Escola Politècnica Superior
de Castelldefels
UNIVERSITAT POLITÈCNICA DE CATALUNYA

PROJECTE DE FI DE CARRERA

TÍTOL DEL PFC: **Evaluation of different methods to modify or enhance the printing process and customize print jobs**

TITULACIÓ: **Enginyeria de Telecomunicació (segon cicle)**

AUTOR: **Javier Gutiérrez Llorente**

DIRECTOR: **Daniel Coloma Pico**

SUPERVISOR: **Dolors Royo Vallès**

DATA: **September 25th, 2006**

Títol: Evaluation of different methods to modify or enhance the printing process and customize print jobs

Autor: Javier Gutiérrez Llorente

Director: Daniel Coloma Pico

Supervisor: Dolors Royo Vallès

Data: September 25th, 2006

Resum

El projecte “Avaluació de diferents mètodes per modificar o millorar el procés d’impressió i customitzar treballs d’impressió” està basat en primerament estudiar el sistema d’impressió per tal de conèixer les parts que es poden millorar o modificar per tal d’aconseguir altres tipus de funcionaments o millores en rendiments.

Un cop el sistema d’impressió esta clar, es decideix quina ha de ser la millora o modificació a realitzar durant el desenvolupament, tenint sempre en compte les possibles necessitats d’un client real estudiant el mercat. La decisió presa redirigeix el projecte cap a la creació d’un sistema anomenat “Pull Printing”.

El sistema Pull Printing consisteix en que els treballadors d’una oficina mitjana o gran només tindran una impressora virtual instal·lada als seus PCs i aquesta impressora s’encarregarà de desar els documents impresos a un servidor. Per tal d’obtenir la còpia impresa, els treballadors s’han d’apropar cap a qualsevol de les impressores de la xarxa de la oficina i introduir el número secret que prèviament han introduït al seu PC a l’hora d’imprimir. La impressora llavors es connecta amb el servidor on estan els documents guardats i els imprimeix. La principal avantatge del sistema recau en la possibilitat de obtenir les copies impreses en qualsevol lloc de l’empresa sense haver d’enviar-les allà específicament.

El desenvolupament d’aquest sistema requereix del disseny i creació de 4 mòduls software instal·lats a 3 dispositius hardware diferents, pel que el projecte es realment complet en quant a tecnologies tractades.

Per últim es fan els tests pertinents per tal d’assegurar el correcte funcionament i acompliment dels requisits de qualitat establerts.

Title: Evaluation of different methods to modify or enhance the printing process and customize print jobs

Author: Javier Gutiérrez Llorente

Director: Daniel Coloma Pico

Supervisor: Dolors Royo Vallès

Date: September 25th, 2006

Overview

The project “Evaluation of different methods to modify or enhance the printing process and customize print jobs” is based in, first of all, the study of the printing system in order to know each of the parts that can be enhanced or modifies looking forward to obtain other kind or better performances.

Once the printing system is clear, it is decided which has to be the customization or enhancement to be developed, always taking in account all needs that a real customer can have via a market study. The taken decision redirects the project towards the creation of a system called “Pull Printing”.

The Pull Printing system is intended to workers in a mid range or large office that will only have a virtual printer installed in their PCs. This printer is responsible for storing the documents printer in a server. In order to obtain a printed copy of the document, a worker has to go to a printer in the office network and type his secret number, which will have been typed before, while printing the document. Then the printer connects to the server where print jobs are stored and prints them. The main advantage of this system resides in the opportunity of obtaining printed copies of stored documents in any place of the company without having to send the documents there.

This system development requires the design and creation of 4 software modules and then installing them in 3 different hardware devices, so the project is really multidisciplinary in technologic fields.

After the development the needed tests are carried out in order to assure the correct performance and the accomplishment of the quality requirements set.

A mis padres, Rosa y Javier,
porque sin su apoyo nunca
habría llegado hasta aquí.

Table of Contents

Index of tables

Index of figures

Section 1. Introduction

1.1. Background

The main motivation of this project is the collaboration between an international commercial company with one of its interns in order to satisfy the company growing need to provide customers with customized solutions that suit each of their requirements as well as to satisfy the project's author academic needs developing a degree final project.

In the world of printing there are really hundreds of possibilities to work with, looking forward to satisfy customers needs. In this certain case the main area of interest reside in printer drivers and printer drivers customizations.

1.2. Introduction

This project is not only based in printer drivers, but in the whole end-to-end printing systems that common users can find in their offices and homes. This is because it is needed a deep understanding of architecture, processes and environments involved in an entire printing system, before being able to determine which is the best way to meet the customers requirements and giving the right solution for their problems.

The main goals of this project are carrying out an analysis of most common printing systems and all their components, and then, taking advantage of the perspective obtained from the previous analysis and also the company market needs knowledge, proposing a good technical solution to develop in this project which involves the previously acquired knowledge and also is able to satisfy customer needs in a real market.

1.3. Project requirements

This project has two main focuses; it starts with the technical documentation analysis about printing systems, later it turns over towards a technical solution development. These two phases can be summarized in a listing of project goals or project requirements:

- Obtain a general view of printing environments and printing architecture.
- Print drivers and print drivers' customization deep knowledge that permits developing a real technical proposal.
- Investigate and test the available tools for print drivers' development.
- Investigate about different technical solutions that the market needs and select one that meet the requirements of the project.
- Development of a technical solution involving the most of previously studied technologies and able to satisfy real market needs.
- Perform the project management

Section 2. Printing architecture

This section provides a global view of the printing architecture and the parts that make up the entire printing system.

2.1. Global printing architecture

Provide a first approach to printing architecture an its main elements.

The printing architecture is always based in almost two components: client and printer. In Figure 1 is shown the most common printing architecture at home users' level. This architecture involves two actors, the client and the printer. Client processes the document to be printed and translates it, by means of the printer driver, to printer language.

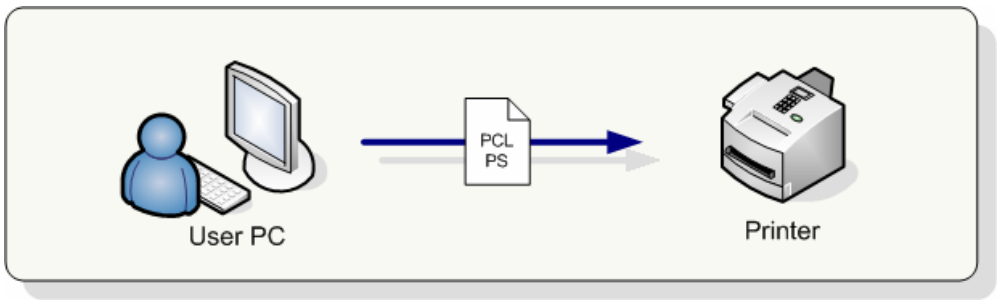


Figure 1. "Client - Printer" global architecture

Figure 1 shows the most common printing scenario at enterprise level. It is based in a print server, which manage one or more printers, and a client (or more) that connects to it, without having any knowledge of the printer.

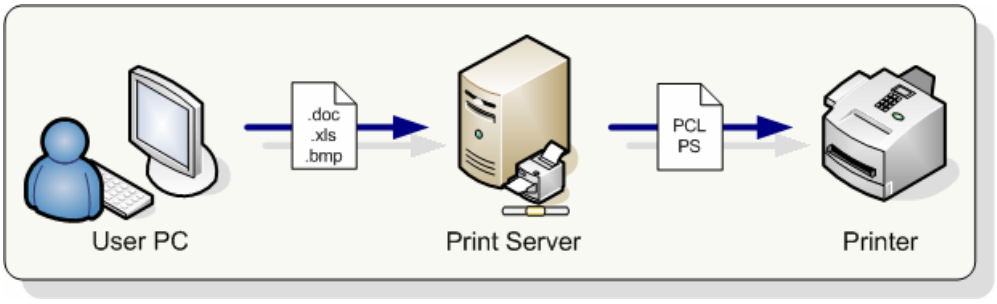


Figure 2. "Client – Server – Printer" global architecture

In both cases what printer receives is a document coded in a language that it can interpret in order to print the graphical representation of what client wanted to print. Although most modern printer devices are able to interpret some typical document types as Adobe[1] PDF directly without translating it to printer language, this is not the most common way to print.

Next sections explain these architectures with a higher detail level and explain how documents are translated at client (or server) side. Also explain how is the communication with printer established and controlled and finally, following this

[1] For more information see [1].

project's objectives, explain how the whole system can be modified in order to obtain a better performance or advanced capabilities.

2.2. Windows printing subsystem

Provide a high-detailed definition of all parts involved in Windows printing subsystem including system processes, printing data flows and printer languages.

The Microsoft Windows printing subsystem involves several tasks and processes from the application which needs to print a document to the printer.

2.2.1. Printing processes

The standard process that a document follows since the user sends it to print and the printer starts to print it is shown in Figure 3. This figure is a high-level approach as it does not show the real process names and some boxes represent a group of processes with the same purpose.

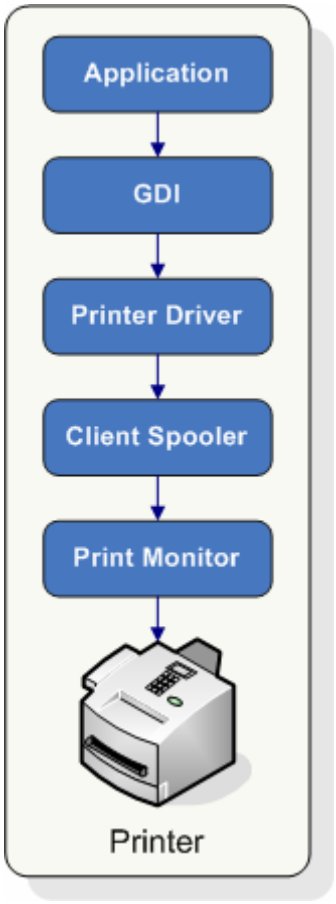


Figure 3. Printing processes high-level approach

The process structure is quite simple as it only has 6 actors, including printer and application. At first point the user, who is writing a document and wants to print it, calls the application printing routine.

This routine communicates directly with GDI (Graphics Device Interface) to translate the document to GDI calls. These calls can be stored in a file called EMF (Enhanced Metafile) to be spooled on disk or also can be passed directly to printer driver in order to generate the print file in printer language and spool it on disk in RAW[2] format.

The print processor is a part of the spooler process. If the job type is EMF, the print processor works with GDI to move the spooled print jobs from the hard disk to the printer driver. If the data type is RAW, the print processor helps in moving the job directly to the printer. The print processor controls certain features of the print job, depending on the kind of job. For example, if the data type of the spooled job is EMF, the print processor can help in printing the document in reverse order or it can print it in booklet form instead of the default one-page-per-sheet form. Winprint.dll is the name of the default print processor on Windows. Winprint.dll is part of Localspl.dll.

Software vendors can develop their own print processors for custom data types. A printer vendor can also develop a custom print processor if the vendor has its own printer driver or supports a data type other than the three that WinPrint.dll

[2] The raw data type indicates that the print job has already been fully rendered by the GDI and printer driver, and it does not need any more processing.

supports. Also, a vendor can write a custom print processor if the vendor wants to add functionality that Winprint.dll does not provide. Usually, the print processor is installed when the printer driver is installed.

The Print monitor is composed of to parts, the language monitor and ther port monitor. The language monitor provides the common language that is needed for the client and printer to understand each other in bidirectional communication so that you can configure the printer and monitor printer status. You can request configuration and status from the printer, and the printer can send unsolicited status information (such as “Paper tray empty”) to the client.

The local port monitor controls parallel and serial input/output (I/O) ports where a printer might be attached. The local port monitor sends print jobs to local devices, including those on familiar ports such as LPT1, COM1, or FILE (Print to File).

A second local port monitor, Usbmon.dll, controls communication through universal serial bus (USB) ports. Usbmon.dll installs automatically whenever you plug a USB printer into the correct physical port on your computer.

All other port monitors that are supplied with Windows OS are remote monitors that enable printing to remote printers. An example is the standard TCP/IP port monitor.

2.3. Network Printing Protocols

Description of protocols used in communication between printers and print servers.

The following protocols are used when providing network print services. Note that some protocols might not be used, depending on the needs of the client computers.

Server Message Block (SMB)

A file-sharing protocol designed to allow network computers to transparently access files that reside on remote systems over a variety of networks. The SMB protocol defines a series of commands that pass information between computers. SMB uses four message types: session control, file, printer, and message.

Line Printer Remote (LPR)

A connectivity tool that runs on client computers and that is used to print files to a computer running a Line Printer Daemon (LPD) server.

Line Printer Daemon (LPD)

A service on a print server that receives print jobs from Line Printer Remote (LPR) tools that are running on client computers.

Remote procedure call (RPC)

A message-passing facility that allows a distributed application to call services that are available on various computers on a network. Used during remote administration of computers.

Internetwork Packet Exchange (IPX)

A network protocol native to NetWare that controls addressing and routing of packets within and between local area networks (LANs). IPX does not guarantee that a message will be complete (no lost packets).

Internet Printing Protocol (IPP)

The protocol that uses the Hypertext Transfer Protocol (HTTP) to send print jobs to printers throughout the Internet.

Transmission Control Protocol/Internet Protocol (TCP/IP)

A set of networking protocols widely used on the Internet that provides communications across interconnected networks of computers with diverse hardware architectures and various operating systems. TCP/IP includes standards for how computers communicate and conventions for connecting networks and routing traffic.

AppleTalk

The set of network protocols on which AppleTalk network architecture is based. The AppleTalk Protocol is installed with Services for Macintosh to help users' access resources on a network.

Simple Network Management Protocol (SNMP)

A network protocol used to manage TCP/IP networks. In Windows, the SNMP service is used to provide status information about a host on a TCP/IP network.

2.4. Printer drivers

Description of printer drivers and UNIDRIVER architecture

Printer drivers contain information that is specific to the printer that is used. Printer drivers reside on users' computers and are used by the GDI to render print jobs.

2.4.1. Introduction to drivers

A printer driver is a software program that understands how to communicate with printers and plotters (devices used to draw charts, diagrams, and other line-based graphics). Printer drivers translate the information that an application sends through the GDI into drawing commands that the printer understands. These drawing commands are for creating text and graphics. Various drivers must be installed on the print server to support different hardware and operating systems. For example, an administrator who uses a computer that is running Windows Server 2003 and who shares a printer with clients that are running Windows 95, Windows 98, and Windows Millennium Edition, might need to install the appropriate drivers so that clients that are running these versions of Windows are not prompted to install the missing drivers.

The printer driver sends information about the printer settings, including the specifications that are needed to produce each character of the document, to the GDI. The printer driver also transmits helper services or utilities required to make the output print correctly.

2.4.2. Universal printer driver (Unidriver)

The Unidriver is also called the raster driver because it supports raster (bitmap) graphics printing and is compatible with many printers. This driver supports the following features:

- Color printing at various depths, such as 4 bits per pixel (bpp), 8 bpp, and 24 bpp.
- Scalable TrueType and OpenType fonts, device fonts (including double-byte), grayscale printing, font substitution, run length encoding (RLE),

Tag Image File Format (TIFF) version 4.0, and Delta Row Compression (DRC).

- An extension interface that allows printer manufacturers to customize the driver for specific models.

The Unidriver contains the following component files:

- Unidrv.dll is the printer graphics driver file for printer languages based on raster images, including most inkjet and dot-matrix printer languages.
- Unidrvui.dll is the configuration file. It displays the user interface for Unidrv.dll.
- Unires.dll, Stdnames.gpd, Ttfsub.gpd and some other support files.
- Raster minidriver, which has the file name extension .gpd, is the data file, and is also called the characterization file. The file name depends on the printer or printer family.

Section 3. Enhancements and customizations

This section explain how can we enhance or customize printing process and print jobs in order to improve the performance or to give more functionalities to printing.

3.1. How to add extra functionality to UNIDRIVER

The UNIDRIVER architecture permits the customization of its drivers using the plug-ins that can be attached to it. The whole driver functionality can be modified by adding a UI plug-in and a render plug-in.

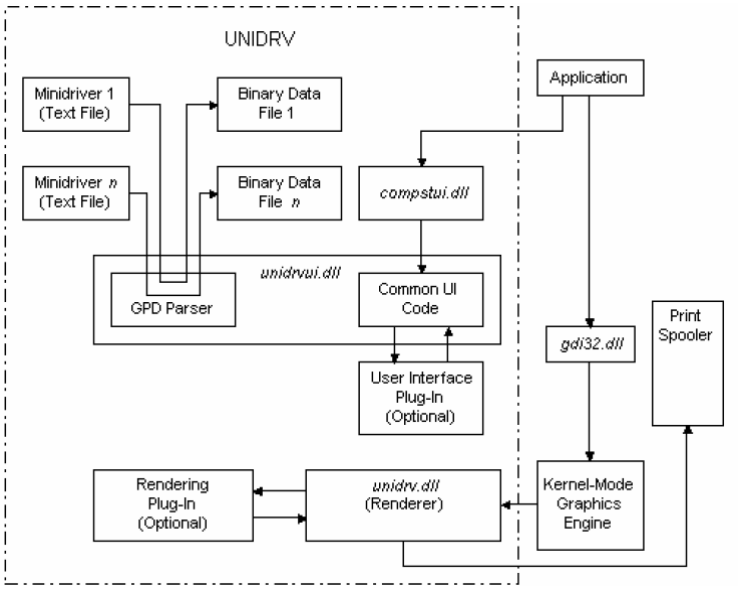


Figure 4. UNIDRIVER modules schema

- UI plug-in

The UI plug-in can add extra functionality to the driver printing preferences UI, or if it is needed it can totally replace the printing preferences window by new customized one.

This plug-in is executed in the client by any program that wants to print and it has the possibility of communicating with the render plug-in to sending some data at rendering time.

- Render plug-in

The render plug-in hooks the driver DDI calls, so it has the possibility for, if it is needed, replace any rendering function of the driver, as the draw circle or write text could be.

The render plug-in is also allowed to modify print job headers because controls the rendering start, data processing and stop.

3.2. How to add extra functionality to printing devices

This section presents different ways to modify or enhance the printing system from printing device side. Also it explains with a deeper detail level several solutions.

There are many ways to extend functionality in printing devices side, depending on the desired enhance one of the options listed below these lines could be more suitable than the others.

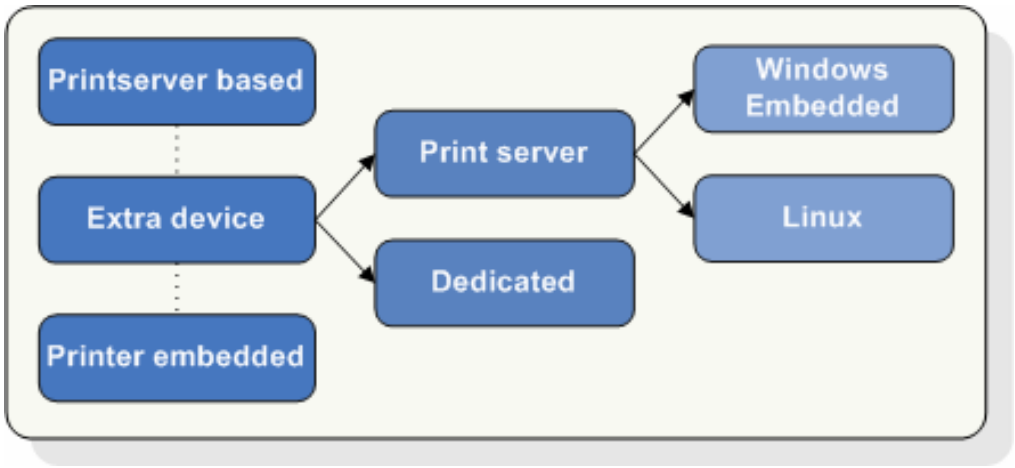


Figure 5. Printing devices enhancements' types

Enhancements at printing device side:

- Embedded: software running in printers embedded OS
- Server based: software running in printers doing tasks of print server
- Extra device
  - Print server: Print server embedded OS in a reduced device.
  - Dedicated: Hardware dedicated to do the job that is made for.

3.3. What can be improved in print server

This section describes things we can modify/improve of printing system

- Accounting

By developing an accounting solution (most printers firmware have a simple accounting application embedded) the administrator can control all kind of user ink/paper/copies consumption in the way of controlling user rights or simply taking a record of the consumption per user.

- Compression / Encryption

The compression or encryption of print jobs is not a customization very requested in the market because most of the printers are located in private enterprise LANs with security controls and with high bandwidth available.

- Watermarking

Watermarking is the technique of adding an image in the documents printed background. It is very useful for companies that always print legal documentation that must contain the identification of copy, or companies that always print documents with an standard header of the company.

- Pull Printing

The pull printing system is intended for midsized or large offices with a high use of printing resources. Depending on customer needs this system can leverage several benefits as traffic bandwidth reduction, printing control and statistics or users comfort.

3.4. Solution chosen

Contains the solution to develop choice and the positive arguments

The solution chosen to be developed in this project is the **pull printing system**. The choice meets all requirements of this project; it is a customization solution that interacts with several technologies in study inside this project. It is also an end-to-end solution that can fulfill real customers' requests in the market

A pull printing system can offer different kinds of improvements to customers printing environments depending on their needs. As this project has limitations of time and human resources the system that will be implemented will offer one of the main benefits of pull printing, but it won't be a complete system as others that are available in the market. A complete pull printing system is designed for offering the following benefits, and several more not as demanded as these ones here:

- Only one printer queue: Normal users have access to only one printer queue installed in the print server. All printers in the office network can retrieve the job sent to the server.
- Job accounting: As all printing jobs are stored and organized in server, it's easy to develop a job accounting control like permitting use of color only to certain users.
- Processing and networking use reduction: As all print jobs are spooled in server the user's PC is able to continue with other processing but rendering. Also as long as jobs are stored permanently at server, network traffic reduces due to the chance of printing N-times each job sent only once to server.
- Paper waste reduction and data protection: Several companies such as Microsoft are deploying pull printing solutions in their sites in order to avoid the printed jobs that remain in printers at the end of the day with no owner. These "forsaken" printings cause large companies to waste huge quantities of paper with the corresponding waste of money and the negative environmental impact. Furthermore if forgotten documents contain confidential information, the company must destroy them properly to avoid data protection breaks and legal troubles.

Section 4. Solution design and implementation

Once all project background has been studied in previous sections, now in this section the main objective of this project is explained. The solution to implement is specified giving firstly a high-level approach, and then the software design with UML diagrams and finally the solution implementation and test details are explained.

4.1. Solution approach

This section explains the basics of what will be implemented in this project.
It details functionalities and high level approach of solution.

The solution that will be implemented in this project is a "**Pull Printing**" system. Pull printing basics are shown in Figure 6. A pull printing system lies in 3 main steps: **(1)** user sends a print job to server (does not select any physical printer device) and it is stored, then **(2)** user can go to any printer connected to same network as print server (no matter which printer) and retrieve his stored jobs. Lastly **(3)** the server sends print job to the printer that requested it and user can get his document printed.

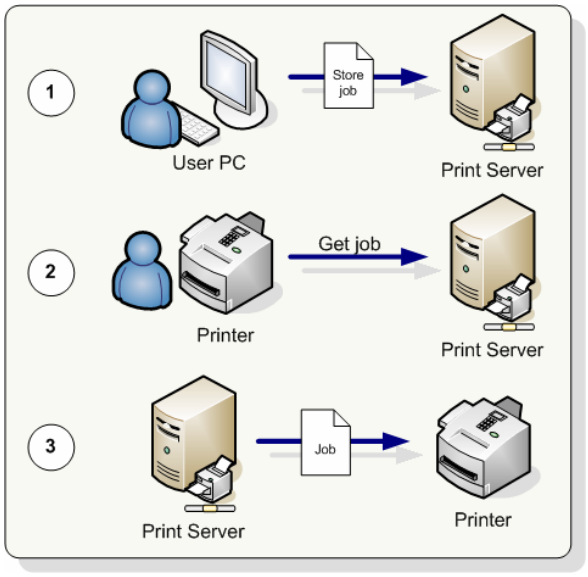


Figure 6. Designed solution: Pull printing

Then in a real scenario, for example inside a building of a midrange enterprise, there will be several users and several printers connected to the network and the communication between printers and users PCs will be centralized in one server.

In Figure 7 is represented a real scenario schema with only 3 users and 6 printers. In this case each user can print to the print server computer and then go to any of the 6 printers in the network to retrieve the job stored in print server.

Note that the communication between users and printers directly is not forbidden, but it's only allowed for printing in a normal direct connection via printer IP address, without a pull printing system.

Since previous considerations about hardware are somewhat clear, now it's also needed to give a first approach to system software part. Pull printing solution involves developing several software modules for each of user PC, print server and printer because all three are involved in this system. In user PC side the print job has to be modified to include the "pull printing" option.

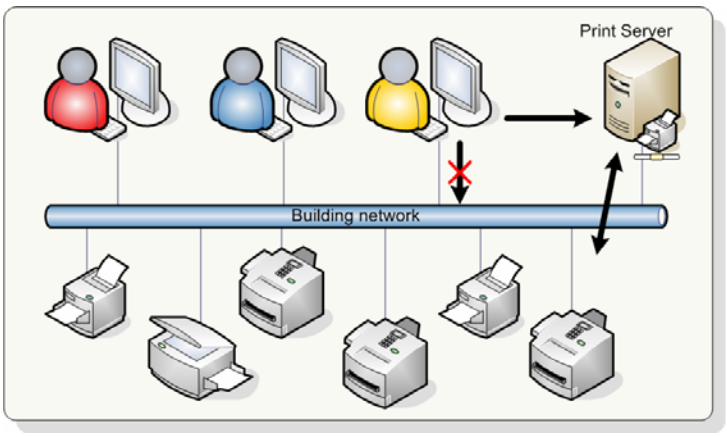


Figure 7. Pull printing *real* scenario

In the server some modifications must be done to print server in order to store jobs and listen printers for pull printing requests. Finally in printers an application has to be developed to permit users get jobs stored in server.

4.2. Design

The design of proposed solution is detailed in this section. It includes process flows, UML diagrams, environment and process schemas and more.

The pull printing solution consists of 3 physical devices: User PC, Print Server PC and Printer Device. These three actors have four software modules installed due to server PC has two functions: store jobs and listen printers for jobs retrieval requests.

In Figure 8 is represented the functional diagram with all three actors and their software components to be developed.

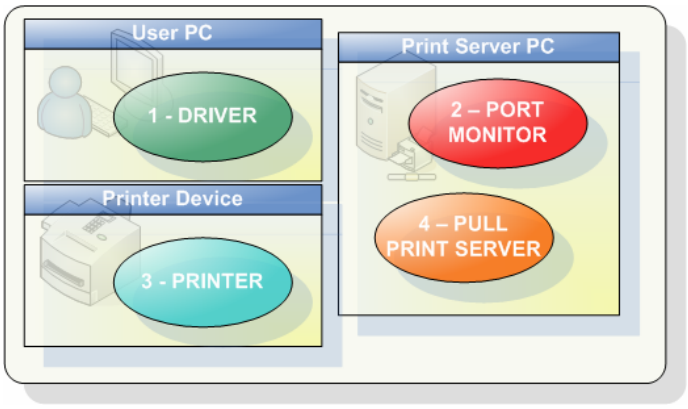


Figure 8. System functional description

In Figure 9 there is represented the pull printing system functional overview, including examples of the user interface of each one of the system parts. In User PC the driver (1) is installed in the "Pull Printing" printer. In Print Server PC the port monitor (2) is installed as the printer Properties windows shows, and is who carries out the disk job storage. The Printer Device (3) retrieves stored jobs sending the PIN number via the network to the Pull Print Server (4).

In this section the whole system design and architecture is detailed and is also represented the entire system flowchart including these 4 software parts, which since this point will be called Driver, Port Monitor, Pull Print Server and Printer.

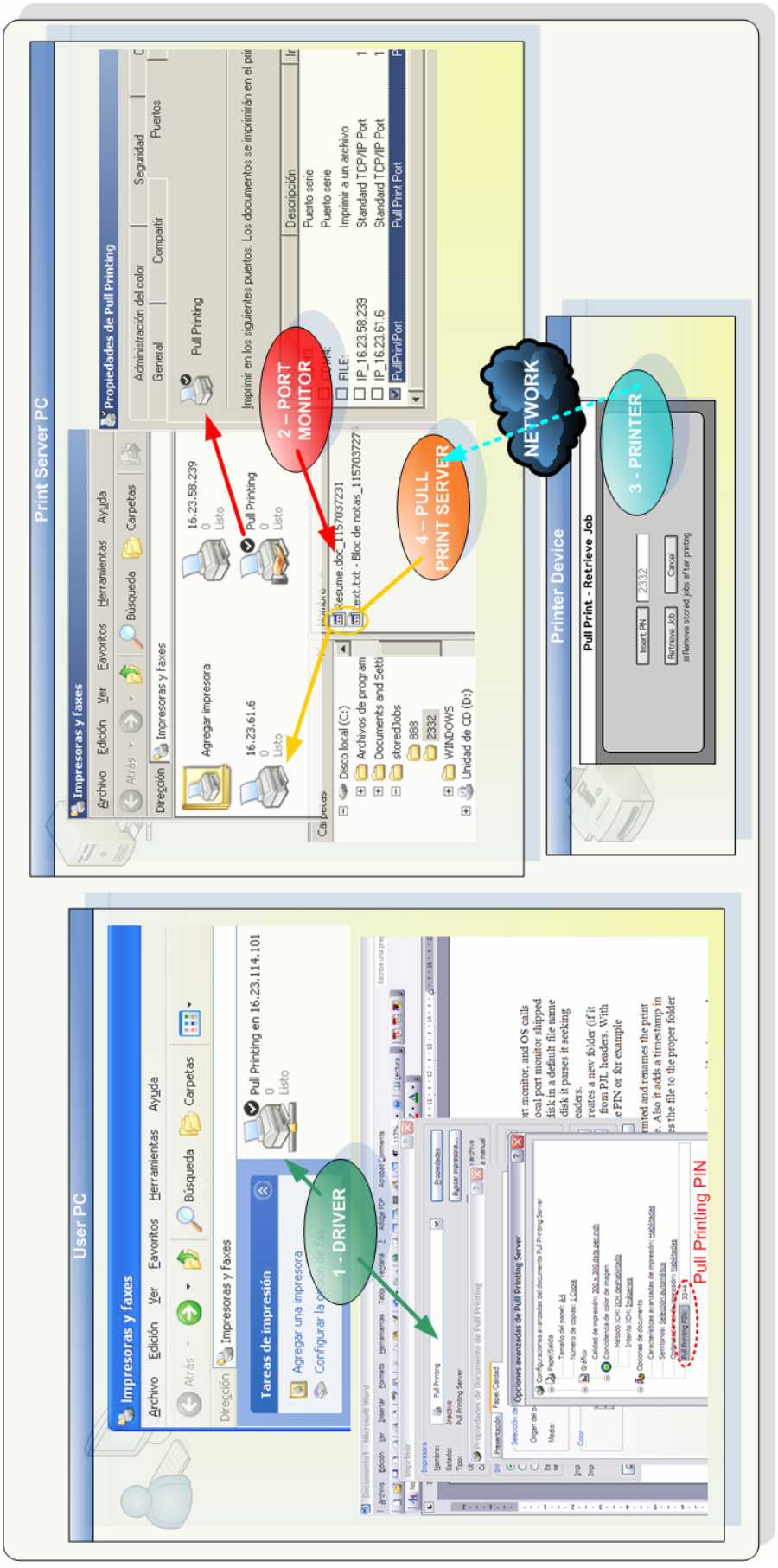


Figure 9. Pull Printing system functional overview

4.2.1. Pull printing system

In Figure 10 it is represented the pull printing system states diagram. Each of states is grouped inside one of the four parts of the system described before. The system has two “entry points”, defined as the points where a user interaction is needed, to start a pull printing job and to retrieve it from the printer. It also has one “final point”: the printer. Note that there are 3 “printer” entities represented in the diagram, even though all 3 are exactly the same device.

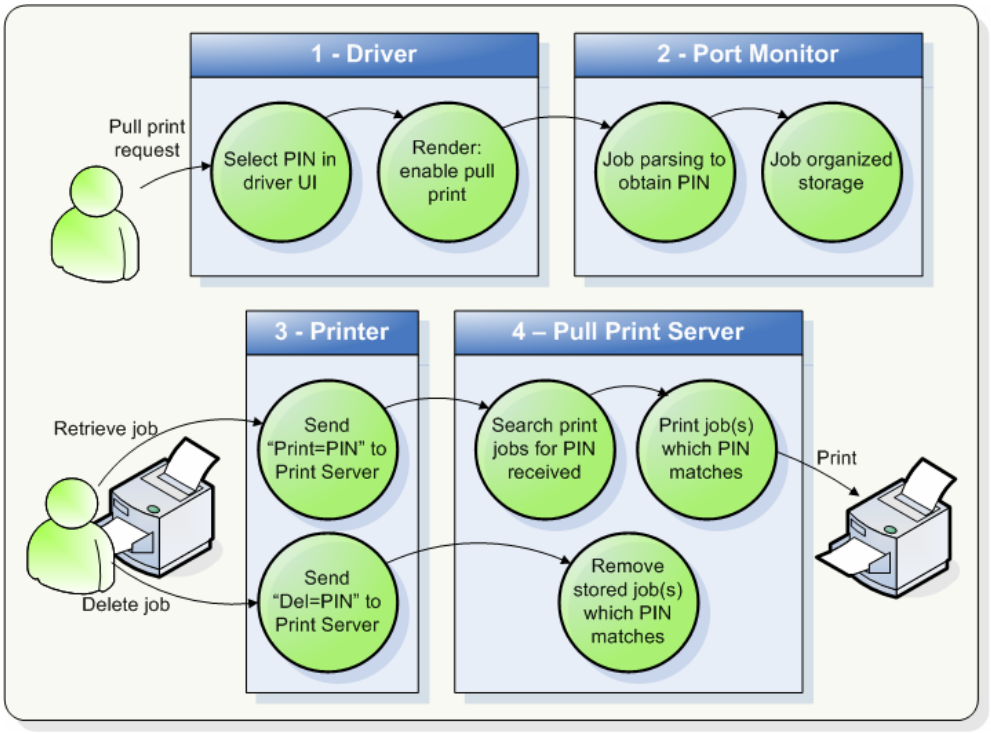


Figure 10. Pull printing solution states diagram

Below is described each one of the parts involved in the pull printing system:

1. Driver

   The Pull Printing Driver, which will be installed in all users PCs, has two purposes. The first one is to show a customized printing preferences dialog with an added field to insert the Personal Identification Number (PIN) to identify the job. The second one is to add a new header line in PJL file including the previously mentioned PIN number.

2. Port Monitor

   The Pull Printing Port Monitor will be installed only in the server. Its purpose is to store print jobs in disk. In order to do that in a proper manner it parses the PJL contents to locate the PIN number, and then it creates a folder named with this PIN and stores the print job there.

3. Printer

   The Printer will have an application with the main goal of asking user for him to type the PIN number and sending it to server. When all print jobs

are properly printed, the app asks user if he wants to delete them from server, and if the answer is yes then the printer sends again the PIN number with another header.

4. Pull Print Server

   The Pull Print Server has two functions: listen printers for PIN numbers sent and then seek disk for the stored print job(s) identified with the received PIN, when found, print or delete them, depending on the request received from printer.

4.2.2. Use case diagram

A good way to understand which will be the main functions of this system is the UML use case diagram shown in Figure 11. This diagram shows the “actors” seen as people (or systems) able to do something and the “actions” they can perform or “use cases”.

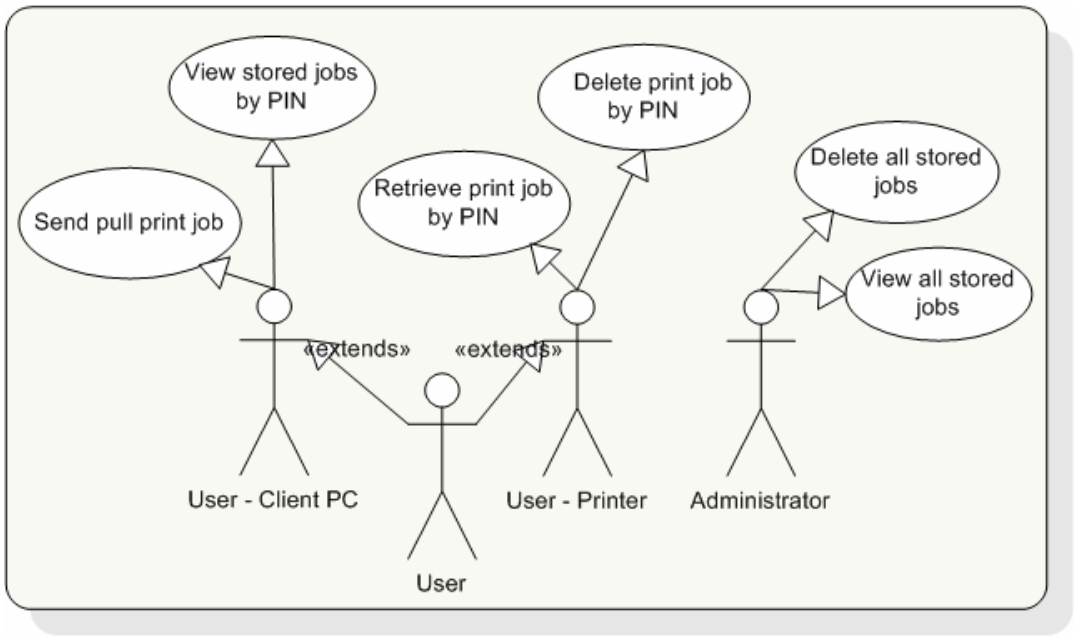


Figure 11. Solution design: Use case diagram

For the “pull printing” solution two abstract actors have been defined called “User – Client PC” and “User – Printer” and one “real” actor, called “User”, who extends other two actors’ actions. From its PC, the User is able to send print jobs and view and delete stored jobs. From the printer the User only can print a job stored in server. Another actor called “Administrator” is created to manage (view/delete) print jobs form all users.

4.2.3. System parts detailed description

Once the pull printing system use cases, actors, flows and parts are detailed in sections “4.2.1. ” and “4.2.2. ”, in this section are described with a higher detail level each one of the components of the whole system, including modules, architectures, dlls, classes, resources, etc.

4.2.3.1. Driver

The driver implementation is based in the structure of UNIDRV driver seen in 2.4.2. . As explained there, UNIDRV permits adding plug-ins to the standard driver, and starting from a working driver for any PCL Laser model, included in all Windows XP releases, it's feasible to create a customized driver in a simple manner.

In this project we are intended to create a new printing preferences dialog with a new option for enabling the pull printing and for asking users to they introduce the PIN number that will uniquely identify the print jobs sent by each user. So a UI Plug-in will be implemented and added to the working PCL driver.

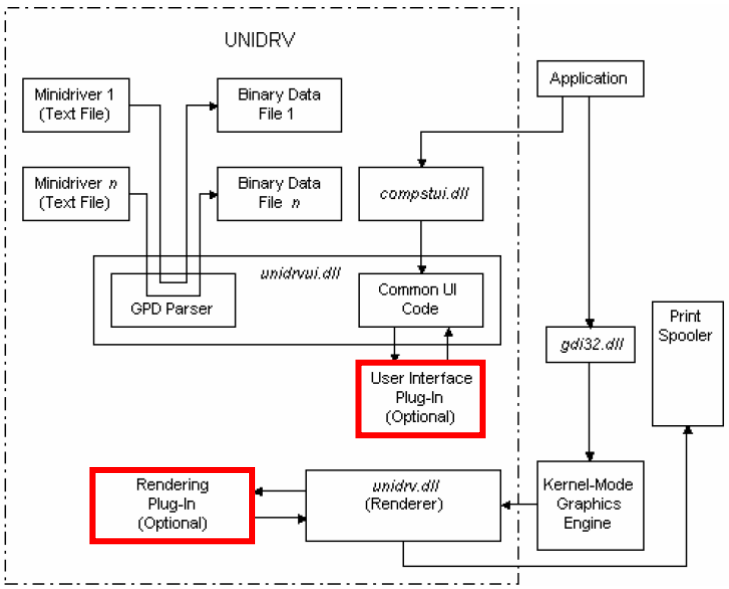


Figure 12. UNIDRV Architecture – Driver customization plug-ins

It is also needed to do some rendering tasks to include the PIN number selected at the printing preferences UI in the print job PJL file. So one render Plug-in has to be implemented in order to add the proper PJL header in the print job.

Figure 12 shows the architecture of UNIDRV based drivers. Note that this figure has two parts highlighted with red boxes, both corresponding with customization plug-ins.

So the development in this case will be focused in create two new plug-ins in order to customize the driver functionality by default. These plug-ins will have to be able to communicate between them, via the DEVMODE structure, because the PIN number will be set in preferences dialog (UI plug-in) and written down in PJL at render time (Render plug-in). To see information about how customizing printer drivers, visit MSDN site at [2].

A good way to starting with the driver design is following the architecture of one of the customization plug-ins examples in the DDK (see Section 4.3.1.1. for more information). The example that suits better the initial requirements noted before is named "WaterMarkUni".

The WaterMarkUni example demonstrates how to produce customizable watermark page by controlling PCL data injected in the print job. The sample includes a UNIDRV-based rendering module responsible for the injection of the PCL data and also a UI module.

This sample is very useful for knowing the required COM interfaces, required functions with sample code and how to use the private DEVMODE section to communicate between the UI and rendering modules. Both plug-ins will need to use the IPrintOem COM interfaces, IPrintOemUI for UI and IPrintOemUni for rendering. These interfaces are well-documented in DDK's help and also in MSDN reference guide, for more information see [3] and [4].

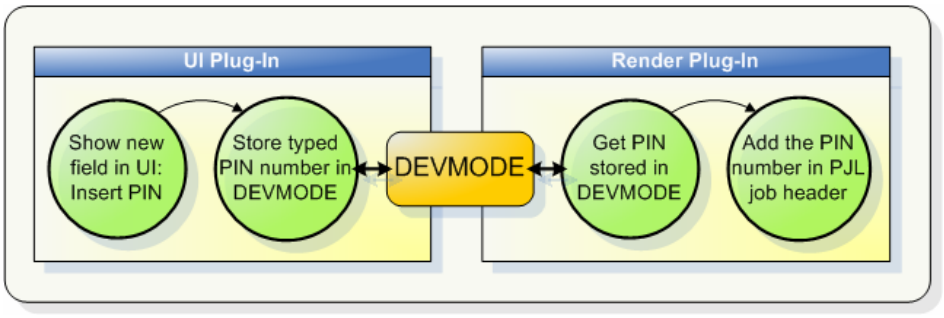


Figure 13. UI and Render plug-in sharing DEVMODE

The schema in Figure 13 shows how UI and Render plug-ins will share the needed information, the PIN number actually, via the shared DEVMODE. The UI plug-in stores the PIN number in

4.2.3.2. Port monitor

The port monitor implementation as well as printer driver implementation follows the UNIDRV architecture and the DDK also provides some examples with sample source code and an introductory documentation including the COM interfaces' functions reference.

As seen before, in 2.2.1. , the port monitor is responsible for the final communication between the printer and the PC. The job that reaches the port monitor usually has been rendered and contains only printer language. The main purpose of the "customized" port monitor, or the port monitor that will be implemented in this project, is to store the print jobs in the hard disk, but also to organize them reading each job's pin number from PJL headers. In order to do this it follows the steps shown in Figure 14.

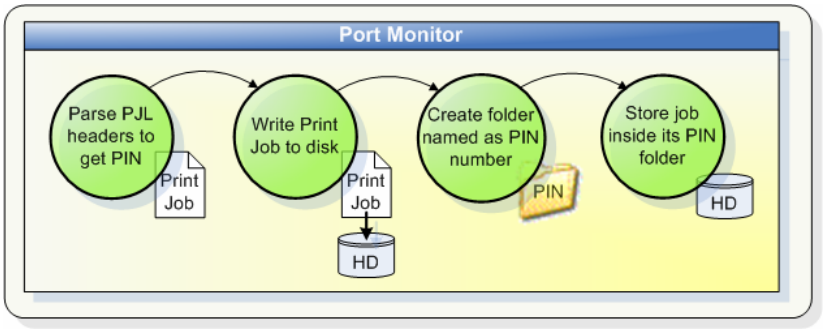


Figure 14. Port monitor flow diagram

This figure (Figure 14) shows how when print job arrives to port monitor, and OS calls its functions to write data to printer, the port monitor acts as local port monitor shipped with all Windows operating systems, writing the print job to disk in a default file name and path. But while port monitor receives data to write to the disk it parses it seeking the PIN number inserted by the driver (see 4.2.3.1.) in PJL headers.

After writing the print job contents to disk, the port monitor creates a new folder (if it doesn't exist yet) and names it with the PIN number obtained from PJL headers. With this organization is easy to obtain all documents stored for one PIN or for example delete them all.

Then port monitor obtains the filename of the document printed and renames the print job stored in the default location with the original filename. Also it adds a timestamp in the filename to avoid conflicts. Once renamed it also moves the file to the proper folder depending on the PIN number.

4.2.3.3. Pull Print server

The pull print server is the system part that carries out the communication with printers and also who calls the last command for printing the documents. As a server, it has to wait for clients requests. When a connection request reaches the pull print server it starts a new thread to attend it. Each new thread will be responsible to parse the data received from printer and to engage the proper actions. This multithreading environment has been designed to deal with high-demand printing environments as long as several printers will be able to ask server for jobs at the same time.

It is important to notice that the server is limited to deal with 15 concurrent threads because, following design considerations, it is expected that each thread will be able to perform the actions and end its execution in a short time, less than 5 seconds since thread is created, so doing a simple calculation, the system can manage:

15 threads/5 sec x 60 sec/min x 60 min/hour = **10800 printings/hour**

The server, as explained in section 4.2.3.4. , will receive from the printer a command for printing jobs or for deleting them. So the actions that each thread will have to do, after parsing the contents of data received from printer, are basically 2: print or delete.

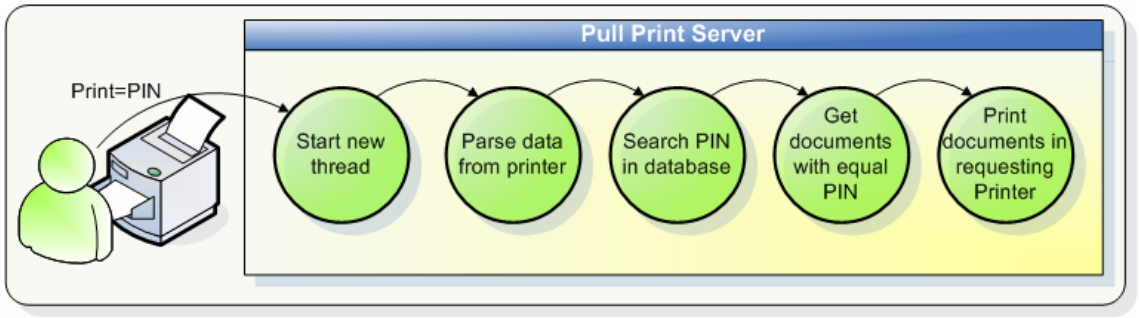


Figure 15. Pull print server data flow chart

To print a job, the pull print server follows the steps shown in Figure 15. When a message from a printer arrives to the pull print server it starts a new thread. Every thread performs the same actions, starting from parsing the data received and then looking for the PIN number in the stored jobs database. If there are jobs stored with this PIN number, then the pull print server decides what to do, either delete or print them, depending on the message from printer. In Figure 15 is represented the printing process, but deleting process is exactly the same.

4.2.3.4. Printer

The application designed for running embedded in the printer has a simple purpose: to send PIN number typed by users to the pull print server. This pin number is encapsulated in a TCP packet with the command to perform, which can be either print or delete jobs identified with that PIN.

As said before the printer embedded application will be controlled by users via the printer´s front panel, which will show the screens designed for this purpose. In 5.1.4. there are several screenshots for the different application states.

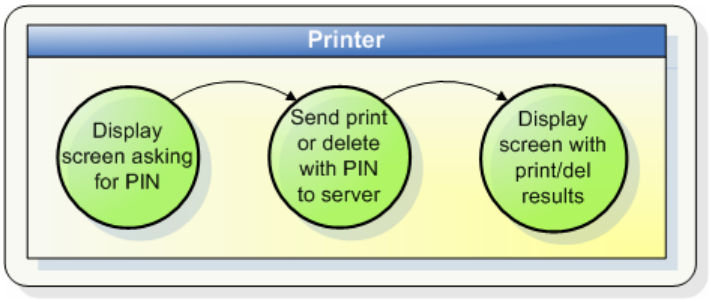


Figure 16. Printer embedded application flow diagram

In Figure 16 there is a diagram that shows the 3 main states of the embedded application. The first one only waits for user input typing the personal PIN number, the second one communicates with printer to send the PIN and lastly the third one shows if the printing or deleting of jobs succeed or not.

4.3. Implementation

This section contains all information about the development process, the tools used, the problems found and also several listings with source code examples.

4.3.1. Environment and tools

As seen before, the pull printing system as 4 parts. Each part has different environment of execution and, as a result, a different environment of development. Below these lines are listed the environment, libraries, programming languages and tools used for developing each part of the whole system. Note that there are only three sections, as print driver and port monitor development are carried out through the same environment.

4.3.1.1. Driver and port monitor development environment

The world of printer driver development, as well as other kinds of drivers, for Windows platforms is based in the Microsoft Driver Development Kit (DDK). The DDK provides driver developers with tools, libraries, compilers, preconfigured environments and source examples, and also a detailed documentation about everything needed to develop, or modify an OS driver. For more information about the DDK visit [3]. In order to provide the DDK compiler with proper libraries it's also needed the Microsoft Platform Standard Development Kit (SDK) downloadable from [7].

For editing source files it is not required any software as DDK is used for compiling and building the driver. But for debugging purposes the platform SDK includes the software WinDBG as a part of Debugging Tools for Windows package, also freely downloadable from [8].

Once these tools are installed any text editor can be used to create the new driver/plug-in/monitor or to edit one of the DDK's examples. Then, in order to compile and build the driver, the proper DDK build environment has to be opened and then DDK "build" command has to be executed in the right directory. For more information about these steps see [9] or download the Driver Developing Introduction Guide here [10].

4.3.1.2. Pull print server development environment

The pull print server will be developed as a Windows console application so the development environment will be the Visual Studio C++ with Win32 system libraries.

4.3.1.3. Printer development environment

The printer side of the pull printing system will be installed in the printer as an embedded application. There is available an SDK to develop applications for the printer embedded OS which will be the environment to develop the printer application, written in Java.

To edit Java files there are available several editors, but the preferred for this project is one of the most powerful IDEs in the market, it is the Eclipse Platform (see Eclipse website at [11]) and it is free of any charge for the developers community.

To compile the embedded applications the SDKs recommend building an environment with the 1.3 version of Java Virtual Machine (see Java website at [12]) and also to use the proper libraries provided with SDK depending on the purpose of the application.

4.3.2. Development

In this section it is included the development process details, all problems found during development stage of the project and some important technical details about each module implementation.

4.3.2.1. Driver development

For the driver development, as said before, it is needed a generic PCL driver in order to add the plug-ins that will be developed. If printers in the network have the same driver, it is recommended to get it as the basic driver, so when users install Pull Printing printer from print server PC, can configure all printing preferences of the original driver. If in the network there are different models it's better to select a generic driver with common printer preferences, as color, orientation or finishing options can be.

A minimal driver uses only DLLs of the system and it looks like only 2 files: a .GPD and an .INF (see 2.4.2. for more information about UNIDRV drivers' files).

In this project the driver development starts from a minimal driver (GPD + INF), but since it is needed to add UI and rendering plug-ins to the driver, one more extra configuration file is needed, as well as plug-ins DLLs: the .INI file.

The INI file contains the names of plug-ins that the printing system will call when printing driver is executing, so we will have an INI file like the listed in .

The INF file has to include this new INI file, as well as both DLLs, each one corresponding to each render and UI plug-ins, in order to the system to copy them at installation time.

Listing 1. GPD definition of CallBack command

```
*Feature: PullPrintingCmdCallback
{
    *Name: "PullPrintingCmdCallback"
    *FeatureType: DOC_PROPERTY
    *Option: NAME
    {
        *Name: "NAME"
        *Command: CmdSelect
        {
            *Order: DOC_SETUP.4
            *CallbackID: 24
        }
    }
}
```

The GPD file has to include, as well as standard PCL printers configuration commands for print jobs, one extra command called CallBack.

This CallBack command, shown in Listing 1, defines a CallBackID which will be the way that the render plug-in will identify the right callback to process. The callback also defines an Order parameter, which will tell Unidrv when it is time to send the command to the rendering plug-in. In this case it has been set to "**`DOC_SETUP.4`**", corresponding to the last header before starting with PCL commands. The actions taken when the callback is called by printing system are defined in render plug-in, explained in the next section.

4.3.2.2. Driver customization plug-ins development

This section explains how the driver customization plug-ins have been developed. The plug-ins, as said before, are based in one example of the Microsoft DDK in order to obtain in a short time a good prototype of a working environment.

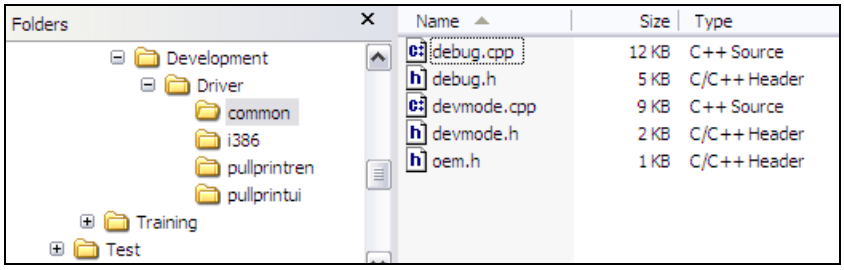


Figure 17. Driver plug-ins common source files

Each plug-in, either UI or rendering, has a standard set of source files following the IPrintOEM COM interfaces and also DLL standard entry points. Some of the source files are shared by both plug-ins because developers can save time in coding files twice, one for each plug-in. These common files, listed in Figure 17, are mainly header files with common libraries and common data structures definitions. One of these data structure definitions, and one of the most important parts of the customized driver, is the private DEVMODE, a modified

version of the standard DEVMODE with several new fields. The private DEVMODE definition, where new customized fields are added to the DEVMODE structure, is shown in Listing 2.

Listing 2. Private DEVMODE definition – devmode.h

```
#define PULL_PRINT_PIN_SIZE        12
#define PIN_DEFAULT_VALUE          L"0000"

//////////////////////////////////////////////////////////
//      OEM Devmode Type Definitions
//////////////////////////////////////////////////////////

typedef struct tagOEMDEV
{
    OEM_DMEXTRAHEADER   dmOEMExtra;
    WCHAR               szPin[PULL_PRINT_PIN_SIZE];

} OEMDEV, *POEMDEV;
```

In the listing there are defined the default values for PIN max size and for the input text box initial value. Then there is the definition of the private DEVMODE data structure, with two extra variables, in this case one new header called *dmOEMExtra* which will contain information about size and signature of the DEVMODE and also a new string called *szPin* that will contain the PIN typed by user in the printing preferences UI.

UI PLUG-IN

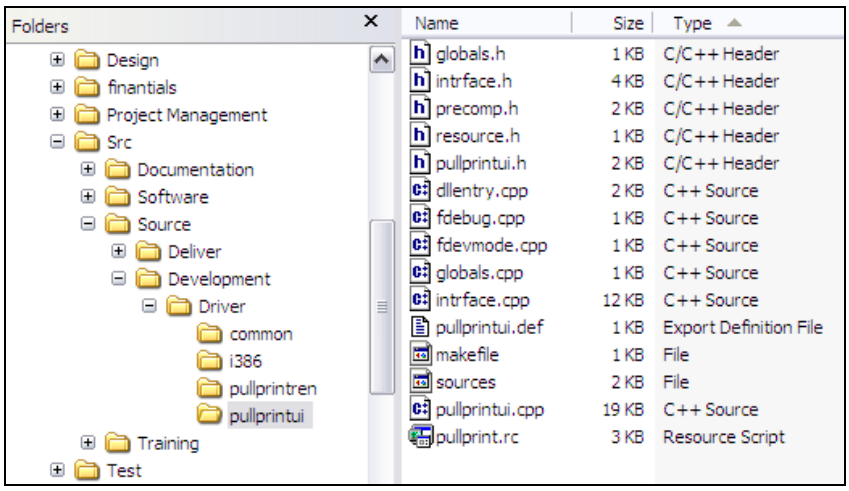


Figure 18. Driver UI plug-in source files

In Figure 18, there is a listing of UI plug-in source files and files needed by DDK to compile them (see DDK help for more info about building plug-ins with DDK) and build the DLLs. The most important UI plug-in source files are reviewed below these lines and after that a deep analysis of functions developed is done.

UI plug-in source files revision:

- Header files (*.h): Declaration of variables and function headers.
- pullprint.rc: Resource file that defines strings and DLL information.
- pullprintui.def: The file that list the exported functions

- dllentry.cpp: Entry points for the system DLL.
- fdebug.cpp: Debugging and logging functions.
- fdevmode.cpp: The file only “redirects” to shared devmode.cpp.
- globals.cpp: Global variables declaration and libraries including.
- intrface.cpp: Source module that implements the OEM COM Printer Customization UI interface (known as IPrintOemUI).
- pullprintui.cpp: Contains the functions that customize the Printer Properties UI.

Below there is a description of the most important files in the development, in this case intrface.cpp and pullprintui.cpp, and which functions has been used to develop the pull printing system.

Firstly there is the description of intrface.cpp which implements the interface IPrintOemUI, so it is who manages the system calls to its functions. Then it is reviewed the pullprintui.cpp source file which includes the functions that generate the modifications in the printing preferences UI when called by intrface.cpp functions.

intrface.cpp

This source module contains all the functions of the IPrintOemUI COM interface but implements only the functions needed by the plug-in, in this case only two: DevMode, which is called by printing system when the DevMode instance is needed and CommonUIProp, which is called by the printing system when printing preferences UI is shown. The other functions, since they are not needed to be customized, return the code “*E_NOTIMPL*”.

Function “*IPullPrintUI::DevMode*”: This function purpose is to return an instance of the customized DevMode data structure seen before. The coding of this function performs a call redirection to “*hrOEMDevMode*” function, contained in the previously explained devmode.cpp source module, as shown below in Listing 3.

Listing 3. IPullPrintUI::DevMode source code

```
HRESULT __stdcall IPullPrintUI::DevMode(
    DWORD  dwMode,
    POEMDMPARAM pOemDMParam)
{
    VERBOSE(DLLTEXT("IPullPrintUI:DevMode(%d,%#x) entry.\r\n"),
                    dwMode, pOemDMParam);
    return hrOEMDevMode(dwMode, pOemDMParam);
}
```

Function “*IPullPrintUI::CommonUIProp*”: This function carries out the customization of the printing preferences UI. In the source code of this function, as shown in Listing 4, there is no other implementation than a call to pullprintui.cpp function named “hrOEMPropertyPage”, function explained below in pullprintui.cpp section.

Listing 4. IPullPrintUI::CommonUIProp source code

```
HRESULT __stdcall IPullPrintUI::CommonUIProp( DWORD  dwMode,
                                POEMCUIPPARAM   pOemCUIPParam)
{
    VERBOSE(DLLTEXT("IPullPrintUI:CommonUIProp entry.\r\n"));
    return hrOEMPropertyPage(dwMode, pOemCUIPParam);
}
```

pullprintui.cpp

This source module contains the functions needed to customize printing preferences UI and also to manage the new data entered by users. These functions are called by intrface.cpp source module when printing system calls its IPrintOemUI interface functions.

Function "*OEMUICallBack*": This function is called automatically by the printing system when a CallBack command is sent by the UI, it is, when users modify any field or object contained in the printing preferences UI. This function purpose is only to capture "Apply" events (callbacks), just in the moment the changes done by users are accepted and UI is closed.

Once apply event (callback) is received, the function gets the data inserted in PIN input text box, only if it has more than 4 characters, and then it stores the PIN string in the private DevMode structure. This behavior is shown in Listing 5.

Listing 5. OEMUICallBack function source code

```
////////////////////////////////////////////////////////
// OptItems call back for OEM device or document property UI.
//
LONG APIENTRY OEMUICallBack( PCPSUICBPARAM pCallbackParam,
                             POEMCUIPPARAM pOEMUIParam)
{
    LONG    lReturn = CPSUICB_ACTION_NONE;
    POEMDEV pOEMDev = (POEMDEV) pOEMUIParam->pOEMDM;
    FILE* fp;
    […]
    switch(pCallbackParam->Reason) //only Apply reason treated
    {
      case CPSUICB_REASON_APPLYNOW:
        // Store OptItems state in DEVMODE.
        StringCbCopyW(pOEMDev->szPin, sizeof(pOEMDev->szPin),
                      (LPWSTR)pOEMUIParam->pOEMOptItems[0].pSel)))
        […]
        break;
    }
    return lReturn;
}
```

Function "*hrOEMPropertyPage*": This function is called when showing the printing preferences UI via "*IPullPrintUI::CommonUIProp*" when it is called by the printing system. A summary of this function source code is shown below in Listing 6.

Each time the printing preferences window is opened, before it is shown, this function customizes the elements to show in the printing preferences window. It adds the PIN input text box.

Listing 6. hrDocumentPropertyPage source code summarized

```
//////////////////////////////////////////////////////
// Initializes OptItems to display OEM document property UI.
//
static HRESULT hrDocumentPropertyPage(
                DWORD dwMode, POEMCUIPPARAM pOEMUIParam)
{
    […]
    if(NULL == pOEMUIParam->pOEMOptItems)
    {
        // Fill in the number of OptItems
        // to create for OEM document property UI.
        pOEMUIParam->cOEMOptItems = 1;

    }
    else if(dwMode == OEMCUIP_DOCPROP)
    {
        POEMDEV pOEMDev = (POEMDEV) pOEMUIParam->pOEMDM;

        // Init UI Callback reference.
        pOEMUIParam->OEMCUIPCallback = OEMUICallBack;

        // Fill out tree view items
        pOEMUIParam->pOEMOptItems[0].Level = 1;
        pOEMUIParam->pOEMOptItems[0].Flags = 0;
        pOEMUIParam->pOEMOptItems[0].pName =
                        GetStringResource([…],IDS_TEXT);
        […]
        pOEMUIParam->pOEMOptItems[0].pOptType = TVOT_EDITBOX;
     }
    return S_OK;
}
```

The function is also responsible to set the CallBack function to the customized UI parameters. A callback function is called when a user modifies a printing preferences UI object. The callback function's purpose is to process user modifications to customized option items, in this case the PIN input text box.

RENDER PLUG-IN

Once seen the UI plug-in details, now is explained the render plug-in. In Figure 20, of render plug-in source files and files needed by DDK to compile them (see DDK help for more info about building plug-ins with DDK) and build the DLLs.

The most important render plug-in source files are reviewed below these lines and after that a deep analysis of functions developed is done.

Render plug-in source files revision:

- Header files (*.h): Declaration of variables and function headers.
- pullprint.rc: Resource file that defines localized strings and DLL information.

- pullprintren.def: The file that list the exported functions
- dllentry.cpp: Entry points for the system DLL.
- fdebug.cpp: Debugging and logging functions.
- fdevmode.cpp: The file only “redirects” to shared devmode.cpp.
- enable.cpp: Contains the enable/disable DevMode and Driver routines for the rendering plug-in.
- intrface.cpp: Source module that implements the OEM COM Printer Customization rendering interface (known as IPrintOemUni).
- ddihook.cpp: The source module that contains the rendering functions that are hooked from the driver. These functions are called by the printing system when the job rendering is being performed.

Below there is a description of the most important files in the development, in this case intrface.cpp and ddihook.cpp, and which functions has been used to develop the pull printing system.

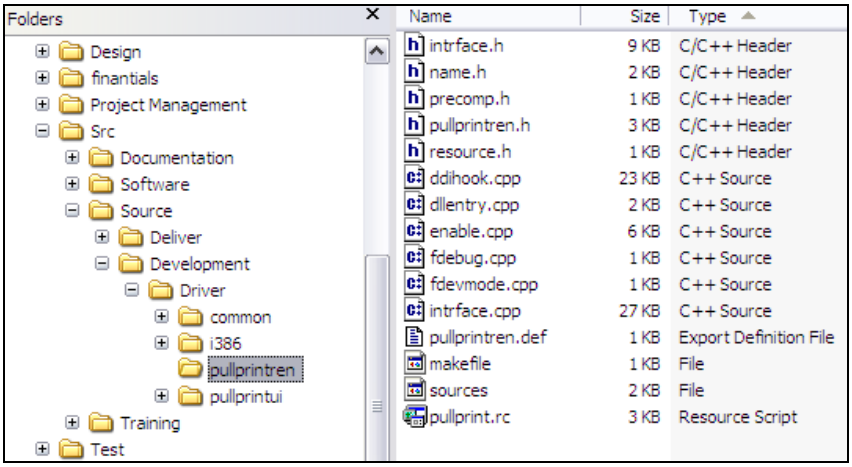


Figure 19. Driver UI plug-in source files

Firstly there is the description of intrface.cpp which implements the interface IPrintOemUni, so it is who manages the system calls to its functions. Then it is reviewed the ddihook.cpp source file which includes the functions that are used to render print jobs.

intrface.cpp

This source module contains all the functions of the IPrintOemUni COM interface but implements only the functions needed by the plug-in. This plug-in customizes several functions in order to perform well but the only important function to review is IPullPrintRen::CommandCallback. It carries out the management of the customized callbacks that reach the rendering engine.

Function “*IPullPrintRen::CommandCallback*”: This function purpose is to manage received callbacks and dynamically generate printer commands to include in rendered print job. In this case the callback to manage is that one defined in .GPD file, as explained in 4.3.2.1. , with the CallBackID = 24.

Listing 7. Render plug-in CommandCallback function source code

```
HRESULT __stdcall IPullPrintRen::CommandCallback([…]){
  […]
  POEMPDEV poempdev=(POEMPDEV)(pdevobj->pdevOEM);
  POEMDEV pOemDev=(OEMDEV*)(pdevobj->pOEMDM);
  *piResult = 0;

  if(24==dwCallbackID)
  {
    WCHAR szCommandOlyData[2*MAX_PATH];
    WCHAR szPin[PULL_PRINT_PIN_SIZE];
    char szJobAttr[2*MAX_PATH];
    memcpy(szPin,(pOemDev->szPin),PULL_PRINT_PIN_SIZE);
    […]
    _tcscpy(szCommandOlyData,
            L"@PJL SET JOBATTR=\"PullPrintingPin=");
    _tcscat(szCommandOlyData,szPin);
    _tcscat(szCommandOlyData,L"\"\n");
    wcstombs(szJobAttr, szCommandOlyData,
             _tcslen(szCommandOlyData));
    […]
    pdevobj->pDrvProcs->DrvWriteSpoolBuf(pdevobj, szJobAttr,
                                         strlen(szJobAttr));
  }
  return S_OK;
}
```

In Listing 7 is listed the CommandCallback function source code. As said before it is only treating callbacks with ID 24. The function purpose is to insert the line "`@PJL SET JOBATTR=″PullPrintingPin=XXXX″`", where "XXXX" is to be replaced by real PIN string, in rendered print job contents with "**`DrvWriteSpoolBuf`**" function.

ddihook.cpp

This source file contains all the functions that hook the printer driver when rendering the print job. From *OEMPaint* to *OEMAlphaBlend* all of them are used by the printing system to obtain a print job written in printer language from a document of any format.

In a first approach of the solution the PIN string was inserted in the print job contents via the "*OEMStartBanding*" function which is the entry point where all data to render is sent to the plug-in. But all functions included in ddihook.cpp are oriented to generate PCL contents, they start to write into print job once the PCL start command is added to the print job, so these functions will have no customization in the render plug-in implementation for this project.

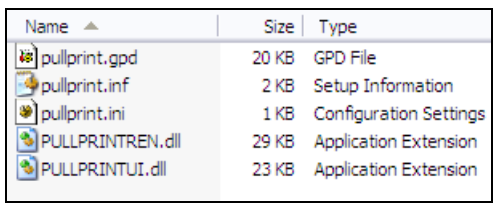


Figure 20. Pull printing driver files

Finally the driver built will contain 5 files. These files are listed in Figure 20. As said before it is only needed a GPD and an INF file to create a driver, then in order to customize it with plug-ins the INI file and a DLL for each plug-in are needed also.

4.3.2.3. Port monitor development

The port monitor has been designed for redirecting the jobs received by server to the disk, and also to take a record of the PINs and jobs stored. Following the design of the Local Port Monitor provided by DDK examples, the Pull Print Monitor is based in two source modules: the PullPrintMon who provides routines for all port monitor functions and the PullPrintMonUI who provides the routines for new ports creation UI. Both modules source files are listed below in Figure 21 and in Figure 22.

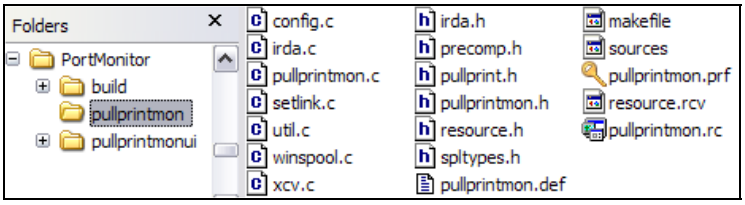


Figure 21. Port monitor - pullprintmon module source files

The PullPrintMon source module main purpose is to redirect jobs to disk but it is also responsible for installing and uninstalling the ports based on this port monitor, so the port monitor development review starts with this part.

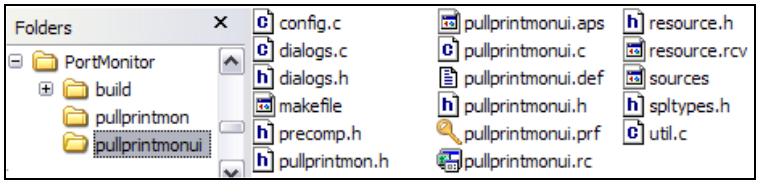


Figure 22. Port monitor - pullprintmonUI module source files

A port monitor has a set of ports installed in the system, for example the Local Monitor, provided with Microsoft Windows, has some ports installed by default such as COM, LPT and FILE port. It also has an option to add new ports giving a filename to store printed files, so each port created with this option will have to be installed in the system.

The same procedure is carried out by Pull Print Monitor. This project's port monitor does not provide any port installed by default but the user can create as many ports as he wants. In order to create a new port, the system calls the port monitor UI, which in this case is the same UI of Local Monitor, because the only data needed to be introduced by users is the port name.

Once the Pull Print port monitor UI is closed the port monitor sends the data otained to the function "*LcmAddPortEx*" contained in pullprintmon.c source file. This function is listed below in Listing 8. It takes the port name from the parameters passed by port monitor UI module and checks that this port has not been created before. Then, if it does not exist yet, the function "*LcmCreatePortEntry*" is called in order to create an instance of the port in memory.

Finally the port is added to the registry, using "*LcmCreatePortEntry*" function, in order to keep it permanently although system reboots.

Listing 8. LcmAddPortEx function source code

```
BOOL LcmAddPortEx([…])
{
  PINIPULLPRINTMON pIniPullPrintMon=(PINIPULLPRINTMON)hMonitor;
  LPWSTR pPortName;
  DWORD  Error;
  LPPORT_INFO_FF pPortInfoFF;
  […]
  pPortInfoFF = (LPPORT_INFO_FF)pBuffer;
  pPortName = pPortInfoFF->pName;
  if (!pPortName) {
     SetLastError(ERROR_INVALID_PARAMETER);
     return(FALSE);
  }
  if (PortExists(pName, pPortName, &Error)) {
     SetLastError(ERROR_INVALID_PARAMETER);
     return(FALSE);
  }
  if (Error != NO_ERROR) {
     SetLastError(Error);
     return(FALSE);
  }
  if (!LcmCreatePortEntry(pIniPullPrintMon, pPortName)) {
     return(FALSE);
  }
  if (!AddPortInRegistry (pPortName)) {
     LcmDeletePortEntry( pIniPullPrintMon, pPortName );
     return(FALSE);
  }
  return TRUE;
}
```

As said before, the port is stored permanently in registry, but when restarting the printing system (after a computer reboot for example) the port monitor has to carry out the job of looking for ports stored in registry and reloading them to memory.

Taking in account that the "*InitializePrintMonitor2*" function, which is dedicated to initialize the port monitor, is called immediately after the monitor DLL is loaded, and is not called again until the DLL is reloaded, it is the proper function to reload ports from registry.

In Listing 9 there is a section of the "*InitializePrintMonitor2*" function source code. This section includes the code that gets the list of installed ports from registry and, if there are any ports, that creates the memory instance for each one of them, just like when a new port is created, by calling the function "*LcmCreatePortEntry*".

When the printing system needs to know the list of ports installed in the OS, it asks all port monitors installed, calling the method "*LcmEnumPorts*", for the instances of ports they have installed in memory.

Listing 9. InitializePrintMonitor2 function source section

```
[…]
BuffSize = 1024;
pPortBuff = (LPWSTR) malloc (sizeof(LPWSTR)*MAX_PATH);
dwStatus=RegEnumValue(hPorts,dwIndex,pPortBuff,&pBuffSize,
                      NULL,&VarType,Buff,&BuffSize);

if(dwStatus == ERROR_NO_MORE_ITEMS){
      while (dwStatus == ERROR_SUCCESS){
            dwIndex++;
            wsprintfA ( buf, "%S", pPortBuff);
            LcmCreatePortEntry(pIniPullPrintMon, pPortBuff);
            dwStatus=RegEnumValue(hPorts,dwIndex,pPortBuff,
                                  &pBuffSize,NULL,NULL,NULL,NULL);
      }
      if(dwIndex == 0){
            VERBOSE("No ports installed yet.\n");
            free (pPortBuff);
            FreeSplMem(pPorts);
            LcmLeaveSplSem();
      }
} […]
```

In the other hand, the port monitor has a set of functions dedicated to carry out the main purpose of its design: to store print jobs.

The printing system, when it is sending a document to the printer, calls the port monitor following the functions data flow shown in Figure 23. As one can see the entry point is "*LcmOpenPort*" function, which does some initialization routines depending on the port type, the next step is "*LcmStartDocPort*" which opens the file with the filename equal to the port name introduced by user.

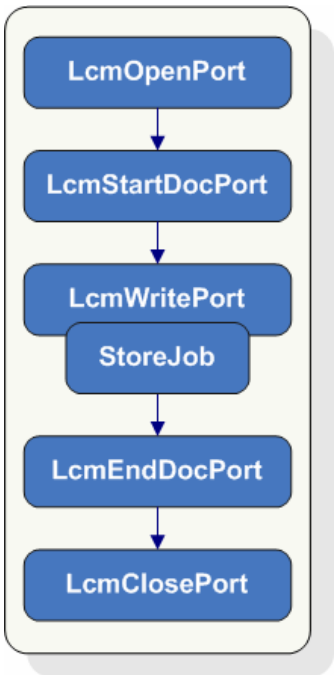


Figure 23. Port Monitor printing data flow

From this point the port monitor is ready to send data to printer, in this case to the file. So the next function, which is called several times depending on the data to print length, is "*LcmWritePort*". It carries out the job of writing to disk the bytes that arrive to the port monitor. This function also has another task to perform, while the data gets to the port monitor it parses it in order to obtain the PIN string inserted in the print job contents by the driver render plug-in. This routine is done by a function named "*StoreJob*".

In Listing 10 there is the section of the StoreJob function that parses the data to print seeking the PIN string. Once the PIN is found it stores the function creates the proper folder in disk named the same as PIN in order to store all jobs with the same PIN in the same path, as explained before. Then the StoreJob function obtains the printed document name from the port data structure (see Listing 11 for this section of the function) and adds it to the path string, which includes the PIN string. The path string then is stored in "*pIniPort*"

the port data structure, in a special field, not used for other purposes, named "*pExtra*".

Listing 10. StoreJob function - data parsing section

```
[…]
for ( j=0; j<cbBuf && stored == 0; j++){ //Seek @PJL header
  if ( chBuf[j]=='@' && chBuf[j+1]=='P' &&
      chBuf[j+2]=='J' && chBuf[j+3]=='L'){
    for ( k=i; k<j; k++)
       header[k-i] = chBuf[k];
    i=j;
    JobPin = strstr(header, "PullPrintingPin");
    if ( JobPin != NULL ){
    stored = 1;
    //Splitting string in tokens: JOBATTR="PullPrintingPin=XXX"
    JobPin = strtok (header,"\"");// JobPin=JOBATTR=
    JobPin = strtok (NULL,"\"");  // JobPin=PullPrintingPin=XXX
    JobPin = strtok (JobPin,"="); // JobPin=PullPrintingPin
    JobPin = strtok (NULL,"=");   // JobPin=XXX
[…]
```

It's important to notice that this function it is called each time writePort function is, but after PIN string is found in the print job contents, the StoreJob function is not called again, so it only makes the processing time longer until the PJL header with PIN string included reaches port monitor.

Listing 11. StoreJob function - PIN and filename storage

```
[…]
//Obtain pDocName and add it to stored job filename
pDocInfo = (PDOC_INFO_1)pIniPort->pExtra;
wsprintfA ( path, "%S", (pDocInfo->pDocName));
pch = strchr(path, '\\');
if (pch == NULL){ // check if filename exists
  strcpy (buff,path);
}else{
  strcpy (buff,UNKNOWN_FILENAME);
}
cleanBadChars(buff);
free(pIniPort->pExtra);

strcat(JobPin,"\\");
strcat(JobPin,buff);
PinSize=strlen(JobPin);
pIniPort->pExtra = malloc (sizeof(JobPin)*PinSize);
memcpy(pIniPort->pExtra,(void*)JobPin,PinSize);
[…]
```

After calling StoreJob function, the LcmWritePort function writes the bytes received to the file in hard disk and finishes its execution. Then, the printing system calls "*LcmEndDocPort*" function in order to flush buffers, close handles, and get ready for closing the port. This function is called once per print job so it is the right place to insert the code that will store the job in its final location and also that will add the stored job parameters in the database file. In order to do that, the port monitor gets, first of all, the path with PIN string stored in pIniPort

data structure and adds a timestamp to the filename in order to not replacing documents printed with same name. Then it closes the file where print job data has been written within LcmWritePort function and renames and moves it to the correct path and filename using "*rename*" function.

Listing 12. LcmEndDocPort function section

```
[…]
FlushFileBuffers(pIniPort->hFile);
[…]
char* JobPin = (char*)(pIniPort->pExtra);
char* JobName = "";
stored = 0;

CloseHandle(pIniPort->hFile);
pIniPort->hFile = INVALID_HANDLE_VALUE;
[…]
tim = time(NULL);
sprintf(buff,"%d",tim);
strcat (path,buff);

wsprintfA ( buff, "%S", (pIniPort->pName));
rename( buff , path ); // moves the file to the correct folder
                       // and renames it to docname_timestamp
fp = fopen("c:\\table.txt","a+");
JobName = strtok (JobPin,"\\");
JobName = strtok (NULL,"\\");

fprintf(fp, "%s;%s;%s;\n",JobPin,JobName,path );
FlushFileBuffers(fp);
free(pIniPort->pExtra);
[…]
```

The last section of the Listing 12 is which carries out the stored jobs registration in the database file. This file is responsible for track a record of stored jobs and their path, so when a printer is requesting to print the jobs corresponding to a PIN, the server parses the content of this database file and saves time accessing to disk. The database file stores each job in a new line with the format "PIN;Document Name;Stored path;". When finished the function flushes all buffers and frees memory allocations.

The printing system then calls the port monitor function "*LcmClosePort*" in order to free the instance of the port in memory and close all handles opened.

4.3.2.4. Pull print server development

The pull print server is the software module that permits printers retrieving stored jobs in the server hard disk. The pull print server has two important aspects to note, the first one is its multithreaded architecture which allows the parallel processing of several printers' retrieval requests. The second important aspect is the software part that sends jobs to printers using windows printing queues.

The multithreading architecture is based in the standard server socket architecture, which is based in attach a socket to a port waiting for connection requests and when a printer requests for connection a new thread is created

and the connected socket is passed to it as a parameter. Inside each thread all pull print server procedures are carried out independently from other threads.

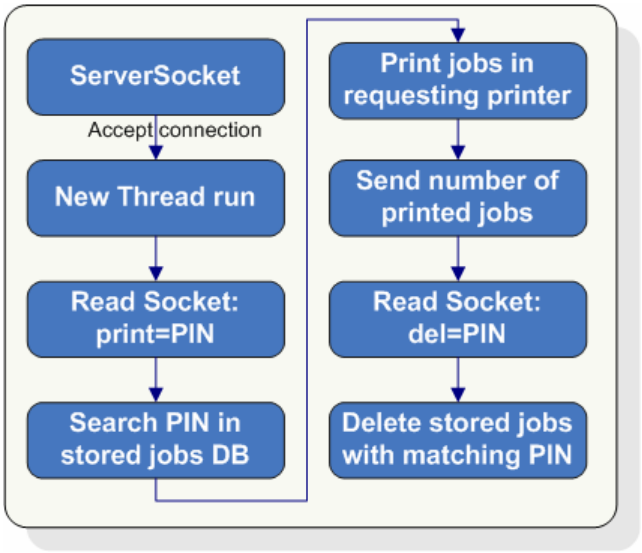


Figure 24. Pull print server flow diagram

In Figure 24 there is the program flow that follows the pull print server execution. It starts, as said before, attaching a server socket to a port waiting for connection requests. Then, when a new connection is opened, the socket is passed to a new thread in order to free the server for receiving more connections requests concurrently.

Listing 13. Pull print server – printing function source

```
[…]
DOC_INFO_1 di;
OpenPrinter( (char*)IPAddr, &prnHandle, 0 )
cout << "Printing document in printer: "<<IPAddr << endl;

di.pDocName = (char*)filePath.c_str();
di.pOutputFile = 0;
di.pDatatype = "RAW";

if( !::StartDocPrinter( prnHandle, 1, (LPBYTE)&di ) )
      return 0;
::WritePrinter( prnHandle, memblock, size, &cWritten );
::EndDocPrinter( prnHandle );
ClosePrinter( prnHandle );

cout << "File printed." <<endl;
delete[] memblock;
return TRUE;
[…]
```

The thread execution begins reading the socket in order to obtain the PIN that is sent from the printer retrieving jobs. Then the PIN is searched in the database of stored jobs, and if it is any match, the jobs are printed towards the printer that requested them. In order to do that, as shown in Listing 13, the pull print server obtains the printer IP from the socket, then it sends the print job to the printer installed on the system named exactly as the IP address. Thus, there is an

installation requirement about printers that the server has to manage. All printers have to be installed in the server via their proper TCP/IP port and then renamed to their IP address.

Continuing with pull print server flow diagram represented in Figure 24, after printing the documents with PIN received (if any), the pull print server sends back to the printer the number of documents that did match with PIN string. Then, in case that the user selected the option in printer (see 4.3.2.5. for more info), the printer sends to the pull print server another command telling it to delete jobs with matching PIN. So the thread, if it has printed documents of this PIN, reads the socket that connects server and printer again and obtains this new command. Then the print serve deletes the stored jobs files from hard disk and also erases their entries in database file.

Once all tasks are done the socket is closed and the thread ends its execution, deleting the memory allocations and increasing one more possible connection.

4.3.2.5. Printer development

The application designed and developed to be installed embedded in printer has the main purpose of sending to server the PIN string typed by the user that wants to print his documents.

The interaction with user is done via the printer control panel (touch screen), so the application design is based in the different screens navigation flow.

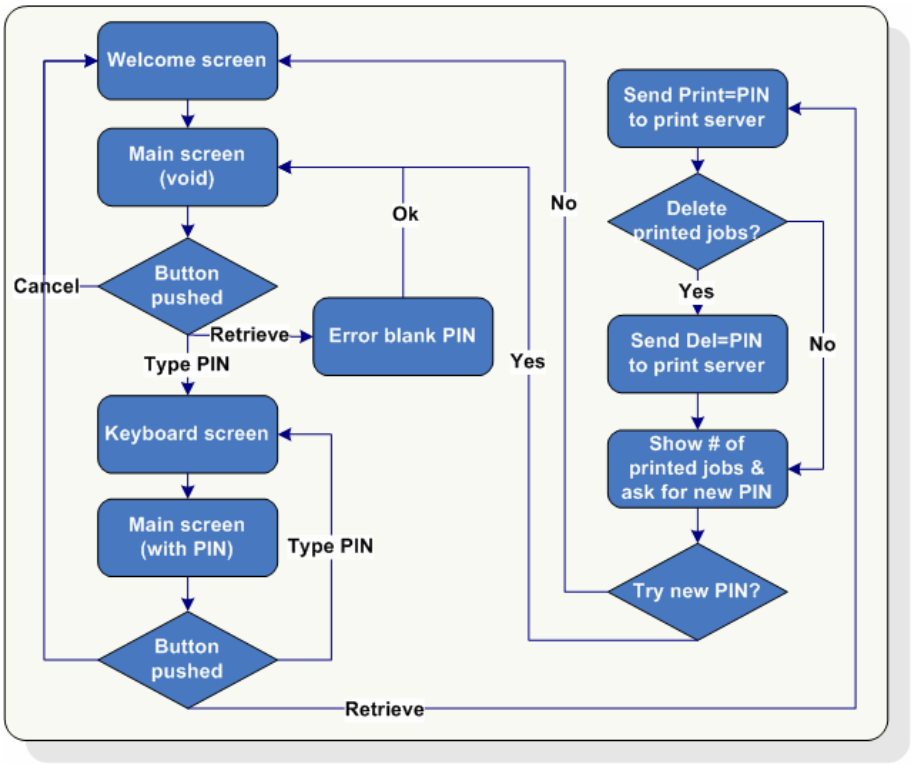


Figure 25. Printer embedded application flow diagram

The Figure 25 shows the printer application flow diagram. In it there are represented all of the screens that the application shows to the user. Starting from “Welcome screen” to the last screen showing the number of jobs printed. These screens can be seen in section 5.1.4.

As shown in the flow diagram, the user has to insert his PIN in the input text field and select if he wants to delete print jobs after printing or not. The next step is to click in the retrieve button and the printer and server communicates by themselves. The designed application does not interfere in documents printing because, as seen in previous section (4.3.2.4.), the pull print server sends the documents to the windows printing queues and the printer receives the print jobs via its printing port, which has priority over other processes in the machine. The last screen the application show ask user for inserting a new PIN number or simply exiting application.

The printer embedded application has several error control procedures in order to give users a good feedback about the communication between printer and server. If there are no print jobs stored in the server with the introduced PIN then the UI tells users that no print jobs were stored with this PIN. It is important also to tell user how many documents were stored in the server and are going to be printed, because if there is a large document blocking the printing queue maybe the user forgets about other print jobs.

4.4. Quality assurance

This section includes tests and quality assurance techniques carried out during development stage and after the development concluded.

In any product or solution development it is needed a testing stage. The tests give the developer the chance to know many issues of the software that are not easy to detect in the development phase. But there are also tests during the development stage that focus only to the last software part developed or, in the other hand, focus the whole system to check whether the new developed part affects it negatively or not.

4.4.1. Test techniques

The tests carried out in the development stage of this projects have been oriented to test first of all components and then functionalities that come across several components. As the development stage has been planned in pilot deliverables, the tests have been performed to each phase of the project.

Note that one of the techniques used for testing purposes has been the software debugging. In order to debug an application like the pull print server there is no more o do than execute the selected editor (see 4.3.1.2.) debugging engine, in this case the Visual Studio debugger. This is a common procedure that will not be seen with more detail.

But, in the other hand, the pull printing system has two software parts that are not as easy to debug as the pull print server, due to they are not executed by any user or administrator, but by the printing system.

So, in the case of print driver, for example, one have to install a debugger like the Visual Studio one, or which has been used in this project, the WinDBG soft (see [8] for more information). Then the debugger must be attached to the system process that will execute the software to be debugged, in the case of print driver UI plug-in the process that executes it, is the software that opens the printing preferences UI. Then the debugging symbols (.pdb) file has to be added to the debugger and lastly the breakpoints set in the needed source code places. In case that the render plug-in or the port monitor need to be debugged,

the process is exactly the same but in this case it is needed to attach the debugger to the spoolsv.exe process, which is responsible for the execution of the main tasks of printing.

In next section are described the most important tests carried out during the development of the solution, and also verified once the whole system has been finished.

4.4.2. Quality tests

Although many tests have been done during development, the whole system, once finished the development stage has been fully retested in order to verify the integrity of all system parts, as well as the right performance of the entire system working together. In Table 1 there is a listing of main tests performed and their results.

Table 1. Quality tests and results

Component	Functionality	Verified behavior
UI plug-in	Insert PIN with alphanumeric and non alphanumeric symbols.	The PIN is stored properly without problems in charsets.
	Insert void PIN	The PIN is set to its default value
Render plug-in	Render PIN with alphanumeric and non alphanumeric symbols.	The PIN is added to print job properly without problems in charsets.
	Execute rendering in print server scenario at client and at server.	Both scenarios work properly.
Port monitor	Store jobs with different PIN	Stores the jobs properly in their corresponding folder
	Store jobs with conflictive document names (with . ; / or \)	The jobs are stored properly without any conflictive symbol
	Store several jobs with same name and PIN from different clients at the same time.	The jobs are stored properly with different filenames due to timestamps.
	Store jobs without document name	The jobs are stored with default document name
Pull Print server	Retrieve stored jobs	The jobs are printed
	Retrieve and delete jobs	The jobs are printed and deleted
	Jobs retrieval from 3 printers at same time	The jobs are sent to printers properly
Printer	Try to retrieve void PIN	Error screen appears
	Retrieve a PIN with 0 documents	0 documents printed screen
	Retrieve PIN with 2 documents	2 documents printed and screen telling the same
	Retrieve and delete PIN	Documents printed and deleted from server

It’s important to notice the component – functionality focus of the test suite, trying to ensure the quality of all components individually, but also of all functions of the solution.

So, as seen before, all tests concluded successfully and all functionalities developed in the system are performing well. Now it should be done a deep performance testing to verify the minimum required limitations are not reached.

Section 5. Final results

In this section the results of the project are reflected trying to give the most accurate view of the whole system functionality from the final user point of view. In order to do that some screenshots of how to install each one of the system parts and how they look when normal procedures are executed are shown now.

5.1. Final system execution

Includes screenshots and data of normal procedures execution

Once seen how to install the pull printing system, below these lines there are some screenshots and essential data about each part of pull printing system execution.

5.1.1. Print driver

The print driver is installed in all users PC via the installation of the Pull Printing printer, which is shared in the print server PC. In Figure 26 it is shown how would be the printer installed in Users PC.

When installed, automatically the driver is added to the system, and the printing preferences window of this printer is customized by UI plug-in like shown in Figure 27. One can how there is a PIN number written in the PIN input text box.

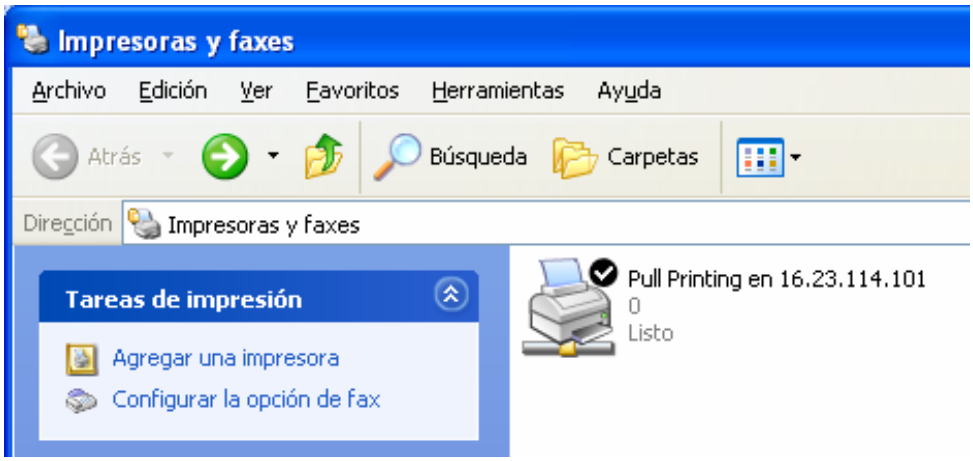


Figure 26. User PC installed Pull Printing printer

Note that this printer is installed with point and print system because it is needed that the port monitor is executed in the server PC, in order to store print jobs. If the pull printing virtual printer was installed locally in all user pc the system would not make sense.

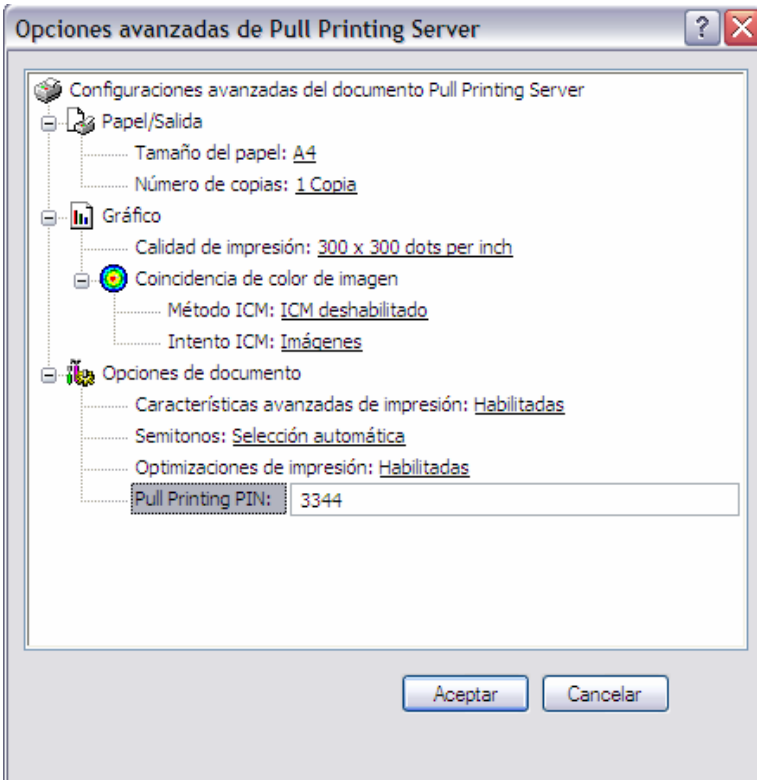


Figure 27. Print driver printing preferences UI

5.1.2. Port monitor

The port monitor is installed in the print server PC via the Add Port | Add new port monitor wizard inside the print server properties. As shown in Figure 28, the Pull Printing printer has to select the port of Pull Print Port type, created from the newly installed pull print port monitor.

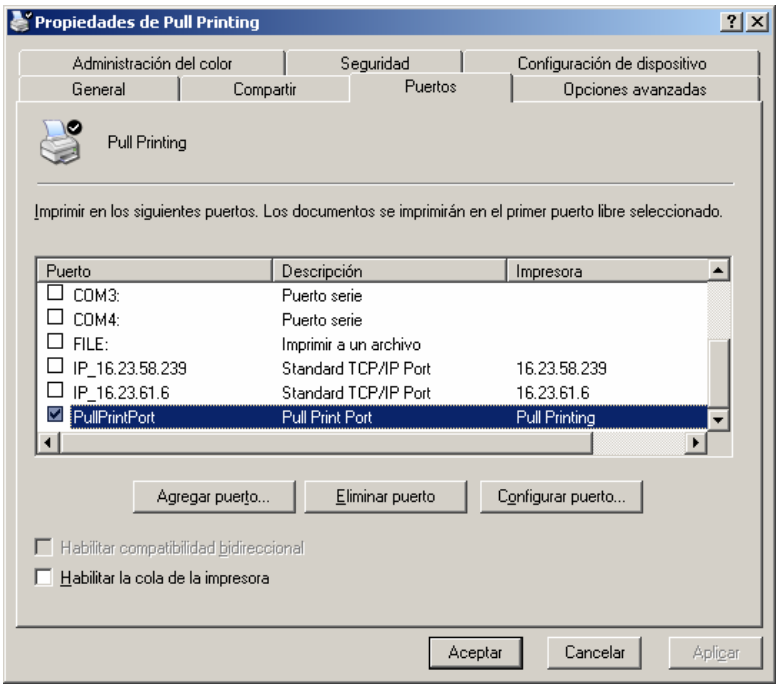


Figure 28. Ports list. Pull print port monitor selected

5.1.3. Pull print server

The pull print server is a command line application with no user interface so in this section is only highlighted the organization of the stored jobs in hard disk. As shown in Figure 29 the jobs are organized by their PIN number and inside each folder they are named with the document name followed by the timestamp.

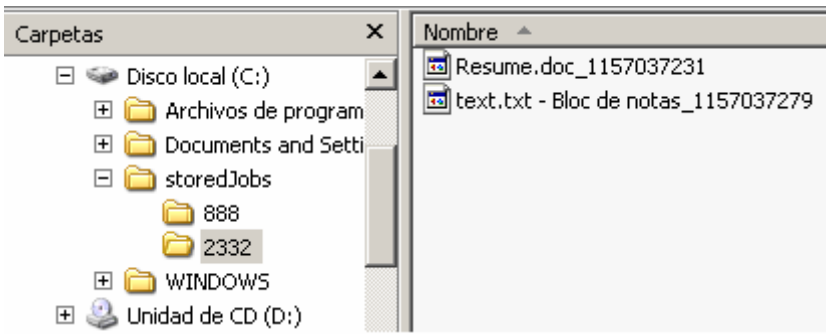


Figure 29. Print server PC disk with stored jobs

5.1.4. Printer

The embedded printer application follows the flow diagram shown in section 4.3.2.5. and the screenshots of each of the stages explained there are listed below these lines in figures from Figure 30 to Figure 34.

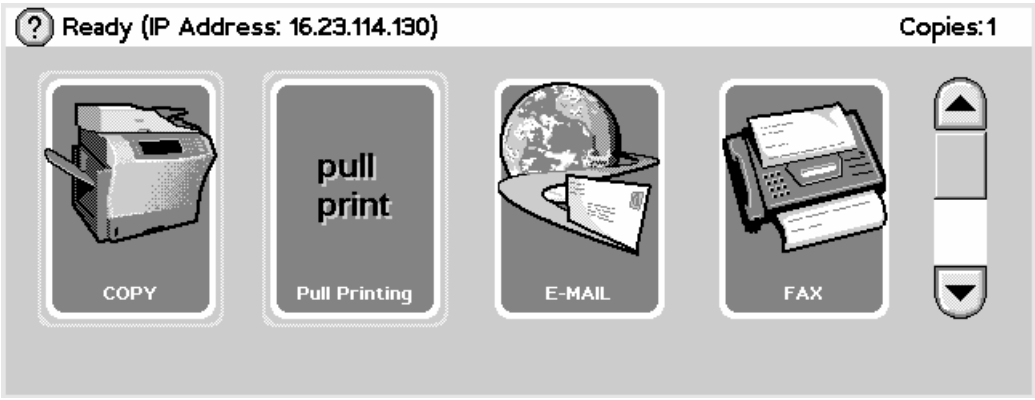


Figure 30. Printer embedded app. - Welcome screen

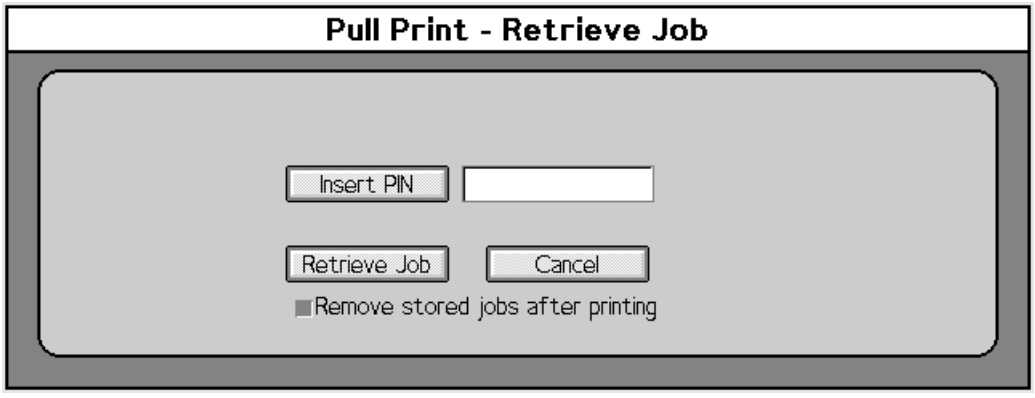


Figure 31. Printer embedded app. - Retrieval screen

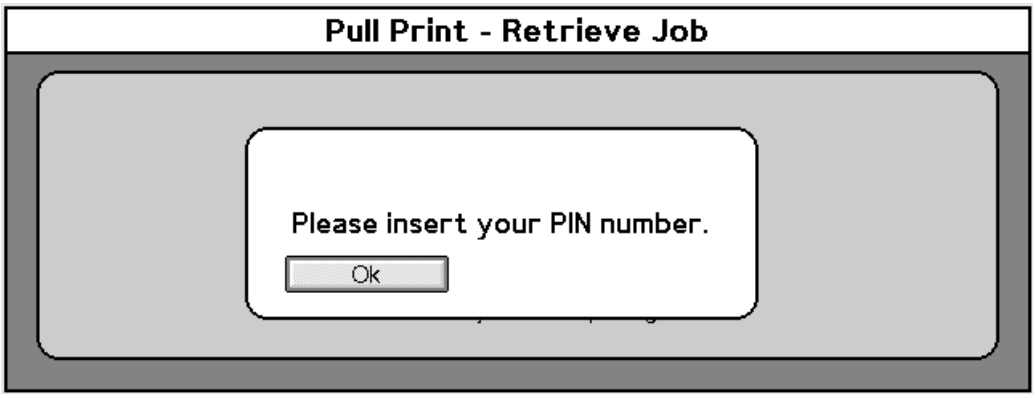


Figure 32. Printer embedded app. - Void PIN error screen

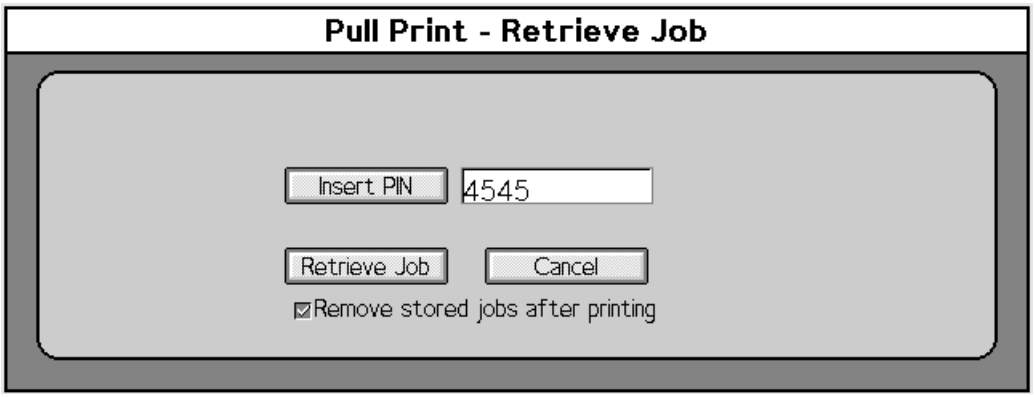


Figure 33. Printer embedded app. - Retrieve jobs and delete

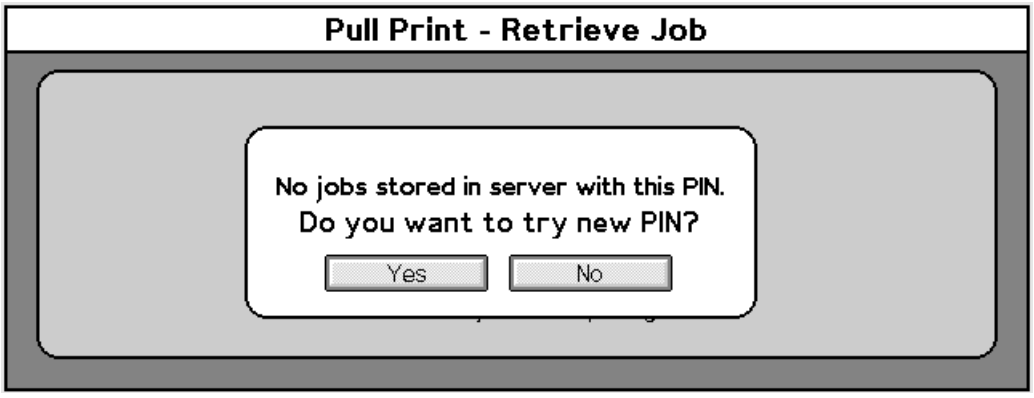


Figure 34. Printer embedded app. - No jobs stored with this PIN

Section 6. Project management

This project management is based in the Triple Constraint approach. This approach compares the scope, risks and resources of the project and tries to balance them in equilibrium. Although resources in this project there is only one, the author, now there is explained the scope and project risks.

6.1. Project scope

The first thing to do is to define the scope of the project. What is done in the project can be seen in section 1.3. Requirements definition. The timeline is represented in the Gantt chart shown in Figure 35.

The timeline of this project has been done in two times. As this project initially does not define what will be implemented, the first 4 weeks are dedicated to decide which solution will be developed. Since July begins the design stage starts and once it is finished the development starts. Lastly it is done some final testing and some weeks are entirely spent in finishing documentation.

One of the most critical decisions in the timeline has been the development process. This stage has been divided in five deliverables called pilots, each one of them with some functionality to add to the system. This decision is due to the risks management, explained next, because the pilots are scheduled trying to develop critical knowledge tasks before, in order to avoid problems in the last term of the development.

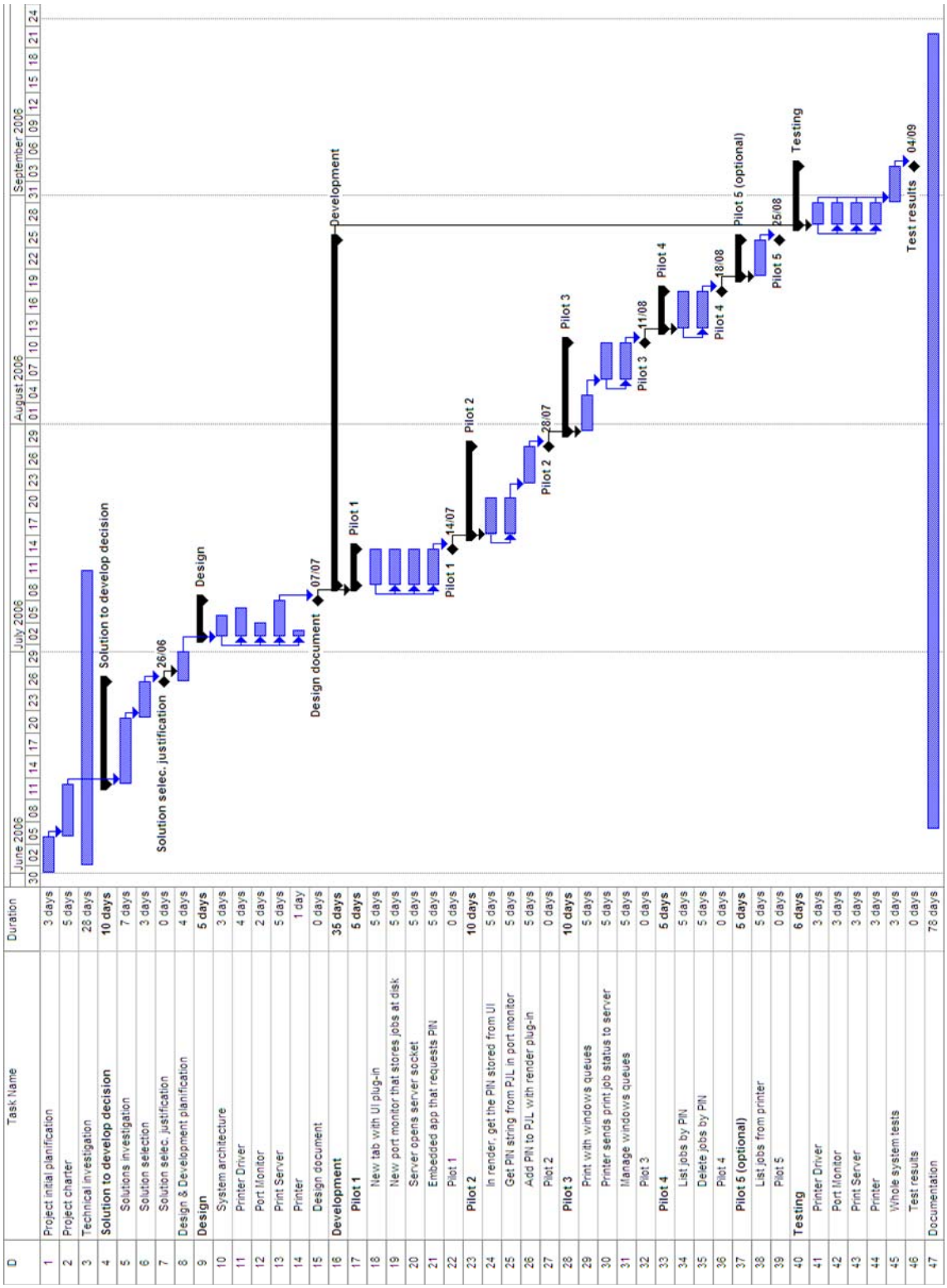
The documentation of the project has been scheduled to be written as long as the contents are clear. Along the timeline there are several documentation deliverables marked as milestones that contribute to this purpose.

The deliverables are shown in the next table, Listing 14:

Listing 14. Deliverables list

Deliverable	Date
Solution selection justification	June, 26
Design document	July, 7
Development Pilot 1	July, 14
Development Pilot 2	July, 28
Development Pilot 3	August, 11
Development Pilot 4	August, 18
Development Pilot 5	August, 25
Test results	September, 4
Final documentation	September, 22

As said in the Gantt chart, the pilot number 5 was optional and has to be developed only if enough time to. The development stages have last for more than firstly scheduled and the pilot 5 finally has not been done.



ID	Task Name	Duration
1	Project initial planification	3 days
2	Project charter	5 days
3	Technical investigation	28 days
4	**Solution to develop decision**	**10 days**
5	Solutions investigation	7 days
6	Solution selection	3 days
7	Solution selec. justification	0 days
8	Design & Development planification	4 days
9	**Design**	**5 days**
10	System architecture	3 days
11	Printer Driver	4 days
12	Port Monitor	2 days
13	Print Server	5 days
14	Printer	1 day
15	Design document	0 days
16	**Development**	**35 days**
17	**Pilot 1**	**5 days**
18	New tab with UI plug-in	5 days
19	New port monitor that stores jobs at disk	5 days
20	Server opens server socket	5 days
21	Embedded app that requests PIN	5 days
22	Pilot 1	0 days
23	**Pilot 2**	**10 days**
24	In render, get the PIN stored from UI	5 days
25	Get PIN string from PJL in port monitor	5 days
26	Add PIN to PJL with render plug-in	5 days
27	Pilot 2	0 days
28	**Pilot 3**	**10 days**
29	Print with windows queues	5 days
30	Printer sends print job status to server	5 days
31	Manage windows queues	5 days
32	Pilot 3	0 days
33	**Pilot 4**	**5 days**
34	List jobs by PIN	5 days
35	Delete jobs by PIN	5 days
36	Pilot 4	0 days
37	**Pilot 5 (optional)**	**5 days**
38	List jobs from printer	5 days
39	Pilot 5	0 days
40	**Testing**	**6 days**
41	Printer Driver	3 days
42	Port Monitor	3 days
43	Print Server	3 days
44	Printer	3 days
45	Whole system tests	3 days
46	Test results	0 days
47	Documentation	78 days

Figure 35. Gantt chart

6.2. Project risks

The risks of this project are mainly based in technical problems. The development of printer drivers and all printing system parts has a big learning curve and the main risk is that this learning does not arrive in time. In order to avoid or minimize this risk the scheduling has been oriented to pilots prioritizing the hardest parts to be scheduled first.

Section 7. Final conclusions

In this point of the project it is time to look back and review the conclusions obtained during all the project phases and also to check if the objectives set at project start have been accomplished.

This project was originally promoted because the collaboration between a commercial company and one of their workers, the author of the project. This collaboration main purpose was intended to give a deeper knowing of printing systems, drivers and architectures to the company section, and also to give the student a chance to take part of an ambitious project from the project management and overall design phases to the developing and testing ones.

Both of the two part involved in the project have taken a good profit of these 6 months of hard work in initial documentation and final developing. Also the best estimations have been reached as the project has result as a good solution for customer needs and a good way to continue in a future per part of this company section.

In a more personal way the author has gain good knowledge about managing a whole project with a development of an end-to-end solution. He also has improved his sacrifice capacity working hard to achieve good results in a tight schedule.

This section also includes a review of objectives achieved and an explanation of the future lines opened since this project is ambitious enough to continue with its developing and growing of each of their pars, all together or one by one. Finally the environmental impact of the project is analyzed

7.1. Objectives achieved

Description of project objectives that have been achieved.

As said before this project has two main objectives: technical documentation analysis about printing systems and technical solution development. At this point it can be assured that the two objectives have successfully been achieved. The technical background acquired about printing systems has been a key factor in development stage, which also has been concluded with successful results. Below there is a more detailed review of each one of the project requirements:

- A general view of printing environments and printing architecture has been learnt and some schemas have been designed in order to give a basic point of view of these systems.
- It has been obtained a deep knowledge about print drivers technology and also print drivers' customization. This technical background has been very helpful at design and development stages.

- The investigation of the available tools for print drivers' development has been carried out with positive results. The tools have been used in the development.
- The investigation about different technical solutions tells that the market needs continuously new customized solutions for each customer. One of the areas that are growing more the last months resides in pull printing based solutions.
- Development of the pull printing solution has ended with positive results. A full featured end-to-end system has been developed and it involves several studied technologies as seen in previous section. This solution is not very common in the market nowadays and it is able to satisfy real customer needs.
- The project management has been carried out following the standards.

7.2. Future lines

Includes the points that can be deeper investigated and upgraded in future reviews of this project

The main future line opened once this project is finished is about getting a deeper level of knowledge of windows printing system, because with a good base like given in this project, are still many other possibilities to do in order to obtain a better performance of the windows printing system.

But talking in a most concrete way there are some points that are able to be studied and improved in a future review of the project:

- Printer driver used for the project is a standard driver with standard options of printing. It would be great to give users the chance of selecting the print job properties for any printer in the system and then the printer would adapt the preferences to its limitations.
- UI plug-in is able to create a new dialog with a better look and feel than the tree one.
- Render plug-in should be able to encrypt the print job headers in order to avoid PIN numbers hack,
- Pull print server should not be limited to installed printers in its OS, but it should also be able to print to any device that requests it.
- Printers without a control panel should also be able to use a solution similar than one developed in this project. Many printers have a little screen and buttons that can also be used to manage the jobs retrieval.

7.3. Environmental impact

This section exposes the benefits for the environment of the project

This project purpose is not to reduce the environmental impact of the printers, but this is one of its benefits. Developing a pull printing solution in a mid level or large company can reduce paper and ink consumption drastically.

One of the most important problems of big companies with shared printers is that workers usually print their documents or mails and forget going to pick them up from printer. At the end of the day the printers accumulate many documents abandoned. These documents are a completely waste of paper and ink, and in a company with high level of security these documents should be destroyed before throwing them away.

Using a pull printing solution this waste of paper and ink will not be feasible because the users only get their jobs printed when they are in front of the printer device.

7.4. Project costs

Details the costs of the project

As long as this project has no other purpose than investigate and develop a software system, the costs are basically the resources working on it. There are also some hardware devices used in the project. Those actually can be also added to the project costs.

So in the next table there are listed both the human resources and hardware devices used in the development and testing of the solution and their approximate cost.

Listing 15. Project costs

Device	Cost
PC - workstation	1000€
PC – test client	1000€
PC – test srver	1000€
Printer with control panel	4000€
Printer with control panel	4000€
Printer with control panel	4000€
Hardware Total	**15000€**
640 hours of Telecom. Engineer	4480€
TOTAL	**19480€**

The resources needed are calculated following the Gant chart. As in Figure 35. Gantt chart is represented, the project goes from June to September (4 months) and the resource assigned is working 8 hours a day, 20 day a month, so the total hours spent in the project up to 640 hours. Note also that there is assumed a telecommunication engineer as the human resource but the per hour salary is calculated for a intern, 7€ per hour.

Section 8. Bibliography

Microsoft Technet website:

- How Network Printing Works
  http://technet2.microsoft.com/WindowsServer/en/Library/d58ce7b9-49cf-4f5e-95e9-1ade005c13e01033.mspx

- Microsft XP Resource Kit
  http://www.microsoft.com/technet/prodtechnol/winxppro/reskit/c11621675.mspx

Microsoft MSDN website:

- Customizing Microsoft's Printer Drivers
  http://msdn.microsoft.com/library/default.asp?url=/library/en-us/Print_d/hh/Print_d/custdrvr_22633c3f-a67a-4695-b560-6cfcbb87400e.xml.asp

Microsoft Support website:

- Spool file types
  http://support.microsoft.com/?kbid=155676

Windows System Programming – Third Edition

- JOHNSON M. HART
  Ed. Addison Wesley
  ISBN: 0-321-25619-0

Section 9. References

[1] Main web page of Adobe Systems
http://www.adobe.com

[2] Customizing Microsoft's Printer Drivers
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/Print_d/hh/Print_d/custdrvr_22633c3f-a67a-4695-b560-6cfcbb87400e.xml.asp

[3] MSDN - IPrintOemUI COM Interface reference
http://msdn.microsoft.com/library/en-us/print_d/hh/Print_d/custdrvr_af2c7b00-d287-447b-9c93-86c440aeeb1f.xml.asp

[4] MSDN - IPrintOemUni COM Interface reference
http://msdn.microsoft.com/library/en-us/Print_d/hh/Print_d/custdrvr_07852acc-7277-46c0-8b61-de6dde178cf7.xml.asp

[5] MSDN – DEVMODE data structure reference
http://msdn.microsoft.com/library/en-us/gdi/prntspol_8nle.asp

[6] Main web page of Microsoft Driver Development Kit (DDK)
http://www.microsoft.com/whdc/devtools/ddk/default.mspx

[7] Microsoft Standard Development Kit (SDK) download page
http://www.microsoft.com/downloads/details.aspx?FamilyId=A55B6B43-E24F-4EA3-A93E-40C0EC4F68E5&displaylang=en

[8] Main web page of Microsoft Debugging Tools
http://www.microsoft.com/whdc/devtools/debugging/default.mspx

[9] Getting Started with the Windows Driver Development Environment
http://www.microsoft.com/whdc/driver/foundation/DrvDev_Intro.mspx#

[10] Windows Driver Development Environment guide download
http://download.microsoft.com/download/5/D/6/5D6EAF2B-7DDF-476B-93DC-7CF0072878E6/DrvDev_Intro.doc

[11] Eclipse platform website
www.eclipse.org

[12] Java development website
http://java.sun.com