# An Intelligent IP-based Call Center with Fault Tolerance Design

**Leung Cheung-chi**

**A Thesis Submitted in Partial Fulfilment**
**of the Requirements for the Degree of**
**Master of Philosophy**
**in**
**Computer Science & Engineering**

**Supervised by:**
**Prof. Y.S. Moon**


**June 2001**

論文題目： 備有容錯設計的IP作基礎之智能呼叫中心

作者：梁祥智

學院：工程學院

學部：計算機科學與工程

修讀學位：哲學碩士

摘要

利用話音在互聯網協定（ＶｏＩＰ）的技術，我們建議了一個用來作線上客戶服務的ＩＰ作基礎之智能呼叫中心模型。爲了令這個應用模型適合於不同行業上的客戶服務，我們將介紹一個三層式的軟件結構用來作模型的實施。這個三層式的設計加強建議模型中各元件的再用性及縮短令這建議模型適應於某一行業的開發時間。

自動來電分配器（Ａｕｔｏｍａｔｉｃ　Ｃａｌｌ　Ｄｉｓｔｒｉｂｕｔｏｒ）是這個建議模型中的主要元件。它運作於三層式的軟件結構之中層。它的功能是選擇適當的服務員去接聽來電及執行來電管理。自動來電分配器依照顧客的資料、服務員的資料、所有現存來電的狀態和過去來電記錄去執行這些這些工作。 我們對來電管理內有的功能及來電分配策略將會作詳細討論。

令自動來電分配器有一個好的界面與其他元件接合及縮短其開發時間，我們考察了兩個著名的伺服器端元件架構，它們分別是Ｅｎｔｅｒｐｒｉｓｅ　ＪａｖａＢｅａｎｓ（ＥＪＢ）及ＣＯＭ＋。因爲ＥＪＢ模型較爲適合自動來電分配器的需要，所以我們在建議模型的實施時會利用到ＥＪＢ模型。

因爲高可用性是呼叫中心的基本條件，所以我們需要考慮到容錯（ｆａｕｌｔ ｔｏｌｅｒａｎｃｅ）這個問題。這篇論文會討論到怎樣去設計軟件架構和怎樣利用ＥＪＢ模型所提供的功能去解決在建議模型中各元件可能的失效問題。我們會提到其他在利用ＥＪＢ模型去實施自動來電分配器時遇到的技術要求。

有了實施的設計才是完整的設計。我們利用討論過的技術去實施了一個實驗的自動來電分配器。我們還會觀察一下這個實施的結果。最後我們會對完成的工作作出總結。

# An Intelligent IP-based Call Center with Fault Tolerance Design

submitted by

**Leung Cheung-chi**

for the degree of Master of Philosophy
at the Chinese University of Hong Kong

## Abstract

Based mainly on VoIP techniques, an Intelligent IP-based Call Center Model is proposed for online help desk services. To make the application model suitable for customer services in different industries, a three-layer software structure is introduced for its implementation. This three-layer design enhances the reusability of components in the proposed model and shortens the development time for adapting the application model to a specific industry domain.

The core component of the proposed model is an Automatic Call Distributor (ACD). The ACD operates in the middle layer of the three-layer software structure. The tasks of the ACD are to choose a suitable operator to answer an incoming call and to perform the call management. The ACD performs these tasks based on customers' profile, operators' information, states of current calls and past call records. The functionality in call management and the routing mechanism are discussed in detail.

To provide the ACD with a well-defined interface with other components and shorten its development time, two prominent server-side component architectures, Enterprise JavaBeans (EJB) and COM+, are examined. As the EJB model suits the ACD's needs more, the EJB model is deployed in the implementation.

Since high availability is essential for a call center, the issue of fault tolerance is considered. This thesis discusses how the software architecture is designed and features provided in the EJB model are used to tackle the possible failure problems of the key components, namely the VoIP gateways and the ACD, in the proposed model. Other technical requirements for the implementation of the ACD using the EJB model are figured out.

No design is really complete without implementation. We implemented an experimental ACD using the discussed technique and made observations of the implementation results. Finally, we complete this thesis with concluding remarks on our completed works.

# Acknowledgment

I would like to thank my supervisor, Prof. Y.S. Moon, who entrusts me to work on this research topic and supports me with much guidance and references along the research period.

Moreover, two classmates, Mr. H C. Ho and Mr. K.N. Yuen, who have worked closely with me for these two years and I thank for their research collaboration and useful suggestions.

At last, I would like to thank my parents, who support me continuously during these two years.

# Table of Contents

# Table of Figures

# 1 Introduction

## 1.1 Background

Traditionally, telephone line network is used to carry analog voice signal between telephones while data network is used to carry digital data between computers. With the advancement of the CPU computation power, voice-coding technologies and computer network technologies, voice can be converted into digital format for transport over IP based networks. There are several developing standards that aid the transmission of voice over IP. For example, H.323 [1] which is targeted to the transmission of multimedia over packet-based network; Session Initiation Protocol (SIP) [2] which aims at the creation of sessions between different parties; Real-Time Transport Protocol (RTP) [3] which transmits real-time packet over network. These factors make the computer networking infrastructure changing towards integrated service networks capable of simultaneously supporting real time conversational services and data services.

Voice over IP techniques can be deployed in many areas, such as international calling, telemarketing, education and customer support. In this thesis, an Intelligent IP-based Call Center Model is proposed for customer support services. In the proposed model, VoIP gateways connect a Public Switched Telephone Network (PSTN) to the Intranet of the call center. Each operator in the call center works with a multimedia PC which provides telephone services and relevant information for the incoming call. When an incoming telephone call comes into a gateway, the gateway routes the call to an Automatic Call Distributor (ACD) [4],

which finds the most appropriate operator to serve the incoming call.

In a traditional call center, an analog and proprietary-designed private branch exchange (PBX) which allows all operators in the call center to share a certain number of external phone lines is used. However, most PBX-based call center designs are limited to choose a low capacity solution that cannot grow beyond a few dozen operators, or a high capacity solution that is cost prohibitive for less than a hundred or more operators. Moreover, the telephone calls and relevant information about the calls are separated. This often leads to problems that make quality customer service incomplete or difficult.

Regarding these limitations, IP based call center designs emerged. The IP network and its inherent ability to connect data devices via packet switching replace analog and proprietary-designed PBXs, so the telephone call and the call information can be linked easily. This linkage makes call handling and customer service a simpler task. The distribution of incoming calls is handled by an Automatic Call Distributor, which is a software implementation utilizing industry standard computer hardware. This enables the system to start small and expand with an almost completely linear cost structure along the growth curve.

## 1.2 Objective

Based mainly on the VoIP techniques, a Customer Relationship Management (CRM) [5] [6] application model, Intelligent IP-based Call Center Model, is proposed for customer support services. Consider the following scenario. A customer

who requests customer support services phones to a call center. According to the caller ID of this new incoming call, the call center tries to search for the identity of this customer. According to the identity and the request of the customer, a suitable operator is chosen to answer this call. When this operator answers this call, the profile of this customer pops up on the operator's workstation. If the customer accesses a webpage and his/her questions are related to this webpage, the web-surfing status of the customer can be transferred to the operator when necessary. This feature is called collaborative browsing [7]. In this way, the customer can receive personalized service.

In this application model, data packets from the Internet and analogue voice signal from PSTN can enter the call center system through a digital voice gateway to replace a traditional analogue Private Branch Exchange (PBX). In such a way, we have a voice and data unified environment [8]. In this unified environment, value-added services such as collaborative browsing and comprehensive profile of customers given to operators facilitate the call center to know ahead enough information about the customers in order to provide them with the right services/products at the right time.

## 1.3 Overview of the Thesis

Chapter 2 of this thesis discusses the application of VoIP in call center. This section first shows a proposed Intelligent IP-based Call Center Model. To test the feasibility of implementing a call center system with VoIP technologies, a prototype of IP-to-IP call center which is used for an education environment is implemented

and mentioned in this section.

To make the proposed model suitable for customer services in different industries, a specific software structure is adopted for the application model. Chapter 3 of this thesis discusses the software structure of the proposed model. The ACD, the key component of the application model, operates in the call management layer of the software structure. This call management layer provides call management such as establishing calls, maintaining call states and supporting advanced telephony features and fault tolerance. Chapter 3 also describes the functional overview of the ACD. It includes how the ACD establishes a call, provides call-waiting and call-forwarding features and the routing mechanism in the ACD

To provide the ACD with a well-defined interface with other components and shorten its development time, two prominent server-side component architectures, Enterprise JavaBeans (EJB) and COM+, are examined in Chapter 4. This chapter shows how the EJB model suits the ACD's needs in the implementation more than the COM+ model does. High availability is essential for a call center, so the issue of fault tolerance is considered. When a functional component fails, this must be identified and reported, and a mechanism to overcome the failure must be provided. This arrangement allows the remaining functional components to continue to provide service. This chapter also discusses how the software architecture is designed and features provided in the EJB model are used to tackle the failures of key components in the proposed model. Other technical requirements in implementing the ACD are discussed. We also describe the way in which the ACD is implemented to fulfill these requirements using the EJB model.

Chapter 5 presents results and observations in implementing the proposed model. Chapter 6 concludes this thesis.

# 2 Application of VoIP in Call Center

This chapter firstly describes a proposed Intelligent IP-based Call Center Model for customer support services. In the proposed model, customers use traditional telephones to phone to the call center. However, operators in the call center are located in an IP network to handle calls. To test the feasibility of this proposed model, a simple call center system which is specifically used in education is implemented and mentioned in this chapter. In this simple system, both caller and callee are located in an IP network and no user is located in a Public Switched Telephone Network (PSTN).

## 2.1 An Intelligent IP-based Call Center Model

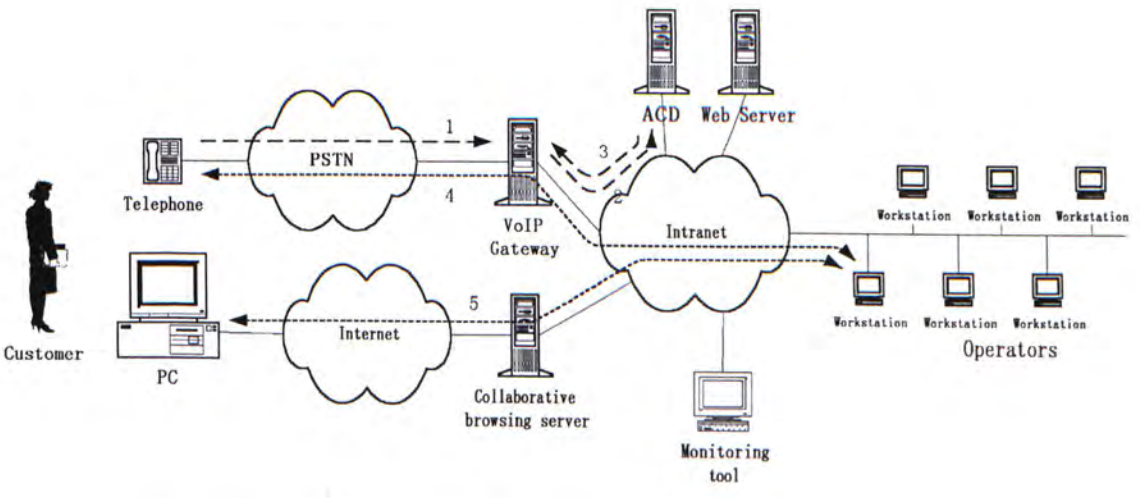



*Figure 2-1 The CRM application model*

The proposed Intelligent IP-based Call Center Model is depicted in Figure 2-1. A customer who requests customer support services uses a PSTN to communicate with an operator in the call center. The VoIP gateway connects the PSTN to the Intranet of the call center. The VoIP gateway digitizes, compresses, puts the voice of the customer from the PSTN into data packets and forwards the packets to the operator. When a new telephone call comes into the gateway, the gateway routes the call to the Automatic Call Distributor (ACD). The ACD finds the most appropriate operator to serve the incoming call. If there is no suitable operator to handle this call immediately, this call is put into a waiting queue temporarily.

### 2.1.1 Major Components

The proposed model consists of four major components, including VoIP gateways, an Automatic Call Distributor (ACD), operators and a monitoring tool.

#### a) VoIP Gateways

The PSTN carries voice as a 64-kbps stream using pulse-code-modulation encoding, while the Intranet of the call center carries voice using the Real-Time Transport Protocol (RTP) and various encoding schemes over a range of bit rates. VoIP gateways convert between voice formats in the PSTN and the Intranet. VoIP gateways enable phone connections to be made between users in the PSTN and the IP network.

When a customer phones to the call center, the call is received by a VoIP gateway. The VoIP gateway sends a call request to the ACD on behalf of a customer. The gateway then converts the telephone call to an IP session with the selected operator.

### b) Automatic Call Distributor (ACD)

The main task of the ACD is to choose a suitable operator to answer an incoming call. When a customer requests to talk to an operator, the ACD assigns an operator that served the customer before or assigns an operator that possesses certain skills to help the customer better. If there is no suitable operator to handle this call immediately, this call is put into a waiting queue temporarily. The ACD also stores customers' profile, operators' information, all current call states and past call records. The ACD processes the call routing mainly based on such information.

### c) Operators

When an operator starts his/her work, he/she logins an workstation. The system then updates the information about an operator who is ready to accept calls and the IP address of the operator's workstation. When an operator is assigned to answer a call, the profile of the caller/customer, if available, will appear on the operator's screen. If the operator is not able to answer the customer's questions, the operator can forward the call to his/her supervisor or forward the call to the ACD server with specified parameters (e.g. skills needed).

### d) Monitoring Tool

The system administrator is equipped with a monitoring tool. When a failure occurs in the system, the system administrator can identify it and then replace the failed component to recover the system. For example, if a gateway makes a request to the ACD and does not receive any reply from the ACD until the timeout, the gateway informs the monitoring tool that the ACD fails. The monitoring tool then checks whether this has been reported before and prompts this information to the screen. The system administrator can replace the failed gateway with a new one. Before the recovery, the system can still provide part of service as other gateways still work.

## 2.1.2 Major Functions

Three main functions, namely typical voice connection, call waiting and call forwarding, are provided in the proposed model.

The call forwarding feature lets an operator forward a call to another operator when the operator is not able to answer a customer's question in this call.

The call waiting feature lets an operator hold a current call and accept a new call. Operators probably do not receive a new incoming call when they are answering calls. The situation may occur only when an operator forwards a call to a specific operator who is answering another call at that time.

These three functions will be discussed in details in Section 3.3, Functional

## 2.2 Experimental Study of an IP-to-IP Call Center – VoIP Application in Education

Due to the popularity of Web, educational organizations use the web for self-learning purposes. They create course homepages which show reading lists, lecture notes and handouts, and provide some form of feedback and assessment in this courseware. It allows the individual to acquire new knowledge and skills at any time, at any location, at any pace. However, a major shortcoming of the Web-Lecture is the lack of interaction between students and teachers in real time. With the integration of the VoIP technology into the Web-Lecture, this problem can be tackled.

An IP-to-IP call center prototype is built to allow students to communicate with their tutors/lecturers using VoIP. As a result, students can study at their own pace. When they encounter problems, they can ask their tutors/lecturers through VoIP. This scenario is a call center that is specifically used in an education environment [9].

There are three important elements in this call-center: call management, voice transmission and collaborative browsing. Call management provides user authentication, call establishment, some advanced call features via a Voice Connection Server. Voice transmission provides two-way voice communication using VoIP technologies. Collaborative browser provides the same browser window to both sides for collaborative working. Voice transmission [10] and collaborative

browsing [11] are constructed by other members in the VoIP research group.

### 2.2.1 Architecture

In this IP-to-IP call center, there are four components: students with their computers, lecturers/tutors with their computers, an Intranet and a Voice Connection Server.

The IP-to-IP call center application is depicted in Figure 2-2. In the proposed system, students' multimedia PCs and teachers' multimedia PC are all connected to an IP network. The voice connection server is connected to the network for user authentication, user identification, call establishment and some advanced call features such as call waiting and call forwarding.

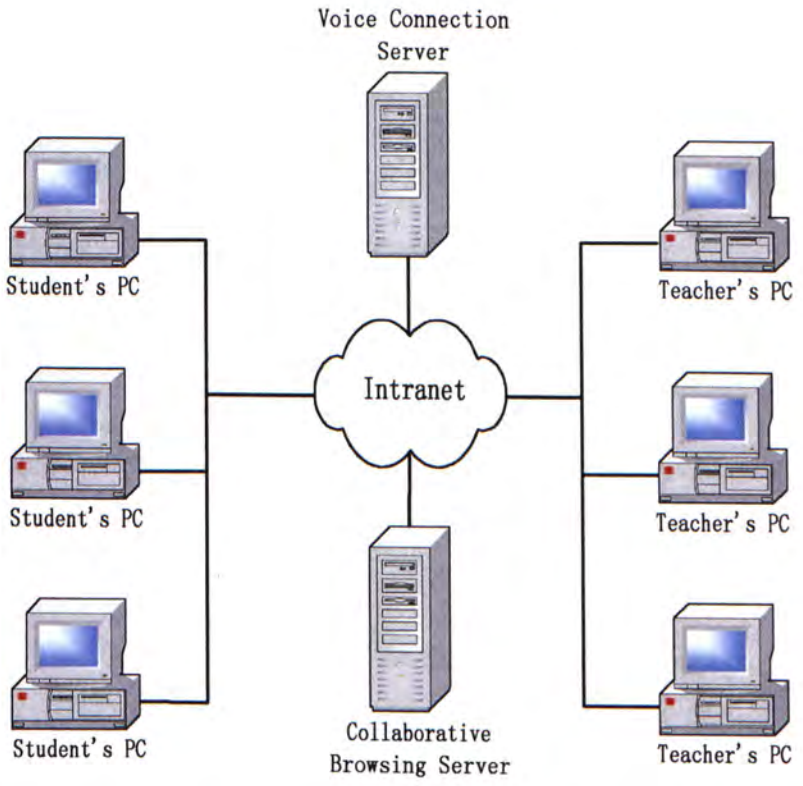



*Figure 2-2 The IP-to-IP center application model used in education environment*

Students attend a lesson on multimedia PCs at their own pace. However, when a student has difficulties in understanding parts of the lesson, he/she may raise questions to a teacher through his/her PC's multimedia system. To do so, the student activates particular VoIP software by clicking the button "Ask teacher" in the main window where class material resides. This request is sent to the Voice Connection Server which finds the IP address of the selected teacher's PC and replies the IP address to the student. The VoIP software in the student then establishes a voice connection with the teacher's PC. As a result, students may ask the teacher questions even when the teacher is not in the same room with the students.

If the questions cannot be answered orally, students and teachers can exchange their ideas through collaborative browsing software. This software component processes the screen transfer to allow teachers to view and control students' navigation of lecture materials. Moreover, under students' permission, the system may pass their web navigation history to teachers such that teachers can understand the overall progress of students more thoroughly. Lastly, teachers may redirect students to pages that answer their questions.

### 2.2.2 Voice Connection Server

The software architecture of the Voice Connection Server is shown in Figure 2-3. The Voice Connection consists of three major components, including a RMI receiver, a core part and a database server. The RMI receiver and the core part are

implemented in the Java programming language.

The RMI receiver receives requests from clients (students or teachers) via Remote Method Invocation (RMI) [12] and transfers these requests to the core part. RMI, which is the Java version of the remote procedure call, provides a way that objects on different computers can interact in a distributed network. When a reply is obtained from the core part, the reply is forwarded to the corresponding client.

The core part provides services to clients (students or teachers), such as logging into the system, establishing calls and call forwarding etc. The core part also maintains the online status of students and teachers who have logged into the system, such as IP addresses of PCs which these people are using.

The database maintains teachers' and students' profiles (e.g. user name, user password, etc) in persistence storage. When the system needs such information, the core establishes a connection with the database server through Java Database Connection (JDBC) [13]. The JDBC API provides a call-level API for SQL-based database access in Java programming language.

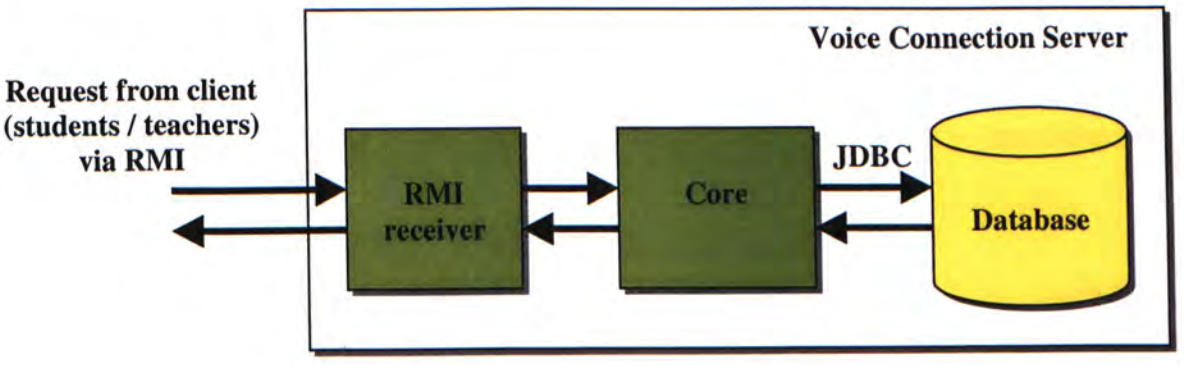



*Figure 2-3 Software architecture of the Voice Connection Server*

### 2.2.3 Call Establishment

Figure 2-4 shows steps to set up a voice connection between a student and a teacher.

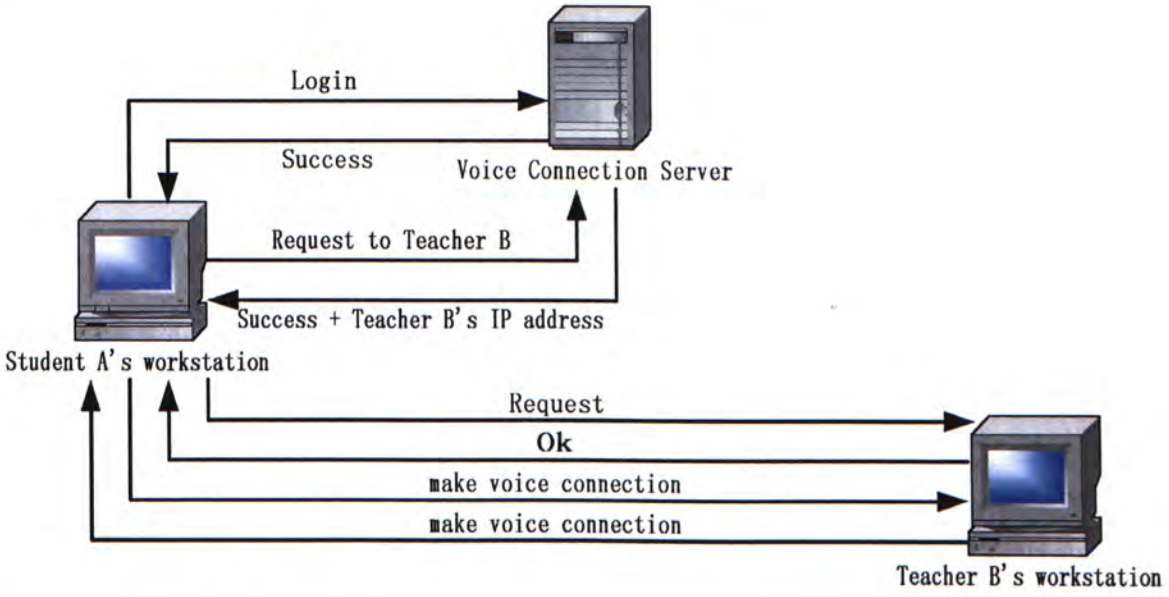



*Figure 2-4 Establishing a call between a student and a teacher*

When students or teachers activate the VoIP client software, they need to type in their user name and password to login the system. The user authentication is done in the Voice Connection Server. When a student wants to talk to a specific teacher, the student sends this request to the Voice Connection Server. The server replies the student with the IP address of the PC which the teacher is using currently. Then the student sends a call request to the teacher and then a voice connection is established between them.

### 2.2.4 A Preliminary Implementation

The implementation of this IP-to-IP call center shows that it is feasible to

implement user authentication, user identification, call establishment and some advanced call features such as call waiting and call forwarding (services provided the Voice Connection Service) as software on top of a PC connected to an IP network.

The implementation is mainly based on the Java programming language and some of its APIs. This provides a platform independent environment. With the deployment of Java applets, our system can be easily integrated with the Web. Also, the rich support from APIs can shorten the implementation time.

During the setup of voice communication, interactions between the Voice Connection Server and two client software (the caller and the callee) are implemented through Java RMI. With the deployment of Java RMI, Network socket programming between client and server which is a quite cumbersome work need not concerned. We can mainly concentrate on the interface and the implementation of the service requested from the clients.

However, it is found that the system is not reliable enough. The Voice Connection Server contains the online status of students and teachers who have logged into the system in volatile memory. When the Voice Connection Server crashes, this information disappears. Even if the Voice Connection Server restarts, this information cannot be recovered.

Secondly, even if the system puts all system information in a database, we still cannot guarantee that all the information stored in database is consistent. When the Voice Connection Server processes operations on behalf a client's request, the Voice Connection Server updates the system information in the database. Meanwhile, if the Voice Connection Server crashes, all database updates may not be finished

before the crash. This makes the information stored in the database in consistent.

Problems related to the system reliability are figured out in this experimental implementation. These problems will be tackled in the ultimate implementation of the proposed Intelligent IP-based Call Center Model.

# 3 The ACD and Its Software Structure

The chapter firstly describes a three-layer software structure for the proposed Intelligent IP-based Call Center model. The ACD operates in the middle layer of this three-layer software structure. The chapter shows how this middle layer interacts with other layers and provides a functional overview of the ACD.

## *3.1 Three-Layer Software Structure*

To make the proposed model suitable for customer services in different applications, a specific software structure is designed.

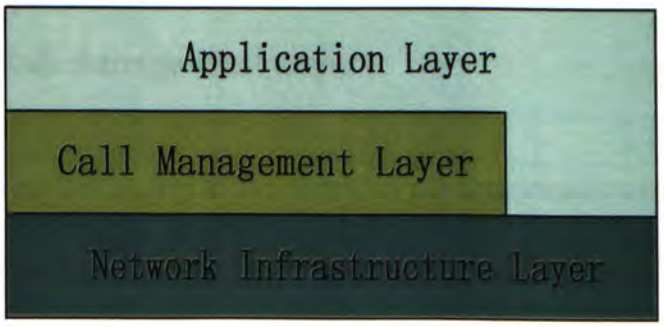



*Figure 3-1 The software structure of the CRM model*

Figure 3-1 shows the software structure. This software structure contains three layers, namely network infrastructure layer, call management layer and application layer.

### 3.1.1 Network Infrastructure Layer

In the button, there is a network infrastructure layer which facilitates the transmission of voice over IP networks. Codecs [14] and transmission protocols for voice and data reside in this layer. Before transmitting voice over IP networks, the voice is digitized from analog form using codecs. In our case, the GSM codec [15] is currently used. After codec is employed to digitize analog voice signal into binary form, we divide the resultant data into segments and place these segments into packets to be transmitted over the network. Realtime Transport Protocol (RTP) which is particularly designed for transmission of real time packets is used to transmit voice packet. Transmission Control Protocol (TCP) [16] is used to transmit the control data.

### 3.1.2 Call Management Layer

Located on top of the network infrastructure layer, the call management layer can be considered as the middleware layer which provides call management such as establishing calls, maintaining call states and supporting advanced telephony features and fault tolerance. Call management of gateways, the monitoring tool and the ACD operate in this layer. The call management layer is based on TCP/IP provided by the network infrastructure layer.

### 3.1.3 Application Layer

Located on top of the call management layer is an application layer which provides services to operators, such as logging into the system, notification of receiving a call, accessing the profile of customers and collaborative browsing service. Operator applications operate on the application layer. The application layer interfaces with the network infrastructure layer through the RTP socket API for sending and receiving voice stream. The application layer interfaces with the call management layer via a self-defined API for sending and receiving control data.

### 3.1.4 Interoperation Between Layers

Figure 3-2 illustrates how these layers interoperate when a customer phones to the call center. When a gateway receives a new incoming call, the call management layer in the gateway initiates a Remote Method Invocation (RMI) request to the call management layer in the ACD. RMI, which is the Java version of the remote procedure call, provides a way that objects on different computers can interact in a distributed network. If there is a suitable operator who is available to answer this call, the ACD then replies to the gateway with the IP address of the operator. If the gateway does not receive any response from the ACD, this implies that the ACD has failed. The call management layer in the gateway informs the call management layer in the monitoring tool that the ACD fails to respond through a RMI request. If the gateway receives the reply with the IP address of the operator from the ACD, the gateway requests a voice connection with this operator. In the operator application, the call management layer receives this request and propagates

it to the application layer. The application layer decides whether to accept this call and sends this decision to the call management layer. Then the call management layer of the operator application replies to the call management layer of the gateway. If the operator accepts the incoming call, a voice connection is established between the gateway and the operator application through RTP packets.

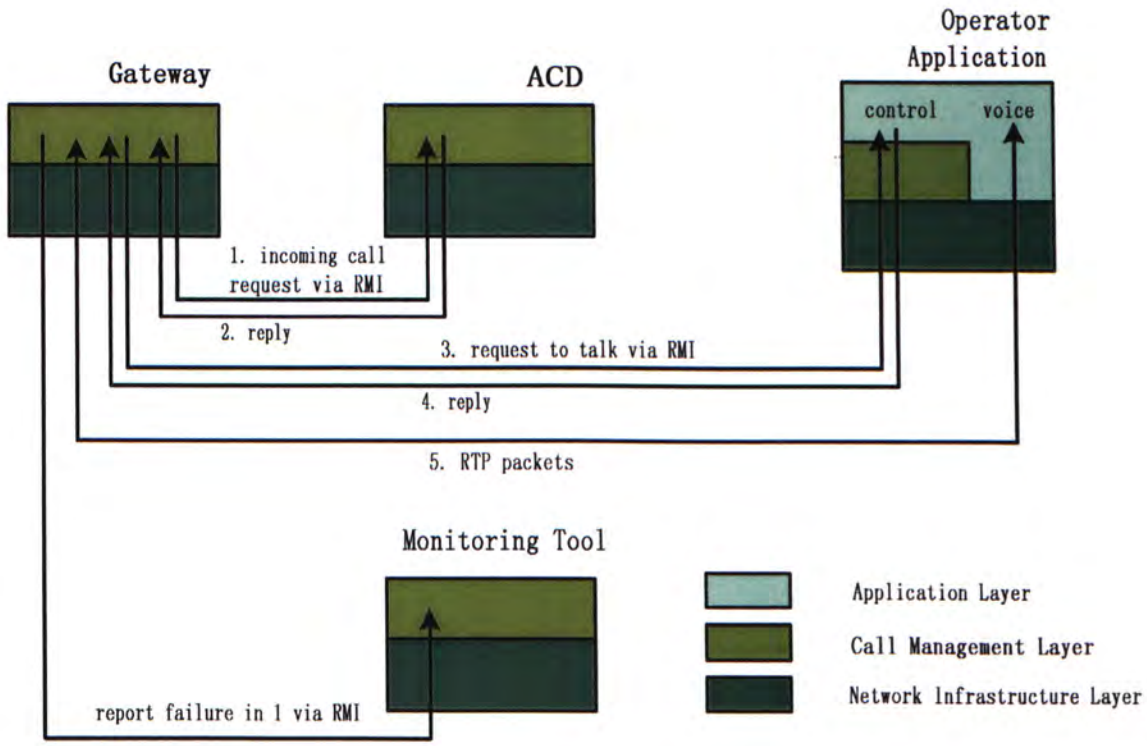



*Figure 3-2 Interoperation between layers when a customer phones to the call center*

## 3.2 Advantages of Adopting this Software Structure

This three-layer design increases the reusability of components in the proposed model and shortens the development time of the application model. Interfaces between various layers are clearly defined. Components in the application

layer can be changed to suit specific needs while components in other layers remain unchanged. For example, if a VoIP enabled web lecture [9] is built, we can keep the components in the network infrastructure layer and the call management layer in the Intelligent IP-based Call Center Model. The collaborative browsing service and the operator software in the proposed model can be altered to suit the needs of the interactive web lecture system. In such a system, students can ask teachers remotely and they work collaboratively to find solutions to problems by exchanging screens and the relevant webpages.

## 3.3 Functional Overview of the ACD

The ACD, as the core component of the proposed model, provides services to gateways and users. Services provided include call establishment, call waiting and call forwarding.

When a customer phones to the call center, the ACD finds the most appropriate operator to serve the incoming call. If there is no suitable operator to handle this call immediately, this call is put into a waiting queue temporarily. This section will describe the routing mechanism in the ACD.

### 3.3.1 Call Establishment

When a customer phones to the call centre, the phone call is answered by a VoIP gateway. The gateway acts as a mediator to request service on behalf of the

customer. Figure 3-3 depicts steps to set up a voice connection between a customer and an operator.

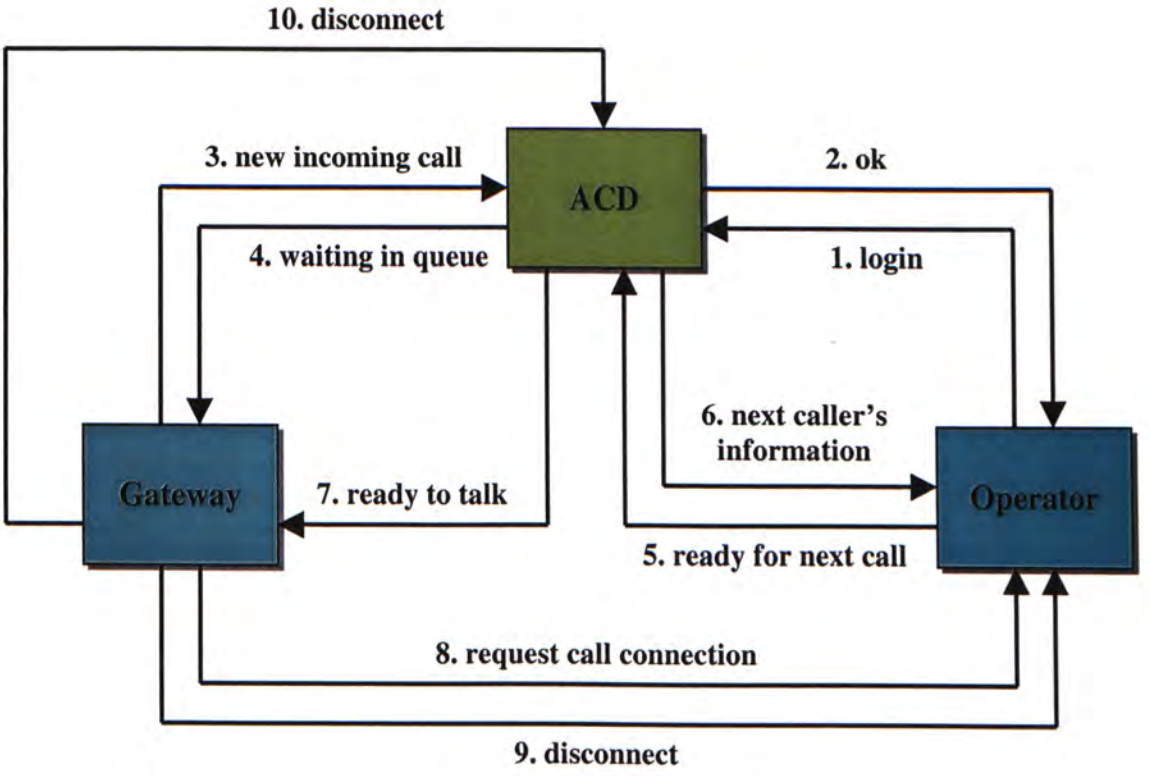



*Figure 3-3 Typical call establishment*

Before a customer phones to the call centre, at least one operator has logged into the system (Step 1 and 2). When a customer phones to the call centre, the gateway sends an "new incoming call" request to the ACD with parameters, such as the caller id, the phone number the customer has dialed and the selection made by the customer from touch-tone input (Step 3). The ACD checks whether an suitable operator is available to answer this call. If so, the ACD informs the gateway an operator is assigned to the customer (Step 7; Steps 4-6 skipped). Otherwise, the ACD

informs the gateway that this call is assigned to a certain queue according to these parameters (Step 4) (The mechanism is explained in Section 3.3.4 "Routing Mechanisam in the ACD" in detail.). When an operator finishes a call, the operator informs the ACD that he/she is ready to answer another call (Step 5). The ACD replies to the operator with the information of the next caller (Step 6) and informs the gateway that this call is answered by this operator (Step 7). Then the gateway initiates a call connection to this operator. When the call is to be finished, either the operator or the gateway can disconnect the call (Step 8) and the one who invokes the disconnection needs to report the disconnection to the ACD (Step 9).

### 3.3.2 Call Waiting

The call waiting feature lets an operator to hold a current call and accept a new call. Figure 3-4 shows how the operator uses this feature. Operators probably do not receive new incoming calls when they are answering calls. The situation may occur only when an operator forwards a call to a specific operator who is answering another call at that time.

We assume that customer 1 and operator 1 have already made a voice connection (Step 1). When customer 2 requests a call connection to operator 1 (Step 2), operator 1 accepts the new call (Step 3). At the same time, operator 1 needs to hold the original call and temporarily close the voice connection with customer 1 (Step 4). The gateway reports the ACD that the customer 1's call is being held (Step 5). The voice connection between customer 2 and operator 1 is established (Step 6). When Operator 1 wants to hold customer 2's call and talk to customer 1 again,

operator 1 send a "hold" message to customer 2 (Step 7) and the gateway reports to the ACD that customer 2's call is being held (Step 8). Operator 1 then informs customer 1 to resume the call (Step 9) and the gateway reports this to the ACD (Step 10). Voice connection is re-established between customer 1 and operator 1.

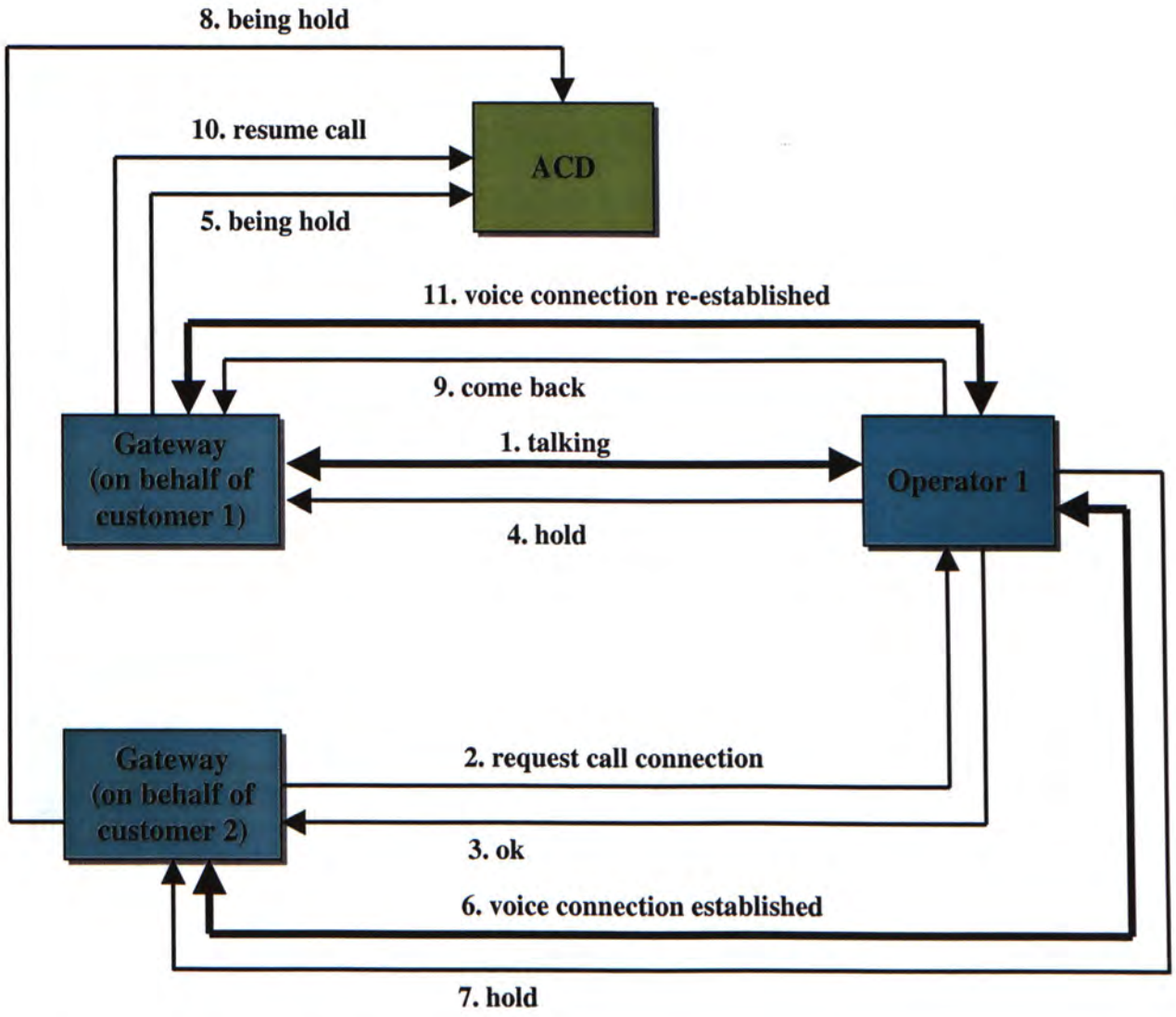



*Figure 3-4 Example of call waiting*

### 3.3.3 Call Forwarding

The call-forwarding feature lets an operator forward a call to another operator when the operator is not able to answer a customer's question in this call. Figure 3-5 shows an example of call forwarding. Here assumes that a customer is talking with operator 1 now (Step 1). When operator 1 finds that he/she is not able to answer the customer's questions, the operator can forward the call to a specific operator or forward the call to another queue (Step 2 and 3). The ACD may give the customer high priority to leave the queue as the customer has waited in a queue once before. When there is an available operator (operator 2) to answer the customer's call, the ACD sends a "ready to talk" to the gateway (Step 4). Then the gateway initiates a call connection to operator 2 (Step 5).

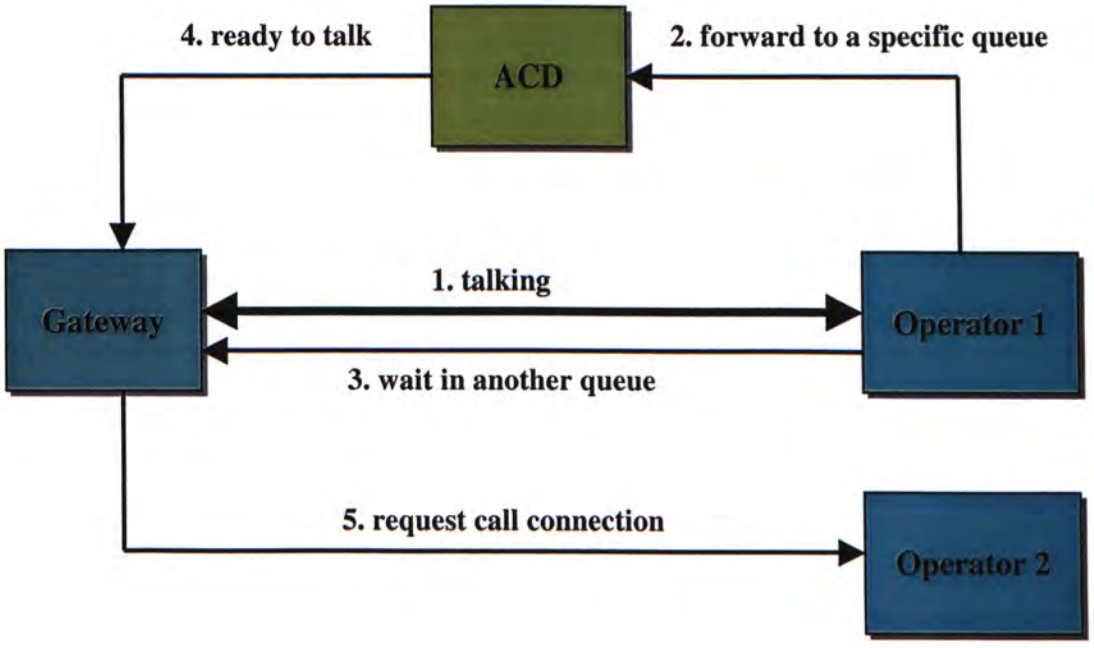



*Figure 3-5 Example of call forwarding*

### 3.3.4 Routing Mechanism in the ACD

In the proposed model, a new incoming call is handled by a suitable operator immediately or waits in one of the waiting queues. To effectively find an operator who has specific skills to solve a certain kind of questions, operators are organized into groups according to skills these operators possess. If an application service provider (ASP) provides call center system help desk service to more than one customer, operators need to handle different kinds of questions from customers of different companies. In this situation, it is also important to organize the operators into groups. This section firstly describes the relationships between queues, operator groups and operators. To provide better service to more valuable customers, the waiting queue is priority based rather than first-come-first-serve based. The policy is mentioned in this section. When an operator has just completed a call, the ACD assigns another call from a waiting queue to the operator. The method of assigning calls to operators is also mentioned in this section.

#### a) Queues, Operator Groups and Operators

To effectively find an operator who has specific skills to solve a certain kind of questions, operators are organized into groups according to skills of these operators. Take a bank's call center system as an example. Questions from customers are probably related to savings account service, credit card service, mortgages and loans and etc. Operators are provided with a series of training and each of them has his/her strength or knowledge related to a certain kind of service. Each operator belongs to an operator group. For example, some operators who are

more capable to handle questions about saving account service form an operator group as shown in Figure 3-6. Some operators who are more capable to handle questions about credit card service, and mortgages and loans form another operator group.

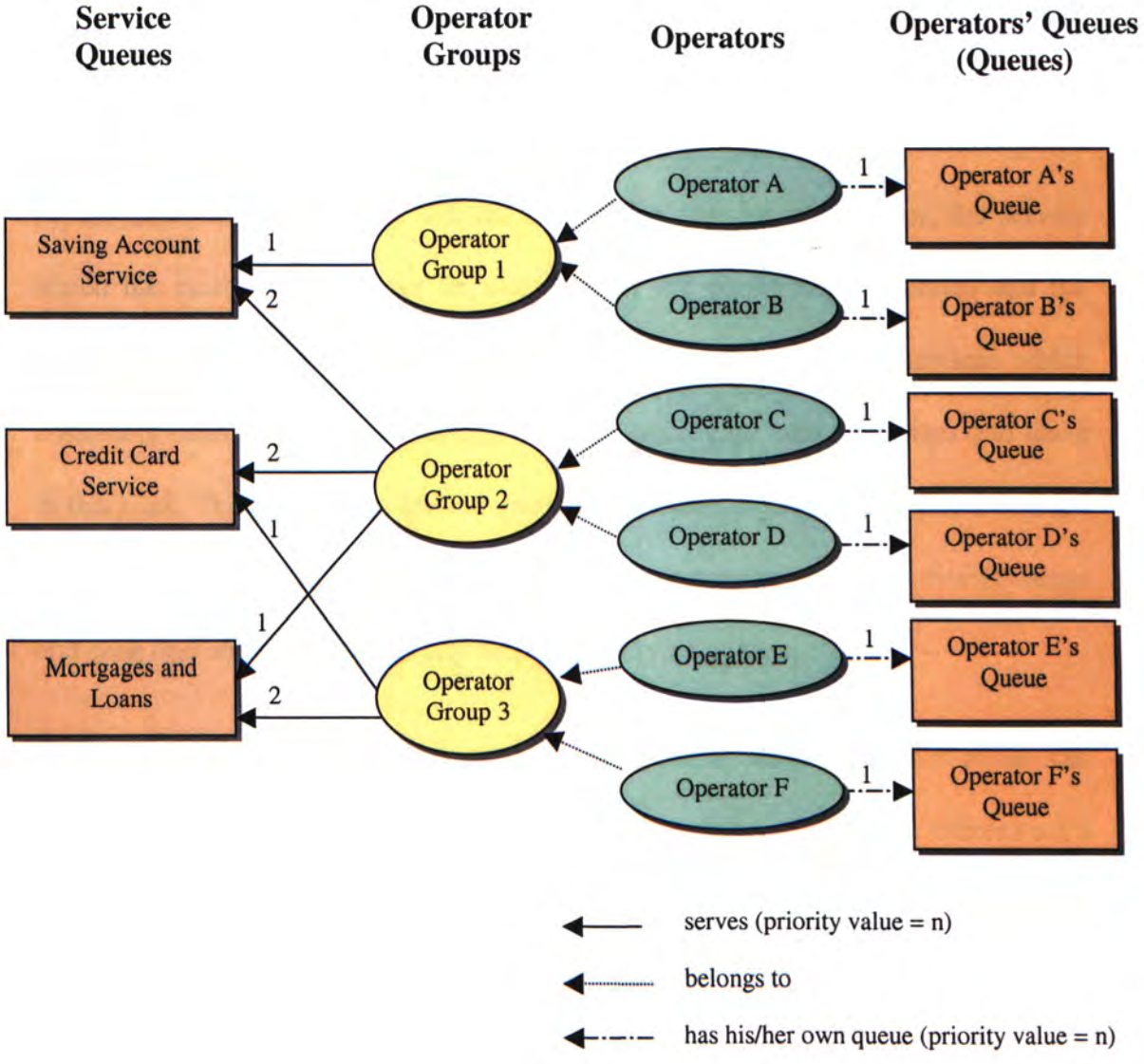



*Figure 3-6 Relationship between Queues, Operator Groups and Operators in a Bank's Call Center*

The notion of operator group may decrease the flexibility in setting the characteristic for individual operators. If a new service queue is created, a number of

operator groups are assigned to serve this service queue. It is not easy to assign a subset of operators in a group to serve this service queue. As an alternative design, operator groups are eliminated. Each operator is assigned to serve a number of queues with different priority value. However, this alternative design decreases the efficiency in finding suitable operators available to answer incoming calls. This issue is to be discussed in Section 4.3.2.

Incoming calls are put into different queues according to the services required by the customers. Before the call has been taken care by an operator, the service which the customer needs can be identified by the phone number dialed and the touch-tone input from the customer. Using the previous example, services which customers need are related to savings account, credit card, and mortgages and loans in this bank. Therefore, three service queues are created as shown in Figure 3-6. Each service queue is served by any number of operator groups with certain priority values and each operator group serves any number of service queues.

Each operator has his/her own queue. When a customer phones to the call centre, he/she may want to talk to a specific operator. When a customer makes such a request, the ACD assigns the customer to the operator's own queue. The number of phone calls entering this kind of queue is limited to a certain value. Otherwise, customers will wait for a long time when they want to talk to the same operator.

### b) Priority Based Call Routing

To provide better service to more valuable customers, waiting queues are

priority based rather than first-come-first-serve based. From the caller id, the ACD can retrieve the customer's profile. Each customer is provided with a priority value in the customer profile. In the same queue, the customer with higher priority gets served faster than those with lower priority.

### c) Routing of New Incoming Calls

When the ACD is informed by a VoIP gateway that a new incoming call arrives to this gateway, the gateway provides the ACD with the information about this call, including the caller id of this call, the phone number the caller has dialed and the selection made by the caller through touch-tone input. The ACD either assigns a suitable operator to answer this call immediately or puts this call in one of waiting queues.

Figure 3-7 shows how the ACD handles this request from the VoIP gateway. First of all, the ACD finds whether the caller id matches any customer's telephone number in the database. If there is a match, it is probably that the caller is this customer stored in the database and the profile of this customer is retrieved from the database; otherwise, the caller is asked to input his/her customer id.

From the phone number the caller has dialed, the ACD can identify a domain of services the caller needs. If necessary, the ACD can further identify the exact service that the caller needs through the selection made by the caller from touch-tone input. Then the ACD checks whether another call with equal or higher priority exists in the queue which is corresponding to the service the customer needs. If another call

with equal or higher priority exists in this queue, this new incoming call is put into this queue; otherwise, the ACD finds an available operator who serves this queue to handle this call. If no operator is available, this call is put into this queue. If exactly one operator is available, the ACD assigns this operator to handle this call. If more than one operator who serves this queue is available, the ACD chooses an operator who has served this customer recently to answer this call.

Incoming calls are put into different queues according to the service which the customer needs.

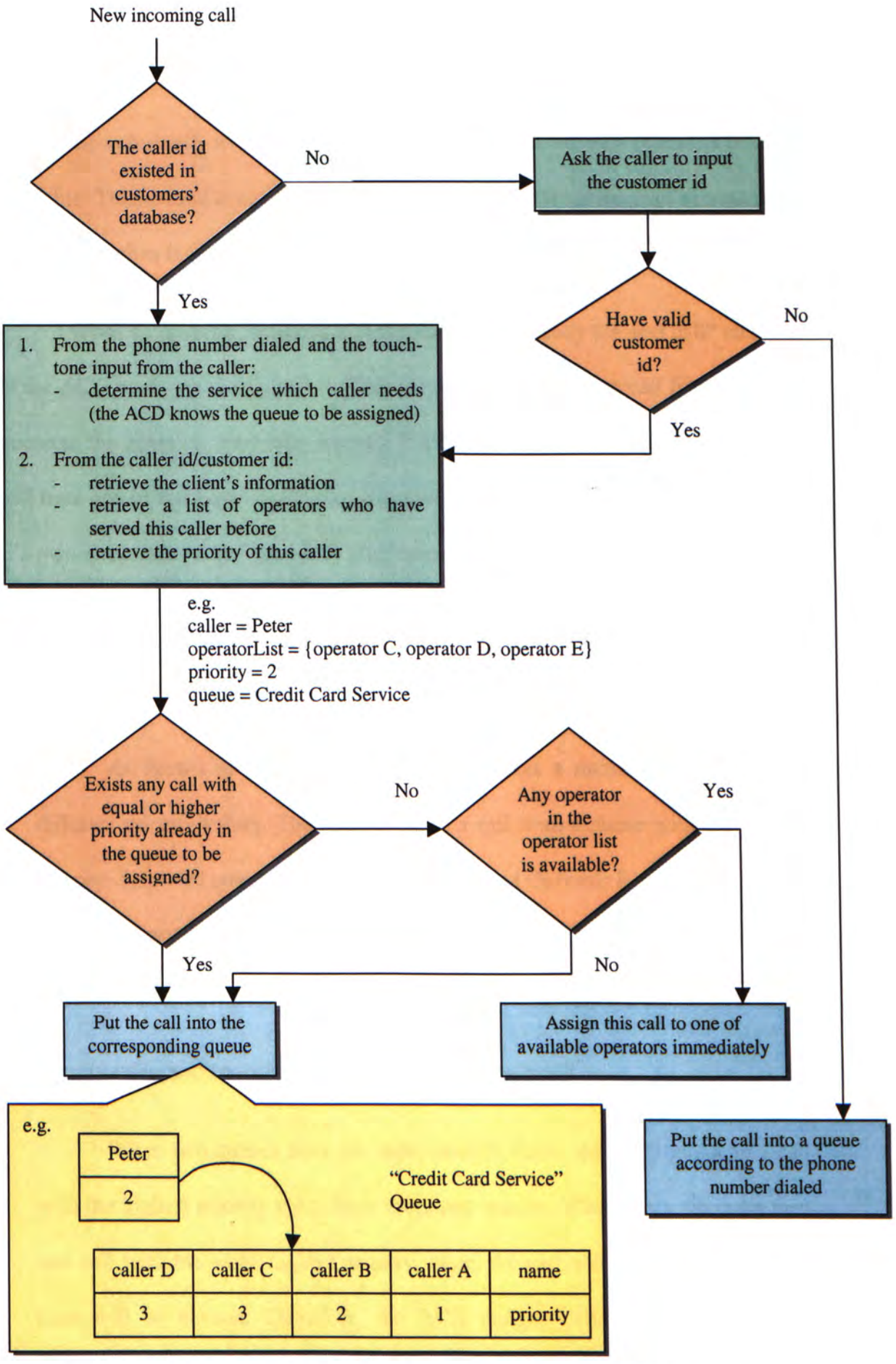



*Figure 3-7 Flow diagram for handling a new incoming call in the ACD*

### d) Assigning Calls in Waiting Queues to Operators

Incoming calls which are waiting in queues get served until operators become available. This section describes how the ACD assigns an incoming call in a queue to an operator when this operator becomes available.

When an operator finishes a call, he/she sends a "ready for next call" message to the ACD as shown in Figure 3-3. Then the ACD assigns a new call from a certain queue to the operator. Here take operator E as an example. The ACD assigns a new call from one of three queues (credit card service, mortgages and loans, and Operator E's own queue shown in Figure 3-6) after operator E finished a call. The ACD makes the assignment based on the following values:

1) priority value associated with each queue for this operator

   As shown in Figure 3-8, Operator E serves a number of queues with different priority values. The ACD chooses a call from a queue with the highest priority. Since the queue for credit card service and Operator E's own queue have the highest priority and these two queues are not empty, the queue for mortgages and loans is ignored.

2) priority value and enqueue time associated with each call

   When two queues have the same priority value, the ACD looks up a call with the highest priority value from these two queues. When there are more than one call with the same highest priority value, the call with the earliest enqueue time will be chosen. Therefore, the ACD assigns caller A's phone call to Operator E.

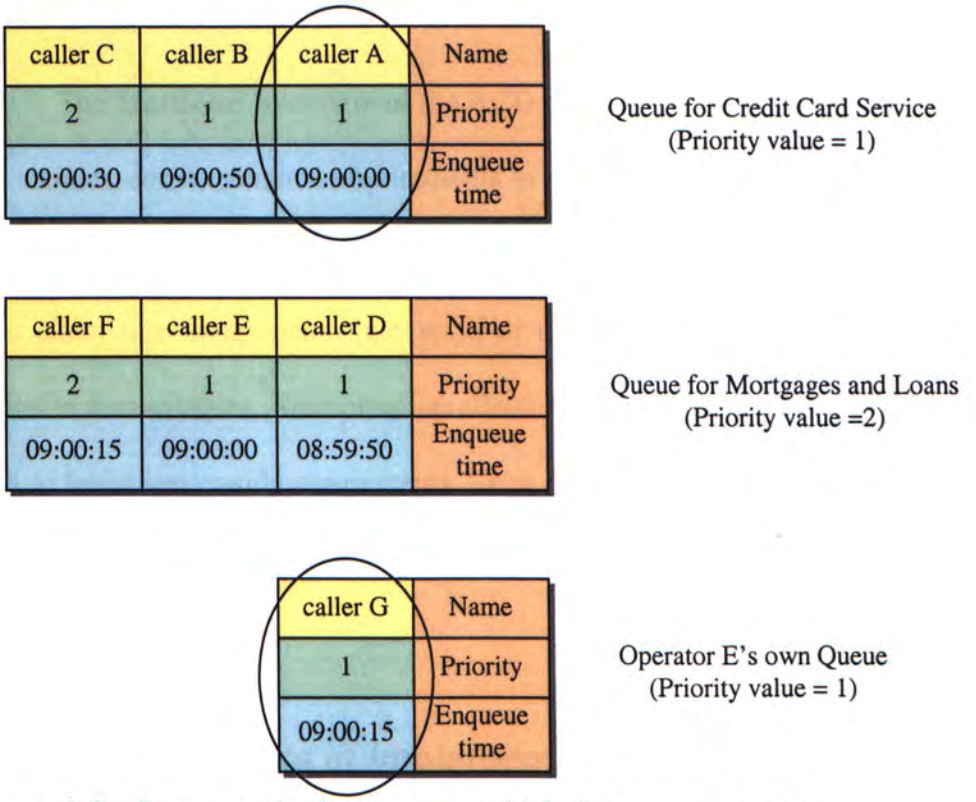



*Figure 3-8 Customers in three queues which Operator E serves*

# 4 Implementation of the ACD

The functional overview of the ACD has been provided in the last chapter. In this chapter, some technical requirements in implementing the ACD are figured out. The ACD can be considered as server-side components which operate upon requests from client-side components (gateways or operators). This chapter introduces two available technologies, Enterprise JavaBean (EJB) [17] and COM+ [18], which are used to build server-side components. Then this chapter describes the way in which the ACD is implemented to fulfill these requirements using the EJB model.

## *4.1 Requirements in implementing the ACD*

In the last chapter, functional requirements for the ACD have been described. In this section, technical requirements related in the implementation are discussed. They include asynchronous method call provided, transaction planning in the ACD and the failure handling of the ACD and other components.

### 4.1.1 Asynchronous Method Call

In most situations, when a client (a gateway or an operator) makes a request to the ACD, the ACD replies to the request immediately. However, when a gateway sends a request from a newly arrived incoming call to the ACD. The ACD may not be able to assign an operator to answer this call immediately. The ACD sends the reply that an operator is assigned to answer this call when a suitable operator is

available to answer this call.

In most architecture models provided for building server-side components, only synchronous method call to server-side components from clients is provided [19] [20]. In synchronous method call, when a client sends a request to a server-side component, the control of the client is blocked. The control is returned to the client until the client receives the reply from the server-side component. In most situations, synchronous method call is suitable to implement the ACD because the ACD can most of time reply to clients' requests immediately (as shown in Figure 4-1 (b)). However, when a gateway sends a request about the arrival of a new incoming call to the ACD, the ACD may find that there is no suitable operator available to handle this new call at this moment. The ACD does not know at what time an operator becomes available to handle this call (as shown in Figure 4-1 (a)). In this situation, asynchronous method call is also needed.

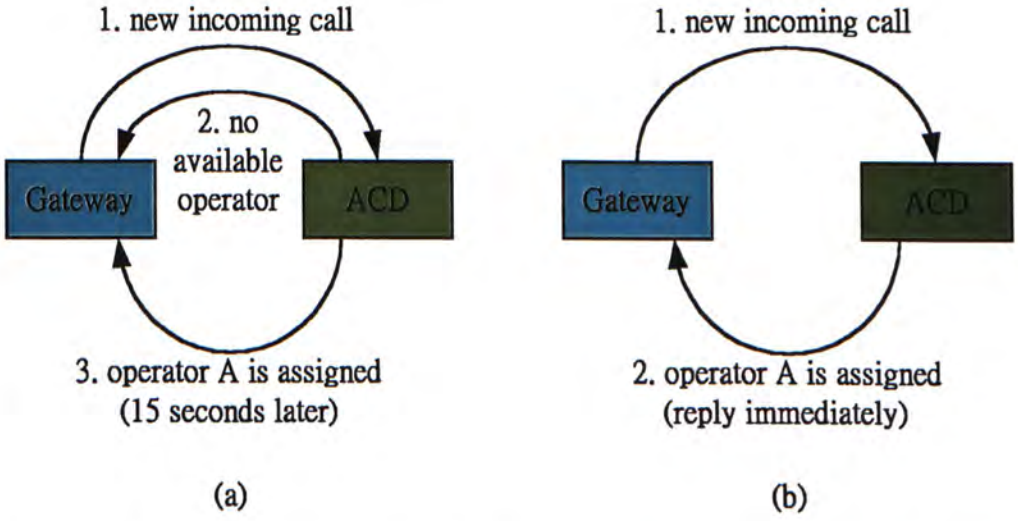



*Figure 4-1 (a) asynchronous method call and (b) synchronous method call*

### 4.1.2 Transaction Planning

When a client (a gateway or an operator) sends a request to the ACD, the ACD should execute a series of operations as a transaction. The transaction must be executed in all-or-nothing manner. For example, when an operator disconnects his/her current call with a customer, the operator sends a "disconnect" request to the ACD. The ACD executes the following operations:

1) The ACD records that this operator has communicated with this customer from time unit A to time unit B in the call history.

2) The ACD records that this operator has stopped communication with this customer since this moment.

3) The ACD changes the status of this operator from "talking" mode to "offline" mode.

This series of operations forms a transaction which either completes successfully and the effects of all of its operations are saved in permanent storage or (if it fails) it has no effect at all.

A transaction should possess four properties as follows:

1) Atomicity

The transaction must be all-or-nothing.

2) Consistency

The transaction takes the system from one consistent state to another

consistent state. For example, after a transaction, it is impossible for the ACD to keep an inconsistent state in which an operator is talking with a customer but the status of the operator is in offline mode.

3) Isolation

Operations taken on behalf of a client are invisible to other operations taken on behalf of other clients.

4) Durability

After a transaction, all its effects are saved in permanent storage. Data saved in permanent storage will survive in case the server process crashes. For example, an operator has logged in to the system and has been talking with a customer. When the ACD crashes, all the status about this operator and this customer are saved in permanent storage. When the ACD restarts, the ACD can retrieve all the status about this operator and this customer.

### 4.1.3 Failure Handling

The call center, like all distributed applications, faces several potential points of failure. A single point of failure that will impact operation must then be identified and a mechanism to overcome each of the potential failures must be provided. This will allow the system to recover relatively quickly from the loss of a functional component. The failure handling for VoIP gateways and the ACD should be considered.

If a VoIP gateway crashes, the system should detect this failure. A failure-handling procedure should be designed to replace the failed gateway with a new one. Meanwhile, other functional components continue to provide services without any influence from the failed gateway. Moreover, the system should call back to those customers whose calls are cut due to the failure.

If the ACD crashes, other components in the system inevitably get affected because the ACD is the core component to provide services to other components. Therefore, it is necessary to improve the reliability of the ACD. If a component in the ACD fails, we try to recover the failed component transparently to clients. Even if the ACD needs to restart, the ACD should keep all consistent system information before the crash after the ACD restarts.

## 4.2 Available Technologies

The section introduces two available technologies, EJB and COM+, which are used to build server-side components.

### 4.2.1 Enterprise JavaBean (EJB)

EJB is Sun Microsystems' specification in distributed system. It is used to develop server-side components at enterprise-level. Distributed components run on different hosts to form the system.

Figure 4-2 shows the software architecture of the EJB model. The EJB Server is the outermost container of the various elements making up an EJB environment. The server manages one or more EJB containers and provides the required support services, such as transaction management, persistence, and client access. The server also provides the operation resources, such as multi-threading, memory, distributed naming, remote invocation, and so on, to the containers and their elements within. This allows application developers to concentrate on the business logic of the application, which is implemented as enterprise bean objects. Since EJB provides a well-defined component-model that promotes the separation of business logic from the underlying services, re-implementing business logic to reflect evolving business practices in EJB is considerably easy.

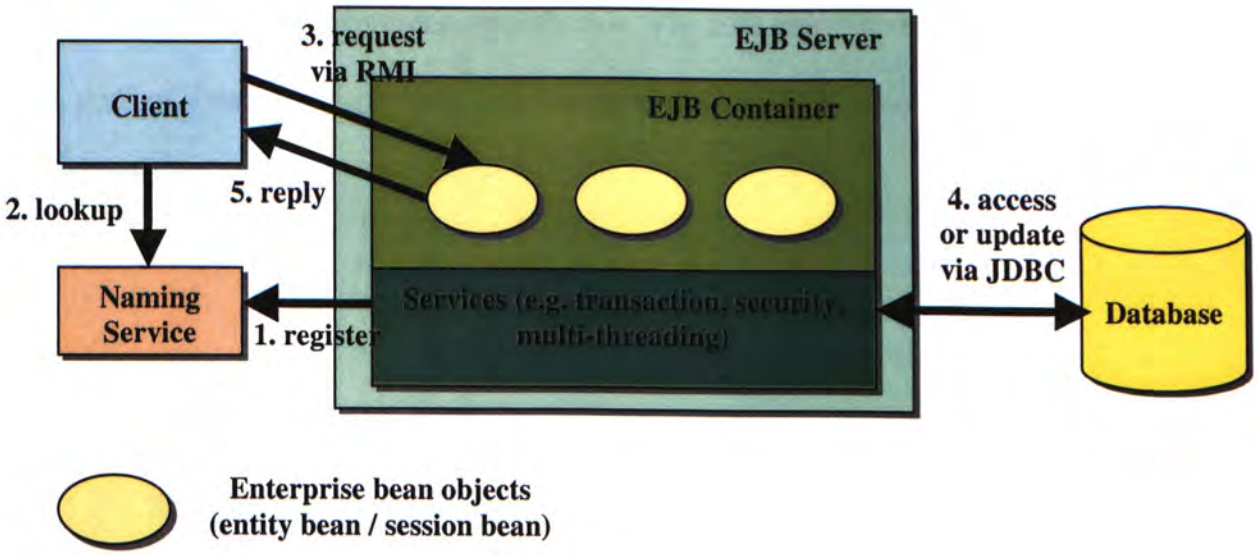



*Figure 4-2 Software architecture of Enterprise JavaBean*

Clients use Java Naming and Directory Interface (JNDI) [21] calls to locate enterprise beans and RMI calls to invoke functions on the bean's well-defined interfaces. JNDI which is a standard extension to the Java platform provides Java

applications with a unified interface to multiple naming services in the enterprise. Encapsulation separates client code from the bean implementation. Server developers can change a bean behavior without requiring corresponding client-side changes.

The EJB model offers two types of enterprise beans: entity and session bean.

### a) Entity Bean

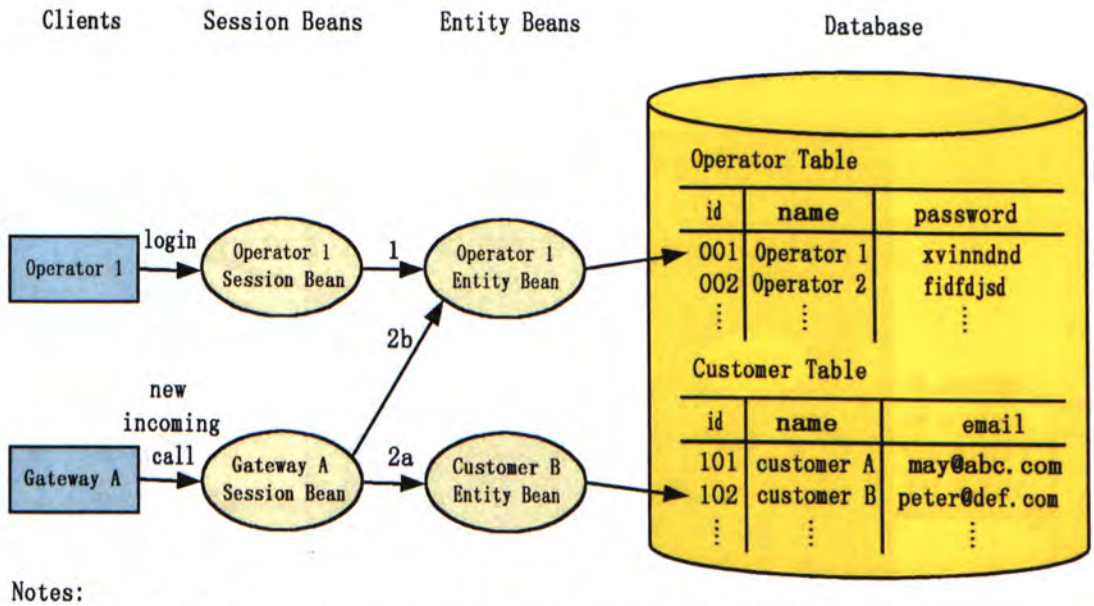
Entity beans are data components. A data component is an in-memory replica of data stored in a database. Data components encapsulate access to permanent data by wrapping that data in an object, and associating methods with that data. A simple entity bean could be defined to represent a single row in a database table, where each instance of this bean represents specific row. The EJB server and container maintain a pool of database connections and transparently handle transactions for generated entity beans.

### b) Session Bean

Session beans are business process components. A business process component represents a logical extension of a client program that runs on the server. It performs operations on behalf of the client. Examples of session beans include allocating a suitable operator to answer a new incoming call, forwarding a call to another operator and verifying the login of an operator.

### c) Usage of Session Beans and Entity Beans

Figure 4-3 shows a usage of session beans and entity beans and their relationships. Take the proposed call center model as an example; operators' information and customers' profile are stored in a database. When an operator logins the system, the request is handled by an operator session bean in an EJB server. The session bean checks whether the password the operator types in is identical to the one stored in the database. The session bean accesses the password stored in the database through an operator entity bean. When a gateway receives a new incoming call, the gateway sends a "new incoming call" request to the ACD. A gateway session bean in the EJB server handles this request. The gateway session bean finds the identity of the customer from the database through a customer entity bean with the caller id of this incoming call. According to the customer's profile, a suitable operator is found from the database through an operator entity bean.




Notes:
1. check whether the password the operator types in equals the one stored in the database
2a. find the identity of the customer through the caller id
2b. find a suitable operator to handle this call

*Figure 4-3 Session beans and entity beans*

### 4.2.2 COM+

COM+, which is an extension of Component Object Model (COM) [22], is Microsoft's architecture for building re-usable, server-side components.

Except the terminology, the software architecture of the COM+ model as shown in Figure 4-4 is similar to that of the EJB model. A Microsoft Transaction Server (MTS) [23], which is the equivalent of the EJB Server in the EJB model, is the outermost container of the various elements making up a COM+ environment. The MTS manages one or more MTS Executives and provides the required support services, such as transaction, security, multi-threading. An MTS Executive acts like the EJB container, hosting MTS objects rather than enterprise beans. MTS objects are typically designed to encapsulate some set of business functionality. For example, a MTS object helps a gateway to choose a suitable operator to answer an incoming call. This allows application developers to concentrate on the business functionality of the application, which is implemented as MTS objects.

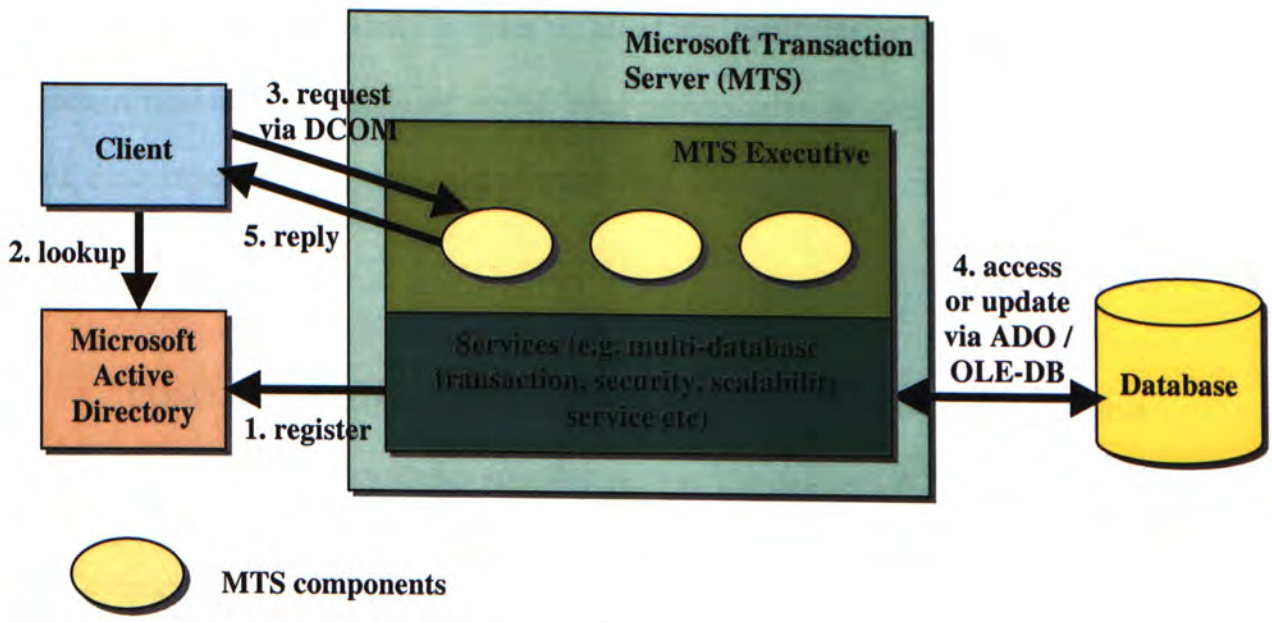



*Figure 4-4 Software architecture of COM+*

Clients use Microsoft's Active Directory [24] to locate MTS objects in the MTS, and use DCOM [25] to invoke methods on those objects. Encapsulation separates client code from the implementation of MTS objects. Server developers can change a MTS object's behavior without requiring any corresponding client-side change.

COM+ supports business process components, which are called COM+ components. However, COM+ has no notion of a data component, so a business process component must operate on rectangular rows in database tables, rather than on data components, which is highly non-intuitive. To integrate with databases in the COM+ model, Microsoft's Active Database Objects (ADO) [26] along with OLE/DB [27] and Open Database Connectivity (ODBC) [28] have to be used.

### 4.2.3 EJB vs COM+

EJB and COM+ are two prominent choices for building server-side components. In this section, in order to select the particular technologies for the implementation of our proposed model, these technologies are compared as follows and the comparisons [29] are summarized in Figure 4-5:

1) Platform supported

COM+ is tied to the NT platform. However, EJB can run on any platform that supports the Enterprise JavaBeans standard, such as Windows 95, NT and Unix.

2) Component types supported

COM+ provides business process components only, while EJB provides both business process components (named "Session Bean") and data components (named "Entity Bean").

The support for data component types is important in developing server-side components [29]. COM+ has no concept of data component, so a developer's business process components must operate on rectangular rows, rather than on data components, which is highly non-intuitive. It is much more natural to call a method, *operator.getPassword()* than to deal with a relational result set.

Moreover, the support for data component types provides abstraction on underlying database schema from business process code. This enables a database administrator to change a relational database schema for performance, or change to a different database product without modifying a single line of code in business process code. For example, operator.getPassord() corresponds to retrieving an attribute "password" in a certain row in a database table "Operator". If a company has defined a different database schema for its operator's information (such as using another name of attribute "passwd" to record operators' login passwords), we can simply change the mapping of the operator.getPassword() to the new attribute for the retrieval in a specific description file. In this way, the EJB development tool automatically generates a new set of program code.

3) State management

EJB provides support for stateful business processes through stateful session beans, whereas COM+ provides no support for stateful business processes. A stateful business process is a business process that spans multiple method calls. A stateless

business process is a business process that spans a single method call. A stateful session bean maintains state on behalf of a particular client, and is tied to that client for the component's lifecycle.

The support for stateful business processes in the underlying EJB architecture provides convenience in the implementation of our proposed model. In our proposed call center, after an operator logins the system, the ACD should hold the state of the operator throughout the login session. This can be well-modelled with stateful business process components.

4) Asynchronous method call

EJB provides no support for asynchronous method call, whereas COM+ provides support for asynchronous method call.

As mentioned previously, asynchronous method call is needed in the proposed model. In COM+, a client thread can execute a method asynchronously. COM+ starts the call and then immediately returns control back to the client. The client thread is then free to do other work and can later obtain the results of the method call.

5) Transactions supported

Both EJB and COM+ support transactions in a similar manner. EJB and COM+ give the developer control of transactions both programmatically and declaratively. In EJB, developers can control transactions programmatically through the Java Transaction API (JTA) [30], or set attributes on components at method-level rather than writing code to an API. Similarly, in COM+, transactions can be

controlled programmatically via OLE Transactions [31], or transactional characteristics can be set on components rather than writing code.

| | EJB | COM+ |
|---|---|---|
| Platform supported | NT only | Windows 95, NT, Unix and etc. |
| Data component supported | √ | × |
| Stateful processes supported | √ | × |
| Synchronous method calls supported | × | √ |
| Transactions supported | √ | √ |

*Figure 4-5 EJB vs COM+*

In terms of the above comparisons, it is found that EJB is suitable for the implementation of the proposed model. In the proposed model, supports for data components and stateful business processes are relatively important. Changes in underlying database schema to suit the requirement in the application layer of the proposed model are necessary. Data components abstract the underlying database schema from business process code. This reduces the need to change lines of business process code when making changes in underlying database schema.

Moreover, stateful business processes are suitable in the implementation of the proposed model. As mentioned in point 3 in Section 4.2.3, states of gateways and operators should be held on the ACD among multiple method calls. Stateful business processes provide an intuitive paradigm in the implementation.

Due to these reasons, the EJB model is employed to construct the core part of the system. PowerTier for Enterprise JavaBeans [32] from Persistence Software Inc. is used. Although asynchronous method call is not supported in the EJB model, the way to provide asynchronous method call in the implementation will be discussed in detail in a later section.

## 4.3 Implementation

### 4.3.1 Mapping the EJB model to the Implementation of the ACD

The ACD is considered as a server-side component in the software architecture. The ACD is formed by a naming service, a PowerTier EJB server and an Oracle 8i database server. The naming service provides location transparency of the EJB server to client-side components. The PowerTier EJB server transparently allocates threads to manage requests from operators and VoIP gateways and provides service such as transaction, database connection, and cache management. The container provides support for enterprise beans. In the ACD, the container registers the EJB class home interface with the naming service. The Oracle8i database server provides persistence for the information stored in the call center system, such as profiles of the customers, status of the operators and status of the incoming calls in

the system.

Figure 4-6 illustrates the various components of the call center system and their relationships. Operators are considered as clients in the software architecture. They contact the naming service and obtain a remote object reference to a session bean in the EJB server in the ACD. Operators contact the ACD through method calls provided in the session bean. For example, when an operator logins a workstation to answer customers' calls, the operator's software uses this object reference to inform the EJB server in the ACD.

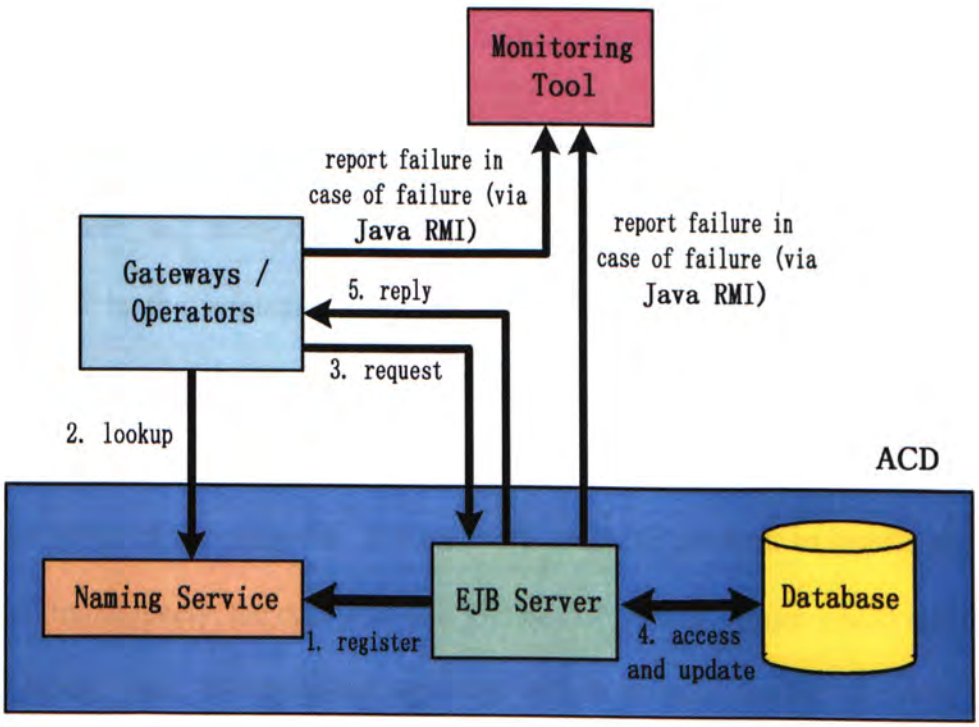



*Figure 4-6 Various components of the call center system and their interactions*

VoIP gateways are also considered as clients in the architecture. They contact the naming service and obtain a remote object reference to a session in the EJB server in the ACD. VoIP gateways contact the ACD through method calls provided

in the session bean. When there is a new incoming call, the gateway uses this object reference to inform the EJB server in the ACD.

The monitoring tool receives failure reports from different components including gateways, operators and the ACD. These components report that other failed components which are unable to reply to requested services. Failure reports are prompted to the screen so that the administrator can replace the failed component. Failure handling procedures will be described in details in the next section.

### 4.3.2 Design of Entity Beans

Entity beans encapsulate access to permanent data by wrapping that data in an object, and methods associated with that data. Ten types of entity beans are created in the implementation. Figure 4-7 depicts the relationships among these entity beans. "CustomerProfile" bean models the profile of customers. When a customer phones the call center, a "OnlineCustomerStatus" bean is created to model the status of the customer. If there is no available operator to handle this incoming call immediately, the customer will join one of waiting queues which is modeled by a "Queue" bean. In the system, operators which are modeled by "Operator" beans are grouped according to their knowledge. An operator group is modeled by a "OperatorGroup" bean. Each operator group is assigned to serve one or more queues. The assignment is modeled by a "OperatorGroup_QueueAssignment" bean. When an operator is assigned to handle an incoming call, a "consultsWith" relationship is created between the corresponding "Operator" bean and the corresponding "OnlineCustomerStatus" bean. When the operator or the customer disconnects the

call, the "consultsWith" relationship is removed and the system keeps a record of this call. The record is modeled by a "Call History" bean.

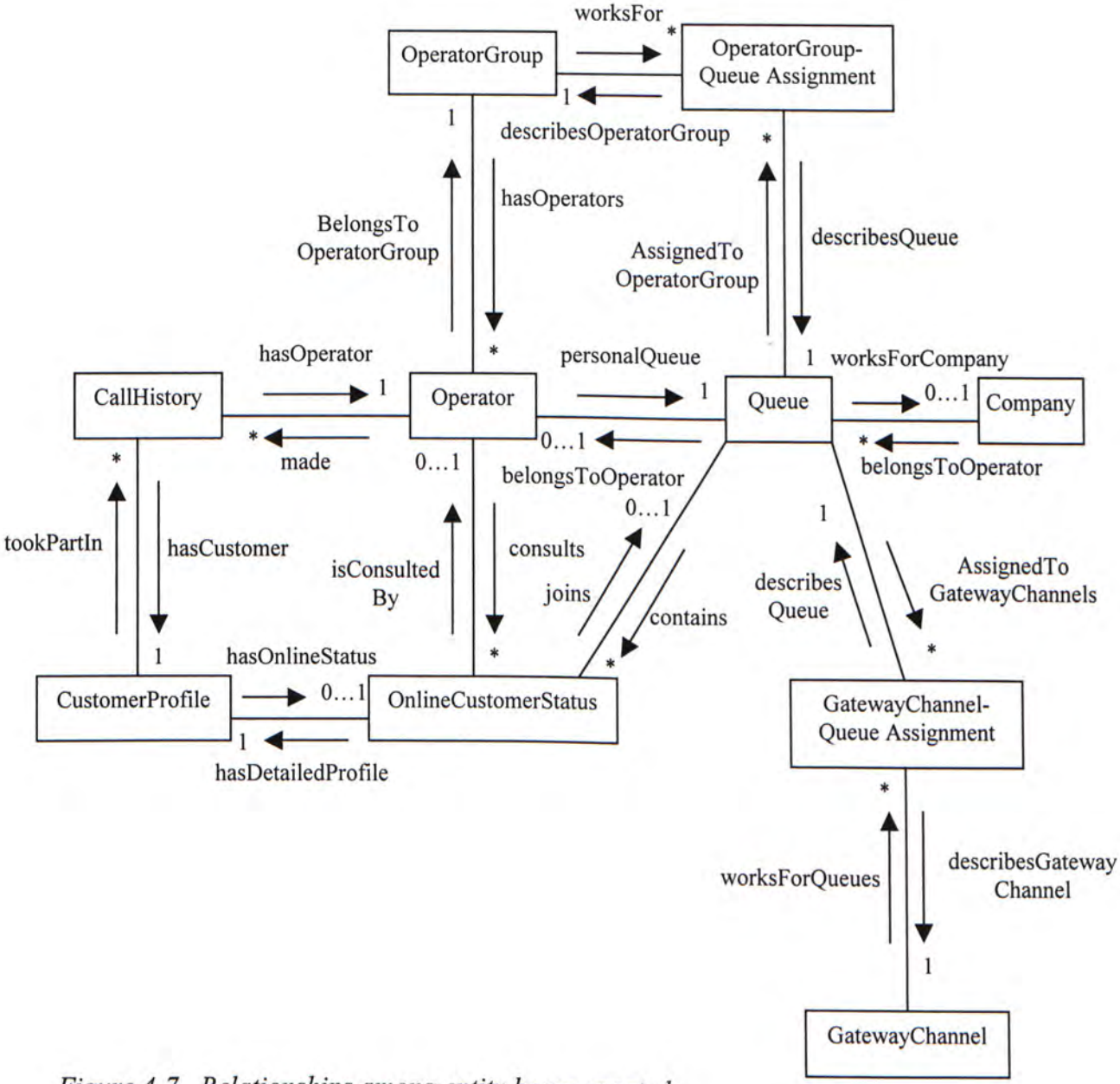



*Figure 4-7 Relationships among entity beans created*

In Section 3.3.4(a), we have mentioned an alternative design in which operator groups are eliminated. In this approach, when assigning an operator to serve a queue, one more entry of information is created in the

"OperatorGroupQueueAssignment" entity. This approach, which largely requires more information stored in the "OperatroGroupQueueAssignment" entity, may increase the time taken to find suitable operators available to answer incoming calls. Due to this reason, this alternative approach is not adopted.

PowerTier for Enterprise JavaBeans provides an Object Builder to assist the creation of entity beans for the implementation. In the Object Builder, the structure of and relationship between beans are described and the mapping of the generated beans to the relational database is specified. Then the Object Builder automatically generates the program code.

### 4.3.3 Design of Session Beans

A session bean represents a logical extension of a client program that runs on the server. It performs operations on behalf of the client. Since there are two types of client (operators and gateways) in the proposed model, two types of session bean are created. Figure 4-8 shows the two session beans and their related methods. For simplicity, parameters and return types associated with each method and some helper methods are not shown in the figure.

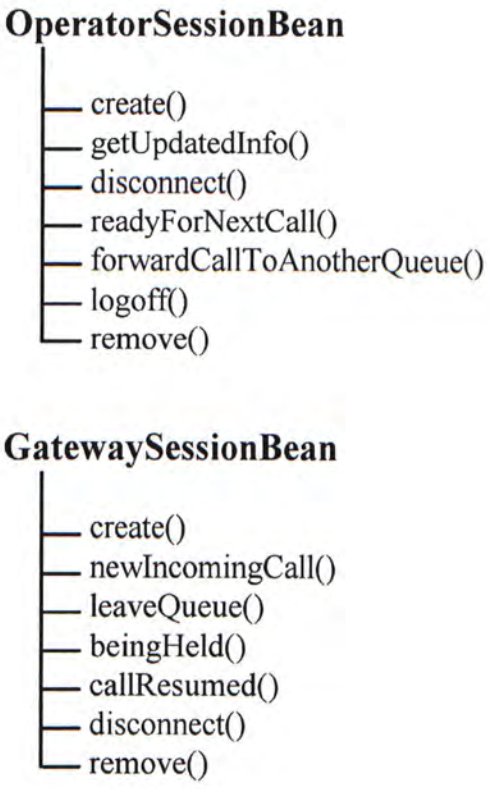



*Figure 4-8 Methods defined in two session beans*

The session bean called "OperatorSessionBean" is used to serve operators. In this session bean, the *create()* method is used for operators to login the system. Operators use *getUpdatedInfo()* method to get updated system information. Operators use the *disconnect()* method to inform the ACD that the current call is disconnected. If an operator becomes available to handle another call after disconnecting a call, the operator can use the *readyForNextCall()* method to ask the ACD to assign another new call to him/her. An operator can use *forwardCallToAnotherQueue()* method to forward his/her current phone call to another operator. When the operator logs off the system, he/she calls the *logoff()* method.

The session bean called "GatewaySessionBean" is used to serve gateways. When a gateway starts up, it creates a session bean by obtaining the bean's home interface and calling a *create()* method. This action informs the ACD that the gateway is ready to work. When the gateway receives a new incoming call from the PSTN, the gateway call the *newIncomingCall()* method to ask the ACD to assign an operator to handle this call. When the caller disconnects the call before an operator is assigned to handle the call, the gateway informs this situation to the ACD by using the *leaveQueue()* method. Two methods, *beingHeld()* and *callResume()*, are used in call-waiting. If the caller disconnects the call which has been handled by an operator, the gateway calls the *disconnect()* method to inform the ACD the call disconnection. The gateway uses the *remove()* method to inform the ACD that the gateway is going to shut down.

### 4.3.4 Asynchronous Method Call

As mentioned previously, the EJB model has no support for asynchronous method calls. If a client calls a remote method, the client is blocked until the remote service finishes processing the call. However, in the proposed call center model, gateways (as a client in the model) need to call remote service in the EJB server asynchronously in some situations. When a gateway informs the EJB server of the arrival of a new incoming call, the gateway may not get an operator to handle this call immediately. The gateway should be free to do other work and obtain the result from the EJB server later.

To tackle this problem, gateways must also act as RMI servers so that the

EJB server can make a remote call to the gateway. This arrangement enables the RMI client-side callbacks to act as an implicit mechanism for clients to call remote methods to the EJB server asynchronously. Figure 4-9 shows how a gateway calls the remote service in the EJB server asynchronously via a RMI client-side callback.

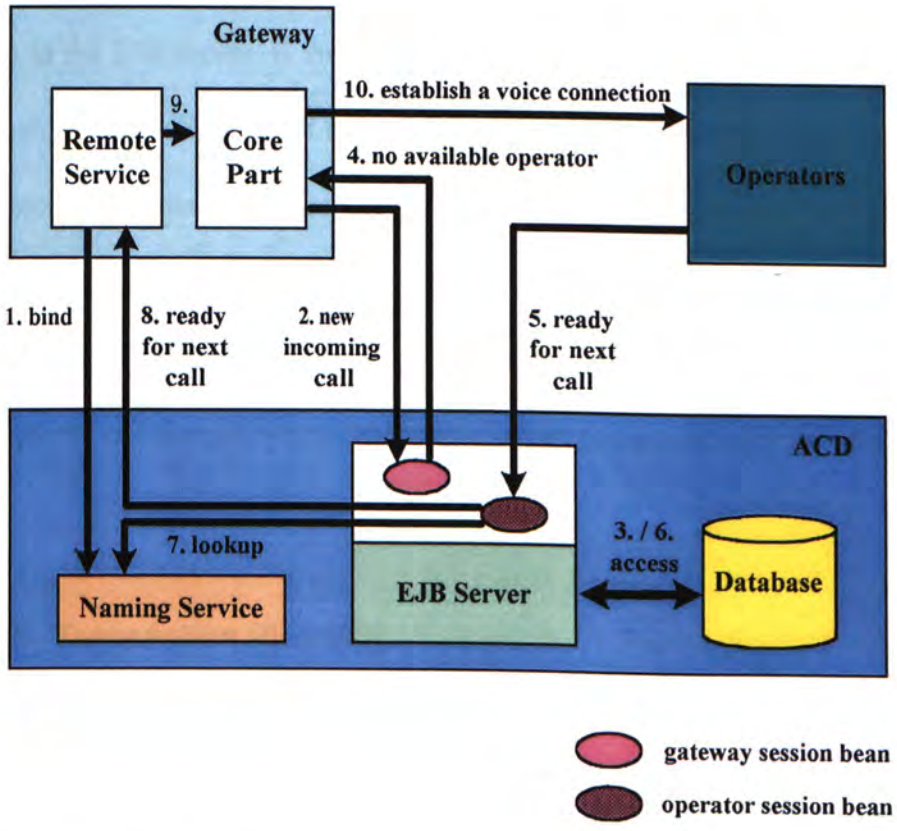



*Figure 4-9 An asynchronous call method achieved via a RMI client-side callback*

To enable the EJB server to locate the remote service in the gateway, the gateway binds its remote service to the naming service in the ACD when the gateway starts. When the gateway receives a new incoming call, the gateway sends a "new incoming call" request to a gateway session bean in the EJB server. The gateway session bean processes the request. If it finds a suitable operator who is currently available to handle this call, the gateway session bean replies to the gateway of this

result immediately. The gateway also establishes a voice connection with the operator assigned subsequently. Otherwise, the gateway session bean replies to the gateway that no suitable operator is available to handle this call at this moment. Then the gateway is free to perform other work. When an operator has just finished another call, the operator sends a "ready for next call" request to an operator session bean in the EJB server. If the operator session bean finds that the operator is suitable to handle the incoming call being held previously, the operator session bean lookups the location of the remote service of the corresponding gateway and calls the "ready to talk" method provided in the remote service of the gateway. As a result, the gateway knows that an operator is available to the call which has been held previously and the gateway establishes a voice connection with the operator.

### 4.3.5 Transaction Planning

The EJB model provides three approaches to demarcating transaction boundaries [33]:

1) Client-managed transactions, where a client makes explicit calls to begin and end each transaction

2) Container-managed transactions, which allow a client to reply on the EJB container to begin and end transactions automatically

3) Bean-managed transactions, where an enterprise bean makes explicit calls to begin and end transactions

The first and third approaches control transactions programmatically, while the second approach controls transactions at the EJB component method-level. The second approach is more preferable in our implementation because the transaction APIs are not written in the application code. This shortens the implementation time and eases the maintenance of the application code. In the second approach, transaction attributes are associated with an entire session or with individual methods on the session bean. Every time a client invokes a method on a bean, the EJB container intercepts the method and delegates the transaction management according to the method's transaction attribute. EJB will automatically commit a transaction if it completes all of its operations without catching any exception. If an exception is caught, the EJB server will rollback the transaction. If the client finds that a rollback transaction, the client will have to retry the transaction. Figure 4-10 briefly summarizes the EJB transaction attributes and their behavior.

| Attribute Type | Description |
|---|---|
| TX_NOT_SUPPORTED | Not transactional. If you call a method with this transaction attribute from within a transaction, the server suspends the transaction until the method returns. |
| TX_BEAN_MANAGED | The EJB container expects the bean to handle the transaction. |
| TX_REQUIRED | The EJB container accepts a call to a bean or method with this attribute within a transactional scope or invokes a new transaction if there is none present. |

| | |
|---|---|
| TX_SUPPORTS | The EJB container does not require a transaction context but allows invocation within a transaction scope. |
| TX_REQUIRES_NEW | The EJB container always invokes in the scope of a new transaction and commits the transaction as soon the call completes. |
| TX_MANDATORY | The EJB container only invokes in the scope of an existing transaction. |

*Figure 4-10 Transaction attributes and their behavior*

In the implementation of the proposed model, all methods in the operator session bean and the gateway session bean are set with the TX_REQUIRES_NEW transaction attribute. This is because a set of operations which one of these methods performs should be committed immediately.

### 4.3.6 Failure Handling

To allow the proposed system to recover relatively quickly from loss of a functional component, single point of failure that will impact operation must be identified and a mechanism to overcome each of the potential failures must be provided. A functional component giving no response until a timeout upon a request is considered to be a failure.

The failure is detected when an exception is caught in calling a RMI method

in the remote object reference representing the failed component. Our focus in this section is the failure handing for VoIP gateways and the ACD.

### a) Failure Handling for VoIP gateways

Operators or the ACD may receive no reply from a VoIP gateway until a timeout upon a request. We consider these as the failure of the VoIP gateway. The failure is detected when an exception is caught in calling a RMI method in the remote object reference representing this gateway. Operators or the ACD reports this failure to the monitoring tool, so the system administrator can take failure handling procedures to recover the system. In the meanwhile, the call center system provides part of service as other gateways can still receive incoming calls. There are some differences in the handling procedure for the failure detected by an operator and by the ACD.

The ACD keeps a list of all active gateways. The system administrator can manipulate the list by calling methods provided in the ACD's remote reference through the monitoring tool. When one of these gateways fails, the system administrator needs to remove the record of this gateway from the list. After the system administrator has replaced the failed gateway with a new one, the system administrator inserts the record of the new gateway into the list.

When the ACD detects that a gateway has failed as shown in Figure 4-11(a), the customer call coming through this gateway is probably in a waiting queue. Since this gateway has failed, the PSTN connection between the customer and this gateway

is probably disconnected. The system needs to update the status of the customer to offline and record this call as a failure call in the call history record. Moreover, the system needs to assign another gateway to callback this customer.

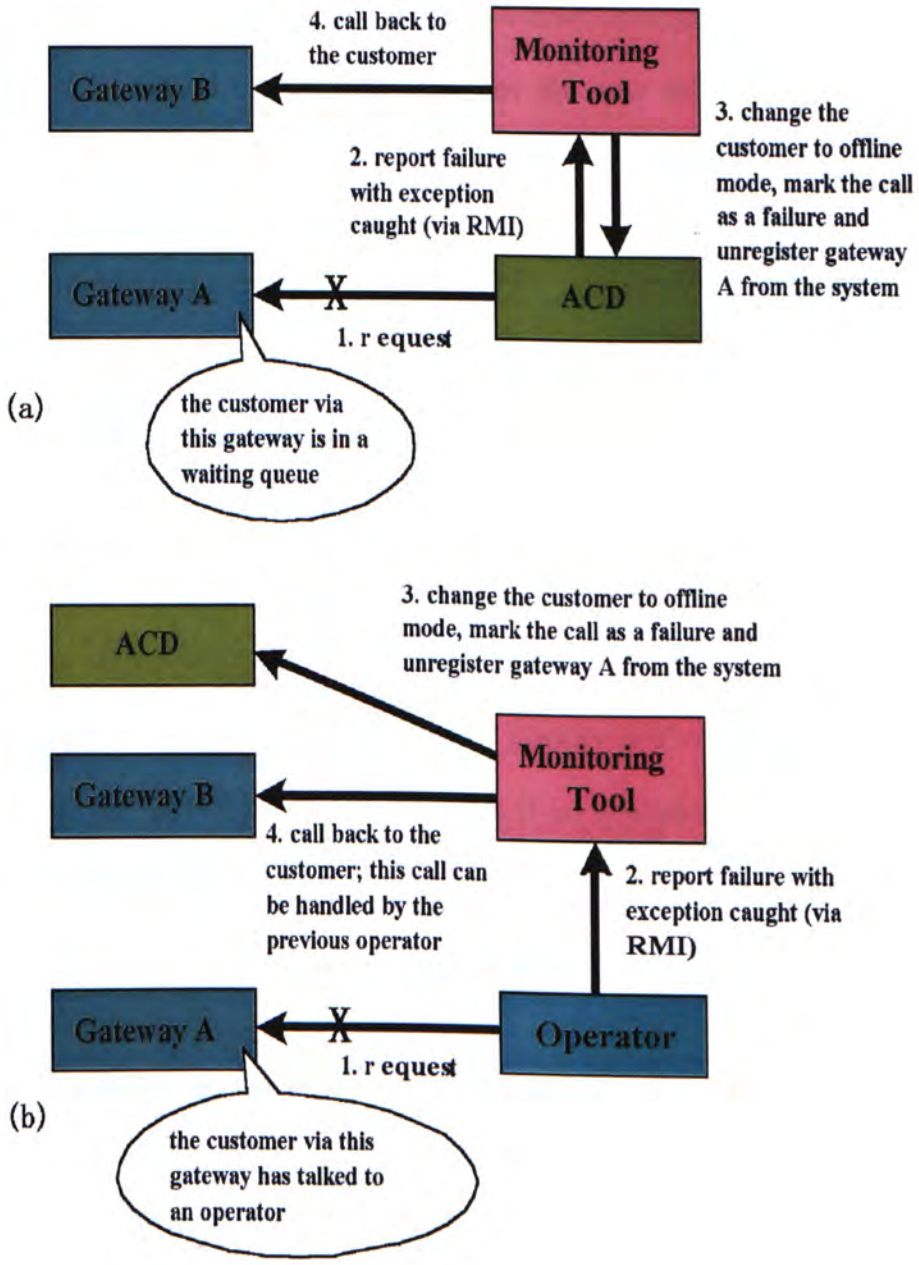



*Figure 4-11 Failure handling for a gateway when (a) the ACD or (b) an operator detects the failure*

When an operator detects that a gateway malfunctions as shown in Figure 4-11(b), the PSTN connection between the customer and this gateway is probably disconnected. The system also needs to update the status of the customer to offline and records this call as a failure in its call history. However, in this case, the customer whose call has come through this gateway probably has talked to an operator. When the system is relinked to this customer through another gateway, the system gives a choice to the customer whether the new call connection is to be handled by a new operator or not.

When a gateway fails, the customer whose call is cut due to the failure is likely to phone to the call centre again. According to the customer's identity and call history, the system can check whether the customer's previous call was cut due to a system failure. If this is the case, the system calls back to this customer through another gateway and this gateway plays back an apology message to the customer. The system gives a choice to the customer whether this new call connection is handled by a previous operator or not. Higher priority is given to this customer so that he/she can get served as soon as possible if no available operator can handle his/her call immediately.

### b) Failure Handling in the ACD

As mentioned previously, the ACD is formed by a naming service, a database service and an EJB server. Therefore, we need to consider the failure handling for these three components in the ACD.

The naming service we use can be regarded as reliable [34]. When operators login the call center system or gateways start up, they go to naming service to retrieve the ACD remote object reference. If the naming service fails, an exception is caught when calling the naming service API. When such a failure is detected, we can simply restart the naming service. It is not necessary for the EJB server to restart and register with the naming server again. This is because the naming service put all registration records in persistence storage and read it on startup. The naming server automatically registers to the naming server when it restarts.

The database server can also be considered as a reliable component as its replication support [35]. The EJB server can lose its connections to the database because of network failures, database server restarts or failovers, or hardware failures. When a database connection is lost, the database client libraries raise exceptions. Alternative database connections which provide continuous data availability can be reestablished with an API provided in the PowerTier EJB server. Restarting the EJB server is not necessary.

The high availability of EJB server can be provided by redundant EJB servers, a set of error detector and resource management. When the EJB server gives no response until a timeout upon a request, we consider this as the failure of the EJB server. The failure is detected when an exception is caught in calling a method in the remote object reference of the EJB server. In the ACD, only one EJB server is active and others are in standby mode. When a primary EJB server fails, this server can be shutdown and a standby server is brought into active mode by registration with the naming service. This new primary server accesses the original database, which still keeps consistent data for the call center system.

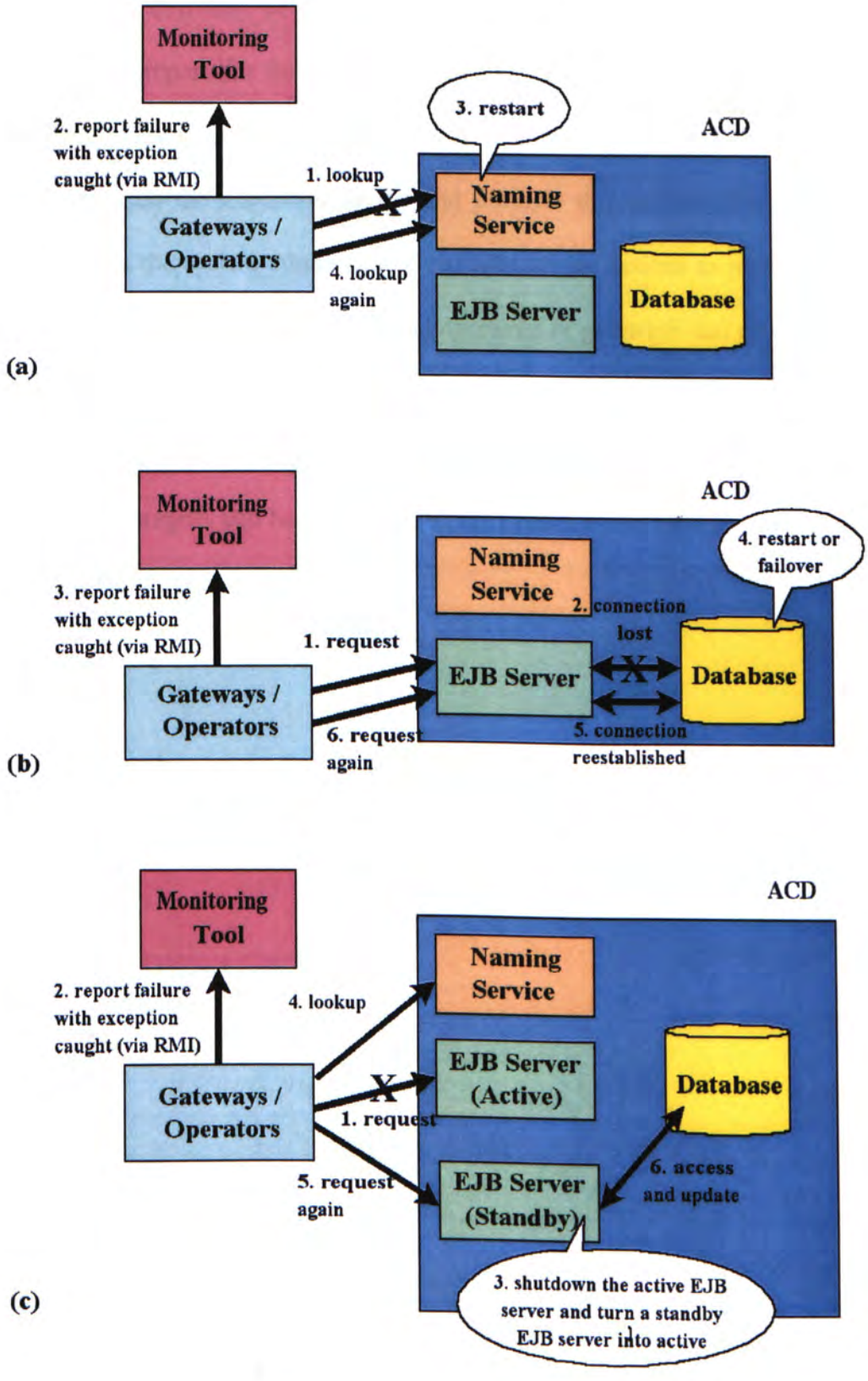



*Figure 4-12 Failure handling for the ACD if (a) the naming service, (b) the database server or (c) the active EJB server fails*

When operators or gateways receive no response from the ACD until a timeout, they report this failure with the type of exception caught to the monitoring tool. The monitoring tool can decide the particular component in the ACD that has failed based on the exception caught and prompts this information to the control screen. Then the system administrator can take proper actions to recover the failed component as shown in Figure 4-12 and operators or gateways can go to the naming service again to get the new ACD's remote reference. With the support of data persistence from the database server and the transaction service provided in the EJB server, data integrity can be guaranteed in the system at the time just before a failure occurs in the ACD.

# 5 An Experiment

This chapter presents an experiment on prototyping the proposed call center model. Some observations on the experiment will also be presented.

## *5.1 Experiment on the Call Center Prototype*

The experiment is conducted to evaluate the performance of the ACD, gateways and operators' software. The evaluation is done by measuring the startup time for the different functional components and the processing time for different requests in the ACD. Due to unavailability of technical information from the PSTN, we are unable to retrieve caller id from VoIP gateway cards up to now. Moreover, there is only one telephone line in our research laboratory, making it impossible to test the feature of call waiting and call forwarding provided in the ACD. Due to these reasons, a simulation program is used as a VoIP gateway in the experiment. This simulation program which is implemented with Java programming is able to send different kinds of requests with different parameters to the ACD.

### 5.1.1 Setup of the Experiment

The setup of the experiment is shown in Figure 5-1.

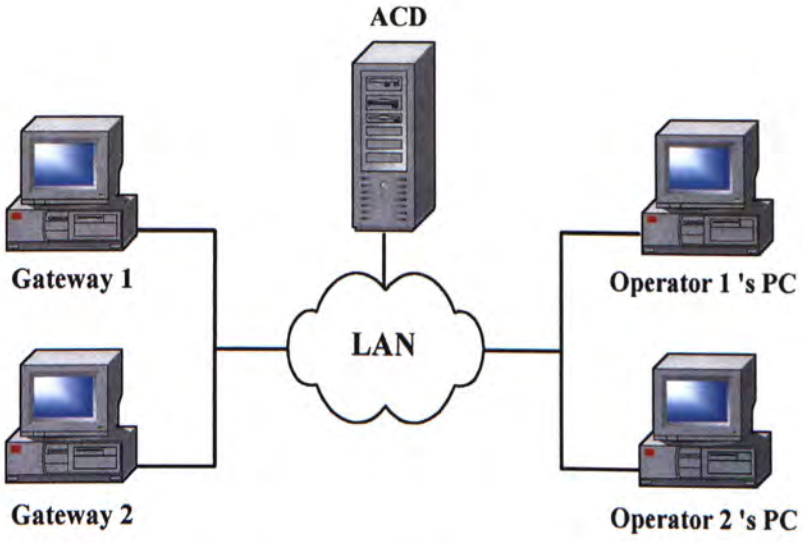



*Figure 5-1 Setup of the experiment*

In the experiment, an ACD and two gateway simulation programs first start up. Then, we start two operators' software to login the call center prototype. These five programs run on five computers. These five computers are connected in a local area network. The configuration of these computers is shown in Figure 5-2.

| IP address | 137.189.89.19 | 137.189.88.161 | 137.189.88.162 | 137.189.88.163 | 137.189.88.164 |
|---|---|---|---|---|---|
| Hardware Model | Pentium III 550MHz | Sun Ultra 5/333 | Sun Ultra 5/333 | Sun Ultra 5/333 | Sun Ultra 5/333 |
| RAM | 512MB | 128MB | 128MB | 128MB | 128MB |
| Plat | Win NT | Solaris 2.6 | Solaris 2.6 | Solaris 2.6 | Solaris 2.6 |
| Program | ACD | Operator 1 | Operator 2 | Gateway 1 | Gateway 2 |

*Figure 5-2 Configuration of computers in the call center prototype*

### 5.1.2 Experimental Results

In the experiment, we firstly measure the startup time for different functional components, including the ACD, gateways and operators' software. Then, the processing time for different requests in the ACD is measured. In this experiment, there are only about 60 entries of customers' profiles and 20 entries of operators' information stored in the database. In a real situation, as the number of customers' profiles and operators' information is greater, the processing time for different requests may increase due to the increase in the searching time on the larger number of customers' profile and operators' information. However, it is expected that the searching time will not greatly increase with the use of indexing techniques.

#### a) Startup Time for Different Components

Startup times for the ACD, a gateway and an operator are recorded. When the ACD starts up, it takes time to load the EJB server and container. When a gateway or an operator starts up, it takes time to locate the EJB server via the naming server and call the *create()* method to create a session bean in the EJB to serve this gateway's or this operator's request.

All the measurements are shown in Figure 5-3.

| Name of functional component | Startup time for the component |
|---|---|
| ACD | 30s |
| Operator | 3.6s |
| Gateway | 3.6s |

*Figure 5-3 Measurements on startup time for different functional compoents*

### b) Possessing Time for Different Requests

When the simulation program sends a "new incoming call" request to the ACD, the average time taken for receiving response from the ACD is recorded. The response is either that an operator is assigned to the gateway or that no appropriate operator is available to answer this incoming call at this moment.

When a call is held by an operator through the feature of call waiting, the gateway sends a "being held" request to the ACD to update the status of this call. The average time taken to finish this request is recorded. When the operator returns to take this call, the corresponding gateway sends a "resume call" request to the ACD to update the status of this call. The average time taken to finish this request is also recorded.

When an operator forwards a call to another operator, the operator sends a "forward to a specific queue" to the ACD to update the status of this call. The average time taken to finish this request is recorded.

When an operator or a caller disconnects their call connection, the operator or the caller sends a "disconnect" request to the ACD to update the status of this call. The average time taken to finish this request is recorded.

When an operator finishes a call and becomes available to receive a new incoming call, the operator sends a "ready for next call" request to the ACD to assign a new incoming call to him/her

All the measurements are summarized in Figure 5-4.

| Name of request | Name of component invoking the request | Average processing time on the request |
|---|---|---|
| New incoming call | Gateway | 700ms |
| Being held | Gateway | 30ms |
| Call resumed | Gateway | 25ms |
| Forward call to a specific queue | Operator | 192ms |
| Disconnect | Gateway | 270ms |
| | Operator | 270ms |
| Ready for next call | Operator | 345ms |

*Figure 5-4 Measurement on average processing time on different requests in the ACD*

## 5.2 Observations

### 5.2.1 Observations on Experimental Results

The ACD takes longer to startup than gateways and operators do. Gateways and operators take a similar time to startup. It is because when the ACD starts up, it needs to start up its naming services and the EJB server. When the EJB server starts up, it takes time to start some of its underlying services. Then the EJB server registers its location with the naming service through JNDI. Gateways and operators take fewer tasks when they start up. They only need to locate the EJB server via the naming service and call *create()* methods to create session beans in the EJB to serve gateways' or operators' requests.

Startup times for gateways and operators are acceptable. Operators take around 4 seconds to login the call center system. Callers probably do not perceive the startup time for gateways because gateways probably have started before callers phone to the call center. Moreover, when a gateway completes an incoming call and receives another new call, the gateway does not need to restart. Although the ACD takes a period of time to startup, operators and gateways probably do not perceive this delay. This is because the ACD probably has started long before operators and gateways start. Moreover, the ACD seldom restarts once it starts.

Processing times for different requests in the ACD are reasonable. The most time-consuming task for the ACD is to process the "new incoming call" request. This is because it needs to search the identity of the caller in the database according the caller id, find operators who have served the caller before in the past call record,

check the current status of operators who are suitable to handle this call and update the current status of the caller. All of these procedures involve searching and updating entries in the database.

The ACD takes less time to process the "being held" request and the "call resumed" request. This is because the EJB server only needs to update the current status of the corresponding users when processing these requests.

When a gateway receives a new incoming call, the customer needs to wait for the ACD to assign an operator to get served. The average waiting time is 700 milliseconds, the processing time of the "new incoming call" request, if there is a suitable operator available to answer this call immediately. The waiting time excludes the time taken to establish VoIP connection between the gateway and the operator assigned. However, in this experiment, there are only about 60 entries of customers' profiles stored in the database. In real situation, as the number of customers' profiles is greater, the processing time for the request may increase due to the increase in the searching time on the larger number of customers' profile.

### 5.2.2 Advantages and Disadvantages of Using EJB

The EJB model facilitates the failure recovery of the ACD in our system by transaction management and persistence service provided in the model. To ensure that our system can keep the processing data and recover quickly in case of any failure in the ACD, all data processed by the ACD are stored persistently in the database server. A row of data stored in a table in the database server is represented

by an entity bean by the EJB model. The entity bean methods provide operations for manipulating the data represented by the bean. Establishing a database connection and accessing the data through SQL statement are not necessary in our programming implementation. Moreover, we need to keep the integrity of the data processed by the ACD after a failure. The EJB server provides transaction implementation in method call level for every operation in the ACD. In case of any failure, the data processed by the ACD can be rollback automatically into a consistent state. After failure handling procedures, data integrity is guaranteed in the system.

Multithreading service provided by the EJB server enables concurrent requests from other functional components. However, resource sharing is not an issue in our programming implementation.

One disadvantage of using the EJB model is that asynchronous method calling is not supported. This increases the complexity in implementing the "new incoming call" requests in the ACD.

Moreover, as the EJB is a relatively new technology, EJB vendors introduce differences in their products to gain proprietary advantage, making it occasionally difficult to learn the technology and to find technical support.

# 6 Conclusions

In this thesis, a three-layer software structure of the Intelligent IP-based Call Center Model has been presented. To test the feasibility of this proposed model, an IP-to-IP call center system which is specifically used in education is implemented. In this IP-to-IP implementation, we identified the importance of system reliability, which becomes one of major concerns in the proposed model. Moreover, we need to ensure that different application modules can be constructed easily on top of the call management layer.

Using the EJB model, we have achieved these two goals in the implementation of the proposed model. Using the EJB model, an ACD is built with a well-defined interface to the application layer. This enables different application modules to be constructed easily on top of the call management layer. The EJB model also facilitates the failure recovery of the ACD in our system by transaction management and persistence service provided in the model. To ensure that the our system can keep the processing data and recover quickly in case of any failure in the ACD, all data processed by the ACD are stored persistently in the database server. The EJB server provides transaction implementation in method call level for every operation in the ACD. In case of any failure, the data processed by the ACD can be rollback automatically into a consistent state. After failure handling procedures, data integrity is guaranteed in the system. This enhances the system availability of the proposed model

Using the EJB model allows us to concentrate on the functional logic of the

ACD and shorten its development time. If we implement the proposed model from scratch with the Java programming language only, we need to write some classes to access the database. When there is any change in the database schema, we need to modify the code in these classes. We also need to deal with the creation of threads in the ACD so as to serve multiple clients simultaneously. To keep the data stored in the ACD consistent, separate and concurrently running threads which share data must consider the state and activities of other threads. This thread synchronization is also handled by us. To handle transactions, we need to call transaction methods explicitly inside the application code. We cannot simply set transactional characteristics at method-level. The EJB model also has its disadvantages in the implementation as discussed in Section 5.2.2.

There are some weaknesses in the proposed model. Although failures of VoIP gateways and the ACD are handled by our application model, these mechanisms involve manual decisions and replacements by the system administrator. Failures of VoIP gateways are detected if exceptions are caught when making requests to these distributed components. Unfortunately, automatic failure recovery mechanism for VoIP gateways cannot be implemented due to the lack of APIs to access the hardware status of VoIP gateways. Thus, we are unable to know whether the exception caught is due to unrecoverable failure or not. To simplify the implementation of the model, exceptions caught when making requests to the ACD are not categorized in detail. This requires the current implementation of the system to involve manual decisions to be made by the system administrator when recovering the failed components. To improve the performance of our system, these exceptions caught should be further categorized. In the ACD, when a database connection is lost,

the database client libraries raise exceptions. These exceptions vary across the different database vendors. This creates difficulty to use another database server from a different vendor.

In the experiment, we have found that the overall performance of different functional components is satisfactory. When a gateway receives a new incoming call, the customer needs to wait for 700 milliseconds, the processing time of the "new incoming call" request, if there is a suitable operator available to answer this call immediately. Preliminary experimental results show that it is feasible to implement user authentication, call routing, call establishment and some advanced call features such as call waiting and call forwarding (services provided in an Automatic Call Distributor) as software on top of a PC connected to an IP network.

However, due to unavailability of technical information from the PSTN, we are unable to retrieve caller ID from VoIP gateway cards. Moreover, there is only one telephone line in our research laboratory. In the experiment, the number of customers' profiles and operators' information stored in the database is smaller than those in the real situation. The absence of such information may limit the integrity of the performance evaluation to a minor degree.

Although an experimental ACD is implemented and its performance is evaluated, we still do not have a complete call center prototype. This thesis presents only a part of the research in our VoIP research group. Our other two group members are working on voice transmission and collaborative browsing respectively. By combining our three persons' research work, a complete intelligent IP-based call center can be created. Moreover, we need to package and modularize the prototype

so as to decrease the configuration time taken to adopt the proposed model to suit the needs of any specific industry.

# Bibliography


[1] ITU-T H. Series, "H.323 – Line Transmission of Non-Telephone Signals – Visual Telephone Systems and Equipment for Local Area Networks which Provide a Non-Guaranteed Quality of Service," May 28, 1996.

[2] M. Handley et al., "SIP: Session Initiation Protocol," IETF RFC 2543, March 1999.

[3] H. Schulzrinne, S. Casner, R. Frederick and V. Jacobson, IETF Audio-Video Transport Working Group RFC 1889, "RTP: A Transport Protocol for Real-Time Applications," January 1996.

[4] Vanderbilt University, "Automatic Call Distribution", http://www.vanderbilt.edu/telecom/automati.htm.

[5] M. Meltzer, "Integrating a Call Center with Customer Information", CRM Forum Resources, 2000.

[6] Y.S. Moon, C.C. Leung, K.N. Yuen, H.C. Ho, X. Yu, "A CRM Model Based on Voice Over IP", IEEE Canadian Conference on Electrical and Computer Engineering, 2000, vol 1, pp 464 –468.

[7] H.C. Ho, "Customer Relationship Management (CRM) by Computer Telephony Integration (CTI) -Collaborative Browsing", Internal Report, Dept. of Comp. Sci. & Eng., The Chinese University of Hong Kong, December 2000.

[8] G. Held, Voice Over Data Networks, McGraw-Hill, 1998.

[9] K.N. Yuen, H.C. Ho, C.C. Leung, Y.S. Moon, "Voice-Over-IP Application in Education – VoIP enabled web lecture", GCCCE2000, vol 1, pp 1023.

[10] K.N. Yuen, "A Comprehensive VoIP System with PSTN Connectivity", MPhil Thesis, Dept. of Comp. Sci. & Eng., The Chinese University of Hong Kong, July 2001.

[11] H.C. Ho, "Three-Tier Feature-based Collaborative Browsing for Computer Telephony", MPhil Thesis, Dept. of Comp. Sci. & Eng., The Chinese University of Hong Kong, July 2001.

[12] Sun Microsystems, Inc., "Java™ Remote Method Invocation", http://java.sun.com/products/jdk/rmi/.

[13] Sun Microsystems, Inc., "JDBC™ Technology", http://java.sun.com/products/jdbc/.

[14] K.N. Yuen, "Studies on the Implementation of a VoIP System in a PSTN – Intranet Environment", Internal Report, Dept. of Comp. Sci. & Eng., The Chinese University of Hong Kong, December 2000.

[15] E. Jorg, J.V. Hans, "GSM Switching, Services and Protocols", John Wiley & Sons, 1999, pp 89.

[16] J.Postel, "Transmission Control Protocol", IETF RFC 793, 1981.

[17] Sun Microsystems, Inc., "Enterprise JavaBeans™ Technology", http://www.javasoft.com/products/ejb/index.html.

[18] Microsoft Corporation, "COM+ - Papers, Presentations, Media Coverage, and Resources for Microsoft COM+ Technology", http://www.microsoft.com/com/tech/COMPlus.asp.

[19] IDevResource.com, "What is Async COM? ", http://www.idevresource.com/com/library/articles/async.asp.

[20] Liaoning University, "Tutorial on Distributed Objects", http://www.lnu.edu.cn/corba/c1/c3.html.

[21] Sun Microsystems, Inc., "Java Naming and Directory Interface™ (JNDI)", http://java.sun.com/products/jndi/.

[22] Microsoft Corporation, "Microsoft COM Technologies - Information and Resources for the Component Object Model-based technologies", http://www.microsoft.com/com/default.asp.

[23] Microsoft Corporation, "COM Technologies: Microsoft Transaction Server (MTS)", http://www.microsoft.com/com/tech/MTS.asp.

[24] Microsoft Corporation, "Active Directory Overview", http://www.microsoft.com/windows2000/server/evaluation/features/dirlist.asp.

[25] Microsoft Corporation, "Distributed Component Object Model (DCOM)", http://www.microsoft.com/com/tech/DCOM.asp.

[26] Microsoft Corporation, "Microsoft ActiveX Data Objects (ADO)", http://www.microsoft.com/data/ado/default.htm.

[27] Microsoft Corporation, "Microsoft OLE DB", http://www.microsoft.com/data/oledb/default.htm.

[28] Microsoft Corporation, "Microsoft ODBC", http://www.microsoft.com/data/odbc/default.htm.

[29] A. Thomas, "Comparing MTS and EJB", Distributed Computing, December 1998, http://java.sun.com/products/ejb/pdf/9812MTSEJB.pdf.

[30] Sun Microsystems, Inc., "Java™ Transaction API (JTA)", http://java.sun.com/products/jta/.

[31] Microsoft Corporation, "OLE Transactions Reference", http://msdn.microsoft.com/library/default.asp?URL=/library/psdk/cossdk/pgoleref_5uan.htm.

[32] Persistence Software, Inc., "Persistence Software: Powertier", http://www.persistence.com/powertier/index.html.

[33] Microsoft Corporation, "Comparing Microsoft Transaction Server (MTS) to Enterprise JavaBeans", http://www.microsoft.com/com/wpaper/mts-ejb.asp.

[34] Persistence Software Inc., "PowerTier for Enterprise JavaBeans Programming Guide", Version 6.0, March 2000.

[35] Oracle Corp., "Oracle8(TM) Server Replication Release 8.0", June 1997, http://www.oradoc.com/ora8doc/DOC/server803/A54651_01/toc.htm.