Winter 2006

PackoutApp

Scott Burau
Regis University

Regis University
School for Professional Studies Graduate Programs
Final Project/Thesis

Disclaimer

PackoutApp

Scott Burau

Regis University

Master of Science in Computer Information Technology

Project Paper Revision/Change History Tracking

Date	Comments
February 20, 2006	Met with Dr. Donald Archer to discuss Professional Project. Changed subject topic to PackoutApp java GUI.
February 25, 2006	Chapters 4-7
March 4, 2006	List Of Figures
March 4-17, 2006	Chapter 8, and overall updates.
March 18, 2006	Complete draft with proof read.
March 20, 2006	Integrated feedback from Week 1 draft.
April 2, 2006	Integrated feedback.
April 2- April 18 2006	Integrated feedback and multiple proof readings
April 18, 2006	Final Draft

Abstract

Currently there is a production line called "Packout" that is used to package products into boxes. The current application used for scanning the products serial numbers is not very user-friendly. The focus of this project concerns the development of a user-friendly Packout monitoring application in Java. The monitoring application provides the users with a way to troubleshoot the Packout application and increase the production of the lines by decreasing the downtime of the Packout lines. The basis for this paper will be to describe the concept of adding messaging to a legacy application, and the creation of a GUI that can interpret and display those messages as a means to "open a window" into another application.

Table of Contents

List of Figures / Exhibits / Addenda

Chapter One – Introduction

Statement of the problem

Currently there is a production line called "Packout" that is used to package products into boxes. The Packout application is a program written in the C programming language that runs on a UNIX server. The only feedback that the users get from this application is a log file that was originally used by the developers for debugging purposes. The log file messages scroll across a telnet window and are lost to the users after they scroll off the screen. Often the log messages scroll by faster than the users can read them. With the limited feedback, the users have a difficult time resolving issues that arise with the Packout application. The users put in a request to the IT department to enhance the Packout application to provide them better feedback. The feedback will allow the users to troubleshoot issues that occur with the Packout application. This project will address the tool that was created to monitor the Packout lines and display essential information, which can help the users resolve issues with the Packout application. Furthermore, it will discuss the concept of adding a messaging feature to an existing application to allow the sending of messages that will be interpreted by another GUI application. The information will allow the users to quickly resolve issues with the Packout application, which will in turn increase productivity by minimizing the downtime.

The Packout lines are similar to assembly lines. Products are packaged together with supporting materials such as owner's manuals, instructions, remote controls, etc. A large pallet of products is brought to the beginning of the line to be packaged into

their individual boxes with the supporting materials. The product has a serial number that is scanned at the beginning of the line to make sure the products serial number exists in a database and that the product should be "packed out" under the specified part number. A label is generated that contains the serial number of the product, and that label is placed on the box the product is being packaged in. The product is then passed down the conveyor line where the support materials are placed into the packing box, and the box is taped shut. The packaged product is scanned again as it is placed onto another pallet of "packed out" products that are ready to be distributed to customers. This second scan is going to associate the product with a pallet. Once the pallet is full, a number of labels are printed containing a list of all of the products serial numbers on the pallet, and a pallet-id.

The Packout application is used in different geographical locations. Each of the geographical locations typically runs three to six Packout lines. For the purpose of this project Chicago, Los Angeles, and New York will all be different geographical locations. Another geographical location will be added in the next year. For each physical Packout assembly line there is an instance of the Packout application running on the UNIX server. If a new line is created in an existing location or a new geographical location the monitoring application should be able to monitor the new lines without any code changes.

The original Packout enhancement request contained the following requests relevant to this project:

- Add a user friendly log that the line workers will be able to use to troubleshoot the application.

- Create a monitoring application that will be able to display information about all of the lines defined whether they are running or not.

The original monitoring application that was proposed via a telnet window was deemed too limiting to be able to effectively provide information that would help the users troubleshoot the problems that arise with the Packout application. A new idea was proposed to write a monitoring application in Java. This proposed application would be a thick client GUI that would be much more user-friendly than a telnet menu system. The Java programming language was chosen due to its extensive libraries that provide a means to create a GUI, enable socket communication, interpret XML, and run on multiple platforms of UNIX and Windows needed for this project.

This paper will describe how by adding messaging to a legacy application and creating a Java GUI to interpret those messages an existing application can become more user-firendly. The paper will embellish how this messaging approach will fulfill the two criteria: to add a user-friendly log, and create a monitoring application that can display the "health and status" of each Packout line at each location. The focus of the paper will address why this approach was taken and how this approach can be applied to other legacy applications.

Thesis statement

This paper will describe the program solution that was developed and implemented to effectively enhance the trouble shooting aspects of a legacy system in order to fulfill specified user requirements that were provided to the programming staff. This was accomplished by add messaging to an existing legacy application and creating a GUI to monitor the legacy application in real time in such a way as to

provide enough information in the GUI to allow for the users to effectively troubleshoot problems while the application is being monitored.

How the project began

The project began as an enhancement request to allow the Packout application users the ability to be able to troubleshoot issues that arise with the Packout application. The suggested monitoring tool via a telnet window described in the original requirements proved not to be a viable solution. A telnet window could only display a limited number of characters. A GUI on the other hand can display much more information on a screen, and is not limited to characters. By creating a Java GUI, the application would have the capacity to display the required information on a screen. The Java language offers an extensive library that would allow more user-friendly features in the monitoring application than a telnet window could offer.

The original request stated the desire to monitor when and if a specific subnet was available. This request wanted an indication if a connection to one of the scanners or printers was not available due to a network issue, it would be advantageous if the users knew for certain if the devices were able to connect to the Packout application, instead of just trying to indicate that a subnet was available. This would be possible if the Packout application could indicate or send a message whether or not a device connection was established or not. Thus, this was the one requirement that sparked the idea of adding messaging to the Packout application. The problem then was how does the message get to the Java GUI to display the information to the user?

Organization of thesis

The organization of the following chapters will include the research into the typical issues the line workers experience with the Packout application. The methodology will progress through the documentation of problems, and resources assigned to the project. The project history will be described. The development effort will be discussed in three separate chapters with each chapter discussing a specific programming unit and the architecture that ties them together. Please refer to Appendix A for an architecture overview. Finally a conclusive chapter will report findings and analysis of the project.

Barriers and/or limitations

This application is currently used in an existing company. Some of the names and information will be changed to protect specific information relative to the company and any security issues that may arise. The code has not been included because the concept of what the code does and why is more relevant than how the code was actually written. The discussions of the architecture, does not only explain how the code works, but also why the architecture was designed in its final form.

Scope of the project

This project will focus on the design, requirements, implementation, and testing of the Packout applications monitoring tool. The discussion of the changes that are made to the existing Packout application will be limited to a high level overview and will not cover the specifics of the changes to the existing Packout application.

Chapter Two - Review of Literature/Research

Definition of terms

There are a number of terms that will be introduced which will be used throughout the rest of this document.

- GUI - Graphical User Interface
- packsrvr - This is the existing Packout application written in the C programming language.
- PackoutAppDaemon - This is the Java program that essentially runs as a daemon on the UNIX server to allow communication between the packsrvr program and the PackoutApp.
- PackoutApp - This is the Java monitoring GUI that will display information to the user about the packsrvr programs that are running for each Packout line.
- Thick Client - An application that is installed locally on the users PC that typically includes application specific logic.
- Thin Client - A local application that allows access to a remote application that is typically installed on a server. The local application typically does not include business logic. A typical thin client is an Internet browser.

Literature and research that is specific/relevant to the project

The research that was done on this project was specific to two distinct areas. The first area of research was devoted to information which would be helpful to the intended users of the PackoutApp to allow them to troubleshoot the existing Packout

application. The second area of research was devoted to the technical aspects of how to implement the new PackoutApp monitoring application.

The original enhancement request came in the form of a Business Requirements Document. Included in the requirements document were samples of the user-friendly log messages that the users requested. The document also included some of the issues with the Packout application that the users need to troubleshoot, and the information that would be useful for them to perform the troubleshooting themselves. The requirements document also contained a mock up of a monitoring system developed by UNIX shell scripting menus in a telnet window.

The most common issues that arise on the Packout application are the availability of the information about the device connections. Each line has two scanners and two printers. When the Packout program is started, it tries to establish connections to these four devices. The users have only an unfriendly log file to try to help them determine if the connections were successful. This log file is displayed in a telnet window and often scrolls past the viewable area of the screen too fast for the users to read. The most significant features that the users requested was to be able to determine device connections, and have better log messages that are more meaningful to the users. For example a Chicago line 1 log message was previously "Adding the device chicago.l1s2.sburau.homelinux.org to the device list" was changed to "Successfully connected scanner2".

A meeting was scheduled with the developers and the managers representing the users. The discussion was focused on the typical problems that the users face, and the developers were able to propose these possible solutions. A mock up of the

PackoutApp was developed as a sample of how the PackoutApp would look and feel to the user and the type of information it would display. The users provided excellent feedback and suggestions that allowed the PackoutApp to be designed to provide them with the information they could effectively use to troubleshoot problems.

The design of the PackoutApp monitoring application was the next most significant area of research for this project. How the PackoutApp was going to get the information from the Packout application, and how the PackoutApp was going to display the information in a user-friendly manner. The idea was to have the Packout application send XML messages across a socket connection to a Java daemon on the UNIX server, and have that daemon send the XML messages to the PackoutApp clients that were connected. The XML language would allow the Packout application that was written in the C programming language to communicate with the PackoutApp written in the Java programming language. A few test programs were written to research how to establish socket connections between C programs and Java programs, and send an XML message across the connection.

Summary of what is known and unknown about the project topic

The developers working on this project have all supported the Packout application in the production environment. This familiarity with the Packout application allowed the developers to be able to provide relevant ideas to be incorporated into the project. The most significant unknown was exactly how to implement each of the different requirements.

The contribution this project will make to the field

The most significant part of this project were the ideas developed to have an application send information about its status to a central application, and allow that information to be distributed to connected clients and instantly displayed and utilized. The PackoutApp is a very specific application. It was entirely designed to monitor the Packout application; however the basic concepts of this architecture could be expanded to allow a more generic monitoring tool. Standards for the XML messages, and message transmissions could allow a generic monitoring tool to be used to monitor any number of applications.

The existing Packout application has been around for about eight years. It is written in the C programming language and also uses Pro*C to connect to an Oracle database. The idea to rewrite the Packout application in Java is something that has been considered for a number of years. The motivation for rewriting it is because it is difficult to find people that know both C and Pro*C and are able and willing to maintain the existing code. Since the application is a mission critical production application rewriting the whole thing is not a small task. That is where the concept of adding a socket connection that sends XML messages to a user-friendly application brings the user much closer to being able to understand the application on a more visual level. With a GUI to look at, the users are able to physically "see" the Packout application through the information being displayed. It brings the application to a more personal level instead of some process running somewhere on a server.

Chapter Three – Methodology

Research methods to be used in investigating the problem

The developers that worked on this project all supported the Packout application from development of new enhancements, to the production issues that arose. This allowed an existing knowledge of issues that typically happen in the production environment with the Packout application to be addressed concurrently. In addition to the work experience of the developers, the requirements outlined a number of specific details the users wanted to address. The developers also met with a number of the users to discuss issues that were frequently mentioned in production and what type of information would help them to resolve issues on their own.

Life-cycle models

The software development life cycle for this project was based on processes that the company has established as their SDLC. The basic underlying SDLC has a foundation of requirements gathering, design, implementation, and testing. The actual process follows a linear flow with occasional iterations when required. The overview of the SDLC is as follows:

- Request for change initiated by the users
- Requirements gathering
- Review of requirements
- Requirements approved and signoff by users
- Writing of technical requirements and design documents
- Development/Implementation
- Unit testing
- Code review
- Formulation of software change request to move code to test environment
- Software acceptance testing

- User acceptance testing
- Change control board for approval to move to production environment
- Production verification of correct code migration and functionality

At any point in the process the project can go back to any previous step when appropriate.

Resource requirements

The development of a new Java GUI and the modifications to the existing Packout application was a significant effort that required three hundred and sixty hours. The levels of effort were discussed among a few developers, and the development manager. It was determined to split the effort into three separate areas and assign each of the three developers a specific area. The initial time estimate was for three developers to each spend one hundred twenty hours on their assigned areas.

Outcomes

The developers, development management, and users were all impressed with how the project has evolved and turned out. The project is currently in the test environment. The success of the application in a production environment has yet to be determined. After the application has passed the test environment it will be moved into the production environment. The date of the move to production is dependant upon the applications approval by the test group and the users acceptance testing.

Summary

The overall process of how this project has taken shape has been very positive. With the experience of the developers, the input and feedback from the users, and the overall design of the project, the current progress of this project has been very

successful to this point. With thorough research, design, and team work the project seems to have sparked a new innovative thought process relating to new development efforts. The idea to add messaging to a legacy application and display the messages in a GUI has introduced a new paradigm to the developers previously unseen. The concept of messaging between applications is not a new concept in itself. It is the underlying basis for such technologies as Middleware, Service Oriented Architectures, and even EDI, which has been around for a while. However adding messaging to an existing application to provide feedback to users is not something the developers have commonly encountered or previously provided to the end user.

Chapter Four – Project History

How the project began

The project began as an enhancement request to develop a tool to monitor and provide user-friendly feedback from the Packout application. The enhancement request included some of the information that the users were requesting to be on the current monitoring tool. The initial requirements document had a mock up of the suggested design of a monitoring tool via a telnet window. Some of the features that the users wanted to monitor could not be accomplished using the suggested telnet window approach. It was hypothesized that it would be much easier to monitor from a different approach using Java to create a thick client GUI.

A mock up of the PackoutApp GUI was made to present to management for their approval to move forward with the new development approach. The prototype was then expanded slightly upon before presenting the idea to the users to get their approval and feedback. This gave the users a chance to suggest anything they would like to add or change.

The original requirements, the users suggestions, and the new development approach were then documented in a Technical Requirements Document (TRD). The Level Of Effort (LOE) was estimated with the plan to use three developers to work on the project. The developers were assigned, and worked together to discuss issues that came up as the development progressed.

How the project was managed

The project was originally designed to use three different developers into three different development efforts that addressed:

1) The changes to the existing Packout application code called packsrvr.

2) The development of the daemon called PackoutAppDaemon.

3) The development of the PackoutApp.

The implementation and architectural design was left to the developers to design. The design of a thick client Java GUI was new to the development group, and the implementation of the project was initially planned at a high level. Some of the implementation details would have to be further expanded on once the development was underway.

An initial timeline was determined, and the development was given the ok to start. Not all of the developers were available to start at the same time. This caused some of the initial development efforts assigned to each developer to fall outside of the originally assigned development areas. This worked out well; it seemed to help initiate a team effort. Once all of the developers were ready to start development on the project, the team worked very well together discussing the design of the project that would interact with the other people's development efforts.

Significant events/milestones in the project

The PackoutApp was developed to create some of the basic screens of the GUI. Once progress of the PackoutApp was started, a skeleton PackoutAppDaemon was created to establish the connections between the three programming units. The PackoutAppDaemon used many of the same java classes that the PackoutApp was

able to use. It was a significant step to get the initial architecture (see Appendix A for architecture diagram) working with the packsrvr connecting to the PackoutAppDaemon, and the PackoutApp clients connecting to the PackoutAppDaemon. By this point the PackoutAppDaemon proved to be a much easier development effort than initially thought. The developers were now ready to start on expanding the different development efforts and concentrate on the functionality of the messaging and GUI presentations.

With the initial architecture working, and the three separate applications able to communicate to each other the XML message structures were initially developed. This gave everyone a basis for how the information was going to be sent from the Packout application, to the PackoutAppDaemon, and then propagated down to each PackoutApp client connected. The format of the XML messages was a topic that evolved over time. It was not easy to come up with messages that would be generic enough to be able to encapsulate all of the information that would be sent in XML. It was decided that to have seven different types of XML messages that all have the same basic XML structure. These include the following: scanner1, scanner2, printer1, printer2, logs, uflogs, and "healthandstatus".

To simplify testing the PackoutApp and PackoutAppDaemon two testing applications were developed that would be used to send test data in the form of the XML messages to the PackoutAppDaemon through the port that the Packout application would use to send information to the PackoutAppDaemon. This enabled the development of the PackoutApp and PackoutAppDaemon to be done without the dependency of the Packout application development changes.

The initial development started at the beginning of November. The initial working prototype was demonstrated to the user group around the end of January. A few remaining tasks would need to be ironed out, and a couple new additions to the applications were added. By the end of January development on the project was complete.

Changes to the project plan

The original project plan was to split up the three separate code pieces to each developer. With the PackoutAppDaemon turning out to be an easier task than originally thought, it allowed the PackoutApp Java development to be done in a parallel effort with occasional code merges. This turned out to be a very effective and creative occurrence as it allowed two developers to work together and get ideas as well as help from each other as needed.

Chapter Five - Packout Application (packsrvr)

The original Packout application has been in production for approximately eight years. The application connects to two scanners, two printers, and a database. The Packout application does a number of things that are proprietary to the company and will not be discussed in this document. All functionality of the application will be discussed at a high level to avoid reference to proprietary information.

The packsrvr is the program binary that exists on a UNIX server. An instance of the program is started for each Packout line that is running. The type of connection to the scanners depends on the type of scanner. The scanners are configured to connect to the UNIX server on a specified port if the scanner is considered a client. If the scanner is considered a server then the packsrvr connects to the scanner. The printers are typically setup as server connections, and the packsrvr program connects to the two printers.

For the purpose of this document the changes to the packsrvr program consist of the following high-level changes:

1) Add a port to each occurrence of the packsrvr program running. This port will be used to send XML messages to the PackoutAppDaemon.

2) The packsrvr program will establish the connection to the PackoutAppDaemon. If the PackoutAppDaemon is not available, it continuously tries to connect every number of seconds as determined in a configuration file setting. If a connection cannot be established with the PackoutAppDaemon then the packsrvr program must continue to work as before this enhancement was added.

3) Add user-friendly log messages that the line workers can better understand. This will help them trouble shoot issues as they arise.

The reasons that the packsrvr is to establish the connection to the PackoutAppDaemon instead of the PackoutAppDaemon connecting to the packsrvr was basically a coin toss. It would allow the PackoutAppDaemon to act as a true daemon in that it just listens on a port for connections. Since the Packout application is a mission critical production application having the packsrvr establish the connections would keep the "control" in the packsrvr program.

XML was determined to be the most flexible means to send messages across the socket connection. The packsrvr program builds the messages it wants to send as a long string of characters. It then sends the XML messages to the PackoutAppDaemon. By sending messages from the packsrvr it really creates a "window" into the Packout application. Enough messages need to be sent from the packsrvr program through the PackoutAppDaemon to the PackoutApp that it can display effective feedback and provide information to the user.

Adding the messaging to the packsrvr binary written in the C programming language provided a means of communication between the "legacy" Packout application, and a new Java application. The idea of adding a type of messaging from an older application to a new application has greatly increased the "user-friendliness" of a legacy application.

Chapter Six - PackoutAppDaemon

The PackoutAppDaemon is a program written in Java that will act as a daemon and run on the UNIX server that the packsrvr runs on. The daemon will listen for connections on a single port from the packsrvr programs running for each line. The PackoutAppDaemon will also listen for connections from the PackoutApp clients on two ports. The PackoutAppDaemon will therefore have three ports open for connections. Each port can support multiple connections.

The PackoutAppDaemon will take messages from each packsrvr connection on port 6000 for example. These messages will then be sent to each PackoutApp client connected on port 6001. The ports are arbitrary and are defined in a configuration file setting. This is the main pipeline for messages to go from the packsrvr binary to the PackoutApp clients.

The PackoutAppDaemon also generates "healthandstatus" messages that are used by the PackoutApp to determine if a packsrvr occurrence is running for a given Packout line. The PackoutAppDaemon will use the ps command in UNIX to determine if a packsrvr occurrence is running for a given line. The packsrvr is passed a configuration file as its parameter. This configuration file for a packsrvr instance running shows up when doing a ps command. The configuration file is read for the PACKOUTAPP_LOCATION and PACKOUTAPP_LINE values. These values are used to create the "healthandstatus" XML message, which is then sent to each connected PackoutApp client though port 6001. Configuration files that do not have an associated packsrvr process running identify Packout lines that are not running.

Each configuration file for all lines is loaded into a PackoutUtilPropertyHolder. Each PackoutUtilPropertyHolder is then added to a single PackoutUtilPropertyManager object, which is sent to the connected PackoutApp clients through port 6002. This data is used by the PackoutApp to determine which lines and location are available to monitor. The configuration files for the packsrvr binary are loaded into a table and displayed on the PackoutApp in the Configuration tab (Figure 4).

The need for the PackoutAppDaemon originates from the need to be able to get configuration file information from the UNIX file system to the clients. The PackoutAppDaemon reads and send the configuration files for the packsrvr binary program to the PackoutApp clients connected. The PackoutAppDaemon is also responsible for sending information to the PackoutApp clients for packsrvr process that may or may not be running on the UNIX server. Another reason for the PackoutAppDaemon is that it allows a central connection point for all packsrvr instances running.

The PackoutAppDaemon is a very simple design. It simply relays XML messages from each of the packsrvr binaries running on the UNIX server to the PackoutApp clients connected. It also performs a few file system and operating system functions. It is an essential means to allow the PackoutApp client get information from the UNIX server.

Chapter Seven - PackoutApp

The PackoutApp (Figure 1) application designed to be installed on a PC at the beginning of each Packout line, as well as on anyone's computer interested in monitoring a Packout line. The line workers will then be able to use the PackoutApp client to monitor the line they are working on. All of the other lines are also available for viewing. This will allow the line workers to see if other lines are having similar issues. The PackoutApp application is a monitoring tool, which is "view only", it will not allow the changing of any configurations.

The PackoutApp is a thick client GUI. The Java SDK provides a robust library that allows the development of the features originally requested by the users and more. The PackoutApp application is made up of five distinct sections: header panel (Figure 1), log tab (Figure 2), device tab (Figure 3), configuration tab (Figure 4), and status overview tab (Figure 5).

Header Panel

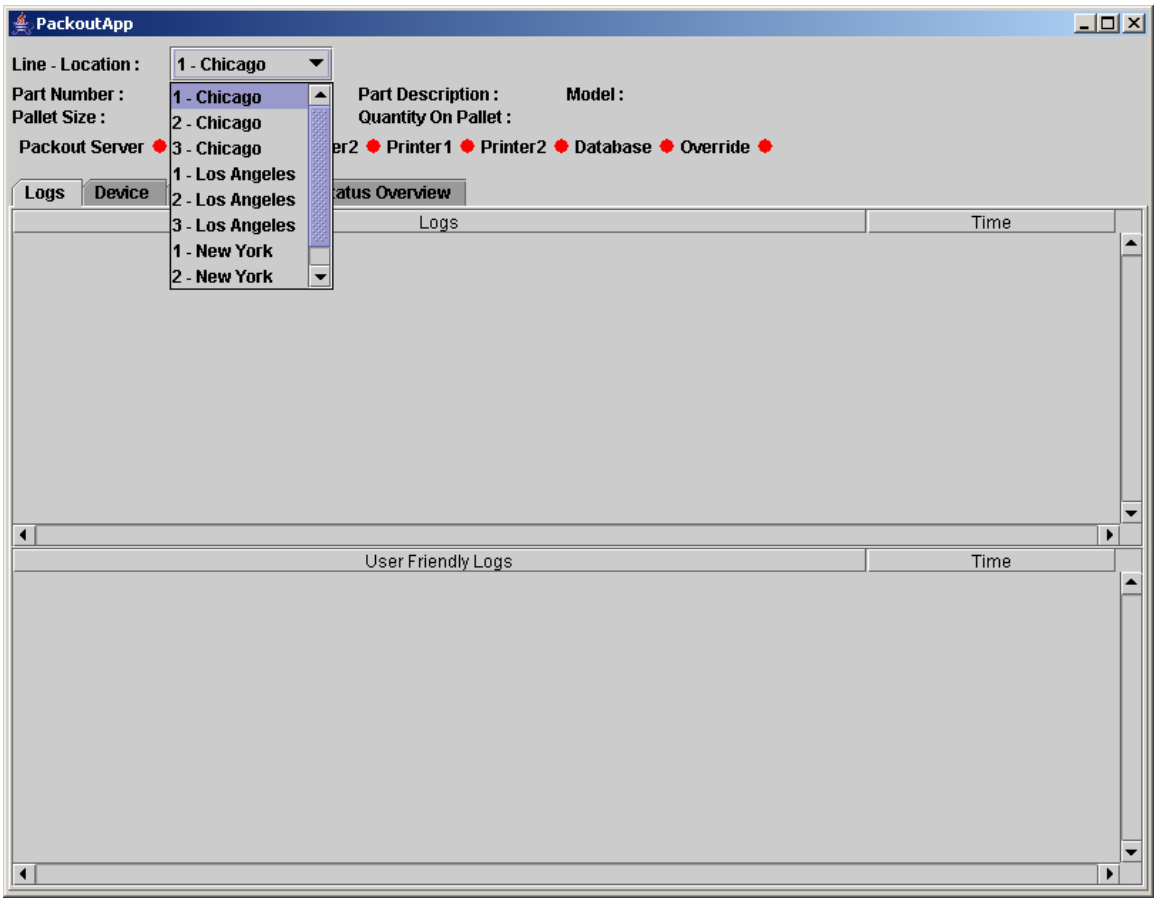


Figure 1. PackoutApp Header Panel

The header panel displays information about a Packout line for a given line and location. A combo box is used to select a line at a listed location. The values in this combo box are values that are included in the packsrvr configuration files. The PackoutAppDaemon sends the information in the packsrvr configuration files down to the PackoutApp in a PackoutUtilPropertyManager object through the object port 6002. Since these values are received from the PackoutAppDaemon, and the PackoutAppDaemon only sends them every number of seconds defined in a property file, the PackoutApp does not open until it receives a PackoutUtilPropertyManager

object from the PackoutAppDaemon. If the PackoutAppDaemon is not running the PackoutApp will not open, and the PackoutApp will display an error message to check to see if the PackoutAppDaemon is running.

The next line of the header panel is information about the product the Packout line is running. This includes the part number of the product getting packaged, the part description, and the model. This is information helpful to the line workers, as well as a manager who may be wondering what a specific line is currently running.

The "Pallet Size" is the number of items for a given part number that should be placed on a single pallet. Once the products are packaged into the boxes they are placed onto a pallet for storage and shipping. This quantity is the number of products specified to go onto each pallet. The "Quantity On Pallet" is the number of products that have been scanned through scanner2 and placed on a pallet.

The last line on the header panel is called the status panel. This is a row of indicator lights that turn red or green to represent processes running or connections. The "Packout Server" indicates if the packsrvr process is running for the location and line. "Scanner1" indicates that the scanner1 at the beginning of the Packout line is connected. "Scanner2" indicates that the second scanner at the end of the line is connected. "Printer1" and "Printer2" are the connections to the printers at the start and end of the line. The "Database" indicates that the packsrvr process has a connection to the database. The "Override" indicator indicates that the packsrvr program is currently running in the "override" mode.

Log Tab

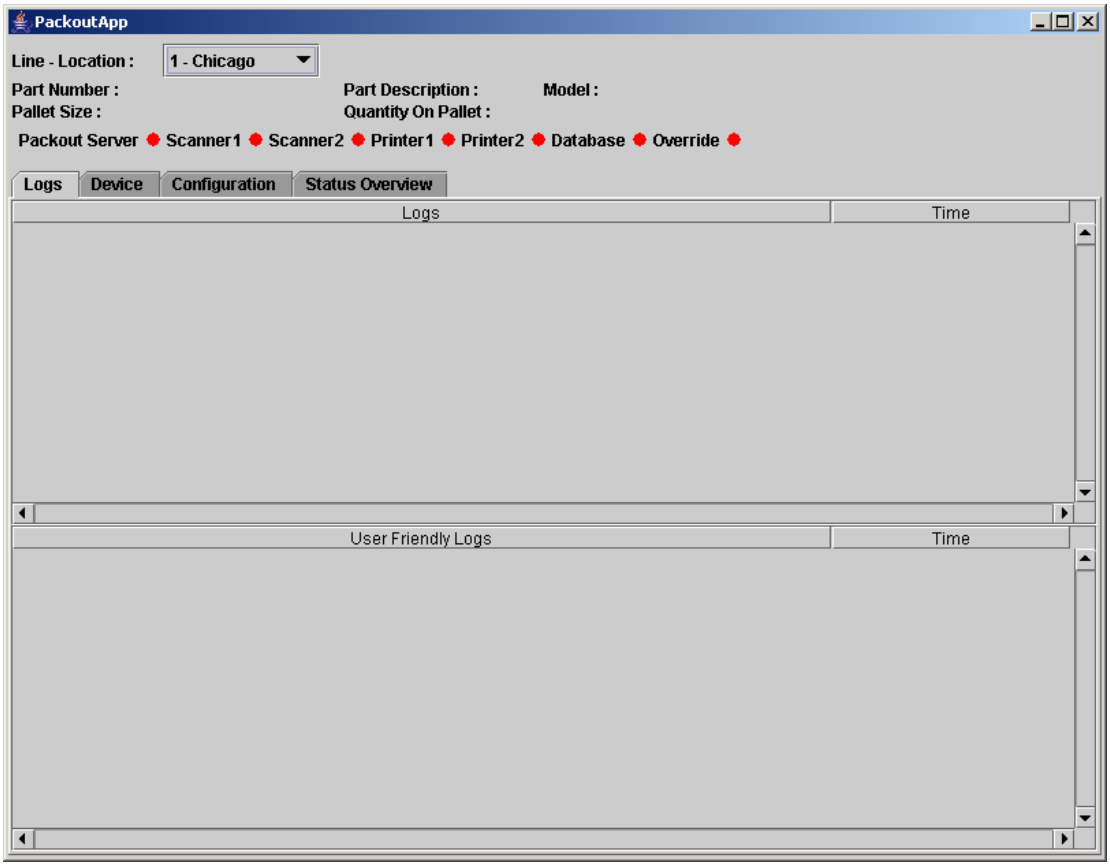


Figure 2. PackoutApp Log Tab

The log tab has two areas. The top section is the "Logs" area; the bottom section is the "User Friendly Logs" area. Both of these areas display messages that originate from the packsrvr program. One of the original requests by the user was to add a user-friendly log that the line workers could better understand. The "Logs" area is the display area for original logs messages the packsrvr program writes to the UNIX server.

Previously the only way the line workers could attempt to debug a problem with the application was to read a log message that was originally used by the developer for development and debugging. These messages were not easy to understand for the

line workers. The original logs were also files on the UNIX server, which the line workers do not have easy access to. For example when the lines are running the log messages are displayed in a telnet session on the PC at the beginning of each line. However, once the messages scrolled off the screen the line workers could no longer see the messages due to the telnet session not having a scroll bar.

This tab alleviates the user problem and permits the user to not only see the original logs, but also the new user-friendly log messages. The addition of the user-friendly log messages should allow the user to better understand what the packsrvr application is doing. These new display areas can also scroll back allowing the user to see a number of lines specified in a property file. The display areas also display the time that the log message was generated on the packsrvr program.

Another unique feature of the "Log" and "User Friendly Logs" is the way it displays the messages. If the message starts with "ERROR" the background of the row is red and the text is white. If the message starts with "WARN" the background of the row is yellow and the text is black. Regular messages will be displayed with a white background and black text. This allows the warning and error messages to immediately stand out.

Device Tab

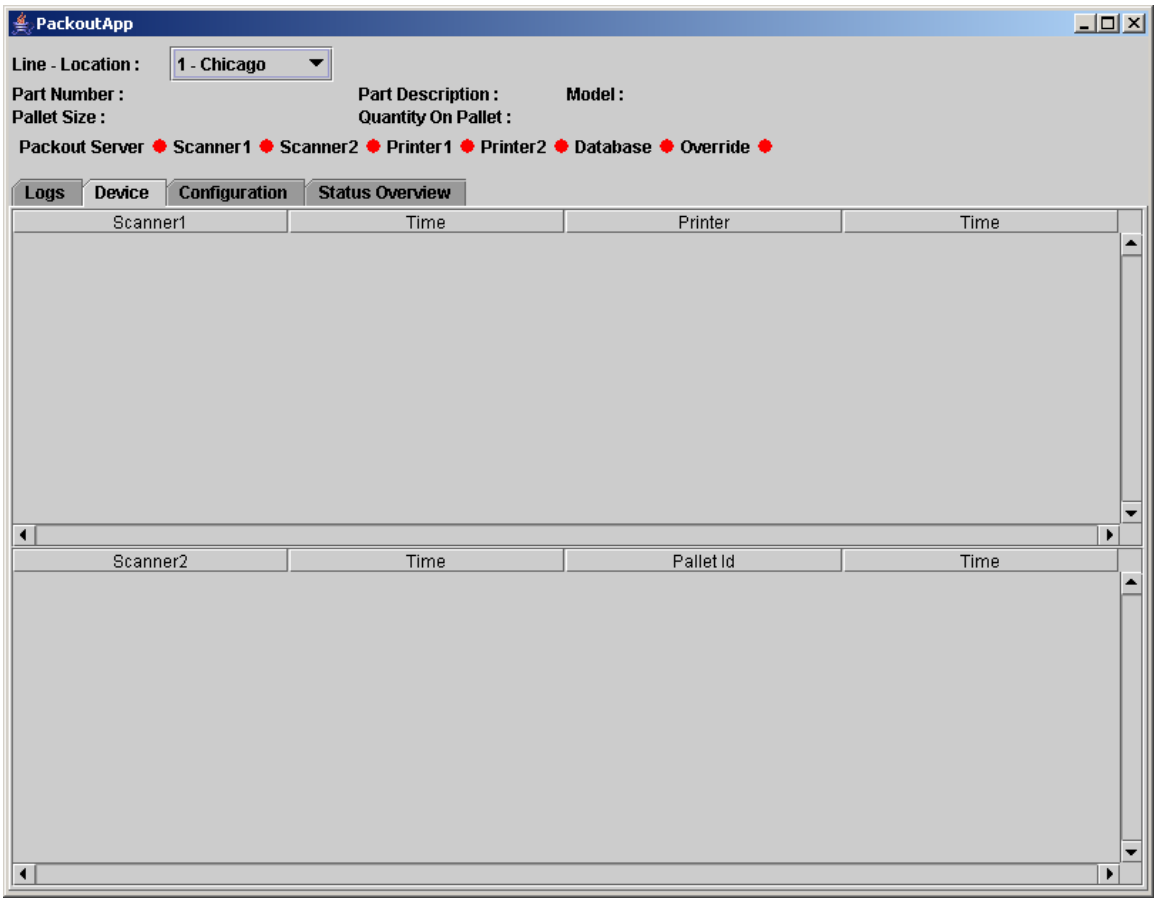


Figure 3. PackoutApp Device Tab

The device tab contains two sections. The top section is for information about the scanner and printer at the beginning of the line. The bottom section is for information about the scanner and printer at the end of the line.

The top section of the device tab has four columns. The "Scanner1" column will display any value that is scanned through the scanner at the beginning of the Packout line. Once the label is printed for that product, the serial number of the product is displayed directly across from the "Scanner1" row where the serial number was scanned. This indicates to the user that the value was scanned and the packsrvr printed a label for the product.

The bottom section of the device tab has four columns. The "Scanner2" column displays the value that is scanned at the end of the line. Once the "Quantity On Pallet" reaches the "Pallet Size" the packsrvr program generates a number of labels for the pallet that now has the full number of products for that pallet. Once the pallet labels are generated the packsrvr sends messages to indicate the pallet-id for each serial number scanned.

Configuration Tab

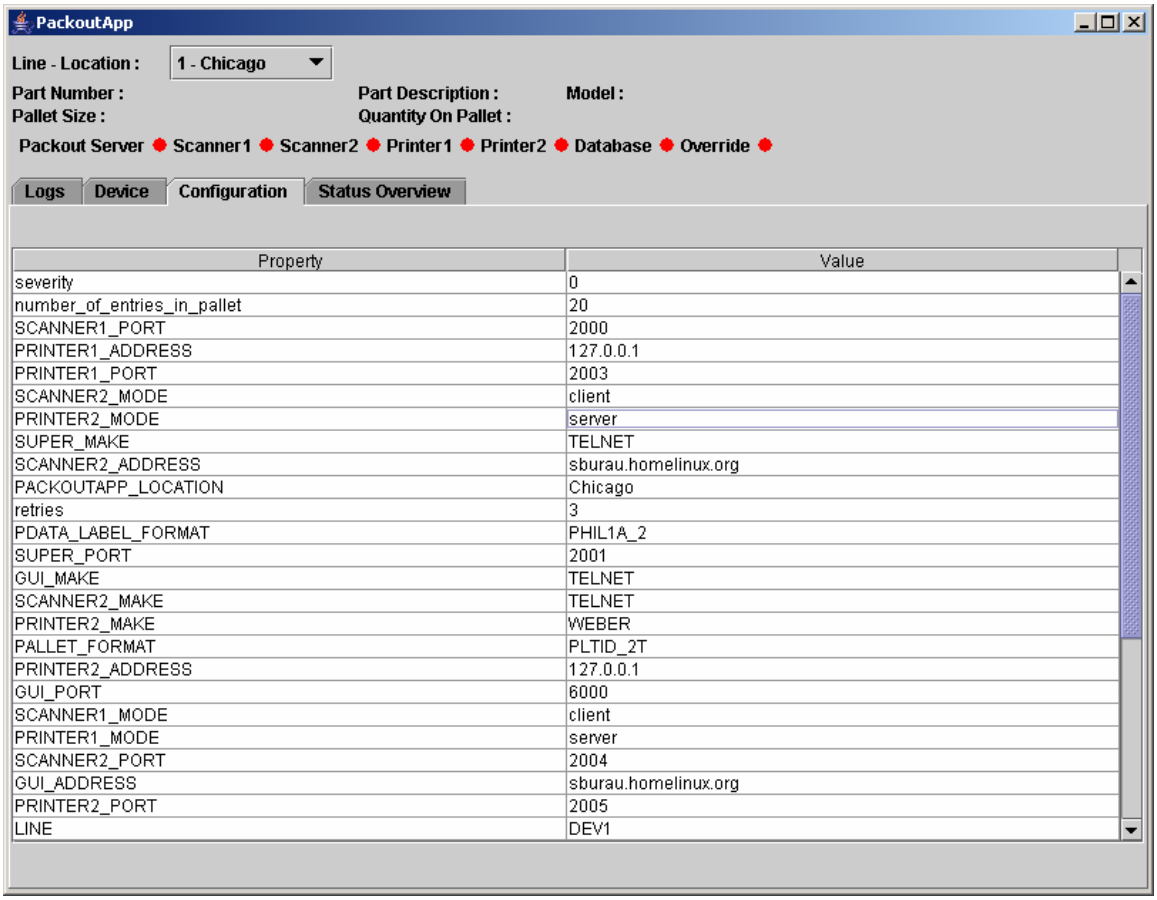


Figure 4. PackoutApp Configuration Tab

The configuration tab allows the user to see all of the values from the packsrvr configuration files on the UNIX server. The users previously had no access to these files. The PackoutAppDaemon reads the configuration files and sends this

information to the PackoutApp every number of seconds defined in a property file. The users can change a few of the setting in the configuration files through some menus in a telnet session. Now the users can see if the values are set correctly which can also help them troubleshoot problems that are encountered on the Packout lines.

Status Overview Tab

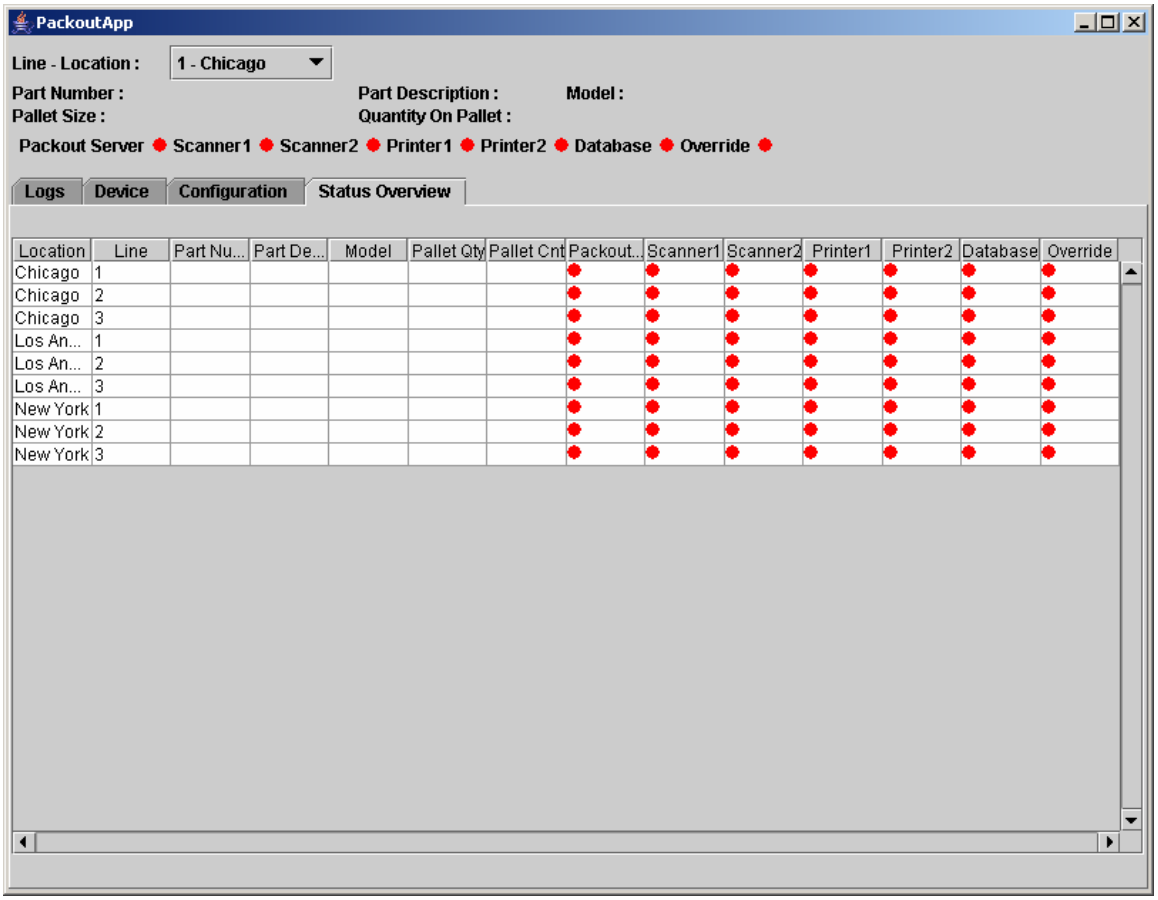


Figure 5. PackoutApp Status Overview Tab

The "Status Overview" tab displays all of the same information that the header panel displays, except it displays all of the information for all of the available locations and lines. This allows the user to see if other lines are having similar issues. This was a feature that the user requested when seeing the initial mock up of the PackoutApp.

Chapter Eight - Findings and Analysis

Summary of the project

The project went very well. The implementation of the design turned out to be a very straightforward process. The users were given a demonstration of the PackoutApp in the warehouse on a Packout line with the application pointing to a development environment. The users were very pleased with how the application turned out, and were given the opportunity to provide feedback and make requests for some minor additions and modifications

The project started as a vague idea about a thick client Java GUI monitoring tool. Many of the coding techniques that were used in this project were new to the developers at the beginning of the project, including Swing, multithreaded Swing, XML SAX parser, sockets, and connection management. The design was well implemented and the object-orientated nature of Java was utilized very skillfully. Many of the objects tried to follow a design pattern as described in Design Patterns: Elements of Reusable Object-Orientated Software.

The developers were allowed to implement the project in whatever way they determined to be the best solution. Working as a group the developers were able to bounce ideas off each other, which allowed for a few different approaches to be analyzed and discussed. This was an excellent learning experience for each of the three developers on this project. Each of the developers learned more about XML, socket connections, and user interfaces than before, and it also was a pleasant change from the typical database development that the group usually does.

Analysis of results

The users received the PackoutApp very well. The demonstrations that the developers performed for the users went well and the user feedback was very positive. The users had few requests for minor additions and changes. The PackoutApp is a read-only monitoring tool; its use is very straightforward. By adding messaging and a GUI to the original Packout application the users now have a visual picture of what the Packout application is doing.

What might have been done differently

The PackoutApp is a thick client. This was done for a number of reasons including the socket connections, multi-threading, lack of an application server, and skill sets of the developers. The PackoutApp was shown to another developer in another group who made a number of suggestions about implementing the PackoutApp as a thin-client in a browser. The suggestions were well received, however the suggested technologies were not in the skill sets of the developers working on the project. If some of these suggestions were brought up at the beginning of the project the PackoutApp may have turned out completely different.

The thick client was chosen because the PackoutApp would need multiple threads to be able to read data from multiple socket connections, and manage its own information. The developers were not aware of how to get a thin client to be able to accept incoming data without the browser sending requests to an application server by a user manually requesting an update. An application server was also not available to the developers to attempt a thin client application.

The one suggestion the other developer made that was the most interesting was to incorporate a Tomcat application server API into the packsrvr binary. This would allow the use of a thin client to take the place of the PackoutApp, and eliminate the need for the PackoutAppDaemon. The packsrvr binary would then be able to use the Tomcat API and become somewhat of an application server in itself. The client could also be sent pages that would request themselves to be updated every so many seconds. With partial page rendering the client could update part of a page instead of rebuilding the entire page. These were all excellent suggestions, however the developers were not that familiar with the suggested ideas to fully agree that it would work.

Discussion of whether or not the project met initial project expectations

The project was able to meet the requirements of the users with a few issues related to determining connections to devices. Depending if the device is setup as a server, or a client, the initial connection is not known until data is received or sent. The reason being is that server devices will listen for data and not establish a stateful connection. Overall the project has been considered a good tool for the users. The deployment into a production environment will be the true test.

Further developments

The PackoutApp has a lot of potential for future development. A number of ideas are already being considered. The users have a number of UNIX shell script menus that provide different functionality these could be included into the PackoutApp.

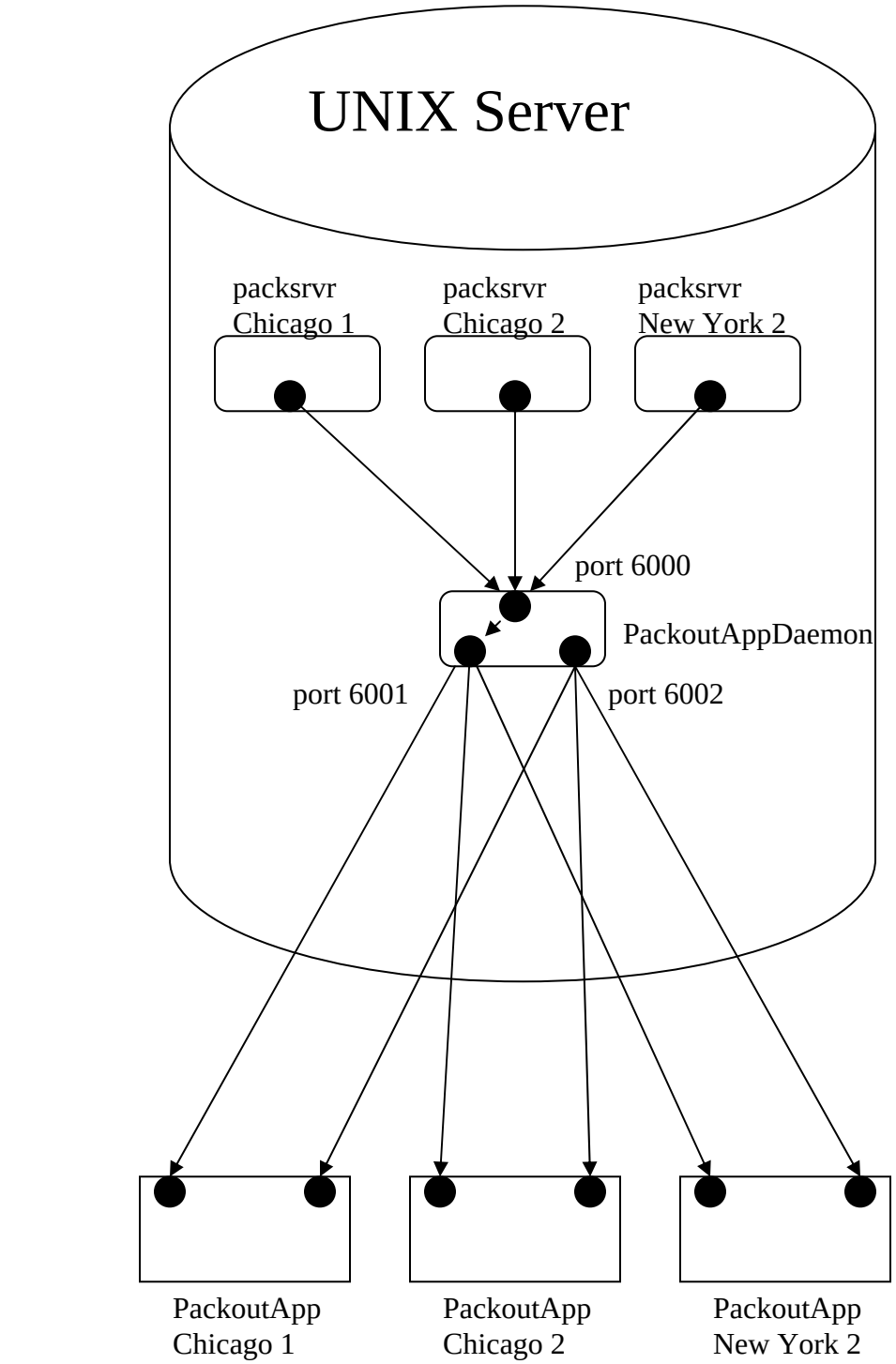
Currently the PackoutApp is read-only, future releases are expected to have the ability to make configuration changes, and even fully replace the packsrvr C program.

Conclusions

The users are excited about the application and the possible future enhancements to the application. Future enhancements to the PackoutApp are being formulated, including incorporating the packsrvr C application into the PackoutApp and eliminating the C code. These future enhancements depend on the success of the PackoutApp over extended use in the production environment, and return on investment that the enhancements may offer.

This was an interesting project for all three developers. The technologies used were challenging and kept the interest of the developers. The concept of adding messaging to an existing application was a new idea to the developers, and a good experience to learn from with respect to concepts and technologies. Adding messaging to a legacy application and putting a GUI in front of the users is an innovative idea that none of the developers have previously encountered.

Appendix A



● port

Bibliography

Chang, Ben, Mark Scardina, Stefan Kirtzov. Oracle9i XML Handbook. New York: Osbone/McGraw-Hill. 2001.

Deitel, H.M., P.J. Deitel. Java: How To Program. New Jersey: Prentice Hall. 2002.

Deitel, H.M., P.J. Deitel, and S.E. Santry. Advanced Java: How To Program. New Jersey: Prentice Hall. 2002.

Gamma, Erich, Richard Helm, Ralph Johnson, and John Vlissides. Design Patterns: Elements of Reusable Object-Orientated Software. Boston: Addison-Wesley, 1995.

Java™ 2 Platform, Standard Edition, v 1.4.2 API Specification. Sun Microsystems, March 2006. <http://java.sun.com/j2se/1.4.2/docs/api/>

Lewallen, Raymond. Software Development Life Cycle Models. CodeBetter.Com. March 2006. >http://codebetter.com/blogs/raymond.lewallen/archive/2005/07/13/129114.aspx>

Matzke, Bervd. Ant: The Java Build Tool in Practice. Hingham, Massachusetts: Charles River Media, Inc., 2004.

Shavor, Sherry, JimD'Anjou, Scott Fairbrohter, Dan Kehn, John Kellerman, Pat McCarthy. The Java Developer's Guide to Eclipse. Boston: Addison-Wesley, 2003.

The Java Tutorial. Sun Microsystems. March 2006. <http://java.sun.com/docs/books/tutorial/index.html>