DIGITAL SIGNALING PROCESSOR RESOURCE MANAGEMENT FOR
SMALL OFFICE PHONE SYSTEMS

A Thesis
presented to
the Faculty of California Polytechnic State University
San Luis Obispo

In Partial Fulfillment
of the Requirements for the Degree
Master of Science in Computer Science

by
J.T. Gilkeson
June 2010

© 2010

J.T. Gilkeson

ALL RIGHTS RESERVED

COMMITTEE MEMBERSHIP

TITLE: Digital Signaling Processor Resource Management
for Small Office Phone Systems

AUTHOR: J.T. Gilkeson

DATE SUBMITTED: June 2010

COMMITTEE CHAIR: Alexander Dekhtyar, Ph.D.

COMMITTEE MEMBER: Gene Fisher, Ph.D.

COMMITTEE MEMBER: Clark Turner, J.D., Ph.D.

ABSTRACT

Digital Signaling Processor Resource Management for Small Office Phone Systems

J.T. Gilkeson

Contemporary small office phone systems are specialized computers that connect a variety of phones within the office and to the local phone company. These systems use digital signaling processors (DSPs) to convert signals from analog to digital and vice-versa. Many different types of applications run on the DSPs and different businesses have varying application needs. Given the systems have limited amounts of DSP resources and growing numbers of applications for a phone system, an administrator needs a way to configure the uses of resources based on their individual business needs.

This thesis provides an overview of a system for configuring resources on various types of DSP hardware some of which are removable and have differing tradeoffs between application uses. The system has to be able to change resource allocations while the phone system is running with minimal interruptions to calls. The configuration system needs to be designed to be flexible enough that new applications or DSP hardware could be supported without major changes to code.

This thesis presents a system that uses a database-driven model along with algorithms that optimize configuration of DSP hardware given the administrator's individual application needs.

ACKNOWLEDGEMENTS

Many thanks to Dr. Alex Dekhtyar for help throughout this process. His help organizing the paper and presentation as well as being willing to meet me at off hours were instrumental in this thesis being completed.

Additional thanks go to Dr. Clark Turner and Dr. Gene Fisher for serving on my committee.

Thanks to the company I work for, who shall remain anonymous, for permission to use my work for this thesis.

One parting thought: when people warn you not to leave school without completing your thesis – don't ignore that advice! Doing a thesis on nights and weekends a decade after completing your coursework is not something I can recommend!

TABLE OF CONTENTS

LIST OF TABLES.....	xi
LIST OF FIGURES	xii
1. Introduction.....	1
1.1 Overview.....	1
1.2 Contributions.....	2
1.3 Road Map.....	4
2. Background.....	5
2.1 Introduction to PBXs and DSPs.....	5
2.2 Resource Allocation with Uniform Hardware	10
2.3 New challenges for Resource Allocation.....	13
3. Solution Overview	16
3.1 Requirements	16
3.1.1 Purpose.....	16
3.1.2 Client / User Interface Requirements.....	16
3.1.3 Server Requirements.....	18
3.2 Architecture.....	19
3.3 Components	21
3.3.1 Hardware and Drivers.....	21
3.3.2 Web-based Administration	21
3.3.3 Resource Management Applet.....	21
3.3.4 Database.....	21

3.3.5	CDataRep Service	22
3.3.6	TAPI and Applications	22
3.3.7	ResourceManager Module	23
4.	Database Design	24
4.1	Entity Relationship.....	24
4.2	Tables	25
4.2.1	Resource Categories (Read-Only)	25
4.2.2	Applications	26
4.2.3	Dsp Types (Read-Only)	27
4.2.4	Dsp	28
4.2.5	Current Dsp Resource Configuration.....	29
4.2.6	Application Group Mapping	30
4.2.7	PbxExtension, SysPort, and FaxPort	30
4.2.8	Advanced Settings	32
4.3	Data	33
4.3.1	Tables with Read-Only Data.....	33
4.3.2	Dynamic Data Tables.....	34
5.	Client / Optimizer	35
5.1	Graphical User Interface (GUI)	35
5.1.1	Main Applet	35
5.1.2	Advanced Settings Dialog.....	38
5.1.3	Making and Applying Changes	38
5.1.4	Bad States.....	39

5.2	Class Design.....	40
5.2.1	Main classes	42
5.2.2	Supporting Classes & Interfaces	44
5.3	Optimization Algorithms	45
5.3.1	reassignPorts	45
5.3.2	recursiveReassignPorts	45
6.	Allocator / Server	48
6.1	Flow Charts	48
6.1.1	Applet Requests Data.....	49
6.1.2	Applet Saves Data.....	50
6.2	CDataRep Service	50
6.2.1	API Methods	51
6.2.2	Bootup.....	51
6.2.3	Assignment Algorithms	52
6.3	ResourceManager library	57
6.3.1	Legacy DSPs	58
6.3.2	PTMC Module	58
7.	Evaluation	59
7.1	Experiments	59
7.1.1	Configuring a basic system.....	60
7.1.2	Adding Resources on a running system.....	62
7.1.3	Reallocating Resources on a running system.....	62
7.1.4	Scalability	63

7.2	Requirements Met.....	65
7.3	Field Deployment / Customer Feedback.....	66
8.	Conclusion / Future Work.....	67
8.1	Future Work.....	69
8.1.1	The Rise of VoIP	69
8.1.2	Voice Recognition	70
8.1.3	Less Reservations and More Dynamic Pools of Resources.....	70
8.1.4	Faster Servers and Thin Clients	70
	Bibliography	72
	Appendix A: Glossary	74

LIST OF TABLES

Table 1: ResourceCatID.....	25
Table 2: ResourceAppID	26
Table 3: DspType.....	27
Table 4: Dsp.....	28
Table 5: DspResource.....	29
Table 6: AppGroupMap.....	30
Table 7: PbxExtension	31
Table 8: SysPort.....	31
Table 9: FaxPort.....	32
Table 10: DspResourceAdvancedSetting	33
Tables 11 & 12: Basic Configuration Old System and New System	61
Table 13: Initial Configuration Results.....	61
Tables 14 & 15: Add Resources Old System and New System	62
Table 16: Adding Resources Results	62
Tables 17 & 18: Reallocate Resources Old System and New System	63
Table 19: Reallocating Resources Results.....	63
Table 20: Scalability New System.....	64
Table 21: Scalability Results	64

LIST OF FIGURES

Figure 1: Resource Distribution Tab.....	11
Figure 2: Resource Management Tab	12
Figure 3: Resource Allocation Dialog	13
Figure 4: Resource Management Interactions Diagram	20
Figure 5: Entity Relationship Diagram	24
Figure 6: Resource Management Applet	36
Figure 7: Advanced Settings Dialog	38
Figure 8: Missing Hardware Error Dialog	39
Figure 9: Applet Class Design	41
Figure 10: reassignPorts Algorithm	45
Figure 11: recursiveReassignPorts Algorithm	46
Figure 12: Client Loading Process.....	49
Figure 13: Client Saving Process.....	50
Figure 14: VerifyDspTables Algorithm.....	52
Figure 15: ReassignDspResources Algorithm	54
Figure 16: RecursiveReassignDspResources Algorithm	55
Figure 17: UpdateDatabaseTables - Part 1	56
Figure 18: UpdateDatabaseTables - Part 2	57

1. Introduction

1.1 Overview

The modern small to medium business or large enterprise with retail chain stores often has onsite a standalone phone system called a Private Branch Exchange, or PBX (see Appendix A for a glossary of terms and acronyms used in this thesis). This PBX routes people's calls, handles common queries using an automated attendant, processes requests such as accessing account information or automating an order, allows the caller to leave voicemail, pages people, receives faxes, or routes outgoing calls using the most cost-effective channels of communication. These systems have limited Digital Signaling Processor (DSP) resources to run these many different types of applications. Given the limited resources and growing number of applications for a PBX, the administrator needs a way to configure the uses of resources based on their individual business needs.

In earlier versions of PBXs, the number of DSP resources was often fixed. The system had either some or all of the resources reserved for basic system functionality. For PBXs that allowed configuration of the resources not dedicated to system functionality such as earlier versions of the PBX the author worked on, the user could specify in a fairly straightforward manner which resources to allocate to individual applications. Due to the relatively small number of applications and the rarity of needing to change the allocation, the burden of having to re-allocate the majority of the resources (which meant all the applications using these resources for calls would need to drop the calls) and possibly reboot when making changes was an acceptable implementation.

As small businesses began to run an increasing number and scale of applications on their PBXs and the need for 24 hours a day / 7 days a week operation became more common, the prior limitations in both resources and configuration were no longer acceptable. Depending on the type of business, the application needs could vary by vast amounts.

In order to accommodate these changing needs, the PBX the author worked on would now need to offer the ability to support modular DSP resources that could be added or removed from the system. The new modules had different application tradeoffs when compared to the basic built in DSP resources. As the number of applications increased and the likelihood adjustments would need to be made to allocation of resources, the need to be able to make changes to the allocation on the fly (without interrupting all the applications) became a requirement. Given that not all businesses would need all of the applications, licensing of applications and enforcement through resource configuration was needed. Additionally, now that the increased role of applications had become apparent, it was clear that whatever system that was to be designed to configure these resources would need to be flexible enough to accommodate new applications and DSP modules without requiring major code changes.

1.2 Contributions

This thesis describes the system created to address the challenges posed in the previous section. Specifically, details of the architecture, algorithms, and data modeling are presented to demonstrate the flexibility and efficiency of the system the author created. Unlike the previous configuration system, which used static data and exposed the user directly to the hardware, this system is designed to be dynamic and focuses on

the applications the user wants to run while abstracting the hardware. The primary contributions of this work are:

- Modeling of the hardware, applications, constraints, and configuration of the system in a database. This modeling approach organizes the applications, provides the basis of rules for configuration, keeps track of the current state of assignment, and maps the assigned resources into the PBX data structures.
- A database-driven configuration client that optimizes remaining resources as the user makes changes. The client does not require code changes to handle new hardware, applications, or resources constraint changes. The optimizer abstracts the hardware so that the user can see what the tradeoffs are of assigning different applications without having to understand all the different permutations of the hardware assignment that are being evaluated.
- A server-side allocator that is responsible for keeping the database updated at system boot with any hardware changes, and during runtime when configuration changes are applied from the client. The allocator processes configuration changes using a similar algorithm to the client but with the goal of finding the first fit solution for the hardware. This first fit approach reduces the amount of processing needed to apply the assignment, which is significant for the server that is processing calls and running applications. The allocator is also responsible for notifying applications and drivers of changes. The allocator attempts to minimize the number of changes made to hardware, which in turn reduces the number of notifications and possible interruptions to calls needed.

1.3 Road Map

Chapter 2 begins by reviewing background information on PBXs and DSPs, as well as how this project relates to previous work, and why this new approach was needed. Next, Chapter 3 presents an overview of the solution. This overview includes requirements for the project, a look at the system architecture, and a description of the components of the system.

Chapter 4 details the database design for the hardware, applications, configuration, and supporting PBX data. The relationships, table structure, constraints, and data are described.

The client / optimizer is introduced in Chapter 5. We explore the interface and how a user interacts with it. The class design is presented in detail. This is followed by descriptions of the data model and optimization algorithms.

Chapter 6 covers the server-side methods and libraries. Diagrams show how the client and server interact to modify the saved configuration.

Chapter 7 is an evaluation that looks at experiments that demonstrate the differences between the old and new system, reviews the requirements and how they were met, and provides feedback from the deployment of the system.

Chapter 8 reviews related work in the field of algorithms.

The focus of the final chapter of this thesis is the conclusion on how well the solution presented dealt with the issues it was addressing, and the contributions that the author made.

2. Background

2.1 Introduction to PBXs and DSPs

The Public Switched Telephone Network (PSTN) is made up of many Central Offices (COs) owned by phone companies that connect individual home and business phone lines from a local area to the network of Central Offices all over the world. In the early days the COs used human operators to connect calls using a switchboard. This switchboard where operators manually connected phone lines between parties was also known as an exchange. As technology advanced and cost saving could be realized by having machines connect calls, COs began to provide basic features like dial tone, dialing, and ring signaling [3]. These features allowed both callers and machines to be able to interact with each other.

A telephone exchange serving an individual organization and having connections to a public telephone exchange is called a Private Branch Exchange (PBX) [9]. The earliest use of the term PBX can be traced back to large companies or hotels that employed switchboard operators to connect calls between local phones and to the outside world. Employing people to connect calls inside an office was obviously cost-prohibitive for small businesses. As home telephone use grew, small businesses needed be able to handle multiple calls without a switchboard – telephone equipment providers responded to this need and created the first small business phone systems.

The earliest small office phone systems were called “key systems” which involved analog phones that had physical push buttons that would light up when a call was on the line. These phone has buttons represented a direct connection (called a trunk) to the

outside PSTN. The phone would ring and one of the buttons would be flashing to indicate an incoming unanswered call, the user would answer the call by pushing in the button and picking up the receiver. To make an outgoing call, the user would push in an unlit button corresponding to a line that was not being used. The phone also had a hold button, which would allow the user to switch lines without disconnecting the last call. The phone usually had an intercom button that was hooked up to a system that would allow calls between phones or to an overhead paging system. The key system was very simple to understand, maintain, and operate, but it also was very limited. Each phone outlet needed to be planned carefully with a large ribbon of phone lines running to it. In order to coordinate answering calls or to transfer a call from one person to another, users would use the intercom to announce to other people that “Bob you have a call on line 4” – some grocery stores still use these types of systems.

As the nation became more connected and the number of telephones in home continued to increase, Bell Labs researched ways to put more calls over fewer telephone lines between locations. In the 1960s, the T-1 was first put into use between COs. The T-1 was a 4-wire phone line that used Pulse Code Modulation (PCM) and Time-Division Multiplexing (TDM) to transmit 24 calls over a single phone line. PCM is a process of encoding analog sound into a digital representation. TDM refers to the process of taking a high-speed digital stream and dividing up the single stream into multiple channels each carrying their own data, in this case each channel represented a phone call. A channel bank sliced a 1.544 Mbit/s digital signal into 8,000 separate frames, each composed of 24 contiguous bytes. Each byte represented a single telephone call encoded into a constant

bit rate signal of 64 Kbit/s. Channel banks used a byte's fixed position (temporal alignment) in the frame to determine which call it belonged to [8].

As phone system manufacturers began to use computers inside their systems, they used this same TDM technology to develop proprietary digital phones. Unlike the key system phones, which required separate wires for each line, digital phones could be wired using a single phone line, which reduced the cost and complexity of setting up a private phone system. Analog telephones use in-band signaling; meaning they send tones (e.g. dialed digits) and sense commands (e.g. ring signals) directly on each individual phone line. A digital phone uses out-of-band signaling over to send call control information (the phone's identification and commands) on a separate channel between itself and the on-site phone system [6]. Many of these systems still used line keys to represent individual trunks; however, each phone had a unique number called an extension. This meant that the on-site phone system could connect a call from a trunk to a phone – the phone no longer needed to have connections to all the trunks. Additionally, the on-site phone system could now intelligently connect calls within the store without using outside trunks or an intercom. The system could restrict calls being made based on the phone being used such as no long distance calls for certain phones. For incoming calls, the phone system could now prevent a second extension from picking up the same call. Calls could now be transferred to other extensions by dialing the number instead of announcing it over the loud speaker. These hybrid systems provided the transition between traditional key systems and a PBX [5].

As companies and their phone systems grew, it became impractical to continue to have phones with buttons representing each trunk. In response to this issue, the modern

PBX was born. In a PBX, the connecting of calls from trunks to extensions and vice versa is done inside the on-site phone system. To dial the outside world, one just picks up an extension and dials the number – the phone system takes care of finding an unused trunk to connect the call. In order to make phone calls within the PBX work, phone systems now needed Digital Signal Processors (DSPs), which could play and decode all of the standard dual tone multi-frequency (DTMF) tones that the phone company used to communicate the state of the phone and detect the buttons being dialed.

The concept of connecting extensions to each other or to trunks automatically is called circuit switching. One of the advantages of circuit switching is that multiple types of extensions can be used seamlessly. This means that normal analog phones, digital office phones, fax machines, and modems can all be plugged into the same PBX. The connection between the PBX and the CO could now be a mix of analog trunks and T-1 trunks.

These early full-featured PBXs were expensive items, only large businesses could typically afford to purchase and maintain them. The COs saw the need to provide this functionality to smaller businesses and developed a service called Centrex (Central Exchange). The Centrex concept was that the phone company would provide PBX-like features over normal phone lines to a company. The small business wouldn't have to have a big expensive PBX on site in order to have their business take advantage of productivity and cost saving that a PBX offered. Centrex developed many innovations such as linking multiple offices in different locations together and being able to dial an extension at other offices without having to dial the whole phone number. Forwarding and transferring calls between extensions became possible [12]. Centrex did have some

downsides though. Any time the company needed to make changes to their setup, they had to go through the phone company. Additionally recurring monthly fees were charged for each service and the number of extensions, as a company grew they added up. Moves, adds, and changes often required the CO to run more lines to the office which took more time and money. Eventually PBX manufacturers were able to make smaller entry-level PBXs that offered the customer more control and more features without subscription fees [11].

As PBXs advanced, the demand for applications increased. Companies wanted an automated operator (Auto Attendant) to connect calls to the desired individual department or extension. Taking a message if the business was closed, if the phone was busy or no one answered after a set number of rings at an extension became a standard feature. Often these functions were sold as separate add-on systems to the PBX. Applications like these, that involve playing audio prompts and detecting input from the user, use DSPs to do the work.

As computer networks began to connect multiple remote offices together and electronic mail became more prevalent, PBXs began to support sending office-to-office calls by converting the analog voice communications to data packets and sending them using Internet Protocol (IP) over these private networks. This provided tremendous cost savings by using the company's data infrastructure instead of sending these calls over PSTN trunks, which incurred high costs for long-distance calls [1]. The connection of calls using data packets is called packet switching, but unlike circuit switched networks that create dedicated connections between endpoints, the connection for packet switching is not dedicated and can degrade if the network is congested as there is no guarantee that

an IP packet will successfully reach its destination [13]. An advantage of using IP packet switching as opposed to TDM switching is that any unused bandwidth is shared, which is much more efficient than dedicated time-slicing [7]. This was the beginning of Voice Over IP (VoIP). Eventually personal computers and phones would support VoIP directly. Converting analog voice communications to data packets and vice versa is very DSP intensive.

2.2 Resource Allocation with Uniform Hardware

The initial versions of the PBX that the author worked on had a small fixed number of DSPs resources, which were of a uniform type. The configuration system allowed the user to manage the DSP devices as well as the individual resources per device.

The PBX had two main uses for DSP resources: telephony applications and VoIP (referred to as IP Telephony). Telephony Application Programming Interface (TAPI) is a Microsoft Windows API, which provides computer telephony integration and enables PCs running Microsoft Windows to use telephone services [15]. The telephony applications on this PBX used TAPI to interface with the PBX.

An individual DSP device could be configured to be dedicated to either TAPI or IP Telephony. DSP devices that were allocated to TAPI had individual TAPI resources (System Ports) that could be assigned to a limited number of applications.

One of the DSP devices was permanently dedicated to TAPI and could support 10 TAPI Applications. The remaining two DSP devices were individually configurable to be used for either 12 TAPI resources or 4 IP Telephony channels.

The GUI application to make these assignments consisted of a dialog box with two tabs, one for Resource Distribution, and another for Resource Management. The administrator could change the various assignments, and apply them when satisfied.

The Resource Distribution tab (Figure 1) contained a set of radio buttons representing the various scenarios for DSP assignment.

Figure 1: Resource Distribution Tab

The choices correspond to either both configurable DSP devices being dedicated to TAPI, one of the devices being dedicated to TAPI and one to IP Telephony, or both of the devices being dedicated to IP Telephony.

The Resource Management tab (Figure 2) consisted of a tree structure that contained the various TAPI applications grouped into categories. It also showed which individual System Ports were assigned to these applications.

Figure 2: Resource Management Tab

Double-clicking on a System Port allowed the administrator to re-assign the port to a different application or the free pool (a category for unassigned resources).

Figure 3: Resource Allocation Dialog

The icons next to the System Ports represented their pending assignment, with an exclamation point indicating the resource assignment has been modified and a red circle indicating that the resources has been de-allocated and are no longer available for TAPI applications.

When the user would change a DSP device from TAPI to IP Telephony, 12 of the System Ports would be de-allocated based on which ports corresponded to the DSP being changed. This meant that the administrator would have to carefully review the de-allocated ports and re-assign the remaining ports to re-do their desired configuration. This also meant when the changes were applied to the server – all the applications that were using the previously configured TAPI ports would have to drop those ports.

2.3 New challenges for Resource Allocation

As the demands for applications grew it necessitated that the number of DSP devices that were needed to support those applications be increased dramatically. The new version of the PBX would now need to support modular DSP resources that could be adjusted to meet the needs of individual customers.

The increased total possible number of DSPs available in a system made the previous way of allocating resources no longer practical. Additionally, the new modular DSPs had different capabilities and tradeoffs as compared to built-in DSPs. Administrators didn't really want to configure individual pieces of DSP hardware and assign out each port one-by-one, this would be a tedious and time consuming process as the number of these resources increased. What administrators really cared about was: how many resources are available, and how many are assigned to each application.

Additionally, many customers now operated stores that were open (or needed to process phone calls) on a 24 hour / 7 days a week basis. If minor changes needed to be made to DSP resources, it was no longer acceptable to disturb a large amount of applications or to require a reboot of the system. The new system would need to be robust and coordinate changes in a way that minimized disruptions.

Some of the new applications being developed for the PBX were considered premium offerings – meaning that they would not be needed by all customers and could be sold as optional add-ons. In order to enforce this policy, a licensing mechanism was developed whereby a customer could buy and install a software license to enable individual premium applications. Some licenses were port-based; meaning that the license would restrict how many calls the application could process at a time. The new resource allocation mechanism would need to enforce whether an application could be assigned resources and if so how many.

With the addition of modular DSP resources, came a new complication: the last saved allocation of resources may not match the hardware in the system after a reboot. It was possible that between boots, hardware could go bad, or be removed. The system would

now need to be able to detect hardware changes and have a robust way of handling mismatches in the configuration.

Last, with the dramatic nature of the changes to requirements and the pace at which they came, this new system would need to be designed in a way that was easy to update to support new applications and new DSP hardware. As more small businesses and retail stores used PBXs to make their businesses more efficient, and as technology continued to be come cheaper – it was very apparent that applications would play a much larger role in future PBXs.

3. Solution Overview

In this section we will review the formal requirements for the system, the architecture of the solution, as well as introduce the components that make up the system.

3.1 Requirements

3.1.1 Purpose

The purpose for redesigning the DSP resource management system is to accommodate more DSP resources and simplify configuration of these resources for the PBX administrator. The amount of time (number of operations) it took for initial configuration or to make changes to a running system with the previous implantation was too cumbersome, the new version needed to reduce the time it takes to make configuration changes.

3.1.2 Client / User Interface Requirements

The previous user interface was difficult to navigate and made making changes to resources overly complicated. The old design of going back and forth between tabs, changing DSP use with radio buttons, and assigning one resource at a time to an application using a pop up dialog box took way too much time. As the number of DSP resources was increasing dramatically, the amount of time to configure them using the old design would be frustrating to use and costly to the company paying the installers / administrators.

3.1.2.1 Change paradigm to “How Many” not “Where”

The administrator wants to configure “how many” resources they need for each application, not “where” the resources are located in hardware. Given this fundamental shift in how resources are allocated, the user interface design should be simplified greatly. Since the administrator no longer needed to know where resources were being allocated in hardware, the configuration could be based on the old resource management concept and the resource distribution concept could be dropped.

3.1.2.2 All Validation Processing Must be Implemented in the Client

The server is running a real time phone system and cannot be bogged down with lots of computation and validations. The client may be communicating with the server over a modem, so round trips are slow. The client must be responsive when making local changes to allocations.

3.1.2.3 Hardware and Application Information Must be Database-Driven

The client shall be designed so that it loads all information about the hardware available and the applications configurable from the server. It shall be designed in a flexible way so that new hardware or applications do not require major code changes. All of the various limits for allocation must be changeable via the database.

3.1.2.4 Advanced Settings for Removed Hardware

The user interface shall provide the administrator with a way to specify which resources have priority in case the system comes up with hardware missing.

3.1.3 Server Requirements

3.1.3.1 Loading & Licensing

The server is responsible for providing the client with the current configuration as well as system constraints for hardware and applications. The server shall be responsible for adjusting various application assignment limitations based on licensing – the client should not need any special coding to deal with licenses.

3.1.3.2 Saving & Notifying

The server will save the configuration that the client passes to it, determine how to distribute the configured resources to the actual system hardware, and notify applications and drivers of the hardware changes. The server should minimize the number of resources shifted when changes to configuration are made.

3.1.3.3 Boot Up

When the system is booted, the server will verify that the resources still exist to support the configuration stored in the database.

- New hardware must be detected and corresponding database representations must be added.
- Moving hardware must be transparent to the administrator. The system will reallocate the hardware automatically.
- If the system detects that hardware has been removed and there is insufficient hardware for the stored configuration, the database will need to mark the stored configuration as disabled. The system will then send the applications a temporary configuration that is derived from the advanced settings so that the system will

still be functional. If the three application groups defined in the advanced settings have sufficient resources to match the saved configuration, the remaining resources will be distributed in a round robin fashion among the applications in the stored configuration. If there are no entries in the advanced settings, the system will use a round robin method of distributing the resources among the applications in stored configuration.

3.2 Architecture

The architecture for the DSP resource management system that the author implemented is client-server based. A thick client is responsible for the allocation user interface, calculation of optimal ways to allocate the resources, and verification that the selected allocation is valid. The client communicates with the server through a web server. Behind the web server lives a service that provides an API to the PBX's central configuration database and handles notifications to components when data they use has been updated.

This project required a new client applet, new API methods and notification mechanisms in the service, modifications to libraries that pass data between the service and other modules / device drivers, as well as new tables in the database.

The Resource Management Interactions Diagram (Figure 4) shows the paths of communication between each of the components for this project. Distinctions are made between User-mode components and Kernel-based drivers and hardware.

Logical View

Resource Management Interactions

Figure 4: Resource Management Interactions Diagram

3.3 Components

3.3.1 Hardware and Drivers

The PBX the author worked on consists of a rack-mounted chassis that contains a Windows-based server and provides slots that take proprietary add-on cards to customize the system. Custom device drivers handle the real-time call processing and hardware interaction.

3.3.2 Web-based Administration

This PBX has a web-based interface for making configuration changes. This interface consists of a mix of Java applets, standard HTML forms, as well as native server-side Windows applications or control panels that are accessed through an embedded Remote Desktop web interface.

3.3.3 Resource Management Applet

The new resource management Java applet is a thick client that provides a user interface for the administrator to make changes to the resource allocation and apply them to the server. Using Java for the client at the time of this project provided many advantages including the ability to write powerful applications using standard programming methods that could run in any mainstream web-browser on any platform, and it was the only widely used web-based thick client at the time.

3.3.4 Database

The database used for the PBX at the time of this project was Microsoft Access-based. Most of the access to the database was ODBC-based SQL queries. Updates to the

database were made as part of system software releases using a dedicated ActiveX Data Objects (ADO)-based program. This program contained versioned upgrade steps, each of which could take in simple text files for data import and SQL statements as well as handle custom ADO code for more complicated operations.

3.3.5 CDataRep Service

CDataRep is the equivalent of a symphony conductor for processing configuration changes and coordinating notifying all of the affected components. It's a Windows service that provides a COM interface for other components to read and write the configuration information contained in the database. It also interacts with the Windows registry to read hardware configuration information as well as keeping any configuration information that is stored in the registry synchronized with the data stored in the database.

At boot up, CDataRep looks at the registry for the current hardware configuration and updates the database to reflect any changes - this consists of disabling data pertaining to removed hardware, populating default data for new hardware, and re-enabling data for hardware that was previously removed but has now been replaced.

3.3.6 TAPI and Applications

This PBX uses TAPI to provide access to system ports (extensions) for applications. When the assignment of system ports to applications changes, the TAPI module for the PBX gets notified and updates itself.

Applications register themselves with CDataRep using a named pipe synchronization mechanism. The named pipe allows CDataRep to inform applications that the system is

adding or removing system port assignments. When the application receives a remove message via the named pipe, it informs CDataRep as soon as it finishes releasing the system port. For sysports that are being re-assigned, CDataRep waits for the old application to release the system port and then informs the new application that the port is now available.

3.3.7 ResourceManager Module

The ResourceManager module is a component that handles coordinating changes to the ownership of DSP resources between drivers. When a built-in DSP changes use from TAPI to VoIP, the TAPI driver must let go of the DSP and when that is finished, the VoIP driver needs to be told to take ownership of it (and vice-versa). These changes are coordinated through talking to the drivers directly, and updating the DSP device type in the Windows registry representation of the hardware.

4. Database Design

The PBX's primary storage for configuration is a Microsoft Access database. The author was responsible for adding new tables and data for the database-driven hardware and configuration models. In addition work was required to refactor existing tables to support this new approach. Part of this refactoring included removing columns that were no longer used in the existing tables – these removals are omitted for brevity.

4.1 Entity Relationship

Figure 5: Entity Relationship Diagram

The Entity Relationship Diagram (Figure 5) shows how the various database tables used in this project map to each other.

PbxExtension serves as a foundation for many of the PBX tables that are beyond the scope of this project – the relationship to the rest of the PBX is not included. FaxPort and SysPort are types of extensions (PbxExtension).

DspResource and AppGroupMap each contain listings of extensions. Each Dsp has a DspType. DspResource, FaxPort, and SysPort all have entries that map to a unique port on a DSP.

There are a number of resource categories (ResourceCatID) that have many applications (ResourceAppID). DspResourceAdvancedSetting contains a list of categories.

4.2 Tables

4.2.1 Resource Categories (Read-Only)

The ResourceCatID table maps groups to categories. This table was refactored to support the new types of resources that appear in the application tree (IP Telephony and Fax).

Table 1: ResourceCatID

CatID	ResourceCategory	CatName
0	None	Unused Resources
256	TAPI	Voice Mail Group
512	TAPI	Call Routing and Queuing
768	TAPI	Third Party Applications
1024	TAPI	Contact Center
1280	TAPI	Music On Hold
3000	IPTelephony	VoIPGroup
4000	Fax	FaxGroup

- CatID is a unique identifier for the group/sub-category. (Primary Key)

- ResourceCategory is a new column that refers to high-level categories such as TAPI, IP Telephony, and Fax that the groups belong to, loosely related (but not a 1 to 1) relationship with port types. VoIP may have different flavors ports that would both be configured in applets under the IPTelephony ResourceCategory.
- CatName is a comment for the database developer that describes what the category is used for.

4.2.2 Applications

The ResourceAppID table maps applications to groups, specifies what type of ports the application uses, the maximum number of resources allowed for the application, and the last saved assigned ports for the application. AssignedPorts is the only field that gets updated, all others read-only. This table was refactored to include PortType, LicenseType, PerPort and AssignedPorts.

Table 2: ResourceAppID

Appl	CatID	AppName	MaxResources	PortType	LicenseType	PerPort	AssignedPorts
1	0	Available	-1	TAPI			0
257	256	Voice Mail	2	TAPI			4
258	256	Automated	6	TAPI			5
259	256	Pager	2	TAPI			6
260	256	AMIS	3	TAPI			7
513	512	Inbound IVR	100	TAPI	CRQ	False	8
514	512	Agent Login	100	TAPI	CRQ	False	9
515	512	Agent	100	TAPI	CRQ	False	5
516	512	IVR Paging	100	TAPI	CRQ	False	0
769	768	Third Party	100	TAPI			0
1025	1024	ACD	100	TAPI			3
1026	1024	Station	100	TAPI			4
1281	1280	Wave Player	1	TAPI			5
3001	3000	VoIP	-1	VoIP			16
4001	4000	Fax Manager	-1	Fax	FaxManager	False	18

- AppID is a unique id for the application. (Primary Key)

- CatID is the id of the group the application is part of. There is a Foreign Key constraint to the ResourceCatID table.
- AppName is a comment for the database developer that describes the application.
- MaxPorts: refers to the maximum number of ports that can be assigned to an individual application, -1 used to indicate unlimited.
- PortType refers to flavor of port, TAPI, VoIP, or Fax.
- AssignedPorts refers to how many ports are assigned to this application. Port is an abstract unit of configuration that can be assigned to an individual application.
- LicenseType refers to the name of the license (optional) that is required to configure ports for the applet.
- PerPort refers to whether the license contains a restriction on the number of ports allowed (true) or whether the license enables/disables the ability to assign up to MaxPorts to the application (false).

4.2.3 Dsp Types (Read-Only)

The DspType table specifies the possible types of DSPs supported in the system, the port types the DSP supports, and the trade-offs in numbers of ports depending on use.

This is a new table.

Table 3: DspType

DspType	PortType	Ports
549-5409	TAPI	12
549-5409	VoIP	4
TAPI549-5409	TAPI	12
PTMC	TAPI	1
PTMC	VoIP	1
PTMC	Fax	1

- DspType is an identifier that refers to an abstract type of DSP. A single DspType may have multiple rows representing different types of use such as TAPI, VoIP, or Fax and can have a different number of ports possible depending on use.
- PortType refers to flavor of port, TAPI, VoIP, or Fax.
- Ports refer to the number of resources that the DSP supports for the given PortType.
- Primary Key is a combination of DspType and PortType.

The data in the table shows that 549-5409 DSPs support either 12 TAPI Ports or 4 VoIP Ports. The TAPI549-5409 only supports 12 TAPI Ports. The PTMC DSPs support 1 TAPI Port, 1 VoIP Port, or 1 Fax Port. These differences in tradeoffs of port capabilities demonstrate the challenge in optimizing the DSP configuration.

4.2.4 Dsp

The `Dsp` table is a new table that maps the abstract DSP used by resource management to a real piece of hardware. This table specifies the location of the DSP, abstract type of DSP, current use of the DSP, and whether the DSP is present in the system. This table is updated at boot and grows as new hardware is detected. PortType is the only configurable setting.

Table 4: Dsp

DspID	Slot	Module	Device	Channel	DspType	SelectedPortType	Enabled
1	5	-1	4	-1	TAPI549-5409	TAPI	Y
2	5	-1	5	-1	549-5409	None	Y
3	5	-1	14	-1	549-5409	None	Y
4	1	1	1	1	PTMC	VoIP	Y
5	1	1	1	2	PTMC	VoIP	Y
6	1	2	1	1	PTMC	Fax	Y
7	1	2	1	2	PTMC	TAPI	Y

- DspID is a unique identifier to enumerate dsps in the system. (Primary Key)
- Slot, Module, Device, Channel is a hierarchical reference (a slot has 0 or more modules, which have 1 or more devices, which have 0 or more channels) to the location in hardware of the physical DSP. -1 means that the sub-reference is not applicable.
- DspType refers to the abstract type of the DSP.
- SelectedPortType refers to the PortType that this DSP is being used for.
- Enabled is a flag to keep track of whether the DSP is still in the system and functioning.

4.2.5 Current Dsp Resource Configuration

The DspResource table is a new table that contains the actual current allocation of resources for the system. It maps the individual ports on a DSP to an application. When resources are assigned or unassigned, this table grows or shrinks.

Table 5: DspResource

DspID	Port	AppID	AppPort	Extension
1	1	256	1	S050
1	2	256	2	S051
6	1	3001	1	

- DspID, Port is a reference to the individual DSP and port that the application has been allocated.
- AppID is a reference to the application that the resource has been allocated to.
- AppPort contains a unique (per group) sequential identifier based on AppID.

- Extension is a reference to the extension corresponding to the individual DSP and port based on type of use (TAPI, VoIP, or Fax).
- Primary Key is DspID and Port.

4.2.6 Application Group Mapping

AppGroupMap is a new table used to keep track of which individual resources have been assigned to hunt groups. It is used when the extension assignments in DspResource change to update the hunt group with the new extensions. A Hunt Group is a group of extensions assigned to a single group extension – so that if the group extension is dialed it will find an available individual extension from within the group to handle the call.

Table 6: AppGroupMap

Extension	AppPort
550	1
550	2
600	3

- Extension is a Hunt Group Extension.
- AppPort is reference to the identifier in DspResource.
- Primary Key is Extension and AppPort.

4.2.7 PbxExtension, SysPort, and FaxPort

These tables represent extensions in the PBX. The description of these tables is abbreviated to only include relevant data to resource management. The full function and use of these tables is beyond the scope of this thesis.

4.2.7.1 Pbx Extension

The `PbxExtension` table is the master table of extensions for the PBX. This table serves as a parent to other tables for the specific type of extension.

Table 7: PbxExtension

Extension	ExtType
S000001	768
F000001	1280

- Extension is an ID that represents an endpoint that a call can be connected to. Some extensions can be dialed directly on a phone, while others such as SysPorts and FaxPorts can only be accessed by drivers or through hunt groups. (Primary Key)
- ExtType is a numerical type corresponding to the use of the extension.

4.2.7.2 System Ports

The `SysPort` table contains additional properties for an extension with an `ExtType` representing a SysPort. A SysPort is a representation of a TAPI port in the PBX. This table was modified to include Enabled, DspID and Port.

Table 8: SysPort

Extension	Enabled	WaveID	DspID	Port
S000001	True	0	4	1
S000002	True	1	4	2
S000003	True	2	5	1

- Extension is a reference to the extension ID in parent `PbxExtension` table. (Primary Key)
- Enabled is a flag to keep track of whether the DSP and Port are still in the system and functioning.

- WaveID is a unique ID that matches the Windows enumeration for this TAPI endpoint.
- DspID, Port is a reference to the individual DSP and port that the application has been allocated.

4.2.7.3 Fax Ports

The FaxPort table is similar to the SysPort table in that it contains additional properties for extension with an ExtType representing a FaxPort. This table was modified to include Enabled, DspID and Port.

Table 9: FaxPort

Extension	Enabled	DspID	Port
S000001	True	4	1
S000002	True	4	2
S000003	True	5	1

- Extension is a reference to the extension ID in parent PbxExtension table.
(Primary Key)
- Enabled is a flag to keep track of whether the DSP and Port are still in the system and functioning.
- DspID, Port is a reference to the individual DSP and port that the application has been allocated.

4.2.8 Advanced Settings

The DspResourceAdvanceSettings table is a new table that contains the administrator's preference for how to distribute resources if the system comes up with

previously allocated hardware missing. It is used to downscale the resource assignments to the remaining hardware.

Table 10: DspResourceAdvancedSetting

Position	CatID	Percentage
1	256	50
2	512	25
3	0	25

- Position refers to the order in the AdvancedSettings dialog for the GUI. (PrimaryKey)
- CatID refers to the group being assigned a percentage of resources. There is a Foreign Key constraint with ResourceCatID table. 0 in this field has a special meaning – distribute evenly among the configured applications.
- Percentage contains the assigned percentage weighting to give the Group.

4.3 Data

4.3.1 Tables with Read-Only Data

Most of the tables with “Read-Only” data were populated when created. Resource Categories, Applications, DSP Types, and Advanced Settings fall into this category. New entries can be added with a database upgrade that accompanies a new version of the system software.

While some of the tables are purely read-only, others allow the individual predefined rows to have some fields updated with configuration data. The ResourceAppID table and DspResourceAdvanceSettings table both contain fields that can be updated. For the ResourceAppID table, the Assigned Ports field is updated when resources are

assigned. In the `DspResourceAdvanceSettings` table, the `CatID` as well as `Percentage` can be updated.

4.3.2 Dynamic Data Tables

The rest of the tables are dynamic in that rows can be added and/or deleted. The `Dsp`, `SysPort`, and `FaxPort` tables have some default rows for the legacy DSP hardware, new rows are added at bootup if new hardware is detected, and the `enabled` flag is set to `false` if hardware is not detected (rows are not ever deleted from these tables).

The `PbxExtension` Table will always have rows for each of the `SysPorts` and `FaxPorts` (which won't ever be deleted), it also has rows representing other types of extensions that are added and deleted.

The `DspResource` table and `AppGroupMap` table are both completely dynamic, meaning they add and delete rows as the configuration changes.

5. Client / Optimizer

This chapter details the design of the client applet. We introduce the user interface, class design, and algorithms that the client uses.

5.1 Graphical User Interface (GUI)

5.1.1 Main Applet

The GUI for Resource Management is a Java Applet that consists of a tree of applications, a dropdown box to assign ports to the selected application, counts of maximum available resources by type, an advanced settings button that pops up a dialog, and a set of buttons to restore previous settings, apply changes, exit the applet, or get help.

Figure 6: Resource Management Applet

5.1.1.1 Application Categories

The various applications are grouped into categories (based on the resource type they use). The tree structure on the left of the applet is used to select the application the user wishes to modify.

- TAPI Resources has many Groups as sub-categories, which in turn contain multiple Applications, they have one configurable resource: TAPI/WAVE Ports.
- IP Telephony Resources has the Voice Over IP Group as a sub-category, which has G.711/G.729A as its only application. This application has one configurable resource G.711/G.729A Ports. G.711 and G.729A are the names of International

Telecommunications Union (ITU) standards for the Coder-Decoder algorithms (Codecs) that are used to convert analog to digital and to compress the data.

- Fax Resources has the Fax Group as a sub-category, which has Fax Manager as its only application. Fax Manager has one configurable resource: Fax Ports

5.1.1.2 Available Resources

At the bottom of the screen, the Available Resources are listed. This list is adjusted to show the maximum possible remaining resources per type as the user assigns resources to the above-mentioned categories.

The available resource listing also enforces any system resources license limitations. If the customer does not have any licenses for a particular resource, that resource will not be shown in the applet. I.e. if the customer has no licenses for RSC VoIP Ports, the free resource list won't contain the 8 possible RSC G.711/g.729A VoIP Ports.

5.1.1.3 Port Assignment Dropdown Box

The dropdown boxes shown in the right hand side of the applet will have ranges from 0 ... X, where X is the number of unused ports for the resource type.

The port assignment dropdown also enforces any application license limitations. If the customer does not have a license for a particular application, the drop down will have 0 as its only value.

5.1.2 Advanced Settings Dialog

The advanced settings dialog allows the user to specify a preference for how resources will be distributed among application groups if the system is booted with previously configured hardware missing.

Figure 7: Advanced Settings Dialog

5.1.3 Making and Applying Changes

When the user allocates resources in the client, it is a local simulated allocation. As soon as a change is made in the client that does not match what was last saved on the server, the Restore and Apply buttons are enabled. If the user does not like the simulated changes they made and wants to start over – they click Restore to reset the allocations to the last saved allocation. When the user wants to actually apply the changes to the

system the click the Apply button. If the user clicks the Done button to exit, but changes have been made in the applet that have not been applied yet, they will be prompted with a Yes / No / Cancel dialog on whether they want to save their changes before exiting.

5.1.4 Bad States

5.1.4.1 Missing Hardware

If hardware is missing, the applet gives a warning dialog letting the administrator know that the system is using a reduced configuration, and the currently configured resources showing in the applet are what were last saved – not the actual configuration being used. The applet will not enable the apply button unless the administrator lowers the assignments enough to fit the hardware remaining (i.e. they reduce resources until the available resources is not 0).

Figure 8: Missing Hardware Error Dialog

5.1.4.2 More Resources Configured Than Allowed (Removed Licenses)

If an application or resource license is removed (not normal), the server-side doesn't make any adjustments to the configuration; however, the applet will show the administrator the configuration as if those resources were removed – meaning it will no longer allow them to assign those resources. The next time the administrator makes any

changes to allocation and saves them, those “unlicensed” resources will be removed from the server.

5.2 Class Design

The Applet Class Diagram (Figure 9) shows the design of the applet. The Model-View-Controller (MVC) paradigm is used to organize and separate the various classes and objects [16].

The model contains representations of the DSP hardware, hierarchical category to group to applications mapping, as well as the advanced setting data.

The controller does all the work of loading the data, saving the data, and processing changes to the configuration. This processing includes executing the algorithms to optimize the assignment of ports in order to maximize the remaining ports available.

The View contains the main applet including the tree panel, port assignment panel, and dialogs for advanced settings and messages.

5.2.1 Main classes

5.2.1.1 View

ResourceManagementApplet – The main applet class which consists of two panes, one with a tree of the applications, and one with the settings for the selected application.

- Init – creates the GUI and calls all of the load operations from the ResourceController.
- Apply – calls the save method from the ResourceController.
- Restore – restores the settings that were in effect when init was called or when the last apply occurred.

AssignPortsPanel – This is the panel for selecting how many ports should be assigned to the selected application.

ResourceTreePanel – This panel contains a JTree that will allow navigation and selection of the applications, groups, and resource categories.

AdvancedSettingsDialog – This dialog implements advanced settings to deal with missing resources.

5.2.1.2 Controller

ResourceController – This class represents the interface between the view and the model. It provides functions to figure out how to assign resources most efficiently, and also to calculate the remaining free resources. It also handles communicating to the server for all loading and saving of data.

- getFreePortsCount – returns the maximum number of ports that are available to be assigned to an application for a given port type.
- assignPorts – updates the application with the new total ports, calculates the most efficient ways to assign the given the desired number of ports of the given port type to maximize the number of free ports for each port type.
- loadPortTypesLimit – gets the types of ports and system limits
- loadDSPTypes – gets the types of DSPs in the system
- loadDSPs – gets the DSPTypes and number of DSPs per type
- loadResourceCategories – gets the resource categories and application groups
- loadApplications – gets the applications for each of the application groups
- loadAdvancedSettings – gets the advanced settings application groups and resource percentages
- save – saves the number of ports assigned to each application and the advanced settings

5.2.1.3 Model

DSP – This class represents a DSP on the system.

DSPType – This class represents the types and capacities of DSPs in the system.

Ports – This class represents a port type and number of ports that are associated with it.

PortType – This class represents the types of ports on the system

TreeNodeVector – This class is used as the root TreeNode and container for a group of TreeNodes.

ResourceCategory – This class contains the various resource categories such as TAPI, VoIP and Fax.

ApplicationGroup – This class represents a group of related applications such as the Voicemail Group.

Application – This class represents an individual application such as the Automated Attendant. Also contains how many ports are assigned to this application.

AdvancedSettingsModel – This class contains advanced settings data.

5.2.2 Supporting Classes & Interfaces

Cacheable – Interface with three methods: Restore, Apply, and IsDirty. Allows user interface to interact with data classes that keep track of their original values vs modified values in a standardized manner. Restore overwrites modified values with original values, Apply overwrites original values with modified values, IsDirty reports whether modified values differ from original values.

JTitleRADHPanel - Main panel for applets. It displays a title bar, a content pane, and a set of Restore/Apply/Done/Help buttons.

RestoreApply - Interface that defines Apply and Restore callbacks from JTitleRADHPanel buttons.

JOKCancelDialog - A modal dialog that contains an ok button and a cancel button, with methods that may be overridden by the inheriting class.

5.3 Optimization Algorithms

The client runs a couple algorithms every time an application's resources are changed in the GUI. These algorithms run through assigning the resources to the modeled hardware in different permutations. The main function of these algorithms is to update how many possible ports of each type can be assigned. By running through all the permutations and updating the maximum remaining ports, we are optimizing for each port type.

5.3.1 reassignPorts

Purpose: Assigns configured application's ports on different DSPTypes with the purpose of finding the maximum total possible number of free ports for each portType..

Uses two lists:

- htPortTypeTotals : contains the total needed ports per port type
- htAssigned : contains the current number of ports assigned per port type.

Description:

```
Algorithm: reassignPorts()  
  
Begin  
  Reset all DSPs (data structures)  
  Loop though all non-configurable DSPs and assigns ports  
  Calls recursiveReassignPorts() for remaining ports  
End
```

Figure 10: reassignPorts Algorithm

5.3.2 recursiveReassignPorts

Purpose: Recursively tries to assign ports on different DSPTypes with the purpose of finding the maximum total possible number of free ports for each portType.

Parameters: Takes in the two lists from reassignPorts

Description:

```
Algorithm: boolean recursiveReassignPorts(Hashtable htPortTypeTotals, Hashtable htAssigned)
Begin
  bFinished = True
  For Each portType in htPortTypeTotals Loop
    If assignedPorts from htPortTypeTotals > htAssigned
      bFinished = False
      Break
    End If
  End Loop

  If bFinished = True
    Base Case: All of the needed ports are assigned
    Update total possible free ports per type if the total is higher than previous total.
    Return True
  Else
    For Each DSPType that is configurable and not currently assigned Loop:
      Look for a DSP with that type
      Recursive Case:
        Assign the dsp to the first port type that still needs ports
        Assign the number of ports that are still needed or the maximum available on the DSP
        Update htAssigned
        If recursiveReassignPorts = True
          bFinished = True
        End If
        UnAssign the dsp from above
        Update htAssigned
      End Loop
    End If
    Return bFinished
  End
End
```

Figure 11: recursiveReassignPorts Algorithm

While the author wrote the algorithms in this project to solve this problem in response to the requirements, similar problems to this have been the subject of many algorithm texts. The problem of finding a configuration of applications to fit within a set of DSPs is very similar to the Bin Packing problem. The Bin Packing problem is summarized as the question of whether a set of items that have varying sizes can fit inside a set number of bins that have a fixed capacity. This problem is a combinatorial NP-hard problem that is known to NP-complete [4].

The client optimizer is optimizing for multiple scenarios – specifically it tries to optimize the remaining resources for each type use, which makes the problem harder than

if we were just trying to optimize for a single resource. By performing a recursive search for all the combinations of assigning resources to DSPs we ensure that the optimizer finds the correct values of maximum remaining resources for all of the resource types in a generic way.

One of the strategies for solving the Bin Packing problem is using a first-fit algorithm, which sacrifices optimal fit for speed. This is basically the same strategy that the allocator employs on the server-side to minimize the amount of time and processing used to apply the configuration to the hardware.

6. Allocator / Server

The server provides an API to the client for loading and saving the configuration of resources. The server modifies the database and registry, and notifies applications and drivers of changes.

6.1 Flow Charts

The flow charts in this section are used to show visually how the client and server interact with each other. These charts include the mapping of method calls and the data that is exchanged.

6.1.1 Applet Requests Data

Client Loading Process

Figure 12: Client Loading Process

As Figure 10 shows, the client has a number of methods to load the various models and configuration data it needs to populate the applet. Each of these methods has a corresponding API method on the server-side, which retrieves the data from the database and passes it back to client.

6.1.2 Applet Saves Data

Client Saving Process

Figure 13: Client Saving Process

When the client applies changes to the server, it does so using a single save method. This save method calls two API methods on the server and passes them the corresponding configuration data to save to the database.

6.2 CDataRep Service

The CDataRep Service serves as the configuration API for the system. It provides clients with methods to read and write data to the database. It also provides a notification infrastructure to inform various services and drivers when data changes. It is an apartment-threaded (processes one API call at a time) COM Service that is written in Visual C++. It stores and retrieves data from the database. It reads information at boot up from the Windows registry and writes notification data to the registry.

6.2.1 API Methods

- PbxGetApplications(BSTR *pbsApplicationsList) – this method returns a string that contains a list of application records. Each record contains the application, which group it belongs to, the type of port it uses, how many ports are currently assigned to it, and the maximum number of ports allowed to be assigned to it.
- PbxGetResourceCategories(BSTR * pbsResourceCatList) – this method returns a string that contains a mapping of resource type to groups that use those resources.
- PbxGetDsps(BSTR *pbsDspList) – this method returns a string that contains a list of DspType records. Each record contains a DspType, how many of those DSPs are in the system, and a list of the types of ports and number of ports that can be configured.
- PbxGetResourceAdvancedSettings(BSTR *pbsAdvancedSettings) – this method returns a string that contains 3 records. Each record has a group and a percentage of resources assigned to that group.
- PbxSetApplications(BSTR bsApplicationsList) – this method takes a string in that contains a list of application records. Each record contains the application, and how many ports are to be assigned to it.
- PbxSetResourceAdvancedSettings(BSTR bsAdvancedSettings) – this method takes a string in that contains 3 records. Each record has a group and a percentage of resources assigned to that group.

6.2.2 Bootup

Upon boot CDataRep needs to verify that all of the DSP resources that were previously configured still exist in the database. If DSPs have been added CDataRep will

add them into the `Dsp` table and will create new extensions (sysports and faxports) for the new resources. If DSPs have been removed, `CDataRep` will update the `Dsp` table to mark the removed DSPs as disabled and will disable corresponding extensions (sysports and faxports). Finally after all changes to the `Dsp` table take place, `CDataRep` will try to re-allocate the saved config

The algorithm for doing the bootup will be as follows:

```
Algorithm: VerifyDspTables()
Begin
  Disable all Dsp & Sysports/FaxPorts.
  Look at the revision of the hardware and enable known legacy 549/5409 DSPs.
  Loop through the hardware tree in the registry
    Add/Enable MapleTree DSPs.
    Add/Enable Sysports for all DSPs.
    Add/Enable FaxPorts for MapleTree DSPs.
  End Loop
  Renumber WaveIds.
  Call ReassignDspResources to reallocate the saved configuration.
End
```

Figure 14: VerifyDspTables Algorithm

If there is insufficient hardware, `CDataRep` will use the advanced settings to assign resources to the various applications. If any resources remain after allocating enough resources to cover the advanced settings, or if no application categories are specified in advanced settings, a round-robin method will be used to distribute resources among the applications configured in the saved configuration.

6.2.3 Assignment Algorithms

The assignment algorithms on the server are run when the client makes configuration changes and when the system boots. The reassignment algorithms mimic the client in that they assign the configured resources to modeled DSP hardware. They differ from

the client in that they assign the resources to the actual hardware, and they are designed to find the first fit of resources to hardware – that is they are finished when they find the first permutation that fits. The algorithm to update the database tables contains logic to minimize changes, notify applications and hardware of the changes, and to update the `DspResource` table with the final assignments. Lastly, the advanced settings algorithm gets called if the system is booted with less hardware available than the saved configuration needs. This algorithm scales back the configuration to fit the remaining hardware.

6.2.3.1 SetApplications

Purpose: API call for the client to save a configuration. This method also prevents a new change from being applied, if the previous changes are not finished.

Parameter:

- `bsApplicationsList`: contains a text representation of a mapping for new port assignments per application

Description:

- Checks whether last allocation is finished if not returns error
- Calls `ReassignDspResources()`

6.2.3.2 ReassignDspResources

Purpose: Assigns configured application's ports on different DSPTypes with the purpose of finding a configuration of DSPs that contain enough ports to cover all of the assigned ports.

Parameters:

- csApplicationsList: text representation of a mapping for port assignments per application
- bCalledFromBootup: Boolean specifying whether this method is being called at startup or from the applet

Uses five lists:

- mapApps : contains the new port assignments per application
- mapPortTypeTotals : contains the total needed ports per port type
- mapAssignedPorts : contains the current number of ports assigned per port type.
- mapDspTypes : contains the database model of dsp port tradeoffs
- mapDsps : contains the database model of enabled DSPs

Description:

```
Algorithm: ReassignDspResources(CString csApplicationsList, bool bCalledFromBootup)

Begin
  Creates the various lists by parsing bsApplicationsList and loading data from the database
  Resets all DSPs
  Goes through all non-configurable DSPs and assigns ports
  Calls the RecursiveReassignDspResources() for remaining ports
  If RecursiveReassignDspResources returns False and bCalledFromBootup = True
    Call ReassignResourcesUsingAdvancedSettings()
  End If
  Calls UpdateDatabaseTables()
End
```

Figure 15: ReassignDspResources Algorithm

6.2.3.3 RecursiveReassignDspResources

Purpose: Recursively tries to assign ports on different DSPTypes with the purpose of finding a configuration that contains enough ports to cover all of the assigned ports.

Parameters:

Takes in mapPortTypeTotals, mapAssignedPorts, mapDspTypes, mapDsps

Description:

```
Algorithm: boolean RecursiveReassignDspResources(mapPortTypeTotals, mapAssignedPorts, mapDspTypes,
                                                mapDsps)

Begin
  If Base Case: All of the needed ports are assigned
    Save this config in the database.
    Return True
  Else
    For Each DSPType that is configurable and not currently assigned Loop:
      Look for a DSP with that type
      Recursive Case:
        Assign the dsp to the first port type that still needs ports
        Assign the number of ports that are needed or the maximum available on the DSP
        Update mapAssignedPorts
        If recursiveReassignDspResources() = True
          Return True
        End If
        UnAssign the dsp from above
        Update mapAssignedPorts
      End Loop
    End If
  Return False
End
```

Figure 16: RecursiveReassignDspResources Algorithm

6.2.3.4 UpdateDatabaseTables

Purpose: Coordinate resource changes for Applications, DSP Hardware / Drivers, and update the Database.

Parameters:

Takes in mapDsps, mapDspTypes, mapApps, updateApps

Uses three lists:

- mapDspResources – contains a map of DSP and Port to resource assignment data
- dspResourceDeleteList – contains a list of DspResources to delete

- dspModuleUpdateList – contains a list of DspModules to update

Description:

```

Algorithm: UpdateDatabaseTables(mapDspS, mapApps, mapDspTypes, bUpdateApps)
Begin
  If this is not being called from bootstrap
    update AssignedPorts fields in ResourceAppID table with entries from mapApps
  End If
  Loop through mapDspS
    Get the dspType for DSP
    Get the number of ports for the new SelectedPortType for this DSP
    Create entries in mapDspResources for each port
    Query DspResource table for current assignments for this DSP

    For each row returned, look in mapApps for resources assigned to the AppID Loop
      If the DSP's SelectedPortType has not changed and the app has 1 or more ports assigned to it
        then consider it already assigned.
        Decrease mapApps ports to assign by 1
        Remove this port from the mapDspResources
      Else
        Add this resource to the dspResourceDeleteList
        If the type of the resource is TAPI or Fax
          Send a pipe message to the application telling it the resource is being removed
        End If
      End If
    End Loop

    Update the SelectedPortType field in the Dsp table
    If the DSP is a Legacy Device
      Call ChangeDevice in ResourceManager
    Else it's a PTMC module
      Call ChangeChannel in ResourceManager
      Add the module to the dspModuleUpdateList (if its not already present)
    End If
  End Loop
  For each module in dspModuleUpdateList Loop
    call ChangeModule in ResourceManager
  End Loop
  For each DspResource in dspResourceDeleteList Loop
    delete the corresponding row in the DspResource table
  End Loop
  Loop through mapDspS
    Get the dspType for DSP
    Get the number of ports for the new SelectedPortType for this DSP
    For each port look for a corresponding entry in mapDspResources
      If none exists it means the port has already been assigned
        skip to next port
      Else
        loop through mapApps and find the first app that has a port type that matches the DSP's
        SelectedPortType and has 1 or more ports left to assign.
        Decrease mapApps ports to assign by 1
        Set the mapDspResources entry for this resource to have this AppID
        If the type of the resource is TAPI or Fax
          Query the database to get the corresponding SysPort or FaxPort extension
          Set mapDspResources entry to have this Extension
          Insert a row in the DspResource table for this entry
        End If
      End Loop
    End If
  End Loop
End Loop

```

Figure 17: UpdateDatabaseTables - Part 1

```

Loop through mapDspResources
  If type of the resource is TAPI or Fax
    Spawn a thread to send a pipe message to the application telling it the resource is
    being added
  End If
End Loop
UpdateAppHuntGroups (goes through DspResource and sequentially assigns AppPortId for each
resource with matching AppID, then uses the AppGroupMap table to update any Hunt Groups
extensions to match the new ones in DspResource)
End

```

Figure 18: UpdateDatabaseTables - Part 2

6.2.3.5 ReassignResourcesUsingAdvancedSettings

Purpose: When the hardware present in the system does not support the saved configuration of resources, re-allocate DSP resources to provide core functionality using groups

Description: Looks at the DspResourceAdvancedSettings preferences and assigns ports to the prioritized application categories one by one until it runs out of room (i.e. calling RecursiveReassignDspResources returns false). If all of the prioritized categories resources were assigned and there was any room left over, it would do a round robin assignment for all other resources until it ran out of room. At that point, it calls UpdateDatabaseTables, which saves the reduced configuration to the dynamic tables in the database.

6.3 ResourceManager library

The ResourceManager library has one purpose, to gracefully change which resources are assigned to a given driver. It is a COM Object written in Visual C++. It talks to two different drivers IpUsr and DspDll.

When ResourceManager is told to change the DSP Resources, it needs to inform DSPMgr and IPDSPMgr of the changes to their resources (using user-mode DspDll, and IPDspUser).

6.3.1 Legacy DSPs

6.3.1.1 Port type changes from TAPI to VoIP

- Update the Windows Registry value for Device from DSP Manager to IP Telephony
- Remove the Device from DSPMgr
- Call Modify for the Device in IPDSPUser

6.3.1.2 Port type changes from VoIP to TAPI

- Update the Windows Registry value for Device to DSP Manager from IP Telephony
- Call Modify for the Device in IPDSPUser
- Add the Device to DSPMgr,

6.3.2 PTMC Module

- Update the Windows Registry values for all of the Channels
- Call Modify for the Module in IPDSPUser

7. Evaluation

This project was created to address specific requirements that were introduced in previous chapters. This chapter details how those requirements were satisfied. The first section of the chapter contains experiments that compare the efficiency of using the new system compared to the old as measured by actions the user takes to perform various tasks. Next, we review the formal requirements and provide details on how the requirements were validated. In addition, we provide a method to verify each validation. Finally, we review the field deployment results and customer feedback.

7.1 Experiments

As a way to measure time and complexity of configuring resources, the total number of clicks to perform an operation will serve as a basic metric. An additional metric will be the number of changes to fields made. These metrics were chosen because they correspond roughly to how much time is spent in the applet configuring items, which directly affects how long the system takes to install in the field. In this section, three basic types of configuration changes will be compared: setting up a basic configuration, adding some additional resources, and reallocating resources based on changing needs. An additional look at how the system scales will be presented. These experiments represent likely operations that customers would perform in the field.

7.1.1 Configuring a basic system

For a very basic configuration, we'll assign 1 Music On Hold resource, 10 Auto Attendant resources, 2 Voice Mail Shared Pool resources, and 2 Pager Outcall resources.

Tables 11 & 12: Basic Configuration Old System and New System

Old System	New System
Open Applet Select Resource Management tab Scroll Down to the Unused Category Double Click System Port 1 Select Voice Mail Group in Group drop-down Select Auto Attendant in Sub-Group drop-down Click OK Double Click System Port 2 Select Voice Mail Group in Group drop-down Select Auto Attendant in Sub-Group drop-down Click OK ... Repeat same 4 steps for System Ports 3 – 10 ... Double Click System Port 11 Select Voice Mail Group in Group drop-down Select Voice Mail Share Pool in Sub-Group drop-down Click OK Double Click System Port 12 Select Voice Mail Group in Group drop-down Select Voice Mail Share Pool in Sub-Group drop-down Click OK Double Click System Port 13 Select Voice Mail Group in Group drop-down Select Pager Outcall in Sub-Group drop-down Click OK Double Click System Port 14 Select Voice Mail Group in Group drop-down Select Pager Outcall in Sub-Group drop-down Click OK Double Click System Port 15 Select Music On Hold Group in Group drop-down Select Wave Player in Sub-Group drop-down Click OK Click Apply	Open Applet Expand TAPI Resources tree node Expand Voice Mail Group tree node Select Auto Attendant node Select 10 in TAPI/WAVE Ports drop-down Select Voice Mail Shared Pool node Select 2 in TAPI/WAVE Ports drop-down Select Pager Outcall node Select 2 in TAPI/WAVE Ports drop-down Expand Music On Hold Group tree node Select Wave Player node Select 1 in TAPI/WAVE Ports drop-down Click Apply

Table 13: Initial Configuration Results

System	Clicks	Field Changes
Old	64	30
New	13	4

7.1.2 Adding Resources on a running system

Add 4 IP Telephony Resources.

Tables 14 & 15: Add Resources Old System and New System

Old System	New System
Open Applet On Resource Distribution Tab, select Allocate 4 channels of IP Telephony radio button Click Apply	Open Applet Expand IP Telephony Resources tree node Expand Voice Over IP Group tree node Select G.711/G.729A node Select 4 in G.711/G.729A Ports drop-down Click Apply

Table 16: Adding Resources Results

System	Clicks	Field Changes
Old	3	1
New	6	1

In this particular case, the original design took fewer clicks due to navigation. This operation is on the Resource Distribution Tab, which is already displayed when the applet opens. Actual field changes were identical at one. The increased flexibility of having all resources categories in a single tree instead of jumping back and forth between tabs usually is a simpler interface to navigate, but this single operation is the exception.

7.1.3 Reallocating Resources on a running system

Configure a total of 8 IP Telephony Resources. This change can't be made directly since there are not enough resources available to accommodate this assignment. As a result resources will have to be reallocated to make this change. We need to free up 3 TAPI/WAVE resources, since we want to keep all of our applications running and Auto Attendant has the most resources, we'll reduce Auto Attendant resources from 10 ports to 7 ports.

Tables 17 & 18: Reallocate Resources Old System and New System

Old System	New System
<p>Open Applet</p> <p>On Resource Distribution Tab, select Allocate 8 channels of IP Telephony radio button</p> <p>Dialog pops up indicating that some TAPI resources must be deallocated and some hunt groups may be affected. Click OK.</p> <p>Select Resource Management tab</p> <p>Find System Ports with icons indicating that a resource is not available for TAPI allocation. In this case, System Ports 13, 14, and 15.</p> <p>Notice that these are configured for Pager Outcall and Music On – which are still needed.</p> <p>Navigate to the Auto Attendant Group and double click System Port 10.</p> <p>Group is already selected as Voice Mail Group, so change selection from Auto Attendant to Pager Outcall in Sub-Group drop-down</p> <p>Click OK</p> <p>Navigate to the Auto Attendant Group and double click System Port 11.</p> <p>Group is already selected as Voice Mail Group, so change selection from Auto Attendant to Pager Outcall in Sub-Group drop-down</p> <p>Click OK</p> <p>Navigate to the Auto Attendant Group and double click System Port 12.</p> <p>Change selection from Voice Mail Group to Music On Hold Group in Group drop-down</p> <p>Change selection from Auto Attendant to Wave Player in Sub-Group drop-down</p> <p>Click OK</p> <p>Click Apply</p>	<p>Open Applet</p> <p>Expand TAPI Resources tree node</p> <p>Expand Voice Mail Group tree node</p> <p>Select Auto Attendant node</p> <p>Change selection from 10 to 7 in TAPI/WAVE Ports drop-down</p> <p>Expand IP Telephony Resources tree node</p> <p>Expand Voice Over IP Group tree node</p> <p>Select G.711/G.729A node</p> <p>Change selection from 4 to 8 in G.711/G.729A Ports drop-down</p> <p>Click Apply</p>

Table 19: Reallocating Resources Results

System	Clicks	Field Changes
Old	16	5
New	10	2

7.1.4 Scalability

The previous changes have been limited to built-in DSP hardware to provide a direct comparison between the old system and new system. The built-in DSPs have a maximum of 36 ports that can be configured. The initial configuration demonstrates the

effort necessary to assign just 15 of these ports. For this last experiment we will assign 132 ports to show how the new system scales.

Table 20: Scalability New System

New System
Open Applet Expand TAPI Resources tree node Expand Voice Mail Group tree node Select Auto Attendant node Select 10 in TAPI/WAVE Ports drop-down Select Voice Mail Shared Pool node Select 2 in TAPI/WAVE Ports drop-down Select Pager Outcall node Select 2 in TAPI/WAVE Ports drop-down Expand Music On Hold Group tree node Select Wave Player node Select 1 in TAPI/WAVE Ports drop-down Expand Call Routing and Queuing Select Inbound IVR node Select 30 in TAPI/WAVE Ports drop-down Select Agent node Select 23 in TAPI/WAVE Ports drop-down Expand IP Telephony Resources tree node Expand Voice Over IP Group tree node Select G.711/G.729A node Select 40 in G.711/G.729A Ports drop-down Expand Fax Resources tree node Expand Fax Group tree node Select Fax Manager node Select 24 in Fax Ports drop-down Click Apply

Table 21: Scalability Results

System	Clicks	Field Changes
Old	N/A	N/A
New	26	8

7.2 Requirements Met

This section reviews how the requirements have been met with the new system.

Validation as well as verification steps are included.

Requirement	Support more DSP Resources
Validation	<p>Modular DSPs can be added to the PBX. Database has been designed to enumerate these DSPs and both client and server use algorithms that are driven by the database tables. 4 Types of Modular DSPs have been sold over the past 7 years and are in use in the field.</p> <p>This can be verified by inserting a one or more Modular DSPs into the system.</p>

Requirement	How Many not Where
Validation	<p>New applet abstracts hardware and presents the user with a tree of applications, each of which has a port assignment drop-down to select how many ports to assign.</p> <p>Using the applet or stepping through the GUI design will verify the shift in how applications are configured.</p>

Requirement	Validation on Client
Validation	<p>The client uses an algorithm to try all the possible configurations of the assigned applications on the DSP hardware. As assignments are made, the client updates the choices in the port assignment drop-downs to ensure the user can only select valid assignments. The client and the server use the same algorithm, which ensures that if the configuration fit on the client, it will fit on the server.</p> <p>This can be verified by making changes on the client and observing that they work on the server. In addition, the validations can be verified by pen and paper calculation of mapping resources to DSPs.</p>

Requirement	Database-driven
Validation	<p>The client and the server both populate their hardware and application models with data from the database. There is no predefined data in the client or server.</p> <p>This can be verified by modifying the tables in the database and validating that the client and server both handle the new values.</p>

Requirement	Advanced Settings for Missing Hardware
Validation	<p>The client allows a user to specify which groups of applications are most important and allocate a percentage of the system resources to those groups. The server detects hardware changes at boot and if the saved configuration does not fit on the hardware, it uses these advanced settings to assign the preferred applications to the remaining hardware.</p> <p>Removing hardware and rebooting the system can verify this.</p>

Requirement	License Enforcement
Validation	<p>The PbxGetApplications API method handles licensing by passing the client an updated maximum ports value for an application based on licensing restrictions. This prevents the client from being able to assign resources to an application that does not have sufficient licenses. The PbxGetDsps API method passes the client and the server a DspType with capabilities that correspond to whether a DSP has those capabilities licensed or not.</p> <p>Trying to configure licensed features without a license, then adding a license and retrying to configure the features can verify these.</p>

Requirement	Notification
Validation	<p>The server notifies applications and drivers of changes. The numbers of changes are minimized.</p> <p>Making calls that use the features to verify that the changes went through can validate the notifications. Trace logs for applications or the server can be parsed to observe the number of notifications sent in relation to the number of resources configured.</p>

7.3 Field Deployment / Customer Feedback

This software was released to the field in November of 2002. More than 9000 systems have shipped with this software. Some minor bugs and cosmetic changes were made from 2003 to 2004. The code has not changed since 2004. Since the release of this software, 4 applications have been added without requiring code changes. At least 7000 systems continue to run this software today.

8. Conclusion / Future Work

As small businesses and branch offices continue to use more communications technology in their day-to-day operations, the need for their PBX to run more sophisticated applications will increase. Being able to customize the applications that run on one's PBX without having to purchase and install additional external add-on systems provides the customer with cost-savings and flexibility as they expand. The project presented in this thesis demonstrated an implementation of a system to meet these needs.

Both technical and marketing constraints for the implementation presented some interesting programming challenges to overcome. The need to optimize the configuration of DSPs while minimizing interruption to the running PBX suggested that any heavy computation must occur in the client. The fact that changes would need to be made on the fly without bringing down the PBX meant that communication between different modules would need to be coordinated. The use of modular DSPs that could be added or removed necessitated that the implementation be both flexible and robust. The system presented in this thesis provides contributions that can be adapted to solve similar problems.

Modeling hardware and application capabilities in a database allow the client and server software to be designed around defined database tables. Placing the rules and limitations into database tables allow the coding to be somewhat generic – new database rows or individual values can be changed without having to modify code. This design

allows the system to be able to handle new hardware and applications without having to go through the cycle of making code changes.

In the case of optimizing resource assignment to various DSP hardware each of which had different capabilities it became clear that requiring the user to do complex calculations and tradeoffs was not going to be an acceptable approach. Instead, the configuration software would need to run these complex calculations and abstract the hardware so that the user could do simple assignments of how many resources they wanted for each type of application.

Most web browser-based configuration software that involves heavy computing is designed around the client being thin and the server-farm at the other end doing the majority of the hard work. In the case of a PBX or other stand-alone device where the server has limited computing power, it may make sense to use a thick client so that the heavy computing is local, more responsive, and interruption to the running device is minimized.

By implementing the optimization algorithm on the client, the user could try many different configurations and figure out which configuration worked best without bogging down their PBX.

The server needed to implement a similar algorithm in order to ensure that the assignment from the client would fit, as well as to handle re-assigning resources at boot when the hardware might change. One key difference in the implementation of the algorithm on the server-side is that it would finish when it found the first fit solution for the hardware. Adding additional logic to the server to minimize changes and

notifications, as well as being able to scale back the configuration if hardware was missing at boot increased the robustness of the implementation.

This system achieved what it was designed to do. It provided the user with a configuration tool that let them assign DSP resources to various applications in an optimal way on a running PBX. New applications have been added without needing to change any code. The system has been deployed on thousands of PBXs in the field for over 7 years and there have been no reported failures with the DSP resource configuration tool.

8.1 Future Work

As with any evolving system, new features and upgrades are inevitable. In the case of PBXs, just like home computers, hardware will become faster and cheaper as time marches on. New applications will continue to be developed for PBXs and are likely to use more and more DSP resources. In this section, some of the likely advancements are explored and discussed.

8.1.1 The Rise of VoIP

As IP Phones become standardized, they will become a commodity and as a result, much more affordable. As this happens, many businesses will find opportunity for savings as well as flexibility by wiring an office for Ethernet only instead of wiring each station for both Ethernet and phones. A PBX that supports more and more IP Phones will need increasing DSP resources to handle things like music on hold, voicemail, conferencing, etc. It is likely that DSPs will become more flexible in how they are configured and used as compared to the simple TAPI vs VoIP vs Fax capabilities that this

system's DSPs have. It is likely that the algorithms to configure these resources will become more complex.

8.1.2 Voice Recognition

Voice recognition software is becoming more commonplace at the consumer level. Many cell phones now offer voice-activated dialing. These types of applications are very DSP-intensive. It is likely that small office PBXs will incorporate more of these applications as they become part of customer expectations going forward.

8.1.3 Less Reservations and More Dynamic Pools of Resources

The system presented in this thesis is designed around reserving resources for specific applications. This paradigm is based on guaranteeing that the system can handle a certain number of calls that uses an application based on the number of resources dedicated to that application. As resources increase, it's likely that customer will want some of these resources to work more dynamically. Meaning, the customer may want to guarantee that at least 8 people can simultaneously access voicemail, but if another call comes in to voicemail and there are resources in the system that are not currently being used, the system should use them dynamically to offer additional access to voicemail on the fly.

8.1.4 Faster Servers and Thin Clients

For this project, the client was designed to be thick in order to preserve computing power on the server for real-time PBX functionality and to provide a responsive client when administering the PBX over slow remote connection. As multiple core processors become standard in PC hardware, it is likely that PBXs will incorporate this technology. In addition, VPN servers have made high speed network access between public and

private networks much more common. These advancements in computing power and network access speeds will make it possible to move to a thin client for complex tasks such as resource management optimization. Having the algorithms in one code-base instead of in two languages would make the code easier to maintain.

Bibliography

- [1] Broussard, Dan. "The Migration from TDM to IP: Getting the Ball Rolling" Level 3 Communications, 2008.
- [2] Das, Kaushik. "IP PBX (IP Private Branch eXchange)" *Ipv6.com Tech Spotlight*. Ipv6.com, Inc. Retrieved 7 May 2010.
- [3] Dennis, Tony. "Evolution of the telephone explained" *The IT Examiner*. ITExaminer.com November 27, 2008. Retrieved 10 May, 2010.
- [4] Garey, Michael R. and Davis S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Murray Hill, New Jersey: Bell Telephone Laboratories, 1979.
- [5] Goeller, Lee. "The PBX: What It Is, How It Works" *The BCR Manual of PBXs*. Business Communications Review, 2006.
- [6] Harte, Lawrence, and Robert Flood. *Introduction to Private Telephone Systems: KTS, PBX, Hosted PBX, IP Centrex, CTI, iPBX and WPBX*. 2nd Edition. Althos Publishing, 2005.
- [7] Hettick, Larry and Steven Taylor. *Technology Backgrounder on Telephony and VoIP Basics*. Distributed Networking Associates, Inc, 2004.
- [8] Leader, Sheldon. *Telecommunications Handbook for Transportation Professionals: The Basics of Telecommunications*. U.S. Department of Transportation: Federal Highway Administration, September, 2004.
- [9] Lightning, Knight. "Introduction to PBXs" *Phrack Magazine* Volume 1, Issue 3. 1986

- [10] Macario, John. "Intro to Voice over Internet Protocol: What does VoIP Mean for My Business" XO Communications, 2009.
- [11] Melamed, Haim. "The Rise and Fall of Centrex Systems" TMCnet, September 2005.
- [12] Quick's Staff, "Back to the Future for Centrex" Quirk Enterprises, Inc. April, 1991.
- [13] Stevens, W. Richard. *TCP/IP Illustrated, Volume 1: The Protocols*. Reading, Massachusetts: Addison-Wesley Publishing Group, 1994.
- [14] White, Curt. *Data Communications and Computer Networks: A Business User's Approach*. 5th Edition. Boston, Massachusetts: Course Technology, 2009.
- [15] Wikipedia Contributors. "Telephony Application Programming Interface." *Wikipedia, The Free Encyclopedia*. Wikimedia Foundation, Inc. Retrieved 16 March 2010.
- [16] Wikipedia Contributors. "Model-view-controller." *Wikipedia, The Free Encyclopedia*. Wikimedia Foundation, Inc. Retrieved 10 June 2010.
- [17] Wilde, Candee. "IP PBX Basics" *Information Week*. InformationWeek.com, May 14, 2001 Retrieved 5 May, 2010.

Appendix A: Glossary

ADO – ActiveX Data Objects, a Microsoft programming interface for database access.

CO - Central Office, the local phone company's building that houses an exchange which connects local phone lines to the network of other central offices around the globe.

Codec - Coder-Decoder, a device or computer program capable of encoding and/or decoding a digital data stream or signal.

Centrex - Central Exchange, a service offered by COs to offer PBX-like features to offices.

DSP - Digital Signaling Processor, a specialized microprocessor which is used to manipulate digital signals.

DTMF - Dual Tone Multi-Frequency, a standard for signaling on analog phone lines.

G.711 - codec for pulse code modulation of voice frequency.

G.729A - codec for Audio Data Compression commonly used for VoIP.

IP - Internet Protocol, a protocol for communicating data on a packet-switched network such as the Internet.

IP Telephony – Another name for VoIP.

MVC – Model-View-Controller, a paradigm for class design separating the data model from the presentation of the data, and the logic for operating on the model.

PBX - Private Branch Exchange, a phone system that connects office phones to each other and allows routes outbound calls to the CO.

PCM - Pulse-Code Modulation, a digital representation of an analog signal.

PSTN - Public Switched Telephone Network, the network of COs that connect phone calls around the world.

PTMC - PCI Telecom Mezzanine Card, a PMC form-factor card for DSP resources.

ITU – International Telecommunications Union, an organization that sets standards for the telecommunications industry.

VoIP - Voice Over IP, a general term for the transmission protocols for delivering voice communications over an IP network.

TAPI - Telephony Application Programming Interface, a Microsoft API on Windows computers to use telephone services.

Hunt Group - A group of extensions assigned to a single group extension – so that if the group extension is dialed it will find an available individual extension from within the group to handle the call.

SysPort – System Port, an extension used to represent a TAPI resource on the PBX described in this thesis.

TDM - Time-Division Multiplexing, the process of sending multiple digital streams of information on a single communication channel by dividing the channel into timeslots.