

# **A Self-healing Dynamically Distributed System for Hotel Management using JavaSpaces**

**By**  
**Tzer Hung Low**

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

July 2001

Copyright 2001 Tzer Hung Low. All rights reserved.

The author hereby grants to M.I.T. permission to reproduce and  
distribute publicly paper and electronic copies of this thesis  
and to grant others the right to do so.

Author \_\_\_\_\_  
Department of Electrical Engineering and Computer Science  
July 19, 2001

Certified by \_\_\_\_\_  
Dr. Amar Gupta  
Thesis Supervisor

Accepted by \_\_\_\_\_  
Arthur C. Smith  
Chairman, Department Committee on Graduate Thesis

A Self-healing Dynamically Distributed System for Hotel Management using JavaSpaces

By

Tzer Hung Low

Submitted to the Department of Electrical Engineering and Computer Science

July 19, 2001

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

## Abstract

We present the design and prototype implementation of a low-cost Hotel Management System. In this system, computing devices are distributed throughout the hotel and connected via Ethernet. A number of central processes dynamically create or destroy processes on each computing device as required. Processes communicate either via JavaSpace or directly through the Jini architecture. Redundancy ensures that the system is robust, while Jini Leases ensure that the system is self-healing. Several services needed for Hotel Management are proposed. One major innovation is the use of biometric authentication tools for identification and access controls. The task of matching a given fingerprint to a lists of possible templates is divided by templates in JavaSpace. Another major innovation is the use of mobile Software Agents for scheduling multiple, bundled events. Our prototype has served well as a “proof-of-concept”, and with further work, we are optimistic that this can become a practical system for Hotel Management.

Thesis Supervisor: Dr. Amar Gupta

Title: Co-Director, PROFIT Program, MIT Sloan School of Management

## **Acknowledgments**

First, I would like to thank Dr. Gupta for providing me the opportunity to work on a challenging and rewarding project. I would also like to thank Annie Lo, Steve Song and Yu Shuo Low for providing me with direction, motivation and support for this thesis. Finally, I would like to thank my family for their support in my endeavors.

# Table of Contents

1	Introduction .....	11
1.1	Current trends in hotel industry.....	12
1.2	Summarized Vision of our ideal hotel.....	14
1.3	About our project .....	15
1.3.1	Case Study: The SunSprings Project.....	15
1.4	Project Scope.....	17
1.5	Paper Roadmap .....	18
2	Detailed Vision of ideal hotel.....	19
2.1	Detailed Vision of ideal hotel for hotel guest .....	19
2.1.1	Renting a room.....	19
2.1.2	Checking in .....	20
2.1.3	Entering the room.....	21
2.1.4	Enjoying built-in facilities in guest room.....	21
2.1.5	Enjoying hotel facilities .....	23
2.1.6	Checking out of the hotel .....	25
2.2	Detailed Vision of an ideal hotel for a hotel manager.....	25
2.2.1	Obtaining the required components .....	25
2.2.2	Setting up the system.....	27
2.2.3	Maintaining the system .....	28
2.2.4	Using the system .....	30
3	Related Efforts.....	32

3.1	Commercially available hotel systems.....	32
3.1.1	Room Reservations System.....	33
3.1.2	Access Control System.....	38
3.1.3	Summary of commercially available solutions.....	38
3.2	Related Efforts in Biometrics Authentication and Identification.....	39
3.2.1	Reasons for choosing Biometrics Authentication and Identification.....	39
3.2.2	A survey of current biometrics literature.....	41
3.3	Related Efforts in Yield Management.....	43
3.4	Related Efforts in Networked Systems.....	44
3.4.1	Traditional Networked Systems.....	44
3.4.2	Remote Object Systems.....	46
3.4.3	Service Discovery Protocols.....	47
3.4.4	The Jini Architecture.....	48
3.5	Related Efforts in Distributed Processing.....	50
3.5.1	Tuple space based coordination languages.....	50
3.5.2	JavaSpaces.....	51
3.5.3	Reasons for choosing JavaSpaces.....	53
3.5.4	Mobile Agents.....	56
3.6	Background Information on Tiny InterNet Interface (TINI).....	58
4	Design Overview.....	60
4.1	Considerations and Criteria.....	60
4.2	Design of the Hotel Management Software.....	61
4.2.1	Class Hierarchy.....	61

4.2.2	Drones .....	64
4.2.3	Summary of specific Drones .....	64
4.2.4	Agents.....	65
4.2.5	Summary of Specific Agent Drones and their Agents .....	66
4.3	Layout of the Hotel Management Hardware.....	67
4.3.1	TINI at every access point.....	67
4.3.2	Desktop Computers .....	67
4.3.3	Floor Servers .....	68
4.3.4	Central Servers .....	69
4.3.5	Rationale for our design .....	70
4.3.6	Discussion on other possible layouts .....	72
5	Details of software implementation .....	74
5.1	Drones: Processes that communicate via JavaSpace.....	74
5.2	Details on some specific Drones .....	77
5.2.1	Queen .....	78
5.2.2	Hatchery .....	79
5.2.3	Registration Drone .....	81
5.2.4	User Drone .....	82
5.2.5	Fingerprint Drone.....	83
5.2.6	King.....	84
5.2.7	Auth Drones .....	91
5.2.8	Application Drones .....	92
5.3	Agent Drones: Processes that can communicate directly.....	93

5.4	Details on some specific Agent Drones .....	96
5.4.1	Database Agent Drone .....	96
5.4.2	Administrative Agent Drone .....	97
5.4.3	Messaging Agent Drone.....	99
6	Specific Scenarios .....	101
6.1	Check-in .....	102
6.2	Access-Control.....	104
6.3	Logging-in.....	107
6.4	Scheduling Activities .....	110
7	Evaluation.....	113
7.1	Fulfillment of design goals.....	113
7.1.1	Provide comprehensive services. ....	113
7.1.2	Provide superior services. ....	113
7.1.3	Computational Efficiency. ....	114
7.1.4	Easy to install .....	114
7.1.5	Easy to maintain .....	115
7.1.6	Easy to upgrade and backup.....	115
7.1.7	Affordable. ....	116
7.1.8	Customizable.....	116
7.1.9	Scalable. ....	117
7.1.10	Web-enabled.....	117
8	Conclusion.....	119
8.1	Future Work .....	119

8.1.1	Other Distributed Object Systems.....	119
8.1.2	Better way to scale JavaSpaces .....	120
8.1.3	Integration with other systems .....	121
8.2	Summary .....	121
9	References .....	123
9.1	General: .....	123
9.2	Commercial Hotel Management Systems:.....	123
9.3	Authentication and Identification.....	124
9.4	Yield Management .....	125
9.5	Networked Systems.....	125
9.6	Distributed Systems.....	127
10	Appendices.....	<b>Error! Bookmark not defined.</b>
10.1	Package Drone.....	<b>Error! Bookmark not defined.</b>
10.1.1	Class drone.Drone .....	<b>Error! Bookmark not defined.</b>
10.1.2	Class drone.anywhere.Queen .....	<b>Error! Bookmark not defined.</b>
10.1.3	Class drone.anywhere.King.....	<b>Error! Bookmark not defined.</b>
10.1.4	Class drone.anywhere.Hatchery.....	<b>Error! Bookmark not defined.</b>
10.1.5	Class drone.anywhere.UserDrone .....	<b>Error! Bookmark not defined.</b>
10.1.6	Class drone.anywhere.AuthDrone.....	<b>Error! Bookmark not defined.</b>
10.1.7	Class drone.anywhere.ScheduleDrone .....	<b>Error! Bookmark not defined.</b>
10.1.8	Class drone.anywhere.BillingDrone .....	<b>Error! Bookmark not defined.</b>
10.1.9	Class drone.anywhere.ReaderDrone .....	<b>Error! Bookmark not defined.</b>
10.1.10	Class drone.anywhere.ChatDrone .....	<b>Error! Bookmark not defined.</b>

- 10.1.11 Class drone.anywhere.agent.AgentDrone ..... **Error! Bookmark not defined.**
- 10.1.12 Class drone.anywhere.agent.AdminAgent ..... **Error! Bookmark not defined.**
- 10.1.13 Class drone.anywhere.agent.AdminAgentDrone . **Error! Bookmark not defined.**
- 10.1.14 Class drone.anywhere.agent.MessagingAgent..... **Error! Bookmark not defined.**
- 10.1.15 Class drone.anywhere.agent.MessagingAgentDrone..**Error! Bookmark not defined.**
- 10.1.16 Class drone.frontdesk.RegistrationDrone ..... **Error! Bookmark not defined.**
- 10.1.17 Class drone.tini.FingerPrintDrone..... **Error! Bookmark not defined.**
- 10.1.18 Class drone.droneEntry.DroneEntry ..... **Error! Bookmark not defined.**
- 10.1.19 Class drone.droneEntry.RefreshEntry... **Error! Bookmark not defined.**
- 10.1.20 Class drone.droneEntry.DeathEntry..... **Error! Bookmark not defined.**
- 10.1.21 Class drone.droneEntry.QueenEntry..... **Error! Bookmark not defined.**
- 10.1.22 Class drone.droneEntry.HatcheryEntry. **Error! Bookmark not defined.**
- 10.1.23 Class drone.droneEntry.RegistrationEntry..... **Error! Bookmark not defined.**
- 10.1.24 Class drone.droneEntry.MessageEntry . **Error! Bookmark not defined.**
- 10.1.25 Class drone.droneEntry.AuthRequestEntry ..... **Error! Bookmark not defined.**

10.1.26	Class drone.droneEntry.AuthResultEntry .....	<b>Error! Bookmark not defined.</b>
10.1.27	Class drone.droneEntry.QueenCreateRequestEntry ...	<b>Error! Bookmark not defined.</b>
10.2	Package gui .....	<b>Error! Bookmark not defined.</b>
10.2.1	Class gui.RegistrationGUI .....	<b>Error! Bookmark not defined.</b>
10.2.2	Class gui.ChatGUI .....	<b>Error! Bookmark not defined.</b>
10.2.3	Class gui.MessagingGUI.....	<b>Error! Bookmark not defined.</b>
10.2.4	Class gui.MessagingManagerGUI .....	<b>Error! Bookmark not defined.</b>
10.2.5	Class gui.LogOnGUI.....	<b>Error! Bookmark not defined.</b>
10.2.6	Class gui.ScheduleGUI .....	<b>Error! Bookmark not defined.</b>
10.2.7	Class gui.ScheduleChooseTargetGUI.....	<b>Error! Bookmark not defined.</b>
10.2.8	Class gui.BillingGUI.....	<b>Error! Bookmark not defined.</b>
10.2.9	Class gui.UserGUI .....	<b>Error! Bookmark not defined.</b>
10.3	Package common.....	<b>Error! Bookmark not defined.</b>
10.3.1	Class common.Utilities .....	<b>Error! Bookmark not defined.</b>
10.3.2	Class common.EventID.....	<b>Error! Bookmark not defined.</b>
10.3.3	Class common.FingerPrintConst.....	<b>Error! Bookmark not defined.</b>
10.3.4	Class common.AgentInterface .....	<b>Error! Bookmark not defined.</b>
10.3.5	Class common.AdminAgentInterface .....	<b>Error! Bookmark not defined.</b>
10.3.6	Class common.MessagingAgentInterface.	<b>Error! Bookmark not defined.</b>
10.4	Package guestApplications.....	<b>Error! Bookmark not defined.</b>
10.4.1	Class guestApplications.Applications.....	<b>Error! Bookmark not defined.</b>

10.5 Corba..... **Error! Bookmark not defined.**  
10.5.1 RoomBooking..... **Error! Bookmark not defined.**

# 1 Introduction

This thesis attempts to build the framework for an innovative Hotel Management System. The proposed Hotel Management System is a cross-platformed, dynamically distributed system that is modular, self-healing, plug-and-play enabled, and that allows for adaptive parallelism. In addition, this thesis attempts to use biometrics identification methods for access control, as opposed to the traditional method of using physical tokens.

The proposed Hotel Management System is not for all hotels. There are many ways to increase revenue or decrease costs in existing hotels. A solution that suffices a five-star hotel in Indonesia where the annual per capita income is only \$2,830 [1.1] may not accommodate a three-star hotel chain in the United States where the annual per capita income is \$31,500 [1.1].

The design we propose is technologically driven. Therefore, it is only appropriate where the marginal cost of providing a service via technology is less than the marginal cost of providing an equivalent service via non-technological means. For example, providing a method of taking room-service food orders via computer may not be viable in Indonesia, where the cost of leasing a computer is more than the cost of paying a butler to service every room. On the other end of the spectrum, international five-star hotel chains with their own Hotel Management Systems with features such as centralized reservations will probably show little interest in our thesis. This thesis neither details a cost-effective way to integrate with their legacy systems, nor does it propose a solution that takes advantage of the same economies of scale as they do.

Thus, the focus is to provide a solution for the middle range of hotels, such as a four-star hotel chain, for whom currently available technology is affordable and can also provide valuable services to bring them on par with the five-star hotels. In line with this focus, the proposed implementation uses inexpensive off-the-shelf components, such as inexpensive desktops and servers, to keep hardware costs low. The system is a plug-and-play solution with a modular design to keep installation, maintenance, and upgrading costs low. The solution is also designed for distributed computing with each computer working in parallel so that it can compete with the computational power of the large five-star hotels with more expensive computer equipment. The distributed computing platform is both dynamic and self-healing so that an isolated failure will not bring the demise of the entire system, thus providing the robustness demanded by hotels. The solution is cross-platformed, so that it can use any operating system (preferably the one that offers the most value). Finally, the solution is web-enabled and scalable to allow for a hotel to grow into a hotel chain.

## 1.1 Current trends in hotel industry

With the evolution of the Internet, hotel customers can gain in-depth information about hotel prices and quality before they step into a hotel. Therefore, current international hotel chains, such as Hyatt, have experienced erosion in their advantages of international reservation and quality assurance. [1.2] Hotel customers, now well informed about price

and quality through information sharing on the Internet, have more choices of lodging. They demand more than just a brand name.

Therefore, the current trend in the hotel industry is to gain market share not by competing on costs, but by services. In some cases, computers can provide service personalization. For example, computers can help keep track of the guest's needs and complaints, provide the guest with administrative services, and help to plan the hotel employee workflow. While there are other methods to obtain service personalization, computers can potentially allow hotels with a smaller budget to compete with hotels that spend considerably more on service personalization.

Following recent trends, business travelers now demand larger hotel suites than smaller hotel rooms. With the faster-paced working environment and the demand for connectivity, business travelers are spending more time in their room, and using the hotel room as an office. Aiding this revolution is the array of devices, such as laptops to Personal Digital Assistants (PDA) to cellular phones, which allow the business traveler connectivity and efficiency.

Therefore, demand is increasing for amenities such as room service, secretarial service, laundry service, transportation service, Internet connectivity, video conferencing and activities management. A competitive hotel will provide all these services.

## 1.2 Summarized Vision of our ideal hotel

Imagine that you are a business traveler that arrives at Perth, Western Australia late at night. You proceed to the hotel at which you frequently stay. Instead of waiting in line to be served by the understaffed night shift, you go directly to one of the numerous computer kiosks in the hotel lobby. A fingerprint reader recognizes your identity, and brings up your reservation if you have one, or offers to sell you a room (dynamically priced) if you do not. Since your credit card information has already been previously stored when you booked the room through the Internet, all that remains is for your to verify the saved preferences of a non-smoking room with a single king-sized bed facing the Swan River. After you have agreed to pay all charges incurred in another fingerprint entry, a computer printout pops out. You head directly to your room.

Upon reaching your room, there are no physical key cards or tokens to lose, or personal identifying numbers (PIN) to forget. A fingerprint is all you need to access your room. Upon entry, you find that the fridge is stocked with your favorite beer and other groceries that you have bought through NetGrocer.com, which is affiliated with the hotel. Your wake-up call has already been set when Microsoft Hailstorm informed the Hotel Website of your busy schedule tomorrow. In fact, your “to-do” list is on the sidebar of the computer/television screen in front of your bed. All that is left for you to do is to check your email using the wireless keyboard by your bed, to ensure that there have been no unexpected changes in plan.

## 1.3 About our project

For this project, we are working in close conjunction with SunSprings Hotel and Spa, a new hotel in Perth, Australia. In this alliance, we are able to focus on the needs of an actual commercial entity, and better understand the cost structure and profit margins of a typical four-star hotel. In doing so, we hope that our project will be able to help other similar hotels around the world upgrade their services for a higher profit margin.

### 1.3.1 Case Study: The SunSprings Project

Much of the features in our initial prototype cater to the needs of SunSprings, which fortunately shares many similar features as our target four-star hotel. For example, SunSprings housekeeping policies are reflective of a typical hotel of the same size.

Depending on the season, there are three full time maids, and anywhere from ten to fifteen part-time maids. One of their duties is a “full” clean of an apartment room when the guests checks out, which takes an average of 30 minutes. On the other hand, a “partial” clean for a guest that stays overnight takes an average of 20 minutes. Our system improves the logistics and quality of the hotel housekeeping by (1) restricting access control of maids only to rooms on their scheduled assignments, (2) dynamically scheduling every maid when he or she shows up for work based on currently available information of “do-not-disturb” messages and rooms that require “full” cleaning or “partial” cleaning, (3) logging information about each maid so that statistics such as room

access and cleaning speed can be reviewed, and (4) allowing the guest to leave specific instructions for the cleaning maid via a computer interface.

However, there are some specific differences that must be pointed out.

- **Dual Usage Design:** SunSprings' designs of the two-bedroom apartment are different from those of a typical hotel. A hotel guest can choose the option of renting the entire two-bedroom apartment, or each of the individual rooms. Therefore, access control requirements become more complex as two different customers can rent the 2 rooms in the apartment. In particular, both customers need to have access to the main door, but each customer needs only to have access to their specific room door.
- **Captive Business Needs:** SunSprings offers full captive business facilities. This is a low cost solution for companies that require the presence of a "virtual office" in Australia. Our solution needs to provide a convenient method for scheduling access to business facilities, and provide additional amenities such as video conferencing.
- **Health Care Center:** SunSprings allocates the full-fledged concept of medical healing and recuperation in the premises. This "hotel health concept" allow guests to create a health care model focusing on prevention and early intervention. In particular, there are many facilities such as

Jacuzzi, Pool and Gymnasium that hotel guests have access to, but may find inconvenient to have a physical key. There are also other non-complimentary facilities such as acupuncture therapy and massage parlors where hotel guests should be able to enjoy without carrying around a wallet.

## 1.4 Project Scope

This project covers the following aspects of creating a Hotel Management System:

- Visualizing where technology can be an advantage.
- Designing the layout of the hardware components, and the architecture of the software components.
- Implementing a software prototype that can be distributed across many computers.

The following is not covered in this paper, since it was not needed for our “proof-of-concept” system.

- Controlling an electric strike lock from a TINI Chipset.
- Interfacing the system with payment systems of major credit cards.
- Connecting the system to existing access control systems or telephone systems.
- Integrating the proposed solution with legacy hotel management systems

## 1.5 Paper Roadmap

Section 2 provides the detail vision of the ideal hotel, first in the perspective of a hotel guest who uses the facilities of the system, and then in the perspective of the hotel manager who, in addition to using the system, also has to purchase, set up, maintain and upgrade the system. Section 3 discusses previous work in commercially available hotel systems, biometrics authentication, yield management, networked systems and distributed systems. It also provides background information on the TINI chipset. Section 4 provides the overview of our system design, both in terms of hardware and software. Section 5 provides greater insight into the software design of our Hotel Management System. Finally, section 6 illustrates the proposed Hotel Management System in action through some scenarios, while section 7 evaluates the performance of the implementation, and section 8 provides the conclusion.

## 2 Detailed Vision of ideal hotel

### 2.1 Detailed Vision of ideal hotel for hotel guest

Every hotel operates in a different environment with different wants and needs. This section serves as a catalog of features, and with software modularity, hotels can customize their own Hotel Management System by picking and choosing the features that best meet their requirements.

#### **2.1.1 Renting a room**

A potential customer first contacts the hotel room reservation system of a hotel chain through a telephone call, a travel agency network, or the Internet. Whatever the means of his communication, our model customer is identified as a previous customer when he presents his credentials. This brings up his user agent, which is able to guide him for the rest of the reservations. The user provides the dates and preferred location of his expected stay, and chooses to let his user agent customize the rest of the preferences: non-smoking room, one king-sized bed, with adjoining kitchen, preferably with a view. The agent also knows that the user prefers SunSprings Hotel and Spa in the heart of Perth, rather than another hotel in the chain a little further from the city. The agent is sent via the Internet to each hotel that is of possible interest, and then split further to inspect every room that is available via the interaction with room agents. A list of possible choices is returned to the user. Finally, by the user's selection of the ideal hotel room, or by the user's explicit

complain in that selection, the user agent learns from this experience to better serve the hotel guest in future visits.

### **2.1.2 Checking in**

The hotel guest finally arrives at the hotel. Being a traveling salesman from a near-by town, he travels light, but expects many amenities to accommodate his needs. He skips the queue at the front desk, and decides to check in at the express kiosk where he can obtain similar service. Upon tapping the “check-in with previous reservation” button at the terminal, he places his fingerprint on the 500 dpi fingerprint reader with built-in encrypted communications to the server. If he had chosen the highest level of security previously, his fingerprint template will only be stored in an encrypted format, and he would have to enter the password to decrypt the fingerprint template for the rest of his stay. If he were even more concern with privacy, he would have to check-in at the front desk to receive a smart card, which will serve as a proxy for his fingerprint. Since he is not so cautious about security, he did not choose either of the options, and the computer system can attempt to identify him amongst the anticipated arriving hotel guests. Since the identification task is performed via an adaptively parallel network of computers, he receives a confirmation almost instantaneously. The computer system has already checked that his credit card is still valid, and another fingerprint swipe represents his approval to pay all charges in relation to his stay at SunSprings. A printout is also available for his room number in case he forgets. In addition, his personalized business card is also printed with his contact information so that others can contact him directly either via telephone, or via the Internet. In fact, he has chosen his personalized business

card to contain his picture and his company logo, since he wishes to maintain a business presence in Perth for his stay but does not have an office.

### **2.1.3 Entering the room**

Upon reaching the room, the hotel guest uses his fingerprint to gain access, using similar algorithms previously described. If he is traveling with companions, they too can have similar access. Furthermore, if he rents many rooms in a large party, every member of the party can have access to all of the rooms, if they so desire.

### **2.1.4 Enjoying built-in facilities in guest room**

When he enters the room, he finds that his user agent has been doing a good job. In fact, his user agent has been running behind the scenes ensuring that the cleaning maid does not forget to include a razor kit, which is a non-standard item for the bathroom. (He has previously informed the agent that he will need one on every trip since he travels light.)

The refrigerator is stocked with only his favorite foods, since he does not intend to eat out during meal times when he is not entertaining. The digital photo frame by his bed has been personalized to a picture of his family, and the automated bathtub is filled with warm water. Furthermore, the fingerprint-controlled safe in his room allows him to quickly store his valuables without setting a password.

He goes to the room computer which duals as a wide-screen television, and he feels that it is one of the best features of the room. The computer interfaces with the user via a wireless keyboard with attached fingerprint scanner, and also via voice recognition. He sees that the user agent has already set his wake-up call on the computer screen, and has even remembered which song he prefers to be played from a selection of music in MP3 format. From his computer, he can browser the Internet with a Java-enabled Web Browser, or lease a selection of applications such as Word Processing, File Storage and Spreadsheet Tools. He can also send a fax, or set up a conference call by entering a series of telephone numbers or choosing entries from his online address book to be included. He can view his hotel bill and his current rate, and extend or shorten his stay. If he has any problems, he can participate in a live-chat with the hotel customer service, or leave a message if they are unavailable. He can also use the same messaging facilities to communicate with other hotel visitors wherever they log in from, or even friends who load a special applet on the web page given in his business card.

He does a little paperwork, but decides that it is really too late to get substantial work done. He is better off with a good start tomorrow. Deciding for breakfast-in-bed, he uses the room service menu on the computer to order his breakfast. The user agent has customized a short list, which he prefers. Although he may override the settings, the user agent has remembered that he likes his eggs scrambled, and his coffee black. The time for delivery is automatically set to 15 minutes after his alarm goes off. The hotel guest is not sure if he likes this. He reconfigures the order to only be placed when he wakes up, just in case he oversleeps.

The last thing our guest would like to do before he goes to sleep is to plan his schedule for tomorrow. He would like to reserve a table for two (preferably non-smoking, but first availability is more important) at the restaurant for lunch if he can also get a reservation at the massage parlor. Otherwise, he would prefer to go somewhere else tomorrow. His companion for lunch has also updated his schedule on a third party Applications Service Provider (ASP). The user agent has to visit the restaurant reservations agent, and the massage parlor reservations agent, and access his companion's schedule to try to make an appropriate arrangement. It does so in a parallel fashion so that the NP-complete scheduling problem can be done in polynomial time. For dinner, he does not have a particular restaurant in mind. He visits the Activities Center on his room computer to find currently available activities that has been filtered by his agent. In addition to having access to the facilities within the hotel, he may purchase specially priced tickets to musicals, amusement parks, bus tours, cinema shows, and other activities that are beyond the boundaries of the hotel.

Having his entire day planned, our guest decides to go to bed. He turn his lights off, and an electronic "do-not-disturb" indicator goes on, disallowing any housekeeping maid to enter the room even if they tried with their fingerprints.

### **2.1.5 Enjoying hotel facilities**

Other than the features in the hotel rooms, our ideal hotel also offers amenities throughout the hotel. From the front desk, the hotel guest can rent notebook computers

that are part of the wireless local area network (LAN) set up within most areas of the hotel, such as SunSprings' bamboo garden with therapeutic oxygen mists. The hotel guest may also rent a wireless Blue-tooth enabled phone headset for use within the confines of the hotel, or a cellular phone if he plans to make and receive calls outside of the hotel. The cellular phone also duals as a Personal Digital Assistant (PDA), and he can synchronize any PDA, rented or his own, at the many computer kiosks around the hotel. From the download points, he can obtain maps of the region, information about hotel facilities and current activities, and his hotel digital business card. He may also hot sync his address book information to the hotel's database, so that he can easily make conference phone calls from his room computer on this and future trips.

In fact, there is little distinction between the room computers, the computer kiosks around the hotel, and the computers anywhere else within the hotel. His fingerprint offers instant identification, which allows him to have access to the same computing facilities anywhere. For example, our hotel guest may have prepared a presentation slides from his hotel room last night, and he may easily retrieve and show the same presentation in the computers in the meeting rooms at the business center. He may be in the hotel lobby ordering room service so that food will be available when he returns. He may also be surfing the web on the rented wireless notebook, and querying if his wife has finished her session of aromatherapy.

Furthermore, all services in the hotel can be accessed and paid for using the guest's fingerprint. This is convenient in exercise facilities where the guest does not have to bring a physical key when he goes jogging or swimming. If he decides to purchase a

drink at the bar, he does not have to remember his wallet. The guest merely has to scan his fingerprint, and his account will be billed directly.

### **2.1.6 Checking out of the hotel**

The guest can choose to check out at the front desk or from any computer in the hotel. This is possible because there is no key to return, and his fingerprint can be accepted anywhere to prove his acceptance to pay for the charges. During checkout he will receive a detailed window explaining his charges, and he will also receive reminders, such as for items left in his room safe. He also has the option of obtaining transportation services such as hiring a taxi to the airport. Finally, before he leaves, he has the option of erasing all biometric information stored, or he may choose to protect the biometric templates by a password.

## **2.2 Detailed Vision of an ideal hotel for a hotel manager**

Our ideal Hotel Management System is useless if it provides all the benefits to the hotel guest, but does not ultimately provide the required returns on investment to the hotel manager or hotel owner. This section will detail the vision on how the typical hotel manager can (1) obtain the components required, (2) set up the system for the first time, (3) maintain the system, (4) use the system, and (5) upgrade and administer the system.

### **2.2.1 Obtaining the required components**

The required components are:

- 1 Tiny InterNet Interface (TINI) for every access control point where a fingerprint reader is located. These are expected to go as low as \$15 in time to come, but are currently about \$50. More information on TINI is available in the next section on Background Information.
- 1 Fingerprint Scanner at every access control point, and at every computer terminal. These are expected to go as low as \$10-\$20 from our conversations with ETrue [1.3]. However, they are currently priced at about \$60-\$80.
- Computer Desktops in every room, and several running at public locations around the hotel. Each computer should come installed with a standard Ethernet Card. These computers cost as low as \$400 especially if they run the free operating system, Linux.
- Computer Servers, preferably 1 on every floor and at least 1 at the hotel front desk. These computers need not be extremely powerful. A Pentium III with 512 Mbytes of SD-RAM, hard drive, CD-R and Ethernet from TigerDirect [1.4] costs less than \$1000. These Computer Servers may need to run the Windows 2000 Operating System to be compatible with some Fingerprint Authentication SDKs.
- Notebook computers, Personal Digital Assistants, Cellular Phones and other optional equipment will have to be purchased separately.

Since the proposed solution only uses off-the-shelf components, the cost is notably low.

The biometrics industry is ripening for the mass-market, and the cost of biometrics hardware is expected to go down as industry leaders such as Compaq make them standard in notebook computers. Next, the solution uses TINI [1.8] extensively. TINI offers a

commercially appealing price tag of \$15, and yet a Java Virtual Machine and a web server through its Ethernet Interface. Furthermore, the ideal solution uses cheap desktop computers and servers rather than expensive ones. Ideally, adaptively parallel algorithms in a dynamic distributed environment will make up for the slower processing power.

We also note that the cost of computing materials will go down over time, especially with items in the lower end of the computing power spectrum.

### **2.2.2 Setting up the system**

Our ideal Hotel Management System is easy to set up. All computers and devices are preloaded with the same small Java client software, of which one function is as a networked software launcher. The software is built on the Jini architecture, thereby offering spontaneous networking (without any configurations) and dynamically loaded services. In addition, the servers on each floor and at the hotel front desk will have additional specialized software, such as the biometrics authentication SDKs, JavaSpaces, Transactions Service, and Lookup Services.

Upon wiring together the computing devices via Ethernet and powering up, each computing device will differentiate itself through a generation of a unique ID, which it will save for the life of the device. From the unique ID, the central server can identify whether the device is a TINI board, or a computer kiosk, or a floor server, and will dynamically load the correct software to run on the respective computing devices. It remains to associate each fingerprint reader with the correct room numbers, assuming

that no labeling is done during the installation of the system components. The hotel manager walks the hotel once, scanning his fingerprint in sequential order. This is the only task that requires a human intervention, for the setup of the system.

The next section will provide an adequate background on the Jini architecture, and the TINI platform.

### **2.2.3 Maintaining the system**

Our ideal Hotel Management System is easy to maintain, because it is (1) self-healing and robust, offers a (2) central and modular code-base that allows for easy upgrades, offers a compelling (3) dynamic load-balancing scheme that promotes scalability.

#### *2.2.3.1 Self-healing*

A self-healing network is one that allows the individual devices to attach to and detach from the network without disturbing the main application. Our ideal Hotel Management System is self-healing because users of services do not rely on specific servers but whatever is currently available.

This makes it particularly easy for a hotel manager to maintain the system. For example, if all the floor servers go down, the TINI boards that host the fingerprint readers can still find an Authentication Service at the Hotel Front Desk. If the floor servers go up again, or if new servers are added, they will announce their presence so that their services will

be used again. As another example, if a hotel manager wants to disconnect a server for reasons such as backup or upgrading, he or she can do so without affecting the performance of the entire system.

Furthermore, every transaction is only valid for a fixed amount of time, the lease period. This prevents the accumulation of redundant data or the permanent blocking of services, since such resources will be freed when the lease expires. Thus, the manager does not have to monitor the accumulation of objects or services in the computing space.

#### *2.2.3.2 Central and Modular Code-base*

Each piece of software in the Hotel Management System is divided into objects by the service they perform, and is stored on central servers. All the other computing devices do not contain any specific software except the dynamic software loader. Therefore, the hotel manager need only install any new versions of software with the central servers. There is no need to walk around the hotel trying to install specific applications on the various computers. Once the software in the central servers is updated, all other computing devices can be called upon to destroy threads of old process and run the new version.

#### *2.2.3.3 Dynamic Load Balancing*

The central servers in our ideal system perform dynamic load balancing, which uses the current system state to optimize performance. This is done by assigning tasks to nodes better suited for certain tasks due to specific advantages in hardware, or by assigning tasks to nodes that just have less work than other nodes. With dynamic load balancing, the hotel manager can scale up the computing power of the system as required, by adding more computers to the network, or adding computers with different computational power.

#### **2.2.4 Using the system**

The ideal system allows for the tracking of employees and the signing of their timecards. The fingerprint readers everywhere can serve as a check-in point for employees. The manager can receive logs on when an employee first arrives at the hotel, or the length of time a housekeeping maid has spent in each room, and the exact rooms that each employee has access to.

The hotel system also manages the scheduling logistics. Through the dynamic interaction of the employee-scheduling agents, room-scheduling agents and the hotel guest agents, each employee can receive a detailed printout of the rooms that he or she has to clean, the type of cleaning that is required, and any special instructions that is left by the hotel guest. The system always has up to date information on whether the hotel guest has turned off the “do-not-disturb” signal, left the room, or checked out of the room, allowing for more efficient usage of cleaning resources.

Employee access control is another natural extension to our system. Unlike traditional hotels, housekeeping maids need not have a master key or a key to a specific floor. They have access only to their scheduled rooms, and only if the electronic “do-not-disturb” signal has been turned off.

Lastly, our ideal system will implement yield management algorithms to maximize revenues through dynamic pricing. Our hotel offers many services, and already has user agents remembering the preferences of each user. By understanding the customer’s desires, yield management can be used to predict consumer behavior at the micromarket level and optimize product availability and price to maximize revenue growth. An example of an application of yield management is dynamically pricing a bundle of services that a potential guest may enjoy at a price that will be attractive for him.

Yield Management can be used to predict the supply and demand constraints in a hotel so that a pricing scheme is optimized. Some non-linearity in pricing exists. For example, it is much more expensive to do a full clean upon a guest checkout, than a partial clean upon an extension of a guest stay. Our Yield Management algorithms need to understand both the revenue and cost sides of the equation.

## 3 Related Efforts

This section provides information of previous work in related fields, as well as, background information needed to understand the technologies used in our hotel system. Unlike many other papers, this section covers a broader range of material, primarily because our project tends to be a broad one that requires the understanding of many fields. Section 3.1 explores current commercially available hotel solutions. Section 3.2 will then explore advantages of biometrics authentication and identification, and discuss recent literature in that field. Section 3.3 gives a short introduction to the field of yield management, while section 3.4 discusses the various models of computer networks. Finally, section 3.5 covers related efforts in parallel distributed computing models and section 3.6 provide a brief introduction to the TINi platform.

### 3.1 Commercially available hotel systems

Hotel management is not a new concept. Large hotel chains have proprietary hotel management systems, some of which are very comprehensive and modern. However, the cost of modern technology has dropped to a point that a good hotel management system can be easily available to smaller hotels and hotel chains. However, it is surprising to find few comprehensive and yet affordable Hotel Management Systems. Instead, there are fragmented solutions such as Reservation Systems, and Access Control solutions.

### **3.1.1 Room Reservations System**

InnSystems [2.1] offers a simple reservation system with a good graphical user interface. It even offers a simple web site where hotel guests can book rooms over the Internet. Other than these simple features, it is generally unappealing compared to other room reservation systems in its class. For example, the system is not compatible with industry standards such as the Structured Query Language (SQL), making it incompatible with other applications.

GuestLine [2.2] offers a competing product to InnSystems. It has a more open architecture, which allows data to be extracted and entered through SQL. It also implements a two-tiered pricing model, where the price increases once a threshold occupancy rate is reached. While it is appealing in some aspects, it falters in other aspects. For example, it does not allow Internet reservations.

Digital Rez Software Corp. [2.3] offers a more competitive solution with ROS2000. ROS2000 is an integrated reservation management software system that can link various motel chains via the Internet. Various database synchronization methods ensure data integrity, and with adequate bandwidth, real-time synchronization can be maintained. Their product also allows for online Internet based bookings from wholesalers, travel agent bookings, and individual travelers. Finally, the integrated report engine provides report generation capability for efficient management.

InnFone [2.6] deserves a special mention because of its unique solution. InnFone uses the telephone network for networking, via connecting all the telephones to a central station that provides telephone control and voicemail. Front desk personnel can query for empty rooms with specific search criteria using special codes on their phone system. A separate computer interface then allows for the actual guest to be registered. The computer interface can also set up calling permissions and rates on each room, and provide a detailed log and billing on each guest's room rates and call usage. The phone system is also used by house keeping who enter special codes as they begin and finish cleaning the room. This allows the computer system to know when the room is available for occupancy.

Other than these examples, there are countless other room reservations systems such as Reservations Plus, Inc. [2.4] and Inn Scheduler [2.5]. Most allow for a graphical user interface to view the general vacancy of the hotel, such as in figure 4.1.

Demo Hotel													
Guest Q&A...	The Help Line												
Search...	April 2001												
View Options ▾	S14	S15	M16	T17	W18	T19	F20	S21	S22	M23	T24	W25	T26
Double	2	2	2	2	3	3	3	3	3	3	3	3	3
Single	2	2	2	2	2	3	3	3	3	3	3	3	3
Green Room 101	Douglas Myers												
Red Room 102													
Jade Room 103	Client Another												
Pearl Room 104													
Tan Room 105													
Beige Room 106													
Show Today	◀	◀	04/14/01	Walk In...	Edit...	Delete	Group...	▶	▶				

Figure 4.1. An overview of the hotel vacancy

In addition, the systems allow for entry of guest information, such as name and credit card number. A typical system is shown in Figure 4.2.

**Reservation Information**

RESERVATION ROOM FOLIO

Company: **ScheduleEase** Created: **04/14/01**

Last: **Myers** First: **Douglas** Confirmation: **3070107566**

Address: **1525 Spruce No 101**  Confirmation Letter Printed

City: **Boulder** State: **Co** Zip: **80302** View By: **Guest Name**

Phone: **3034422677** E-Mail: **macinn@aol.com**

Notes: **This is a demonstration reservaton card. this card contains information about the guest.** Card: **Yisa** Exp: **10/03**

# **6728-45006-5891**

Agent: **Select Travel Agent - Karen Stockton**

Clerk: **Reservation Clerk** Ref: **Marketing Reference Menu**

Type: **Corporate Class** Tag: **Primary**

Type: **Government** Tag: **Federal**

Type:  Tag:

Type:  Tag:

Type:  Tag:

**0 Messages**

Figure 4.2. A typical interface for guest information entry and retrieval

Finally, a typical system will allow for room rates to be set, and an invoice to be printed when the guest checks out. Figure 4.3 shows a typical screen for a guest checkout.

**Reservation Information**

RESERVATION ROOM FOLIO

Adult  Child

Arrive  Depart  Nights

Room  Depart

Rate  Charges

Type  Discount

Date  Subtotal

Type  Tax

TOTAL

DEPOSIT INFORMATION  
Type

Date	Type	Qty	Adjust	Occupancy	Qty	Adjust	Additional	Daily Total
04/14/01	Adult	1		50.00				50.00
04/15/01	Adult	1		50.00				50.00
04/16/01	Adult	1		50.00				50.00
04/17/01	Adult	1		50.00				50.00
04/18/01	Adult	1		50.00				50.00

Figure 4.3. A typical screen for a guest checkout

We find most of these commercially available solutions to be targeting the low-end hotel chains, rather than the middle range of hotels that we are targeting. They are generally lacking in many features. For example, they do not allow for yield management where prices change dynamically according to factors such as demand and availability. They do not harness the true power of the Internet to customize preferences, such as wake-up call schedules or breakfast-in-bed orders, either with ad hoc methods or through support for technologies such as the Microsoft Hailstorm initiative.

### 3.1.2 Access Control System

Hotel Access Control Systems are often packaged as a separate product. An example of one company that provides access control solutions to Hotels is TesaLocks [2.7]. An image of their solution can be seen in Figure 4.4 below.



Figure 4.4. TesaLocks Solution for Hotel Access Control

Whenever a new hotel guest checks in, a person in the front desk will have to obtain a new key card, and manually code the card using a special device. The key card then unlocks the correct hotel door but only for the length of the hotel guest's stay.

TesaLocks' system also logs the entry and exit of individuals in a hotel room, through the different key card used.

### 3.1.3 Summary of commercially available solutions

Specific criticisms about currently available solutions have already been discussed. In general, the biggest problem with currently available solutions is that they are

fragmented. For example, they provide for room reservations and sometimes billing solutions, but they are separate from access control systems, or the wake-up call system, or the room service system. In order for the hotel to function, it is then necessary for a human counterpart to perform the necessary tasks of interfacing between these systems. Therefore, to check into a hotel, a person has to be running the front desk. While the check-in process for a previously reserved room may be automated, the hotel staff has to enter the access control codes on another system, physically code a key card, and present it to the hotel guest. It is hard to implement an express check-in kiosk where a frequent customer can check-in without the assistance of a hotel staff.

In conclusion, currently available hotel management software for the middle to low-end hotels leaves much to be desired. While the use of the Internet and databases has become more widespread, they have not been implemented or have not been implemented effectively in the existing hotel management systems. Furthermore, the hotel management software packages concentrate on room reservations and leaves room pricing to be accomplished with simple or tedious schemes. The software packages are also independent from the access control system. Although some communication can take place between the two systems, the lack of unity discourages the implementation of express check-in kiosks and causes more work for the hotel desk personnel.

## 3.2 Related Efforts in Biometrics Authentication and Identification

### 3.2.1 Reasons for choosing Biometrics Authentication and Identification

There are basically three methods of authentication, each posing a different challenge.

- **What you have.** In this case, a user can gain access to a resource if he or she has the required token. In the case of a hotel, this is typically a key card, such as in the commercially available access controls systems discussed in the previous section.
- **What you know.** In this case, the user can gain access by providing a password. Examples of this are the keypad entry systems, where a person has to enter a sequence of numbers before being let in.
- **Who you are.** In this case, a user gains access via presenting a set of biometric features that is unique to the user. For example, the user may provide a voice sample or a fingerprint image to be authenticated.

To choose an access control system, we had the following criteria:

- The system should be convenient to use.
- The system should be simple enough to be used by all types of users.
- The system should integrate seamlessly with the rest of our hotel management system.
- The system should have an unparalleled level of security.
- The system should not just offer authentication but also identification, so that it can dual other features such as billing in services obtained within the hotel.

From the following criteria, the best possible implementation of access control was through biometrics. A physical token system such as the use of keys, key cards, iButtons, Smart Cards and USB tokens did not meet our criteria because the hotel guest is

inconvenienced by having to carry the card whether he or she goes. Furthermore, a physical token can be stolen and hence pose as a security hazard. Moreover, it was hard to use off-the-shelf components to vend physical tokens, as necessary for an automated express check-in kiosk.

On the other hand, a password access system did not meet our criteria either. The system is not simple enough to be used by all users, especially the elderly or the young who have problems remembering passwords. This leads to passwords being written down, and hence a security hazard.

In both physical tokens and password schemes, true identification is not provided. What has been identified is really only that a particular person is carrying the valid physical token, or that he or she knows of the password for some reason. [3.1] This makes it more difficult to be used for identification, especially for purchasing services around the hotel. For example, if someone disputes a charge for a restaurant meal, he or she may claim that his or her keycard or password was compromised, and it may be hard for the hotel to prove otherwise. On the other hand, if a valid biometric data was submitted, the hotel may have a stronger claim on that transaction, especially with error rates such as 0.001% [3.1, 3.2, 3.3].

### **3.2.2 A survey of current biometrics literature**

According to the BioAPI Consortium [3.4], the process of using Biometric technology for authentication can be divided into two sub-tasks. The first task is Enrollment, where the

physiological or behavioral characteristics of the individual is recorded and calibrated into a template. The second task is Verification or Identification, where the individual is verified with a single template or matched with a multitude of templates.

Jain, Hong, Pankanti and Bolle [3.5, 3.8] describe in detail how enrollment and identification can be implemented in a fingerprint identification system. Ridges are first detected using a local maxima algorithm on a grayscale fingerprint image. Minutiae detection follows by using the ridge map as a starting point and smoothing for breaks between ridges. This information can be stored in a template for future identification. The identification procedure will align the observed fingerprint against this template (considering rotations), and obtain a fitness score.

Ratha, Connell, and Bolle [3.6] point out possible attacks on biometric identification systems. One possible attack is the capture and replay of previously sent biometric data. This brings up the need for the biometric device to encrypt and time-stamp all information leaving the reader. Another possible attack is a brute force approach, where different fingerprint combinations are attempted. Increasing the image resolution can prevent this.

Finally, privacy concerns about capture of biometrics information can be alleviated through Cancelable Biometrics [3.7], where the biometric information such as a fingerprint can be warped using a randomly generated seed and a one-way

transformational mapping, before being stored to a template. Destroying the seed effectively renders the template useless, therefore canceling the identification ability.

### 3.3 Related Efforts in Yield Management

Robert G. Cross defined Yield Management or Revenue Management as "the application of disciplined tactics that predict consumer behavior at the micromarket level and optimize product availability and price to maximize revenue growth". Yield Management was applied successfully to the airline industry in 1980s when deregulation was the catch phrase. Since then, it has been applied to other service industries (that tend to have a large latitude in pricing) such as in rental car companies and hotels.

Ryzin and McGill [4.1] provide a good overview of how to forecast demand functions, control overbooking to maximize profits in the airline industry. While still useful, much of the yield management literature is more focused the airline industry, which has a different operating environment than a hotel. For example, the network structure is different, since airline passengers go from an origin to a destination via journey legs, while hotel guests stay within the same room in consecutive nights stays. As another example, hotel patrons often change their length of stay at a moments notice, but airline passengers less frequently change the journey legs on their flight. However, Baker [4.2] specifically focused on the hotel industry and developed heuristics to model and optimize for consecutive nights and varying length stays.

## 3.4 Related Efforts in Networked Systems

This section provides background information on Networked Systems. It begins with the traditional models of networking where data is transferred from client to server. It then discusses Remote Object Systems where code and data is transferred. Thereafter, section 3.4.3 discusses the evolution of static computing networks to dynamic spontaneous networks through Service Discovery Protocols. Finally, section 3.4.4 describes the Jini Architecture as a specific example of a Service Discovery Protocol.

### 3.4.1 Traditional Networked Systems

Since computers were first built with applications that were stand-only units, the obvious solution merely move data to the computation on computers that can handle such data. By spreading computation over a set of computers, computational power is increased. Furthermore, some computers are specialized for particular applications, or conversely, some applications are written for certain operating environments. Since some pieces of code cannot run on all computers, moving data to computers that can run the required chunk of code is required.

However, different machines represent the same data in different formats. The byte order, floating point format or word length may vary from computer to computer. For example, the Ethernet standard suggests that bytes be encoded in "little-endian" style [5.1], i.e. the least significant bit first. Other devices suggest that bytes should be encoded in "big-

endian” style, i.e. the most significant bit first. For data to be moved from one computer to the next, they must share the same understanding of the way data is formatted.

Various standards for the description and encoding of data are created. For example, External Data Representation (XDR) [5.2] was a Sun Microsystems protocol from the mid 1980s. XDR has been used to communicate data between such diverse machines as the SUN WORKSTATION, VAX, IBM-PC, and Cray. Another standard is the X.409, ISO Abstract Syntax Notation, which is similar to XDR except for the fact that XDR uses implicit typing, while X.409 uses explicit typing.

Using one of these standards, data can be transported from one computer to another without any misunderstandings. Once the data is in the remote computer, the programmer needs to specify what operations are applied to the data.

One way of specifying this is through a Remote Procedure Call (RPC). Birrell and Nelson [5.3] provided an excellent background for the remote procedure call concept. In a basic implementation, the caller process initially sends a call message to the server process and waits for a reply message from the server. The call message includes the procedure's parameters, and the reply message includes the procedure's results. Once the reply message is received, the results of the procedure are extracted, and caller's execution is resumed. Therefore, a RPC makes a remote call look to a programmer like a local function call. Sun Microsystems RPC protocol [5.4] is one implementation that allows for concurrency so that many clients can call the server at one time, and

asynchronous behavior so that each client will not block but rather may do useful work while waiting for a reply.

### **3.4.2 Remote Object Systems**

Yet another way to handle data in a remote computer is through a Remote Object System. The motivation for a Remote Object System is that RPC models can ship complicated data structures across a network, but they do not provide explicit support for objects in Object Oriented Programming (OOP). OOP has many advantages, particularly code-reuse and modularity, and more information can be found in [5.5, 5.6]. Such objects carry code as well as data in a “black-box”.

An example of a Remote Object System is Common Object Request Broker (CORBA). CORBA is language-independent, using an Interface Definition Language (IDL) for specification of interfaces of a distributed object system. [5.9]. In *Distributed Object Computing With CORBA* [5.7], Steve Vinoski provides a good understanding of CORBA and how to apply CORBA in C++ applications to allow C++ objects to run on remote servers. Another example of a Remote Object System is DCOM. A detailed comparison and discussion of CORBA and DCOM can be found in [5.8].

Java Remote Method Invocation (RMI) is Java’s version of a Remote Object Model, and it allows an object on one Java Virtual Machine (JVM) to run methods of another object running in another JVM while hiding the low-level data representation and network communication.

All of these systems from traditional models such as RPC or more recent Remote Object Models such as CORBA share one similarity. They attempt to make the network transparent, so that remote objects can be treated as they are local objects. [5.10] There is a good rationale for this. By making the network transparent, programmers do not need to relearn a new language, and much of the programs written for local software can be reused as networked software.

### **3.4.3 Service Discovery Protocols**

While Remote Object Systems allows objects on one machine to run on another machine, the client has to seek the server before it can call upon the objects of the server.

Traditionally, in a fixed computing environment, this is performed by a lookup service such as DNS, LDAP, and CORBA Naming Service. [5.12] These protocols do not allow for spontaneous discovery and configuration of network services, since they were primarily designed for a statically computing environment.

With the rising number of Internet Services, automatic service discovery become an important feature. [5.13, 5.14] Services need to be able to make themselves available to other devices or users without having any configurations. This means that users of such services do not have to search for the IP addresses of installed devices or install device drivers. They should instantly be able to locate and access the services that are available in the network.

There are many Service Discovery Protocols such as Jini, Universal Plug-and-Play (UPnP) [5.17], Salutation, Service Location Protocol (SLP) and Intentional Naming System (INS) [5.15]. All of these protocols allow for services to announce their presence and capabilities in a network as they become available, so that others can discover such services with minimal administration and human intervention. A good comparison between these protocols can be found in [5.14].

### **3.4.4 The Jini Architecture**

For this thesis, the Hotel System is built on the Jini architecture. It is therefore important for to give a brief background on Jini.

Jini is built on top of RMI, which provides for the low-level data representation and network communication between services. Through Jini, services can be federated easily into a single dynamically distributed system. [5.10, 5.11] The dynamic nature of a Jini federation enables services to be added or withdrawn at any time, without adversely affecting the performance of the federation. Thus, services can offer true “Network Plug and Play” functionality --- they can locate other services in the network, advertise their services, and obtain any required device drivers to communicate with other services.

This is performed by three basic protocols: Discover, Join and Lookup. Services in a Jini network register themselves with the Discover and Join protocols. Discovery occurs when a device or application finds a lookup service, either by a Multicast request protocol if the location of a lookup server is unknown or otherwise by a Unicast request

protocol, while Join occurs when the device or application registers with the found lookup service. Lookup occurs when a client or user needs to locate and invoke a service. A finer-grain selection of services can be accomplished through the use of Entries that contain descriptive information of each service [5.11].

There are 4 features of Jini that are particularly important for an implementation of a commercial Hotel Management System.

The first feature is the security model, with notions of a principal and access control list. This provides an additional layer of security in ensuring that housing keeping maids do not have access to customer information for example.

The next feature is the concept of a lease. A lease is a term of commitment that is negotiated between a service provider and a service user. This allows for the self-healing properties of a Jini federation. A particular service cannot be tied up forever upon a device or network failure, but in the worst case, will become available upon the expiring of its leases.

The third feature is the concept of a transaction. Using the two-phase commit protocol [5.16], a user can ensure that a series of operations either occurs together or not at all, but in no case will one operation occur without the rest of the operations occurring. This is particularly useful in ensuring that one account is debited only if another account is credited.

The last feature is that of distributed events. An object may allow other objects to register interest in events in the object and receive a notification of the occurrence of such an event. This enables distributed event-based programs to be written with a variety of reliability and scalability guarantees.

### 3.5 Related Efforts in Distributed Processing

This section explores recent work on Distributed Processing that can be performed on the Networked Systems discussed in the previous section 3.4. Section 3.5.1 begins by describing the Tuple-space model where processes share a distributed virtual memory space. Section 3.5.2 then details JavaSpaces, which is used in the hotel software implementation. Section 3.5.3 explains the rationale for choosing JavaSpaces. Finally, section 3.5.4 describes mobile agents as another means of parallel computation, also used in the hotel software implementation.

#### **3.5.1 Tuple space based coordination languages**

Linda [6.1] was the beginning of tuple space based coordination languages. A tuple is an order set of fields, each field being defined by a type and a value. Tuples can be shared between nodes through shared associated memory called tuple space. Tuples can be searched and retrieved using templates. Templates are tuples that may have unspecified values to match any value in tuple space.

Processes do not communicate directly, but rather by reading and writing variables into tuple space. This concept results in interesting properties such as anonymous asynchronous communication, networked variable sharing and expressive parallelism. These properties are particularly useful for coordinating processes. Anonymous asynchronous communication allows a central server, without blocking, to communicate to many other process that many not even be running yet or may be temporarily disconnected from the network. (This can be contrasted to parallel processing that involves message passing or RPC calls between known servers to known clients.) Networked variable sharing allows for processes to perform atomic transactions. Finally, tuple space exhibits expressive parallelism because any task can be easily expressed into many tasks, and left in tuple space to be evaluated as and when processes become available.

There are many variants of Linda systems, such as the Piranha system [6.6], which offers “adaptive parallelism” by allowing a set of processors to withdraw from computation as it proceeds. There are also many tuple space based coordination languages other than Linda. Examples include JavaSpaces, GLOBE [6.2] and IBM’s T-Spaces [6.3]. We have chosen to build our Hotel Management System using JavaSpaces, which we will detail in the next sub-subsection.

### **3.5.2 JavaSpaces**

Like Linda, a JavaSpace holds entries, which are typed groups of objects, in an associative shared memory. Entries can be written to a JavaSpace and searched for using

templates. Similar to Linda, templates are entries with unspecified or null values for some objects to denote wildcards.

The two lookup operations on JavaSpace are read and take. Read returns any entry that matches the template, while write does the same except that that entry is also removed from JavaSpace. In both cases, if there is no matching entry, a null value is returned.

A powerful extension to Linda is the Distributed Event Model. Processes do not have to keep querying a JavaSpace for template matches. Instead, they can request to be notified of such an event. Another improvement is the concept of Leasing, which is similar to that of Jini. This frees the JavaSpace of garbage that may collect otherwise over time.

Like Linda, all operations that modify JavaSpaces are transactionally secure. For example, this will provide assurances that no two processes will receive the same entry in two take operations. In addition, multiple operations in several JavaSpaces of several threads can be clustered together as one atomic piece through the use of Jini Transaction objects. This will ensure that all of the operations or none of the operations will occur.

This is an improvement to Linda, which does not support multiple tuple spaces.

JavaSpaces also make it easy to perform parallel algorithms through a compute server.

[6.5] A compute server is a computing device that accepts tasks, computes them, and returns results. A master process can generate tasks and write them as objects in JavaSpace. Worker process will then read such objects and call specific methods of the

object that may run computationally intensive code, before writing the results back into JavaSpace. The worker process is thus calling a similar function as eval() in Linda-type systems.

By adding Jini Transactions into the take(), eval() and write() operations of the worker processes, JavaSpace can implement an adaptive parallel system similar to Piranha [6.6], where a process can withdraw at anytime without adverse effects. While all of the computation that has been done thus far is lost, the simplicity of the design is appealing. This strategy is thus used in the Hotel Management System. As a specific example, processes that identify a given fingerprint with a list of templates may withdraw or fail without adversely affecting the result, since the Jini Transaction aborts and return the unaccomplished task back into JavaSpace.

A more detailed description of JavaSpaces can be found in [6.4].

### **3.5.3 Reasons for choosing JavaSpaces**

A critique of JavaSpaces for scientific computation can be found in [6.7]. The biggest complaints about JavaSpaces are in space and time considerations. It was found that null objects written in JavaSpace take up much more space than comparable systems. The number of writes and reads per second is also lower than other technologies such as T-spaces. The report also found that JavaSpaces was slower than sequential C++ in applications such as Image Processing, Prime Number Generation, and Sequential Iterators. In Image Processing, the file size was large, and consequently, the

communication to computation ratio was too large to rationally allow a parallel processing environment. In Prime Number Generation, each prime was an entry in the JavaSpace. Since there are 216K primes in the benchmark test, and since JavaSpace is considerably slower than typical Linda systems for reads and writes, the combination of these two factors quickly became the bottleneck. Lastly, in Sequential Iterators, communication occurred between known senders to known receivers to share boundary information. As a result, the advantages of JavaSpace were not reaped. Conversely, the disadvantage of communicating through a central process became apparent.

We agree with most of what is said in [6.7]. JavaSpace, and even Java in general, is currently slower than most other platforms. However, in some cases, the problems do not apply to our proposed hotel system. In other cases, the advantages for the use of JavaSpace far outweigh the disadvantages.

At face value, the example of Image Processing seems similar to our attempt to parallel fingerprint recognition. However, after more in-depth research, we find that the example of Imaging Processing does not apply to us. The largest entries that we are dealing with are small fingerprint images or templates of images, both of which are less than 50 Kbytes. Moreover, these fingerprints are captured by TINI, which is a very weak client unable to process the required authentication algorithms. Next, we are processing each fingerprint image as a whole. The division of labor comes when we are trying to match a given fingerprint with many templates given a One-To-One matching algorithm. If each match takes 1 second and is sequentially executed, it would require 50 seconds to identify

a user against 50 possible templates. However, if 50 computers work on this problem in parallel, the task is completed in 1 second plus an insignificant overhead in reading and writing from JavaSpace. The total is still a fraction of 50 seconds and a more acceptable figure.

The next example of Prime Number Generation shows how quickly a JavaSpace can become a bottleneck if it has too many entries. For this reason, we choose to put only current entries on the JavaSpace. Information that is not required for day-to-day operation, such as information of departed hotel guests is stored in a SQL database for future retrieval. This ensures that the number of entries is far lower than the 216K entries of their example.

The last example where JavaSpace fails to provide acceptable performance is with Sequential Iterators. Where specific messages have to be passed from a known process to another, direct communication is preferred to indirect communication. For this reason, the hotel software implementation uses another means of parallel computation. The next section on Mobile Network Agents explains how they can be beneficial compared to JavaSpace.

For the tasks that are ideal for a tuple-system, however, there are some strong reasons to stick with Java and JavaSpace. They are listed here.

- i. Java is platform neutral, which is important for our Hotel System. This means that code written once, can literally run anywhere, whether it is on a \$15 TINI processor board running a JVM, or a \$400 desktop running Linux, or a \$1000 server running Windows 2000.
- ii. In Java, process startup latency is incurred only once, allowing the cost of this latency to be amortized over the running period of the process. This is not true for other shared memory solutions such as PMI and PVM.
- iii. We inherit all the advantages of the Jini platform, such as “Network Plug and Play”. A hotel manager does not need to configure the system for every node. In fact, he or she never even has to install or update each node with new software. Each remote node only needs to have a software launcher class, and will download new versions of code from the server when it is commanded by the server to do so.
- iv. JavaSpace is persistent. During the failure of a JavaSpace, much of the data that has been saved before the crash can be retrieved again.

### **3.5.4 Mobile Agents**

This section discusses previous work on Mobile Network Agents, and attempts to discover when they can be an alternative means of parallel computing to JavaSpaces.

Mobile network agents are programs that represent a user in a network. They can migrate autonomously from node to node to perform computations on the behalf of its creator

[6.8]. At any node, they can obtain access to local services, data, or even other agents, if they have the required credentials. Unlike remote procedure calls where a process invokes procedures of a remote host, mobile network agents are executable code that travel and interact with the remote system. Therefore, a natural platform for networked agents is therefore Remote Object Models such as CORBA and RMI. Indeed, there are many mobile agents built on top of these platforms, such as [6.9].

It is not difficult to implement a framework for Mobile Agents in our Hotel Management System given the reliance on the Jini Architecture and JavaSpaces. The more important question is when JavaSpaces proves to be a more natural and efficient solution for processing a task than Mobile Agents, and when the converse occurs.

The last section 3.5.3 reveals that JavaSpaces is the ideal platform to match a specific fingerprint to a set of possible templates, since the task is naturally divisible and can be done anonymously. In general, when communication can occur between anonymous senders to anonymous receivers, JavaSpaces is preferable to Mobile Agents and is used in our software implementation.

On the other hand, agents are sent to specific services to perform a task. Where communication occurs between known senders to known receivers, Mobile Agents have the upper hand, since JavaSpace will prove to be an extra hop and a bottleneck. For example, mobile agents handle the task of scheduling better than JavaSpace. Using mobile agents to solve a scheduling problem is not new. In [6.10], the Ajanta project

proposes that each user should have a “calendar” agent. In arranging a meeting, a scheduling agent visits each one in turn to find a possible time-slot.

This project attempts to bring the scheduling capabilities proposed by Ajanta to the next level. Instead of having a fixed goal of scheduling a meeting, vacationers often have a list of activities that they will like to complete, but they do not have a preference on the order the list of activities is completed. In the worst case, finding a solution is a NP-complete problem, since each combination has to be attempted. An agent is a natural method of solving this problem. In addition to being able to visit each “calendar” agent to perform more complex queries without costing network bandwidth and latency, the agent can spawn into multiple copies for different decisions that he or she might have to make. Thus, while the number of agents become quickly exponential, if the network traffic or “calendar” agents are not a bottleneck, it is possible to find a solution in polynomial time. In comparison, solving the scheduling problem with JavaSpace is less efficient. The JavaSpace will fill up quickly with agents or messages between agents, thus falling for the same pitfalls as the Prime Number Generation or Iterative Solver Applications described earlier in Section 3.5.3.

### 3.6 Background Information on Tiny InterNet Interface (TINI)

The proposed implementation uses the Tiny InterNet Interface (TINI) platform [1.8], developed by Dallas Semiconductor, extensively. TINI combines a small but powerful chipset, using the DS80C390 microcontroller, and a Java-programmable runtime

environment that maintains code in Flash memory in the absence of power. The DS80C390 microcontroller supports several distinct forms of I/O such as serial, 1-Wire and Controller Area Network (CAN) bus. It also provides several general-purpose port pins that can be used to perform simple control tasks such as driving relays and status LEDs. Lastly, TINI also features 10/100Base-T Ethernet interface. A summary of the features of TINI can be seen in figure 4.5.

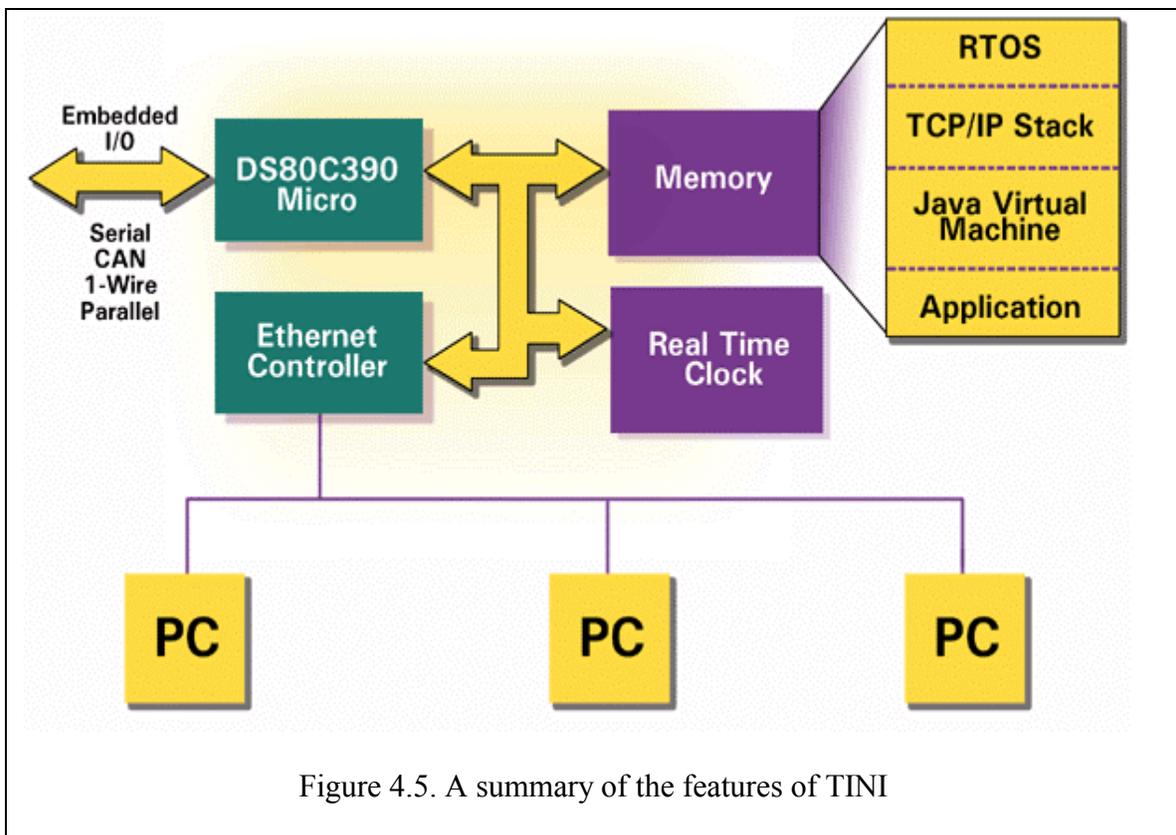


Figure 4.5. A summary of the features of TINI

Using the TINI chipset, a fingerprint scanner can be connected to an Ethernet Network using Jini Applications running on the JVM. The general-purpose port pins on the TINI chipset can also trigger the relay of an electric strike lock of a door.

## 4 Design Overview

### 4.1 Considerations and Criteria

Much of the consideration and criteria for a good hotel system is vocalized in our detail vision of the ideal hotel. In the interest of brevity, this section summarizes the properties an ideal Hotel Management System. The ten criteria are:

- i. Provide comprehensive services.** The solution needs to provide a broad range of services for Hotel Management, rather than a fragmented solution that has to work in conjunction with other solutions.
- ii. Provide superior services.** Services provided need to be judged by quality.
- iii. Computational Efficiency.** The system should be computationally efficient.
- iv. Easy to install.** The system should be plug-and-play enabled.
- v. Easy to maintain.** The system should be robust and self-healing. Failures should be isolated.
- vi. Easy to upgrade and backup.** It should be easy to upgrade the system software. During the upgrading or backup process, critical operations in the hotel should not be disrupted.
- vii. Affordable.** In order to be valuable to hotel managers, the system needs to be relatively inexpensive.

- viii. **Customizable.** The system should be modular, so that features can be customized for different hotels.
- ix. **Scalable.** The solution should be scalable to meet the needs of different sized hotels.
- x. **Web-enabled.** The solution should have no boundaries. By being web-enabled, the system should be able to serve potential customers worldwide, and also interface with other hotels to potentially work in a hotel chain.

## 4.2 Design of the Hotel Management Software

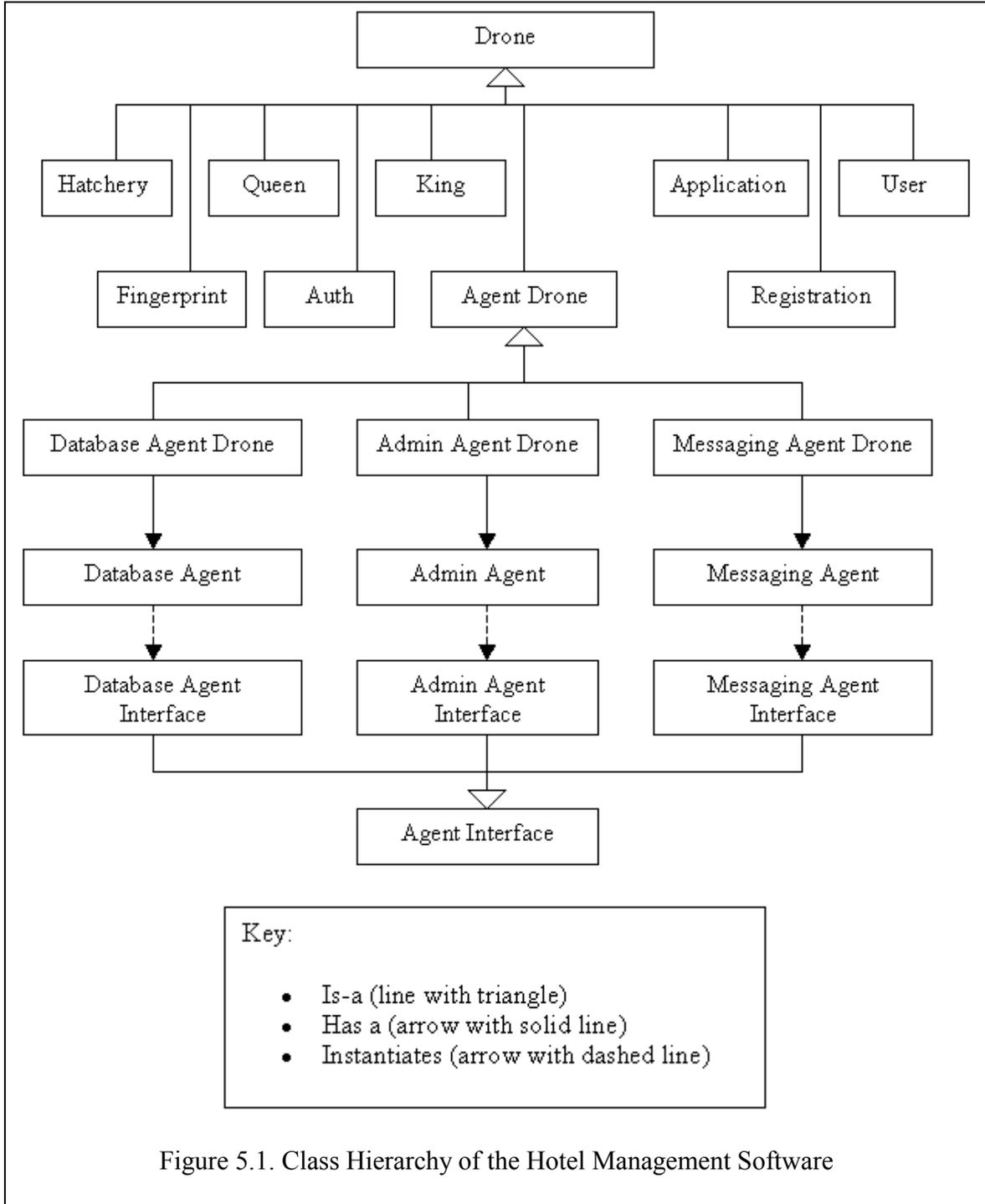
In this section, we will provide an overview of the design of our Hotel Management Software. More details on specific drones can be found in the discussion in section 5, or in the scenarios in section 6.

### 4.2.1 Class Hierarchy

The classes in our Hotel software are all related through the class hierarchy shown in figure 5.1 on the next page. All of the classes shown are either a subclass of Drone, or in short, Drones, or they are an implementation of Agent Interface, or in short, Agents.

Specific types of Drones include the Hatchery, King, Queen, Application, User, Fingerprint, Auth, Registration and Agent Drones. Each of these Drones has specific functions that are necessary for the smooth running of the Hotel Management System. Agent Drones are special because they contain Agents. Examples of Agent Drones are

Database Agent Drone, Admin Agent Drone and Messaging Agent Drone. Each of these Agent Drones houses their respective Agents.



## 4.2.2 Drones

What is a Drone? In our JavaSpace model, a Drone is merely a program that reads and writes entries to and from JavaSpaces. These entries are Java Objects and can encapsulate both data and code. A Drone can read entries from a JavaSpace to obtain new tasks that need to be completed. A Drone can also return completed tasks back into a JavaSpace.

## 4.2.3 Summary of specific Drones

In this list, we summarize the function of some Drones. A more detailed discussion can be found in Section 5.

- i. Hatchery.** A Hatchery watches the JavaSpaces for a Hatchery Entry, which encapsulates code for another Drone. The Hatchery, thus, retrieves the Hatchery Entry and causes a new Drone to run on a separate thread on the same server.
- ii. Queen.** The Queen monitors the performance of Drones on all the servers, and can destroy Drones, or create them at Hatcheries. This allows for workload balancing between several tasks.
- iii. King.** The King obtains a large task and divides it into many smaller tasks. For example, the King can retrieve a One-To-Many Fingerprint Authentication Request, and write corresponding One-To-One Fingerprint Authentication Request back into JavaSpace.

- iv. **User Drone.** The User Drone allows a hotel employee or guest to log into the system via a fingerprint scan, and provides a graphical user interface where the hotel employee can launch Application Drones.
- v. **Application Drones.** Application Drones are then the super class of Drones that provide specific services for hotel guests. Application Drones allows for Word Processing or Room Service Ordering.
- vi. **Fingerprint Drones.** Fingerprint Drones wait for a fingerprint to be scanned at a reader. When this occurs, they send out Auth Request Entries into JavaSpace. If they receive an “Accepted” reply, they contain code to trigger relays that may unlock mechanisms such as an electric strike lock.
- vii. **Auth Drone.** The Auth Drone is a drone that accesses the Biometrics Authentication SDK provided by a third party vendor. They usually obtain One-To-One Authentication Requests from JavaSpace, process the result, and return it.
- viii. **Registration Drone.** The Registration Drone provides a graphical user interface for a receptionist to check a hotel guest into the hotel, or for a hotel guest to check himself in at the Express Check-in Kiosk.

#### 4.2.4 Agents

While Drones provide a good model for an adaptively parallel dynamically distributed system, our discussion has shown that performance can be disappointing for communication between known senders to known receivers. Agents solve this problem. Agents are Jini Services that can communicate and negotiate with one another directly using Java’s RMI technology. Therefore, they do not need to use a JavaSpace as an

intermediary. Agent Drones facilitate the creation, housing, monitoring, and transport of Agents, and are thus the link between the Agents and Drones.

#### **4.2.5 Summary of Specific Agent Drones and their Agents**

In this list, we summarize the function of some Agent Drones. A more detailed discussion can be found in Section 5.

- i. Database Agent Drone.** The Database Agent Drone saves important information from JavaSpace to the Database. The Database Agent can also be contacted directly to retrieve or save information from or to the Database.
- ii. Admin Agent Drone.** The Admin Agent Drone serves as a general administrator for a hotel guest, employee, or facility. Scheduling can be done through the meeting of 2 or more Admin Agents. Messages can be left for the intended party through his or her Agent. The Admin Agent also remembers personal settings and preferences. Therefore, it can be called upon to provide its master's contact list, or to filter from a menu of choices items that its master would be interested in.
- iii. Messaging Agent Drone.** The Messaging Agent Drone can also be classified as one of the Application Drones that provide the hotel guest or employee with application software. The Messaging Agent Drone provides for a graphical user interface from which the user can select to chat live with other users through their Messaging Agent, or to leave messages with offline users through their User Agent. In future implementations, the Messaging Agent will also be able to send a receive electronic mail (E-mail); Every user will be provided an E-mail account,

or he or she may choose to customize the settings so that his or her mail can be received.

### 4.3 Layout of the Hotel Management Hardware

This section describes the preferred layout of the Hotel Management Hardware.

#### 4.3.1 TINI at every access point

TINI, being the cheapest component that is yet fully network-enabled, will proliferate in our hotel. At every access point, such as guest room and conference room doors, restaurant table, and parking lot gate, a TINI chipset would be embedded. One port of TINI would be connected to a fingerprint reader. Another port may be connected to a relay to run the electric strike lock or other access control mechanism, depending on the location. Using conventional Ethernet, the TINI device is connected to the rest of the network. The input from the fingerprint reader is streamed via Ethernet to another computer with higher computing power.

Typical Drones that are expected to run on the TINI device is the Hatchery, FingerPrintReaderDrone and the AdminAgentDrone (for the room).

#### 4.3.2 Desktop Computers

The next cheapest computing device is the desktop computer. These include the computers in each hotel guest room, the computers in the business center, the computers at the hotel front desk, and the computers at the express check-in kiosks. These computers will include a keyboard, mouse, and fingerprint reader. Some of the computers, such as the computers in the guest rooms, will have all 3 devices embedded in an integrated wireless keyboard. On other computers, such as the Express Check-in Kiosks, the computers will also have a credit card scanner and a printer.

Unlike TINI, these computers will meet the minimum processing speed required to process information from the fingerprint readers, and such information may be streamed from TINI via Ethernet. Some manufacturers of fingerprint devices that abide by the BioAPI standard only provide black box software for reading a fingerprint image directly from a serial or parallel port. A crude method of solving this problem is to connect one parallel port of the desktop to another parallel port via a null-modem cable. Using the javax.comm package, the Ethernet stream can be written out of one parallel port, to be read in via the other parallel port by the appropriate black box software.

The desktop computer will probably run Drones such as Hatchery, UserDrone, AuthDrone, RegistrationDrone, AdminAgentDrone (for the hotel guest), Application Drones, and MessagingAgentDrone.

### **4.3.3 Floor Servers**

Near the elevator shaft or near the fire exits of each floor, there will be a floor server. Due to fire regulations on the distance between each guest room door and the nearest exit, we do not anticipate the need for bridges to carry the Ethernet signal across the floor.

The floor server is a more powerful and expensive computer than the desktop computers. In addition to providing the same services as the desktop computers, such as authentication, the floor server would also be running its own JavaSpace. By this design, the division of traffic into multiple JavaSpaces is segregated naturally by floor. On guest room floors, there is no reason to anticipate more traffic on one floor compared to another floor. The JavaSpaces will be named according to the floor number, thus allowing Jini devices on that floor to select the corresponding JavaSpace to be their preferred JavaSpace.

However, the robustness of the system will be compromised if the devices can only have access to one JavaSpace, which might fail. This is not so, since the devices on each floor are connected to the same network on other floors through cabling down the elevator shaft.

The floor servers will usually run the Hatchery, King Drone, Auth Drone, AdminAgentDrone (for hotel employees), and the Database Drone.

#### **4.3.4 Central Servers**

The most powerful computers are the central servers. A few of these are expected to be running at the Manager's Office or Front Desk where they can be physically secure. The Database is expected to be running on these computers or in close proximity. This makes these computers extremely appropriate to run Drones that may access the Database. These include the King Drone, when hotel guests need to be match with specific entry points, and the Database Drone, when an entry is retrieved or written to the Database.

In addition, these are the only computers that need to be updated in later versions of the software. Recall that all the other computers run the Hatchery Drone. The Central Server contains the codebase for the Jini Federation, and can cause new version of Drones to be created (and old versions to be destroyed) on any other computer through the Queen Drone that is running.

In addition, the central servers can also run the Hatchery, Auth Drone, and AdminAgentDrone (for hotel employees).

#### **4.3.5 Rationale for our design**

Our rationale for the current design has been motivated by the following factors:

- i. Several inexpensive computers are better than one expensive one.** Much literature like [6.11] suggest that for operations that can be parallelized, several inexpensive computers perform better than a single computer of the same total cost. Without going into details of Amdahl's law, we note that the presence of numerous distinct hotel guests results in a high degree of parallelism. In light of

this, we are motivated to purchase the number of components inversely proportional to their individual costs. Therefore, we intend to have the highest number of TINI chipsets, followed by desktop computers, then by floor servers, and finally central servers.

- ii. Hotel floors are a natural method of segregating floor servers and JavaSpaces.** While every drone has access to all JavaSpaces, we find that many tasks occur at the same place. For example, a hotel guest will frequently attempt entry at the same hotel room on the same floor. He or she will frequently log in to the computer system from his or her room, rather than from another room. It makes sense for such Drones to access a nearby preferred JavaSpace, for example that on a floor server. This allows for the accumulation of shared variables in one JavaSpace and thus reduces the searching or duplication costs. Furthermore, the proximity of the floor server also reduces network latency.
- iii. Important Servers in a central location desirable.** To reduce the cost of maintenance and security, we find it desirable to have the key servers at a central location. Unlike floor servers, the key servers may contain the actual database for long term storage, or RAID hard drives or other devices for backup purposes. The key servers may also be the firewall between the computers in the Hotel and the rest of the Internet. They also contain the codebase, which may need to be upgraded from time to time, unlike the rest of the computers in the hotel. By placing them in one location, security and maintenance is centralized.
- iv. A higher computing power in the first few levels is desirable.** It is naturally that a disproportionate amount of the computing power required is concentrated in

the first few levels of the hotel, where there are numerous computer kiosks for guests accessing hotel services, and where there may be many guests checking in or out at the front desk. Furthermore, authentication requests at room doors can appreciate the few number of templates required to verify the valid room occupants and the ample opportunity of caching. Identification requests at the computer kiosks on the ground floors do not have the same privilege. The number of different users at those computer terminals may quickly overflow the local caches. If the users are not required to provide further information such as last name, a fingerprint identification request will have to match the given fingerprint to all currently checked-in hotel guests and all valid employees. Rather than treating the first few public levels of the hotels like any other floor, higher computing power than that provided by floor servers is required.

#### **4.3.6 Discussion on other possible layouts**

Due to our thoughtfulness in “Plug and Play” software design, there are many other layouts that can also lead to a functional Hotel Management System. For example, we are also working in close conjunction with a 20-room motel in Johor, Malaysia. The costs concern has led to the elimination of all room computers or floor servers. The entire set up consists of only 20 TINI chipsets for each room door, 2 desktop computers for Express Check-in and 1 central server. The central server can run all of the Drones, as well as the Database, JavaSpace, and other Jini Services. While we are out of specific examples, we can still suggest other designs, such as having more than one “floor server” on each floor, or having one “floor server” serve more than one floor, or having no

central servers but using the floor servers as their replacement, or having one JavaSpace per central server, rather than per floor server.

It is useful to note that other feasible hardware layouts are perfectly viable where our assumptions in section 5.3.5 do not hold. For the case of the Malaysian motel, there are not enough rooms on each floor to justify a computer on each floor. For the other hotels that do not wish to provide hotel guests with software applications, far less computing power is required; There is no need to identify each hotel guest and then launch the required application. Finally, for hotels, with non-standard floor or room sizes, some floors may then demand more computing power, while other floors less. As much as the Queen still performs some load balancing across computers, the segregation by floors become an unnatural division, and the segregation by the number of expected users to a server is more appropriate.

## 5 Details of software implementation

In this section, we go into greater depth on the design of our software implementation.

### 5.1 Drones: Processes that communicate via JavaSpace

Most running processes in our implementation are instances of the abstract class Drone. The set of Drones form the core of the hotel system, and they work together as a Federation in their specialized forms to provide the various services for the hotel.

A Drone implements the Jini Lookup and Discovery Protocols, and therefore has the ability to search for Jini Services such as JavaSpaces, Lookup Registers, Transaction Managers, and Agents. In order to avoid the bottleneck and the central point of failure of a single Lookup Service, we use multiple Lookup Services in our Hotel Management System, and Drones use a Multicast Discovery Protocol to find all the Lookup Services.

All Drones can communicate with each other by writing entries to JavaSpaces. This is the method that most Drones communicate, with the exception of Agent Drones where communication with embedded Agents can occur directly.

It has been recognized that a single tuple space will very quickly be a bottleneck as well as a single point of failure in the system [6.2]. In recognition of this, our hotel implementation supports any number of JavaSpaces. Therefore, the failure of any computer or JavaSpace will not be fatal, and it will be less likely that any one JavaSpace will become the bottleneck.

Drones have built in features that recognize the multiple JavaSpaces, and the potential for failure. In particular, Drones exhibit the following behavior:

- i. They attempt to find new JavaSpaces from scratch every hour. This is an end-to-end solution to obtaining an accurate list of JavaSpaces, especially if a JavaSpace has gone down but has somehow avoided removal, or if a JavaSpace has been added but has somehow avoided detection.
- ii. In addition, they attempt to find new JavaSpaces when they do not know of any valid JavaSpaces. This prevents the Drone to become inoperatable for the rest of the hour.
- iii. Drones have an efficient algorithm to remove duplicate JavaSpaces. They first sort the array of known JavaSpaces, before proceeding to remove the duplicates by a one-time scan. Thus, the algorithm has running time  $O(n \log n)$  in the number of JavaSpaces detected.
- iv. Drones are aware that JavaSpaces are not foolproof. Upon encountering an error with a JavaSpace, they fail gracefully and remove that JavaSpace from the array of known JavaSpaces. Because they find new JavaSpaces every hour, this behavior is not fatal for any JavaSpace-Drone relationship.

- v. Drones select their preferred JavaSpace or one of their preferred JavaSpaces when they need to perform a write operation. When they need to perform a lookup operation, they scan through all JavaSpaces, starting from their preferred JavaSpace or preferred group of JavaSpaces. By default, their preferred JavaSpace is a randomly selected known JavaSpace so that the workload to be distributed evenly across JavaSpaces in the average case. However, the Queen can set the preferred JavaSpace or group of JavaSpaces. This allows for much flexibility towards the goal of load balancing. One choice that we have discussed is setting the preferred JavaSpace to that on the same floor as the device on the floor server.
- vi. A Jini Lease governs the writing of any Entry to JavaSpace naturally. Therefore, upon the failure of the Drone and hence inability to renew the lease, “garbage” entries are removed eventually, and the system is self-healing.
- vii. Every drone avoids querying the JavaSpaces continuously. Instead, they make use of Java’s Remote Event Model, and ask each JavaSpace to notify them of a matching entry. Every hour during the lookup process, the leases to the old notifications are destroyed, and a new set of notifications to the newly found JavaSpaces is created. This is another end-to-end solution. If the Drone has somehow not renewed its notification lease and is it not aware of its failure, we are guaranteed that it will receive notifications again within an hour.

We have gone into great depth about how Drones access multiple JavaSpaces, since every Drone will use JavaSpaces in some form or another. Our programming style, however, has made it particularly easy to search for other Services besides JavaSpaces.

Much of the code in finding Services, and removing duplicate or bad Services is reused. For Drones do require other Services that there may be multiple copies. Examples include Transaction Managers or Agents such as Administrative Agents. In such cases, the exact same safeguards apply to these Services as they do apply to JavaSpaces.

Finally, perhaps the most important property of the Drone is that it can be created, monitored and destroyed by a remote Queen. Every computing device will run a Hatchery, which allows new Drones to be created on that device. This allows the entire code base to be on the central servers, where they can be monitored and upgraded easily, promoting the paradigm “write once, install once, run everywhere”. Monitoring and killing Drones are accomplished via two different entries: the Refresh Entry and the Death Entry. Every drone signs up notification of such entries by default. When they are notified of a Refresh Entry, they will attempt to remove any of their old Queen Entries and write a new Queen Entry to a JavaSpace. The Queen Entry includes information on the server and process that the Drone is running on, the type of Drone that is running, and also additional details such as the performance of the Drone. Every Drone is also notified of Death Entries that match their exact identification, i.e. both their Server and Process Identification Numbers. Upon such notification, they will promptly remove all personal entries from the JavaSpaces, and cancel their existing leases, and finally kill themselves permanently.

## 5.2 Details on some specific Drones

After the detailed discussion on the Drone architecture, it is time to detail specific Drones of interest.

### **5.2.1 Queen**

Recall that each Drone writes a Queen Entry to a JavaSpace, and upon the command of a Refresh Entry, all Drones will refresh their Queen Entries. “Queen” entries are important because they provide information to Queen Drones about all running processes in the hotel. They provide information on the performance, unique identifier and type of each Drone.

One of the Queen’s tasks is to observe the different Queen Entries and then to perform load balancing between computers, as well as, strive an optimal balance between the types of Drones. For example, the Queen can kill Drones that have poor performance on a certain server, and hatch similar Drones on other servers with a better computing speed and resources. As another example, if one type of Drone is required more than other types, the Queen can kill Drones of one type, and replace them with Drones of another type. There can be any number of Queen Drones, and each Queen Drone can monitor other Queen Drones as well. In order to force all Drones to update information on their process, the Queen uses the Refresh Entry. To kill a Drone, the Queen sends a Death Entry for that particular Drone. Finally, to create a Drone, the Queen sends a Hatchery Entry with the actual Drone embedded within to a particular Hatchery.

The Queen is also running on the server with the code base. Therefore, there will be occasions when other Drones require a new Drone to be hatched. For example, the hotel guest may be trying to launch an Instant Messaging service. For this purpose, the Queen signs up to be notified about Queen Create Request Entries. Upon notification of such requests, the various Queens running rush to fulfill all pending requests on the JavaSpaces by removing a Queen Create Request Entry from the JavaSpace, while fulfilling the request by writing the appropriate Hatchery Entry.

### **5.2.2 Hatchery**

The Hatchery has already been described in detail. It waits to be notified of a Hatchery Entry. Upon such notification, it takes the required Hatchery Entry from JavaSpace, and hence the new Drone. It then runs the new Drone on a separate thread. As somewhat of a security measure, the Hatchery will code in a discriminatory fashion, unlike the eval function in Linda-type [6.1] systems.

It is desired that all computing devices need only to run the Hatchery, and upon the running of the Queen, the Hotel Management System will dynamically configure itself, run the appropriate Drones at the proper locations. With this as the goal, our method of sending targeted Hatchery Entries via JavaSpace from the Queen to the Hatchery is not an obvious optimal choice.

An alternative we considered was to have the Queen post all the different possible Hatchery Entries to the JavaSpace. This is appealing if the number of Drones were small,

since this will mean that the Queen will not have to repeatedly upload the same Drone to JavaSpace. However, this is not so. The number of Drones is actually much larger than the type of Drones. This is because we may choose to personalize each Drone before sending it to the Hatchery. For example, we want the Administrative Agent Drone to have its master's preferences and settings. As another example, we want a particular Drones to set a certain JavaSpace as its preferred JavaSpace either because it is underutilized or that it contains much of the information that the Drone will need. As yet another example, we want the Auth Drones on the third floor to have all the fingerprint templates of customers on the third floor pre-cached. The Queen, by design, is in close proximity to the Database. It therefore makes more sense to construct a pre-configured Drone at the Queen, rather than sending generic Drones that require future configuration.

With that being said, it then seems that the communication is from a Queen to a known Drone. Doesn't this stimulate a design of direct communication between the Queen to the Hatchery, rather than going through JavaSpace? There are two reasons why we have not chosen this path. The first reason is that a Hatchery may demand many new Drones at one time. As one example, the User Drone is an extension of a Hatchery, and the user may be trying to launch many programs concurrently. Many Queens in parallel may serve such requests. We do not want communication with the Hatchery to be the bottleneck, and thus block the Queen from serving other Hatcheries or performing other duties. The second reason is that it is not true that the communication is always from a Queen to a known Hatchery. We anticipate future versions of the software, where the ontology of the Hatchery Entry becomes more descriptive, and specifies information such as the floor the

Hatchery is running on. The Queen can then send a Hatchery Entry directed to the entire floor, perhaps containing a User Drone with its preferred JavaSpace preconfigured to that of the floor's.

### **5.2.3 Registration Drone**

The Registration Drone offers a Graphical User Interface (GUI) to assist the operator in registering or checking-in hotel guests. Previous reservations can be brought up via conventional text searches, or via a fingerprint entry. The Registration Drone can also be used to checkout existing hotel patrons, or to search for the room number of an existing hotel guest (if they have permitted such a search).

The Registration Drone is meant to run on the Express Check-in Kiosk, as well as, the front desk computer terminals. When a hotel guest has checked into the hotel, the Registration Drone obtains the required guest information and writes a Registration Entry into a JavaSpace. Any drone that needs to update or retrieve information for this customer can then easily access this Registration Entry. Conversely, upon checkout, the Registration Drone writes a Check Out Entry into a JavaSpace, thereby notifying all Drones that may require such information.

In addition to checking in hotel guests, the Registration Drone is also used to sign in hotel employees. This is used for convenience. A hotel employee can report to work by merely pressing his or her fingerprint at the nearest available kiosk.

Just as the sign in process is analogous to checking in, the sign out process is analogous to checking out. When an employee signs in, he or she will obtain appropriate access to certain facilities. When he or she signs out, such privileges are removed.

We have thus blurred the distinction between the Hotel guest and the Hotel employee. In a sense, both groups are users of the same Hotel Management Software. They need to schedule events, be informed of messages, and declare when they arrive and when they leave the hotel. By forcing them to go through the same check-in process, we increase the simplicity of our system through code reuse, not just in the Registration Drone, but also in other Drones that are informed of the check-in and checkout.

#### **5.2.4 User Drone**

Another Drone with a GUI is the User Drone. The User Drone can be found on the computers in the guest rooms, and on any computer that can be accessed by a hotel guest. For example, the User Drone will run on wireless notebooks that are leased by the hotel, or computer terminals around the hotel such as in the Business Center, or even computers in the Express Check-in Kiosks.

A hotel guest will use such computers by first signing on. He or she can be identified via means of a fingerprint scan. In receiving his or her fingerprint, the User Drone will write One-To-Many Auth Request Entries to the JavaSpace. The request is One-To-Many, since there is one fingerprint, but many possible users that fingerprint might possibly match. Optionally, the hotel guest can enter specific information, such as his or her last

name, to aid the time taken for this identification process. However, since many distributed processes will handle this request, we anticipate that this is not necessary. Upon receiving an approved message in an Auth Reply Entry, the User Drone will display the software launcher menu. That menu will display possible applications. Applications will include Word Processing, Web Browsing, Spreadsheet, Instant Messaging, Room Service Ordering, Account and Billing Information, and Scheduling and Activities Management. When the hotel guest makes his or her choice, a Queen Create Request Entry is written to the JavaSpace. In return, a Queen processing the request will reply with a Hatchery Entry containing the Application Drone desired.

In addition, the User Drone also retrieves targeted advertising from the Advertising Drone and special messages from the hotel guest's Administrative Agent. The targeted advertising displayed can promote deals on room service meals, or electronic coupons for some hotel activity. The special messages, on the other hand, can inform the hotel guest that he has pending Instant messages or E-mail, or may display an announcement from the Hotel Manager.

### **5.2.5 Fingerprint Drone**

Fingerprint Drones usually run on TINI with an attached fingerprint reader, or in close proximity to one. They can be found in areas of the hotel that require access control, such as the guest room doors or the fitness center door. In such cases, the Fingerprint Drone authenticates a given user to give authorized users access to facilities. In other cases, this set of hardware and software can be found at payment points such as in the hotel's

restaurant. In such cases, the Fingerprint Drone serves to identify a given user, so that the correct account can be billed.

Wherever the Fingerprint Drones are, they basically wait for a fingerprint to be scanned at the fingerprint reader. When this occurs, they send out One-To-Many Auth Request Entries into JavaSpace. Like the User Drone, they actually send two One-To-Many Auth Request Entries. The first entry is sent when a finger is first detected, and contains no fingerprint image. The second entry is sent when a fingerprint is captured, and this entry contains an actual fingerprint image. Both entries contain a uniquely generated Case Identification Number or CaseID, which can be used to identify this request while different Drones process the request.

If the Fingerprint Drone receives an “Accepted” reply in an AuthResultEntry to these Auth Request Entries, they can trigger relays such as those on the TINI chipset, which may then unlock mechanisms such as an electric strike lock. They can also bill the correct account for the amount charged.

## **5.2.6 King**

### *5.2.6.1 Details of implementation*

It is the King’s duty to coordinate and breakdown tasks meant for parallelism on the JavaSpaces. One such problem is user authentication. Given the problems with hashing a fingerprint images, most commercially available technology can only match a given

fingerprint image with a single template. It is not difficult to note that if there are 600 hotel guests, and if each authentication step takes only 0.1 seconds, a hotel guest would have to wait for a full minute to be authenticated at a computer kiosk. By using multiple Authentication Drones (or Auth Drones, for short) to work on the task, identification of the same fingerprint can take as little as 1 second. This is done without imposing burden on the user to declare fact such as last name.

Recall that the User Drone or the Fingerprint Drone submits 2 One-To-Many Auth Request Entries, one with a fingerprint image and one without, for each authentication request. Each authentication request is assigned a Case Identification Number (or CaseID for short). All the entries that relate to this request, whether written by a User Drone, Fingerprint Drone, Auth Drone or King will be identifiable by this one CaseID.

The King only takes One-To-Many Auth Request Entries without fingerprint images from any JavaSpace on its list. When a King takes such an Entry from a JavaSpace, it reads the Entry Point variable that reveals the location of the device in which a user is trying to gain access to. If a User Drone writes the Auth Request Entry, for example, the Entry Point variable could point to a computer kiosk at the hotel lobby, or a desktop computer in a specific guest room. If the Auth Request Entry is a written by Fingerprint Drone, the Entry Point could be a specific guest room door, a specific payment device in the hotel restaurant, or business center room door. Therefore, Auth Request Entries are written when there is a need to authenticate a particular user before he or she is allowed access to a resource, or when there is a need to identify a particular user so that user

specific operations can be performed. Whichever the case, the King determines the set of users that need to be matched with a given fingerprint image by checking with the Database Drone. For a guest room door, this will include all the guests living in that room and the assigned cleaning maid (if the room is not in “Do not disturb” mode). For a business center room door or a computer kiosk, the list is the set of all currently checked-in hotel guests.

For each user in this list, the King writes a One-To-One Auth Request Entry a JavaSpace. Preferably, the King will choose the JavaSpace, which is running on the same floor as the Entry Point, so that the network latency is reduced, and the cache hits is increased. Each Auth Request Entry will identify the same CaseID and the identifier for the user for which the fingerprint is to be matched to. The AuthRequestEntries are small, since they do not include the fingerprint image or the fingerprint template of the targeted user.

These One-To-One Auth Request Entries are now processed by Auth Drones. The King waits for results of such operations in Auth Result Entries. If a certain Auth Request Entry has a delayed reply, the King can resend the request in case it is somehow lost in transit or through a faulty Auth Drone. When the King is notified of the Auth Result Entries, the King processes each Entry. If a “Rejected” match is seen, the King will remove that user from the list of possible users. When the list is empty, all authorized users have been rejected. The King will then send an Auth Result Entry back to the originator of the request, whether a Fingerprint Drone or a UserDrone, with a “Rejected” message. If an “Accepted” match is seen, the King will not bother to wait for the rest of

the pending replies, but rather, will immediately send an Auth Result Entry back to the originator with an “Accepted” message. The case is marked closed, and other Auth Result Entries for this request is ignored.

#### *5.2.6.2 Rationale for design*

There are many issues weighing on our minds before we decided on the current implementation for the King Drone. In particular, we are interested in:

- i. Reducing the time taken for an authentication or identification request.
- ii. Reducing the burden on the network.
- iii. Optimizing for the average case rather than the worst case.
- iv. Anticipating and reacting to unexpected failure.

We have discussed the advantages of using a tuple-type dynamically distributed system in adaptive parallelism, especially if the computation to communication ratio is high. The key disadvantage in this application is the need to send the fingerprint images and fingerprint templates to the Auth Drone. To this end, we attempted to optimize for the average case by the following features:

- i. **Two Auth Request Entries.** While there is additional burden on the User Drone or Fingerprint Drone in sending two Auth Request Entries for each case (one with and the other without a fingerprint), there are many advantages. The fingerprint

image recorded by the User Drone or Fingerprint Drone only transits at one node (the JavaSpace), which is the minimum required for a tuple-type system. The User Drone or Fingerprint Drone sends the fingerprint image in an Auth Request Entry to the JavaSpace, and this Auth Request Entry is retrieved directly by Auth Drones that require the fingerprint image.

The King saves time by only retrieving Auth Request Entries without images, which are considerably smaller in size. The King also saves some time since the Auth Request Entry without a fingerprint image can be placed on JavaSpace slightly before that with a fingerprint image. The King can therefore handle more One-To-Many Auth Request Entries in a given time.

In addition, this procedure reduces network traffic by decreasing the amount of data that is transferred between the JavaSpace and the King. On the other hand, the Fingerprint or User Drones are also relieved of the burden of sending the fingerprint image to each Auth Drone. Auth Drones can anonymously read the fingerprint image from the JavaSpace, which may have a hardware and software advantage in handling this task.

- ii. **First Accept Accepted.** With Type I and Type II errors occurring with less than 0.1% possibility in typical fingerprint authentication systems, we assume that the first accepting response from a Auth Drone will be the only accepting response. By doing this, we can return an accepting result in an expected half the time taken than if we were to wait for all the One-To-One Auth Requests Entries to be

replied to. We also save the computing power in Auth Drones by calling the case closed, and terminating pending Auth Request Entries. On the other hand, we are betting that the chances that two users matching a fingerprint image is insignificant. For cases such as a guest room entering a door, we are not interested which fingerprint templates match a given fingerprint, but only that some fingerprint template matches the given fingerprint. This trade-off seems to be a fair one to make.

- iii. **Auth Drone caches information.** In our implementation, the Auth Drone has to retrieve 3 entries to perform a One-To-One Authentication Request. It has to obtain the order from the King that details which fingerprint image has to match with which fingerprint template. It has to obtain the actual fingerprint image that was submitted by the user. It has to obtain the fingerprint template of the user. One reason we provided for the King not sending its orders with the original fingerprint image was that we didn't want the King to waste time downloading the fingerprint image in the first place. There seems to be little reason for the King not to send its orders with the fingerprint template to be matched. The King works in close conjunction with the Database Drone. It may even be running on the same machine as the Database. Given that the King has already looked up the users who templates need to be matched with the fingerprint, it is simple for the King to attach the template to the Auth Request Entries that it sends to Auth Drones.

In our opinion, however, this is actually less efficient. Attaching fingerprint template will slow the King down, and prevent the King from writing all the Auth Request Entries rapidly so that many Auth Drones can work on those Entries. The JavaSpace will also contain more redundant data, since it may contain multiple copies of the same template in Auth Request Entries with a different CaseID. The Auth Drones will also be forced to download the fingerprint template, although this is not a disadvantage if it does not have a cached copy of the template.

Our solution calls for a clear separation between the task of deciding which image is matched with which template, the image, and the template. This allows the Auth Drone the choice of not obtaining either the fingerprint image or fingerprint template if it already has it cached. Furthermore, the chances of a cache hit are increased, since the King chooses the JavaSpace closest to the Entry Point to write Auth Request Entries. Auth Drones that prefer that JavaSpace will have an advantage in processing those requests, and thus are more likely to see the same fingerprint templates used. Although in the worse case, the Auth Drone will have to obtain the 3 Entries in separate transactions, which is slightly slower than obtaining a single large Entry, the potential for cache hits and the removal of the other disadvantages listed make our implementation our ideal choice.

- iv. **King does not wait passively for a reply.** While there are safeguards built into JavaSpaces, Jini, and the Auth Drone, we decided that an end-to-end solution was needed to ensure that every case is resolved within a reasonable time. If for some reason, a Auth Request Entry was not worked on or somehow disappeared from a

failing JavaSpace, there is a possibility that the King will not receive all the Auth Result Entries that it anticipates. Rather than leaving this case open and unresolved indefinitely, the King notices that the case is unresolved within a reasonable time, and resends the Auth Request Entries that it did not obtain replies to.

We provided one specific example where the King is used to divide the task of authenticating or identifying a user with a list of authorized or possible people. The King can have many more responsibilities. For example, there may be a need to perform computations in Yield Management to set optimal room rates based on demand and supply predictions. As another example, there may also be a need to compute optimal advertising and offers that are provided for each hotel guests. Such computations can be done at night, where there is more computing power. The Queen can create new Drones for such computations and remove other redundant Drones, just for the nighttime. The King can the coordinate and divide these tasks as well.

### **5.2.7 Auth Drones**

AuthDrones are notified of One-To-One Auth Request Entries from a JavaSpace. Upon notification, they retrieve Auth Request Entries starting from their preferred JavaSpace, rather than the JavaSpace that they obtained the notification from.

When the Auth Drone take a One-To-One Auth Request Entry from JavaSpace, it obtains corresponding the fingerprint template from the Registration Entry in JavaSpace, if it

does not have it cached. It also obtains the corresponding fingerprint image, if it does not have it cached. It then calls upon 3<sup>rd</sup> party software to match the fingerprint image with that of the template. The Auth Drone then returns the result in an Auth Result Entry. For this entire process, the Auth Drone uses a Transaction Manager to ensure that the Auth Request Entry is only removed from JavaSpace, when the Auth Result Entry is successfully written to JavaSpace. Furthermore, if a result is somehow unavailable within a reasonable time, the entire transaction aborts, and the Auth Request Entry will be available to another (possibly faster) Auth Drone.

After returning the result in an Auth Result Entry, the Auth Drone caches the fingerprint template. If it is out of the allocated memory space, it replaces the fingerprint template that has not been used for the longest period of time. Next, the Auth Drone attempts to take another Auth Request Entry from the JavaSpace with the same CaseID as the last request. This enables it to avoid downloading the same fingerprint image again, since it is cached as long as there are other One-To-One Auth Request Entries with the same CaseID. When there are no more One-To-One Auth Request Entries with the same CaseID, the Auth Drone then proceeds with other One-To-One Auth Request Entries on that JavaSpace, and then those on other JavaSpaces.

### **5.2.8 Application Drones**

There are several Application Drones that can be created by the User Drone. These applications provide services to the hotel guests and employees. One such example is a Messaging Agent Drone, where a hotel guest or employee can communicate with another

user in the hotel. Another example will include a Room Service Drone where the hotel guest can place an order. Yet another example will be a Browser Drone, which the user can use to surf the web. A good way to implement the Browser Drone will be the license Hot Java JavaBeans component [1.7] from Sun Microsystems. Another useful Application Drone is the Scheduling Drone. This Drone provides a graphical user interface for hotel employees and guests to schedule activities through the negotiations of their Administrative User Agent and that of others. The Scheduling Drone allows room-cleaning maids of a hotel to schedule cleaning sessions with rooms that need cleaning, or allows a hotel guest to schedule a restaurant appointment. This list is not all-inclusive. There can be many other Application Drones for purposes, such as Drones for Word Processing, Wake-up Call, or Spreadsheets.

### 5.3 Agent Drones: Processes that can communicate directly

Drones, by themselves, are not actual Jini Services. They communicate with each other through JavaSpaces, thus taking full advantage of the perks of a tuple space dynamically distributed system. For example, many Auth Drones can work in an adaptive parallel anonymous fashion to match a fingerprint to many templates. Auth Drones can be created or destroyed by a King at any time without affecting the user. As another example, the hotel management can send advertising information to all User Drones, but does not have to manually find and connect to each User Drone. Using JavaSpaces as a repository of

share variables also allow for atomic operations, which come in handy for account related operations such as billing.

However, as mentioned in the section 3.5 that covers related work on distributed processing, JavaSpaces tend to be inefficient for communication between known senders to known receivers, since information must transit at a JavaSpace. For such applications, we have suggested the use of mobile software agents that can communicate directly without JavaSpaces. Furthermore, we have found that the Jini architecture was well suited for the development of such agents, since the underlying problem of agent discovery and transport was already solved in the Jini framework.

Our Agent class is thus created to meet this need. Agents are Jini Services that can communicate and negotiate with one another directly using Java's RMI technology. Therefore, they do not need to use a JavaSpace as an intermediary.

On the other hand, it is still useful to retain the Drone concept. For our Hotel System have "Plug-and-Play" convenience, it is desirable that all computing devices only need to run the Hatchery Drone, except the central server which will contain the rest of the code base. There may also be exceptional cases where an Agent may need access to a JavaSpace.

To marry the two solutions, we have written an abstract class AgentDrone that extends Drone. Therefore, it has all the functionality that a Drone would have. Namely, (1) it can

find multiple JavaSpaces through Multicast messages to multiple Jini Lookup Services; (2) it can maintain that list of multiple JavaSpaces by removing duplicate and bad JavaSpaces and by adding new JavaSpaces as they become available; (3) it can be transported, usually from a Queen, to any Hatchery, and run as a new process on the remote host; (4) it can be killed either locally or remotely, usually by the user or a Queen; (5) it can write Drone Entries to JavaSpaces, or be notified about new Drone Entries appearing in any JavaSpace; (6) it can manage its leases with all JavaSpaces on Drone Entries that it has written or on notification on Events that it has subscribed to.

Moreover, Agent Drone has additional functionality that exposes an embedded Agent (that implements Agent Interface) to the rest of the world. That is, each Agent Drone serves to house, announce and transport an Agent or its services.

For each Jini Lookup Service found, Agent Drone will spawn a new thread to register its internal Agent Service to the lookup server, with appropriate Service Attributes so that others can easily find their desired Agent. This is done in the standard fashion dictated by the Jini architecture, as show in Figure 5.1.

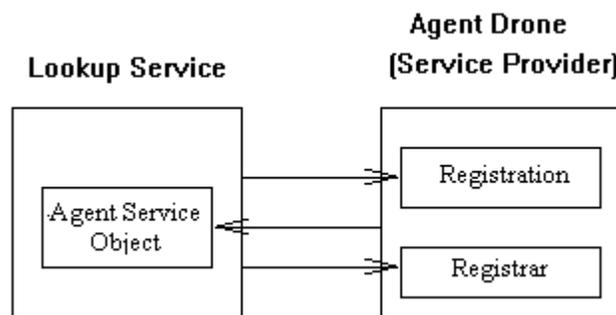


Figure 5.1. Pictorial showing Object passing  
when an Agent Drone registers at a Lookup Service

The Agent Service Object is the proxy for the Agent at remote hosts. It can either be fat, with all the code in the proxy, so that all computation is effectively occurring at the remote client, or thin, using RMI for active communication, so that all computation occurs at the Agent Drone. It can distribute computation between the client and the server. This choice is left open, and specific Agents should choose the architecture that is best suited for the application.

## 5.4 Details on some specific Agent Drones

Now that we have gone through the properties of a general Agent Drone, it is time to explain the differences between each specialized Agent Drone.

### 5.4.1 Database Agent Drone

Although not yet implemented, we anticipate that a Database Drone to work closely with the Registration Drone to either retrieve information of former hotel guests in the hotel chain, or to save current information of the same for long term storage. The Database Drone also works closely with the Queen Drone in providing information on how to personalize particular Drones that may be requested by a certain hotel user.

## 5.4.2 Administrative Agent Drone

Administrative Agent Drones, or Admin Agent Drones for short, house Admin Agents. Admin Agents, as the name suggests, serves as a scheduling and note-taking assistant to a user. The users are every facility, employee, or guest.

When a hotel guest first checks in through the Registration Drone, the Queen is informed of the new guest, and spawns a new Administrative Agent Drone for that user (at an appropriate Hatchery). The Queen also monitors the presence of that Drone, and creates a new one in an event of a failure. During the checkout process, the Queen then kills the Administrative Agent Drone for that user.

We have mentioned that the Registration Drone treats a hotel guest checking in much like an employee signing in to work. Here, we extend the analogy. Every hotel employee has his or her own Administrative Agent, which is created in a similar fashion as a hotel guest.

Furthermore, every room door and every hotel facility will have an Administrative Agent Drone. Much of these Drones are automatically created during the initialization process, but the hotel manager can also manually create them. For example, the hotel manager may wish for hotel guests to schedule for activities outside the hotel such as Golf.

One function of the Administrative Agent is to schedule meetings between two entities. This always involves at least 2 Administrative Agents. In order to attempt different scheduling possibilities, the Administrative Agent can spawn multiple copies of itself, and send each copy to visit other Administrative Agents. In order to schedule a series of events, the Administrative Agent visits the different Administrative Agents in series. Thus, the Boolean “or” can be accomplished via the creation of daemon Administrative Agents, while the Boolean “and” can be accomplished via increasing the number of foreign Administrative Agent visits. Upon reaching an impossible situation, the daemon Administrative Agent can return to its immediate master to report failure. A better understanding of the scheduling algorithm can be obtained through the scenario in section 6.4.

This simple scheme allows for a multitude of applications. For example, a business traveler’s agent may visit each conference room’s agent and each restaurant’s agent to attempt to schedule a lunch appointment and a conference room reservation. As another example, a cleaning maid’s agent may attempt to schedule with the room’s agent to clean the room. As a last example, a vacationer’s agent can try to schedule a restaurant meal, a golf game and an acupuncture session all on the same day, but without any ordering preference.

Another function of the Administrative Agent is to remember the preferences and settings of the user. For example, a hotel guest or employee can set a contact list that will appear on the Messaging Manager by default. As another example, a hotel guest can set his or

her wake-up call preferences. In future implementations, the hotel guest will also be able to choose the picture to appear on the digital photo frame. The agent will also use the hotel guest's past preferences and purchases to personalize offerings such as room service and promotions.

The third function of the Administrative Agent is to take messages for the user. As one example, another person can leave a message for the hotel guest through the messaging system, even though the hotel guest is not online. As another example, the hotel guest can leave requests with the room's agent (to be passed on to the cleaning maid's agent when a cleaning session is scheduled).

### **5.4.3 Messaging Agent Drone**

The Messaging Agent Drone is yet another Agent Drone. It provides a graphical user interface for one user to send "Instant Messages" to another user.

In our first implementation of an Instant Messenger, messages were passed from one party to another party via Javaspaces. We noted high network latencies on a typical system.

This was not surprising since a Javaspaces may prove to be a bottleneck, and the message has to travel across an intermediate node.

Our discussion in section 3.5 has shown the use of agents more appropriate for message passing between known senders to known receivers. In light of this, we re-implemented the Instant Messenger service to be a sub-class of Agent Drone.

In fact, the Messaging Agent Drone does work very closely with other Messaging Agent Drones and Administrative Agent Drones. In particular, a Messaging Agent Drone will first seek the user's corresponding Administrative Agent to obtain his or her "Contact List". The graphical user interface will then display the list of friends and hotel staff, and will note if they are online or offline. In double-clicking a name, a new window will appear for a chat session between the user and the chosen target.

If the target is online, the Messaging Agent Drone will attempt to find the target's Messaging Agent, and the target's Messaging Agent Drone vice versa. Both Agents can then receive messages from each other in a life chat. With a direct connection between Messaging Agent Drones, the network latency is much better, and the JavaSpaces are not bogged down with additional requests and entries. Therefore, we have implemented two chat modes. In the Messaging Mode, the user can type and edit his or her sentences before clicking send. In the Chat Mode, the Messaging Agent Drone will periodically send the text in the textbox if there are any changes from the last send.

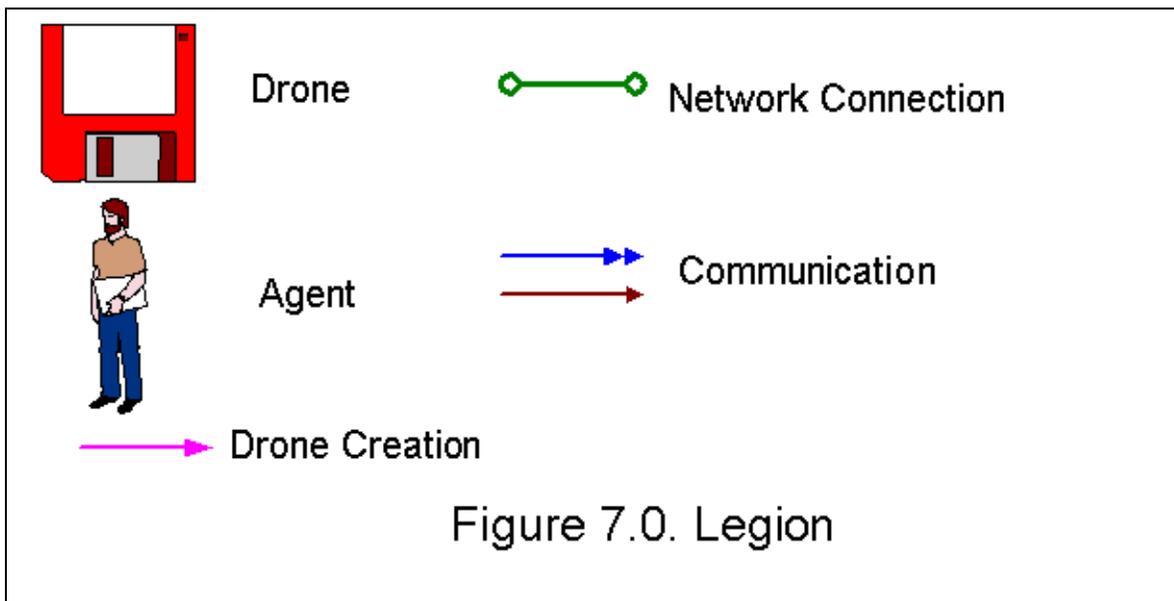
Unlike commercially available chat programs, such as ICQ [1.5] and AOL Instant Messenger [1.6], our implementation offers some advantages:

- i. No central Server
- ii. Dynamic Switch Between Messaging and Chat Modes
- iii. One sided-chats possible.

## 6 Specific Scenarios

Section 4 described the software and hardware architecture of the Hotel Management System and Section 5 entered into greater depth on the workings on each specific Drone. However, the division of the Hotel System via the different Drone processes may not give a complete understanding of how different Drones work together to provide services for hotel guests and employees. This section makes up for this shortfall by going through different scenarios that may arise in a typical hotel, thereby dividing the Hotel System via high-level services rather than low-level processes.

The legend for the rest of the figures in this section is provided in Figure 7.0.



## 6.1 Check-in

A detailed pictorial explaining the check-in process can be seen in Figure 7.1. The steps of the process occur in the sequence of the numbered circles. The steps are:

- i. The Registration Drone communicates with the Database Agent to retrieve information on previous bookings, room availability, or customer profiles. It also saves new information to the Database. Here, the hotel guest checks in and is assigned a room.
- ii. The Registration Drone writes a Registration Entry to JavaSpace, so that other Drones are aware that the hotel guest is currently living in the hotel.
- iii. A Queen is notified of the Registration Entry of the new guest.
- iv. The Queen retrieves stored settings and preferences of the guest, if any. Using the saved settings, the Queen creates a new Admin Agent Drone, which will provide administrative services to the guest while he or she is living in the hotel.
- v. The Drone is packaged in a Hatchery Entry, and written to JavaSpace.
- vi. An appropriate Hatchery had been selected based on its proximity to the guest room and its current workload. This Hatchery retrieves the Hatchery Entry.
- vii. The Hatchery spawns a new thread that runs the personalized Admin Agent Drone. From this point on, any process can contact the guest's Admin Agent to schedule activities, leave messages, or otherwise interact with the hotel guest.

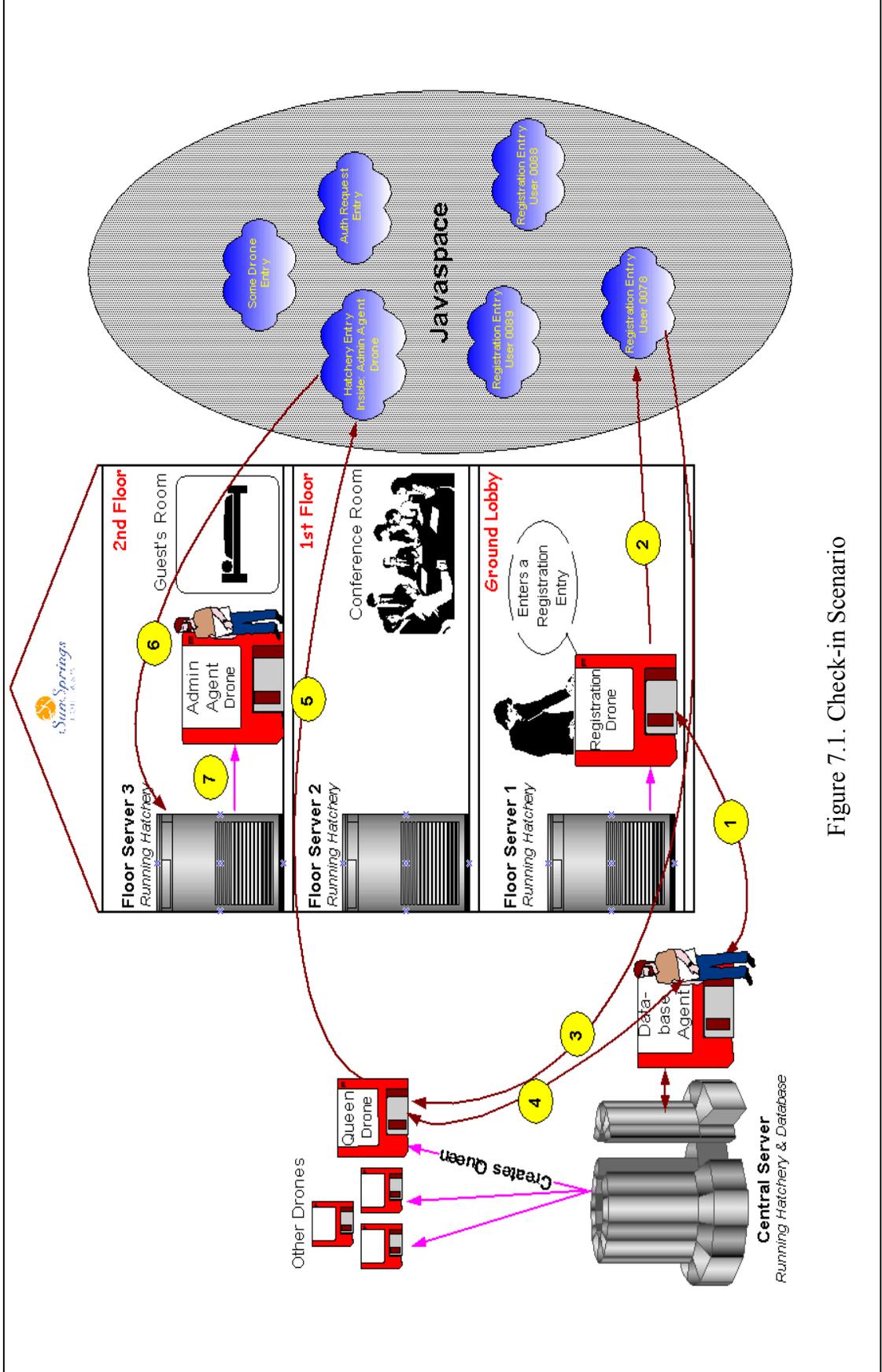


Figure 7.1. Check-in Scenario

## 6.2 Access-Control

A detailed pictorial explaining the access control process can be seen in Figure 7.2. The steps of the process occur in the sequence of the numbered circles. The steps are:

- i. Upon detecting a fingerprint, a Fingerprint Drone sends a One-To-Many Auth Request Entry without a fingerprint image to a JavaSpace. The Auth Request Entry identifies the Fingerprint Drone with information such as its location, process ID and server ID.
- ii. The Fingerprint Drone then writes a One-To-Many Auth Request Entry with a fingerprint image when it is read correctly.
- iii. A King takes the One-To-Many Auth Request Entry without an image, since it is smaller and available first.
- iv. The King then communicates directly with a Database Agent to discover the list of users that might have access to the resource.
- v. For each user in the list, the King writes a One-To-One Auth Request Entry to a preferred JavaSpace based on the location of the Fingerprint Drone. In this case, it is the “JavaSpace 2<sup>nd</sup> Floor”.
- vi. Auth Drones throughout the hotel take the One-To-One Auth Request Entry in a Transaction. The Auth Drones on the 2<sup>nd</sup> Floor prefer “JavaSpace 2<sup>nd</sup> Floor”, and are most likely to take such Entries before those on other floors.
- vii. Each Auth Drone also has to retrieve the fingerprint image, if it does not have it cached.

- viii. Each Auth Drone also has to retrieve the fingerprint template, if it does not have it cached. Again, since these Auth Drones work on similar cases on their preferred JavaSpace, the chances of a cache hit is high.
- ix. The Auth Drones write their results back to a JavaSpace, and commit the transaction. If the process has taken too long, the transaction will be aborted freeing the Auth Request Entry for another Auth Drone.
- x. The King reads the results, and waits for either every Auth Reply Entry to reject the fingerprint or the first Auth Reply Entry to accept the fingerprint. If the King waits too long, it will resend the Auth Request Entries that have not been replied to.
- xi. The King sends the result to the Fingerprint Drone. The result in this case is “Accepted”.
- xii. The Fingerprint Drone activates the TINI relay, and unlocks the guest room door.

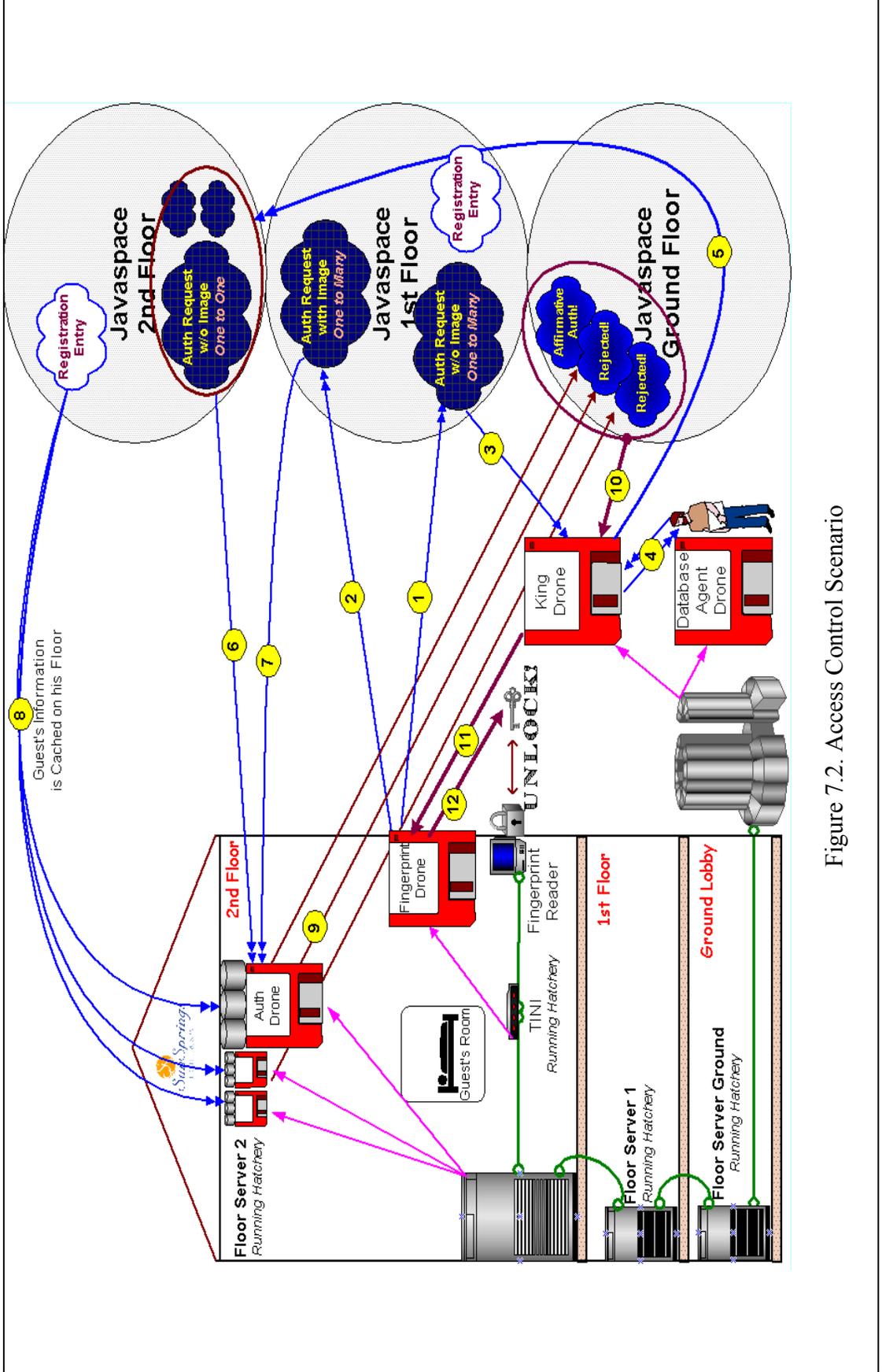


Figure 7.2. Access Control Scenario

### 6.3 Logging-in

A detailed pictorial explaining the logging-in process can be seen in Figure 7.3. The steps of the process occur in the sequence of the numbered circles. The steps are:

- i. The King writes an “Accepted” Auth Reply Entry to a JavaSpace, in response to a User Drone’s identification request.
- ii. The User Drone takes the Auth Reply Entry, which contains information of the user that has been identified. It then displays a selection of software that the user might use. In this case, the user chooses “Instant Messaging”.
- iii. The User Drone writes a Queen Request Entry for a Messaging Agent Drone.
- iv. The Queen replies with a Hatchery Entry with a customized Messaging Agent Drone for the User Drone. The User Drone is a subclass of Hatchery.
- v. The User Drone creates a new Messaging Agent Drone.
- vi. The Messaging Agent Drone contacts the corresponding Admin Agent for the user. (Recall this has been created during the check-in process.) The Admin Agent reveals the user’s contact list.
- vii. The Messaging Agent Drone displays the user’s contact list on a menu. Upon double-clicking a friend’s name, the message windows appears on both of the corresponding computers, as the two Messaging Agents communicate text messages to one another. There is a choice of Message Mode and Chat Mode.

- viii. If the foreign Message Agent cannot be contacted, a message is left in the corresponding Admin Agent Drone.

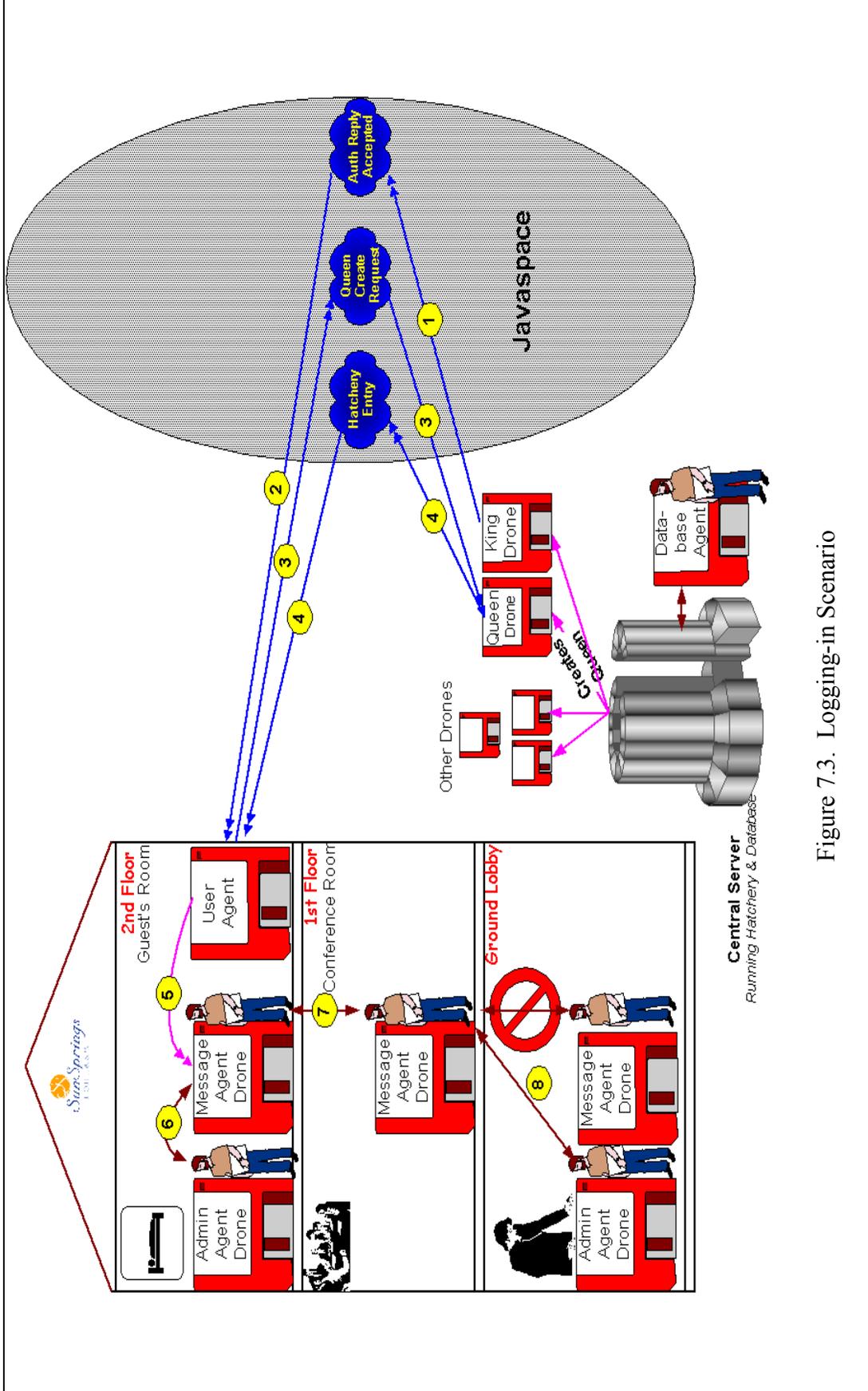


Figure 7.3. Logging-in Scenario

## 6.4 Scheduling Activities

A detailed pictorial explaining the scheduling process can be seen in Figure 7.4. The steps of the process occur in the sequence of the numbered circles. The steps are:

- i. A hotel guest or employee logs in at the User Agent as described in section 6.3. For this scenario, we will assume that the user is a hotel guest who is free from 12:00pm to 1:00pm. He or she plans to spend half-an-hour in a conference, and the other half-an-hour at lunch, but does not have a preference for the ordering. He or she chooses to leave unspecified the type of conference room or the name of restaurant. Thus, the hotel guest launches the Schedule Drone.
- ii. The hotel guest makes his or her intentions clear to the Schedule Drone. The Schedule Drone then contacts the guest's Admin Agent with the orders.
- iii. Since there are three conference rooms, the guest's Admin Agent makes 3 copies of itself, and sends them to the Admin Agent of each conference room. We shall call these copies, "Level-One Agents", and the original Admin Agent, "Master Agent".
- iv. A Level-One Agent has failed to negotiate a possible schedule with a conference room's Admin Agent. The conference room is not available at any time from 12:00pm to 1:00pm. This Level-One Agent returns to the Master Agent reporting failure. Since there are two more Level-One Agents that have not reported, the Master Agent does not report a failure.

- v. Each Level-One Agent remaining realizes that the user wishes to schedule a conference AND lunch. They therefore have to visit another Admin Agent, this time that of a restaurant. However, there are 2 restaurants in the hotel. Since either one will do, they have to make two copies of themselves, or Level-Two Agents, and send the Level-Two Agents to each restaurant's Admin Agent.

One of the Level-One Agents has tentatively scheduled a conference room for 12:00pm. It therefore sends Level-Two Agents that are aware of this. None of the restaurants are available at 12:30pm. Thus, both Level-Two Agents return to this Level-One Agent reporting failure.

- vi. Since all the Level-Two Agents reported failure, this Level-One Agent reports failure to the Master Agent. However, the Master Agent still has 1 Level-One Agent that has not reported failure. Therefore, it does not report a failure to the user.
- vii. The last Level-One Agent has tentatively scheduled a conference room for 12:30pm. It therefore sends Level-Two Agents that are aware of this. Both of the restaurants are available at 12:00pm. Thus, both Level-Two Agents tentatively mark 12:00pm on their schedules for the corresponding restaurants.
- viii. Both Level-Two Agents realize that their task is done, since there are no more events that have to be scheduled. They return directly to the Master Agent. The Master Agent shows the two available options to the hotel guest. The guest must take the third conference room at 12:30pm, but can choose either restaurant at 12:00pm.

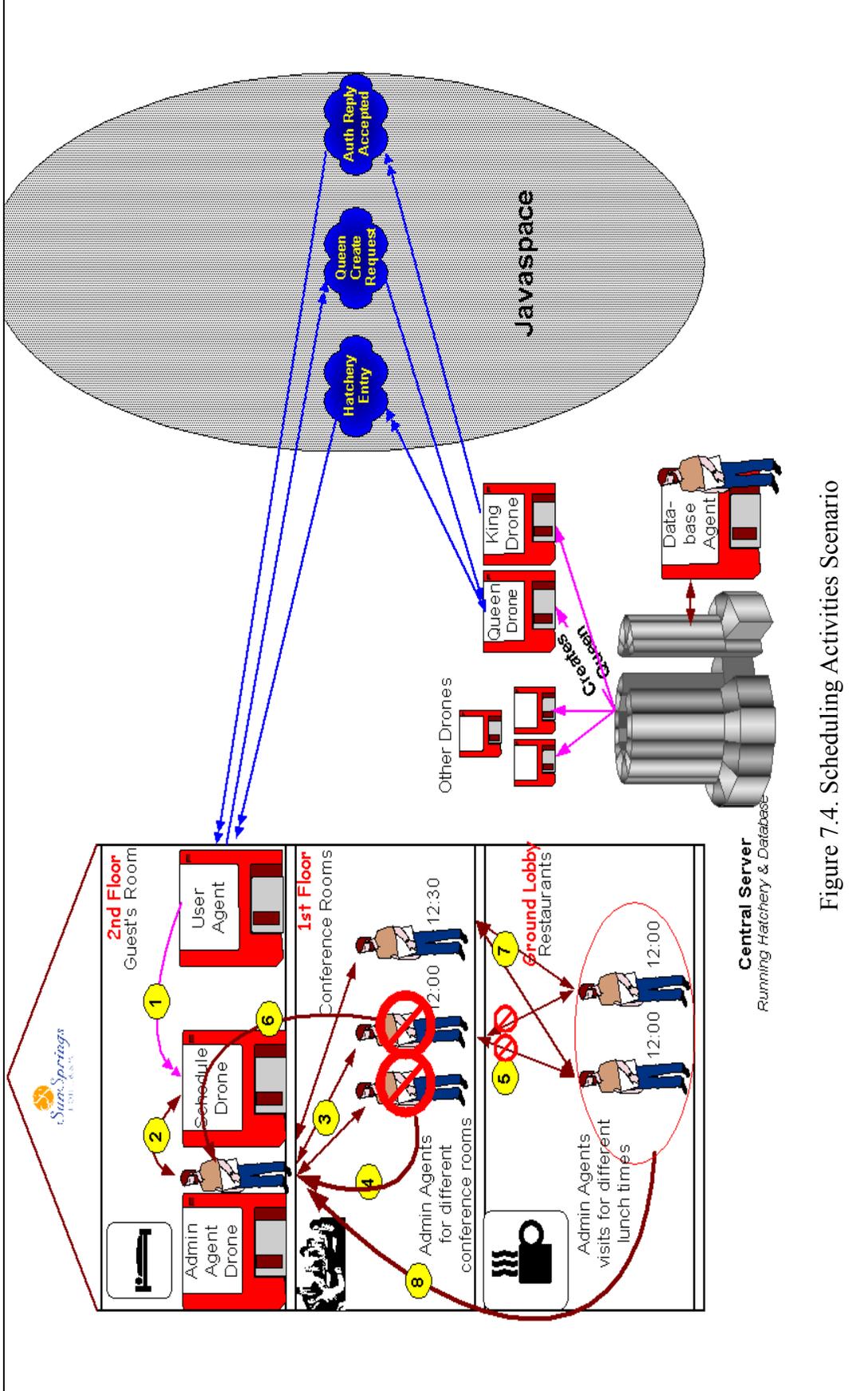


Figure 7.4. Scheduling Activities Scenario

## 7 Evaluation

### 7.1 Fulfillment of design goals

We had listed ten design goals before the implementation of the hotel system in section 4.1. Here, we will discuss how well we fulfilled our goals.

#### **7.1.1 Provide comprehensive services.**

Our implementation focuses on a small set of services for the proof of concept. Namely, we implemented Registration, Access Control, Instant Messaging, Scheduling and Billing. This is fairly good for the first proof-of-concept, but we hope that there will be many more services to come in the future. This includes an interactive Web Page that accepts reservations, a yield management algorithm that predicts and reacts to demand and supply conditions, employee tracking and payroll software, automated wake-up call software, and an algorithm to display advertisements to guests in an intelligent fashion.

#### **7.1.2 Provide superior services.**

We believe that some of the services implemented can provide high utility to hotel guest and employees. For example, the Biometrics Access Control System and the Instant Messaging System perform very well, and are preferable to their traditional counterparts. We do admit that more work needs to be done in other areas. For example, the current

Scheduling Drone can only schedule events in half-an-hour intervals. As another example, the Billing Drone needs to categorize items on a billing in a clearer fashion.

### **7.1.3 Computational Efficiency.**

We feel that our system is computationally efficient, since much of our time is spent optimizing the system. We have used two methods for parallel processing, and can therefore choose appropriate methods for different scenarios. We have used caching extensively, and thus perform certain tricks to increase cache hits. We have avoided bottlenecks through redundancy and planning.

However, some work needs to be done with the Scheduling Algorithm to limit the number of possibilities. With many unspecified variables, the number of Admin Agent clones can explode exponentially. Initially, we had wanted to explore all possible options. However, we now realize that it is only practical to employ heuristics to minimize the number of possibilities to only those that the user might be interested in. Furthermore, it may not be worthwhile to pursue unlikely situations, such as a step that decreases the most degrees of freedom.

### **7.1.4 Easy to install**

The system is plug-and-play enabled by inheritance from the Jini architecture. However, one overhanging problem is to efficiently identify each device with the actual physical

location in the hotel. This may be more of a problem with the actual hotel, rather than a computer science problem.

### **7.1.5 Easy to maintain**

The system has been tested extensively, with the forced failures of certain Drones, Agents or Jini Services. As long as there are redundant processes, the system does not fail. Exceptions are noted, and alternatives are sought.

The system is also left running for an extended period of time. Due to the transient nature of lease-based Entries and Event Notifications, garbage does not accumulate at any resource over time.

Therefore, our system is very easy to maintain, again from inheritance of the Jini Architecture.

### **7.1.6 Easy to upgrade and backup.**

We envisioned it to be easy to upgrade and backup the system, since the codebase can be running on one computer. While this is true, we have encountered failures in certain Drones when the codebase is changed while they are running. Queens, for example, may be unable to create Drones that are modified in new codebase. While it is much simpler to upgrade the system than a comparable Hotel Management Systems, we still had to remain vigilant during the replacement of the codebase for failures that may occur in

Drones that assume that the codebase is not modified from start time. New versions of those Drones are created immediately to replace the old versions to prevent the stoppage of service.

### **7.1.7 Affordable.**

We have spent a few hundred dollars on this project, but we have not bought all of the required parts. There is a long waiting list for the TINI chipset, which sells as much as \$50 if available. This is a few times greater than the predicted price of \$15 in time to come. We experienced a similar situation with some fingerprint readers and their SDK. We purchased a PreciseBiometrics Fingerprint SDK for \$400, which is far higher than the \$50 we expected to pay. However, we hope that these prices will drop with the bulk orders required in an actual hotel, and with time.

A big cost in our hotel implementation is the need for a desktop computer in every room. With a hundred rooms, the cost of this can be considerable. There is little we can do about this, except to possibly remove the numerous desktop computers at the expense of the service. Perhaps a limited number of wireless notebooks that are rented out for a fee can replace these desktop computers.

### **7.1.8 Customizable.**

In the design of the hotel system, tasks are broken down into small components and performed by Drones. These Drones mostly communicate via a common JavaSpace.

Therefore, it is not difficult to customize our system for many purposes. For example, hotels that require biometrics access control but not Instant Messaging can choose to run the Auth Drones, but not the Messaging Agent Drone.

### **7.1.9 Scalable.**

Scalability has been built into our system design. There is a distributed federation of Jini Services and Drones that provides no boundaries in size. In the potential confusion of this mass of processes, we have also implemented a partial structure by allowing Drones to prefer certain Services, such as the JavaSpace running on the same floor. This ensures lower network latency and higher cache hits, and yet retains the redundancy of the Jini Federation.

One improvement we will like to make is for the Queen to perform a more in depth analysis of the performance of each server or Drone, thus allowing for better load balancing between servers. This will, in turn, increase scalability with the higher assurance that bottlenecks are absent.

### **7.1.10 Web-enabled.**

By being Jini-enabled, our solution is web-enabled. We have tested our system with many other Jini Networks that we manage to find on the World Wide Web. Therefore, we have utilized JavaSpaces that are miles away from our actual location. This is promising since it breaks down the boundaries of a hotel. Two different physical locations can

function effectively as one Hotel or one Hotel Chain, and share the same database, marketing tools, payment gateways, or even computational resources.

This are, however, some factors that prevents the system from being fully Web-enabled. As mentioned before, we have yet to implement an interactive Web Page for hotel customers to reserve a room. In addition, we have not looked into how our systems can interface with non-Jini systems, such as Microsoft Hailstorm or XML technologies.

## 8 Conclusion

There are many areas where the proposed Hotel Management System can be improved upon. We will outline a few ways of extending the capabilities in this section.

### 8.1 Future Work

#### 8.1.1 Other Distributed Object Systems

Jini is but one method of federating services together in a boundaryless environment.

There are some other tools that have been developed to network web services, and it is worth considering a combination of these tools so that the Hotel Management System can be compatible with other third party services and devices.

##### *8.1.1.1 Corba*

One upcoming standard is Corba. It is language-independent, using an Interface Definition Language (IDL) for specification of interfaces for a distributed object system.

An example of an interface for room booking written in IDL is given in the appendix in Section 10.5.1.

Corba has bindings to a number of languages, the most recent of which is Java. Further work needs to be done so that the Hotel Management System can interact with other third party services that support Corba.

#### *8.1.1.2 XML, Microsoft.NET & Hailstorm*

Yet another upcoming standard that clusters together distributed web services is that of Hailstorm. Hailstorm is a component of the .NET initiative by Microsoft, and promises to be an open web services architecture based upon the de facto standard dynamic duo of XML and SOAP.

The motivation of Hailstorm is the numerous web services that require users to provide disparate bits of information. Microsoft's central Passport service integrates the cluster of personal information, and allows the user to use a single key to access a multitude of services. An interesting development for future work will therefore be for the Hotel Management System to interface seamlessly with Microsoft Hailstorm to retrieve the customer's information such as address, credit card number, schedule and contact list.

#### **8.1.2 Better way to scale JavaSpaces**

Another interesting project for future work will be to design a better way to scale JavaSpaces. Currently, we used a simple scheme of rotating between JavaSpaces starting from the preferred one. There can be better implementations to support multiple Javaspaces more naturally. For example, there could be a registry that points directly to

the JavaSpace that contains the Entry a program is looking for. Another area of improvement will be to decrease the space and time requirements for writing entries to a JavaSpace. IBM has already done good work in this field in their implementation of T-Spaces [6.3].

### **8.1.3 Integration with other systems**

To be commercially viable, the Hotel System we propose must be integrated with other systems such as credit card processing, employee payroll and accounting systems.

## **8.2 Summary**

We have presented the design and prototype implementation of a low-cost Hotel Management System. In this system, computing devices, such as servers, desktops and TINI chipsets, are distributed throughout the hotel and connected via Ethernet. Queens are processes on central servers dynamically create or destroy Drone processes on each computing device as required. Our system uses the Jini architecture and JavaSpace extensively, and is therefore dynamically distributed. Redundancy ensures that the system is robust. There are multiple copies of Drones, Agents and JavaSpace running at any one time. Furthermore, Jini Leases ensure that the system is self-healing. If a process dies unexpectedly, its leases with JavaSpaces, Lookup Services and other Jini Services will expire in due time.

Drones communicate via JavaSpace, which is a naturally method of breaking down some tasks for parallel processing. For example, fingerprint identification can be performed quickly by letting Auth Drones match the fingerprint to different templates. This is done in an adaptively parallel fashion, since Auth Drones can withdraw at anytime without affecting the outcome of the identification process. Transactions ensure that an Auth Drone will never lock a request forever.

Agents, on the other hand, communicate directly through the Jini architecture. This is appropriate when communication occurs with a specific target. A good place this is used is in Scheduling. Admin Agents can negotiate schedules with each other without the burden of going through JavaSpace. Furthermore, an Agent can actually transport itself to run on the foreign host, so that bandwidth is conserved for more complex scheduling requests. An Agent can also make multiple copies of itself, when attempting to schedule multiple events for multiple parties. This allows many possibilities to be attempted at the same time.

Our prototype has served well as a “proof-of-concept”. Obviously, more works need to be done. However, we are optimistic that this can become a practical system for Hotel Management.

## 9 References

### 9.1 General:

[1.1]: Data source: 1999 CIA World Factbook

[1.2]: SunSprings Hotel and Spa. *Business Plan*. 2001.

[1.3]: ETrue, Inc. <http://www.etrue.com>

[1.4]: TigerDirect. <http://www.tigerdirect.com>

[1.5]: ICQ. <http://www.icq.com>

[1.6]: AOL Instant Messenger. <http://aim.aol.com/aimnew/oldreg/home.html>

[1.7]: Hot Java Component. <http://www.sun.com/software/htmlcomponent/index.html>

[1.8]: TINI Chipset. <http://www.ibutton.com>

### 9.2 Commercial Hotel Management Systems:

[2.1]: InnSystems. [http://www.innsystems.net/products/products\\_interfaces.html](http://www.innsystems.net/products/products_interfaces.html)

[2.2]: GuestLine. <http://www.guestline.com/contsol.html>

[2.3]: Digital Rez Software Corp. [http://www.digitalrez.com/MO\\_HO/MotelMulti\\_1.asp](http://www.digitalrez.com/MO_HO/MotelMulti_1.asp)

[2.4]: Reservations Plus, Inc. <http://www.aps-rplus.com/demo.htm>

[2.5]: Inn Scheduler. <http://www.innscheduler.com/>

[2.6]: InnFone. <http://www.keysystemus.com/InnFone.html>

[2.7]: TesaLocks <http://www.tesalocks.com/>

### 9.3 Authentication and Identification

[3.1] E. Newham, *The Biometric Report*, SJB Services, New York, 1995.

[3.2] R. Clarke, Human Identification in Information Systems: Management Challenges and Public Policy Issues, *Information Technology & People*, Vol. 7, No. 4, pp. 6-37, 1994.

[3.3] S. G. Davies, Touching Big Brother: How Biometric Technology Will Fuse Flesh and Machine, *Information Technology & People*, Vol. 7, No. 4, pp. 60-69, 1994.

[3.4] The BioAPI Consortium. *BioAPI Specification 1.1*. The BioAPI Consortium, March 2001.

[3.5] A. Jain, L. Hong, S. Pankanti, R. Bolle. *An Identity Authentication System Using Fingerprints*. Proceedings of the IEEE, Vol. 85, No. 9, pp. 1365-1388, 1997.

[3.6] N. Ratha, J. Connell, R. Bolle. *A biometrics-based secure authentication system*. Proc. 1999 IEEE Workshop on Automatic Identification Advanced Technologies (WAIAT-99), Morristown NJ, October 1999.

[3.7] N. Ratha, J. Connell, R. Bolle. *Cancelable Biometrics*. 2000 Biometrics Consortium Workshop, September 2000.

[3.8] Anil Jain and Sharath Pankanti. *Automated Fingerprint Identification and Imaging Systems*. IBM TJW Research Center.

## 9.4 Yield Management

[4.1] Garrett J. Van Ryzin, Jeffrey I. McGill. *Revenue Management: Research Overview and Prospects*. Transportation Science 0041-1655 Vol. 33, No. 2, May 1999

[4.2] Timothy Kevin Baker. *New Approaches to Yield Management: Comprehensive Overbooking/Allocation Heuristics for the Hotel Industry*. PhD thesis, The Ohio State University, 1994.

## 9.5 Networked Systems

[5.1] Danny Cohen, "On Holy Wars and a Plea for Peace", IEEE Computer, October 1981.

[5.2] Sun Microsystems, "XDR: External Data Representation Standard", [RFC-1014](#), June 1987.

[5.3] Birrell, A. D. & Nelson, B. J., "Implementing Remote Procedure Calls", XEROX CSL-83-7, October 1983.

[5.4] Sun Microsystems, "RPC: Remote Procedure Call", RFC-1050, June 1988.

[5.5] Polyhedra, Inc., *Some Background Concepts: Object-Oriented Systems*.

[5.6] Letha H. Etzkorn and Carl G. Davis. *An Approach to Object-oriented Program Understanding*. Technical Report TR-UAH-CS-1995-01, Univ. Alabama, 1995.

[5.7] Steve Vinoski, Distributed Object Computing With CORBA, C++ Report magazine, August 1993

- [5.8] Chung, Huang, Yajnik, Liang, Shih, Wang, Wang. *DCOM and CORBA Side by Side, Step by Step, and Layer by Layer*.
- [5.9] Object Management Group (OMG). *CORBA Scripting Language Specification*. Version 1.0, June 2001
- [5.10] W. Keith Edwards. *Core Jini, 2<sup>nd</sup> Edition*. Prentice Hall, 2001.
- [5.11] Sun Microsystems. *Jini™ Architecture Specification*. Version 1.1. October 2000.
- [5.12] Robert E. McGrath. *Discovery and Its Discontents: Discovery Protocols for Ubiquitous Computing*. Presentation at Center for Excellence in Space Data and Information Science. NASA Goddard Space Flight Center. April 5, 2000
- [5.13] ChoonHwa Lee, Sumi Helal. *Protocols for Service Discovery in Dynamic and Mobile Networks*. Computer and Information Science and Engineering Dept. University of Florida, Nov 2000.
- [5.14] Christian Bettstetter, Christoph Renner. *A COMPARISON OF SERVICE DISCOVERY PROTOCOLS AND IMPLEMENTATION OF THE SERVICE LOCATION PROTOCOL*. Technische Universität München (TUM), Institute of Communication Networks, D-80290 Munich, Germany. 2000.
- [5.15] William Adjie-Winoto, Elliot Schwartz, Hari Balakrishnan, Jeremy Lilley *The design and implementation of an intentional naming system*. Proc. 17th SOSP, Kiawah Island, SC.
- [5.16] Moss, Elliot, *Nested Transactions : An Approach to Reliable Distributed Computing*, The MIT Press, Cambridge, Massachusetts, 1985, pp.31-38.
- [5.17] Universal Plug and Play. <http://upnp.org/>

## 9.6 Distributed Systems

- [6.1] D. Gelernter. *Generative communication in Linda*. ACM Transactions on Programming Languages and Systems, 7(1):80-112, 1985.
- [6.2] Larsen, Spring. *Global Object Exchange: A dynamically fault-tolerant and dynamically-scalable distributed tuplespace for heterogeneous, loosely coupled networks*. Thesis, University of Copenhagen, October 1, 1999.
- [6.3] Wyckoff, McLaughry, Lehman, Ford. *T Spaces*. IBM Systems Journal, August 1998
- [6.4] Sun Microsystems. *JavaSpaces Specification*. Revision 1.0 Beta, July 17 1998
- [6.5] Eric Freeman *Build a compute server with JavaSpaces*.  
<http://www.javaworld.com/javaworld/jw-01-2000/jw-01-jiniology.html>
- [6.6] Carriero, Gelernter, Kaminsky, Westbrook. *Adaptive Parallelism with Piranha*. Technical Report 954, Yale University Department of Computer Science, Feb. 1993.
- [6.7] Noble, Zlateva. *Distributed Scientific Computation With JavaSpaces?* Technical Report, MET College Department of Computer Science, Boston University, 2001.
- [6.8] Tripathi, Ahmed, Karnik. *Experiences and Future Challenges in Mobile Agent Programming*. Microsystems, October 2000.
- [6.9] Ronald Ashri and Michael Luck. *Paradigma: Agent Implementation through Jini*. Department of Electronics and Computer Science. University of Southampton. 2000
- [6.10] Singh. *Development of Distributed Systems using Mobile Agents in Ajanta*. Thesis Defence, University of Minnesota, 1999.
- [6.11] Steven Schmelzling. *Distributed Processing: A Synopsis*. CS-384 Design of Operating Systems Research Paper, January 2000.

