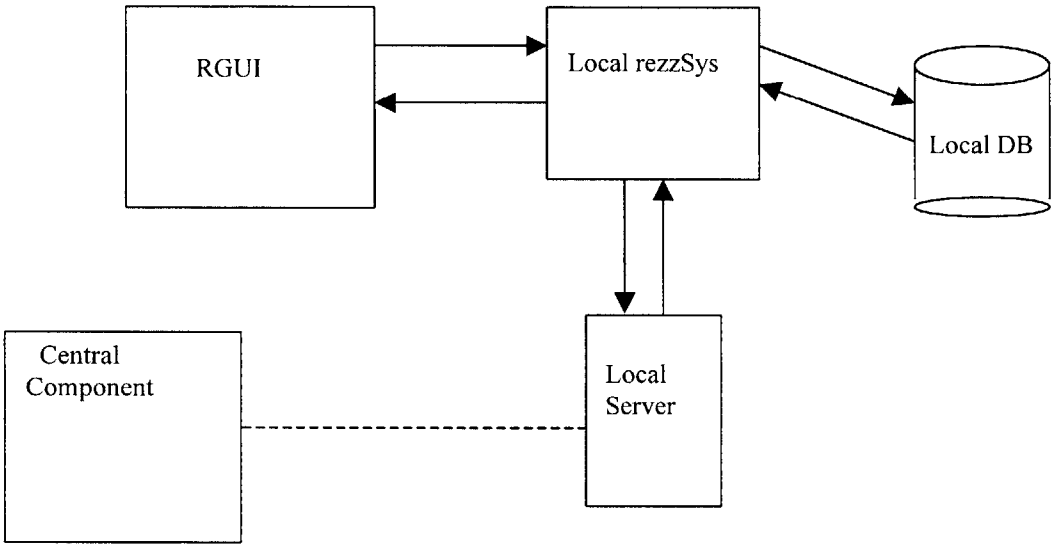

Figure 5.7 Restaurant Component

5.2.1 RezzSys Graphical User Interface (RGUI)

The graphical user interface is very important. It should be user-friendly and easy to use. The maitre d' should be able to manually enter reservations. He/she should be able to choose the actual physical table he wants for a reservation. For instance, if Joe Bloggs calls the restaurants and asks to be seated at a particular table, say table 10 without loss of generality, then the maitre d' should be able to choose table 10 and a particular time slot and make a reservation. At the

same time, the maitre d' should be able to see what tables are available at a particular time and date very easily and readily since the time on the phone with a particular customer is limited.

Besides allowing the maitre d' to enter reservations into the system manually, he/she should be able to do a "Quick Enter". That is to say, enter a reservation without having to look for an open table and/or finding an open time slot. The quick enter option should simply take in the customer phone number, date, time, and the table number (optional) to make a reservation. This makes the work of a maitre d' significantly easy, especially if the reservation that is needed to be made is on a different day - for example, two weeks in advance. It saves him the time and trouble of switching days ("jumping" to another date). Other important GUI features include ease of canceling reservations and changing reservations. Also, it should be easy for the maitre d' to pull up a customer profile of a customer, and enter customer information in an easy manner.

Even though the restaurant system comes with a standard RGUI, it is simply the skeleton and can be modified to suit the needs of the particular restaurant. This can be accomplished because the RGUI is entirely separate from the back-end. Currently, the RGUI adjusts itself to specific restaurant needs by reading a configuration file on run time. The configuration file has all the start-up information for the particular client. This allows for a general purpose RGUI to be built for all the restaurants. The configuration file has the following information:

- The time difference between reservation times.
- The time the restaurant open.
- The time the restaurant closes.
- The approximate time that a table gets occupied for.
- The port the central server (RCS) is running on.
- The host the RCS is at.
- The name of the logfile.
- The local database driver
- Restaurant ID
- Whether or not there are two pages or one page (a split screen)

Conceivably, the RGUI could be more flexible (more configurable) and the RGUI can itself be customized. The basic RGUI can be seen in figure 5.8

As can be seen in figure 5.8, the RGUI is intuitive and easy to use. In this particular example, there are two pages: one for the "morning" and one for the "evening". The particular starting and ending times are specified in the configuration file described earlier.

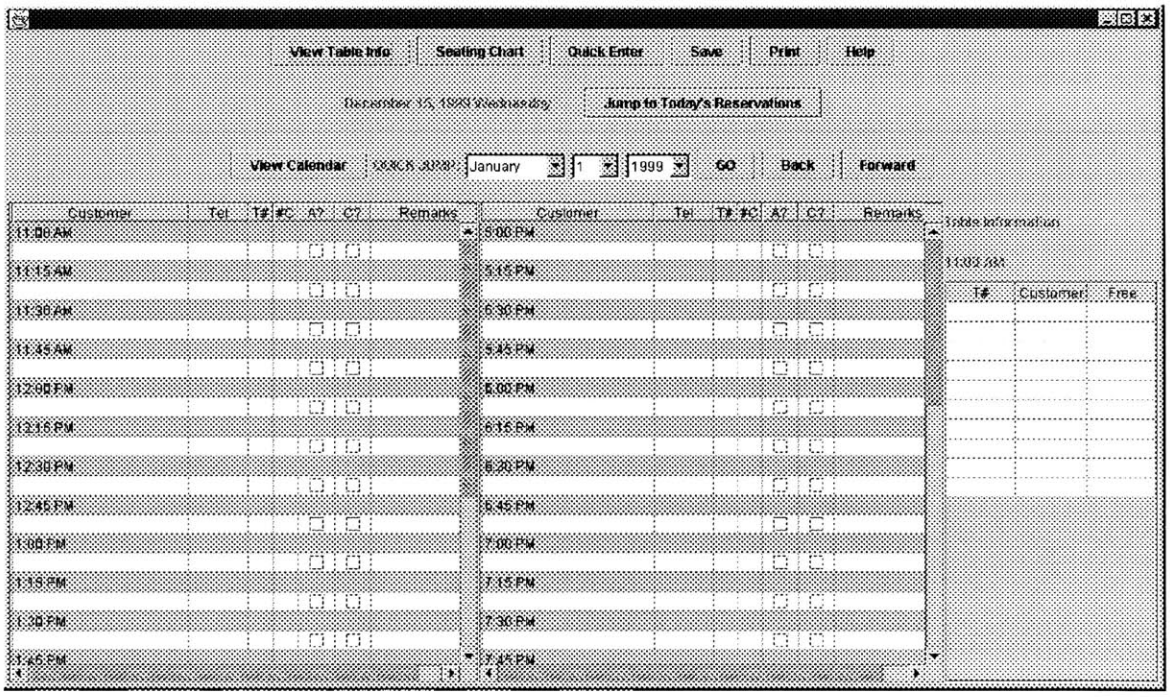

Figure 5.8 RezzSys Graphical User Interface (RGUI)

The rows represent the different reservation records on a particular day (the day is displayed at the top of the frame). A maitre d' can manually add a reservation record by simply right clicking on the mouse and choosing the add reservation option. This will add a reservation to the database (if there is no conflict) and will visually add it to the table.

The right hand side displays all the table information for the particular time slot highlighted. It indicates which tables are open, which are booked, and what time unavailable tables will free up. This makes it very convenient for the maitre d' to find out what tables are available and which ones are not while talking on the phone with a customer.

The maitre d' can also add a reservation for a different day by simply clicking on the "Quick Enter" (QE) button located at the top. By clicking on the QE button, another frame pops up. The maitre d' can then enter any date, customer information, and the relevant reservation information. He has the option to specify a table but it is not necessary.

The maitre d' can also jump to any day. He simply has to enter in the day for which he would like to see reservations for and press on the "Go" button for the Quick Jump. This changes the view and loads the relevant reservation information for that day.

The RGUI, besides allowing the maitre d' to enter reservations easily, also visually informs the maitre d' when a reservation has been made over the web. If a person makes a reservation for the same day that the RGUI is displaying (i.e. if the maitre d' is viewing the reservations for the 4th of January, and a person makes a reservation for the 4th of January), then a reservation record is added automatically to the RGUI and a small white message pops up in the top left hand corner that the maitre d' can click away. If someone makes a reservation for a different date, a record is not visually added to the GUI and only the message pops up.

5.2.2 The Back-End System

The back-end system consists of all the software components on the client system except for the GUI. It consists of a local server, a local database, a system driver, and other key software components.

The Local Server

The local server is a multi-threaded server that is spawned as a separate process when the stand-alone application is initialized. It is responsible for listening to requests made by the central server on the web server side. One of its primary responsibilities is to carry out requests pushed by the central server. The local server is one of the components that allows for Remote Procedure Calls (RPC) to be carried out.

The local server simply listens on a particular port (1976 for example) and waits for a connection to be made by the central server. When a connection is established, it spawns off a separate thread – a connection object. The connection object then engages in a dialogue with the RCS, using a predetermined protocol.

```
carry_out_request(Connection)
    result = makeReservation(Connection.CustomerName, Connection.Time,
Connection.Table(Optional))
    if result = true
       return to Connection "Success"
    else
       return to Connection "False"
```

Local Server listens for a connection and then carries out request.

However, as mentioned earlier, reservations are made through only one point of access – the restaurant system. This prevents concurrency issues that can arise from having two points of reservation.

When the local server receives a reservation request, it calls *reservations'* makeReservation method. *reservations* is a an object with static members and methods; all the methods can be used without creating a reservations object; all the members of reservations are shared. Because the local server is a process that the RezzSys launches, it runs in the same

environment as the entire system. That is to say, when the local server calls the reservations object, it is in fact calling the same reservations object that exists in the RezzSys system. If a reservation is successfully made, then the makeReservation method returns "true", otherwise it returns "false".

Besides carrying out reservation requests, other important but less complicated tasks that the local server performs is sending an acknowledgement to the central server. As mentioned earlier, the central server routinely checks to see whether the restaurant system is up and connected. Consequently, it is the responsibility of the local server to return an acknowledgement every time the central server checks to see if it is alive.

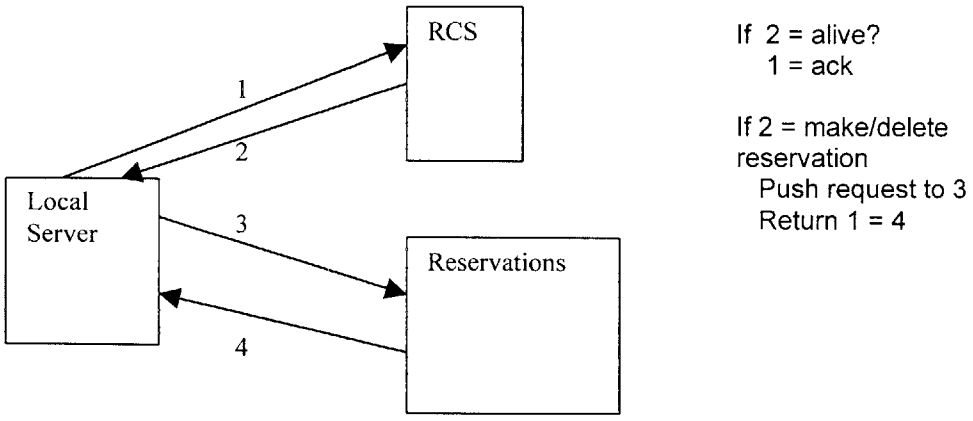

Figure 5.9 Acknowledgement Process

The Local Database

The local database stores all the reservation information for the restaurant. This information, besides being present on the local database side, is also present in the central database. The words in square brackets represent the prefix of the table name on the central database side.

```
create table [restaurantid_]restaurant_tables (
        table_number        varchar(4) not null primary key,
        capacity            varchar(4) not null,
        fuzzy               char(1) check (fuzzy in ('y', 'n')),
        fillrank            number(2),
        reservable          char(1) not null,
        smoking             char(1) not null
);
```

SQL Statement to create the "restaurant_tables" table

Each record in the table restaurant_tables represents a physical table in the restaurant. This table is restaurant dependent and is directly related to the physical environment of the restaurant. The table is set on initial installation of the system on the restaurant side and is changed only if a new table is added or if a table property changes. Each table (record) has the following properties:

i) Table Number

Each physical table has a table number. This is true of practically all restaurants. This is the number of the table on their seating chart, for example.

ii) Capacity

This can simply be described as the number of "predetermined chairs at the table"; the number of people that can be seated at the table.

iii) Can the table be crowded? (fuzzy)

This is an interesting, not so obvious field. It is a binary field that can either hold the value yes or no. What this value indicates is whether the table can be "crowded"; whether an extra chair can be added without discomfort.

iv)The Fill Rank Order (fillrank)

This is a field that is used in the reservation algorithm discussed later in the paper. It is a value set by the owner, manager, or maitre d' of the restaurant to signify the order in which he/she prefers the tables to be reserved.

v) Reservable? (reservable)

This is another boolean variable that is either "yes" or "no". It indicated whether the system can reserve the table automatically or not. If the reservable variable equals "yes", then the system can reserve the table for a customer assuming all the reservation criteria is met (discussed in depth later in the paper). If the value is "no", however, the table can only be reserved manually; the maitre d' can only reserve the table on the client side application. This is an important feature since many times restaurants have VIP tables that should not be assigned by the system.

vi)Smoking? (smoking)

Another binary variable that indicates whether the table is in the smoking section.

```
create table [restaurantid_]reservations (
        cid                 number(9) not null references customers,
        start_time          date not null,
        end_time            date not null,
        people              varchar(4),
        smoking             char(1) check (smoking in ('y', 'n')),
        table_number        varchar(4) references [restaurantid_]restaurant_tables,
        misc                varchar(2000)
);
```

SQL Statement to create the "reservations" table

Each record in the reservations table represents a reservation of a table; Joe Bloggs reservation from 1 to 2:30pm on the 31st of December for example. Each record consists of the following fields:

i) Customer ID (cid)

Customer ID holds the "id" of the customer. It is basically a reference to the customer in the customers table.

ii) Start Time (start_time)

Start time is the time (actually, it is stored as a time and date) the reservation starts from.

iii) End Time (end_time)

End time is the time the reservation ends at.

iv) People (people)

People is the number of people the reservation is made for.

v) Smoking (smoking)

Smoking is a boolean value that can only take on values yes and no. This is important since many times customers have strong smoking preferences.

vi) Table Number (table_number)

This is the table number that the reservation is for. Most restaurants have physical table numbers for each table in the restaurant.

vii) Miscellaneous (misc)

This is any information about the particular reservation; for instance, a customer may want a special wine at dinner or may want flowers at the table.

```
create table [restaurantid_]customers (
        cid             number(9) not null primary key,
        fname           varchar(30) not null,
        lname           varchar(30) not null,
        email           varchar(100),
        phone           varchar(25),
        smoking         char(1) check (smoking in ('y', 'n')),
        misc            varchar(2000)
);
```

SQL Statement to create the "customers" table

Lastly, the local database holds a customer table. This contains the customer information where each record represents a customer. This table is configurable by the restaurant; a restaurant can choose to have customized fields. Different restaurants may want to have different fields; just like different websites have different user registration information that they profile. The most basic fields that the customer table has, which is common to all restaurants, are customer id (cid), first name (fname), last name (lname), email address (email), phone number (phone), smoking (smoking), and miscellaneous information about the customer (misc).

Local Reservation System (RezzSys)

The local reservation system (RezzSys) is the software that runs on the client machine. It drives the RGUI, communicates with the local database, spawns a local server, and is responsible for everything from scheduling to updating the data. The RezzSys is also responsible for connecting with the central server. On startup, RezzSys registers with the central server. It makes a socket connection to the central server port (specified in the configuration file - central server IP and port). After making a connection, the RezzSys registers its IP address, the port that its local server is running on, and its unique restaurant id. For example, if the restaurant machine IP address is 12.34.202.3, and its local server is running on port 1942, and the central server resides at server.mit.edu on port 1977, RezzSys would connect to the central server at server.mit.edu on port 1977 and transmit:

IP : 12.34.202.3
Port : 1942
RID : 77
(or whatever the restaurant id is)

RezzSys is also responsible for all the scheduling, "getting" and "putting" of data, and driving the RGUI. It is an object oriented design that interacts with different "objects". These "objects" either serve as static interfaces or as actual objects residing in memory.

The RezzSys is a modular computer software system which is composed of many software components. At the initialization stage, the RezzSys launches two main objects : the RGUI and the local server. It also instantiates key software components namely the "reservations" and "tables" objects. The reservations interface basically takes care of all the reservations. It contains different methods for making reservations, deleting reservations, changing reservations, et cetera. The reservations and tables modules in turn use the rezzDB module that handles the physical puts and gets. rezzDB allows updates and retrievals from the local database. One can think of it as an "interface" that handles the database transactions.

"Tables" Object (TO)

The tables object is a singleton object. This means that only one instance of the object can exist in memory; two "tables" object can never exist in the same environment. The specifics of the implementation are discussed later in the paper. The TO is crucial for the RezzSys, and in fact, is a form of cache: table cache. The information contained in the TO is the latest information for that day; it is the most up-to-date information. Consequently, it is used extensively by the reservations object to enhance performance.

The TO stores the information about the state of the tables for a particular day. It uses the rezzDB interface to get the initial table information from the local database. After it captures all the table information, it sets each of its state depending on the day and the reservations of that day; it uses rezzDB to get all the reservations, and sets the state of the table accordingly.

In order to improve performance, the table information is stored in a particular way; it is stored according to the capacity of the table. This allows the system to identify which tables satisfy the capacity constraint. Not only are the tables stored according to capacity, but the tables within each capacity group are also sorted according to its fillrank. This facilitates the reservation algorithm discussed later.

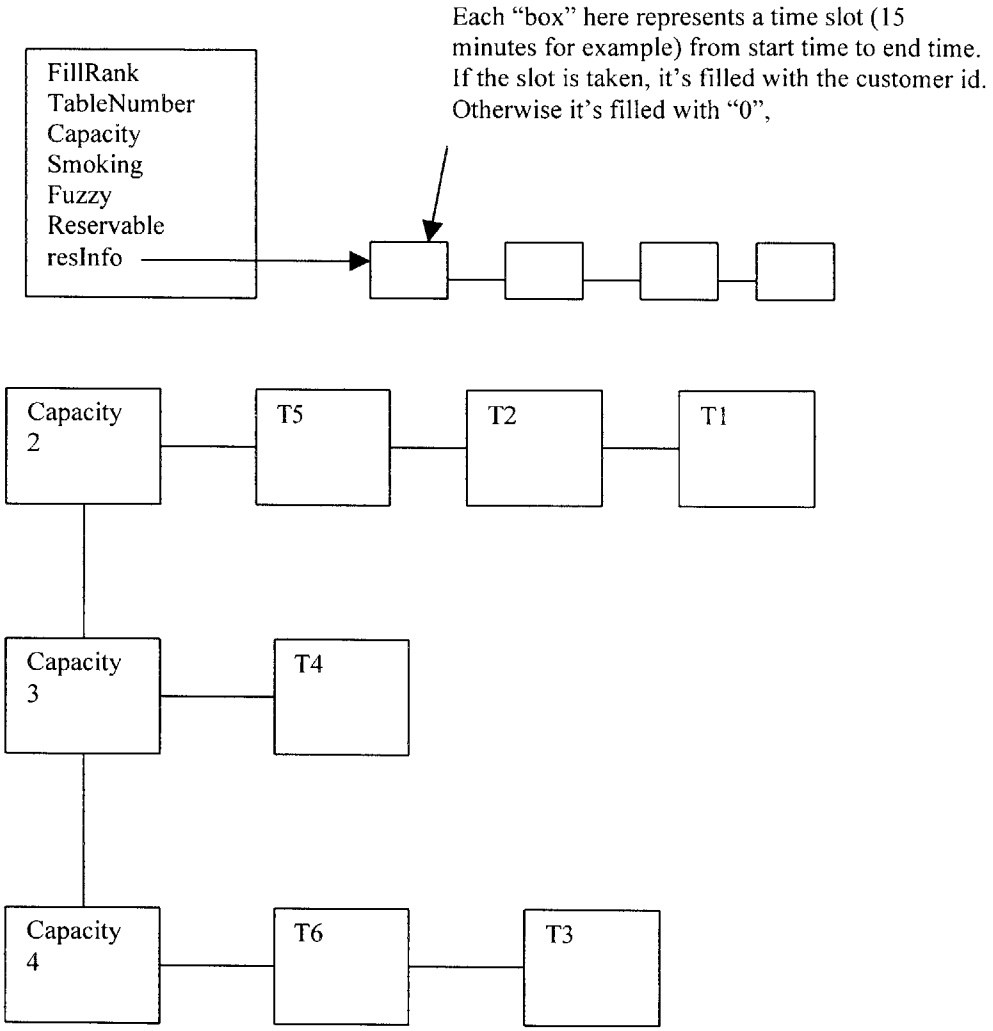

Figure 5.10 Tables Memory Datastructure

Initialization of data structures

AllTables = getAllTables from rezzDB (getAllTables gets all the tables sorted by fillRank)
AllReservations = getAllReservations from rezzDB for a particular day
for i = 1 to AllTables.length

```
    put AllTables[I] in the "capacity row" = AllTables[I].capacity

for j = 1 to AllReservations.length
   table = getTable  corresponding to AllReservations[j].tableNumber
   for k = AllReservations[j].starttime to AllReservations[j].endtime
        table.resInfo[k] = AllReservations[j].cid
```

"Reservations" Object (RO)

The reservations object, as mentioned earlier, is a static object that handles all the reservations. Not only is it important for the functions it permits, but also because it caches customer and reservation information. It stores, in memory, the customer profile information, reservation information for a particular day, and "the day".

Memory

The customer profile information is stored in memory as a table where the customer id is the key, and the customer record is the value. The customer profile information is loaded on rezzSys initialization. The "reservations" object loads all the relevant information from the database using the rezzDB interface (which is described later in this paper).

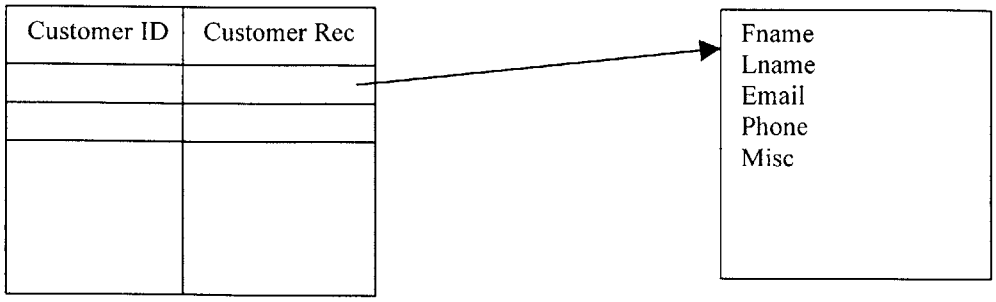

Figure 5.11 Customer Hashtable

The above memory object changes when a new customer is added, in which case a new entry is added to the table in memory, or a customer profile is changed.

The reservations object, besides containing the customer table, also contains all the reservations on a particular day. The "day" depends on what the state of the system is at; for example, in the RGUI one can jump to a particular day. That maps directly to the date stored in the reservations object. The reservations stored in memory are the reservations for this particular date. It is this data structure that governs the viewable data on the RGUI.

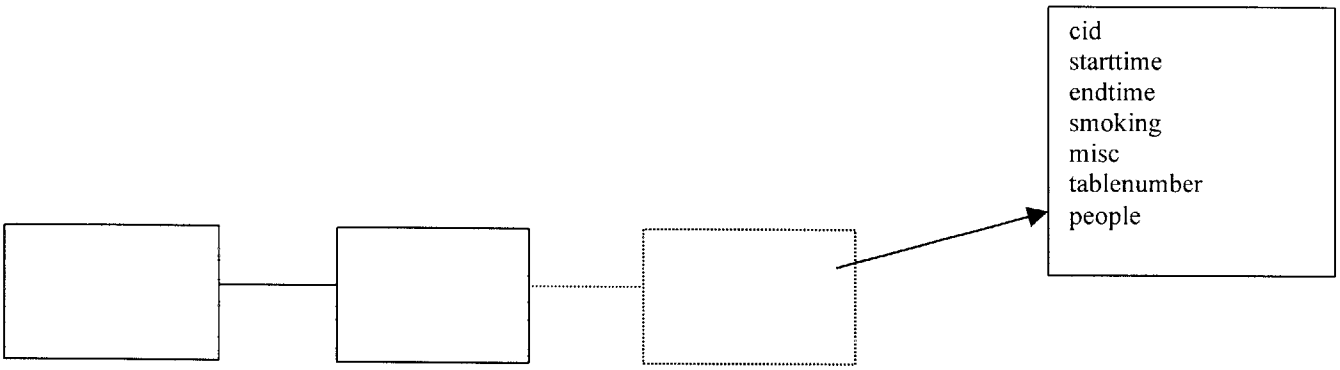

Figure 5.12 Reservations Datastructure

As mentioned earlier, the RO provides a rich set of static methods that are essential to the system. It provides a higher-level interface to make reservations, delete reservations, and change reservations. By calling a reservation method, the memory, as well as the physical data on the database, are changed. The method that will be discussed in this paper in detail is the makeReservation method.

MakeReservation method

The makeReservation method takes as input a reservationRecord. It then uses the algorithm described below.

Reservation Algorithm

Here is a simplified description of the algorithm used by the system:

Reservation Algorithm
Take in reservation record r.
get all the tables that satisfy the criteria
sort all these tables according to the fillrank (predetermined by the owner, manager, or maitre d' - described in the table restaurant_tables on the local database)
choose the first table in this set, and make the reservation (change the state and update the data)
if the reservation date is the same as the date the system is on, then make the change to memory as well.

All tables that satisfy "the criteria" means, all tables that meet the following requirements:

checkCriteria(table, r)
    table.smoking = r.smoking
    table.capacity = r.people OR (table.capacity = r.people - 1 AND table.fuzzy = true)
    table.schedule[r.starttime] to table.schedule[r.endtime] = false (table is unoccupied for the time period required for the reservation)

One of the key properties of the makeReservation method of the RO is that it only allows one thread to execute the method at any given time. This prevents issues of concurrency to occur. As described earlier, there is only one point of reservation. However, even one point of reservation can lead to undesirable concurrency issues. In the general case, many threads will be trying to make reservations at the same time; either through the Internet or the restaurant side. In both cases, the make reservation method of the RO is called.

Consider the following scenario. Suppose r1 and r2 have conflicting reservation information. i.e. r1.tableNumber = r2.tableNumber and the r1's reservation timeslot overlaps with r2's reservation timeslot.

	Thread 1	Thread 2
t1	Execute RO.makeReservation(r1)	
t2		Execute RO.makeReservation(r2)
t3	Find tables that fit criteria	
t4		find tables that fit criteria
t5	Table1= choose first table	
t6		table2 = choose first table
(note that table1 points to the same object that table2 points to)		
t7	Change state of table1	
t8		change state of table2

Figure 5.13 Race Conditions Table

Notice how Thread 2 has overridden partially or fully the reservation Thread 1 made. This is extremely undesirable. This problem is solved by using semaphors. Thread 1 locks the makeReservation method of RO, thereby preventing Thread 2 to access the method. This ensures that all necessary complete changes are made to memory before any other process attempts to make a reservation. At first, this locking may seem like a bottleneck to performance, but as discussed later, performance is hardly affected by locking the method.

Choice of Algorithm

There does not exist an algorithm that can optimally choose the correct table. This is because the system does not know all the future reservations. This makes reserving tables a difficult task.

All the algorithms to make reservations must have the following step in common:

Find all tables that fit the reservation criteria.

Call this set of tables set T. Now, the table chosen from this set T is what differentiates different algorithms. For any algorithm A_1, and adversary can come up with a set of reservations R such that R cannot be entirely satisfied using algorithm A_1 but can be satisfied using another algorithm A_2:

$\forall A_i, \exists R$ and A_j s.t. A_i cannot schedule R but A_j can.

[$R = \{r_1, r_2, r_3, .., r_k\}$ where r_n is a reservation and reservation r_{n-1} arrives before r_n]

Advantages of the FillRank Algorithm

By using the FillRank algorithm and choosing the tables in a predetermined order, the maitre d' can try to force all the tables of a particular physical area to get filled up first before another area is open. Consider a restaurant with two floors - a first floor and second floor where the kitchen is located on the first floor. It is in the restaurant's best interest to have as little seatings on the second floor and maximize the number of seatings on the first floor. This makes service easier and faster. In this case, the restaurant would rank all the tables on the first floor with high numbers and all the tables on the second floor with lower numbers. Within first and second floors, the restaurant may rank the seats next to the windows higher so customers get a better view.

rezzDB

The rezzDB is an object with static methods and no data members. That is to say, rezzDB does not have to be instantiated to use its methods. The rezzDB's primary responsibility

is to put data into the database and get data from the database when needed. The rezzDB also returns a commit (yes or no) if and only if there exist two copies of the data.

The only objects that use the rezzDB interface are the TO and the RO.

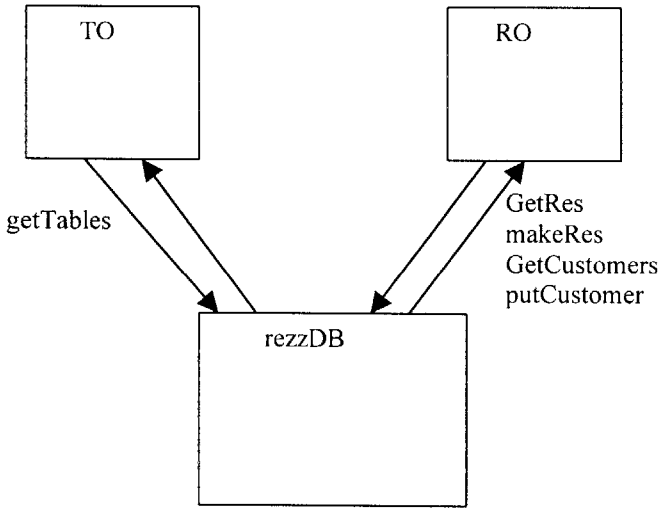

Figure 5.14 Interaction of rezzDB with TO and RO

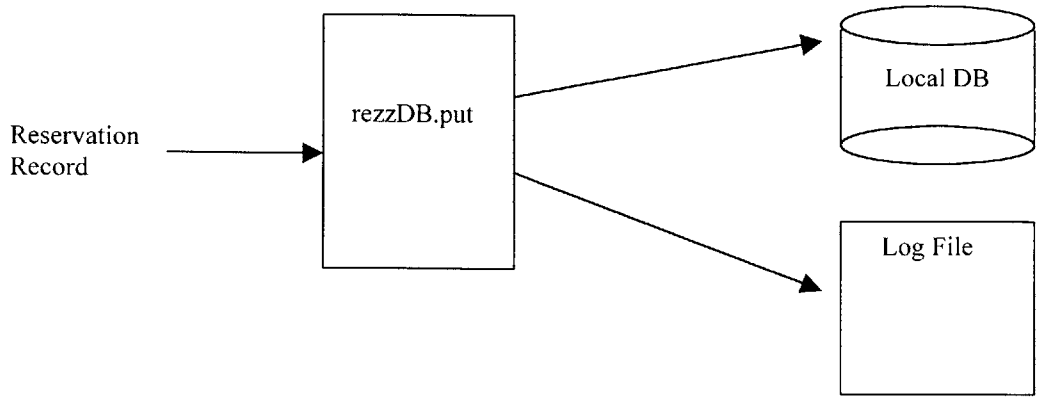

Figure 5.15 Interaction of rezzDB with Local Database

In the above diagram it is clear how the rezzDB puts the information to the local database as well as on a log file. Because the RO makeReservation method is locked and it is the only method that invokes the rezzDB "put" methods, the rezzDB "put" methods are also locked; only one process can make changes to the database at any one point in time.

The rezzDB writes to the local database for obvious reasons. However, the question arises why the rezzDB writes to the log file (please note that even when a deletion is made, this command is written to the log file). It writes to the log file as a record of what needs to be done to the central database. In fact, the put methods could write to the central database directly, but that would slow down the performance; it would have to check whether or not the system is connected to the Internet, make a connection to the central database or central server, and then make a transaction. All this would take too long and the thread waiting for the commit (yes or no) would have to wait an unnecessarily long time. Instead, by writing to the log file, another process can continuously flush the contents of the log file to the central server in the background. This improves performance and ensures that the central database is updated.

The process that flushes the log file to the central database is simply:

Log Flushing

Reads entries (commands) from the log file
Opens a connection to the central server
Pushes the commands to the central server to carry out

Chapter 6

End-To-End System – Putting Everything Together

This section describes the entire Online Reservation System from end-to-end. As mentioned earlier, there are two ways a reservation can be made: over the phone and via the Web.

6.1 Scenario 1 - The Customer makes a reservation over the telephone

When a customer makes a reservation over the telephone, a number of steps take place. The very first step that the maitre d' takes after receiving the call is either i) jumping to the day of interest or ii) performing a quick enter (QE)

6.1.1 Jumping to the Day of Interest

If the customer wants to make a reservation on a particular date D, the maitre d' can go to date D and make the reservation manually. The following events take place when jumping to a particular date to make a reservation:

i) The reservations and tables object get loaded with all the reservation and table information for that particular date (the cache gets preloaded into memory). The reason why the entire cache gets loaded with the information for that particular date is because the maitre d' is likely to perform several operations on that particular date.

ii) The RGUI is visually updated to show the appropriate information of date D.

iii) The maitre d', after jumping to the day of interest, can either manually enter a reservation or do a quick enter. In terms of performance and operations, they are virtually the same

since the system performs the same set of actions for a reservation made on the system (RGUI) date.

iv) After the maitre d' enters the relevant reservation information, the system checks its memory to see whether the reservation can be made – the system does not need to check the database for availability since there is zero probability that the memory is stale (i.e. has information that is not necessarily up-to-date). [As mentioned earlier, the method that checks for availability and makes the reservation is synchronized so that only one thread can execute that method at any given time.]

v) If the reservation can be made, the memory is updated and then the database is updated. One could argue that the database does not need to be updated at this time, and the memory can simply be flushed when the date is changed. If the system never abruptly shut off, this would be a reasonable course of action. However, if the system did abruptly shut off, then all the reservation information in memory would be lost.

vi) After the local memory and database are updated, a request is made to the central server to make an update on the central database. If the central server is down, the transaction is logged in a local file and is pushed across when a connection can be established with the central server.

vii) The reservation is visually displayed on the RGUI.

6.1.2 Quick Enter (QE)

Occasionally, a customer will want to make a reservation on an unusual date; for example, a year from the current date. In these cases, the maitre d' will make a QE reservation

since "jumping" to the date will result in an unnecessary overhead (loading the reservations and tables objects). A QE reservation involves the following steps:

i) The system checks to see whether the reservation can be made, by checking the local database. (Note: it does not check the memory since the memory does not have the relevant information.)

ii) If the reservation can be made, the reservation record is added to the local and central databases. However, there is no visual change.

6.2 Scenario 2 - The Customer makes a reservation over the Internet

When a customer makes a reservation over the web, the steps involved are similar to those described in *Scenario 1*. Since the reservation request is pushed to the restaurant system, it uses the same steps described above.

When a reservation is made over the web, the servlet pushes the reservation request to the central server. Then, the central server checks to see if the restaurant of interest is online. If it is, then it simply opens a connection to the restaurant, pushes the request, and waits for a response to display to the user. If the restaurant is offline, the servlet will not allow the customer to make a reservation.

Chapter 7

Implementation

This section discusses some implementation specifics of the Online Reservation System. In particular, it discusses the choice of the programming language (Java), synchronization, multi-threaded programming, singleton objects, and caching.

The implementation of the beta version of the Online Reservation System is in Java. The client application, the servers, and the GUI are programmed using Java. There are several reasons for choosing Java as the programming language of implementation:

i) Platform Independence

Platform independence is key in the choice of Java as the programming language. The client side can choose any platform that they are comfortable with. This is important from a business point of view; for example customer A might be more comfortable with a Macintosh, customer B might be more comfortable with Windows 95/NT platform, while customer C might want a Linux box. By using Java, the platform does not matter and thus development time is immensely reduced and the client can choose any operating system they like.

ii) Object Oriented

Java is extremely object oriented in nature. Consequently, this works very well since the ORS is a very modular design.

iii) Development Time

Java has many predefined libraries that make development quicker and easier. The

Java documentation available at http://www.java.sun.com/products/1.2/docs/ is also extremely helpful.

The Web Server used is Sun's JSWDK version 1.0.1. This web server is used because it supports Java Server Page (JSP) technology. JSP technology is an excellent choice of technology for the implementation of the web server back-end. It allows for the separation of the GUI development from the logic development. Besides separating the two tasks, it is also a good choice because of its performance. Java Server Pages, like Active Server Pages, are executed on the server side. A JSP-enabled web server compiles the .jsp pages into servlets thereby improving performance dramatically the next time the page is hit. ASP, on the other hand, is interpreted each and every time. Also, JSP has an advantage over standard CGI scripts because a process is not necessarily forked on each request. If a servlet is resident in memory, then the servlet is reused, saving initialization time and memory.

The central database used is Oracle 8.0.5. Ideally, Oracle 8.1.5 should be used since it allows the distribution of the database, improving performance. Database access is obtained using JDBC – Java database connectivity. This is a good choice of implementation because if the "type" of database were to change, only another driver would need to be loaded and everything else would remain the same. On the client side, the database used is Microsoft Access.

As mentioned in this paper, concurrency issues are dealt with by locking the appropriate methods and data objects such that only one thread is allowed to use it at a particular time. Java facilitates synchronization; an object and/or method can be synchronized by using the Java

keyword *synchronized.* By utilizing this keyword, the Java Virtual Machine locks the object/method, thereby ensuring that only one process uses it.

Java, because of its object oriented nature, allows the programmer to concentrate on the design aspects and not worry about implementation details. By taking advantage of inheritance, many software components can be reused easily and effectively. Multi-threaded objects, for example, can be created using the *Thread* class in the java.lang package. Another case where inheritance is used in the development of the system is in the creation of the rezzDate object.

```
import java.lang.*;

Class example_process extends Thread
{

}
 // extends is a Java keyword

class rezzDate extends rezzCalendar
{

}
```

In some cases, singleton objects are used in the system. The reservations object, for example, is a singleton object. A singleton object is an object like any other object except for the property that one and only one object can ever exist in the same environment; only one object can be instantiated at any given time; only one object can exist. Singleton objects are powerful in this respect and can serve as shared objects.

A singleton object is implemented by the use of a private constructor. In Java, all objects must have at least one constructor. A singleton class, unlike most classes, has a private

constructor. This means that only methods of the class can execute the method; in turn, only a method of the class can create a singleton object. Besides having a private constructor, a singleton class contains a private pointer to the singleton object within itself. Another object can retrieve the object by using a static method of the singleton class. This static method, getInstance for example, first checks to see whether the private member is equal to null. If the private member (pointer of the singleton type) is equal to null, then a new object is created and returned to the object calling the method. Otherwise, the object is simply returned.

```
class reservations
{

        private reservations r;
        // other declared members

        private reservations()
        {
                // private constructor method
        }

        public static reservations getInstance()
        {
                if (r == null)
                        r = new reservations();
                return r;
        }
}
```

Since performance is an important consideration of the system, the implementation of the cache is important. The system's cache is implemented by using Vector and Hashtable objects that are part of the java.lang package in the Java Development Kit (JDK). The Hashtable object provides a O(1) method of looking up objects. By using the put and get methods of the Hashtable object, different objects can be stored in memory. The Vector object is also a very

useful object to store different objects with. A Vector object can store different objects and can be of any variable length. This is useful since using a standard array requires knowing the size of the array beforehand. For a Vector, the size dynamically changes and does not have to be specified at creation.

Chapter 8

Testing

8.1 Fictional Restaurant

In order to test the Online Reservation System, a fictional restaurant was created. This hypothetical restaurant, based on a real restaurant, has two floors with a kitchen on the ground floor. It consists entirely of two, three, and four person tables. The restaurant is open from 11am – 10pm for lunch and dinner and has 1.5 hour reservation blocks. Also, a reservation can be made every fifteen-minute block.

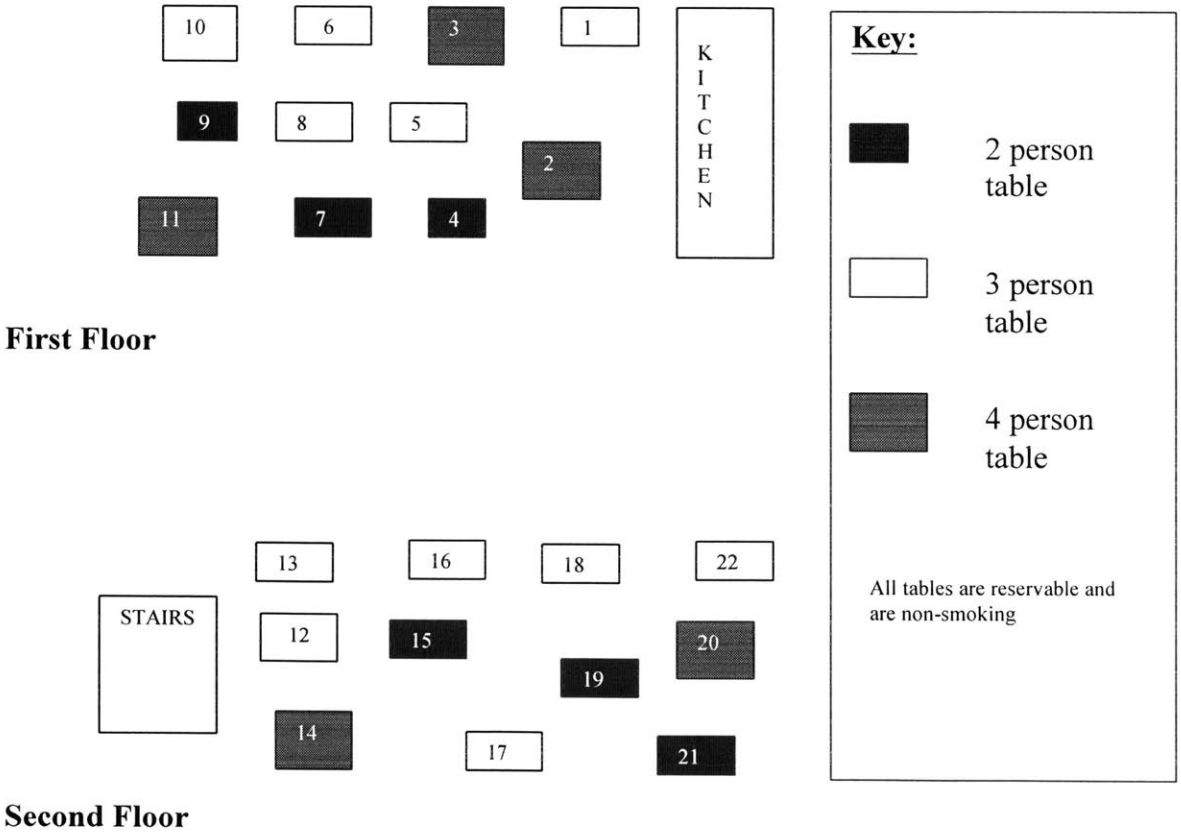

Figure 8.1 Fictional Restaurant Floor Layout

The above floor plans are not drawn to scale and the numbers inside the rectangles denote the ranking number and physical number of the particular table. The ranking order, as mentioned earlier, tells the system in which order it should try to reserve the tables. In this case, the ranking order is based on the distance of the table from the kitchen.

To test the system, software was written to emulate customers using the web to make reservations. Each software agent ("customer") randomly chose a fifteen-minute time block between 11am and 8:30pm and a random number between 1 and 5 inclusive. The number and frequency of customers using the system followed a poisson distribution. A set of reservations was made via the Internet as well as manually for the same day[16]. The actual state of the system was compared to the expected state after each reservation attempt.

8.2 Results

The following are the reservations that were made in order and the results (W means web request, and M stands for manual entry):

Time	Number of People	Type of Reservation	Result
11:15pm	4	W	OK
1:30pm	3	M	OK
1:45pm	5	W	OK
8:00pm	2	W	OK
5:45pm	3	M	OK
1:45pm	3	W	OK
2:15pm	2	M	OK
6:00pm	4	W	OK
4:30pm	3	W	OK
3:30pm	2	W	OK
5:30pm	1	W	OK
6:15pm	2	W	OK

[16] All the reservations were made for the same day so more boundary conditions could be encountered.

4:15pm	2	M	OK
2:45pm	3	W	OK
3:15pm	4	W	OK
3:30pn	4	W	OK
4:45pm	2	M	OK
Local Restaurant disconnected			
4:00pm	3	W	OK
5:15pm	1	W	OK
7:15pm	4	W	OK
12:15pm	2	M	OK
7:30pm	3	W	OK

Figure 8.2 Test Result Table

Aggregate Results
Reservation requests: 185 (only 22 are shown above)
Successful requests: 185
Failed requests: 0
Average time taken for a web request: 3 seconds (not including the very first one)
Average time taken for a manual entry: 1.5 seconds

As can be observed from above, the system handled 185 requests successfully. The average time taken is rounded to the nearest tenth second and, of course, is variable. In some cases, the time was extremely short, in others it was slightly longer. The average time taken for a Web request is also dependent on the location of the different software components. If all the software components, for example, were scattered across the world, the average request time would be higher. One can also assume that the average request time would decrease as the number of processors and/or the speed of the processors increase.

Chapter 9

Performance

This section discusses caching and system performance of the ORS.

9.1 Caching

The system is very efficient because it caches data in memory. Upon starting the local system, RezzSys loads all the data of the present day into memory. If the maitre d' decides to visually jump to another day, the data of the new day is loaded in memory. This enhances performance and prevents unnecessary database transactions that can be costly. Fortunately, even if the maitre d' performs a quick insert – that is, attempts to insert a reservation on day B when he is on day A – it does not update the memory. So, the maitre d' can continue to add reservations to day A without having to flush the memory.

9.2 System Performance

The performance of the overall system can be assessed by making the following assumptions:

- Let the number of restaurants be N.
- Distribution of central database – let P equal the number of machines/processors assigned to the central database.
- The probability distribution function (pdf) for reservation inter-arrival times is negative exponential with mean $1/\lambda$ and the pdf for service times (duration of telephone conversations) is negative exponential with mean $1/\mu$. Incoming customer reservation traffic

is equally distributed across all restaurants; arrival rate is equal to $\lambda = \lambda_T/N$ (total arrival rate divided by the total number of restaurants)

- The pdf for service times (the time it takes to service a reservation request) is negative exponential with mean $1/\mu$.
- The central server is on a fast enough machine to allow as many concurrent threads as needed.

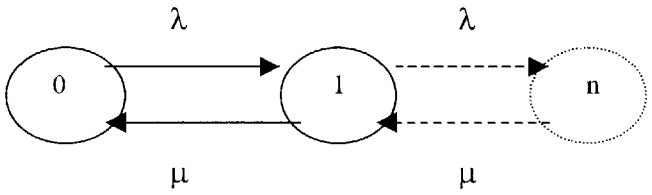

Figure 9.1 State Diagram

With the above assumptions, the ORS can be modeled as a M/M/1 type queuing system with a first come first serve (FCFS) service and an infinite system capacity. In a M/M/1 type queuing system, the expected wait time for an entry is equal to $W = 1/(\mu - \lambda) = 1/(\mu - \lambda_T/N)$[17]. The average service time, $1/\mu$, can be estimated by using the values for P and the speed of the client processor (S). $1/\mu$ is directly proportional to P and S; the greater the value of P and/or S, the less the value of μ.

[17] Formula obtained from *Urban Operations Research*

From Chapter 8, we know that μ is equal to 0.444 and λ is equal to .0017 assuming there are an average of 50 reservation requests per day (8 hour booking block)[18]. With these values, the total time for a request to be executed is approximately equal to 2.26 seconds.

[18] μ = 1/average mean time = 1/2.25 (2.25 is the average of 1.5 and 3)

Chapter 10

Extensions

The ORS can be extended in many different ways. Firstly, the reservation algorithm can be modified so that it can handle more complicated situations. Currently, tables are assigned only to requests that match the table capacity exactly (except for the tables with the fuzzy option equal to true). This prevents large reservations from being made over the Internet. Suppose there is a party of 10 that would like to make a reservation at a restaurant. The party would be forced to make two or three separate reservations. A possible extension could be to have the system make more than one reservation at adjacent tables if the number of people is larger than the capacity of the largest table. This would allow tables to be joined together to accommodate large groups.

Another possible extension is to put customers on a waiting list. This way, if a table opens up, a customer is notified either through email or phone. Not only is this a useful feature when the restaurant is fully booked, but it can also be important in the case where the restaurant is not connected to the Internet. Currently, a customer can only make a reservation when the restaurant is connected to the Internet. By incorporating the concept of a waiting list in the product, a customer can be put on the waiting list if the restaurant is not connected to the Internet.

Lastly, for this product to become a commercial product, other important, but not so technical, features are required. A seating chart is one such example. Instead of the seat information displayed on the right hand side, a graphical representation of the tables should be available. Also, the maitre d' should also have the option of swapping reservations from one

table to another by simply clicking and dragging the mouse. This can prove to be very useful and time saving to the maitre d' of the restaurant.

Chapter 11

Conclusion

The Online Reservation System described in this thesis document is an affordable real-time reservation system for restaurants to extend their business to the Internet. It solves many of the problems encountered by the customer and maitre d' during the conventional reservation process, while accommodating the old reservation process. It is an efficient system that successfully handles issues of connectivity, concurrency, and hard disk crashes.

The ORS addresses technical challenges and is also designed and implemented keeping the business aspect in mind. The GUI is designed to assist the maitre d' in his/her daily work; customer profiling is available so that the restaurant understands the customer's needs before he/she arrives. Also, because the ORS is developed using off the shelf software and hardware, it is easy to expand the system, making it an affordable solution for any small restaurant.

The ORS, because of its modularity, can feasibly be extended into a general online reservation system for small client-driven businesses. With the design and implementation of a proper grammar and language structure, constraints and algorithms can be created to describe different businesses that lie on top of the described system architecture: restaurants, hotels, airlines, boutiques, or any other client-driven business. The possibilities are limitless.

References

- Anon. *Electronic Seat Reservation System,* Process Control and Automation, April 1959
- Anon. *How to Make a ReserVec Reservation*, Between Ourselves (TCAL internal monthly newsletter), March 1961
- Anon. *What is Inventory Updating?*, Between Ourselves (TCAL internal monthly newsletter), March 1962
- C.J. Bashe et al. *IBM's Early Computers,* MIT Press, Cambridge, Mass., 1986.
- D.H. Collins. *Wings Across Time*, The Story of Air Canada, Griffin House, 1978
- Larson, Richard C. and Odoni, Amadeo R. *Urban Operations Research*, Prentice-Hall, NJ, 1981
- L.E. Richarson. *The Electronic Reservations System for Trans-Canada Air Lines*, Proc. Computing and Data Processing Society of Canada, Univ. of Toronto Press, Toronto, 1960
- R.F. Burkhardt. *The Sabre System: A Presentation*, October 1, 1964
- R.F. Meyer. *American Airlines Sabre (A),* Boston: Harvard Business School, Case No. EA-C758, 1967
- W.R. Plugge and M.N. Perry. *American Airlines 'Sabre' Electronic Reservations System*, AFIPS Conf. Proc., Western Joint Computer Conf., 1961
- Cisco DistributedDirector Information from http://www.cisco.com
- General Internet usage figures from CyberAtlas, at http://www.internet.com, Inc.
- SABRE Information from http://www.sabre.com
- Reservation systems case study from http://www.scit.wlv.ac.uk/~cm1995/cbr/cases/case06/FOUR.HTM

Bibliography

1. Anderson, Thomas E., *High Speed Switch Scheduling for Local Area Networks*, ACM Transactions on Computer Systems, Computer Science Division, University of California, Berkeley, 1992

2. Birrell, Andrew. *An Introduction to Programming with Threads*, Digital Equipment Corp, Palo Alto, January 1989

3. Birrell, Andrew D., Nelson, Bruce J. *Implementing Remote Procedure Calls*, ACM Transactions on Computer Systems, V.2 N.1, Xerox Palo Alto Research Center, Palo Alto, February 1984, pg. 39-59 February 1984, pg. 39-59.

4. Gifford, David; Spector, Alfred. *The Cirrus Banking Network*, Communications of the ACM, August 1985 v. 28 n. 8

5. Gray, Jim. *High Availability Computer Systems*, Digital Equipment Corp; Siewiorek, Daniel P. Carnegie Mellon University. Computer v. 24 n. 9, September 1991, pp. 39-48

6. Lampson, Butler. *Hints for a Computer System Design*, ACM Transactions on Computer Systems, Xerox Palo Alto Research Center, Palo Alto, 1983, pg. 33-48.

7. Rosenblum, Mendel; Ousterhout, John K. *The Design and Implementation of a Log-Structured File System*, ACM Transactions on Computer Systems, V. 10.
N. 1. February 1992. pp. 26-52

8. Saltzer, J.H. *Topics in the Engineering of Computer Systems*, Chapter 2: Objectives of an information system, February 1985.