Unifying Unified Voice Messaging

Jon Finke – Rensselaer Polytechnic Institute

ABSTRACT

Roughly 18 months after installing a unified voice messaging system, we picked it up and merged it into our pre-existing production email domain. This paper deals with both technical aspects deploying a unified messaging system, as well as the cultural shock of merging the very different operational domains of Email support with Telecommunications support into a shared support model. As an added bonus we will discuss the merging two Exchange/Active Directory worlds into one with minimal impact on the existing users of both systems. Lastly, we will discuss some issues of operating a partially unified voice messaging system.

Introduction

After 12 years of service, our campus voicemail system failed. We were finally able to restore service after a 10 day[1] outage, but it was now clear to everyone, that this system needed to be replaced. Although we had been evaluating replacement voicemail systems prior to the failure, this changed the process from a theoretical exercise into a crash product selection and deployment exercise. Several years earlier [3], our division had implemented a Microsoft Exchange email service and this was in use by about 700, mostly administrative users. The eventual proposal was to install a Unified voice messaging system from Cisco that uses Exchange as a back end message store.

One of the attractive features of this product was unified voice messaging – that is your voicemails would appear as emails with .WAV files attached in your inbox. You could listen to them via the telephone like normal voicemail, or you could ‘‘play’’ the message via your email client. Since they were email messages, you could also use your email client to file or forward the message to other people. You could also get the system to ‘‘read’’ your email to you over the telephone (although good spam filtering is helpful here). Another benefit is that the new system could join the existing Windows 2000 domain and use the existing exchange servers. This allows us to leverage both existing hardware and existing support staff. The proposal went forward to the President’s cabinet and emergency funding was allocated.

Unfortunately, no one had actually talked with folks who supported the existing Exchange installation and we received some push-back from them when we mentioned making changes to the Active Directory schema and creating another 5000 mailboxes to go along with the 700 they currently supported, as well as adding this new service with full administrator rights to the domain. So instead, we found a few more servers and installed a second Windows 2000 domain with it’s own Exchange server. Although this allowed us to cut over from the creaky old voicemail system so we had new hardware, it created some new problems for us. The first was that we now had to support a number of windows machines, along with a our own exchange server, and between our PBX Switch engineers and our Network Engineers, we were not in a happy place [6]. The second, and perhaps bigger problem, is that we had promised to deliver Unified Messaging, not just a replacement for the old, telephone only interface, voicemail system. In order to deliver on this promise, we had to find a way to merge the two domains. We also wanted to avoid another hard cut over like the initial installation, where all of our users had to set up their mailboxes and record greetings but then lost all of their existing messages.

[1] Some replacement parts were obtained via EBay

Initial Installation

Our initial installation was a replacement of our Octel voice messaging system with the Cisco Unity voicemail system. Since we were setting up a stand alone Exchange domain for this system, we did not have to worry about unified messaging at this point, just plain old voicemail accessed via a telephone set. At that time, we had an Intecom S80 PBX and a few experimental (for us) Voice Over IP (VOIP) phones. The PBX communicated with the voicemail system using a bunch of analog telephone lines and a single RS232 serial line for control and switching information. This single interface would prove to be a major factor in how we managed the switchover. Since the PBX only understood (and could communicate with) a single voicemail system, we had to do a hard switchover; bring up the new system, move the wires from old to new, and shut off the old system.

The Unity product includes a web based tool for administration, as well as several bulk load tools to process CSV files. Through these tools, an administer could create and destroy subscribers (which includes the exchange mailbox). We had two issues with this approach; the first is that a GUI interface is just too much work to manage ongoing changes,[2] and the

[2] We had previously automated the creating and expiration of student voicemail assignments based on room assignments in the Housing Office database.

second is that we would have the creation and deletion of Exchange accounts in the hands of Telecom staff[3] instead of the **Exchange Administrator**. We needed a way to give telecom staff the ability to add and delete voicemail boxes, and we wanted to automate and integrate with other systems as much as possible. Although the Cisco interfaces had some provisions to enable individual users to do some of their own maintenance, we also wanted the ability of departmental administrators to manage the people in their departments. We have found the departmental administrator concept a very useful and powerful thing on our directory (white pages) deployment [2] and we wanted to be able to use that with the voice mail system.

In our deployment of our original windows domain, we had automated a good deal of our data flow [3] and were comfortable with that concept. But in order to enable the distributed control and management that we wanted, we had to strictly enforce our policy and practices. To do this, we really needed to replace the Cisco tools with our own tools. To this end, we wrote our own web tool to allow the management of mailboxes and call handlers, and enforce our naming conventions and additional record keeping along the way. One of our new rules, is that all mailboxes need to be ''owned'' by either a department or an individual, thus the new tool can display all of the mailboxes (and call handlers) owned by a specific person or department. In fact, the only way to create a new mailbox or call handler, is to first select the owner, and from there, creation can continue. The tool also attempts to avoid errors and do as much of the ''thinking'' as possible. When a new mailbox is created, the tool first checks to see if that number (telephone extension) is already assigned to another mailbox. If it is in use, it will note that fact, and give several options such as expiring the existing mailbox which would free up the number, or creating an Enhanced Call Processor.[4] Creating the ECP automatically and then assigning the original mailbox to ''1'' and the new mailbox to ''2'' saved a few steps for the telecom staff and helped avoid procedural errors. This would also generate a ''work item.'' In the initial deployment, a staff member would still need to record the greeting (''Press one for Sam, press two for John'').

One of the tasks our telecom staff dreaded at the start of the semester, was setting up all of the student ECPs. They would have to record greetings for over 800 call handlers (''press one for Marquia, press two for Sharon''). We made two changes to this process. The first was to use some Text to Speech routines to preset the subscribers name when we created a mailbox. We were actually pretty pleased with how well these routines did with many of the names. The second change was when we set up the call handler, we would generate the greeting by concatenating the names for each mailbox with the appropriate ''press xxx for'' phrases. This has the additional feature that if the subscriber records their own name, when we regenerate the greeting for the call handler, it will use the subscribers name recorded in their own voice.

We had previously written a tool to manage student voice mail accounts. This tool would look at the room assignments from residence life, and automatically assign mailboxes based on the room (which determined the phone number.) In the earlier version, this would generate a CSV file that would be loaded into the Octel system. This tool was modified to create the mailboxes and populate the appropriate call handlers. In this case, there were two types of handlers, a ''Direct Transfer to xxxx'' for single rooms and ''Student Residence ECP xxxx'' for shared rooms. Each student mailbox was assigned a six digit ''directory number.''[5] By using a call handler to access all student mailboxes, we did not need to worry about changing the extension assigned to mailbox, and as students changed rooms, they could keep the same six digit directory number, even as their room number (and phone number in their room) changed.

Subscribers, Call Handlers and Distribution Lists, Oh My!

A detailed discussion of the Unity Voice Messaging system would take a few days, and during our deployment, we purchased a web based training package to help jump start the Unified Messaging Team. But a brief overview of how Unity works would be helpful. The core building block of the Unity system is the call handler. This defines how a call is processed, what messages are played, what options the caller has during the call, and what happens after the greeting has been played.

Each call handler has some basic attributes including a name, an owner (which is a unity subscriber or unity distribution list) and an optional recorded name and an optional extension. It can also have a number of different types of greetings; a standard greeting, a ''closed'' greeting to be used when the university is closed, and alternate greeting, a busy greeting and an internal greeting. Each of these greetings can be enabled or disabled (the standard is always enabled), a source for the greeting which can be a recorded message (stored in a .WAV file, a system greeting (if there is a recorded name, it will be ''*RECORDED NAME* is busy or unavailable'' otherwise just ''Extension *XXXX* is busy or unavailable'') or simply blank. There are some switches to control if caller input is accepted during the greeting, how many times to replay the message and finally, what to do

[3] It isn't that we don't trust our Telecom staff, but they do not have the training to administer Exchange and they are not in the department that is responsible for the Exchange service.

[4] ECP – A TLA we inherited from the Octel voicemail system – press 1 for John, press 2 for Sam

[5] Our campus uses four digit dialing – these six digit numbers are not directly dial-able, but can be dialed from within the voicemail system.

after the greeting is played. These options include taking a message, transferring to another subscriber or call handler, transferring to another extension or simply hanging up.

Call handlers can also accept caller input – a touch tone key press that can transfer to another subscriber – or any of the options for after playing the greeting. This can be configured for each of the 12 touch tone key presses. You can also configure how long a message can be left, who the message goes to, and what happens after the message is recorded. There are more options and settings available, but we are not currently using all of them.

The next part of the Unity system is the subscriber. This is a call handler with a message store attached, and some user information such as a PIN, a display name for the directory (both Active Directory and some directory lookup functions in Unity) and the location of the mail store. In general, the messages are stored in an Exchange server. This allows subscribers to access the messages via Outlook, or via the telephone (The Unity system is also an Exchange user that can open any mailbox). There is an option for messages to be sent immediately via SMTP to some other mail server. Although this means they can't access their voicemail via the telephone, since it is out of Unity's reach, they do get it via their preferred email client. This option is attractive to some of our users who don't use Exchange.

Unlike stand alone call handlers, subscribers (or their call handlers) must have an extension. Subscribers can also have alternate extensions, which allows a person with more than one phone line to have all of their calls go to one voice mailbox. Another attribute of subscribers, is notification and the message waiting indicator (MWI). The MWI is set when there are ''unread'' voice messages in their inbox.[6] The system can also notify you of voicemail (and emails if you wish) via phone calls, pagers or email notification.

Another part of the Unity system, are distribution lists. These can be used to send group messages to folks and they can also be the owner of call handlers. We have two categories of automatically maintained lists, an ''All Hands XXXX'' list for some departments who request it, and ''department admin XXXX'' list with the department administrators for any department that owns call handlers. These are created automatically when a call handler is created. The owner of a call handler can update the greeting, this allows several folks in a department to maintain the greetings easily.

Although Unity has an ''owner'' for each object, we maintain our own owner information in our Oracle database which allows us to delegate control and administration more effectively.

[6]There is a childlike attraction to selecting ''mark as unread'' in Outlook and having the red light on your phone turn on.

Functional Layout

In Figure 1, we have a diagram of our configuration. We have our old legacy phone system (PBX), which is connected to the voicemail system via both analog lines for voice traffic, and a serial line for control information. This PBX is also connected to our Call manager to provide a path for VOIP phones to talk with PBX based telephones. Both the Call manager and the PBX are have trunks to connect to our regular phone carriers for inbound and outbound phone service.

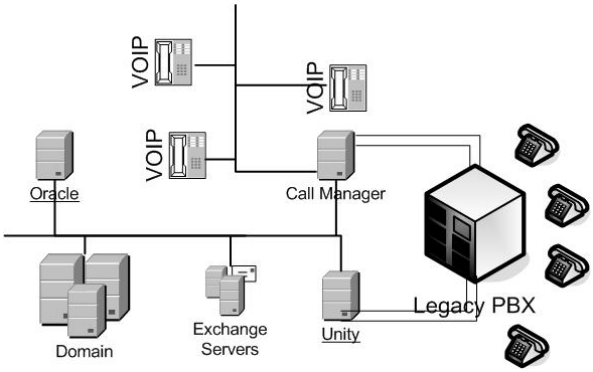

Figure 1: Function Diagram.

The call manager and the voicemail system (Unity) are also connected to the campus network, albeit behind firewalls. They use the network to communicate with each other, and Unity also connects to the Exchange message store. The call manager also uses the network to communicate with the individual voip telephones. The configuration (user provisioning at least) of the active directory domain, the call manager and the Unity system are all driven by an Oracle database. This database also handles our Identity Management functions and also drives our LDAP directory databases and many other aspects of our operation.

Write Only Memory

In our initial version of Unity, Cisco offered a C# API [7], so we built a process (the Unity Queue Runner) that would run on the Unity server, accept commands via a queue in our database [5] and process them via the API. We were pressed for time on this installation (recall that the original system had already failed hard for 10 days and was showing signs of additional failures) and this command path was one way only. We had to issue commands and assume that they worked. We kept track of what we thought we had done, and hoped that we were right.[7] At this point, we had no way to read back the contents of the configuration database on the Unity server.

[7]This actually worked quite well, after 18 months of operation, we had only had a handful of errors – all due to human intervention

What's In A Name

We ran into a few issues with our naming conventions. One of the first problems we had, is that when the Unity Queue Runner was passing commands to Unity, it would reference the mailbox or call handler by it's name. On a few occasions, someone using the Cisco tool, had changed the name directly in Unity. As a result, the command would fail – since the object wouldn't be found. Although there were only a few of them, they generated quite a few headaches for the technical staff, as well as problems for the front office folks. Eventually we were able to convince the front office folks that ''never change a name,'' meant exactly that. We actually could go into the SQLServer database on Unity to find and fix them, but it was very annoying.

A more common problem, and very annoying, is that we used names like 'Fac/Staff ECP nnnn or 'Student ECP nnnn' for our call handlers. What we did not know at the time, that the call handler search function on the Cisco tool only used the first eight characters, and there was no way to search by extension. Thus, the search tool would return a list of all 'Fac/Staff ECPs,' an unsorted list at that. This did have the minor benefit of making the Cisco tool almost useless for working with call handlers, which forced the front office staff to use our own tools and reduced the first problem a bit.

Another part of our early naming convention, was to embed the department name or abbreviation into the name of call handlers for departments (ECPs and Voicemail trees). While this made some degree of sense for the Octel system (where we inherited the practice), it proved to have several issues with Unity. In the original deployment, we ran into departments that did not have abbreviations, and the resulting name was too long. Even with the abbreviations, the search problem mentioned above hit us just about every time. The other issue that we ran into, is that departments changed their name and abbreviations periodically, which then meant that we had to rename call handlers, which we can't easily do.

Since our tools to manage the Unity system had additional fields to capture owner information (and whatever else we wanted), we no longer needed to overload the call handler names like we did with the Octel system. Some of these naming problems we were able to correct before our second upgrade; we were able to destroy and recreate all student call handlers over the semester break. In other cases, we kept the old (longer) names and new call handlers would get the shorter names following the new convention.

Octel to Unity Switchover

The migration from the old Octel system to the Cisco Unity system was in some respects easier. People understood that the Octel system was about ready to die, and had lower expectations. On the other hand, there was a lot more busy work required. We were able to get a dump of the Octel configuration, which included the directory number, class of service, the owner ''name'' (this was a free text field, with no validation) and some other configuration information. We needed to map this information on to the new system, and assign owners to everything.

The student voicemail was pretty easy – we were already generating those assignments in Oracle and loading them into the Octel; we wrote some interface code, and that process could talk to Unity. That left the fac/staff mailboxes to be migrated. We wrote a tool that helped us assign owners to Octel mailboxes (and save this info in Oracle). Some of this could be done automatically. If an Octel mailbox (directory number) was listed exactly once in the telephone directory, and that person was an active employee, we made the match. The rest needed to be cleaned up manually. The tool would display unmapped mailboxes, and the front office staff could work through the list, displaying each one and attempting to match. The tool would display matching directory and billing information.[8] Other choices included marking the mailbox as obsolete, or just tossing it to the end of the list. Eventually, all Octel mailboxes got an owner or marked as obsolete.

The matching tool let us work with preliminary data (an earlier snapshot) and one week before the switchover, we froze the Octel database – we had campus announcements out, and after the freeze date, we would process no more changes to the Octel system. We then took that snapshot and finished matching mailboxes with people. At that point, we were able to pre-create all of the new mailboxes on the Unity system. Up to that point, we just had a few pilot users, mostly accessing it via the few VOIP phones we had deployed at that time. Another tool we wrote allowed individuals to pick up their voicemail PIN via the web. We had previously done this for students, and now we were able to expand this for all users.

The cut-over was scheduled for a weekend at the start of the summer (June 6th) – we swung the wires, moved the direct access numbers (there was a number for folks to call to get their old messages) and we were done. On Monday, everyone with a mailbox had to set it up, record their name, set up their greetings, change their PIN, and remember to call the special access number to get any last messages from the Octel. Messages were not transfered from the old system. After 60 days, the old system was turned off and removed from service. If you had not picked up your old voicemail, too bad. We also made the decision to NOT offer voicemail service for students during the summer. This gave us time to handle problems with the fac/staff mailboxes.

Preparing to Merge the Domains

After a year of running a stand alone voice mail system, the decision was finally made to move the

[8] Although we do not charge for voicemail, we often would have owner information for a number (telephone) that we did bill for in the telephone billing system. Sadly, this information did not include a validated owner.

voicemail domain into the regular email domain. This was required to allow us to offer fully unified voice messaging, and would also reduce the support overhead coming from two exchange domains. Shortly after our initial voicemail roll-out, we started a roll out of VOIP service to some new buildings. Part of this roll-out included our own control interface to the Cisco Call Manager, similar to what we did for the Unity project. Unlike the Unity interface, this one used XML [4] and was bidirectional. One of the side effects of this, was the appearance of VOIP phones on the desks of staff who were involved with the Unity project. This was mostly to enable testing of the VOIP service, but proved to be handy in other ways. One of the very valuable features, is that unlike the Intecom PBX, the Call Manager (the ''brains'' of the VOIP system) understood how to talk to more than one voicemail system, and could talk to them via TCP/IP.

When we first deployed Unity in the Stand Alone configuration, we contracted with IBM to provide us with a consultant to assist us in the planning (including system requirements), configuration and deployment. Ben worked out very well, he certainly knew his stuff and got us going. As an added bonus, Ben was a Rensselaer graduate, so he was familiar with the campus and the general operation. But going forward, we wanted to break the dependency. It got annoying to the technical staff, that when some issue came up, the first reaction was to ''Call Ben,'' before even attempting to handle the problem ourselves. The decision was made to handle the migration entirely in house; we had to develop our own expertise.

This is Only a Test

We decided that we should replace the Unity server as part of the upgrade. One of the challenges we had with the initial deployment, is that once we went live, we had no test system – all changes had to be made on the production system. It also meant that we had no backup system available. Although the Unity server was a pretty standard box running Windows 2000, it did have some special interface cards in it to talk to the PBX. In the 12 months from the original deployment, the cost of the server hardware came down considerably and since it was going into an existing domain, we did not need domain controllers, an exchange server, etc. So although the new server was part of the production windows domain, it was not in production and we could use it for development and practicing the migration.

When we originally installed Unity, we knew that we would eventually be merging into the win.rpi.edu domain. But since we were concentrating on the move from Octel to Unity, this sort of slipped our mind. One of our first projects to get ready for the merge, was to modify every package, interface, queue and tool to understand the concept of more than one domain. Although the changes were generally pretty small, it touched everything we had developed. Although a Domain_Id field was added to just about every table, it really only mattered for the call handlers, subscribers and extension (directory number) tables. All other objects were associated with call handler or subscriber and could inherit the domain where needed. The main Unity interface tool had a switch added, to let the user select which domain they were working in. All searches, displays and other functions would be limited to records from that domain. Up until the cut-over, this switch defaulted to the stand alone Unity installation. The other key place for this information was in the command processing stream. Although the Meta_Change_Queue understood multiple queues, the Unity_Maint package had to select the appropriate queue for each command. During the testing of these changes, it was amusing to see a ''Create Call Handler'' going to the new system, and all of the caller input configuration go to the old system.

We got the new Unity server installed, and made it part of our production Windows domain (win.rpi.edu). Despite having a queue runner program working on the stand alone system, we wanted to merge the queue runner for the Unified[9] system with the original ADSI_Sync program that we were already running to maintain the windows domain. We also needed to be very cautious in making changes in this domain – just about all of our administrative staff was already using this exchange server and active directory, and disruptions or bad entries were not acceptable. Up until we got very close to switch over, the queue runner process was being run on demand, often times with break points in it, so we could be very sure of what was happening. Another significant operational change, is that although the front office telecom staff got admin accounts on the new Unity server, these accounts were read only. Their only create/modify access was via the interfaces we developed. Fortunately, running the stand alone system gave us time to make available most, if not all, of the needed commands and functions.

Code Generator

Our command interface was much like a checkbook. We made transactions and recorded them, and so maintained a picture of what we thought was the state of the system. But we never had the facility to ''balance'' the checkbook, and it was possible to miss transactions. In addition, one of our objectives with the cut over to the Unified system, that no one would know that we had done it – greetings needed to be preserved, PINs stay the same, and so on. These changes were all made directly in Unity via telephones, often by the end users and our system never saw them.

After our initial cut over from Octel to the Stand Alone Unity system, our next big project was a Voice

[9] We eventually settled on the name ''Unified'' for the new system and ''Stand Alone'' for the old one.

over IP deployment. Despite there being a GUI tool from Cisco, we wanted to have our own interface the the Call Manager[10] to handle provisioning of telephones with better integration with our billing and network systems. The call manager used a SQLServer database internally and provided an XML based interface that could both make configuration changes and return information from the database. It also provided a general SQL interface so that we could make arbitrary queries. Implementing our front end required that we maintain shadow copies of many validation tables in our Oracle database. Writing this interface code was tedious and prone to error, but was also very consistent in format. We developed a code generator; we would define the Oracle shadow table, and then point the code generator at it, and it would spit back paragraphs of PL/SQL code and record definitions that we could then use.

When we returned to the Unity project, we also wanted to get data out of SQLServer, and we had a very limited interface[11] that would allow us to query the SQLServer database on the Unity system. We wrote a PL/SQL package for Oracle, that would generate a query to be passed to SQLServer, that would concatenate multiple columns into one, doing appropriate type conversions to produce a CSV format return that we could then write more PL/SQL code to parse and store. Getting all of this to actually work (part of one query required seven single quotes in row to get the appropriate single quote in the final result) cried out for automation, and so we wrote another tool to handle all of this. One of the things that SQLServer has, is a set of tables known as the data dictionary, that define all of the table and columns (and other things) in the database. With this, we could query the data dictionary and get back descriptions of all the table and columns in the database and store them in the Oracle database. We then added a tool that allows you to select a table, and columns of interest, the type of query (just one record based on a primary key, a list of all keys in a table, etc.) and eventually build up a set of queries of interest. Then with this, we would generate an entire PL/SQL package with record definitions, parsing routines, query routines, etc. This package could then be compiled in Oracle in used by other parts of the system.

The first application of the code generator, was to regenerate the interface used by the code generator itself to get data dictionary information. The code generator made it very easy to access almost all of the data we needed from the SQLServer database on the Unity system. We could get lots of the mailbox configuration information, some statistics about the mailboxes and even the PIN. We actually could only get an MD5 hash of the PIN, but we discovered that when you pass a 32 character hex hash of a PIN to the `Set_DTMF_Access` routine, it would recognise it as already hashed and set it directly. Thus although we could not determine what someone had set the PIN to,[12] we could determine that they HAD set it, and we could copy that value to the new system.

Call Manager Integration

Between our initial deployment and the migration to Unified messaging, we had one major change in our environment; the VOIP deployment. When a voice mailbox is configured, you need to indicate which phone switch it is on. This is so Unity knows which switch receives the MWI. As people were moved from the PBX to VOIP, we sometimes forgot to update Unity, so a few days after someone got moved to VOIP, we would get a trouble call that their message waiting light wasn't working anymore. We modified our call manager configuration tool, to check the Unity configuration and change it automatically if needed. This greatly reduced the number of MWI trouble calls we were getting. This type of cross system integration is a nice feature of doing our own tool development.

Recorded Names and Greetings

Much of the subscriber configuration information was stored in the SQLServer database, and the actual voice messages were stored as email on the Exchange server. But the recorded names and greetings were stored as `.wav` files on the file system on the Unity server. We could at least get the name of the file from the SQLServer database. Our next step was to define a new command for the queue runner that would ask it to open a sound file and save it into the Oracle database. The file would get converted to Hex, passed to a an interface routine in Oracle, that would convert it back into binary and store it as a BLOB.[13] We were already saving ID card photos as BLOBs and we had reasonable support for managing this information and spitting it back out via web tools. This was pretty neat, as it allowed our admin tools to ''play'' the greetings and recorded names for subscribers and call handlers. We were able to ''recognize'' names that we had recorded and greetings that we had generated by the file names, so we didn't bother to actually save these in the Oracle database. Collecting names and greetings was not a fast operation.

Since it took so long to copy all of the recorded names and greetings, we next wanted to develop a system that would just get new recordings. Unfortunately, although the SQLServer database had the file name, it did not know the file size or last changed information. So, we added yet another command, a request for file information. When the queue runner got this, it would get the file size and last data info, and store that in the

[10] The Call Manager is the ''brains'' of the VOIP system.

[11] Only the first column of the query would be returned and the total returned data was truncated at 32 Kbytes.

[12] OK, 5-8 numeric digits and known hash, we could brute force it.

[13] Binary Large Object – how Oracle can store large binary data values such as a .WAV file.

Oracle database. This made for some complex update code; it would check to see if a subscriber had a recorded name. If it did, it would send a request for the file info for it. But it might be minutes or even hours before it got a response (an update in the table), so it would request info on a lot of files, and mark them all as pending. Some time later, it would look at the pending to see if they had been updated, and if so, it would then ask for a fresh copy of the file. A more direct query approach have made writing this a lot easier. But in the end, we had a system that could, in the span of less than a day, ensure that we the most recent greetings and names.

Migration Tool

When we moved from Octel to the Stand Alone Unity system, we had written a migration tool to help us with that project. So, we wrote a new tool that would let us examine all of the call handlers and subscribers on the stand alone system and figure out how to map them to the Unified system. This also provided a platform to test the migration procedures. When we had installed the Unified system, we did not populate it with subscribers, and just set up a few call handlers and templates to provide a starting point.

The first folks migrated were part of the pilot group – generally the technical people who were actually doing the migration, and would not (or could not) complain if their voicemail got messed up. Since these were going to be ''unified'' clients, we did not need to change their mailbox name. The first step in migrating someone, made an entry in the migration table, and changed their status. The first state was generally ''wave wait'' – which also generated a request for the sound files. Once the sound files arrived, the new name could be set and when we were ready, we pressed the ''create'' button. As time went on, we got better, identifying more special cases we had to handle, and getting all of the configuration information that we needed. These early adopters did not get their messages copied over until much later.

The tool was then changed to list sets of subscribers, based on class of service, who were not yet in the migration table. Check boxes made it pretty simple to decide if we wanted to migrate them, and what their new mailbox name should be. This allowed us to review every subscriber and call handler well ahead of time and get them started down the path. Once a subscriber or call handler was created, a link to the new entry was also stored in the migration table, so we built up a record of everyone and everything that got moved.

Prior to Cut-over Weekend

We had originally planned on doing the migration during the week between Christmas and New Years. This had the advantage of the students being out of the dorms and most of the staff on vacation. However, the wife of the exchange administrator was due to deliver their third child[14] that week, and since it wasn't critical that we migrate, we decided to wait until the entire upgrade team was available. The cut over was moved to the first weekend of Spring break. The real driving force was that was the first weekend that all members of the upgrade team (at least the key members) did not have other obligations. It certainly is nice to be able to select a time that works well for everyone.

Despite having had the pilot group using the new server, and many of them still unified, there were still things that we were unable to test. For example, we were pretty sure that creating 5000 new mailboxes on our exchange server would not have any unexpected side effects, but we wouldn't know until we tried. So two weeks before the cut-over, we started creating the new mailboxes using the migration tool. After the first few batches of 100 went through ok, and backup ran ok, we sent over 1000, things again looked ok, and we were able to finish the bulk of them a week before the cut over. We also added some bulk processing options to the migration tool, to speed the processing along.

How Long?

Another process we started early, was setting up the sound files. We did not want to load the sound files too early – users of the voicemail system were still updating their greetings and we did not want to ''freeze'' changes until shortly before the cut over. But since our collection process was able to detect files that had changed, we started loading sound files, and a good thing we did! Although everything worked in tests, and the extraction of all sound files only took 36 hours, moving the sound files from the database and converting them back into .WAV files on the Unified system was very slow and our initial estimate was almost two weeks! Some research into that determined that most of the time was in the hex to bin conversion, and some web searches revealed discussion of just how slow some of the library routines that did this are. An alternate library gave us a speed increase of 20, which was good enough to get us in under the deadline.

We wrote another program that would copy mailboxes from the Stand Along exchange server and deposit their contents into the Unified exchange server. It would take a crosswalk file, since almost all of the mailbox names were changing as part of the process. This also was going to append email, since members of the pilot group already had exchange mailboxes as their primary mail service. We couldn't figure a good way to test this – although we had a Exchange test domain, it is isolated from the ''real world,'' since it tries very hard to get out. We were also concerned that any testing here would raise more questions than it answered. Since this program was based on some tools we had used during an Exchange recovery operation, we had some feel for the timing,

[14] Katia Birgit Hill, 7 lb, 13 oz

and we were confident that it could run in less than two days – well within our planned upgrade window.

Cut Over!

Friday at 6:00 PM rolled around, and we started going through the step by step upgrade plan. We had taken some concepts from Brent Chapman's Invited Talk [1] from LISA last year, and we had a communications person and a communications plan, so that people who needed to know what was happening, were informed as milestones were reached (or not).

In the months leading up to the cut over, we had developed an implementation plan including time estimates, deadlines, dependencies and staff for each step. We also developed a more detailed tactical plan for the upgrade weekend, including communication schedules (i.e., – we were going to give the CIO a status report at 9:45 PM, he wasn't supposed to call us prior to that time.) The plan even included dinner menu selections for the team members. Some aspects of the plan called for processes to run – instead of standing around watching, most of the team could eat dinner.

We also had a test plan, which involved other people from the division testing aspects later on in the weekend, as well as specific things to test. We also requested that these testers try those same tests on the existing system prior to the cut-over, so that they would know what to expect. We did not want to waste timing troubleshooting a problem that existing on the old system and was ''successfully'' migrated (and we found bunches of these.)

Final Results

We had reserved the entire weekend for the upgrade. Within an hour of the start, the wiring changes were done and the new system, was accepting messages for people on the PBX. Within an hour after that, we had moved most of the VOIP people over. We encountered some problems that did not show up in testing, where some phones did not want to change their voicemail profile and continued to direct calls to the old system. Migration of old messages was scheduled to start Saturday at 9:00 am and by noon, we had copied all of the voice messages over from the old system to the new system, and although we were still doing testing, for all intents and purposes, the upgrade was done and most of our users did not notice any outages. Their messages were available, their greetings and recorded names were intact and their PINs still worked.

Oops

We missed a few things. Although we were finally able to get all of the VOIP phones switched over, we overlooked the tool used to add new phones – it was carefully assigning new phones to the old system. Fortunately, once we figured this out, we were able to correct it and had the users get their messages from the old system. A few months after the cut-over, we were still occasionally finding new VOIP phones with voicemail on the old system. This was finally tracked down one module that had a literal string with the PKID of the old voicemail profile embedded in it. This was noticed when the old voicemail profile was deleted from the VOIP Call Manager and certain configuration functions started to fail.

We had instructed our users that changes made after a certain date (a week before the cut-over) may not be migrated. We had a few cases where people recorded alternate greetings and enabled them and left for vacation. Unfortunately, the process to migrate greetings was not synchronized with the process that copied the state of the greeting switch, so as a result, a few people had voicemail greetings wishing folks a ''Happy Thanksgiving'' during the week of spring break.

The Unity system (at the release we had), could be configured to work with two phone systems. So in the initial deployment, the IBX was switch 0 and the call manager was switch 1. When the new Unity system was installed, the first switch that was configured was the call manager so it became switch 0, and the IBX was switch 1. This didn't matter too much, since phones are generally configured by name. What this did do, was change the default switch for lots of mailboxes. Since this didn't matter for mailboxes that didn't have a real phone, this shouldn't be a problem.

One of the nightly jobs the Unity server runs, is to refresh the Message Waiting Indicator (MWI) on all the telephones. This is that nice little red light that tells you that you have a message waiting. Or for the folks with standard telephone sets, the ''stutter'' dial tone, so that when you pick up the phone, you get some indication that you have voicemail. Of course, we have found that many of the folks who have the standard phones, don't make a lot of phone calls, and so never pick up the phone to hear dial tone, and so never learn that they have voicemail. This also doesn't work well where you have shared phones. Which mailbox controls the single light? One option here is to use email notification – when you have a new voicemail, it sends you email telling you about this. While this makes no sense for unified people (sending you an email to tell you that you have an email?), it is nice for folks with just the basic voicemail service. Many of our mailboxes are set up with 5 or 6 digit ''extensions'' (fac/staff shared phones that were migrated from the Octel system often kept a 5 digit extension, where the first 4 digits matched the phone, and the 5th was the ''press N'' value.) So every night, Unity would attempt to update the MWI setting for 5000 phones – many of which did not actually exist. The connection to the IBX was a serial line – so this update would take almost an hour. While this was going on, new MWI changes would get stuck at the back of the queue. This wasn't a big problem at 4 am, but was mildly annoying.

When the new system went live (or actually when created 4900 more mailboxes), we noticed that the default switch was the call manager and not the IBX, but hey, the update of the MWI would run a LOT faster via TCP/IP. Shortly after that, we noticed that the job that was processing the call detail records (CDR) for phone billing, was timing out – it was taking too long to run. Further investigation revealed that each of these MWI updates was generating two CDR records for each mailbox, one for invalid extension and one for service not available. This increased the number of CDR records each day from around 2000 to over 10,000. This was also filling up the CDR space on the call manager a lot faster, and wasting space on the billing system. For a short term fix, we wrote yet another process to pre-delete these CDR from the call manager before the billing job runs. Who would have thought that such a trivial decision would prove to be so annoying.

Another thing that bit us was with licensing limits. We originally had 5000 licenses, and our usage was getting really close to the line, so we got 50 more. This involves loading a new license file and restarting the system. We did this and operations continued. However, this happened after we had built the new system, but before we cut over, so it was still running with the license limit of 5000 and not 5050. Shortly after the migration, we learned that you do NOT want to exceed your license limit. An updated license file, a brief outage and full service was restored.

Conclusions

In retrospect, doing the initial installation as a stand alone exchange domain, and then later migrating it into our production email domain was a good thing. We had no operational experience with the Cisco Unity product, and despite having run the previous voicemail system for 12 years, there were some significant differences. We had to make a lot of decisions when we first deployed Unity, and several turned out to be sub optimal.[15] We were able to correct a number of these as part of the migration process.

We also got a test system out of the whole process. One of the first things we want to try, is does our system restore actually work.

Are We Done Yet

A rather annoying oversight from an implementors standpoint, is that we had not defined a way to determine when the migration was ''done.'' There is a different mindset, sense of urgency, etc. between ''Cut over Weekend'' and ''Normal Operations.'' It may seem like a trivial point, but it has an impact on the people involved. One of the problems we had, is that there were things that were configured wrong on the old system and we ''successfully'' duplicated those wrong configurations on the new system. It was nice to finally just fix problems as they were encountered, and not worry was it a migration problem or a new or even an existing problem. It was really annoying to get a complaint about someone's message waiting indicator not working, and after an hour of digging, determine that it NEVER worked in the first place.

We also left the old system on line – perhaps for too long. This allowed some folks to continue to use the old system without anyone knowing it – which led to some confusing problems. This was only a problem with the call manager. The PBX was a hard switchover, a physical cable move, but the call manager continued to talk to both system, and it wasn't until we deleted the voice mail profile on the call manager the referenced the old system, did we shake out the that last few references to it.

Culture Clash

The email folks and the voicemail folks had developed many years of policies and procedures. With the merging of the two worlds, these differences need to be identified and resolved. For example, in the past when a new person got a telephone, they would often have to flush out the messages and greetings from the previous owner of the phone. We have NEVER re-used email accounts! What was once a reasonable and acceptable practice, is now obsolete!

Another issue we have the actual deletion of mailboxes. In the stand alone domain, the module that deleted a subscriber record, would also delete the exchange mailbox. But with our ''real'' windows domain, the administrators still review and delete mailboxes by hand. I expect once the mailboxes from last springs students hits the ''delete'' queue, they may reconsider this. For now, we delete them as subscribers (which frees up the Unity license) and they are marked as disabled in Exchange.

One of the more annoying problems, is dealing with full mailboxes. With the stand alone domain, we had a rule that would permanently delete messages that had been ''deleted'' by the user. Mailboxes would still occasionally fill up, but that didn't really cause much of fuss. Unfortunately, we have some users of the exchange service, who have decided that the trash can, is just another folder and that it is a good place for long term storage (I would hate to see their office). We have missed our chance to put the same rule we had on the stand alone system, trash gets emptied automatically!

Futures

Although we are now positioned to provide unified voice messaging to anyone on campus, there are some issues to be worked out. We have the basic problem that while anyone with a telephone (and in some cases, that may not be a requirement) can have a voicemail box, the requirements to get an exchange mailbox are much stricter. There are also some additional features we would like to offer, and some other minor problems that we would like to fix.

[15] Which is a nice way of saying ''wrong.'' But we didn't know any better at the time.

Partially Unified Environment

Unlike many Unity installations, we will be operating in a mixed environment where many people are not unified. This can result in some potential issues for our non unified users. Although we have attempted to hide voicemail only users in active directory, and we also tag their names with the word "voicemail," these addresses can still "leak" out. The simplest way would be when a unified user receives voicemail from a non unified, but on campus, user. The voicemail will be from a recognizable name (i.e., "David Hudson (Voicemail)." It is a trivial matter in outlook to then add that address to your address book, and then you can easily access what is supposed to be a hidden name. Ok, so what is the problem here? Say some wacko (perhaps with tenure) leaves me voicemail saying that he wants some specific hostname changed. As a unified user, I forward this, after typing some comments like "Ignore this idiot – let me know if he contacts you," but I accidentally get Dave's voicemail entry. Dave then listens to his voicemail, is told by Unity that I forwarded a voicemail to him, and plays him the original message, But since there is a voice message, it does NOT read him my comments. If I had just sent Dave a regular email to his voicemail address, he would have been told he has an email (which might confuse him), and it would then read it to him, but in this case, since there was voice, only that would be played.

There are a few other oddball cases like this, but we were concerned enough that we have delayed widespread deployment until we can resolve this. Work is underway to add some hooks in the Exchange server that will detect these cases and generate a bounce to the sender. Once that is done, I expect that we will resume the more general deployment.

Student Voicemail Becomes Opt-In

One of the things we had overlooked in this project, was anti-virus scanning. In the original stand alone deployment, we did not need anti virus scanning, as the exchange server was pretty well isolated from the rest of the world. However, in the unified environment, we need the scanning, and you have to scan all mailboxes. This is licensed on a per mailbox basis, so there was a jump in the licensing cost. Cisco is also changing their licensing, and the per mailbox cost is going to be going up.

Historically, we had provided voicemail service to all on campus students. Not only was the old Octel system paid for, the vendor had gone out of business, so even if there had been per mailbox licensing costs, there was no one to pay them to. It was easier to create mailboxes for everyone, and ignore the idle ones. Now that model has changed, and there is significant costs for mailboxes. We don't mind paying for active ones, but paying for idle ones is a bit annoying. Many of our students come to campus now with their own cell phones. One of the things we have noticed, is a large drop in long distance revenue from students – they are no longer using their dorm room phones to make long distance calls. What is more, they don't seem to be using them all that much to receive calls – at least many are not bothering to set up their voicemail boxes. We did some analysis of usage; one of the fields saved by Unity is the last time a mailbox was opened via the phone. For the spring semester, of the 2500 student voicemail boxes we had on line, only 500 actually called it! We had enabled email notification for students, and all mailboxes had a recorded name (done automatically), but it was pretty clear that we were wasting a lot of licenses.

After some discussions with the CIO, Residence Life and the Office of the First Year Experience, we decided to make student voicemail an opt in service. The web tool they use to pick up their voicemail PIN, will let them request/create a voicemail box if they so chose. After about three weeks since the dorms opened, we have had 370 out of a possible 2925 students request a voice mailbox for their dorm room.

Department Administrator Tools

We have identified departmental administrators for each department. These people can update directory information for people in their departments, as well as issue short term VPN/Wireless access accounts and manage guests. We will be adding a tool to help these people administer the voice mailboxes owned by their department or people in their department. This will likely be rolled out when a similar tool to administer VOIP equipment is released.

User Call Handler Options

One feature we hope to release to users once we understand the billing issues, is the ability to maintain your own caller input selections. For instance, when you listen to my message, you are given the option to press six, and your call will be transferred to my cell phone. This allows people to contact me via my cell phone, yet I don't need to give out my cell phone number.

There are a number of other advanced features in the Unity systems that we may be exploring as well.

References and Availability

The bulk of this project is written in PL/SQL. The core package is the Unity_Maint package (see Figure 2) that provides access the to Oracle tables that store our version of the configuration, and communicates with the Unity server via the Meta_Change_Queue package. This package is basically site independent – none of our business rules are applied here, just the basic communication and configuration. The bulk of the site specific operations, and those requiring work in more than one table (such as creating ECPs) is in the Unity_Functions package. Later efforts to work in expiring mailboxes was put in the Unity_Expire package. This division was mostly to accommodate a lot of changes in expiration processing without impacting tools using the Unity_Functions package. The primary

administrative tool is provided via the Web_Unity_Admin package. The student voicemail tool predated this project by several years and is in the Web_Voicemail_Admin package which in turn used the Tcom_Voicemail_Maint package. These existing packages were joined to the new project via the Unity_Dorm_Functions package. Some of this structure is the result of when things were deployed and several packages could be merged.

The Unity Queue Runner program took commands from the Meta_Change_Queue, translate them via the Unity_Record package and then processes them appropriately. Although in our original Windows 2000 integration process we were maintaining Active Directory directly, in this case, we made the changes in the Unity server (which has it's own SQLServer database), and the Unity application would make the changes in Active Directory. The Unity application would also create Exchange mailboxes as needed (Deleting mailboxes has been disabled for now).

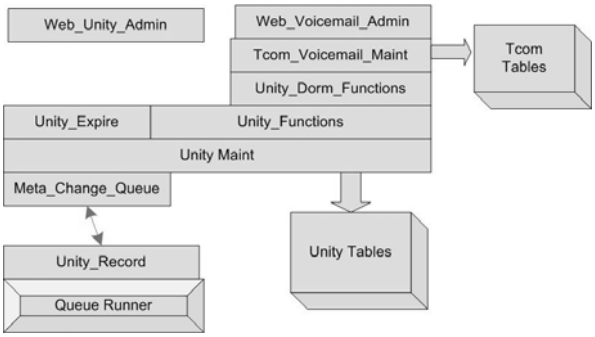

Figure 2: Code Structure.

Most of the PL/SQL source code for the Simon system as well as the full table and view descriptions are available via the web at http://www.rpi.edu/campus/rpi/simon/misc/Tables/simon.Index.html and http://www.rpi.edu/campus/rpi/simon/misc/Tables/SIS-index.html . Some of the modules that interface with the Cisco Call Manager are currently under contract with another vendor, and we are not making them publicly available until the details of the contract are worked out. The migration tools were changing during the course of the migration, what we used at the start is different from tool we used at the end of the project. These tools were not intended for extended use, although we will keep the source code around in case we migrate again. There are also a number of scripts that were put together as needed.

If you ask nicely, I will try to answer questions and might be able to dig out some of the C and JAVA code that makes up other parts of the system.

Acknowledgments

I would like to thank Rudi van Drunen for his shepherding of this paper with me. I also want to thank Rob Kolstad for his excellent (as usual) job of typesetting this paper. I also want to thank the other members of the Unified Messaging Team here at Rensselaer for their contributions to the project, as well as my girlfriend Joyce who encouraged me throughout the entire process.

Author Biography

Jon Finke graduated from Rensselaer in 1983 with a BS-ECSE. After stints doing communications programming for PCs and later general networking development on the mainframe, he then inherited the Simon project, which has been his primary focus for the past 15 years. He is currently a Senior Systems Programmer in the Communication and Middleware Technology department at Rensselaer, where he is developing identity management systems and integrating Simon with the rest of the Institute information systems. In addition to the Simon project, Jon is also involved with the support of the Telecommunications billing system, and providing data and interfaces for Unity Voice Messaging and CISCO VOIP deployment projects at Rensselaer. He hopes to someday find out what Middleware is. When not playing with computers, you can often find him merging a pair of adjacent row houses into one, or inventing new methods of double entry accounting as treasurer for Habitat for Humanity of Rensselaer County. Reach him via USMail at RPI; VCC 319; 110 8th St; Troy, NY 12180-3590. Reach him electronically at finkej@rpi.edu. Find out more via http://www.rpi.edu/~finkej .

Bibliography

[1] Chapman, Brent, "Incident command for IT: What we can learn from the fire department," *The 19th Large Installation Systems Administration Conference (LISA 2005)*, December, 2005.

[2] Finke, Jon, "Institute white pages as a system administration problem," *The Tenth Systems Administration Conference (LISA 96) Proceedings*, pp. 233-240, USENIX, October, 1996.

[3] Finke, Jon, "Embracing and extending Windows 2000," *The Sixteenth Systems Administration Conference (LISA 2002)*, USENIX, November, 2002.

[4] Finke, Jon, "Generating configuration files: The directors cut," *The Seventeenth Systems Administration Conference (LISA 2003)*, USENIX, October, 2003.

[5] Finke, Jon, "Meta change queue: Tracking changes to people, places and things," *The Eighteenth Large Installation Systems Administration Conference (LISA 2004)*, pp. 231-239, USENIX, November, 2004.

[6] Finke, Jon, "When worlds collide: The two-sided sword of technology integration," *;Login: The USENIX Magazine*, Vol. 30, Num. 3, pp. 6-7, June, 2005.

[7] Microsoft, *Visual c#.net,* 2001, http://msdn.microsoft.com/vcsharp .