2011

AUTOMATED DISCOVERY AND INSTALLATION OF NETWORK-ATTACHED PERIPHERAL DEVICES

Lawrence Alan Mandel

AUTOMATED DISCOVERY AND INSTALLATION OF NETWORK-ATTACHED PERIPHERAL DEVICES

(Spine title: Automated Discovery and Installation of Peripheral Devices)

(Thesis format: Monograph)

by

Lawrence Alan Mandel

Graduate Program in Computer Science

A thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science

The School of Graduate and Postdoctoral Studies
The University of Western Ontario
London, Ontario, Canada

THE UNIVERSITY OF WESTERN ONTARIO
SCHOOL OF GRADUATE AND POSTDOCTORAL STUDIES

CERTIFICATE OF EXAMINATION

Supervisor	Examiners
Dr. Michael Bauer	Dr. Lu Xiao
	Dr. Mike Katchabaw
	Dr. Hanan Lutfiyya

The thesis by

Lawrence Alan Mandel

entitled:

Automated Discovery and Installation of Network-Attached Peripheral Devices

is accepted in partial fulfillment of the
requirements for the degree of
Master of Science

Date_____

_____
Chair of the Thesis Examination Board
Dr. Olga Veksler

Abstract

Networks today are pervasive and numerous. They are accessed using a variety of client devices such as traditional laptop and desktop computers, phones, tablets, music players, and video game consoles. Networks may contain many categories of services, of which an increasingly common one is the network attached peripheral device. Network attached peripheral devices, such as printers, fax machines, and video projectors, are available to client devices that have installed and configured the associated device driver software. Practically, this means that network attached peripheral devices are hidden from or unavailable to client devices until a user performs the manual discovery of the network attached peripheral device and the installation of the requisite device driver software. This paper presents a system architecture that allows for the automatic discovery and installation of network attached peripheral devices with no user intervention.

Keywords

Network attached peripheral device, Location-based services, Automatic configuration, Automatic installation, Service discovery, Sandbox environment, Printer

Simplicity is the ultimate sophistication.

~Leonardo DaVinci

For my wife, Elana, and my children, Erica and Benjamin.

Acknowledgments

My thesis would not have taken form without the help of some great mentors, friends, and family.

To my thesis supervisor, Professor Michael Bauer, thank you for providing me with the opportunity to do this work, for your patience and guidance in selecting a topic, and more generally for all that you have done to help me see this work through to its conclusion.

To my friend and colleague at IBM, Dr. Arthur Ryman, thank you for your support in my endeavor to further my education through a Masters of Science degree.

To my parents, Marie and Fred Mandel (ז"ל), and my in-laws, Rena and Richard Levy, thank you for building homes that cultivate the spirit of exploration and fun and for teaching me the reward of creating something new.

To my wife, Elana, and children, Erica and Benjamin, thank you for all your love and support, and for making play time really, really fun.

I also want to acknowledge and thank my employer, IBM, for funding my research and supporting my pursuit of higher education.

Table of Contents

List of Figures

List of Listings

Preface

The problem of enabling a client device to use a network attached peripheral device, such as a printer or video projector, is one that I experience almost daily. As an IBM employee with colleagues and customers travelling to the IBM Toronto Lab on a regular basis, I regularly experience the difficulties that these people encounter using our local network attached peripheral devices. The people whom I work with are the technical elite who should be expected to be able to operate these devices quickly and easily. Yet day after day I see time wasted configuring these devices and various workarounds being employed simply to make use of the devices, such as borrowing a machine in order to print.

It is my frustration of witnessing the unsuccessful discovery and installation of these network-attached peripheral devices that I want to alleviate through the work that I have put into this thesis.

document becomes a significant burden for a client device that may only make use of the NAP a single time.

There are two high-level requirements to enable a client device to make use of a NAP:

1. Discover the location of the NAP on the network.
2. Register the NAP with the system and install the device driver software.

Much effort has been spent researching methods to perform service discovery [3-5] as necessary for requirement 1. For requirement 2, each operating system may have a method of registering common types of NAPs. There is also an opportunity to reduce the burden of installing device driver software using de facto and industry standard TCP/IP based communications protocols, such as Socket API, AppSocket, and Internet Printing Protocol [6,7] for printing. However, not all NAPs have the benefit of having standard communications protocols defined. Standard communication protocols are also static and slow to evolve.

The problem in this scenario is not the lack of a method to discover a NAP, register it, or communicate with it, but how to remove from the user the burden of manually performing these steps. This thesis proposes an architecture with unique aspects of service description and a new model of device driver software installation and execution. This architecture supports the automated discovery of NAPs and installation of the required device driver software. This thesis does not investigate new methods of discovery or look to implement a new driver protocol.

Chapter 2 reviews the existing literature for some of the related pervasive computing and location-based services projects in order to extract some basic lessons. Next key existing network service discovery technologies are highlighted. Because the implementation of the architecture is specific for printers, a review of existing printer communications protocols is also conducted.

In Chapter 3, we begin to tackle the problem by discussing the high-level requirements and identifying other software systems from different domains that can contribute ideas for the system architecture.

In Chapter 4, we present a system architecture for the automated discovery of a NAP and installation of any required device driver software. This chapter focuses on quality attributes of the system, the components in the system, and the system communication sequences.

In Chapter 5, we construct a printer-specific implementation of the system architecture presented in Chapter 4 to prove the architecture's viability and demonstrate single-click printing. We review the implementation details including the system requirements and class structure. We then take a look at the prototype system in action.

In Chapter 6, we discuss the results of implementing the prototype. We review issues encountered while developing the prototype and identify other considerations when implementing this type of system.

Chapter 7 concludes this thesis with a summary of the work presented and identifies future work related to this subject.

Chapter 2

2 Literature Review

To start our exploration for a solution that allows NAPs to be automatically discovered and installed, we review related projects in research and production. We then review technologies specific to network service discovery. Finally, we review common network printer protocols to set the stage for the printer implementation that will be constructed in Chapter 5.

2.1 Related projects in research and production

A number of projects in the pervasive computing and location-based services fields have relevance to this thesis. In this section, we review some of the related projects and highlight the lessons that can be applied from them.

2.1.1 Cooltown

Cooltown, a project by Hewlett-Packard that has been around since the late 1990s, is a technology focused on connecting people, places, and things. The architecture of this system is based on web standards including hypertext transfer protocol (HTTP), hypertext markup language (HTML), and extensible markup language (XML). The use of standards is key for this system because there are many different operating systems and platforms that may host services that will interact with the system. The high-level vision for this system is one in which the people, places, and things that are connected communicate with one another in order to provide simplified access to the information that is currently relevant to the user.

This project shares the goal of simplifying user interaction with devices connected to the system, such as NAPs, but does not address the immediate need of eliminating the manual discovery and installation of these devices. Instead it focuses on ensuring that each device is addressable and can communicate using the specific document types described by the Cooltown project [3].

This project demonstrates that web standards such as HTTP and XML or HTML are a viable mechanism for communicating information about NAPs.

2.1.2 JetSend

JetSend is a "media-independent communications protocol" [8]. Developed by Hewlett-Packard, the goal of this protocol is to remove the requirement to install device drivers by having a flexible communications protocol that not only transfers data but also includes a description of the data. JetSend achieves interoperability by describing the data using surfaces, whereby each surface represents a component of the data. The components are assembled in a tree structure.

At its core, JetSend solves the problem of installing device drivers by having a single communications protocol and building a single driver into each operating system. This solution has the drawback of requiring a single communications protocol, thus limiting user options when it comes to data transfer. The use of JetSend may preclude the use of different or more efficient data transfer technologies that may be required for higher quality video, audio, pictures, and ever larger documents. The work on JetSend provides an alternative approach to the work presented in this paper but, as a communications protocol, it can also work alongside the solution presented herein [8].

2.1.3 Universal Plug and Play (UPnP)

Universal Plug and Play (UPnP) is a system developed by Microsoft that connects devices together using common protocols instead of device drivers to facilitate communication. UPnP aims to require zero configuration, which includes automatic

discovery of devices and services. This technology is built on web standards including transmission control protocol over internet protocol (TCP/IP), user datagram protocol over internet protocol (UDP/IP), HTTP, and XML. The use of web standards allows for the flexibility to implement UPnP in any language on any operating system [9].

In a UPnP system, each device is described by an XML document that is exposed via an HTTP server present in each device. This means that each device both describes and advertises its existence on the network.

UPnP provides good lessons for the elimination of device drivers including the use of standard web technologies, such as HTTP, TCP/IP, and XML, and having a service describe and advertise itself on the network. In its use of common protocols UPnP is limited in that in order to be UPnP-compliant a device must use the defined protocols. If defined protocols do not exist, the device cannot participate in the UPnP system.

2.1.4 Jini

"The primary goal of Jini is to enable any service to interact with one another without worrying about drivers, protocols, and operating system compatibility." [10] Jini tolerates unreliable network connectivity and services that may come and go at any time. It provides a mechanism for a service to define itself and advertise its existence, and provides a mechanism for consuming services to locate the service. Jini is an extension of the Java system but can also support non-Java artifacts. Jini's infrastructure is built on remote method invocation (RMI), a method for moving objects between systems in a distributed environment. Trusted classes, as determined by the Java class loader, can be loaded on the client machine, thus providing a way to execute code, such as that needed to interact with a service, on a client machine. In the Jini universe, everything is a service. It eliminates the need for device drivers because the code that is required to interact with the services is downloaded automatically at runtime [10].

Jini requires considerable computing resources and has defined minimum requirements for storage, processing power, and networking capabilities in order for a device to

interact with the system. Jini requires the Java runtime environment (JRE), which in turn requires substantial memory.

With respect to this thesis, the lessons from Jini are the ability of a service to describe itself and advertise its existence and the ability of a system to download and execute trusted code. In Jini's case, this means that there is no need for device drivers. For the system presented in this thesis, downloading and executing device drivers will likely be necessary.

2.1.5 Calypso Concept and Odessa Architecture

The Calypso Concept and Odessa Architecture is a system to enable the use of third-party applications on multi-function peripherals. To solve the installation issue, this system takes an approach where the client device installs the applications locally and the local code can then execute on any NAP [11].

This system requires that the user select and install the applications that they want to use when interacting with a NAP; thus it is subject to the same limitations of the traditional manual discovery and device driver software installation process that this thesis aims to solve. This concept of this system is noteworthy because it is the antithesis of the work presented in this paper; it deploys code from the client to the NAP in an attempt to improve the process of enhancing NAPs.

2.1.6 Aura

Project Aura introduces the concept of a *personal Aura*, which is system that acts as a proxy for the user. This system interacts with and establishes communications with the appropriate resources when the user enters a new physical location. The system shields the user from the complexity of configuration and gives them a single point of access, via their Aura, to local resources connected to the system.

The Aura system operates at a higher level of abstraction than the work presented in this thesis; therefore, despite having a similar overall goal of reducing complexity for the user, it takes a much different approach to its implementation. The Aura system tracks people's movements as opposed to device movements to and from locations. The system saves the user's working state when in one location and has the ability to migrate that working state to a new location. The migration is hardware-independent, so a user is not required to carry hardware with them but rather can have their work follow them virtually from location to location. The user can then pick up their work on a different terminal in a different location without actively transferring their work [12]. The Aura system provides a higher level view of what is possible. The concept can be enriched by the work presented in this thesis because this work will assist client devices in discovering and installing NAPS as they move from one location to another.

2.1.7 Satchel

The Satchel system, developed in the mid 1990s, provides the facility to access documents and perform document-related services, such as printing, faxing, scanning, and e-mailing. This system is based on web standard protocols and requires the use of a web browser. A user can interact with the system via a variety of devices. This system was demonstrated running with a PARC Minder (a device with only two lines of display), a Nokia 9000 Communicator personal digital assistant, and a traditional computer. In a Satchel system, all documents are part of the system itself and need not be transferred to the controlling device.

Satchel contains ten services: fetch, enquiry, print, view, scan and squirrel, conversion, beam, do-it, e-mail, and fax. These services can be used together to perform more complex operations. For example, the print service can act on any document to print, regardless of the document type. In order to handle various document types, the print service may invoke the conversion service to translate between a submitted document type and a printer-supported document type [13].

The architecture of the Satchel system focuses on document management. The discovery and installation of NAPs is not addressed by this system This is clearly a different focus than the work presented in this thesis, but there are two lessons that can be taken from this work. First, the system operates using web standard technologies. It is important for a new system to use existing successful standards where possible because the infrastructure to support these standards is already in place. Second, we can also deduce a lesson in combining services since doing so may be useful in breaking down device driver software for reuse.

2.2 Network service discovery technologies

One of the two keys to the system presented in this thesis is the ability to discover a NAP. A NAP provides a service on the network. Service discovery technologies can be used to discover the available NAPs.

A number of service discovery technologies exist in production today. This section contains a list of service discovery technologies that, while not exhaustive, identifies many of the available options to locate a network device that are in use today. A system need not be limited by the selection of a single service discovery technology but rather can make use of multiple technologies to achieve the goal of NAP discovery.

2.2.1 Service Location Protocol

Service Location Protocol (SLP) is an Internet Engineering Task Force (IETF) standard that allows for the discovery of network-based services according to the service type and characteristics. Currently at version 2, this protocol uses multicast Dynamic Host Configuration Protocol (DHCP) in support of service discovery.

An SLP system contains three types of agents: user, service, and directory. User agents assist in service discovery for client software. Service agents advertise the location and characteristics of a network service. Optional directory agents contain a repository of service information. Furthermore, discovery can be active or passive. Active discovery is

when a client initiates a search for a network service. Passive discovery is when a service broadcasts its availability on the network [14].

SLP supports both small and enterprise networks through its design, which aims to minimize the impact of service discovery on the network.

2.2.2 DNS-based Service Discovery

DNS Service Discovery (DNS-SD), currently an IETF Internet draft, allows clients to discover network services using standard DNS queries. Service names are already supported by DNS. DNS-SD adds the capability to make a query to a service pointer. The service pointer query returns a list of all the available services of a specified type. This list can then be used to select a specific service.

DNS-SD is compatible with Multicast DNS, which supports zero-configuration networking, and standard unicast DNS. The primary benefit of using DNS-SD is that it is based on the existing DNS technology, which is widely deployed and well understood [15].

2.2.3 Zeroconf

There are three solutions in the Zeroconf system: address auto-configuration, name-to-address translation, and service discovery. For service discovery, Zeroconf stores service information in DNS resource records using Multicast DNS. Zeroconf service discovery allows for the discovery of network services based on type. For example, a client can query for the available printing services on the network [16].

Bonjour is an implementation of Zeroconf developed by Apple and used in its OS X operating system. Bonjour already supports the discovery of printers.

2.2.4 Universal Plug and Play (UPnP)

UPnP, which was presented earlier in this chapter, has a service discovery component and is therefore mentioned again in this section.

UPnP service discovery functions using two methods: service advertisement and service discovery. Service advertisement is performed by devices to advertise their available network services using multicast messages. Service discovery is performed by client devices that wish to consume network services also via multicast messages. An XML document contains the service description details and is available to clients via HTTP [9].

2.3 Printing protocols

Since the implementation will focus on printers as a specific type of NAP, this section reviews three common network-based printing protocols.

2.3.1 Line Printer Daemon Protocol (RFC 1179)

The Line Printer Daemon Protocol, also known as RFC 1179 and LPR, is the original network print specification [17]. This protocol makes use of TCP/IP for line printing with a spool daemon running on the print server. This protocol has a number of limitations: the source port is limited to the range 721-731 inclusive; the print server must run on port 500; few print file types are supported; the protocols lacks extensibility; there is no method to obtain status information; and print options are limited to basic functions [18]. The Line Printer Daemon Protocol is a memo, not a specification, and has been superseded by other network print technologies.

2.3.2 Socket API

Socket API is a flexible job transfer protocol. Hewlett Packard JetDirect is the de facto standard implementation. This extremely simple protocol allows printing over TCP/IP using any port. Socket API supports client status updates as long as the connection

between the client and the server is open. Once the connection is closed, there is no built-in method to obtain status. The Socket API protocol has no built-in print queue. Responsibility for managing the print queue is left to the user. The user must ensure that a previous print job has completed before sending a new job to the printer. This has obvious implications in any environment with multiple users because coordination can become cumbersome. The Common UNIX Printing System (CUPS) includes an implementation of Socket API [6].

2.3.3 AppSocket

AppSocket is similar to Socket API with two notable differences: 1) the printer has both TCP and UDP ports for communication, and 2) a client device can terminate its connection with the printer by sending an end-of-job sequence in the data stream [6].

2.3.4 Internet Printing Protocol

The Internet Printing Protocol (IPP) is an IETF standard, with version 2 currently in approved candidate status as of February 2011. This standard includes many features that address issues of peripheral devices as network services including URI-addressable resources, a document model with extensible attribute support, built-in status communication, and security [7]. The original IPP standard work started in November 1996 by Novell and Xerox, with IBM, Lexmark, and Sun joining in shortly after the formation of the working group to collaborate on the specification [19]. This standard has a reference implementation in the CUPS [20]. This standard falls short of the goal of this thesis to have a zero-installation solution for NAPs because it does not address the issue of installing printer driver software.

The IPP Everywhere work being conducted by the IPP working group is currently investigating the zero-installation problem. The focus of IPP Everywhere is an adoption of standard document formats, discovery protocols, and schemas specific for printing [21] to eliminate the requirement of printer-specific device driver software.

Chapter 3

3 Design Considerations for the System

As discussed in Chapter 1, there are two high-level requirements to enable a client device to make use of a NAP:

1. Discover the location of the NAP on the network.
2. Register the NAP with the system and install the device driver software.

A third requirement was implicitly mentioned earlier, which we will explicitly list here:

3. Execute the device driver software.

With these three requirements in hand, we need to analyze the operations that support the automation of these requirements in order to understand the ramifications of changing the existing discovery, installation, and execution models. In this chapter, we tackle these problems.

3.1 Discover the NAP

A typical method to discover a NAP such as a printer is for a user to manually invoke a dialog box or a setup wizard and select a method to query the network. As shown in Figures 1 and 2, both the Microsoft Windows XP and Ubuntu Linux 10.10 operating systems use this approach. In both cases, the user selects a menu item to add a new printer and makes a selection in the dialog box to add a network printer, after which the network is queried and the results of the query are displayed.

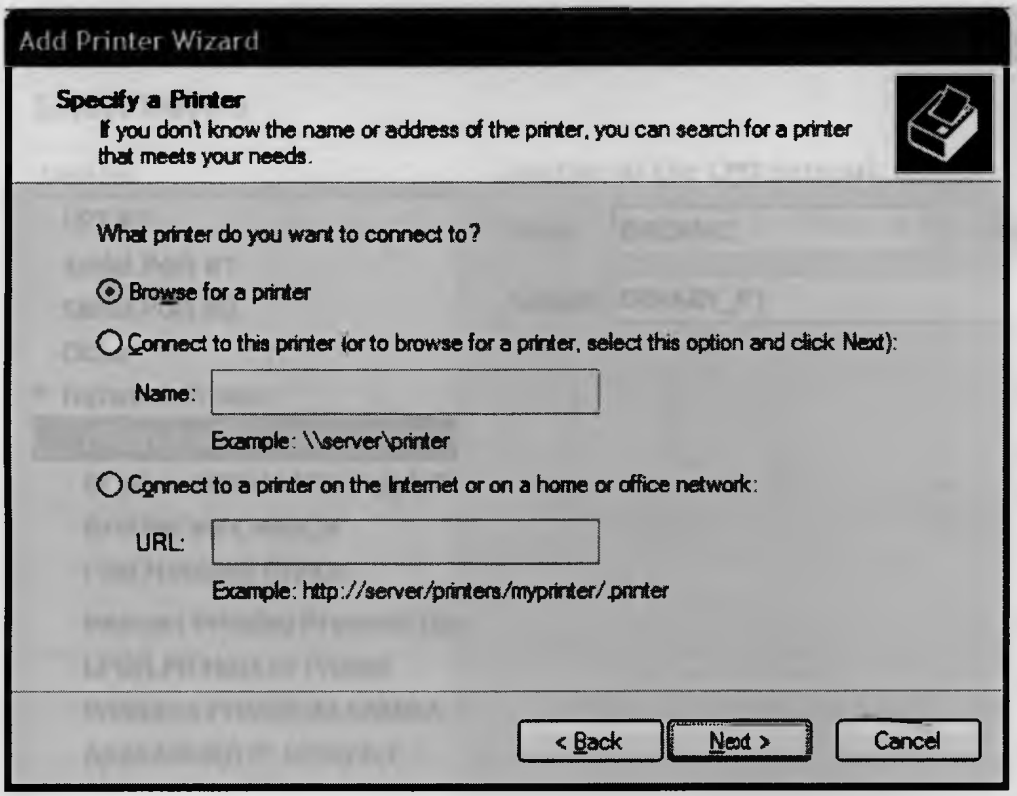

Figure 3-1 Discovery of a network attached printer in Windows XP

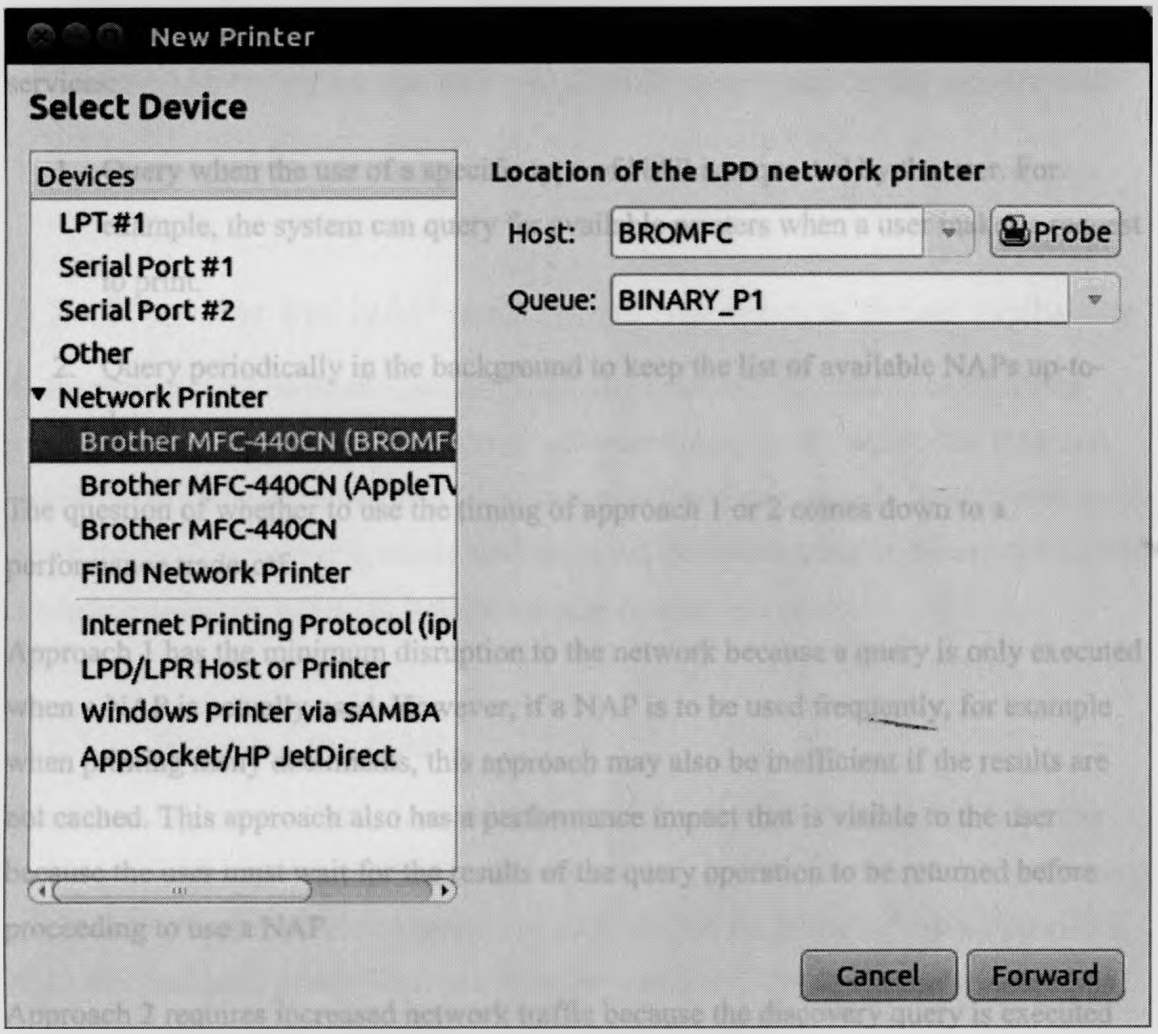

Figure 3-2 Discovery of a network attached printer in Ubuntu Linux

In order to discover a network attached printer, these dialog boxes may make use of a variety of service discovery protocols, some of which are detailed in Section 2.2. Service discovery is itself a topic of much research. The list of discovery protocols identifies many options, each with their own benefits and drawbacks and each with their own applicable scenarios and implementing technologies.

Beyond the method of discovering a NAP is the question of when to query for a NAP. Some of the service discovery protocols, such as SLP, detail when to query for new services. Others do not provide this guidance, leaving the decision up to the client application.

We will discuss two approaches to the timing of when to query for the available NAP services:

1. Query when the use of a specific type of NAP is requested by the user. For example, the system can query for available printers when a user makes a request to print.
2. Query periodically in the background to keep the list of available NAPs up-to-date.

The question of whether to use the timing of approach 1 or 2 comes down to a performance trade-off.

Approach 1 has the minimum disruption to the network because a query is only executed when a NAP is actually used. However, if a NAP is to be used frequently, for example when printing many documents, this approach may also be inefficient if the results are not cached. This approach also has a performance impact that is visible to the user because the user must wait for the results of the query operation to be returned before proceeding to use a NAP.

Approach 2 requires increased network traffic because the discovery query is executed more frequently. This may or may not have a noticeable affect on the network's performance. This approach should result in improved performance for the user because the list of available NAPs is populated as soon as a selection is made in order to make use of a specific type of NAP.

The solution in approach 2 contains a further complication in the timing of the background discovery queries. Two cases in which the network may be queried for the list of available NAPs are:

- When a network state change occurs, such as establishing a new network connection or terminating an existing network connection. In this case, the state of NAPs on the network should be considered to be unknown.

- At some refresh rate interval while a network connection exists, such as every 5 minutes. The refresh rate may depend on the type of NAP or the network itself.

This problem is not trivial to solve and furthermore may not have one correct answer.

3.2 Register the NAP and install the device driver software

In the current NAP installation scenario, the user manually registers a NAP with the system and also installs the device driver software manually. To register the NAP and install the software automatically, the system needs to obtain enough detailed information about the NAP to register it, locate and download the device driver software, and perform a silent installation, that is, an installation with no user interaction.

3.2.1 Register the NAP

In order to register the NAP, the system needs access to a description of the NAP that includes the information required by the registration process. As we have learned through our literature review of related projects, a good method for providing a description of a NAP that has already been employed in systems such as UPnP is to use web standards such as HTTP and XML. Following the UPnP model, each NAP can provide a description of itself in an XML document exposed via a web server. The advantages of XML are that it is in widespread use, it is platform agnostic, and it can be flexible for describing different types of NAPs. The system can use the information available in the NAP description to register the nap with the operating system.

3.2.2 Locate and download the device driver software

In order to provide a location from which to download the device driver software, the description document should include a reference to a location from which to retrieve the device driver software required for the NAP. In keeping with web standards, the location should be a uniform resource locator (URL). This method will provide a means to locate the device driver software.

With the device driver software available via a URL, the download method can also use a web standard technology such as HTTP or FTP in order to make the device driver software download as available as possible. It should also be possible to make the software download available via multiple methods and have the client pick the most suitable method for download.

A follow-up question related to how to download a device driver is *when* to download a device driver. While a NAP needs to be registered with the operating system in order to be available to the user, the device driver software may not need to be downloaded or installed until a request is made to use a specific NAP. There are performance implications to downloading and installing the device driver software for a NAP when it is located, in terms of bandwidth, disk space, memory, and processor usage. These implications add up to a desire to defer the download and installation of the device driver software until it is required.

Some print systems, such as CUPS, require that the device driver software be installed before a printer can be registered with the system. To support the requirement to have the device driver software installed before a NAP is registered, a device driver proxy can be used. The proxy acts as the device driver software and, when a request is made to use a specific NAP, the proxy automatically downloads and installs the device driver software at that time and passes the request to the real device driver software.

3.2.3 Perform a silent installation

Silent installation is a common option for software installers. This option is typically used for centrally managed software deployment in enterprises and scripted installation. The key to performing a silent installation is to require no user interaction. All information should be provided to the installation script or program when it is launched.

While it is preferable that the device driver installation should require no information, this may not be reasonable in all cases. Any additional information that is required can be provided in the NAP description document.

3.3 Execute the device driver software

There is a key difference with the device driver software in the new automated scenario: the software cannot be trusted. Software that is manually downloaded and installed can, to a certain degree, be trusted because the user is actively performing the installation task. In the manual scenario, the user should have the opportunity to verify the software signature, and can choose whether they trust the organization that has developed the software. In the automated scenario, the user does not manually locate or download the device driver software. It may be possible to automatically verify the software signature if a signature is provided and, as is done when downloading third-party code in Mozilla Firefox, the user can potentially be prompted to accept the download. Even in these cases, the system should follow the best practice exercised by web browsers such as Mozilla Firefox and Google Chrome for third-party code such as Flash, Java, and JavaScript applications: run the third-party code in a sandbox environment.

3.3.1.1 Sandbox requirements

The primary goal of a sandbox is to protect the system, or as Goldberg, Wagner, Thomas, and Brewer put it,

> *An application can do little harm if its access to the underlying operating system is appropriately restricted.[22]*

A sandbox is an environment for executing untrusted code that has limited access to the operating system and system resources such as memory, file storage, and network connections. A simple implementation of a sandbox environment is to create a user account on the system that has read-only file permissions and execute untrusted code using this user account. In this sandbox implementation, the untrusted code cannot write to the file system or execute programs and so should not be able to make changes to the file system.

Sandbox techniques are employed in production systems such as Java, which includes a sandbox environment in the Java virtual machine (JVM) [23]. There are also good

examples of sandboxes in modern web browsers such as Mozilla Firefox and Google Chrome, which apply this technique to protect the system from automatically downloaded third-party code such as Flash, Java, and JavaScript applications. Sandbox environments are still an active area of research as well with projects like Vx32 [24] and Native Client [25]. Although the goal of this thesis is not to design a method of restricting resources via a sandbox, we will identify one key requirement of the sandbox. The sandbox must allow the device driver software to communicate a result to the print system. In the case of CUPS, the result is the processed document that is ready to be transferred to the printer.

3.4 Summary

In this chapter, we covered some of the design considerations for the system and flushed out our scenarios to allow for further requirements and design decisions to be gleaned. In the next chapter, we will use this information to help create the system architecture.

Chapter 4

4 System Architecture

The architecture of this system introduces a new take on service description and introduces a new model of device driver installation and execution. It takes lessons from systems including Cooltown, Satchel, UPnP, Java, and web browsers such as Mozilla Firefox and applies them to help solve the problem of automatically discovering a NAP and installing the associated device driver software. The primary stengths from these other systems that are applicable to this system are the use of the XML and HTTP web standards for communication, the remote distribution of software, and the execution of untrusted code in a runtime sandbox. The architecture of this system bolts onto the existing operating system infrastructure for a specific type of peripheral device.

4.1 Quality attributes and tactics

The design of the system needs to incorporate the following quality attributes: flexibility, extensibility, security, useability, and performance.

4.1.1 Flexibility

The system must have the ability to support multiple operating systems and communications protocols.

Operating systems in common use today include Microsoft Windows, Linux variants (RedHat, SUSE, Ubuntu), UNIX, BSD, and MacOS. Mobile operating platforms in common use today include Symbian OS, iOS, Android, Blackberry OS, and Windows

Mobile. The architecture should not employ any design that is specific to an operating system.

Communications protocols are important for a system that is to support many different NAP devices. Some peripheral devices may require a document protocol, where the entire content of the information required to perform an action are sent before the action is performed. For example, it is typical to transfer the entire content of a document to a printer before printing begins. Other peripherals may require a streaming protocol, where the action to be performed is initiated before the entire content of the information has completed transferring. For example, it is typical to stream a live or large prerecorded video in order to start displaying the video quickly. Both of these communication mechanisms must be supported. The system should support the use of multiple communications protocols via device driver software.

Furthermore, the retrieval of the NAP description document and device driver software should be accessible to as wide an audience as possible. The system should use existing web standards for the retrieval and format of these artifacts.

4.1.2 Extensibility

The system must support the use of hitherto undefined protocols, platforms, and communication mechanisms so as to support innovative new communication techniques and peripheral devices. This quality is vital so that the system does not quickly become out of date, as was the fate of RFC 1179.

4.1.3 Security

Security is important from both a client and a server perspective.

The client concern is to prevent the system from being compromised by the automated installation of untrusted code and by establishing a secure communication channel with the NAP.

The server concern is to restrict access to the NAP to those with the appropriate authorization and avoid compromising the integrity of the NAP.

4.1.4 Usability

The driving factor behind this system is to improve ease of use as it relates to the discovery and installation of a NAP. The system must improve the use case of the initial use of a NAP that has not been previously installed and configured on a system. The system should also, where possible, integrate with the existing operating system's subsystem for each specific type of NAP as opposed to defining a new method of interaction with each type of NAP, such as requiring a user to print from a web browser.

4.1.5 Performance

The performance of the system is very important because it correlates to the usability quality attribute. Users may be connected for a limited time and may only be at a given location for a limited period of time. For example, a user may be at a location for an hour-long presentation and need access to the system only for the one hour that they are on site. While no specific time measurements have been recorded for how long a user is willing to wait for the system to be configured, the perceived performance must be that the system is configured quickly. The system should run background queries for the available NAPs in order to have an up-to-date list when the user requests the use of a specific type of NAP.

The system should only download and install the required device driver software when it is required to make use of a NAP. The system should employ a proxy driver to allow for the NAP to be registered without downloading and installing the device driver software.

4.2 System design and architecture overview

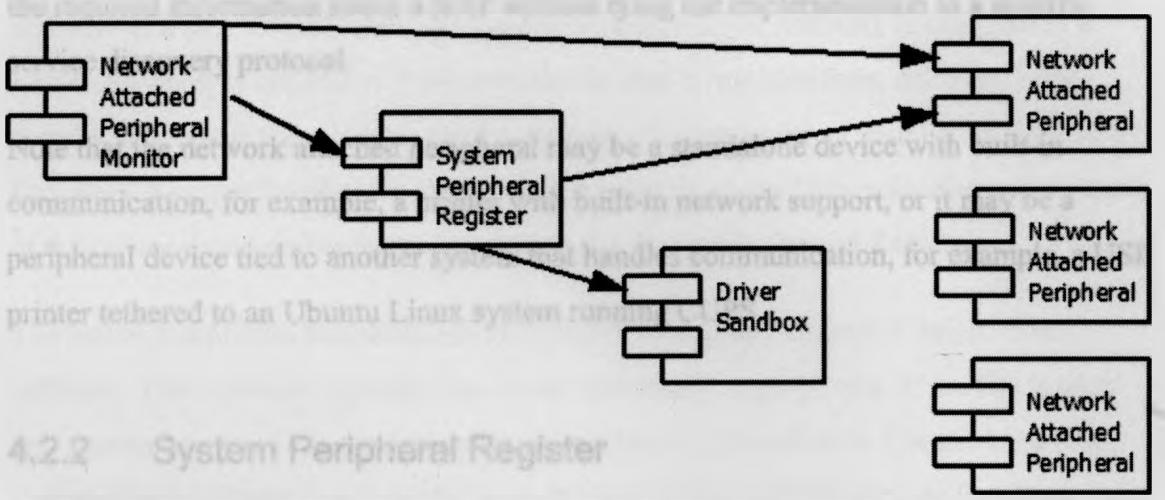

Figure 4-1 System architecture

The architecture of the system consists of four components: network attached peripheral, system peripheral register, driver sandbox, and network attached peripheral monitor.

4.2.1 Network Attached Peripheral

The network attached peripheral is the peripheral device that is to be used in conjunction with the system. For example, this may be a printer, a video projector, or an audio device. This device must have a mechanism for communicating as part of the network. Although existing service discovery protocols contain information such as the type of NAP and an identifier, they fall short of the required information that a NAP must provide for this system. In this system, the NAP uses a novel approach of providing all of the information required to register itself with an operating system. (The specific requirements for Ubuntu Linux are outlined in Section 5.1.3.) The description also includes the download location for any required device driver software. Providing the device driver software location is a change from the current model of providing the software on an installation disk bundled with the device. In this architecture, the device driver software can be bundled on the NAP itself or made available on the Internet for automatic download. In this system, the NAP provides all of this information in an XML description document

that is accessible via HTTP. The description document supports the flexibility and extensibility quality attributes by providing a mechanism for the system to obtain all of the required information about a NAP without tying the implementation to a specific service discovery protocol.

Note that the network attached peripheral may be a standalone device with built-in communication, for example, a printer with built-in network support, or it may be a peripheral device tied to another system that handles communication, for example, a USB printer tethered to an Ubuntu Linux system running CUPS.

4.2.2 System Peripheral Register

The System Peripheral Register is the standard platform registrar for a specific type of peripheral device. For example, on Ubuntu Linux, CUPS acts as the printer registrar. This component supports the flexibility and extensibility quality attributes through the ability to function with different platform registrars.

The Network Attached Peripheral Monitor will register each NAP and a proxy driver that it creates with the System Peripheral Register. The proxy driver partially supports the performance quality attribute through the deferred installation of the actual device driver software. It also partially supports the usability quality attribute by automatically downloading and installing the device driver software. After the device driver software has been downloaded, the System Peripheral Register communicates directly with the Driver Sandbox to execute the device driver software and with the NAP to execute an action.

This system interacts with the existing operating system's subsystem for a specific type of NAP. It does not require the replacement or modification of the existing subsystem. In this way, this system can work alongside the existing model of manually discovering and installing a NAP. This is an important feature because it supports a transition period between the two discovery and installation models.

4.2.3 Driver Sandbox

This system may automatically deploy new executable files in the form of device driver software to a user's machine, so the deployed device driver software is executed in a Driver Sandbox environment. This protects the user's machine from untrusted code, thereby supporting the security quality attribute. The proxy driver installed in the System Peripheral Register invokes the device driver software in the Driver Sandbox. The results of the device driver software are returned to the System Peripheral Register.

The use of a restricted execution environment is new in the context of device driver software. This software typically has access to a much larger portion of the file system and system resources. In some cases, such as when it is installed on Ubuntu Linux, the device driver software may even be executed as the root user. The Driver Sandbox changes this model and should significantly restrict the ability of device driver software to interact with the operating system.

4.2.4 Network Attached Peripheral Monitor

The components previously listed interact with one another directly. The Network Attached Peripheral Monitor is the central component required to manage the state of the system. The Network Attached Peripheral Monitor is responsible for querying for NAPs, retrieving their description document, generating the proxy driver, and registering the NAP with the System Peripheral Register. This component also identifies when a NAP is no longer available and consequently deregisters it with the System Peripheral Register.

The Network Attached Peripheral Monitor supports the usability quality attribute by automatically registering and deregistering NAPs with the operating system.

The process flow introduced above introduces a change in behaviour of existing NAP dialog boxes. In the current model, all installed NAPs will always be listed regardless of availability. In the new model, only those NAPs that are available will be listed. For example, in the current model, all installed printers are listed whenever the print dialog box is displayed, regardless of whether all of the printers are currently available on the

network. (Your printer at home will be listed in the print dialog box when you are at the office, for example.) In the new model only those printers that are currently available will be listed in the print dialog box, resulting in a list that contains only the printers of which a user can currently make use.

4.3 System communication sequences

There are three key use cases for this system:

- Discover and register a peripheral device with the system
- Use a peripheral device
- Deregister a peripheral device with the system when it is no longer available

This section covers the communication between the components in the system for these three key use cases.

4.3.1 Discover and register a peripheral device with the system

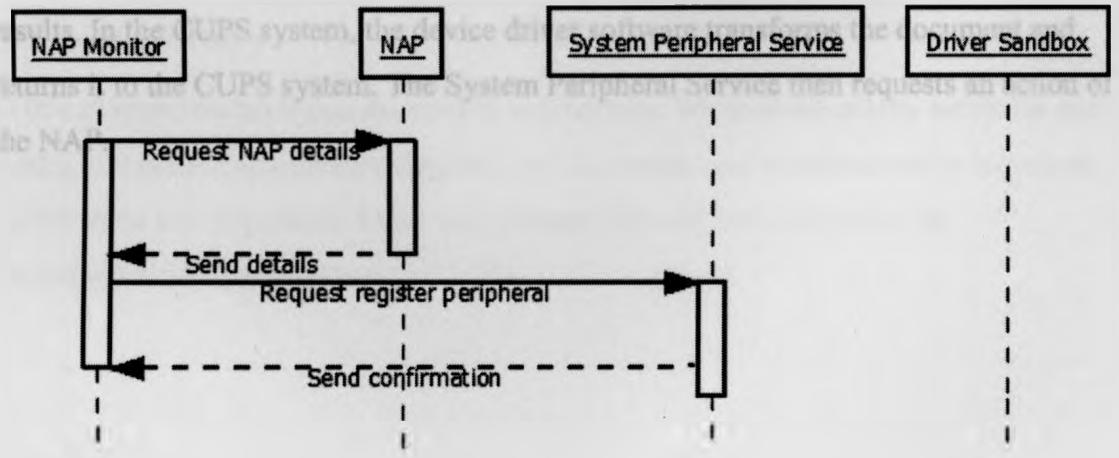

Figure 4-2 Component communication to register a peripheral device

The NAP Monitor system polls the network at a regular interval for the available NAP devices. Once a NAP is identified, the system requests the details from the NAP and then requests that the System Peripheral Service register the NAP.

4.3.2 Use a peripheral device

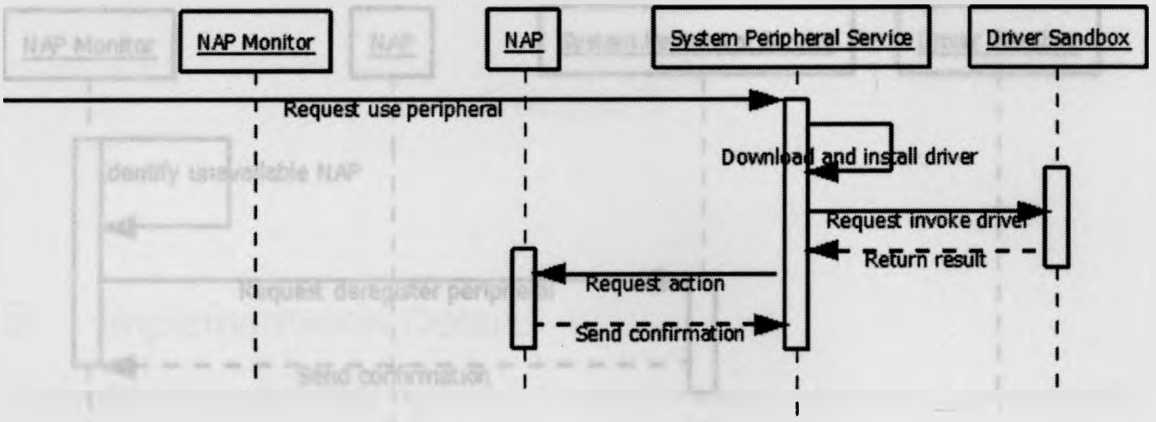

Figure 4-3 Component communication to use a peripheral device

When a NAP is to be used, a request is initiated from outside the system to the System Peripheral Service for the specific peripheral device type. For example, a request to print on Ubuntu Linux is made to the CUPS system. The System Peripheral Service invokes the proxy driver installed with the System Peripheral Service, which in turn downloads and installs the device driver software and invokes the device driver software in the Driver Sandbox. The device driver software then performs an operation and returns the results. In the CUPS system, the device driver software transforms the document and returns it to the CUPS system. The System Peripheral Service then requests an action of the NAP.

4.3.3 Deregister a peripheral device with the system

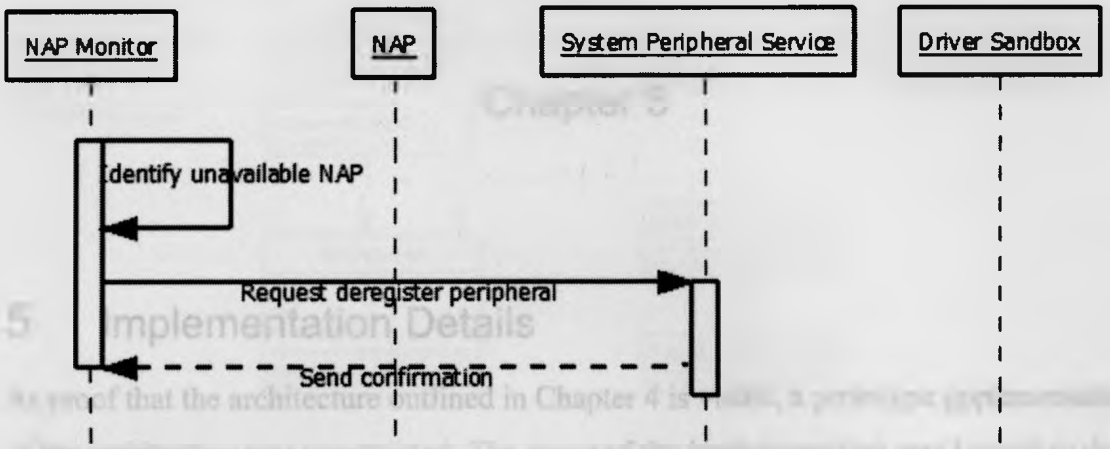

Figure 4-4 Component communication to deregister a peripheral device

The NAP Monitor polls the network for available NAPs at a regularly scheduled interval. When it identifies a NAP as unavailable, the NAP Monitor then sends a request to deregister the NAP with the System Peripheral Service. This action removes the NAP from the system.

4.4 Summary

In this chapter, we reviewed the system architecture. We covered quality attributes and tactics, the system architecture diagram, and the component communication sequences for the three key sequences. In the next chapter, we will look at a prototype implementation of this system.

Chapter 5

5 Implementation Details

As proof that the architecture outlined in Chapter 4 is viable, a prototype implementation of the architecture was constructed. The scope of the implementation was limited to that required to demonstrate the architecture without implementing proven technology. The high-level details of the scope are as follows:

- The implementation is restricted to print devices.
- The implementation uses a file-based locator.
- The implementation is restricted to Ubuntu Linux 10.10. The implementation therefore integrates with CUPS, the print system in use in Ubuntu Linux.
- No security is employed in terms of authentication and authorization. All printers are available to all users.

This system is implemented in Java and has a requirement on Java 1.5. The locator portion of the system makes use of a shell script that is specific to Linux.

In this chapter, we start with an overview of the NAP Monitor system and provide details about the network attached printers. We then conclude with a demonstration of the use of the system.

5.1 NAP Monitor

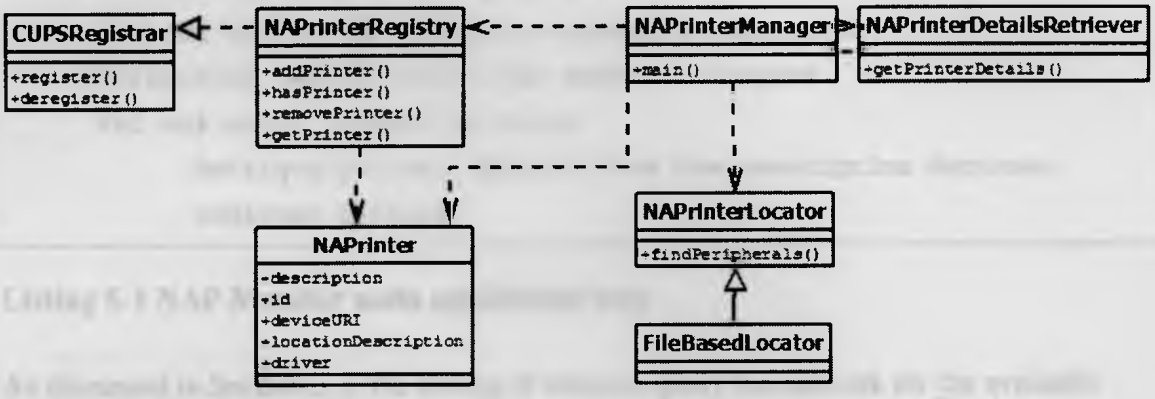

Figure 5-1 NAP Monitor class diagram

The NAP Monitor application is run on the client device. In this implementation, a client device is an Ubuntu Linux 10.10 system. Because the NAP Monitor is a Java application, a Java runtime environment (JRE) 1.5 or later must also be installed.

As shown in Figure 5-1, the implementation of the NAP Monitor centres around the `NAPrinterManager`. The `NAPrinterManager` contains the application's main method or entry point and controls the other objects in the NAP Monitor application, namely the `NAPrinterLocator`, `NAPrinterDetailsRetriever`, and `NAPrinterRegistry`.

5.1.1 NAPrinterManager

The `NAPrinterManager` contains the main method of the application and controls the application's execution. This class initiates a loop, shown in Listing 5-1, to handle the primary tasks of the system. The steps in the loop make a request to locate network attached printers from the `NAPrinterLocator`, deregister any printers that are no longer available with the `NAPrinterRegistry`, retrieve the details of the newly located printers using the `NAPrinterDetailsRetriever`, and register the newly located printers with the `NAPrinterRegistry`.

```
While application has not received a quit request
      Locate network attached printers
      Get the set of all printers currently registered
      Deregister any printers that were not located
      For any newly located printers
            Retrieve printer details from the description document
            Register printer
```

Listing 5-1 NAP Monitor main application loop

As discussed in Section 3.1, the timing of when to query the network for the available printers depends on the service discovery technology in use and the type of peripheral that is being queried. This implementation uses a file-based locator (discussed in more detail in the next section) and as such has no disruption to the network. For this prototype the network is therefore queried for changes frequently—every 10 seconds.

5.1.2 NAPrinterLocator

The `NAPrinterLocator` is an abstract class for performing service discovery to locate devices on the network. A concrete `FileBasedLocator` class has been provided that identifies printers based on the list of network attached printer URLs in a text file on the system.

```
#!/bin/sh

PRINT_DIR="/usr/local/nap/print"
PRINTERS_FILE="$PRINT_DIR/printers.txt"

HOME=0
OFFICE=1

echo "Initializing directories and files"
if [ ! -d $PRINT_DIR ]; then
   echo "Creating print directory"
   mkdir -p $PRINT_DIR
fi

if [ ! -e $PRINTERS_FILE ]; then
   echo "Creating printers file"
   touch $PRINTERS_FILE
   chmod 766 $PRINTERS_FILE
fi

network=-1
while [ true ]
do
   if [ "`ifconfig | grep 10.10`" ]; then
      if [ $network -ne $OFFICE ]; then
         network=$OFFICE
         echo "Changing to office network"
         echo "Office network has the following printers:"
         echo "1. Brother MFC-440CN"
         echo "http://10.10.1.111/brothermfc440cn/" > $PRINTERS_FILE
      fi
   fi
```

```
   if [ "`ifconfig | grep 192.168`" ]; then
      if [ $network -ne $HOME ]; then
         network=$HOME
         echo "Changing to home network"
         echo "Home network has the following printers:"
         echo "1. Samsung ML-1710"
         echo "http://192.168.1.140/samsungml1710/" > $PRINTERS_FILE
      fi
   fi

   sleep 5
done

exit 0
```

Listing 5-2 networksimulator.sh

The `FileBasedLocator` works in conjunction with a network simulator shell script, shown in Listing 5-2. `networksimulator.sh` changes the printer URLs listed in the text file on the system based on the network portion of the system's IP address. The shell script runs in a loop and updates the text file when a network change is detected. For the prototype, the script is hard-coded with two network locations, as described in Section 5.2, each with one network attached printer.

The `networksimulator.sh` script was created outside of the NAP Monitor application in order to simulate printer device changes on the network outside of the NAP Monitor application. This more closely simulates the control of NAP information since the NAP Monitor application will not have access to the NAP list when using a different service discovery mechanism.

This simple file-based locator scheme effectively demonstrates the changing devices on a network without spending too much time implementing an existing service discovery technology, which is not the focus of this thesis.

5.1.3 NAPrinterDetailsRetriever

The details of each network attached printer are contained in an XML description file named `printer.xml`. The contents of the description file are as follows:

- id – An ID for the printer. This should be a unique ID on the network.
- description – A textual description of the printer. This can be used by user facing tooling such as the print dialog box.
- location - The location of the printer on the network. The location includes both a URI and a textual description of the location.
- PPD file – A PostScript Printer Description (PPD) file contains further CUPS-specific information about the printer such as device capabilities and printing options.
- driver – The relative or absolute location of the printer device driver software file. This file will be downloaded, extracted, and installed.
- filter – The name of the filter file. This file name is used in the generation of the proxy driver and to identify the filter or main device driver entry point for the CUPS system.

A sample `printer.xml` file for the Brother MFC-440 CN printer is shown in Listing 5-3.

```
<?xml version="1.0" encoding="UTF-8"?>
<printer id="Brother-MFC-440CN" xmlns="http://print.nap.org">
      <description>Brother MFC-440CN</description>
      <location>
            <description>BROMFC</description>
            <deviceuri>lpd://BROMFC/BINARY_P1</deviceuri>
      </location>
      <ppdfile>Brother-MFC-440CN.ppd</ppdfile>
      <driver>BrotherMFC440CN.tar</driver>
      <filter>brlpdwrappermfc440cn</filter>
</printer>
```

Listing 5-3 Sample printer.xml description for the Brother MFC-440CN printer

The `printer.xml` file is retrieved by the `NAPrinterDetailsRetriever`. This object downloads the file and parses it for the information in order to create a `NAPrinter` object that represents a specific printer.

This `NAPrinterDetailsRetriever` communicates with a web server deployed with the network attached printer in order to retrieve the `printer.xml` file using HTTP. No security is currently supported in this communication channel.

5.1.4 NAPrinterRegistry

The `NAPrinterRegistry` functions as a typical registry in that it maintains a set of the discovered network attached printers. Each printer is modeled with a `NAPrinter` object. The registry stores `NAPrinter` objects using the printer URL as the key to identify and retrieve objects as required. The `NAPrinterRegistry` communicates with the `CUPSRegistrar`, sending it requests to register and deregister each printer.

5.1.5 NAPrinter

A `NAPrinter` object represents a specific network attached printer. The object contains all of the information found in the printer description document as needed to register a printer with the system and create the device driver proxy.

5.1.6 CUPSRegistrar

Once a newly available network attached printer has been located, it must be registered with the operating system's print subsystem. The `CUPSRegistrar` handles communication between the NAP Monitor system and CUPS, the print subsystem employed by Ubuntu Linux 10.10. The `CUPSRegistrar` provides the facility to register and deregistered a printer with the CUPS system. Registration is performed using the CUPS command line `lpadmin` command. For example, the following command will register the Brother MFC-440CN printer:

```
lpadmin -p  Brother-MFC-440CN -L BROMFC -D "Brother MFC-440CN" -P
Brother-MFC-440CN.ppd -v lpd://BROMFC/BINARY_P1
```

The information used to register a printer with the `lpadmin` command is provided in the printer description document.

Deregistration is also performed using the CUPS command line `lpadmin` command. For example, the following command will deregister the same Brother MFC-440CN printer:

```
lpadmin -x Brother-MFC-440CN
```

```
#!/bin/sh

if [ ! -e /usr/local/nap/print/filter/brlpdwrappermfc440cn ] ; then
  wget 'http://10.10.1.111/brothermfc440cn/brlpdwrappermfc440cn' -O \
/usr/local/nap/print/filter/brlpdwrappermfc440cn
  chmod 755 /usr/local/nap/print/filter/brlpdwrappermfc440cn
fi

if [ ! -e /usr/local/nap/print/driver/Brother.tar ] ; then
  wget 'http://10.10.1.111/brothermfc440cn/Brother.tar' -O \
/usr/local/nap/print/driver/Brother.tar
  tar -xpC/usr/local/nap/print/driver/ -f \
/usr/local/nap/print/driver/Brother.tar
fi

sudo -u napsandbox -c /usr/local/nap/print/filter/brlpdwrappermfc440cn\
"$@"

exit
```

Listing 5-4 Proxy driver for Brother MFC 440CN printer

The `CUPSRegistrar` is also responsible for the creation of a proxy driver in the CUPS system. The proxy driver will download the printer device driver software and marshal all use of the device driver to the sandbox area.

The device driver software is downloaded to the `/usr/local/nap/print/driver` directory on the system. This is the home of the sandboxed user account. A local user account *napsandbox* is created with limited permission to access the file system. This user account can read the file system but not write to it outside of its home directory shown above. This is especially important when working with the CUPS system because CUPS runs as the root user.

Note that many current device driver software packages expect to be able to install to any location on the system. This is the case with the out-of-the-box Brother MFC-440CN software. The software may need to be modified, as was the case with the Brother printer, to support running in this location.

An example proxy driver for the Brother MFC-440CN printer is shown in Listing 5-4. In this example, the proxy driver checks for the existence of the filter and driver. If either file does not exist, it will be downloaded. The system then changes to the *napsandbox* user, a user with restricted privileges, and executes the device driver via the filter.

Note that not all printers require specific device driver software. In the case that a printer makes use of a generic device driver included in the CUPS system, as is the case with the Samsung ML-1710 laser printer, the generic device driver will be used. Using the generic driver removes the need to install a specific device driver or execute the device driver in the sandbox area.

5.2 Network Attached Printer

Each network attached printer must provide the information about itself that is required by the NAP Monitor system. No printer includes the required information as part of their standard interface, so a supplementary method must be used to provide this information.

Each printer exposes this information in an XML description document, `printer.xml`, which the NAP exposes via HTTP. A sample `printer.xml` file can be seen in Listing 5-3. An Apache HTTP server is configured for each printer to serve the XML document via HTTP and exposes the printer information on the network.

To demonstrate the effectiveness of the system, two networks were configured. For differentiation, the networks are named *home* and *office*.

5.2.1 Home network

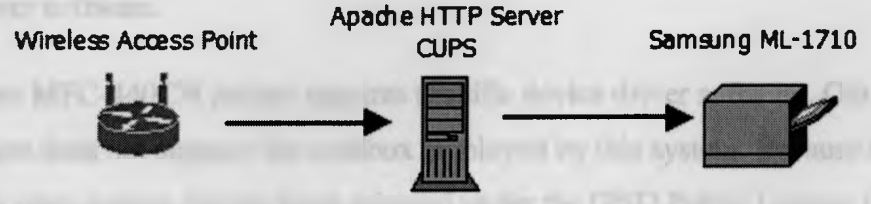

Figure 5-2 Home network configuration

The *home* network includes a wireless access point, Samsung ML-1710 printer, and Ubuntu Linux 10.10 server. The Samsung printer does not include any network support but rather is a USB printer. This printer is coupled with an Ubuntu Linux system and exposed on the network via the CUPS built-in ability to share a printer. A tandem Apache HTTP server is installed on the Ubuntu Linux system to serve the `printer.xml` description document.

The Samsung ML-1710 printer makes use of a generic device driver software package bundled with CUPS and so does not provide a device driver software package for download.

5.2.2 Office network

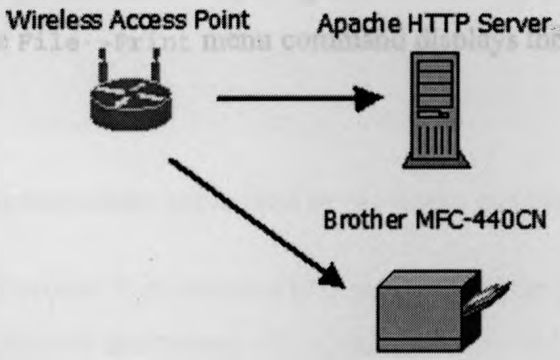

Figure 5-3 Office network configuration

The *office* network includes a wireless access point, Brother MFC-440CN printer, and Ubuntu Linux 10.10 server. The Brother printer provides built-in support for TCP/IP and LPD allowing it to print over the network. A tandem Apache HTTP server was installed on the Ubuntu Linux server to serve the `printer.xml` description document and the device driver software.

The Brother MFC-440CN printer requires specific device driver software. Out of the box, this software does not support the sandbox employed by this system. Because the software is open source, having been released under the GNU Public License (GPL), the

software package was modified to support relative and variable paths in order to install and execute it in the sandbox environment.

5.3 Using the prototype system

The NAP Monitor system is now set up. The NAP Monitor is running on the client Ubuntu Linux 10.10 system, which has no printers installed. Two networks, *home* and *office*, have each been configured with one network attached printer.

5.3.1 Connecting to the home network

When connected to the *home* network via the wireless access point, the client device is assigned the IP address 192.168.1.100. Opening the Firefox web browser to a page of our choice and selecting the `File->Print` menu command displays the print dialog box shown in Figure 5-4.

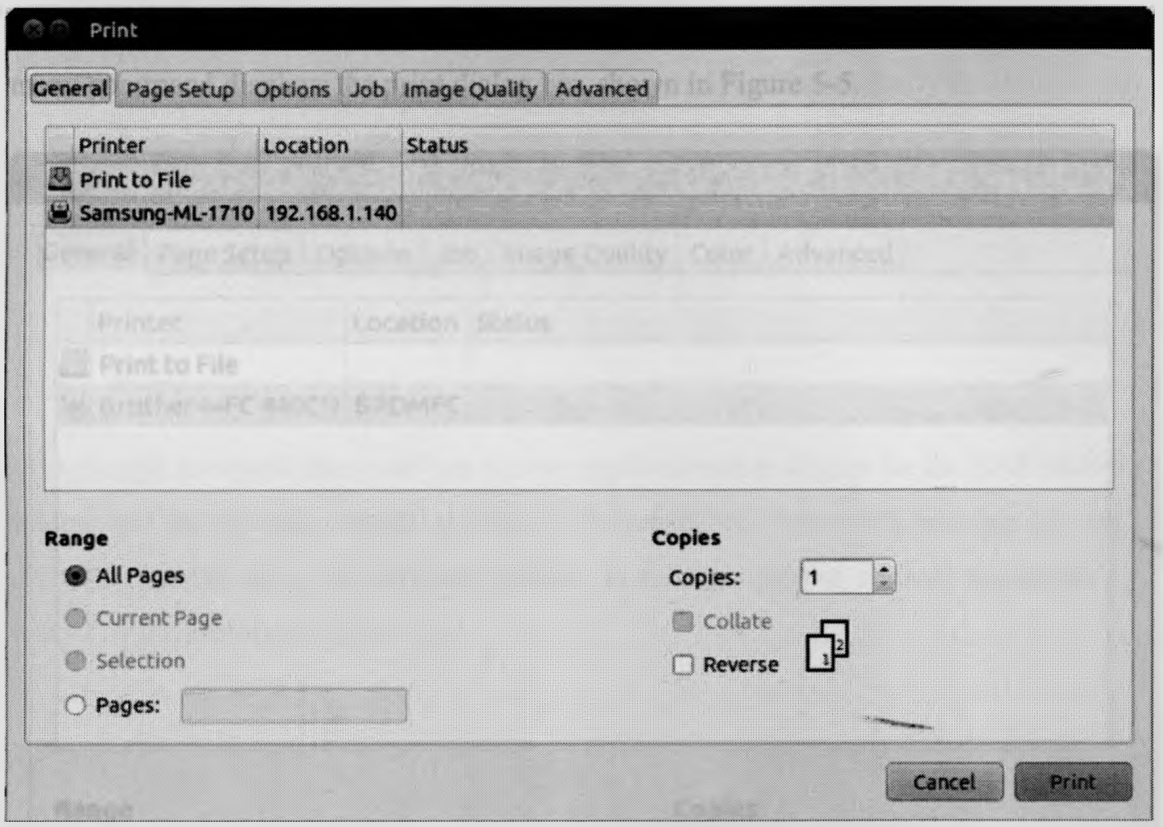

Figure 5-4 Print dialog box while connected to the home network

The Samsung ML-1710 printer is automatically discovered by the NAP Monitor application and registered with the system. On clicking the `Print` button, the web page open in Firefox is printed by the Samsung printer. The click of the `Print` button is notable as this system has introduced single-click printing. With one click, a print job has successfully executed on a printer that was not previously installed on the system.

As a reminder, the Samsung ML-1710 printer is exposed on the network via CUPS shared printing and uses generic device driver software that is bundled with CUPS. No device driver software had to be downloaded or installed for this printer.

5.3.2 Connecting to the office network

Next the network connection is changed from the *home* network to the *office* network. This change results in the assignment of IP address 10.10.1.100 to the client device.

Opening the Firefox web browser to a page of our choice and selecting the `File->Print` menu command displays the print dialog box shown in Figure 5-5.

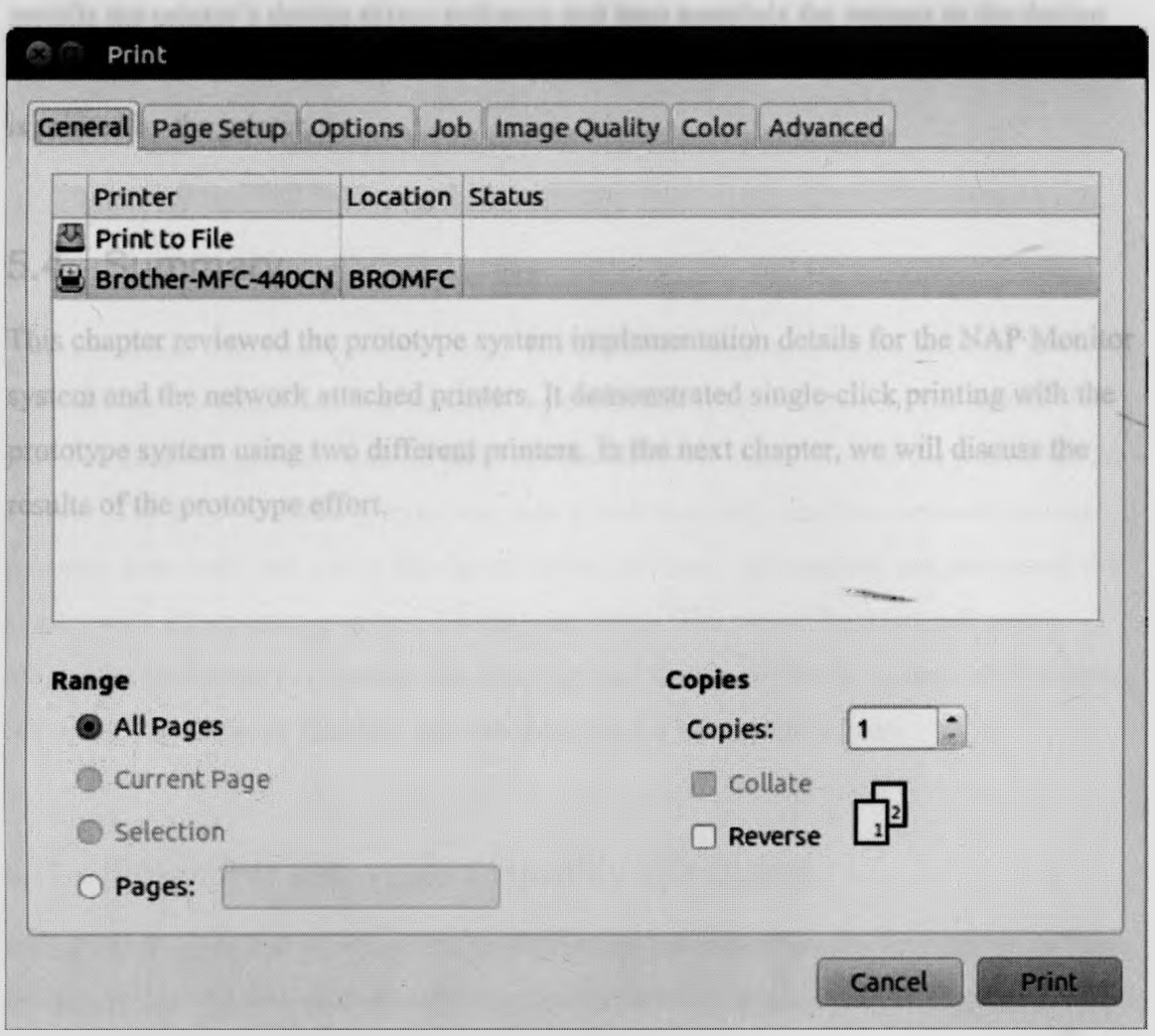

Figure 5-5 Print dialog box while connected to the office network

The Brother MFC-440CN printer has now been discovered by the NAP Monitor application and is displayed in the print dialog. Note that the Samsung ML-1710 printer is no longer displayed on the list of available printers. The NAP Monitor application has detected that this network attached printer is no longer available and has deregistered it with the system. When the user clicks the `Print` button, the web page open in Firefox is printed by the Brother printer. As a reminder, the Brother MFC-440CN printer has an Ethernet port and is connected directly to the network.

In this case, the Brother printer does require specific device driver software in order to function. After the print button is clicked, the proxy driver automatically downloads and installs the printer's device driver software and then marshals the request to the device driver software in the sandbox area. The results are returned to CUPS and the document is printed by the printer.

5.4 Summary

This chapter reviewed the prototype system implementation details for the NAP Monitor system and the network attached printers. It demonstrated single-click printing with the prototype system using two different printers. In the next chapter, we will discuss the results of the prototype effort.

Chapter 6

6 Discussion and Results

Our discussion starts on a positive note. The prototype implementation worked. It successfully demonstrated that the architecture of the system allows for the use of network attached printers with no previous discovery or installation of the printer on the system. The NAP Monitor system was able to automatically discover network attached printers, download and install the device driver software, and register and deregister the printer with the operating system's print subsystem. The use of the print subsystem allows the prototype to populate the existing print dialog on Ubuntu Linux 10.10 making the use of the network attached printers intuitive for the system's users.

6.1 Realizing the system quality attributes

In Section 4.1, we discussed system quality attributes and tactics for implementing them. In this section, we revisit these attributes and review our success in realizing them.

6.1.1 Flexibility

The system must have the ability to support multiple operating systems and communications protocols.

There are two components to this quality attribute: support multiple operating systems and support multiple communications protocols. The system and implementation were partially successful at realizing this quality attribute.

The architecture of the system is designed in a way that is platform- and NAP-agnostic. In this way, the architecture supports this quality attribute. However, the implementation of the prototype system does not attempt to prove this part of the architecture.

First, the prototype implementation is specific to Ubuntu Linux and interacts only with CUPS. It will be beneficial to expose the system to a wider variety of operating systems. Mac OS X also uses CUPS as its print subsystem, so it is reasonable to expect that the system can function on this operating system as well. However, OS X may require different device driver software. More investigation is required to understand how the system can be integrated on other operating systems such as Microsoft Windows and UNIX variants such as Solaris and AIX. A secondary group of operating systems that has not been addressed are mobile operating systems, which typically do not have print subsystems that expose a print option to the user and will therefore likely require a different integration mechanism.

A further complicating factor in the integration with the mobile operating systems is that there are significantly fewer device driver software packages available for these platforms. Generic device driver software, as used by CUPS with the Samsung ML-1710 printer, can help in many cases. Generic drivers that expose all or a portion of a NAP's functionality can make a NAP more generally available. There is active research on this front being conducted by the IEEE Printer Working Group via IPP Everywhere and the Linux Foundation Open printer workgroups.

Second, the prototype system as implemented is printer-specific. This system can be made into a generic framework for a variety of types of NAPs by creating generic components, interfaces, and abstract classes and creating NAP-specific implementations. For example, the registry can be modified to register NAPs by type, and specific system registrars such as the CUPSRegistrar can be implemented to plug into other operating system's subsystems.

Shifting our attention to the type of NAP supported by the prototype implementation, printers are a very mature category of peripheral devices. Others types of peripheral devices are not as ubiquitous, have not been in existence as long, and have therefore not

seen the same level of research and development. It may be a challenge to support other types of NAPs whose operating system's subsystem in less than mature. For example, there are existing configuration dialog boxes for video and audio devices on operating systems such as Microsoft Windows and Ubuntu Linux, but the list of these devices is fairly static today and users do not interact with them frequently. Other types of NAPs do not currently have operating systems with subsystems at all. Many NAPs are currently treated as file storage devices, for example, mobile phones, cameras, and music players, but this treatment does not provide the facility to use the capability of these devices. For example, I cannot make a phone call, take a picture, or play an audio file on the device. In these cases, it may be required to provide other software to integrate the NAP with the system.

Turning our attention to the second portion of the quality attribute, both the architecture and the implementation were successful in supporting multiple communications protocols. The system architecture includes a mechanism that supports the use of different device driver software. This mechanism was demonstrated in the prototype implementation through the use of two different printers, the Brother MFC-440CN and Samsung ML-1710, each of which used a different form of communications protocol and associated device driver software. Further testing with additional device driver software may reveal additional complications. This topic is addressed further in Section 6.2.1.

6.1.2 Extensibility

> *The system must support the use of hitherto undefined protocols, platforms, and communication mechanisms so as to support innovative new communication techniques and peripheral devices.*

The system architecture and implementation were once again partially successful at realizing the extensibility quality attribute.

The system architecture includes a mechanism that allows for the use of any device driver software. New software can automatically be downloaded and installed on the client

device. The new software can implement support for new protocols, platforms, and communication mechanisms.

The prototype implementation of the system demonstrated this quality attribute through its use of two different printers that made use of different device driver software. The implementation was limited in terms of platform support in that it only runs on Ubuntu Linux and only interacts with CUPS. Further work should include a port of the system to other platforms as discussed in Section 6.1.1.

6.1.3 Security

> *The client concern is to keep from compromising the system with the automated installation of untrusted code and by establishing a secure communication channel with the NAP.*
>
> *The server concern is to restrict access to the NAP to those with the appropriate authorization and avoid compromising the integrity of the NAP.*

In terms of security, only a small portion of the quality attribute was explicitly addressed in the system architecture and prototype implementation.

The system architecture requires that the system include a sandbox environment for executing untrusted device driver software. The architecture also spells out that web standards should be used if possible. In this way, the architecture leaves the door open for implementing standard solutions for secure communication via technologies such as secure HTTP (HTTPS) and standard authentication and authorization schemes.

The prototype implementation also only addresses the untrusted code portion of the quality attribute. The prototype implementation includes a sandbox for executing all downloaded device driver software. The prototype system has no user-level security outside of the driver sandbox. Security will need to be built into any production system. The prototype does not make use of HTTPS or any other secure communication

technology or standard authentication and authorization techniques when a user requests the use of a NAP. While there is a very large knowledge base in this area, the specifics of authenticating a user will need to be investigated in terms of any change required to the user interaction with the NAP operating system's subsystems.

An additional concern discovered during the prototype stage is the security of the newly available HTTP server packaged with each NAP. This concern relates to protecting the NAP itself. The HTTP server will need to be appropriately secured so as not to compromise the NAP.

6.1.4 Usability

> *The system must improve the use case of the initial use of a NAP that has not been previously installed and configured on a system. The system should also, where possible, integrate with the existing operating system's subsystem for each specific type of NAP.*

On the usability front, the system architecture and prototype implementation were very successful. Both fully support automated discovery of a NAP and the ability to download and install the NAP with no user intervention. This support, which was successfully demonstrated with the prototype implementation, effectively removes all manual steps from the user, requiring only a single click of the print button in order to print a document to any network attached printer.

The system should be further tested with additional types of NAPs in order to demonstrate its use with NAPs other than printers.

6.1.5 Performance

> *The perceived performance must be that the system is configured quickly...The system should only download and install the required device driver software when it is required to make use of a NAP.*

The system architecture and prototype implementation partially support the performance quality attribute.

Aside from the proxy driver, the system architecture does not specifically build in performance-related capabilities. The proxy driver does support the latter half of this requirement.

The prototype implementation does address both performance-related quality attributes. The prototype queries for available NAPs in the background, which results in an up-to-date list as soon as the user selects to print. This provides the perception that the system is always configured. (We will discuss this topic further in Section 6.2.2.)

The prototype also implements the driver proxy, which defers the download and installation of device driver software until it is actually needed, as prescribed in the system architecture. The proxy driver implementation does have a performance implication on the first use of a printer since the device driver software download and installation can take significant time. In the base of the Brother MFC-440CN printer, the time to download and install the device driver software on a local wireless G (54 Mbps max throughput) network with no other network traffic was measured at 15 seconds or more. While the overall installation time is significantly reduced from the manual installation model, 15 seconds or more is an unusually long time to print a small document. Increased print time may or may not prove to be a stumbling block for users. More measurement and user tests are required to determine the severity of this increase in print time.

6.2 Issues while prototyping

A number of issues were encountered while prototyping this system. In this section, we will identify the issues in order to understand the prototyping process and provide a base of knowledge from which to conduct further experimentation.

6.2.1 Inconsistent device driver software

The network attached printers do not use device driver software in a consistent way. The Brother MFC-440CN printer requires specific device driver software while the Samsung ML-1710 printer makes use of a generic device driver software package included in CUPS.

The Brother printer device driver software is installable simply by extracting an archive file. It is foreseeable that other printer software may have different installation requirements, such as requiring the use of an installer such as InstallShield. In addition, not all device driver software packages may support silent installation in that they may require user interaction. These device driver software packages will need to be modified in order to function with this system.

Furthermore, the Brother software installs out-of-the-box into specific directories on the system and hard-codes absolute file paths in its internals. This installation and execution model does not support the sandbox implemented in the NAP Monitor system, so the Brother software had to be modified to support relative and variable directory paths.

6.2.2 Printer refresh frequency

The refresh frequency for the available list of network attached printers may impact the system or network performance. It is therefore important to select a refresh frequency carefully. In the prototype system, the refresh was conducted every 10 seconds. We settled on the 10-second interval after attempting longer refresh periods. Unfortunately, the longer refresh periods resulted in unexpected behaviour: in some cases, a printer was not listed in the print dialog box when the print dialog box was invoked and only appeared later, after it was discovered. Other refresh strategies that can be tested include refreshing the list when the network connection changes, for example by detecting a change in the access point media access control (MAC) address, and refreshing when the user selects to print. In the latter strategy, it will likely be useful to communicate query status information to the user, which will require a change to the print dialog.

The 10-second refresh period had no effect on network performance in the prototype implementation; this was due to the file-based nature of the NAP discovery mechanism. It may not be practical to refresh the system this frequently in all cases. A refresh operation has performance consequences for the network due to increased traffic and for the client due to increased processing demands.

6.2.3 Unexpected errors

This system introduces the potential for network communication in the way of device driver software downloads during a print operation due to the addition of the device driver proxy. This new source of network communication is also a new source of failure in the system. Because of the limited user interaction in this download and installation of the device driver software, the system may not fail gracefully. In our experiments, if the download or installation of a device driver software package does fail, the user is alerted that the print job has failed and must then look into the CUPS log files to determine why. The device driver proxy also hides certain device driver software failures from CUPS. It may not be immediately apparent why the print job failed and, consequently, it may be difficult to diagnose the source of the failure for users who are not familiar with CUPS.

6.3 Other considerations

The prototype is limited in a number of ways, so there are opportunities for improvement. Some of these opportunities are outlined in this section.

6.3.1 Clean up of device driver software

In the simple scenario used for the prototype, there is only a single network attached printer available in each of two network locations. Deregistering a printer from the system does not remove the associated device driver software. Once the device driver software for a printer is downloaded and installed, it is never removed from the system. In a larger scenario, this will lead to increased disk usage with software remaining on the system after it has outlived its use. A solution for this is to employ some sort of clean-up

routine, such as a least recently used algorithm, to rid the system of device driver software that is not in use.

6.3.2 Viability in a network with many NAPs

In a real deployment, there may be hundreds or thousands of peripheral devices of different types available on a network. It may be crippling for a system to register a large number of NAPs or continually query for the available NAPs on the network.

Another concern when a lot of NAPs are in use is how this affects the usability of the system. For example, a user may find it difficult to identify which printer they want to use if too many printers are listed in the print dialog box.

Chapter 7

7 Conclusions

The existing manual discovery and installation of device driver software for network attached peripheral devices limits the ability of users to actually use the peripheral devices. The goal of this thesis as stated was to eliminate these manual steps.

In this thesis, we have presented an architecture for a system with unique aspects of service description and device driver execution that automates the discovery and installation of device driver software for network attached peripheral devices. The architecture integrates with the existing operating system peripheral device specific subsystem to make it as seamless as possible for the end user. It also enables single-click use of a network attached peripheral device. A prototype implementation of the architecture specific to printers was developed and presented that demonstrated the viability of the architecture.

Through the development of the system architecture and the prototype implementation, a number of additional questions have been identified. Following are some of those questions presented as future work that can be performed related to this subject.

- Demonstrate the applicability of this system architecture on additional operating systems such as Microsoft Windows, Mac OS X, and UNIX variants, and mobile operating systems such as iOS, Android, and Blackberry OS.

- Demonstrate the applicability of this system architecture with additional network attached peripheral devices such as video projectors and audio devices.

- Perform user tests with different NAP query timing using different service discovery protocols to determine a good balance between refresh frequency and performance. Additional refresh strategies should also be explored.
- Explore the use of existing service discovery technologies for including additional description information.
- Explore the use of multiple service discovery technologies in a single system.
- Identify the set of sandbox requirements for the driver sandbox.
- Explore new security concerns introduced by the system architecture.
- Identify ways in which to reduce the need to install device drivers. This work is already being investigated for printers by the printer working group. Additional work can be done for other types of NAPs.

Bibliography

[1] Janson P, Svobodova L, Maehle E. Filing and printing services on a local-area network. In: Proceedings of the eighth symposium on Data communications. SIGCOMM '83. North Falmouth, Massachusetts, United States: ACM, 1983. pp. 211–220. doi:10.1145/800034.800922

[2] Rabin J, McCathieNevile C. Mobile Web Best Practices 1.0. 2006. Available: http://www.w3.org/TR/2006/WD-mobile-bp-20060113/. Accessed 14 Jul 2011.

[3] Kindberg T, Barton J, Morgan J, Becker G, Caswell D, Debaty P, et al. People, Places, Things: Web Presence for the Real World. Mobile Networks and Applications 2004;7:365-376. doi:10.1023/A:1016591616731

[4] Park DG, Kim JK, Sung JB, Hwang JH, Hyung CH, Kang SW. TAP: touch-and-play. In: Proceedings of the SIGCHI conference on Human Factors in computing systems. CHI '06. Montreal, Quebec, Canada: ACM, 2006. pp. 677–680. doi:10.1145/1124772.1124873

[5] Oliveira da Silva A, de Souza Schneider PH, D'Avila Cabral F, Benso da Silva AC, Batista de Oliveira J, Bezerra EA. Towards service and user discovery on wireless networks. In: Proceedings of the second international workshop on Mobility management & wireless access protocols. MobiWac '04. Philadelphia, PA, USA: ACM, 2004. pp. 79–82. doi:10.1145/1023783.1023799

[6] Patrick A Powell. LPRng Reference Manual. 2010. Available: http://www.lprng.com/LPRng-HOWTO/LPRng-Reference.html. Accessed 24 Jul 2011.

[7] Bergman R, Lewis H, McDonald I eds. Internet Printing Protocol Version 2.0 Second Edition (IPP/2.0 SE). IETF. 2011 p. Available: ftp://ftp.pwg.org/pub/pwg/candidates/cs-ipp20-20110214-5100.12.pdf. Accessed 24 Jul 2011.

[8] Meadows J. An Introduction to the JetSend Protocol. EE Times India 2000. Available: http://www.eetindia.co.in/ARTICLES/2000JAN/PDF/EEIOL_2000JAN03_EMS_NETD_TA.pdf?SOURCES=DOWNLOAD. Accessed 24 Jul 2011.

[9] Presser A, Farrell L, Kemp D, Lupton W, Tsuruyama S, Albright S, et al. UPnP Device Architecture 1.1. UPnP Forum. 2008 p. Available: http://upnp.org/specs/arch/UPnP-arch-DeviceArchitecture-v1.1.pdf. Accessed 15 Jul 2011.

[10] Harihar K, Kurkovsky S. Using Jini to enable pervasive computing environments. In: Proceedings of the 43rd annual Southeast regional conference - Volume 1. ACM-SE 43. Kennesaw, Georgia: ACM, 2005. pp. 188–193. doi:10.1145/1167350.1167407

[11] Rhodes BJ, Chemishkian S, Schwartz EL, Savitzky S, Yu H. Automatic discovery and execution of personal applications from shared IO devices. In: Proceedings of the 7th IEEE conference on Consumer communications and networking conference. CCNC'10. Las Vegas, Nevada, USA: IEEE Press, 2010. pp. 1163–1164. Available: http://portal.acm.org/citation.cfm?id=1834217.1834491. Accessed 15 Jul 2011.

[12] Sousa JP, Garlan D. From Computers Everywhere to Tasks Anywhere: The Aura Approach. 2001. Available: http://www.cs.cmu.edu/~aura/docdir/sg01.pdf. Accessed 24 Jul 2011.

[13] Flynn M, Pendlebury D, Jones C, Eldridge M, Lamming M. The Satchel system architecture: mobile access to documents and services. Mob. Netw. Appl. 2000;5:243–258. doi:10.1023/A:1019172931873

[14] Guttman E. Service Location Protocol: Automatic Discovery of IP Network Services. IEEE Internet Computing 1999;3:71–80. doi:10.1109/4236.780963

[15] Cheshire S, Krochmal M. DNS-Based Service Discovery. 2011. Available: http://files.dns-sd.org/draft-cheshire-dnsext-dns-sd.txt. Accessed 17 Jul 2011.

[16] Stirling D, Al-Ali F. Zero configuration networking. Crossroads 2003;9:19–23. doi:10.1145/904080.904084

[17] Leo J. McLaughlin III. RFC1179: Line Printer Daemon Protocol. IETF. 1990 p. Available: http://www.ietf.org/rfc/rfc1179.txt. Accessed 14 Jul 2011.

[18] System Administration Guide: Printing: IPP Compared to the RFC-1179 Protocol. 2010. Available: http://download.oracle.com/docs/cd/E19253-01/819-7355/gezsg/index.html. Accessed 14 Jul 2011.

[19] Frequently Asked Questions - Internet Printing Protocol - Printer Working Group. n.d. Available: http://www.pwg.org/ipp/faq.html. Accessed 16 Jul 2011.

[20] Internet Printing Protocol - Printer Working Group. n.d. Available: http://www.pwg.org/ipp/. Accessed 16 Jul 2011.

[21] Sweet M, McDonald I. IPP Everywhere First Edition. IETF. 2011 p. Available: ftp://ftp.pwg.org/pub/pwg/ipp/wd/wd-ippeve10-20110326.pdf. Accessed 15 Jul 2011.

[22] Goldberg I, Wagner D, Thomas R, Brewer EA. A secure environment for untrusted helper applications confining the Wily Hacker. In: Proceedings of the 6th conference on USENIX Security Symposium, Focusing on Applications of Cryptography - Volume 6. San Jose, California: USENIX Association, 1996. pp. 1–1. Available: http://portal.acm.org/citation.cfm?id=1267569.1267570. Accessed 17 Jul 2011.

[23] Gong L, Mueller M, Prafullchandra H, Schemers R. Going beyond the sandbox: an overview of the new security architecture in the java development Kit 1.2. In: Proceedings of the USENIX Symposium on Internet Technologies and Systems on USENIX Symposium on Internet Technologies and Systems. Monterey, California: USENIX Association, 1997. pp. 10–10. Available: http://portal.acm.org/citation.cfm?id=1267279.1267289. Accessed 17 Jul 2011.

[24] Ford B, Cox R. Vx32: lightweight user-level sandboxing on the x86. In: USENIX 2008 Annual Technical Conference on Annual Technical Conference. Boston, Massachusetts: USENIX Association, 2008. pp. 293–306. Available: http://portal.acm.org/citation.cfm?id=1404014.1404039. Accessed 17 Jul 2011.

[25] Yee B, Sehr D, Dardyk G, Chen JB, Muth R, Ormandy T, et al. Native Client: A Sandbox for Portable, Untrusted x86 Native Code. In: Proceedings of the 2009 30th IEEE Symposium on Security and Privacy. IEEE Computer Society, 2009. pp. 79–93. doi:10.1109/SP.2009.25