Multi Agents for Heterogeneous Operating System Environments

Abhishek Mathur

A thesis submitted in total fulfillment of the requirements
for the degree of

Master of Applied Sciences in Information & Technological Sciences

at the

School of Information Sciences and Engineering

University of Canberra

July, 2007

In loving memory of my aunt

Gretchen F Mathur

Abstract

As IT industries develop, upgrade and diversify, heterogeneous operating environments running a mix of new and legacy systems become increasingly important. Such environments are currently inadequate due to lack of compatibility with each other. This thesis investigates how agents can be utilised to facilitate such heterogeneous environments, aiding enterprise systems in building bridges between incompatible software and hardware systems. An autonomous agent has independent agency and decision-making astuteness. When placed in heterogeneous environments to interact with other such agents, the consequences of its action and its preferred choice of actions are greatly influenced by actions of other agents interacting in heterogeneous environments.

The main objectives of this thesis include examining the roles of agents in heterogeneous operating environments, development of a novel multi agent base architecture and an associated framework for single and heterogeneous environment. The research work also studies the plausible application to test the developed proof of concept by developing application and using the framework that utilises Windows services in a totally incompatible Solaris based Sun Ray ultra thin client environment.

The work includes a novel method of modeling agent based communication architecture suitable for correspondence between two inherently different operating systems – Solaris and Microsoft Windows. The circumstances in which coordination or coordination failure occurs between these systems are investigated. The proposed method of agent based communication that can potentially overcome the barriers formed by two completely different software and hardware architectural regimes.

An analysis of printing services in MS Windows and Solaris environments, review the age long problem of lack of device drivers for commonly (and cheaply) available Ink Jet printers for Unix (and like) operating systems. A novel method is proposed that uses agents in heterogeneous environment to overcome this problem. A new architecture that utilises Windows based printing services on a Sun Ray ultra thin client is presented to test and evaluate the proof of concept.

This thesis is motivated by the need to provide a low cost printing solution to Sun Ray users. Most Windows based desktop users currently have access to variety of low cost printing solutions. Printer vendors ship device drivers only for Windows or at most Macintosh, as other operating systems such as Solaris, MVS, z/OS are used for corporate solutions and low cost desktop printing have not been a major requirement in the past.

Acknowledgement

I am very grateful for the advice and support of my supervisors Associate Professor Dharmendra Sharma and Dr Wanli Ma, for their feedback and for keeping me focused in my research. Their comments and time have greatly improved and clarified this work. Thank you to ActewAGL and Sun Microsystems Canberra who provided the valuable resources and monitory support. I am particularly thankful to Mr. Scott Carr for providing opportunities to work at Sun Microsystems' Laboratory and all the encouragement he provided.

Table of Contents

Table of Figures

Chapter 1

Introduction

Computers of the 21st century have formed an image of being great problem solvers and have become an indispensable part of the human society. Computer scientists and researchers have engaged themselves in solving varying predicaments in business, industries, and production and telecommunication markets. Much like electricity, water, telephones and other utilities, organisations recognise computers as a necessary tool to carry on their business process. Unfortunately, most organisations are unable to make informed decision on computers, as on other needs. This is mainly because computer systems are more complicated than business systems. Moreover, different vendors provide solutions based on totally different underlying technologies that are generally non-compatible. An enterprise with a matured IT setup, which has undergone several upgrades with time, would consist of a blend of solutions from different vendors. This brings about a heterogenous operating environment in the enterprise. Making an arbitrary edict on IT based system has mostly resulted in relentless disasters [81].

We would expect a right system that has the potential to evolve with growing business requirements would comprise of a mix of various hardware and operating systems, which are dedicated for specific tasks. For an accounting system it is unlikely to run on the same system as World Wide Web. For example, a small business may be using a Windows based accounting software and have an Intel infrastructure to support its Accounting, Finance and Human Resource divisions. At

the same time, it may have deployed a Linux based infrastructure to cater to its Internet / Intranet portals and email servers to cut costs and provide high security with stringent firewalls. Moreover, changing business needs demands regular upgrade in computer systems, which brings about diversity in the overall IT arrangement [37]. Such an enterprise would have evolved with time and several upgrades and architecting the best solution for added functionality would lead to heterogeneous operating environment. Complexity of such environments grows in arithmetical proportions with changing demands.

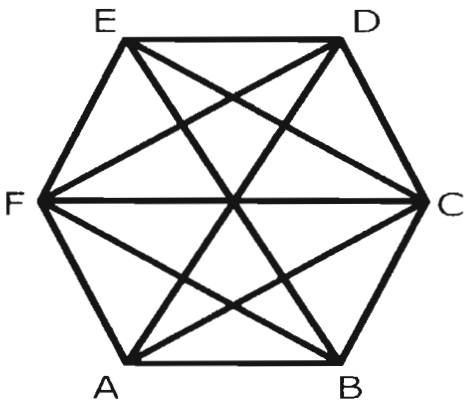


Figure: 1. Inter communication in a mixed environment.

Operating systems can be grouped in various ways. By functionality: operating systems for supercomputing, render farms, mainframes, personal computers, handheld devices, real time systems, or embedded systems [37]. Supercomputers provide scientific computing, usually modelling real systems. Perhaps the best known Supercomputers are built by Cray Research [43]. Render farms are collections of computers that work together to render animations and special effects. Mainframes provided the bulk of business computing through time-sharing. Mainframes and mainframe replacements (powerful computers or clusters of computers) are still useful for some large-scale tasks, such as centralized billing systems, inventory systems, database operations, etc. IBM's MVS (Multiple Virtual Storage) was the most widely used Mainframe operating system, which is superseded by OS/390 and z/OS. The payroll, accounts receivable, transaction processing, database management, and other programs critical to the world's largest businesses

are usually run on an MVS or successor system. Mainframes are used where large amount of processing is required over a short period of time. When mainframes were in widespread use, there was also a class of computers known as Minicomputers, which were smaller, less expensive versions of mainframes for businesses that could not afford true mainframes [77]. With such diversity in computer architectures and in order to utilise the best of all the operating systems in a heterogeneous operating environment, there arises a need to construct bridges between various operating system softwares. The current work investigates the needs and proposes a server based heterogenous environment to adequately capture the process and cater for the modern day needs. By deploying agents in such heterogeneous operating environments, bridges can be constructed to utilise services of an operating environment in another.

1.1 Motivation

Most devices for desktop computers such as printers, scanners, cameras are only supported on Microsoft operating environments or at most on Macintosh. This leaves a large percentage of UNIX (and like) users unable to utilise these device due to lack of appropriate device driver.

Need for device drivers for such devices on unsupported platforms have instigated this research work. Application of agents and agent based technologies to the existing driver mechanism in order to gain efficiency benefits [48] has also been a motivating factor to carry out this study.

1.2 Aims and Objectives

The primary aim of this research work is to develop the Multi Agent Printing System (MAPS) framework for heterogeneous operating environment which allows making use of system software of one operating system in another. In particular, the work is an attempt to make use of peripheral devices such as printers and digital cameras, in an operating environment where the manufacturer did not originally provide any

support. The research also aims at testing and deploying this MAPS framework on various operating systems – standalone, networked and heterogeneous, in particular on – Sun Ray ultra thin clients running on Solaris Operating System (OS). Finally, it aims at testing the performance and usability of such a solution in the Sun Ray thin client environment.

The research objectives associated with this thesis are stated as follows:

1. Investigate the various distributed computing architectures and technologies available to current heterogeneous enterprise solutions.
2. Study the peripheral device architectures of Sun Rays running on Solaris Operating System and MS Windows. This study requires investigation of:
   - USB interface and its software libraries on Linux / Solaris.
   - USB Digital Cameras and their operability in Solaris and like systems.
   - Low cost USB printers and their available device drivers on Windows, Solaris and Linux.
3. Analyse the extent of work already done in cross platform interoperability, in terms of peripheral device utilisation. Investigate the efforts made to port, tweak or re-engineer existing device drivers for unsupported Operating Systems.
4. Design and implement the Agent based Service Oriented Architecture (ASOA) for exploiting cross platform services. The architecture suggests an ecosystem of agents where each agent has a life and is dependent on other agents for its proper functioning.
5. Design the Multi Agent Printing System (MAPS) framework for utilising unsupported printers in Linux / Solaris, making use of device drivers installed on a supported Operating System. This will provide ready solution for interoperability of devices without having to re-engineer or tweak device drivers for unsupported operating systems.
6. Deploy and test the above framework on single, networked and cross platform environments over TCP/IP.

7. Test the performance of MAPS printing on Sun Ray ultra thin clients and MS Windows over simulated WAN.

1.3 Background

Sun Microsystems (Canberra), TransACT Communications – a Canberra based communication carrier, and the University of Canberra have participated to research and develop the Community Computing Project. This project strives at providing computing as a utility service to the Canberra community. The main aim is to utilise Sun Ray thin clients running on Solaris over pre-existing xDSL network. This idea offers basic desktop computing, Internet access with almost no maintenance required at the user end, for a low cost. Thin clients can be plugged into the xDSL socket and can be used seamlessly as an appliance. User experiences a GUI rich desktop computing power with no administration overheads while all the housekeeping jobs can be centrally administered at the server. This introduction of computing appliance allows exceptional ease of use – just as in a telephone device which plugs into a wall socket and is ready to for use at once. Sun Ray thin client technology is chosen as the right delivery infrastructure.

Sun Rays primarily run on Solaris servers. Solaris server maintains session state of each client, and sends the display data over the network to the respective client, while clients capture the keyboard and mouse movements and transmits them back to the server for processing [41]. Clients have no computational power and only have enough memory to maintain the session token. Recently, Sun Microsystems had allocated resources to develop a PPPoE client which also resides on the client server.

One of the problems encountered is to provide Sun Ray users with low-cost printing. Most home inkjet printers in the market are non PostScript compliant. Device manufacturers have their own proprietary print drivers for these printers and are driven totally by their native script (.prn file) generated by Windows device driver. All such printers are shipped with Windows drivers, but never has been an effort to utilise these printers in Solaris (or like) environment.

The challenge is to make the printers work for Solaris, and therefore, provide Sun Ray users with printing capability to the printers of their choices from the market.

The first attempt to solve such a problem was to port existing Linux device drivers. Many users of the open source community have successfully used Gimp or Ghost Script [75] to enable non Postscript printers with Linux. Such a solution works with only a handful of printers and more importantly, it is heavily reliant on open source resources.

Bridging different operating systems into a heterogeneous operating environment to provide a neat and clever solution to an enterprise is the technological challenge. This shall enhance utilisation of services in one operating system in another. In particular, we are keen on utilising devices manufactured primarily for Microsoft Windows, on a Solaris thin-clients running on xDSL networks. This challenge of utilising a robust server operating system (Solaris) for desktop applications and at the same time providing compatibility with wide range of Windows devices gives us the challenge and encouragement to take on this research work.

This research work would provide an end user with a low maintenance Sun Ray thin-client, ready to be plugged in to an xDSL modem and simultaneously make the most of Windows compatible printers through its USB ports.

Intelligent or adaptive systems have become the major focus of research within fields of data mining and decision-making. Capabilities of such systems can mutate to improved states, adapting and learning from experience. Real time decision support systems have contributed in a big way to problem solving techniques. Software environments now allow a developer to build a knowledge level model and to create the final operational version [62]. Such systems are already being used in portfolio management and medical decision support areas [61, 63, 76]. Our effort is to redirect

such forces towards building bridges between non-compatible operating systems and to exploit resources of one operating system from another.

1.4 Scope of Thesis

The intent of this research is to develop mechanisms for enabling high-level software such as applications, message-passing libraries, and runtime systems for the better management of resources over heterogeneous environments.

In particular, the research work demonstrates how device drivers for peripherals built for one environment may be utilised in another, without having to write new drivers or port from similar environments. The thesis presents Multi Agent Printing System (MAPS) framework that is an implementation of Agent based Service Oriented Architecture (ASOA) model. MAPS will demonstrate the use of peripheral on one operating system, but utilises its device drivers in another. The architecture presents two advantages of such a framework: It will now be possible to make use of printers in environments where manufacturer did not provide the device drivers. Secondly, the framework proposes the use of agent technology and exploits its benefits of flexibility and performance.

1.5 Expected Outcomes

The primary expectation from this research is to provide Unix and like users the ability to use low cost USB printing devices of their choice that are conventionally only supported on MS Windows, without having to develop separate device drivers for Unix. It is projected that deployment of agents for various inter platform communication and hoousekeeping jobs will provide flexibility, robustness and extensibility.

The aim of this project is to provide a heterogeneous operating environment for software and the ability to use cross platform resources such as printers, scanners and other peripheral devices. This will demand use of system software of the

incompatible operating environment. The aim is to provide Windows compatible (non postscript) printers in Sun Ray thin client environment by utilising resources in their native environments as shipped by manufacturers. This will enable use of devices in environments which have been thus far incompatible, without having to reverse engineer a device driver for every device.

This novel technique proposes the new MAPS, for printing systems, which utilises services in heterogeneous environments and makes them available in previously non compatible environments.

The MAPS technique uses a TCP/IP link between the two operating systems and convert Solaris based print signals into the device's proprietary signals in Windows environment rather than re-engineering the proprietary standards in Solaris. Once we have device specific signals generated, they can be sent back to Solaris to be forwarded to the corresponding device.

The conversion process could become complex when supporting a range of different devices from different manufacturers.

1.6 Structure of Thesis

The thesis consists of seven chapters and is organised as follows. Chapter 1 provides the introduction, aims and objectives and gives brief description of each chapter. Chapter 2 identifies the problems encountered by the Unix community when using low cost USB printers and other peripheral devices. It analyses the current solutions of porting and re-writing device drivers and efforts made by open source community to solve these problems. This chapter also discusses printing architectures in Unix and the significant USB standards. Chapter 3, "Heterogeneous Services" describes distributed computing and various technologies used in distributed systems by identifiying distributed software development approaches, service based models and heterogeneous computing. This also includes discussions on Sun Ray thin client and Solaris and how it differs from distributed systems. The chapter also discusses Sun

Ray deployment on Wide Area Networks (WAN) as tested in the "Community Computing Project". The chapter also introduces Agent base computing and Intelligent Agents.

Chapter 4, "Multi Agent based Service Oriented Architecture" presents the generic ASOA framework for services and its various components. It discusses the various agents invloved and their communication mechnism. Chapter 5 implements the ASOA framework for printing services and presents the MAPS printing architecture. This chapter discusses the various specifics of MAPS agents, their roles, deployment and communication. The deployment is carried out on Solaris and MS Windows machines.

Chapter 6 "Evaluation and Performance of MAPS" carries out qualitative and quantitative test on the MAPS implementation. It describes the test bed to carry out quantitative test of MAPS printing. Chapter 7 concludes the research work and sheds light on possible future extensions of this study.

There are also appendices which enclose supplement documents relevent to this research work. Appendix 1, presents walk through scenarios of use of MAPS implementation. These scenarios are based on the research work carried out on Sun Ray thin clients and Solaris and MS Windows environment. Appendix 2 is a copy of research paper titled "Using Windows Printer Drivers for Solaris Applications – An Application of Multiagent System" presented at the 10th Internation KES Conference in October 2006 at Bournemouth, UK. The paper presents the agent based print conversion process, which is part of this research work. Appendix 3 is a copy of "Sun Ray Thin Clients on Broadband Networks" report. It reports the feasibility study carried out to test Sun Ray thin clients on a Wide Area Network. The report is part of the "Community Computing Project" conceded in association with Sun Microsystems and ActewAGL, Canberra.

Chapter 2

Research Problem and analysis

2.1 Introduction

One of the most daunting, long-term barriers to establishing Advanced Engineering Environments (AEE) is the integration and portability of software tools for design and development across dissimilar operating systems, computer networks, and programming languages & governmental and corporate cultures. Applications deployed in heterogeneous environments quite often require solutions for interoperability; i.e., the ability of various systems to work together in a meaningful and coherent fashion.

2.2 The Research Problem

This research is aimed at a vary similar problem of utilising peripherals (digital cameras and printers) that are originally designed for a specific operating environment only. In particular, the aim is at utilising printers which are manufactured for Windows Operating System in a Sun Ray environment which runs on Solaris Operating System.

In recent times, the cost of hardware has decreased significantly with rapid increase of technological advancements. Printing has become cheaper and more readily available. However, most new printing devices are supported in Windows or at most in Apple Macintosh environments. Windows users have a large choice of printing

devices that they can pick from the market. Manufacturers such Hewlett-Packard, Canon, Lexmark, Dell and Brother have a large number of low cost desk jet and laser printers which are apt for home and small business users [19, 18, 26, 36]. But unfortunately, this is not true with most Unix (Linux or Solars) users. Most of such common printing devices are supported on various MS Windows operating versions and Macintosh, but have no or little support for Unix environments. Table 1 shows a comparision of some of most common Inkjet and Laser printers and their supported environments.

Currently (at the time of writing this thesis) a non-Windows user has to do much research before he / she can consider purchasing a printer that will be guaranteed to work on their system. This leaves out almost all the newer models and brands from ones consideration and a significant portion of the older or existing models.

The main cause of incompatibility is the absence of printer driver from the manufacturer.

2.2.1 Analysis of Device Drivers

A device driver, or a software driver is a specific type of software, developed to allow interaction with the hardware devices. This usually constitutes an interface for communicating with the device, through the specific computer bus or communications subsystem that the hardware is connected to, providing commands to and/or receiving data from the device, and on the other end, the requisite interfaces to the operating system and software applications [27].

Device drivers are specialised hardware dependent program, which is also operating system specific, that enables another program, typically an operating system or applications software package, to interact transparently with the given device. It

Brand	Canon PIXMA iP4200	HP Color LaserJet 2600n	HP LaserJet 1020	Canon i9950	Lexmark Z513
Image					
Printing Technology	Inkjet	Laser	Laser	Inkjet	Inkjet
Output type	Colour	Colour	Monochrome	Colour	Colour
Connectivity technology	USB 2.0	Ethernet, USB 2.0	USB 2.0	FireWire IEEE-1394, USB 2.0	USB 2.0
Media Capacity	300 pages	250 pages	150 pages	150 pages	
Cartridges	Black x 2, Cyan x 1, Magenta x 1, Yellow x 1	Black x 1, Cyan x 1, Magenta x 1, Yellow x 1			
Duplex printing	Duplex printing supported	Without duplex printing	Without duplex printing	Without duplex printing	Without duplex printing
OS required	Apple MacOS X 10.2.1, Microsoft Windows 2000, Microsoft Windows ME, Microsoft Windows XP, Microsoft Windows 98	Apple MacOS X 10.2, Microsoft Windows 2000, Microsoft Windows 98 SE, Microsoft Windows ME, Microsoft Windows XP	Microsoft Windows 2000, Microsoft Windows 98 SE, Microsoft Windows ME, Microsoft Windows XP	Apple MacOS 8.6, Apple MacOS 9.0, Apple MacOS X 10.2.1, Microsoft Windows 98 SE, Microsoft Windows XP	Microsoft Windows 2000, Microsoft Windows ME, Microsoft Windows XP, Microsoft Windows 98

Table 1. Printer features comparision [84].

usually provides the requisite interrupt handling required for any necessary asynchronous time-dependent hardware interfacing needs [27].

The key design goal of device drivers is abstraction. This layer of abstraction exists between the Kernal and the hardware. A device driver seperates specific hardware characterstics from the operating system implementaton. In Microsoft Windows, programs do not send commands directly to the priner. Instead, they send commands to an abstract layer managed by the operating system and the driver program (the device driver) transforms the the abstract command to one or more commands acceptable by the actual hardware.

A device driver has three sides: one side talks to the rest of the kernel, one talks to the hardware, and one talks to the user. Figure 2 shows the interaction between programs, Operating System, device drivers and the actual hardware.

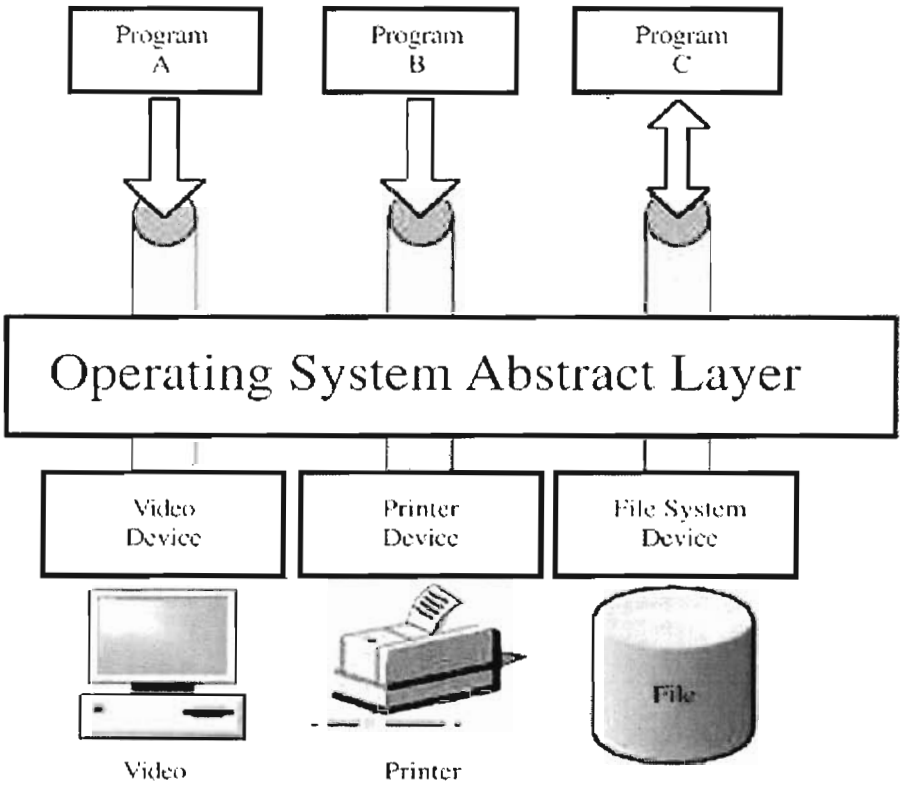


Figure: 2. Device Drivers and their access [53].

2.2.2 Device driver philosophy

Every model of hardware (even within the same class of device) is different. Newer models that provide more reliable or better performance are often controlled differently.

Computers and their operating systems cannot be expected to know how to control every device. This is simply because the operating system writers are not the people who manufactured the devices, and may have no or little knowledge of its functioning. To solve this problem, operating systems essentially dictate how every type of device should be controlled. The function of the device driver is to translate these OS mandated function calls into device specific calls [66].

Depending on the specific computer architecture, drivers can be 8-bit, 16-bit, 32-bit, and more recently, 64-bit. This corresponds directly to the architecture of the operating system for which those drivers were developed. For example, in 16-bit Windows 3.11, most drivers were 16-bits, while most drivers for 32-bit Windows XP are 32-bit. More recently, specific 64-bit Linux and Windows versions have required hardware vendors to provide newer 64-bit drivers for their devices.

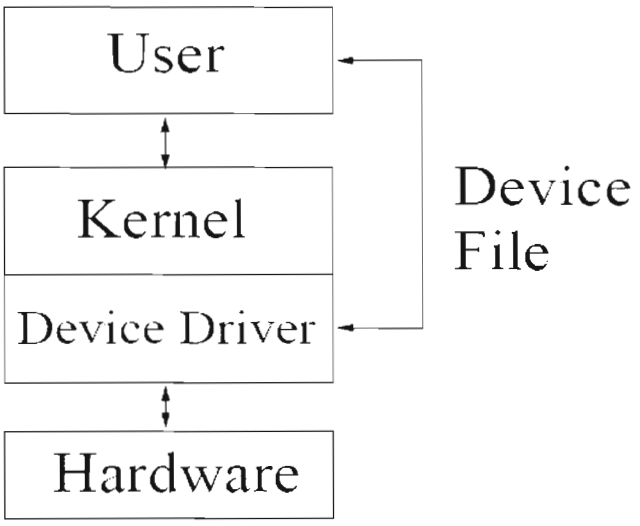


Figure: 3. Interface of a device driver [53].

Printing in a Sun Rays ultra thin client environment is limited to printers which are Postscript complient. Such printers are usually laser printers designed for heavy industrial printing, and hence bear higher costs.

In order to talk to the kernel, the driver registers with subsystems to respond to events. Such an event might be the opening of a file, a page fault, the plugging in of a new USB device.

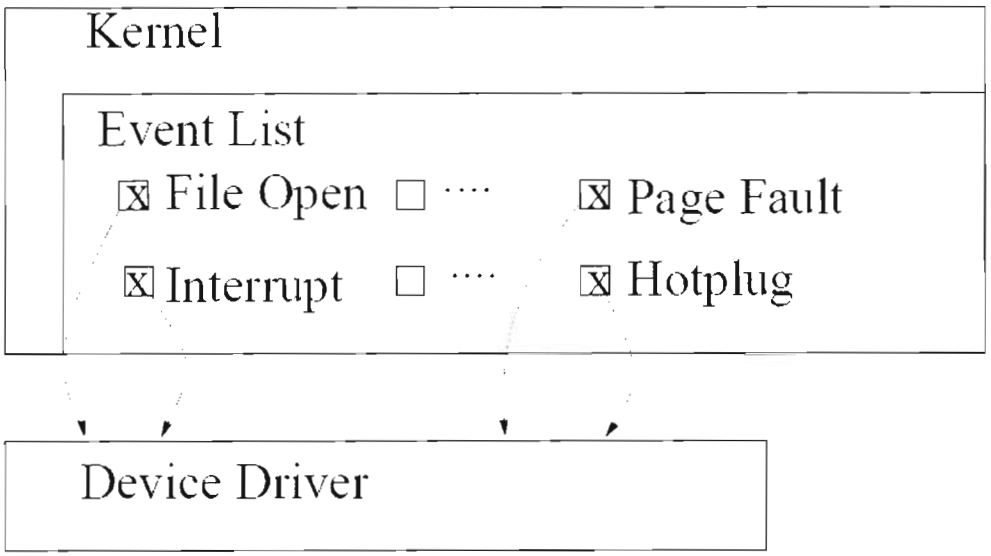


Figure: 4. Kernal Interface of a Device Driver

In UNIX everything is a file, and users talk with device drivers through *device files*. Device files are a mechanism, supplied by the kernel, precisely for this direct User-Driver interface. The other common kind of device file is a block device file. These device files are generally located at /dev/.. folder in the Unix file system [53].

2.2.3 USB – Universal Serial Bus

USB (Universal Serial Bus) is designed to connect peripherals such as mice, keyboards, scanners, digital cameras, printers, hard disks, and networking components to PC [11, 59].

Most operating system supports USB, therefore the installation of the device drivers is fast and simple. USB devices are easy compared to other ways of connecting devices to computer such as parallel ports, serial ports and special cards that are installed inside the computer's case.

USB interface provides a single, consistent, user-friendly way to connect up to 127 devices to a computer [36]. They are "plug and play" standard adopted by hardware manufacturers. Like serial and parallel "buses", it "transfers" data to and from the computer. USB has evolved with time and has several version numbers. Version 2.0 transfers data 40 times faster than version 1.1 [39].

2.2.4 USB pinout signals

USB is a serial bus. It uses four shielded wires- two for power (+5v & GND) and two for differential data signals labeled as D+ and D- in pinout. NRZI (Non Return to Zero Invert) encoding scheme is used to send data with a sync field to synchronies the host and receiver clocks. In USB data cable, Data+ and Data- signals are transmitted on a twisted pair. No termination is needed [36].

Pin	Name	Cable color	Description
1	VCC	Red	+5 VDC
2	D-	White	Data -
3	D+	Green	Data +
4	GND	Black	Ground

Figure: 5. USB pinout

The USB pinout is similar for type A or B connector. The difference is in the shape [78].

2.2.5 USB Cable Assemblies and Adaptors

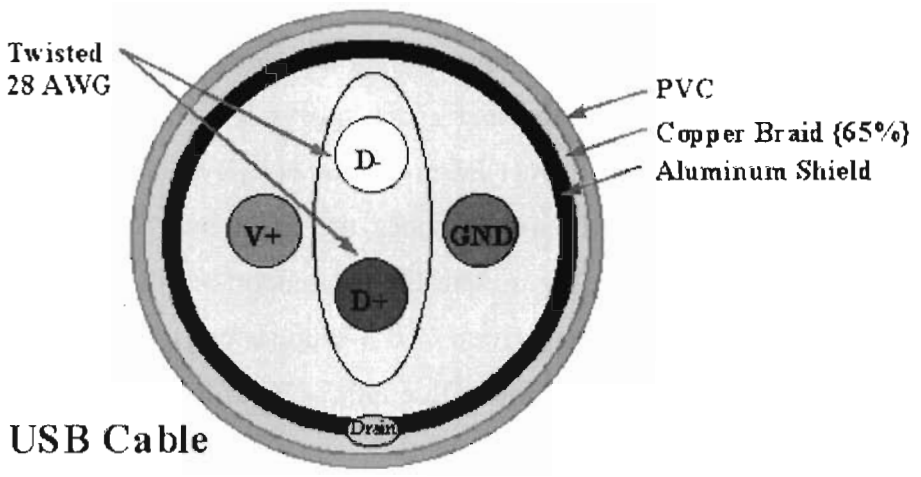


Figure: 6. USB Cable cross section

USB cable has 4 wires. The data wires are 28 AWG (American Wire Gauge), the power wires are 20 to 28 AWG (American Wire Gauge). Two power lines are un-twisted and 2 data lines are twisted. Longer cables use 20 AWG for power. The wires are color coded as shown in Figure 6. Cables have an A plug on one end and a B plug on the opposite end. The maximum length of each cable is upto 5 meters and can be increased by using USB Hubs. The USB interface is designed to operate down to -20 degrees C [78].

2.3 Investigation of available solutions

2.3.1 UNIX Native Printing Architecture

Solaris, like most other Unix systems uses Common Unix Printing System (CUPS) for its printing services. CUPS provides a portable printing layer for its operating systems. It is developed and maintained by Easy Software Products [28] to promote a

standard printing solution and is the standard printing system in MacOS X and most Linux distributions [24].

CUPS uses the Internet Printing Protocol (IPP) as the basis for managing print jobs and queues and adds network printer browsing and PostScript Printer Description (PPD) based printing options to support real-world printing. CUPS consists of a Unix print spooler and scheduler, a filter system that converts the print data to a format that the printer will understand, and a backend system that sends this data to the print device. CUPS uses the Internet Printing Protocol (IPP) as the basis for managing print jobs and queues. It also provides the traditional System V and Berkeley command line interfaces, along with limited support for the Server Message Block (SMB) protocol. The device drivers CUPS supplies can be configured by using text files in Adobe's PostScript Printer Description (PPD) format. There are a number of user interfaces for different platforms that can configure CUPS, and it has a built-in web-based interface. CUPS is provided under the GNU General Public License and GNU Lesser General Public License [25].

2.3.2 Open source solutions

The problem of utilising devices manufactured for one environment in another was initially looked at in conjunction with available solutions. UNIX experts recommend specific brands and / or models of printers for specific UNIX flavors [57] simply because a large portion of low cost (Inkjet or Laser) printers are not shipped with drivers for UNIX and like systems, and have little support for Operating Systems other than Windows. Linuxprinting.org has a comprehensive list of what printers work with what versions of UNIX. However it is evident that the newer models of printing devices take considerable amount of time till someone has worked out a compatible UNIX companion or have reengineered a device driver and made available to the open source community.

As part of the experiment, an effort was made to tweak and recompile existing Linux USB printer drivers for Sun Ray and Solaris environments.

2.3.3 Using compatible or closely compatible drivers

LinuxPrinting.org is a significant effort by the open source community for providing printing help in Linux and similar environments. A HP 540C Inkejet printer was successfully installed on Linux platform using the following steps:

a) Finding and installing the driver.

The printers page in the database was located and it was found that HP 540C InkJet printer is not a PostScript printer and would require a separate driver. A quick check on the database reveiled that the driver **hpijs** is recommended for this printer and is provided by HP. The PPD file was also available for use.

b) Downloading and installing the PPD file

The PPD file for HP 540C (HP-DeskJet_540C-hpijs.ppd) was downloaded and copied to the directory ***/usr/share/cups/model/***

c) Restart CUPS deamon

With ***root*** credentials, the CUPS deamon was restarted using the following commands:

```
killall -HUP cupsd
/etc/init.d/cups restart
/etc/software/init.d/cups restart
```

d) Applying CUPS filter using Foomatics

Foomatic-rip and foomatic-gswrapper is downloaded as below:

```
cd /usr/bin

wget http://www.linuxprinting.org/foomatic-rip

wget http://www.linuxprinting.org/foomatic-gswrapper

chmod 755 foomatic-rip foomatic-gswrapper

ln -s /usr/bin/foomatic-rip /usr/lib/cups/filter/foomatic-rip
```

e) **Configuring CUPS**

A new printer was added using the "Add printer wizard" at ***http://localhost:631/admin***

f) **Adjust the print margins.**

Files ***align.ps*** and ***alignmargins*** were downloaded and ***alignmargins*** executed as ***root*** as below:

```
cd /tmp

wget wttp://www.linuxprinting.org/download/printing/align.ps

wget
http://www.linuxprinting.org/download/printing/alignmargins

chmod 755 alignmargins

su

./alignmargins
```

g) Installing beh - The Backend Error Handler

This makes the behaviour of CUPS on errors during the communication with the printer (e. g. printer turned off) configurable instead of CUPS simply disabling the queue. The ***beh*** script is downloaded and copied to the ***/usr/lib/cups/backend/*** directory. The beh script is made executable and CUPS restarted as below:

```
chmod 755 beh

killall -HUP cupsd
```

h) Make print queue using beh

```
lpadmin -p hp450c -E -v beh:/1/0/5//dev/rabinow-lj4m
```

After this step the printer was configured successfully and ready for use. A test page was printed using the following command line:

```
lp -d foo1 -o PageSize=Letter /etc/motd
```

The above test was carried out for printer was tested and rated to be working "Perfectly" with Linux. However, ther are printers which have either no support or are rated as working "Partially" or as "Paperweight", for eg, Canons BJC-5100 and BJC-8500.

2.3.4 Developing USB drivers

In order to test the support and resources available to develop and tweak new drivers for USB devices, a digital camera driver was developed for Solaris. To be able to carry on this development process, indepth background information on USB architecutre and Solaris LibUSB device libraries was researched.

2.3.4.1 Sun Ray USB Architecture

One of the unique things about Unix and similar operating system is that it regards everything as a file. These files are of three categories; ordinary or plain files, directories, and special or device files. Device files include things such as disks, CD-ROMs, tapes, terminals, serial ports, and sound cards. All files contain data of some kind.

In Sun Ray architecture, each terminal's file systems are mounted under a separate node. Usually this node is denoted by the MAC (physical) address of the Sun Ray terminal. All USB ports are mounted under this node for each Sun Ray.
Currently, there are two ways to access a USB device on a Sun Ray; through Kernel level USB drivers or by Application level drivers. Kernel drivers provide basic I/O operations to devices and are generic. Application drivers may be written and designed for specific device and are more specific.

One of the research efforts was to port Linux GPhoto 2.0 digital camera drivers to Sun Rays using Sun's LibUSB interface which is discussed in the next section.

2.3.5 LibUSB Applications

LibUSB is an open sourced software project, maintained at libusb.sourceforge.net. The aim of this library is to provide user level applications access to USB devices regardless of the Operating system [45]. Most application level USB drivers on Linux are written utilising LibUSB and it seems to be becoming a de-facto standard in the Linux world. This API is the lowest level of software interacting with the OS and provides device handles to applications / drivers running above this layer. Figure

7 illustrates the different layers of software interacting at different levels. LibUSB works for both USB and parallel interfaces, however, in Sun Ray architecture we are only interested in USB ports [45].

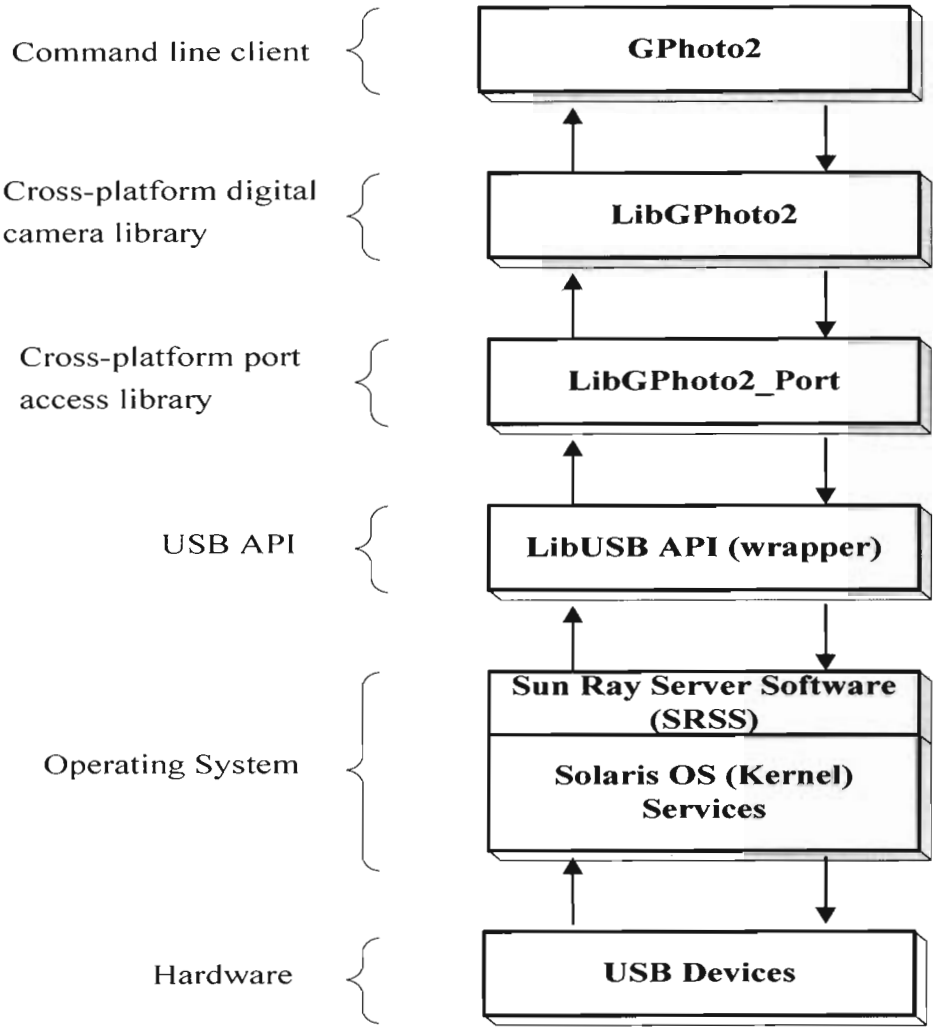


Figure: 7. LibUSB and GPhoto architecture

Sun Rays having a significantly different architecture require a separate version of LibUSB, which is currently being developed by Sun Microsystems. The pre-release version of LibUSB for Sun Ray was used in successfully porting GPhoto drivers to Sun Ray.

Since LibUSB for Sun Rays have same interface as that of Linux and open sourced operating systems, most applications running on Linux utilising LibUSB, in theory, can also be ported to Sun Rays.

Figure 7 shows different layers of applications compiled over LibUSB for Sun Rays. GPhoto 2 is a well-known open sourced digital camera driver and has been shipped with various flavors of Linux.

2.4 Summary

A device driver is a specific type of software, developed to allow interaction with the hardware device. They provide abstraction between the OS kernel and the hardware. These are specialised hardware dependent programs that are also OS specific. Most printers are shipped with MS Windows and Mac OS device drivers. Systems such as Linux, Solaris (and other Unix like) OSs are usually not supported. To make use of such printers in a Unix like environment some effort has been made by the open source community to port or tweak drivers for Linux. Sun Rays running on Solaris too have had some success in overcoming this problem using similar techniques by porting other drivers or using libraries such as LibUSB.

Chapter 3

Heterogeneous Services

3.1 Introduction

With computational power being dirtributed to daily use appliances, the commulative work of such intelligent appliances have a potential to solve more complex problems. This research work makes use of this distributed technique to solve problems. This chapter looks at some existing distributed technologies and their uses. A comparision of thin clients against fat intelligent clients shows advantages of one on the other. It also presents the concept of agent based computation and its ammulgamation with heterogeneous systems.

This chapter discusses possibilities and technologies relevant to utilise distributed services in order to solve the problem of deficient device drivers.

3.2 Distributed Computing – Client Server architecture

A Client Server architecture may be defined as a network architecture in which each computer or process on the network is either a client or a server. Servers are powerful computers or processes dedicated to managing disk drives (file servers), printers (print servers), or network traffic (network servers). Clients are PCs or workstations on which users run applications. Clients rely on servers for resources, such as files, devices, and even processing power [21].

In such an environment, application processing is divided between client workstation and servers. It implies the use of desktop computers interacting with servers in a network in contrast to processing everything in a large centralized mainframe [20].

The Client Server approach introduced a database server to replace the file server. Using a relational database management system (DBMS), user queries could be answered directly. The Client Server architecture reduced network traffic by providing a query response rather than total file transfer. It improves multi-user updating through a GUI front end to a shared database. Remote Procedure Calls (RPCs) or standard query language (SQL) statements are typically used to communicate between the client and server [67].

3.3 Thin Clients versus Fat Clients

Thin and fat clients in Client Server architecture may be defined by the use of tiers. Tiers are used to describe the logical partitioning of an application across clients and servers. In two-tier architecture, processing load is split into two. Majority of the application logic runs on the client, which typically sends SQL requests to a server – resident database. This is fat client architecture, because a chunk of the application runs on the client side. 3 tiers splits the processing load between the clients that run the Graphical User Interface logic, application server (running logic) and the database or legacy application. Three-tier moves the application logic to the server – called the fat server or thin client [68].

Thin client is the hardware at the client end in a client - server model. This client is designed to be small and light so that bulk of the data processing is carried out at the server. The term 'thin client' is becoming increasingly popular. It inhabits the territory owned by Netscape and Sun Microsystems promoting Java based applications running on thin clients on networked computers. On the other side, the territory owned by Microsoft and Intel, is building even larger applications running locally on the desktop computers.

The term 'thin client' is also used for software, but is more often used for computers such as networked computers, which are designed to serve as the client for client / server architecture. A thin client is a computer without a hard disk, and even memory and a microprocessor, whereas a fat client includes disk drive and has all the hardware needed to do bulk processing locally. However, the data itself is stored on the servers dedicated to provide file storage services [80].

Most enterprises deploy thin client architecture to reduce the Total Cost of Ownership (TCO). By centralising services at a remote server, administrative tasks are accessible locally and are carried out more efficiently.

3.4 Heterogeneous Computing

3.4.1 Distributed Object Systems

The first model of distributed programming is the message-passing model [47]. The idea is very simple and literal: if one person wishes to communicate with another, it does so by simply sending a message, a packet on the network, to the other program. Message passing systems form the core of all networked systems. The Internet itself is a large message passing system, with extensions such as long-lived connections and address resolutions. Many Internet applications such FTP, The Web or Email are based on simple message passing.

Remote procedural call (RPC) was the first successful abstraction on top of message passing architectures [54]. If message-passing systems are analogues to assembly language programming, RPC is like procedural programming, adding the abstraction of the function call to distributed systems. In RPC, if a program wants to communicate with a program on another computer, it just calls a function on that other computer machine, much like it would call one on its own machine. RPC adds "Transparency" to distributed programming, the illusion that talking to a remote program is not any different than talking to a local program.

With applications distributing, there arises a need for Middleware. Middleware is connectivity software that consists of a set of enabling services that allow multiple processes running on one or more machines, possibly on different platforms, to interact across a network. NFS, the usual Unix Network File System, is the most commonly known RPC based application.

Distributed Object Systems work similar to RPC, only with an Object Oriented Abstraction on top of procedural calls. Instead of calling a fixed method name, distributed object systems gives program references to remote objects, allowing the program to manipulate, call methods, and store the remote object just as a local object. In essence, distributed object add to the concept of function from RPC.

Distributed objects are the current state-of-the-art in building distributed systems. The major standard is the OMG CORBA, a language neutral specification for communicating object systems [56]. Competitors to CORBA include MS DCOM architecture [13] and the various distributed object systems layered on top of Java [65]. Some of the most common Middleware components are CORBA, COM and SOAP [17].

3.4.2 CORBA: Common Object Request Broker

Object Request Broker (ORB) technology promotes the goal of object communication across machine, software, and vendor boundaries. The relevant functions of an ORB technology are

- Interface definition
- Location and possible activation of remote objects
- Communication between clients and object.

An object request broker acts as a kind of telephone directory. It provides a list of services to the clients and helps establish connections between clients and these services [72]. Figure 8 illustrates some of the key ideas.

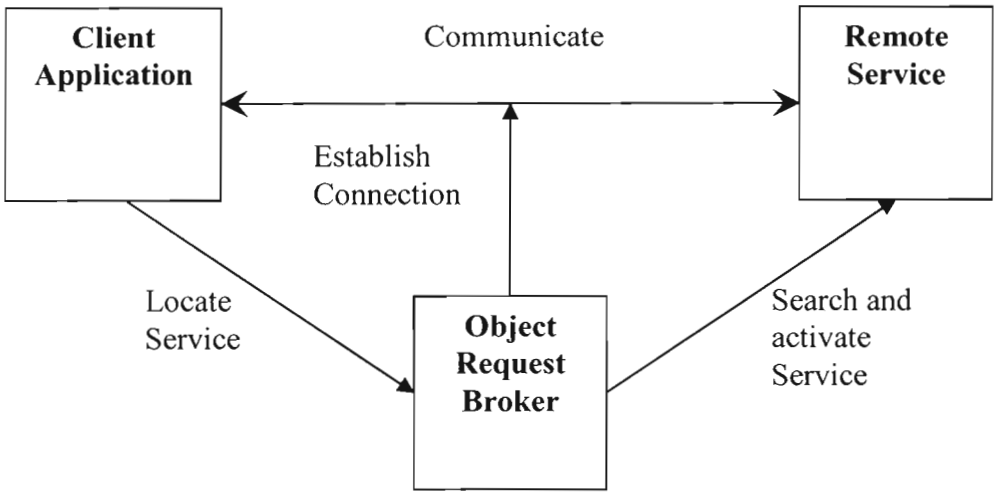


Figure: 8. Object Request Broker

The next technical step toward object system interoperability is the communication of objects across platforms. An ORB allows objects to hide their implementation details from clients. This can include programming language, operating system, host hardware, and object location. Each of these can be thought of as a "transparency," the same and different ORB technologies may choose to support different transparencies, thus extending the benefits of object orientation across platforms and communication channels [12].

3.4.3 COM: Component Object Model Technologies

COM refers to both a specification and implementation developed by Microsoft Corporation that provides a framework for integrating components. This framework supports interoperability and reusability of distributed objects by allowing developers to build systems by assembling reusable components from different vendors which communicate via COM. Microsoft COM (Component Object Model) technology in the Microsoft Windows-family of Operating Systems enables software components to communicate. COM is used by developers to create re-usable software components, link components together to build applications, and take advantage of Windows services. The family of COM technologies includes COM+, Distributed COM (DCOM) and ActiveX Controls.

The Microsoft Distributed Component Object Model (DCOM) extends the Component Object Model (COM) to support communication among objects on different computers—on a local area network (LAN), a wide area network (WAN), or even the Internet. With DCOM, your application can be distributed at locations that make the most sense to your customer and to the application.

COM+ brought together the technology of COM components and the application host of Microsoft Transaction Server (MTS). COM+ automatically handles difficult programming tasks such as resource pooling, disconnected applications, event publication and subscription and distributed transactions. COM+ infrastructure also provides services to .NET developers and applications [23].

3.4.4 SOAP: Simple Object Access Protocol

SOAP, or Simple Object Access Protocol is an XML-based object invocation protocol. SOAP was originally developed for distributed applications to communicate over HTTP and through corporate firewalls. SOAP defines the use of XML and HTTP to access services, objects and servers in a platform-independent manner.

SOAP defines a simple mechanism for expressing application semantics by providing a modular packaging model and encoding mechanisms for encoding data within modules. This allows SOAP to be used in a large variety of systems ranging from messaging systems to RPC.

SOAP consists of three parts:

The SOAP envelope construct defines an overall framework for expressing what is in a message; who should deal with it, and whether it is optional or mandatory.

The SOAP encoding rules defines a serialization mechanism that can be used to exchange instances of application-defined data types.

The SOAP RPC representation defines a convention that can be used to represent remote procedure calls and responses [70].

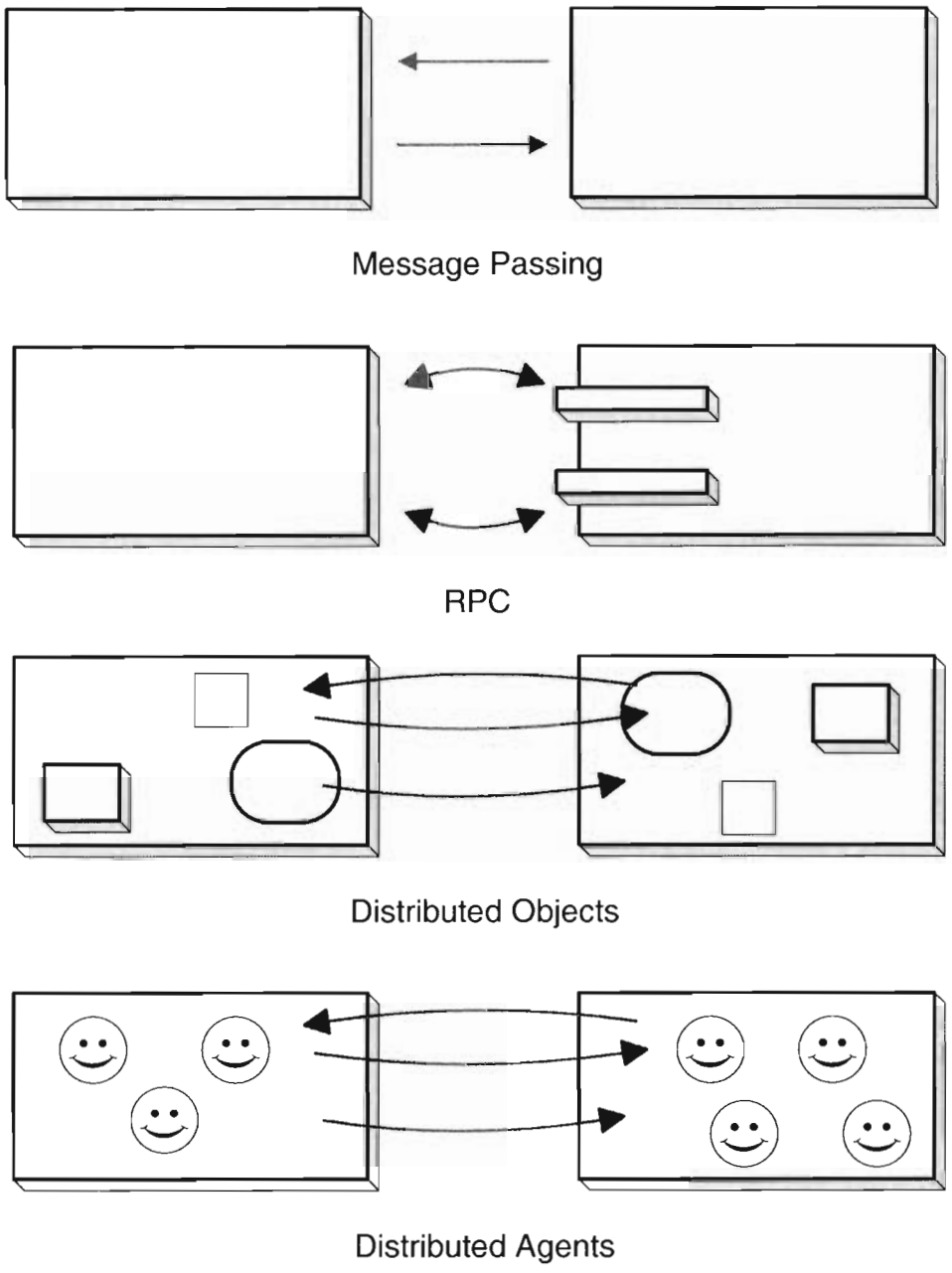


Figure: 9. Distributed Systems Models

3.4.5 Distributed Agents

Distributed objects are a good way to describe things on the network, the sources and capabilities that are out there and available, however distributed object systems are mostly static; they do not say much about activity on the network. To address this, a concept of process needs to be added to distributed objects. Distributed agents are about autonomy to objects, adding agency, to make them active participants on the network. Figure 9 characterises message passing, RPC, distributed objects and distributed agents.

Software agent research is very open and diverse. Almost anything that is a program has been called an "agent", from web robots that filter information to software help systems to large artificial intelligence constructions. If there is a consensus in software agent research, it is that agents that are autonomous, proactive and adaptive [9]. Agents are discussed in detail in the next section.

3.5 Computing in a Sun Ray environment

Sun Ray ultra thin client consists of monitor, keyboard, mouse, a built-in smart card reader, 10/100 Base-T network port, an embedded audio speakers and USB ports to support keyboard, mouse and other peripherals. These provide users with access to all the applications and utilities they normally use on their workstations or PCs. The actual computing, however, is performed on one or more remote servers. The physical desktop unit - the thin client itself - needs only enough memory and computing power to recognise keystrokes and mouse events and to display pixel data received from the server. No computing is performed locally [42].

Sun Ray is simply an input device, taking keyboard and mouse actions and transmitting them across the network to the Solaris server. The server then transmits back the screen pixels, including mouse movements, characters typed, and any audio back to the Sun Ray .

Performing all CPU intensive computing tasks on a large server allows a great deal of work to be done with considerably greater efficiency - in terms of time, money and hardware vis-à-vis on a multiplicity of individual workstations or PCs at the same time [42].

Absence of application files, operating system, and storage media on the client defines the stateless nature – which also means that the desktop does not have to be administered. Users are administered centrally at the server. Administrators are freed from physically maintaining individual clients and performing repetitive task of setting up each client machine [42].

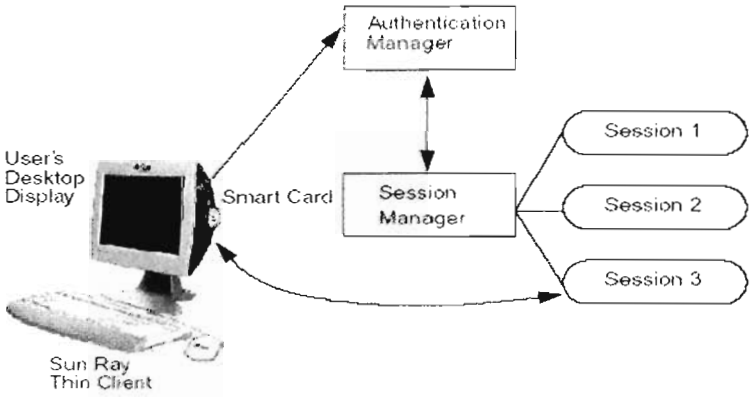


Figure: 10. Sun Ray Session Management

Remote processing allows all sessions to be run at a central server, which maintains and authenticates session logins. Smart card allows hot swapping of sessions between terminals. Session Manager simply switches the session from one terminal to another and provides true mobility to users. Figure 10 shows Sun Ray session management. Sun Rays on a Wide Area Network can totally eradicate the necessity of laptops and users may now carry only a smart card for hot docking! When combined with a separate Microsoft Terminal Server with Citrix MetaFrame, along with Sun Ray's support of the Citrix ICA software client, the Sun Ray can also run Windows applications [6, 86].

3.5.1 Sun Ray thin client and Solaris

The key of this thin client architecture lies in the Sun Ray Server Software (SRSS). This layer of software runs on the Solaris OS and is responsible for transmitting screen refreshes, keyboard and mouse movements to the clients. Figure 11 shows Sun Ray software layer on Solaris Operating System. The client sends back the mouse movements and keyboard strokes to the Server, which are then processed at the server side.

The Sun Ray Server Software 1.2 is compatible with both 32 and 64 bit applications of Solaris 8 operating environments. Common Desktop Environment (CDE) is pre-bundled with the system; however it also supports KDE, Java Desktop and other common desktops.

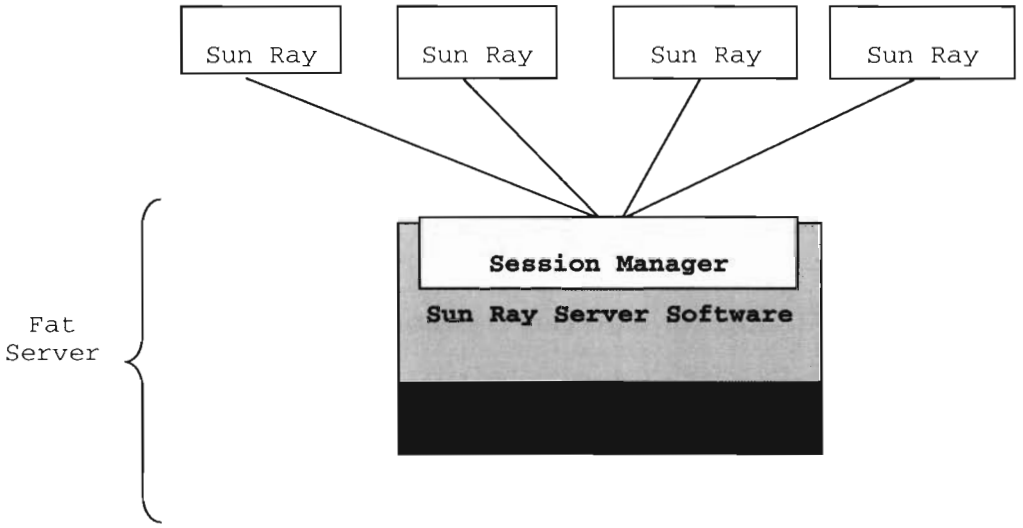


Figure: 11. SRSS and Solaris

3.5.2 Sun Ray on Wide Area Network

Sun Rays are designed to work best on a Local Area Network; however, with little innovation and tweaking, they have been tested to work efficiently on a Wide Area Network too.

The results in Figures 12 and 13 plot the web data transferred and latencies of X, ICA, RDP AIP, VNC and Sun Ray thin clients. These results were conducted using the Slow Motion Bench Marking [85]. The research conducted by University of Columbia shows that using thin client computing in a WAN environment can deliver acceptable performance, even when client and server are located thousands of miles apart on opposite ends of the country. Sun Rays have shown to deliver excellent performance on all application benchmarks. Although, performance varies widely among thin-client platforms and not all platforms are suitable for WAN environment.

Measurements have shown that the bandwidth efficiency of a thin-client system is not a good predictor of performance over broadband network. It is shown that Citrix ICA and Microsoft RDP usually transfer less data overall for each benchmark compared to the other systems while Sun Ray typically transferred the most amount of data overall for each benchmark. However, in terms of user-perceived performance, Sun Ray significantly outperformed both ICA and RDP over broadband. [44, 55, 85]

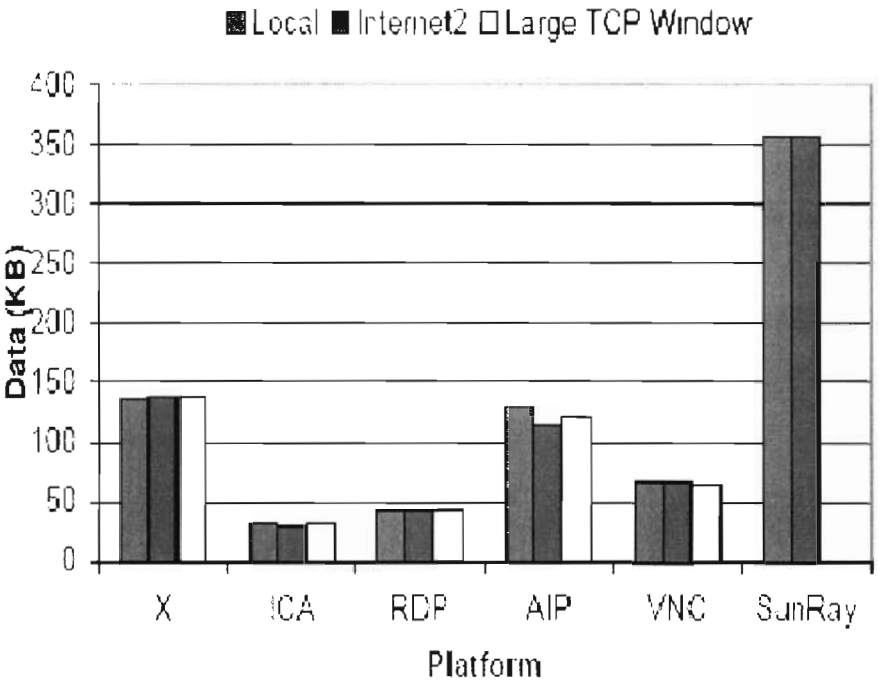


Figure: 12. Comparative Analysis of Web Transfer in Thin Clients [44].

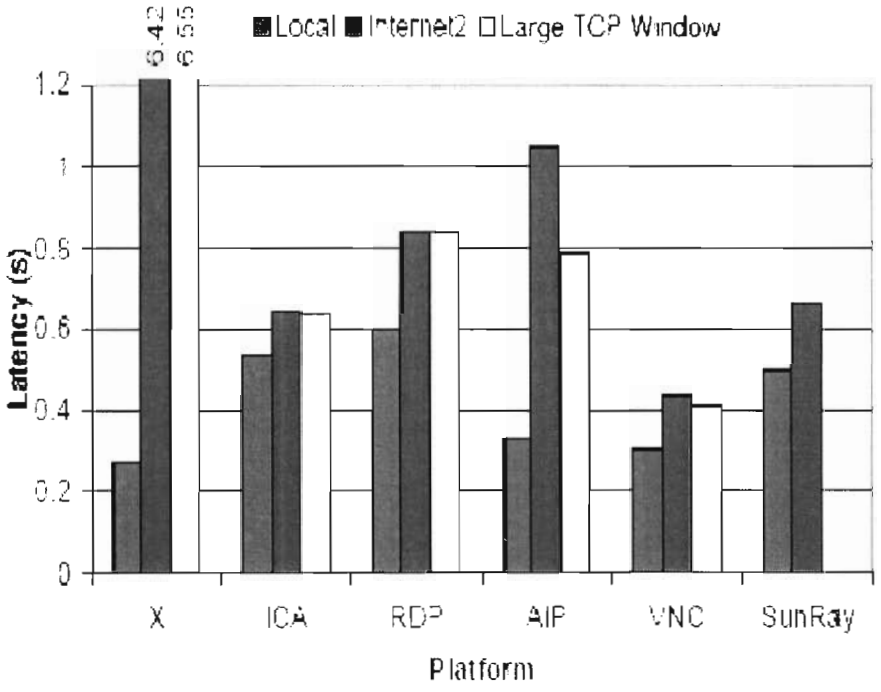


Figure: 13. Comparative Analysis of Latency in Thin Clients [44].

We had also conducted a test of "Proof of Technology" outside laboratory environments by setting up a Sun V20 server at TransACT House telecommunication facility which was given a static IP address. Sun Ray Server Software was installed on the server and clients were connected at ActewAGL House through an ADSL connection over 1 Mbps connection. Figure 14 shows the setup.

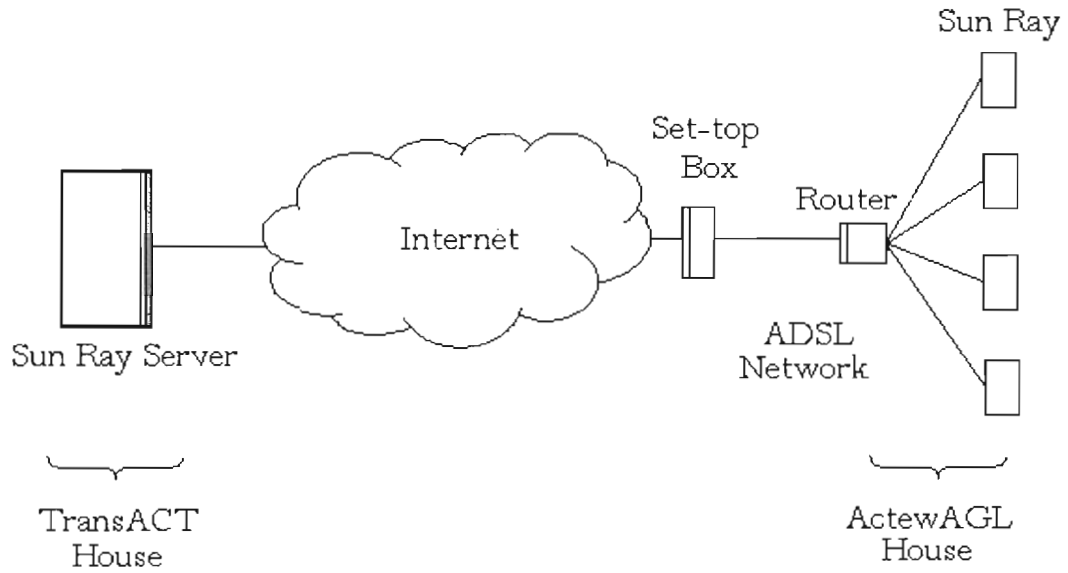


Figure: 14. Sun Rays on Wide Area Network

The main concern on a broadband network is for graphic intensive applications, which require large screen refreshes sent to the clients. On a slow network, (256Mbps or lower) the efficiency deteriorates, while on faster nets (1000 Mbps or higher) the performance shoots up. Since clients have no compression software in their firmware, heavy applications dominate the network bandwidth forfeiting the performance. However with high speed networks become increasingly available, use of thin clients over the WAN becomes more and more beneficial.

Use of Sun Rays over broadband network is currently not designed for CAD/CAM or Multimedia applications. Most home users and schools however would be most suited for this architecture.

3.6 Agent based computing

3.6.1 Historical Context

There is universally accepted definition of an agent [10]. Russel and Norveg (1995) define an agent as an entity that can be viewed as perceiving its environment through sensors and acting upon its environment through effectors. They may also be defined as computer systems to which one can delegate tasks.

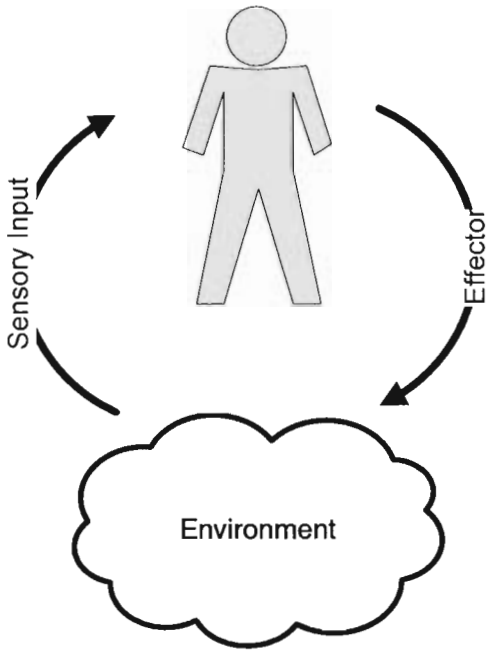


Figure: 15. Agent interacting with its environment

They differ from conventional software in that they are long-lived, semi-autonomous, proactive and adaptive [50]. Artificial Intelligence (AI) and agent systems have been closely related over the last thirty years. AI is interested in studying the components of intelligence (e.g. the ability to learn, plan) while the study of agents deals with integrating the same components. This distinction however does not imply that all the problems within AI must be solved in order to build an agent [10]. Typically, and agent is defined as entity that can be viewed as

perceiving its environment through sensors and acting upon its environment through effectors [22].

3.6.2 Multi Agent Systems

As agent technologies matured, it broadened its goals to developed and implemented multi agent systems (MAS) to attack more complex, realistic and large-scale problems which are beyond the capabilities of an individual agent [10]. By forming communities of agents, a solution based on modular design can be implemented where each member of the agency specialises in solving a particular aspect of the problem. When agents are deployed in heterogeneous environment, their skills are perfected for a specific aspect of the problem in an environment. Often, some agents may be generic enough to be deployed in multiple environments, however for more complex task they are written exclusively to operate in a pre-defined environment. Figure 16 illustrates a simple agent brokers operating in a heterogeneous environment.

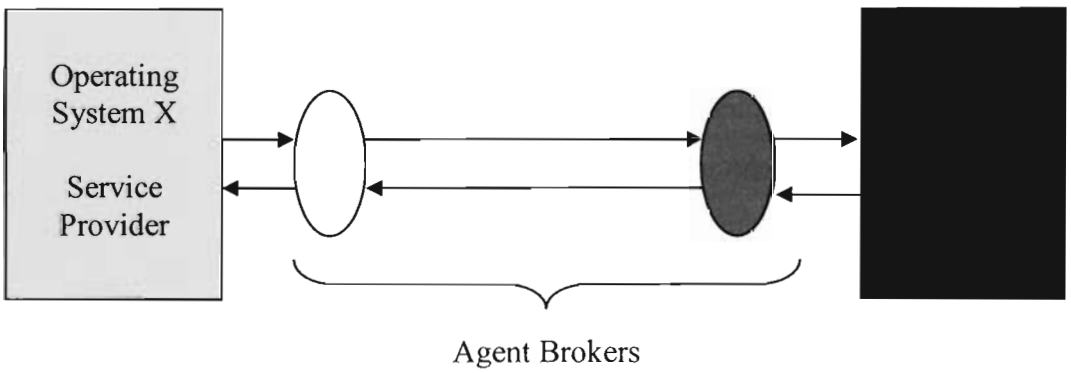


Figure: 16. Translator Agent in Heterogeneous Environment

By distributing the work load into MAS, agents can be used to solve problems that are too large for a centralized agent to solve because of resource limitations and/or to avoid a one point bottleneck or failure point. Secondly, to keep pace with business needs legacy systems, which may not be able to be rewritten due to a combination of

cost, time, and technical know-how, can be made to interoperate with other agents in an agent society by building an agent wrapper around them [31].

Multi-agent system is a system composed of several agents, collectively capable of reaching goals that are difficult to achieve by an individual agent or monolithic system. The exact nature of the agents may differ from an implementation to another. They are often claimed to be autonomous. For example a household floor cleaning robot can be autonomous in that it is dependent only on a human operator to start it up. On the other hand, in practice, all agents are under active human supervision. Furthermore, the more important the activities of the agent are to humans, the more supervision that they receive. Multi-agent systems can manifest self-organization and complex behaviors even when the individual strategies of all their agents are simple. Multi agent systems can benefit from one or more of the following:

- beliefs, desires, and intentions (BDI),
- cooperation and coordination,
- organisation,
- communication,
- negotiation,
- distributed problem solving,
- multi agent learning,
- scientific communities,
- dependability and fault-tolerance

Some advantages of MAS have been published as below:

MAS enhances performance in the following areas: (1) computational efficiency through concurrency, (2) reliability via redundancy, (3) extensibility of agents by changing the number and capabilities of the agents, (4) maintainability via modularity and (5) reuse of agents in different agencies to solve different problems [31].

Collective intelligence from distributed agents has proven to improve the reasoning behaviour in many real world problems that have distributed problem solving characteristics intrinsically [69].

3.7 Intelligent Agents

An intelligent agent is defined as "an entity reacting to and learning from its environment, and capable of planning to achieve its goals for the benefit of its client"[33, 34, 51]. The idea of intelligent agent has captured a large scale attention and many computer scientists have taken up the challenge of building and deploying Intelligent Agents in various problems. Intelligent Agents are capable of flexible autonomous action to meet its design objectives. Flexibility of these agents include:

Reactivity: Intelligent agents perceive and react in a timely fashion to changes that occur in their environment in order to satisfy their design objective. Agent's goal and/or assumptions that form a procedure that is currently executing may be affected by a changed environment and a different set of actions may be need to be performed.

Pro-activeness: Reacting to an environment by mapping a stimulus into a set of responses is not enough. They posses goal directed behavior.

Social Ability: Intelligent agents are capable of interacting with other agents (and possibly humans) through negotiation and corporation.

Researchers have used intelligent agents to solve complex problems of human gene mapping [2] and in space missions by NASA [64].

3.8 Summary

A number of distributed technologies are available. These form a mix of homogenous and heterogenous systems. Sun Ray ultra thin clients have been tested to work over xDSL network and have been found to be most suitable compared to ICA, RDP, X and other thin clients on WAN.

Technolgies such as SOAP, CORBA and Distributed Agent technologies can be designed and developed in true heterogenous environments. This also opens possibilities to utilise resources (services) of one OS in another. Multi Agent can be deployed on systems spanning more than one OS and can allow cross platform resource sharing. Collective intelligence from distributed agents has proven to improve the reasoning behaviour in many real world problems that have distributed problem solving characteristics intrinsically. Multi agents can be deployed to provide cross platform services.

Chapter 4

Multi Agent based Service Oriented Architecture

4.1 Introduction

This research presents Agent based Service Oriented Architecture (ASOA) which is a multi agent based generic architecture that is designed to provide services in an heterogeneous environment. This study puts together agent technology and its flexibility to device this generic framework which presents mechanisms to provide services to cross platform applications. It also shows how these services can be utilised. The ASOA framework provides communication protocols and demonstrates how services can be requested and utilised over a mixed or standalone environment. The next section describes agent distribution and communication used in ASOA.

4.2 ASOA Agent Distribution and Communication

Agents described in previous chapter interact with other agents. However, the design of these systems is concerned with the individual agents only. The macrolevel of communication on the other hand, considers a multiagent system from a holistic perspective, where interaction, coordination, and cooperation between agents are designed in advance of run time. The concern here is with the global system structure

and mechanisms to support interaction, including agent communication protocols, and to enable effective coordination [58].

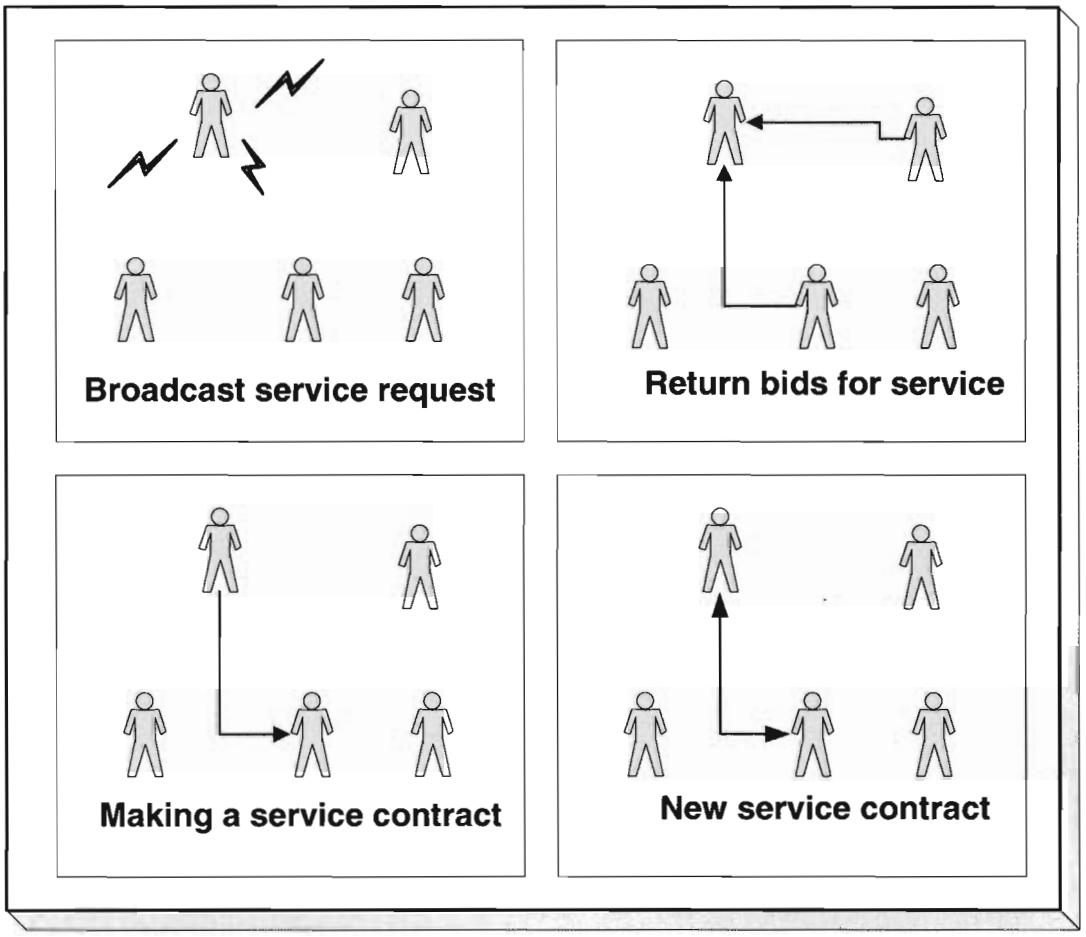


Figure: 17. The Contract Net Protocol

To controll the problem-solving behavior of a collection of distributed agents, there is always a requirement for a coordination mechanism. The most commonly employed coordination mechanism is the contract net protocol (CNP), [15]. This is concerned with the dynamic configuration and coordination of agents to form hierarchies that can achieve complex and distributed tasks in advance of the run time. The contract net protocol proposed by Smith and Davis coordination [58], provides a mechanism by which nodes (or agents) can dynamically create relationships in

response to the current processing requirements of the system as a whole, thereby enabling opportunistic task allocation.

In the contract net framework, an agent with a task to be achieved forms contracts with others who proceed to accomplish this task. The steps to forming and maintaining a contract are defined by the contract net protocol, described as follows, and shown in Figures 17 and 18.

1 An agent decomposes a task into a number of subtasks. For each subtask, the agent issues a task announcement describing what needs to be performed, along with eligibility requirements.

2 Once the agents receive a task announcement, if they fulfil the eligibility specification, it may bid for the task. An agent will not bid unless it is free to perform the task.

3 Once all bids have been received by the original (potential manager) agent, it ranks them according to criteria associated with the task, and awards a contract to the bidder with the highest ranked bid.

4 There is now a contract between the manager who made the task announcement and the bidder (the contractor) with the highest ranked bid.

The contractors issue reports to the manager, which may be interim reports, or final reports containing a result description. Finally, the manager terminates a contract with a termination message.

Contractor agents may, in turn, subcontract parts of their task by announcing their own task announcements. In this way, the CNP is repeated with the contractor eventually becoming a manager for these subcontracts. This leads to a hierarchical configuration in the contract net, as shown in Figure 18.

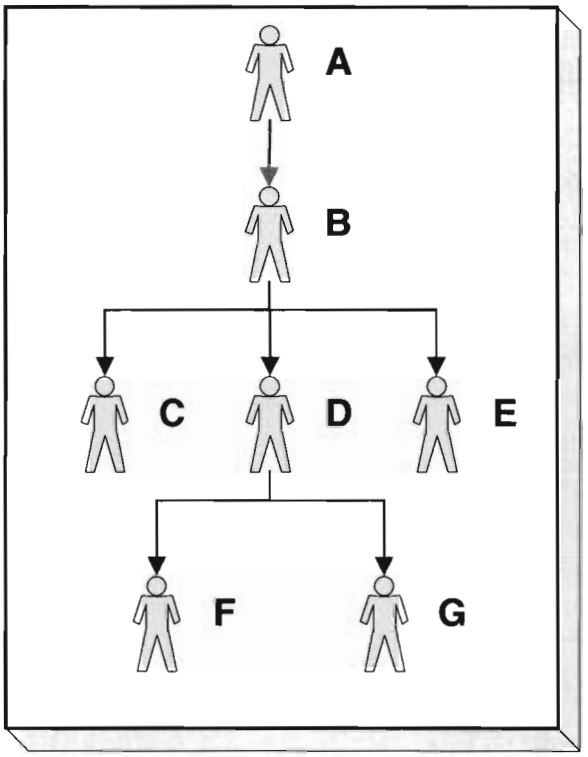


Figure: 18. The Contract Net manager-contractor hierarchy

4.3 The ASOA Framework

The ASOA framework broadly encompases three different kinds of agents, (a) Service Provider Agents and (b) Service Requester Agents and (c) combination agents - agents which are capable of serving others as well as being served by others. A combination agent is principally which has two sub agents which are service provider and service requester.

The framework can be as complex with several thousands of such agents living in harmony over several different Operating Systems, forming an ecosystem of agents, or simply two agents. The word "ecosystem" is borrowed as a metaphor to suggest a system where a software system in the operating environment is like plants and animals in the natural world. There is strong similarity between complex interaction of organism in a natural ecosystem and the complex interaction of software

components in a network system. The word "ecosystem" also refers to the field of artificial life and other complex systems and computer science work [7, 8].

4.4 Agent based design

4.4.1 The Agent Ecosystem

ASOA presents the concept of ***ecosystem of distributed agents*** as an organizing principal for building distributed systems. The idea presents ways and techniques for these distributed systems to not just communicate with each other, but also to use each others resources, making calls at system software level. In a computational ecosystem, similar to the food chain in a natural habitat, software agents have a specific cycle. Every agent is born in a particular environment, on a particular computer. The agents share and serve other agents while at the same time also request for resources and services from other agents. This mutual understanding is predominantly seen in nature where one living organism depends directly or indirectly on other for survival at the same time helping others for their survival. Computer scientists on several occasions have learnt from nature and applied the principals in solving various problems [16]. The agent lives its computational life, possibly serving other computers and/or requesting for services from other agents as the need arises, until it meets its end by its own choice, either by the hand of a user, or by an accidental mishap like a power failure. Agents interact with their local computational environment, consuming CPU, memory and raw information resources from the local world, possibly using effectors on the computer such as display, printers, scanners and other peripherals.

ASOA incorporates several participating systems with agents resident for communication, processing and request handling. Discussed in the following sections, agents live in ecosystem where they are allowed to request for services and at the same time must server if a need arises. To be served, it first needs to find an appropriate agent who is capable of serving its request.

4.4.2 Agent Presence

Many tools have been written to communicate some sort of social presence online. The old Berkeley Unix tools *finger* and *rwho,* for example, provide simple information such as whether a person is logged on to a particular computer, how many minutes it has been since they have typed, and how long ago they read emails, etc. These tools are limited - they require that one knows what computer might be logged in, the information they provide is not well standardized across implementations, and current practices is to not even run these services, as they are potential security problems.

The contemporary equivalents to these services, tools such as ICQ [38], Yahoo! Messenger [83] and Skype [71] are more reliable and better designed for the distributed nature of today's networks. They all share a common architecture – every user runs a client program that notifies a central server when they are online. The server then manages putting individual users into contact with each other. Once two clients have found each other, they typically user pear to pear messaging to continue communication. This semi-centralised approach seems to scale reasonably well, although the more popular services have reliability problems.

However, these systems are all very limited in that they provide only a small amount of information across heterogeneous platforms and are mainly limited to exchanging text based messages and files.

4.4.3 ASOA Presence Architecture

ASOA framework spans over several kinds of Operating Systems, with agent systems living in different environments. The entire set of agents is a Resident Agent Systems (RAS). Where, each agent in itself is a smaller subset of RAS. Each of these Resident Agent Systems subsets may constitute of service provider, a service requester or a combination of both. The service provider does the heavy-duty task of converting system calls from one operating system into another as requested. The service requester on the other hand is a lightweight agent class, mostly comprising of

communication methods. See Figure 19 for illustration of RAS and the ACB control framework

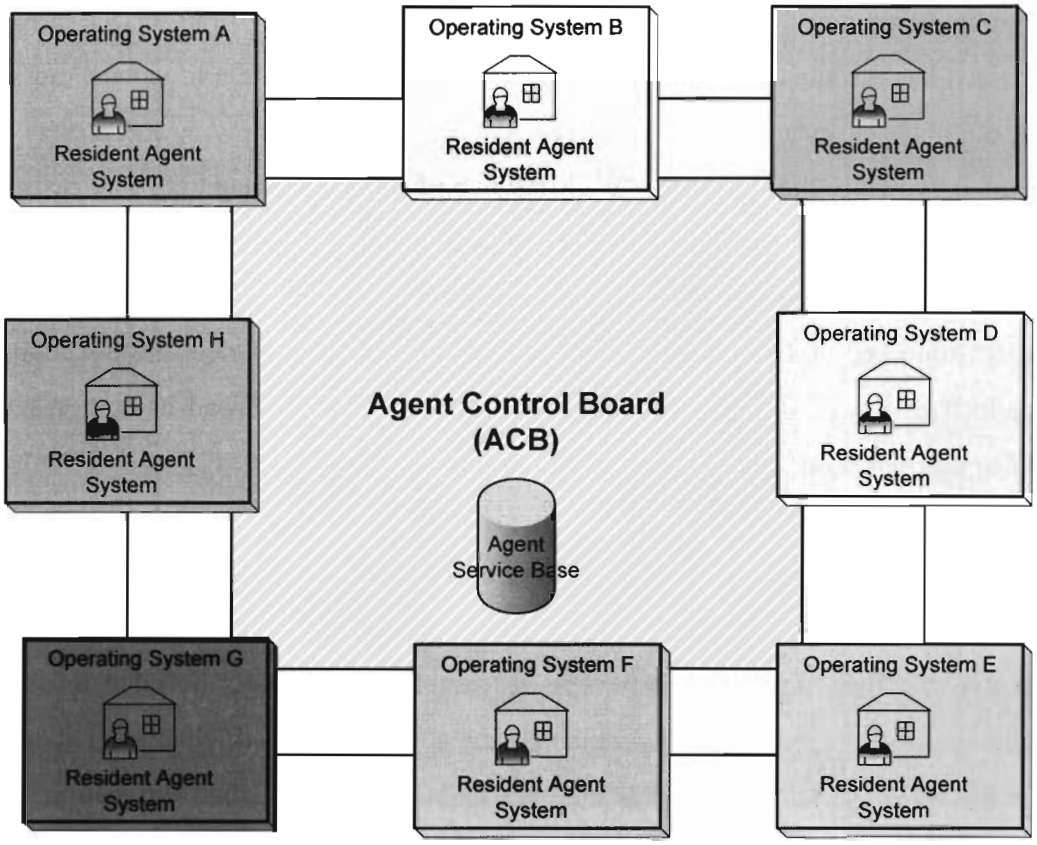


Figure: 19. ASOA Heterogenous Control Framework

Each Resident Agent System participating in ASOA framework consists of worker agents (sub RAS) and the Agent Control Board (ACB). Each RAS is seen as a single entity (or agent) to the outside world (requester or user agents). In other words, the internal functionality of a RAS is transparent to other systems and is seen as one single entity with a single interface for communication.

ACB is a central server, primarily responsible for maintaining an up-to-date list of services different Resident Agents Systems are capable of providing. The ACB of agents higher up in the hierarchy, listens for broadcasts sent by RAS of different Operating Environments and update its Agent Service Base (ASB) a service

database, for future reference. When need arises, a client RAS contacts the Agent Control Board with its query consisting of the type of service it requires. The ACB accepts the query and searches its database for other agents who are capable of servicing this request and their location. The network location of the service provider is passed back to the querying agent. The querying agents send a broadcast message to these network locations for service agents to place bids. Knowing the presence of agents is typically the job of the ACB that is *pinged* by the agents to broadcast their presence and type for services they can provide. The ACB *listens* to such broadcasts and updates its database. This allows the Resident Agents to concentrate on their actual jobs and not searching and routing for services. Once the location of prospective service providers are known through the ACB, broad case message is sent to place bids and the Contract Net protocol is adapted.

The RAS subsets may constitute of smaller agent (systems) geared to either serve, or be served or both. Figure 20 illustrates the simplest case of ASOA framework with just two Resident Agent Systems. The request for a service is initiated by a user application. The application makes a request to the ACB, which directs it to the appropriate RAS Service Recipient available locally to the application. The application is unaware of any other agent system or the functioning of this local agent system. This RAS provides the application with a proxy interface that gives the application methods and objects to instantiate to make use of the service. Note however, the objects and methods exposed are only a dummy interface with no computational power of their own. The Proxy Service Agents forward the request to Request Builder agent that packages the request and forwards to Transporter Agent (TA). TAs are primarily responsible for finding a RAS Service provider. It initiates the process of searching and allocating contracts to other RAS agents using the Contract Net Protocol discussed earlier.

Once a contract is secured with the most 'intelligent' bidder, request details are forwarded to the RAS service provider. The only interface exposed by this RAS is the TA. TA accepts the request and is forwarded internally to the Request Allocation

Agent (RAA). Consider an ecosystem where RASs coexists on dozens of different types of Operating Systems that are inherently incompatible. Requests coming from one operating environment may be totally misinterpreted by another if treated raw. The RAAs job is to interpret the signals received and search for an appropriate Request Conversion Agent (RCA). To be able to server more than one operating environment, understanding of different parlance and knowledge of applying appropriate conversion tool is required. The signals are required to be correctly transformed from one foreign environment to native signals. This conversion task is carried on by the Request Conversion Agent (RCA). The actual task is then carried out by the Service Builder Agent (SBA). Functionality of SBA would depend on the type of service this RAS is required to provide and its behavior widely varies. The service response is forwarded back to Transporter Agent to be sent back to the requester. See Figure 20.

The ASOA framework provides generic function architecture to solve a heterogeneous service provision problem. It needs to be tailored and implemented further to be applied in a specific problem.

4.4.4 Agent Communication

Communication between agents is heavily dependent on the contents and type of messages being communicated. This research assumes that these factors have been pre-medicated and contextual meaning is hard coded in algorithms of participating agents. The research also implies participation of pre defined operating environments where request predicated of two operating environments have been defined manually. The communication is carried over a pre-defined protocol with use of limited vocabulary.

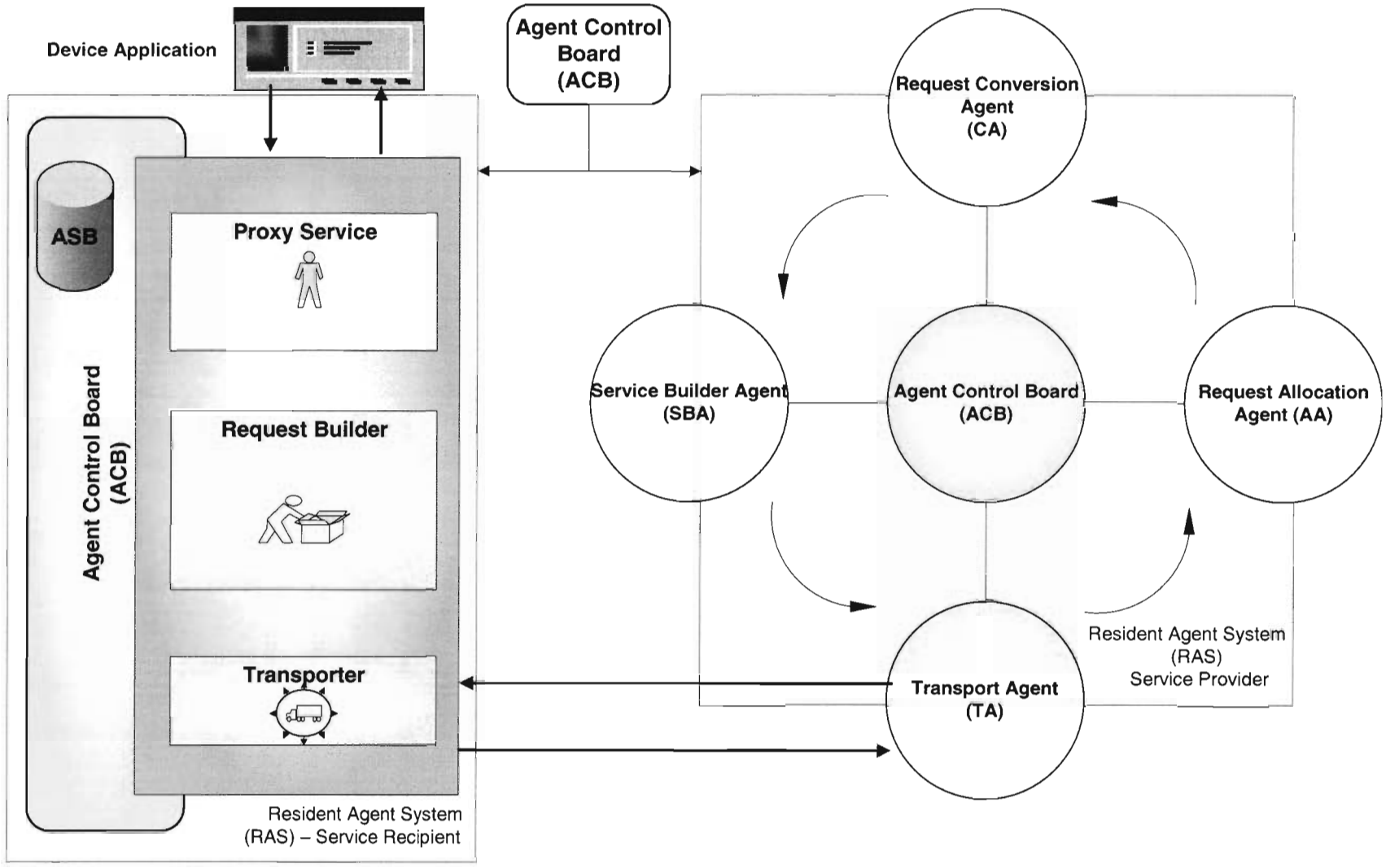


Figure: 20. ASOA – A case of single RAS provider and RAS recepient

On the other hand, however, where large number of agent participants in varying operating system environments is considered, they may require the interpretation of messages at run time. Messages sent from one operating system may need to be mapped with the services predicates in another. Such interpretation is dealt when implementing ASOA for individual services on specific platforms.

4.5 Summary

Agent based Service Oriented Architecture (ASOA) is a generic muti agent framework which can be deployed over heterogenous systems to solve a problem. The framework itself is abstract and only provides general guidelines for agent existance and communication. The agent communication protocol is based on Contract Net Protocol (CNP). The ASOA framework encompasses three different kinds of agents – Service Provider, Service Requester, and Combination agents. The overall set of agents in ASOA is seen as a single entity or Resident Agent System (RAS). Each RAS can itself be comprised of sub systems or sub RASs. A request for service is open for bidding by service provider RASs and a contract is made with the most suitable bidder. The contractor RAS can sublet tasks to sub RAS systems if required.

Chapter 5

Implementing MAPS Framework on Sun Solaris and MS Windows

5.1 Introduction

Multi Agent Printing System (MAPS) is an implementation of previously discussed ASOA framework. This implementation provides printing services over heterogenous environment. The idea is to provide device driver services for a printer which is physically connected to a Sun Ray, but makes use of agent services from Windows environment to make use of Windows device drivers. The printer used in this research is only supported on Windows and has no compatible drivers in Solaris. The remote service is transparent to the user and provides a generic solution for all non compatible printers. The MAPS implementation of an ecosystem of distributed agents is fairly simple and straightforward. Simple agents are designed in JADE [40]. MAPS derives most of its functionality, clean design and multiplatform operability from Sun's Java language runtime environment. Support for distributed agents comes from JADE.

MAPS Java classes are broadly split into three categories: RAS Requester Agent, RAS Server agent and a class for the Agent Control Board.

5.2 Technology Base

Java is more than another programming language; it is an entire environment with Virtual Machine architecture. Java's most hyped characteristic is the write once, run anywhere capability. Its capacity to compile into an intermediate Java bytecode and further compile at runtime in the machine specific environment makes it possible to run on any environment. Java can run from a large enterprise system running on Windows Server to a mobile phone with Bluetooth printing. This provides the MAPS framework the ability to run anywhere. In addition to this, the Java VM also provides an entire layer of abstraction to the MAPS framework, allowing plugging in various kinds of agents in any environment.

JADE has excellent agent message handling. However, for this project, Java's native socket programming is used to communicate with other agents. The packets exchanged adhere to FIPA [30] compliant Agent Communication Language (ACL) messages.

5.3 Agent Management

All agents in the MAPS framework are autonomous and do not depend on external sources for their computational life. However, they do request for services that the user may have demanded, for which they have to communicate with other agents. Each agent sends out messages to the Agent Control Board (ACB) about the types of services it can provide to others when a need arises, and registers itself with the ACB.

When the ACB server starts, it wipes out the old registrations and gears itself to listen for any new registration requests. Once a RAS is registered, the ACB pings it registrant after a set duration of time ensuring its registration data is up-to-date.

When a RAS sends a query to the ACB for a particular type of service, the ACB searches its database and returns a valid agent address that is capable of serving. Further communication to the server agent is requested by the RAS agents themselves.

In the implementation set up, for the sake of simplicity, there is only one printer service instantiated. This Resident Agent Service is registered with the Agent Control Board of the Sun Ray.

5.3.1 Sun Ray Printing

Sun Rays relies on Solaris for all its printing requirements, which in turn uses CUPS to support Postscript. The final state of most print requests are in Postscript format which are spooled to the Unix print spooler directory. These spooled files are picked up and piped to the printer port on which a Postscript printer is configured. The physical printer then follows the instructions provided in the Postscript file and performs the mechanical task of printing.

It is important to observe that the Solaris Operating System generates a standard Postscript file format which is followed by all compatable printers.

5.3.2 Windows Native Printing

The internal spooling of Windows is similar to Solaris in principle. However it does not have a standard printing script, but the script generated is heavily dependent on the printers native architecture. Often the script is binary data dictating the movement of printer head and ink release mechanism. This non-standard script also means that every printer requires to have its own printing mechanism and shall require a specific driver. Therefore most printer vendors have their own proprietary standards hence a separate device driver for every printer.

5.4 MAPS – The Printing Framework

5.4.1 Heterogeneous Ontology based messaging and mapping

Agents in MAPS framework communicate by exchanging messages over a TCP link. Each message contains a set of one or more message parameters that can be structured into the types of communicative act, participants of communication, contents of the message, description of message and finally the message conversation control signals. Every message

originates from a RAS that is either requesting a certain service or informing the ACB of its presence. The participants in communication at a given time only consist of one sender and one receiver. The message parameters will consist of Predicates tightly bound to the originating operating environment, which in most cases may not be applied directly in the destination environment. However, this incompatibility issue can be addressed beforehand. If the two participants have been predefined, ontologies can be mapped manually before installation. In cases where the two participants have never met before and one-to-one mapping has not been carried out manually, there could arise a completely new avenue of research work in mapping the ontology at run time.

In this MAPS implementation, a Windows compatible USB printer is connected to a Sun Ray client running on Solaris. A Sun Ray application requests a page to be printed in this non-compatible printer. For successfully printing to this printer, the system needs to convert the Solaris based Post Script format into the printer specific format using services delivered on the Windows machine. The requester RAS initiates a message for the Windows RAS that will consists of Printing Predicates such as Printer model, make, colour/monochrome, number of copies, pages per page etc. a Solaris resident RAS agent requests print conversion service to another RAS agent residing in Windows environment.

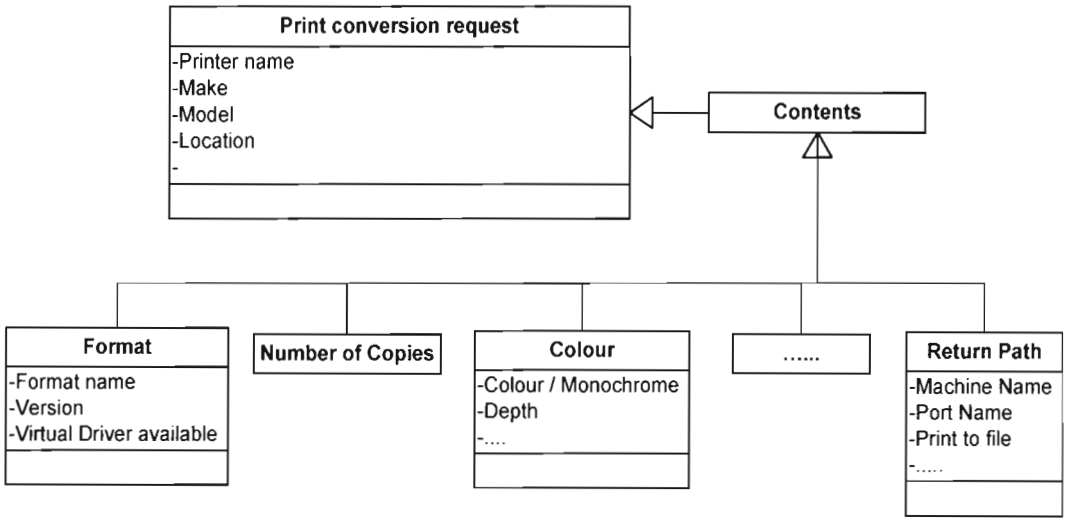


Figure: 21. Part of Print Conversion Ontology

Ontology of one Operating environment may not have one-to-one mapping with another on another environment, causing the interoperability problem during the negotiation process. For example, in Printing Ontology above there is a concept called "Version" which in another environment may be "Release No". Even if the same term is used to represent the product in both ontologies, they may have different characteristics and different relationships.

Ontology mapping is the technique of finding equivalence between the concepts of two ontologies (similarity). If two concepts equate, they mean the same thing, or closely related things [5]. The approach aims at finding correspondences between concepts based on the concept names, their characteristics, relations and the description of the concept. Several related works [1, 14, 73] in the multi agent systems area have been developed and different approaches have been implemented trying to solve the heterogeneity problem in agents' communication [3] describes an approach to ontology negotiation that allows web-based information agents to resolve mismatches in real time without human intervention. The terms exchanged consist primarily of the query content and document descriptors for query results. The fundamental tasks are interpretation, clarifying, relevance evaluation, and ontology evolution.

Once this contextual problem is resolved, the framework then accepts the message and applies the interpretation for the necessary work required for fulfillment of the request. Figure 21 shows the message passing cycle in the MAPS framework at a microscopic level – which includes only two Resident Agent Systems.

MAPS is a novel agent based printing framework proposed in this research work that utilisies print services in a heterogenous environment. While deployment of agents alone does not solve the problem of printing to a non compatible printer, the framework also has a conversion engine for conversion of non compatible print scripts to printer specific compatable ones. Deployment of Multi Agent systems has proven advantages of both energy saving and user satisfaction through value added service [4]. Agents can give users proactive, personalized assistance with a computer application. Such agents are analogous

to personal assistants in the real-world work environment [60]. Both agents and assistants gradually learn a target individual's preferences and habits to enhance collaboration and productivity.

In this research work, the proof of concept was shown by utalising the MAPS framework for print on a Windows compatible printer attached to a Sun Ray thing client.

Implementation of this architecture is divided into two factions – Service Consumer Agent at the Solaris end and Service Provider Agent at the Windows end. The two environments will be linked through a TCP/IP socket and agents interact with each other passing the relevent information.

The core problem lies in converting the Postscript file generated by a Sun Ray session in Solaris into the printer specific (.prn) file. Since this prn file is heavily
dependent on the printer architecture, this can only be generated by the printer's native device driver. The task of conversion will be carried out at the Windows end utilising the Windows device driver for this printer.

Before the two agents are implement, the two Operating Systems's communication is worked out. Socket based client - server program in Java are deployed so that binary and text data can be sent back and forth seamlessly. In this case, the Java client resides on the Solaris side and sends Postscript data to the Java server running on the Windows end. Copy of same Transport Agent is used to send processed data back to Solaris from Windows.

Solaris based Resident Agents System are responsible for gathering requests and probing for device description when a request needs to be serviced. They will also be responsible for sending the requested script over to Windows based agents, which perform the task of transforming UNIX script into Windows script that can be readily understood by the device attached on the other end [79]. A Postscript reader (GhostScript) capable of printing to a local virtual Windows printer is used. The print driver then performs the conversion of Postscript to printer specific script. After the conversion, the script is sent back to the

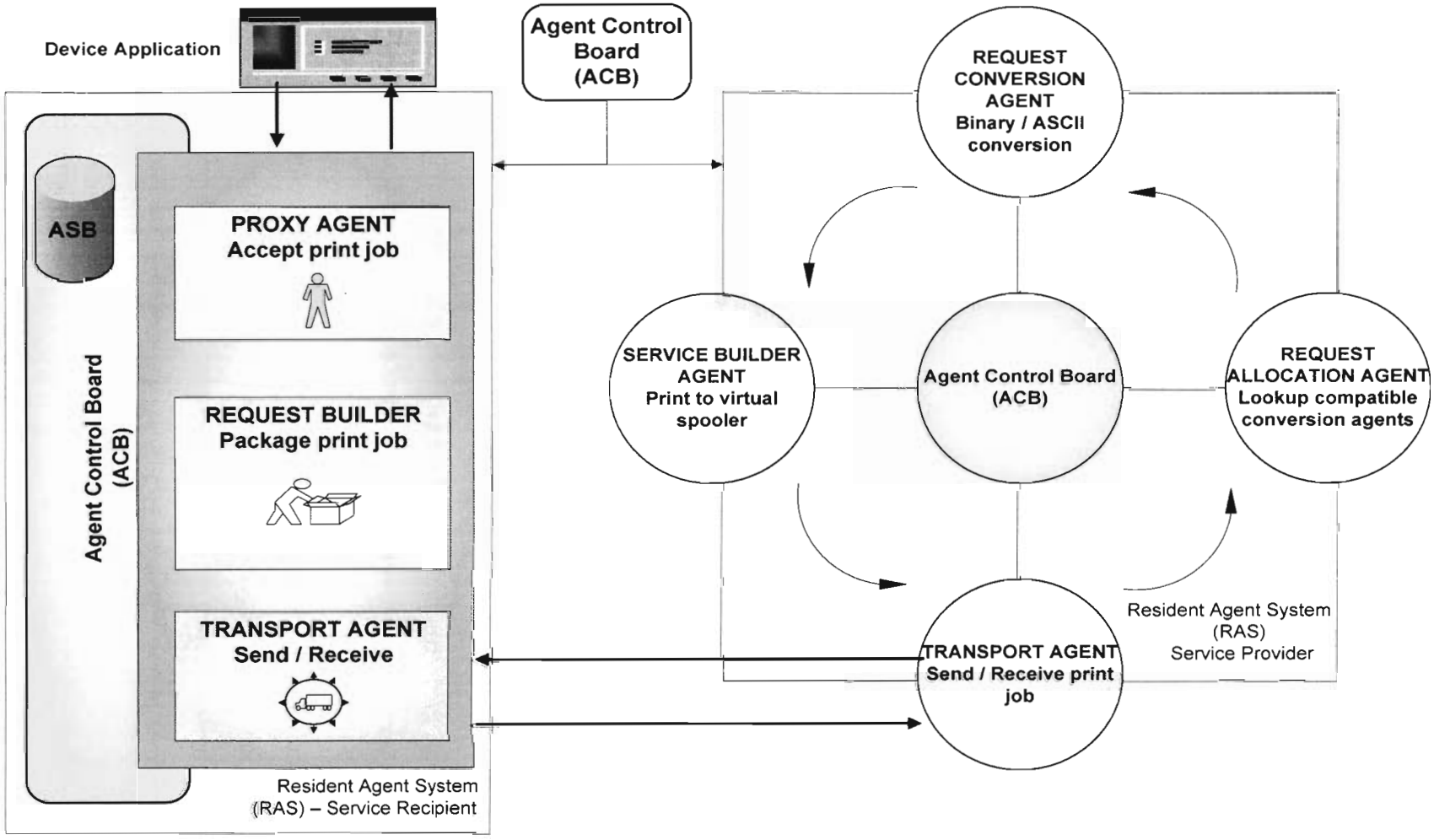


Figure: 22. MAPS, an implementation of ASOA

discovery agent in Solaris for dispensation. Figure 23 shows the test bed setup to support this architecture.

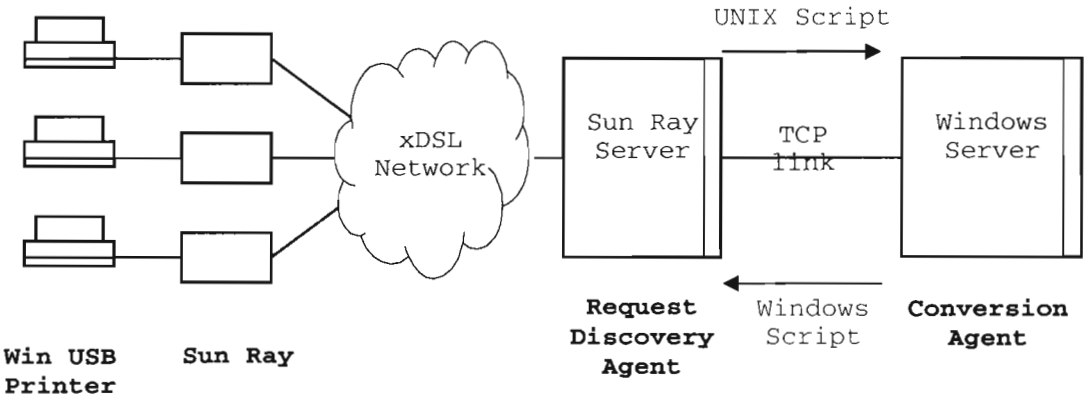


Figure: 23. MAPS Test Bed

5.4.2 Setting up Solaris Proxy Agent

This agent is responsible for accepting a print request from the Sun Ray user, packaging the (Postscript) file and sending it to the Windows agent for processing. On return of the processed file, it then redirects the binary print data to the requesting client for printing. A generic Postscript printer is configured in Solaris and attached to a spooler. Sun Ray user profiles are built to print on this printer. There is no physical printer attached to this spooler. Hence all requests will simply be lying pending in the spooler directory. Postscript being the inherent printing format in Solaris, all request in this spool will be .ps files [79].

The Java client (Transporter Agent) which has been configured earlier, will run after few second interval and collect these Postscript files from the spooler directory. These files are packaged and sent to the RAS listening on the Windows end.

At this stage the Conversion Agent at Windows end takes control, does the necessary processing and sends back the printer specific binary data to the Transporter Agent listening in Solaris. The Transporter Agent in Solaris accepts this binary data and pipes it to the USB port of the client which was is the originator of the request. On successful transmission to

printer connected on the USB port, the file in the print spooler directory can be either deleted or archived away for statistical data.

5.4.3 Setting up Windows Conversion Agent

The crux of this project lies in this conversion engine. Its task is to receive the Postscript files and convert them into printer specific format. These are then transmited back to the requesting agent on Solaris. We need to have the vendor specific printer device driver installed for the printer we are supportig on Sun Ray . The Java Server listening for any requests receives the Postscript file and stores this file for conversion. In order to utilise the device driver for conversion we need to print the Postscript file and capture the output which is normally sent to the printer. A TCP port is then configured and a printer is set up to this port. In our case we configured the local host IP and port number 1522. Print request sent to this printer will be forwarded to the TCP port on localhost:1522. A Java socket listener is then put in place which listens to this TCP port. We reused the Java server code used above in communication between Solaris and Windows.

We then use GhostScript with the 'Device=mswinpr2' option to use Windows print drivers and print the Postscript file. The print output is piped out to the TCP port configured above and captured in a binary file by the Java listener. This file is sent to the Solaris listener by the communication link already setup.

Instead of configuring a TCP port for capturing the print output, we could also use third party tools such as 'Print Distributor' [6] which are capable of creating virtual printers and capturing printer output in storage files.

5.4.4 Collecting Solaris print request (Solaris Proxy Agent)

To give users a transparent perception of smooth printing, the print requests in Solaris print queue have to be captured when user give a print command. A virtual print spooler and its associated directory needs to be setup in Solaris where print requests from clients can be captured and stored. The outcome of the print command is a Postscript file generated by

CUPS. The Solaris RAS picks up this file from the spooler directory. Thus far the system is using Unix native printing system and requests (the physical file) are in Postscript format. The virtual spooler should not be configured with any physical printer. This will allow the bypassing of normal printing to a locally attached Postscript printer. The files from this spooler then have to be transported to a Microsoft Windows machine for conversion. We can allocate this task of scanning the spooler directory and forwarding requests to a Windows machine to the Request Discovery Agent. The agent will collect this information, make a TCP connection with the predefined Windows machine over an allocated port number and transfer (push) the data for conversion. At this stage we are assuming that we have already queried the Agent Control Board (ACB) for a suitable service provider agent. The Request Discovery Agent will also probe the printer type and send a header metadata, indicating the printer model, make and its manufacturer data. Ontology of this message has been worked by the agents and / or are currently hard coded in the algorithm.

5.4.5 Conversion engine (Windows Conversion Agent)

On the Windows end, a Conversion Agent, part of WindowsvRAS has been *listening* for requests. It will receive the Postscript data and store it for conversion. This data contains a header indicating the type of conversion required- ie: what printer the request is for, what model etc.

We need to install Windows printer drivers of printers which we are supporting and ensure that these drivers are available for the Windows based Conversion Agent. The conversion process involves printing the Postscript file (received from Solaris RAS) to the Windows (virtual) printer using the Windows print drivers. This simple technique of printing can convert any format to the required printer format without having to re-engineer a new device driver. However, instead of leting the Windows printing system to physically print on the printer, we need to capturing the output sent to printer in a file. This output file consists of all the commands required to print a page on the printer. For this we will also require a Postscript reader, capable of printing. In our test bed we have used GhostScript [32]. Capturing the output from a printer can be tricky as there are no APIs provided by

Microsoft to fully automate printing to a file instead of the printer [49, 82]. This research proposes two ways to solve this problem:

(a) Conversion using third party tools

There are various third party Windows utilities which can accept print requests and archive the print script for later use. One such example is Print Director [52]. Print Director is capable of creating a virtual print spooler in Windows which may or may not be bound to a physical printer. The virtual printer can be programmed to redirect the output to a storage file (instead of a physical printer). The Windows resident agent must pull the Postscript file and print it to Print Director printer, which in turn prints the output to a file in a predefined directory. The script file thus generated is compatable with the printer we are using in the Sun Ray environment.

(b) Conversion using a TCP listener

Alternatively, we can configure a TCP printer in Windows which prints to a predefined port. This functionality can be fully automated. A listener is then required to listen to this TCP printer, capture the data and store it in a script file. For our use we have tewaked the listener used in the Resident Agent System (RAS) and re-used most of the code.

5.4.6 Transmitting printer script data to Sun Ray

The script we have now is stored at the Windows machine and can be sent back to Solaris Server. Once again it is the responsibility of the Resident Agent System to do this transmission of data back to the requester. A client socket pushes data to a listener in Solaris. The listener receives the bit stream and stores it in a buffer. This transaction is encapsulated in the two Transporter Agents.

Once the converted script has been received by the Solaris based RAS, the final step is to send this script to the USB port of the requesting Sun Ray client. This can be done simply by using the ***cat*** utility and redirecting the output to USB port. The printer connected to this USB port fully understands the script as it is generated by its native Windows driver. The result is a print out.

5.5 Summary

Multi Agent Printing System (MAPS) is a JADE implementation of ASOA framework discussed in the previous chapter. This implementation demonstrates the provision of printing services over Solaris and Windows network, and provides device driver services for a printer which is physically connected to a Sun Ray, but makes use of agent services from Windows environment to utilise Windows device drivers. In this scenario, two sets of agents are designed, one installed on Solaris and the other on Windows platform. Sun Ray agents make requests to RAS agents in Windows for printer device driver specific services. Requests and responses are transmitted using predefined message passing ontologies.

Windows Agents receive the print requests from Solaris which are in postscript format. These print signals are converted into the printer specific signals using device drivers installed on Windows OS. Several methods have been discussed to carry on this conversion task. The printer specific signals so generated, are sent back to the Solaris requester agent. These signals are in turned piped through to the USB port of the Sun Ray thin client where the physical printer is connected. Since the print signals are in the printers native format, the printer is now capable of carrying out the commands and produce a print out.

Chapter 6

Evaluation and Performance of MAPS

6.1 Introduction

The experience of writing MAPS framework was very positive. The biggest achievement has been that the development of this novel technique now provides the ability to print on unsupported printers without having to re-engineer or reverse engineer a new device driver. The solution is put to test for qualitative and quantitative tests.

6.2 Evaluation

The hardest part was to design the conversion agent responsible for converting system calls of one operating system into another. Although, the resultant design turned out to be a lot simpler than originally anticipated, much needs to be done to take the existing MAPS code base from its current form to production environments. However, as a whole, MAPS has been an effective educational and research experience in an effort to bring incompatible operating systems a step closer to each other.

The work done has helped develop the vision of bringing heterogenous operating environments closer and more operable with each other. There is much value in having a working system: the code base and the proof of concept is developed enough to allow for those future applications. In addition, in the course of this research, much has been learnt about heterogenous operating environments and distributed multi agent applications.

Comparision to any other solutions for an incompatible peripheral will easily prove MAPS to be a simpler and far more effective solution. Other solutions as discussed in Chapter 2 port drivers from the closest working operating systems or re-engineer one. In either case the results are not guranteed to be what can be obtained by using a device driver which has been exclusively written for that device. MAPS uses the resultant outputs from a devicecs native device driver, hence there is minimal posssibilities of any performance degradation or conversion errors. The only drawback is that it requires a machine which has the device driver of the device in question.

6.3 Performance Testing

(a) Test of correctness

The test of correctness can be confirmed by visual inspection of the printouts. The original framework was written with HP Deskjet 540 printer. Subsiquently, tests were conducted with Lexmark, Canon and Dell printers too. Supporting a new printing device with the current framework only required its native device drivers on the Window machine to be installed and no code was modified. This flexibility and generic nature makes the solution attractive and cheap.

(b) Qualitative Test

Once again this test was carried out with visual perception. High graphic images were printed in colour and sent through the MAPS framework. The results were virtually the exact quality print outs one would expect in the printers native (Windows) environment.

(c) Quantitative Test

Two factors influence the print speed in this architecture. The first, being the printers capacity to print n pages / minute. This remains constant in our setup as it really depends on the printer hardware. The second is the speed of data transfer between the Windows machine and the Sun Ray client. This latency depends on the time required to carry out the

print conversion process and time required to send data back over the network to the client. To test this byte transfer rate we set up the network discussed below.

Since Sun Ray architecture is quite different from Intel PC architecture, where the client does not carry out processing or decision making – to the extent that even screen refreshes are also sent by the remote server. All instructions are carried out by the server, which may be available locally on the network, or on a geographically remote area through a broad band connection.

The performance of Sun Rays should be judged by what the user experiences on the client side, and not just by measuring the server performance [46]. The overall performance of a thin-client depends on three factors: performance of the server, network latency and screen display quality. Hence the performance of the overall system needs to be measured, taking into account all parameters engendering.

Thin-clients are dumb machines; even the user session resides on server and not on the client. This makes it impossible to install any measuring software on the client to monitor any performance issues. Slow-motion technique suggests measuring the network activity between the server and the client, which will provide an overall perception of the user's experience.

We suggest adapting the Slow-motion technique to measure thin-client performance. This overcomes the problem of de-coupled screen updates from the application logic and allows the server to run the application without being constrained by the slow display update speed [46, 55, 85].

6.3.1 Test bed Construction

Technique proposed by Columbia University is adopted for benchmarking and measuring the performance of MAPS on a Sun Ray thin client. To measure the latency, the network is setup as shown in Figure 24 below.

The test setup consists of two sets of client server systems, a network simulator machine, a packet monitoring machine, and a benchmark server. Broadband band environment is simulated to conduct tests on up to 100Mbps connections.

As shown in Figure 24, the network simulator machine is placed between the server and the client to control the bandwidth between the two. This machine is configured with 2 network interface cards and allows all packets to be sent / received passing through this machine. The simulation software, The Cloud [74], controls the bandwidth by restraining the number of packets sent per unit time interval between the two network interface cards.

The following table describes the components required to set up the test bed.

Role	**Configuration**	**Description**
Sun Thin Server	Sun UltraSPARC IIi, 256 MB RAM, 10GB HDD 2 x 10/100BaseT NICs	Sun SPARC machine running on Sun Solaris operating system.
Sun Ray Thin client	Sun Ray 100 / 150	Sun Ray connected via a router to the network
Windows Terminal Server	P IV 1 GHz, 512 MB RAM, 20GB HDD, 2 x 10/100BaseT NICs	Intel based machine running Windows Terminal Server.
Windows Thin client	P III 1 GHz, 512 MB RAM, 20GB HDD, 10/100BaseT NICs	Intel based machine running on Windows 2000, configured to run RDP.
Packet Monitor	P III 1 GHz, 512 MB RAM, 20GB HDD, 2 x 10/100BaseT NICs	Windows 2000 machine running a packet monitoring software - Etherpeek
Network Simulator	P III 1 GHz, 512 MB RAM, 20GB HDD, 2 x 10/100BaseT NICs	Windows 2000 machine running a simulation software – Cloud from Shunra

Role	Configuration	Description
Benchmark Server	P III 1 GHz, 512 MB RAM, 20GB HDD, 2 x 10/100BaseT NICs	Windows 2000 machine running IIS server

It is required to measure the delays caused by the simulator machine itself and account for the differences in round trip for packets with and without the simulator machine.

The Packet Monitoring machine will run a monitoring software Etherpeek NX [29]. This application will timestamp and register all packet traffic visible in the route. The hub will expose all packets sent or received to or from the client and thus be visible to the Packet Monitoring machine.

The fat Sun Ray server has three network interface cards. The first connecting to the network simulator, the second to the Windows Conversion Agent and the third to the Benchmark Server. The seclusion of Benchmark Server to the Sun Ray server alone ensures no leakage of its packets to interfear with the Packet Monitoring machine.

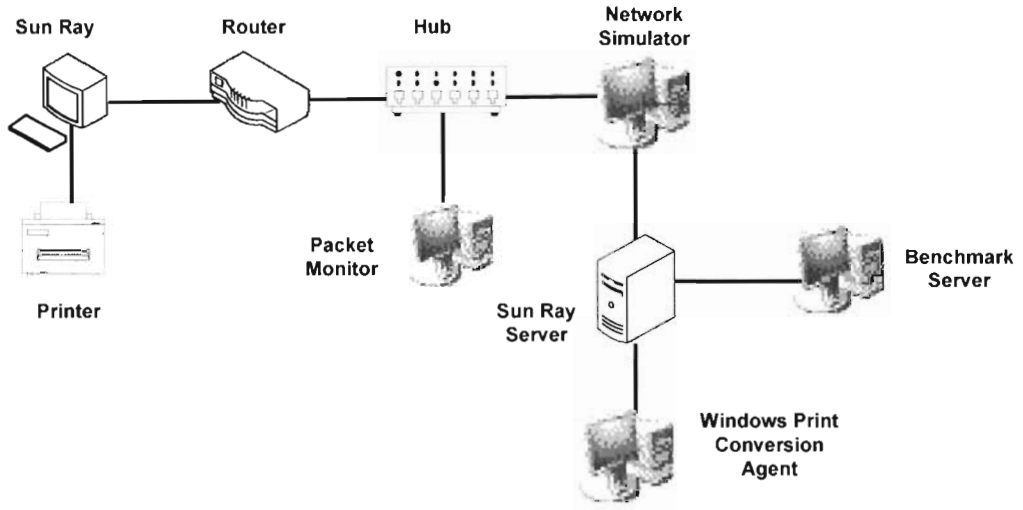


Figure: 24. Test bed for Sun Ray print latency measurement

6.3.2 Observations

Conversion process is inversely proportional to the network latency. Large graphical print requests over a slower network (256Kbps or slower) can take more than 5 minutes to process. It is interesting to note that often the printer specific print files, generated by Windows device driver are significantly larger than its postscript counterpart. The transmission of larger files over a slow WAN makes the conversion process obvious to the user. On faster network speeds however (10Mbps or faster), the conversion process works seamlessly and is transparent to the user. Three processes contribute to the overall processing time: A) Capturing the postscript file in Solaris and transmitting it to the Windows machine. B) Converting this postscript file into printer specific file and C) Transmitting the printer specific file back to Solaris server. In the proposed setup, the Windows machine is locally connected to the Solaris server on a fast port, and hence transmission of postscript file to the Windows machine is not noticeable. This is because the Sun Ray session exists on the Solaris server and not on Sun Ray terminal itself. Step C takes the longest, as the converted printer specific file is relatively bigger than its postscript counterpart, and more importantly it has to be transmitted to the Sun Ray's USB port. Step B forms the main bottle neck of the entire process, and latency can be improved by throttling up the network speed. Processing time required for Step B depends on hardware resources available on the Windows machine. With support for multiple users, faster machines with more memory and processing power will be required.

6.4 Summary

Tests conclude that MAPS implementation allows use of any printing device on Solaris platform which is supported on Windows. It utilises the Windows device drivers and converts the Solaris print signals into native signals. Using the device's native drivers allows minimal conversion errors. The qualitative test carried out by perception shows outputs produced at printer's best capability. There was no qualitative degradation due to conversion process. Test of speed revealed that network latency is directly proportional to the number of pages printed per minute.

Chapter 7

Conclusion and Future Work

7.1 Conclusion

The work in the thesis provides a new and novel method to make use of printers which are totally incompatible in a given operating environment. The solution is generic and utilises device drivers installed on a compatable operating system. This mothodology has proved to be significantly better than any other mothod used so far, which involves re-engineering, tweeking or porting of device drivers.

The research work investigated various distributed computing architectures and technologies available to current heterogeneous enterprise solutions. The task was carried out specifically to design and implement the Agent based Service Oriented Architecure, capable of providing cross platform services. The ASOA is a generic framework and provides guidelines for hosting and utilising services of one operating system in another. It also provides place holders for transportation, communication and conversion processes required in order for one operating system to effectively communicate and make use of another operating system. The key idea is to be able to have access to resources of one environment in a totally incompatable environment.

ASOA framework was implemented for printing services in Windows and these services were utilised on a Sun Ray thin client. This MAPS implementation is designed in JADE and is focused in utilising Windows printer device drivers in Sun Ray Solaris. MAPS

agents are deployed on Windows and Solaris machines, which form the focal conversion system.

This technique exploits the benefits of Multi Agent System and does not have the drawbacks of other methods which involve porting, tweaking or reverse engineering of device drivers. More importantly, introduction of a new printer make / model does not require any software change. New devices can be introduced to this heterogenous system by simply installing the Windows device driver and making it available to the Conversion Agent.

The ASOA framework can be implemented to solve other similar problems of unsupported devices such as scanners, digital cameras, and other USB devices. Moreover, ASOA framework can be modified to use applications of one operating system in another by changing the interface at the client end. For example, most banking software which run on COBOL or COOLGEN provide a very basic user interface. Most such systems still have a character user interface. ASOA can be implemented to form the communication medium between Windows based graphic rich front end and a Mainframe (COBOL) application.

7.2 Future Work

Further work may be done to benchmark the efficiency and speed of conversion. As the project is to run the clients over a xDSL network, sending large binary data to the client may incure network overhead. The agents and communication link may be optimised to suit the shear bulk of printing.

More importantly, as the number of user grows there will be a need to support printers from multiple vendors concurrently, and Conversion Agent will be required to multi thread and select the correct Windows device driver for conversion process.

Secondly, an interesting extension of this study is to outsource the conversion process to other agents on the Internet. This will allow multiple conversion services to be available which provide a whole range of devices from different vendors. This process will make

different kinds of devices and peripherals available to users very quickly, without having to setup and dedicate a Windows machine to host the device drivers. The concept of agent ecosystem, where agents live and share in mutual harmony can be implemented for such a system.

While this research work presents a novel approach to printing in a mixed operating systems environment, it does not provide various user interfaces and tools to control and housekeep the various printing operations. With so many advancements being made in the field of Artificial Intelligence, it can be easily incorporated into this work to extend the concept of agent ecosystem. However, it is beyond the scope of this thesis and will be covered in future extensions.

On a wider perspective, making available system software and resourses of one Operating Environment to work seamlessly in another, opens a whole new world of posibilities. The study indeed has value in extending it to study co-existanse of applications such as IBM's DB2 running on z/OS and SQL Server runing on MS Windows, allowing the two RDBMS to share schemas, jobs, and security rolls. Several enterprises have a mix of such systems and applications spanning over two or more RDBMSs. ASOA can be re-implemented to provide a common gateway for controlling administrative tasks such as jobs, permissions, indexes, statistics and other database objects.

Bibliography

[1] Aart, C., Pels, R., Caire, G. & Bergenti, F. (2002, July). "Creating and Using Ontologies in Agent Communication", In Proceedings of 1st International Conference on Autonomous Agents & Multiagents (AAMAS´02). Second International Workshop on Ontologies in Agent Systems (OAS2002). Bologna, Italy.

[2] Angeletti,M., Culmone, R. & Merelli, E . *An intelligent agents architecture for DNA-microarray data integration.* Retrieved September 10, 2006, from *http://www.bioagent.net/WWWPublications/Download/Nettab01.pdf*

[3] Bailin, S.C., Truszkowski, W. (2001, July 18-20). "Ontology Negotiationbetween Scientific Archives". In Proceedings of TheThirteenth International Conference on Scientific StatisticalDatabase Management. Fairfax, Virginia.

[4] Davidsson, P., Boman, M. (2000). "A Multi-Agent System for Controlling Intelligent Buildings". Fourth International Conference on Multi-Agent Systems (ICMAS'00). icmas, pp.0377.

[5] Dou, D., Mcdermott, D., & Qi, P. (2003). "Ontology Translation on the Semantic Web". In Proceedings of International Conference on Ontologies, Databases and Applications of Semantics (ODBASE 2003).

[6] Frogmore Computer Services Ltd. The most flexible print to file software. Retrieved October 29, 2006, from Website: http://www.printdistributor.com/print-to-file.html.

[7] Huberman, B.A., editor. (1988). *The Ecology of Computation.* Elsevier Science Publishers.

[8] Langton, C., editor. (1987). *Artificial Life (Proceedings of the First International Conference.* Addison-Wesley, ISBN: 0-201-09356-1

[9] Maes, P.(1994, July). Agents that Reduce Work and Information Overload. Communications of the ACM, 37(7):31-40. Retrieved October 29, 2006, from

Website: *http://pattie.www.media.mit.edu/people/pattie/CACM-94/CACM-94.p1.html.*

[10] Rudowsky, I. (2004, August). Intellligent Agents, Americas Conference on Information Systems. Brooklyn College, NY.

[11] USB, *Homepage.* Retrieved September 10, 2006, from *www.usb.org.*

[12] Wallnau, K. & Foreman, J. (1997). Object Request Broker, Carnegie Mellon Software Engineering Institute.

[13] Wiley, J. & Sons.(1997). Roger Sessions. *COM and DCOM: Microsoft's Vision for Distributed Objects*. ISBN: 0-417-19381-X.

[14] Willmott, S., Constantinescu, I. & Calisti, M. (2001). "Multilingual Agents: Ontologies, Languages and Abstractions". In Proceedings of the Workshop on Ontologies in Agent Systems. 5th International Conference on Autonomous Agents. Montreal, Canada.

[15] Smith, R. G. (1980). "The Contract Net Protocol," *IEEE Transactions on Computer.* Vol. 29, No. 12, pp. 1104-1113.

[16] *Ant Colony Optimization and its Application to Adaptive Routing in Telecommunication Networks*. Retrieved November 16, 2006, from *http://www.idsia.ch/~gianni/Papers/thesis-abstract.pdf.*

[17] Bray, M. Middleware, Carnegie Mellon Software Engineering Institute.

[18] Brother, *Homepage.* Retrieved September 10, 2006, from *www.brother.com.*

[19] Canon, *Homepage.* Retrieved September 10, 2006, from *www.canon.com.*

[20] Client Server Architecture, Answers. Retrieved October 29, 2006, from Website: *http://www.answers.com/client%20server%20architecture.*

[21] Client Server Architecture, Weboopedia. Retrieved October 29, 2006, from Website: *http://www.webopedia.com/TERM/C/client_server_architecture.html.*

[22] Coen, M.H. (1995). SodaBot: A software Agent Construction System, MIT AI Lab, USA.

[23] COM: Component Object Model Technologies, Microsoft. Retrieved October 29, 2006, from Website: *http://www.microsoft.com/com/default.mspx.*

[24] Common UNIX Printing System, *Homepage.* Retrieved September 10, 2006, from *www.cups.org.*

[25] Common UNIX Printing System. Retrieved July 16, 2006, from Website: *http://en.wikipedia.org/wiki/Common_Unix_Printing_System.*

[26] DELL, *Homepage.* Retrieved September 10, 2006, from *www.dell.com.*

[27] Device Drivers, definition. Retrieved September 10, 2006, from Wikipedia.

[28] Easy Software Products, *Homepage.* Retrieved August 13, 2006, from *http://www.easysw.com.*

[29] Etherpeek. Retrieved July 13, 2006, from Website: *http://www.wildpackets.com/products/etherpeek/overview.*

[30] FIPA *Homepage.* Retrieved November 16, 2006, from *http://www.fipa.org.*

[31] Genesereth and Ketchpel.(1994) in Rudowsky, I.(2004, August) Intellligent Agents, Americas Conference on Information Systems. Brooklyn College, NY.

[32] Ghostscript, Ghostview and Gsview, University of Wisconsin, Computer Science.

[33] Grimley, M. J. & Monroe, B. D. (1999). Introduction. Protecting the Integrity of Agents: An Exploration into Letting Agents Loose in an Unpredictable World. Retrieved October 29, 2006, from Website: *http://www.acm.org/crossroads/xrds5-4/integrity.html.*

[34] Hayes-Roth, B. & Larsson, J. E. (1995). A Domain-Specific Software Architecture for a Class of Intelligent Patient Monitoring Agents. Retrieved September 17, 2006, from Website: ftp://ftp.ksl.stanford.edu/pub/KSL_Reports/KSL-94-20.ps

[35] Hewlett-Packard, *Homepage.* Retrieved September 10, 2006, from *www.hp.com.*

[36] How USB Ports Work. Retrieved September 10, 2006, from *www.HowStuffWorks.com.*

[37] Hughes, P. (2004). Operating Systems Comparison; Department of Computer Architecture. Retrieved September 2004 from http://studies.ac.upc.es/ETSETB/ARISO2/Documentacion/oscomp.html

[38] ICQ, Home Page. Retrieved October 10, 2006, from www.icq.com.

[39] Jacobs ,K. How does USB Work? PC Mechanic. Retrieved September 10, 2006, from *www.pcmech.com.*

[40] JADE *Homepage.* Retrieved October 10, 2006, from *http://jade.tilab.com/*

[41] Jae, S., Nieh, J., Selsky, M., Tiwari, N. (2002). *The Performance of Remote Display Mechanisms for Thin-Client Computing*. Columbia University. Retrieved August 12, 2006, from *http://ww.ncl.cs.columbia.edu/publications/usenix2002_fordist.pdf.*

[42] Jae, S., Nieh. J., Selsky. M., & Tiwari, N.(2002). The Performance of Remote Display Mechanisms for Thin-Client Computing, Columbia University. Retrieved October 29, 2006, from ww.ncl.cs.columbia.edu/publications/usenix2002_fordist.pdf.

[43] Journal of Undergraduate Sciences Online. (1996) Vol 3, No 3. Harvard Computer Society Retrieved February 2005 from http://hcs.harvard.edu/~jus/0303/news.pdf

[44] Lai, A., Nieh, J.(2002, June). *Limits of Wide-Area Thin-Client Computing.* Retrieved October 28, 2006, from *www.ncl.cs.columbia.edu/publications/sigmetrics2002_i2thin.pdf.*

[45] LibUSB Documentation. Retrieved October 29, 2006, from Website: *http://libusb.sourceforge.net/documentation.html.*

[46] Mathur, A. (2005). "*Sun Ray Thin Clients on Broadband Networks*", Research Colloquium, University Of Canberra, School of Information Sciences and Engineering.

[47] Message Passing Interface Forum, A message passing interface Standard, 1995. Retrieved October 29, 2006, from Website: *http://www.mpi-forum.org/docs/docs.html.*

[48] Michael, S. Intelligent Agents –Inevitable Tools for Teleworkers, Institute for Information Processing and Microprocessor Technology.

[49] Microsoft Help and Support; KB Article ID 820644, (2003) DOCBUG: "PrintDocuments Class" does not support PrintToFile feature". Retrieved Septemebr 17, 2006, from Website: *http://support.microsoft.com/default.aspx?scid=kb;en-us;820644.*

[50] MIT Media Laboratory, Software Agents of the MIT Media Laboratory. Retrieved October 29, 2006, from Website: *http://agents.www.media.mit.edu/groups/agents.*

[51] Molina, M., Shahar,Y., Cuena, J. & Musen, M.A. (1996). "*A Structure of Problem-solving Methods for Real-time Decision Support: Modelling Approaches Using PROTEGE-II and KSM*". Retrieved July 17, 2006, from Website*: http://smi-web.stanford.edu/pubs/SMI_Abstracts/SMI-96-0631.html.*

[52] MSDN, .NET Framework Class Library. Retrieved September 17, 2006 from Website: *http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpref/html/frlrfsystemdrawingprintingprintdocumentclassprintersettingstopic.asp*

[53] Muli Ben-Yehuda, IBM Haifa Research Labs and Haifux - Haifa Linux Club. Retrieved September 10, 2006, from *http://www.mulix.org.*

[54] Network Working Group Sun Microsystems. RPC:Remote Procedural Call Protocol Specificaion. RFC 1057.(1988).

[55] Nieh, J., Yang, S. *Measuring the Multimedia Performance of Server-Based Computing.* Retrieved October 29, 2006 from *www.ncl.cs.columbia.edu/publications/nossdav2000_fordist.pdf.*

[56] Object Management Group (OMG). *The Common Object Request Broker: Architecture and Specification (CORBA), revision 2.0*. Object Management Group (OMG), 2.0 edition.

[57] Open Printing, The Linux Foundation. Retrieved August 18, 2006, from Website: *http://www.linux-foundation.org/en/OpenPrinting.*

[58] Parunak, H. V. D.(1996) "Applications of Distributed Artificial Intelligence in Industry," *Foundations of Distributed Artificial Intelligence,* G. O'Hare and N. Jennings, (eds.), Wiley, pp. 139-164. New York.

[59] Pinouts.ru. Retrieved September 10, 2006. Website: *http://pinouts.ru/Slots/USB_pinout.shtml.*

[60] Polite Personal Agents Silvia Schiaffino and Analía Amandi ISISTAN Research Institute. (2006). Universidad Nacional del Centro de la Provincia de Buenos Aires, and Comisión Nacional de Investigaciones Científicasy Técnicas, Argentina, IEEE Intelligent Systems.

[61] Ramesh V, Canfield K, Quirologico S & Silva M. An Intelligent Agent-based Architecture for Interoperability among Heterogeneous Medical Databases. University of Maryland, Department of Information Systems.

[62] Ramesh, V., Canfield, K., Quirologico, S. & Silva, M., *An Intelligent Agent-based Architecture for Interoperability among Heterogeneous Medical Databases.* University of Maryland, Department of Information Systems. Retrieved June 10, 2006, from *http://hsb.baylor.edu/ramsower/ais.ac.96/papers/ramesh2.htm.*

[63] Rapaport W. J, Winkelstein P. & Shapiro C. P, Intelligent Natural-Language Understanding of Patient Medical RecordsScientific Rationale and Plan.

[64] Remote Agents, Nasa Ames Research Center. Retrieved September 10, 2006, from Website: *http://ic.arc.nasa.gov/projects/remote-agent/*

[65] RMI: Remote Method Invocation. Retrieved October 29, 2006, from Website: *http://java.sun.com:80/products/jdk/rmi/index.html.*

[66] Rubini, A. & Corbet, A. (2001). Linux Device Drivers, Second Edition.

[67] Sadoski, D. (1997). Client/Server Software Architectures--An Overview, Carnegie Mellon Software Engineering Institute.

[68] Sharma, D. (2004), Client Server Computing Lecture Notes. Retrieved from Website: *http://www.ise.canberra.edu.au/u4349/lect41.htm*, University of Canberra, 2 June.

[69] Sharma, D. (2006). Multi-Agent Reasoning System Environment, University of Canberra, School of Information Sciences and Engineering.

[70] Simple Object Access Protocol (SOAP). Retrieved October 29, 2006, from Website: *http://www.xml.org/xml/resources_focus_soap.shtml.*

[71] Skype *Homepage.* Retrieved October 10, 2006, from Website: *www.skype.com.*

[72] Steinke, Steve.(1995, December). "Middleware Meets the Network." *LAN: The Network Solutions Magazine 10*, 13.

[73] Suguri, H., Kodama, E., Miyazaki, M., Nunokawa, H., & Noguchi, S. (2001). "Implementation of FIPA Ontology Service", In Proceedings of the Workshop on Ontologies in Agent Systems, 5th International Conference on Autonomous Agents. Montreal, Canada.

[74] The Cloud, Shunra. Retrieved Novemebr 17, 2006, from Website: *www.shunra.com*

[75] The Linux Foundation, *Homepage.* Retrieved September 17, 2006, from Website: *http://www.linux-foundation.org.*

[76] Tseng Chiu-Che Tseng,. Gmytrasiewicz, P. J. (2002). *Real Tim Decision Support System for Portfolio Management.* Retrieved September 10, 2006, from Website: *http://www.cs.uic.edu/~piotr/papers/hicss02-portfolio.pdf.*

[77] Tseng, T., Gmytrasiewicz, P. J. (2002) Real Tim Decision Support System for Portfolio Management. Rereieved February 2005 from Website: http://www.cs.uic.edu/~piotr/papers/hicss02-portfolio.pdf

[78] Universal Serial Bus, InterfaceBus.com. Retrieved September 10, 2006. Website: *http://www.interfacebus.com/Design_Connector_USB.html#b.*

[79] Wanli, M., Tran, D., Sharma, D. & Mathur, A. (2006). *Using Windows Printer Drivers for Solaris Applications – An Application of Multiagent System*, Tenth International KES Conference. pp.1176-1183.

[80] Webopedia, definition. Retrieved October 29, 2006, from Website: *www.webopedia.com/TERM/T/thin_client.html.*

[81] Wilson, S. *Failed IT Projects* (The Human Factor). Retreived January 2006 from Website: http://faculty.ed.umuc.edu/~meinkej/inss690/wilson.htm

[82] Windows XP Professional Product Documentation (2005) Microsoft; Retrieved September 17, 2006, from Website: *http://www.microsoft.com/resources/documentation/windows/xp/all/proddocs/en-us/print_add_tcpip_port.mspx.*

[83] Yahoo Messenger *Homepage,* Retrieved October 10, 2006, from Website: http://*messenger.yahoo.com.*

[84] Yahoo Shopping portal. Retrieved September 10, 2006, from Website: http://*shopping.yahoo.com.*

[85] Yang, S., Nieh, J. & Novik, N. (2001, June). *Measuring Thin-Client Performance Using Slow-Motion Benchmarking*, Columbia University. Retrieved October 29, 2006, from Website: *www.ncl.cs.columbia.edu/publications/usenix2001_slowmotion.pdf.*

[86] Yang, S., Nieh, J. & Novik, N.(2001). *Measuring Thin-Client Performance Using Slow-Motion Benchmarking,* Columbia University. Retrieved October 29, 2006, from Website: *www.ncl.cs.columbia.edu/publications/usenix2001_slowmotion.pdf.*

Publications and Presentations

Wanli, M., Tran, D., Sharma, D. & Mathur, A. (2006). Using Windows Printer Drivers for Solaris Applications – An Application of Multiagent System, Tenth International KES Conference, Bournemouth UK..

Mathur, A. (2005). Sun Ray Thin Clients on Broadband Networks, Research Colloquium, University Of Canberra, School of Information Sciences and Engineering, Canberra Australia

Sharma, D., Mathur, A. & Wanli, M. (2006-2007). Agent Based Services Oriented Architecure for Heterogenous Environments. (Submission to KES 2007 currently being prepared)

Appendix 1

Scenarios of Use
MAPS implementation on Sun Ray
Solaris and Windows

The scenarios are based on MAPS implementation on Sun Ray Server and MS Windows machine to support printers on Sun Ray that are only supported on Windows. The scenarios involve real time Sun Ray users requirements which may include LDAP integration to the solution.

The following assumptions are made:

- Sun Ray users move freely from one terminal to another moving their sessions with them to the new terminal.
- Several printer makes and models may need to be supported on the same Sun Ray network.
- All users currently have permissions to print on any printer participating in the MAPS setup.

Scenario 1 – Base Case

Setup:

Solaris Server configured to run Sun Ray software. PC based Windows machine connected to Solaris server via TCP. Supported device drivers for Printer 1 is installed on the Windows PC.

Sun Ray server is connected to three Sun Ray thin clients. Sun Ray 1 is connected to Printer 1

Use Case:

User is logged in to Sun Ray 1 and requests a print job.

System Response:

The Sun Ray server captures the user request, and forwards it to Windows machine. The Conversion Agent converts the printer script and sends back to Solaris server. Solaris server streams the printer script to the USB port of Sun Ray 1 where Printer 1 is connected.

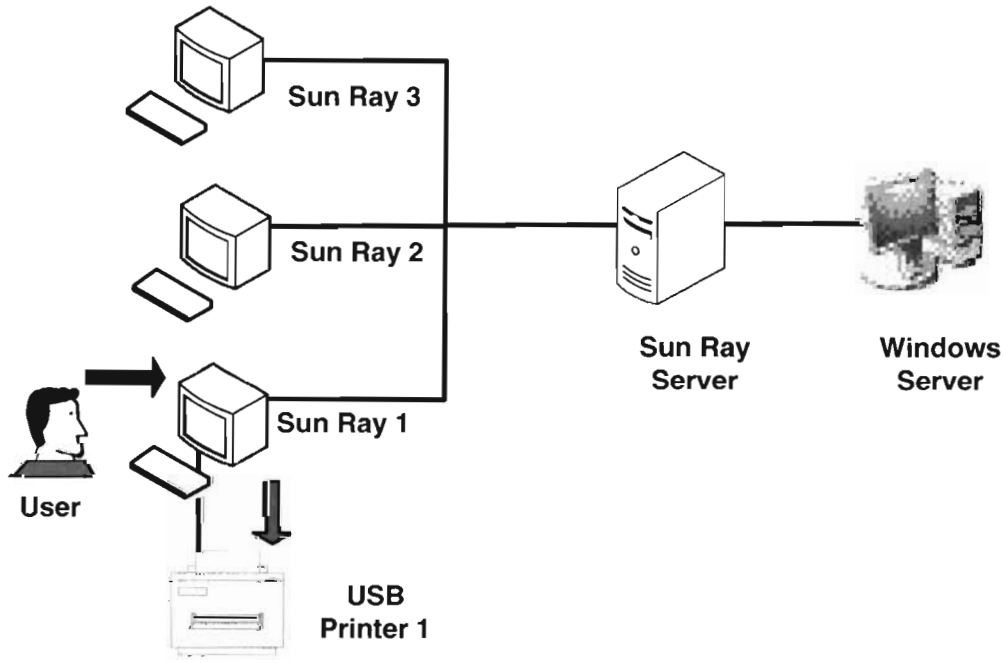


Figure: 24. Scenario1 - Base Case

Scenario 2 – Printer of choice

Setup:

Solaris Server configured to run Sun Ray software. PC based Windows machine connected to Solaris server via TCP. Supported device drivers for Printer 1 and Printer 2 are installed on the Windows PC.

Sun Ray server is connected to three Sun Ray thin clients. Sun Ray 1 is connected to Printer 1 and Sun Ray 3 is connected to Printer 2

Use Case:

User is logged in to Sun Ray 1 and requests a print job for Printer 2.

System Response:

The Sun Ray server captures the user request from Sun Ray 1, and forwards it to Windows machine. The Sun Ray server detects that Printer 2 is the closest printer to User by using an LDAP query. The Sun Ray server sends its meta data with the request to Windows Server. The Conversion Agent converts the printer script and sends back to Solaris server. Solaris server streams the printer script to the USB port of Sun Ray 3 where Printer 2 is connected.

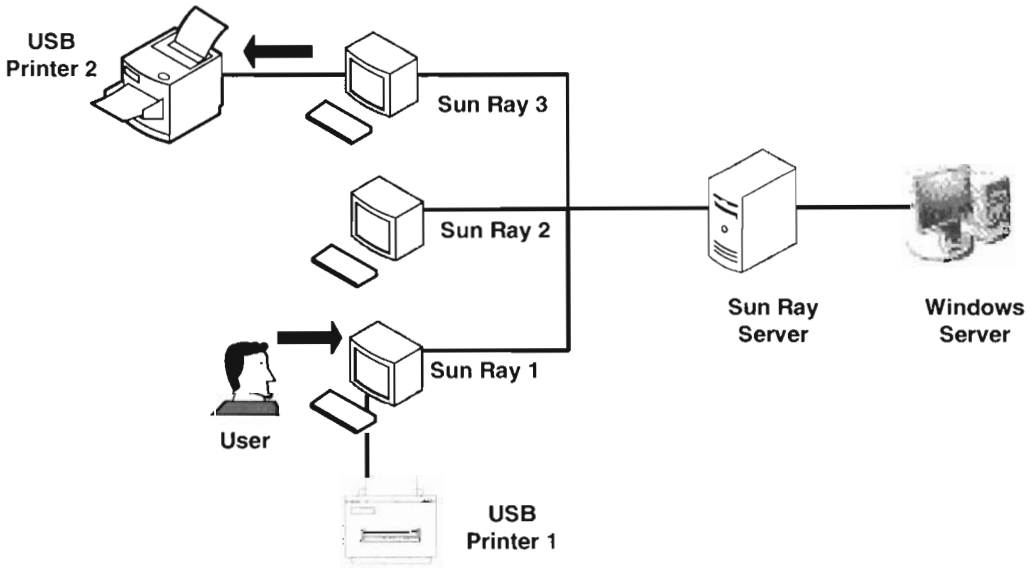


Figure: 25. Scenario 2 - Print to Printer of Choice

Scenario 3 – Nearest printer

Setup:

Solaris Server configured to run Sun Ray software. PC based Windows machine connected to Solaris server via TCP. Supported device drivers for Printer 1 and Printer 2 are installed on the Windows PC.

Sun Ray server is connected to three Sun Ray thin clients. Sun Ray 2 is connected to Printer 2 and Sun Ray 3 is connected to Printer 3

Use Case:

User is logged in to Sun Ray 1 and requests a print job for a Printer geographically closest to him/her.

System Response:

The Sun Ray server captures the user request from Sun Ray 1, and forwards it to Windows machine. The Sun Ray server detects the Printer 2 and sends its meta data with the request to Windows Server. The Conversion Agent converts the printer script and sends back to Solaris server. Solaris server streams the printer script to the USB port of Sun Ray 2 where Printer 2 is connected.

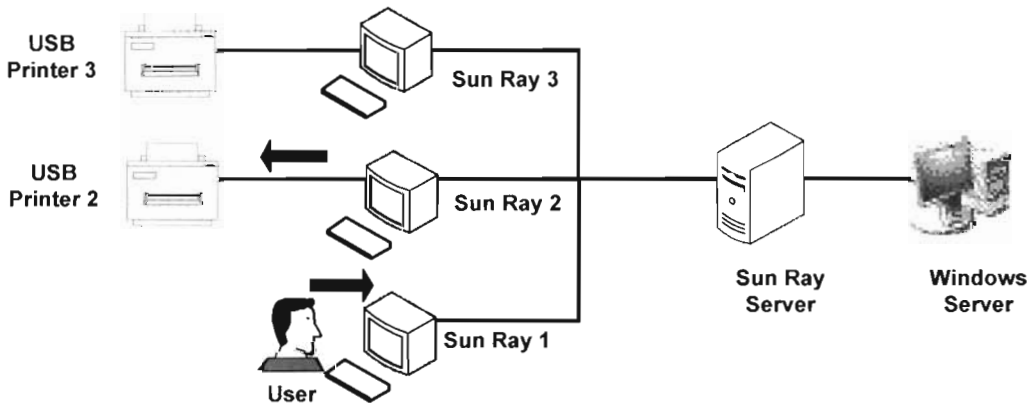


Figure: 26. Scenario 3 - Print to nearest printer

Scenario 4 – Follow me printing

Setup:

Solaris Server configured to run Sun Ray software. PC based Windows machine connected to Solaris server via TCP. Supported device drivers for Printer 1 and Printer 2 are installed on the Windows PC.

Sun Ray server is connected to three Sun Ray thin clients. Sun Ray 1 is connected to Printer 1 and Sun Ray 3 is connected to Printer 3

Use Case:

User is logged in to Sun Ray 1 and later moves his session to Sun Ray 3. He then requests a print job.

System Response:

The Sun Ray server captures the user request from Sun Ray 3, and forwards it to Windows machine. The Sun Ray server detects Printer 2 and sends its meta data with the request to Windows Server. The Conversion Agent converts the printer script and sends back to Solaris server. Solaris server streams the printer script to the USB port of Sun Ray 3 where Printer 2 is connected.

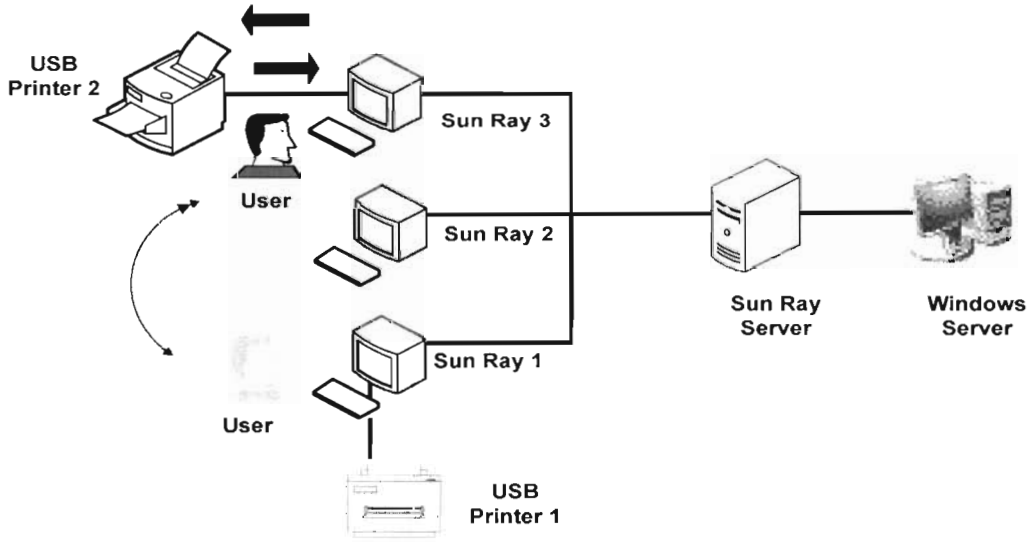


Figure: 27. Scenario 4 - Follow me printing

Appendix 2

Using Windows Printer Drivers for Solaris Applications – An Application of Multiagent System

Proceedings by
Wanli Ma, Dat Tran, Dharmendra Sharma, and Abhishek Mathur

10th International KES Conference, 2006
Bournemouth, UK

Using Windows Printer Drivers for Solaris Applications – An Application of Multiagent System

Wanli Ma, Dat Tran, Dharmendra Sharma, and Abhishek Mathur
School of Information Sciences and Engineering
University of Canberra
{Wanli.Ma, Dat.Tran, Dharmendra.Sharma,
Abhishek.Mathur}@canberra.edu.au

Abstract. This paper proposes using multiagent system technology to solve the problem of printing across heterogeneous operating systems without re-implementing printer drivers. The printing problem comes from a real world application, where we have to print from Solaris operating system applications to printers which only have Windows operating system drivers. Multiple intelligent agents are distributed on both Windows and Solaris platforms to integrate services running on these platforms. They are responsible for printing task interception, operating system spooling, printing format conversion, load balancing, and intelligent data routing. This approach avoids re-implementing Windows printer drivers on Solaris, yet provides seamlessly printing for Solaris applications to those printers.

Keywords: Multiagent, MAS, operating systems, printer drivers.

1 Introduction

One of the major problems facing today's IT operation is the interoperability among heterogeneous operating systems. Popular daily used operating systems, such as Windows family, Linux and Unix family, MacOS family, and mainframe operating systems etc., are not compatible with each other. Some applications only run on certain operating systems. For any sizable organization, it is very likely there is coexistence of heterogeneous operating systems in operation. The situation will stay as it is for a long time yet to come according to ComputerWorld [1].

Significant effort has been invested by the research communities and IT industries to improve the interoperability among heterogeneous operating systems. For example, middleware solutions, notably, Microsoft .NET environment [2], CORBA specification [3], and J2EE specification [4], and distributed operating system approaches, such as [5], are proposed and implemented, yet we are still not be able to easily integrate applications on different platforms. In real life, we are still battling the platform issues, such as how to run application X on platform Y.

The problem we encountered is to provide Solaris Sun Ray users with printing capability. It is from a real life project which aims at providing the community with basic function, almost nil maintenance, and low cost computers and Internet access. Sun Ray thin client technology is chosen as the delivery infrastructure. When coming to printing, most home user level printers on the market are not PostScript compliant. Manufactures have their own proprietary printer drivers. All of the printers have Windows drivers, but none of them has Solaris drivers. The challenge is to make the printers work for Solaris, and therefore, provide Sun Ray users with printing

capability to the printers of their choices from the market. Existing technology fails to provide a neat solution.

Multiagent system (MAS) technology has its root from distributed artificial intelligence [6]; however, the continuous and autonomous nature of agents and the communication among these agents make them a good candidate for distributed computing, and even just general software applications, for example, agent based software engineering [7], autonomic computing [8, 9], and many other distributed applications [10-14].

This paper proposes using multiagent system technology to seamlessly integrate services running on heterogeneous operating systems and thus solve the printing problem. Our proposal is alone the line with the idea of the autonomic computing system proposed by Tesauro et al [8, 9], but our primary focus is different. At this stage, our main task is to integrate Windows printer drivers into Solaris operating system.

The rest of the paper is organized as follows. Section 2 describes the problem in details, and Section 3 presents the multiagent solution. In Section 4, we discuss our prototype implementation and future work. Section 5 concludes the paper.

2 The Problem

The problem is, in essence, printing to low end ink jet printers from Solaris operating system. To describe the problem clearly, we will first briefly explain Sun Ray thin client technology and then the ink jet printers available on the market. Afterwards, the problem becomes clear.

2.1 Sun Ray Thin Client Technology

A Sun Ray thin client unit consists of a monitor, a built-in smart card reader, a network port, an embedded audio speaker, and 4 USB ports, which support keyboard, mouse, and other peripherals. The unit itself has little CPU power and memory capacity. It only needs to recognise keystrokes and mouse events and also to display pixel data received from the server. It is actually just an I/O device: taking keyboard and mouse actions and transmitting them across the network to the server. The server then transmits back the screen pixels, including mouse movements and the characters typed. The actual computing is performed on one or more remote servers, which run Sun Ray Server Software (SRSS) on Solaris operating system platforms. No computing is performed locally.

The absence of application files, operating system, and storage media on the local client defines the stateless nature – which also means that the desktop, i.e., the think client unit, does not have to be administered locally. Thin client users are administered remotely at the server end. Administrators are freed from maintaining individual clients and performing repetitive task of setting up each client unit.

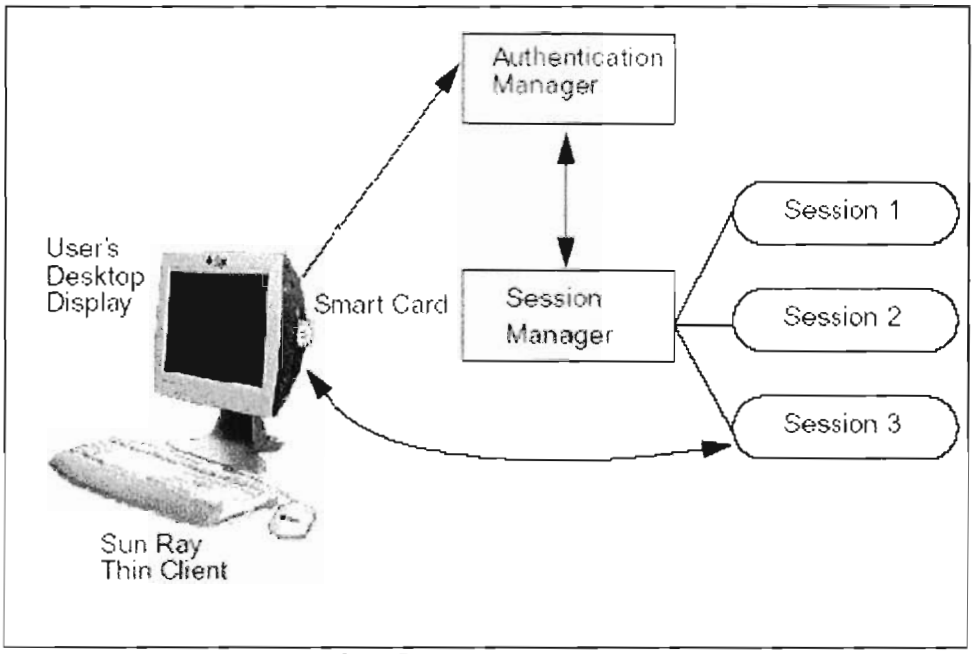


Fig. 1. Sun Ray Session Management

Remote processing requires all sessions to be run at a central server or servers, which maintain and authenticate sessions. Smart card allows hot swapping of sessions between terminals. Session Manager simply switches the session from one terminal to another and provides mobility to users (Fig. 1).

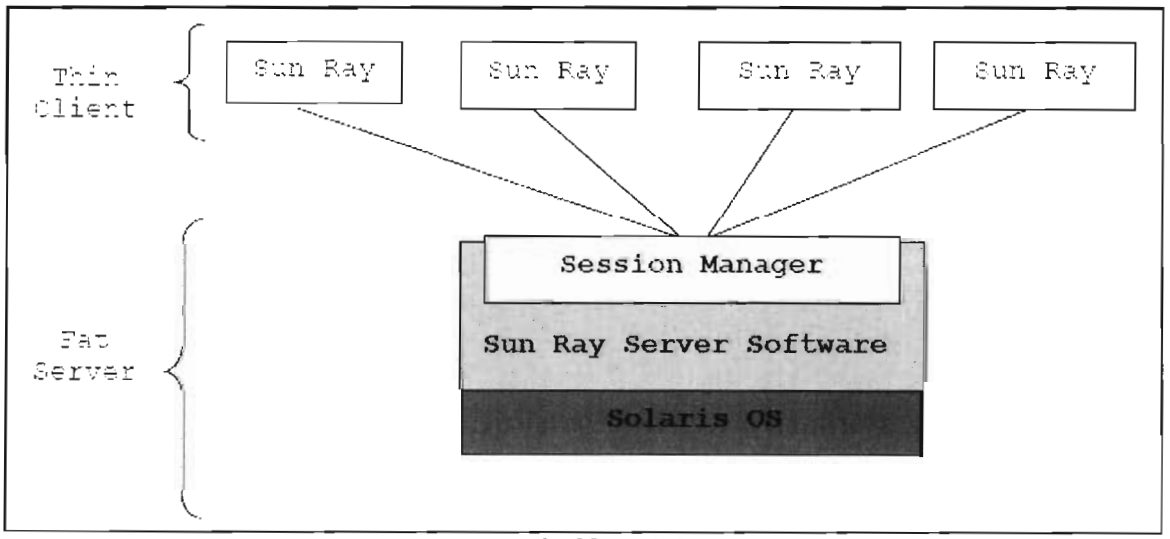


Fig. 2. SRSS and Solaris

The key of this thin client architecture lies in the Sun Ray Server Software (SRSS). It runs on the Solaris OS and is responsible for receiving and transmitting screen refreshes, keyboard stokes, and mouse movements between a client and its session (Fig. 2). It also manages user applications – starts and terminates applications on user behalf.

2.2 The Available Printers

Most industrial strength printers are PostScript compliant and also have their own network cards. They do not have problems with Solaris printing. However, the low end ink jet printers for home users are not PostScript compliant. These printers have very little intelligence themselves. They heavily rely on the printer drivers running on the host computers to do all the necessary operation, including data format conversion. This type of printers is very popular with home users and small businesses. They are cheap to buy, can print on different media, and also can print in colors. A laser printer with the same ability costs a lot more.

Due to their low data processing power, printer drivers become the critical part of operating these printers. All ink jet printers have Windows drivers, but most of them do not have Solaris drivers. It is impossible to buy a randomly chosen printer and plug it into a Solaris machine and hope it will work smoothly. To make things worse, the printers on the market updates very fast just like any other IT products. It basically means that the printer drivers have also to be updated as fast.

2.3 The Problem Encountered

To be able to print, from an ordinary Sun Ray user's point of view, it would be the same as with other personal computers: buying a USB printer (almost certain, cheap ink jet type) and connecting it to the USB port of the Sun Ray client unit. Afterwards, the user would naturally expect printing starts.

In fact, on the contrary, the system has great difficulties to work with the printer. Sun Ray applications are running on Solaris operating system and controlled by Sun Ray Server Software (SRSS), as discussed in the previous section. The printing problem becomes the problem of printing from Solaris operating system to the printer. SRSS can recognise a new USB device being plugged; however, it won't be able to control it without proper driver. We checked all available ink jet printers on Australian market in the middle of 2005 and failed to find any one providing a printer driver for Solaris. The problem further manifests itself into multiple problems:

- The fundamental problem is that there is no proper printer driver for Solaris operating system.
- If there were a printer driver from a particular printer, there is no guarantee of drivers from other models or brands or manufacturers. A Sun Ray user may buy a printer which does not have the driver for Solaris operating system.
- If there were a driver for the current printer, there is no guarantee of a driver for the next updated model.
- The available printer models and the associated printer drivers on the market keep changing.

The lacking of printer drivers is the critical problem we have to solve in order to provide Sun Ray users with printing capability.

3 The Proposed Solution

In essence, to make printing from Solaris possible, we have to find a way to drive the printers. There are 2 possible solutions to this problem. One is to redevelop or port printer drivers for Solaris. The other way is to find an alternative by pass. The former is not a real solution. There are a number of serious drawbacks with the approach:

- Full cooperation from printer manufacturers is needed for redeveloping the drivers; however, it is almost impossible. It is impractical to develop a printer driver without the detailed internal knowledge about the printer's hardware and firmware. Manufactures always treat this knowledge as trade secrete and won't make it public.
- Without the cooperation, the redevelopment work has to rely on reverse engineering. Reverse engineering is tedious and time consuming. It may also have legal ramifications.
- Reverse engineering cannot keep the pace with the market, given the fact of many models, frequent model updates, and many manufacturers on the market, let alone new manufacturers entering into the market.

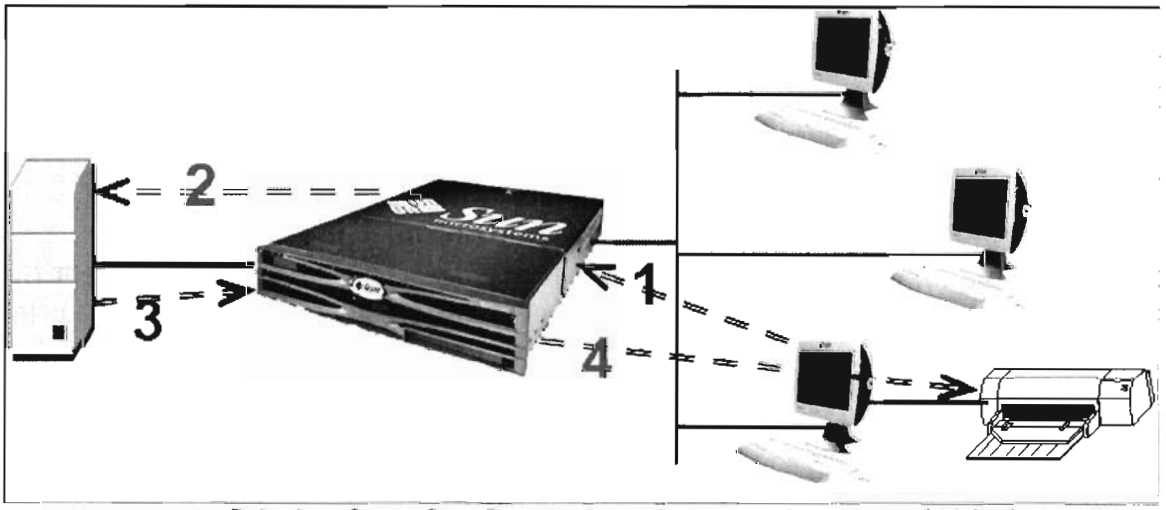


Fig. 3. Printing from Sun Ray to its printer via Solaris and Windows

To avoid all these difficulties, we propose an innovative approach: using Windows printer drivers for Solaris. The idea is simple: a Windows box is connected to a Solaris box (or several Solaris boxes) via network and is working as conversion channels. The Solaris box sends a printing job to a conversion channel, which corresponds to a particular printer. The conversion channel then translates the printing job into the proprietary format, which the printer understands, via its Windows printer driver. The printing job is then sent back to the Solaris box and then to the attached printer (Fig. 3).

Under the proposal, printing now becomes a very complicated task involving multiple operating systems and communication. To coordinate this complicated task,

multiagent system is needed. A number of intelligent agents are distributed on both Solaris and Windows platforms, intercepting printing task, synchronizing operating system spooling, instructing printing format conversion, and also balancing the load and routing the data.

Solaris resident agents are responsible for capturing printing requests and probing for printer device descriptions. They are also responsible for sending the printing requests over to Windows based agents, which perform the tasks of converting printing data to the forms which can be understood by the printers connected to the Sun Ray clients (on Solaris platform). After the conversion, the data is sent back to the Solaris agents in and then to the printer, Fig. 4.

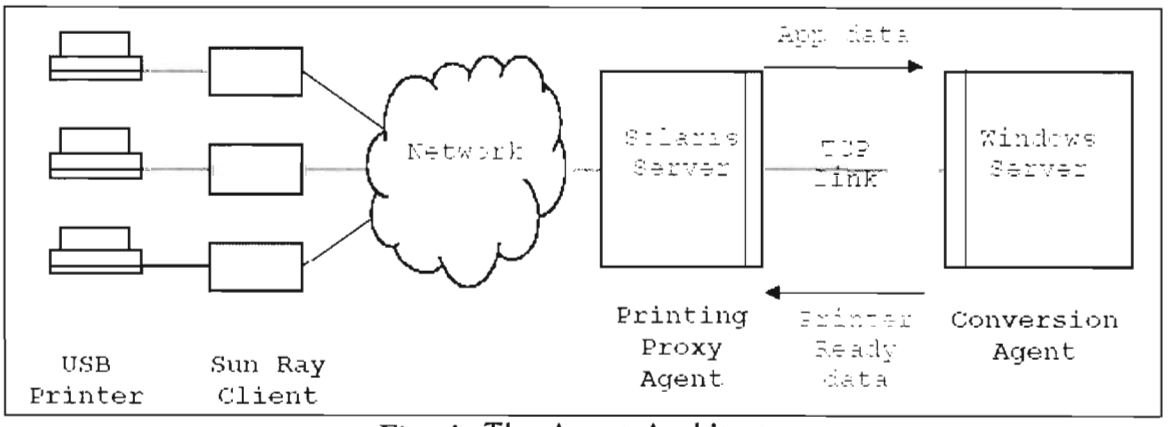


Fig. 4. The Agent Architecture

A printing request is now handled by the agents in the following steps:

- **Intercepting the printing requests from a Sun Ray client**

On the Solaris end, there are many *printing proxy agents*. An agent will intercept the printing request whenever a user performs a printing operation. A virtual print spooler and its associated directories are setup on Solaris to help the interception. The intercepted data will then be sent to a conversion agent on the Windows end. The printing proxy agent has the knowledge of the current user, the current application, and the printer model etc. related information. The knowledge is used to decide to which conversion agent the task will be sent.

- **Data conversion**

On the Windows end, there are many *conversion agents*. Upon receiving a message from a printing proxy agent, a conversion agent will convert the received data into printing format by simulating printing operations of human users on Windows operating system. Of course, the printer driver of this particular printer has to be installed on the Windows machine, but there is on need for a physical printer. With the help of the printer driver, the converted data is in the very format that printer can understand.

- **Transmitting the data back to Solaris and then the printer connected to the Sun Ray client**

The conversion agent is then transmits the converted data back to the printing proxy agent. The data is further forwarded to the printer connected to the Sun Ray client.

4 Prototype Implementation and Future Work

On the surface, the implementation is straightforward – only 2 types of agents. Actually, it is rather complicated. There are a number of factors which complicate the situation. For example, there might be more than 1 Solaris hosts and more than 1 Windows hosts. The agents on these hosts have to negotiate with each the best pair of printing proxy agents and conversion agents, with respect of the current working load of each host and the printer types. In addition, intercepting printing request on Solaris side and mimicking human printing operation on Windows side require programming at the deep hearts of both operating systems.

To prove the concept, at the very beginning, manual interception and redirection of data were performed, and printing was successfully done from Solaris applications to an HP inkjet printer connected to a Sun Ray client. Solaris application data was converted to the printer understandable format via a Windows machine as conversion channel. Up to date, a prototype implementation is very well under the way. So far, the implementation of the agents has been done. The reminding work is the communication among the agents.

Implementing the communication among the agents does not only bring us to the full implementation of the proposed system, it also prompts us to think about the future work.

In the near future, related to the printing problem itself, we need to decide how to do load balance and also if we allow mobile agents. Being able to understand the load of each involved host is critical for real life applications. We haven't yet decided how we are going to tackle the problem in our proposal. It might be helpful to introduce another type of agents – traffic control agents – to coordinate the interactions among the printing proxy agents and conversion agents. More investigation is needed. As to the communication among the agents, both broadcast and peer-to-peer are employed. To allow mobile agents or not is not so critical at this stage. It is more about agent research than solving the printing problem per se. However, we do see the benefit of having mobile agents, especially for load balance purpose and also in ad hoc network environment.

Further down the track, we'd like to test this proxy and conversion agent idea on other applications, such as (i) bringing Windows multimedia formats (e.g., Windows media files) to Solaris and other operating system environments and, in more general sense, (ii) providing Windows like GUI to other operating systems without much programming of redevelopment.

5 Conclusion

The paper proposed to use multiagent technology seamlessly integrating the printing services on Windows and Solaris operating systems. Printing proxy agents, which reside on the Solaris operating system side, are responsible for capturing printing

requests and then pass the captured data and other related information to conversion agents, which reside on the Windows operating system side. A conversion agent is responsible for converting the printing data into the right format the final printer understands. With the effort of printing proxy agents and conversion agents, not only is it possible for Solaris to print to the printers which only have Windows printer driver; but also the whole printing operation is transparent to the end users.

In essence, our proposal is about making applications on one operating system as the pluggable services to the other operating system. Multiagent technology makes the integration smoothly and seamlessly with minimum effort. In the future, we expect to apply this proposal to wider areas of software application integration across heterogeneous computer systems and distributed computing.

Acknowledgments. The research work is supported by The Completion Grant of Division of Business, Law and Information Sciences, University of Canberra.

References

1. Thibodeau, P. *Mixed IT environments remain king with large users.* 2005 [cited 2006
   January]; Available from:
   http://www.computerworld.com/hardwaretopics/hardware/
   story/0,10801,106776,00.html?source=NLT_AM&nid=106776.
2. Microsoft.*NET homepage.* 2006 [cited 2005 December]; Available from:
   http://www.microsoft.com/net/default.mspx.
3. CORBA. *CORBA homepage.* [cited 2005 December]; Available from:
   http://www.corba.org/.
4. Sun. *J2EE homepage.* [cited 2005 November]; Available from:
   http://java.sun.com/javaee/index.jsp.
5. Kon, F., et al. *2K: A Distributed Operating System for Dynamic Heterogeneous Environments.* in *9th IEEE International Symposium on High Performance Distributed Computing (HPDC'00).* 2000. Pittsburgh, USA: IEEE Press.
6. Stone, P. and M.M. Veloso, *Multiagent Systems: A Survey from a Machine Learning*
   Perspective. Autonomous Robots, 2000. **8**(3): p. 345-383.
7. Jennings, N.R., *On agent-based software engineering.* Artificial Intelligence, 2000. **117**: p. 277-296.
8. Kephart, J.O. and D.M. Chess, *The vision of autonomic computing.* Computer, 2003.
   36(1): p. 41-52.
9. Tesauro, G., et al. *A Multi-agent Systems Approach to Autonomic Computing.* in *Third International Conference on Autonomous Agents & Multiagent Systems (AAMAS'04).* 2004. New York, New York, USA: ACM press.
10. Honavar, V., L. Miller, and J. Wong. *Distributed Knowledge Networks.* in *IEEE Information Technology Conference.* 1998.

11. Pan, L., et al. *Mobile Agents – The Right Vehicle for Distributed Sequential Computing.* In *9th International Conference on High Performance Computing – HiPC2002.* 2002. Bangalore, India: Springer-Verlag.
12. Martin, D., A. Cheyer, and D. Moran, *The Open Agent Architecture: a framework for building distributed software systems.* Applied Artificial Intelligence, 1999. 13(1/2): p. 91-128.
13. Drashansky, T., et al., *Networked agents for scientific computing.* Communications of the ACM, 1999. 42(3): p. 48-53.
14. Zhang, Z., et al. *Multiagent system solutions for distributed computing, communications, and data integration needs in the power industry.* in *General Meeting of the IEEE Power Engineering Society.* 2004: IEEE Press.

Appendix 3

Sun Ray Thin Clients on Broadband Networks

A report by Abhishek Mathur
Research Colloquium 2005, School of Information Sciences and Engineering,
University of Canberra,

Information Sciences Extension Studies (4420) PG

Semester 1, 2004

Sun Ray Thin Clients on Broadband Networks

Version 1.4

By:

Abhishek Mathur (114760)

Supervisors:

Dr Dharmendra Sharma

Dr Wanli Ma

UNIVERSITY OF CANBERRA

I Table of Contents

II Acknowledgements

I have been supported by Scott Carr of ActewAGL, my supervisors Dr Dharmendra Sharma and Dr Wanli Ma throughout the preparation of this report. ActewAGL and University of Canberra have generously provided funds, resources and support for the Sun Ray laboratory and other equipments required to complete this project. I am specially grateful to Nick Allan and Cristian Villalon of Sun Microsystems, who provided me the resources at Sun labs at Canberra. Their insight and support were invaluable and greatly improved the final design and content of the report.

Dane Fatouros, also of ActewAGL, has offered valuable guidance in understanding the existing TransACT network and utilising it to implement the Sun Ray architecture over a Wide Area Network.

1.0 Executive Summary

While many vendors are suggesting thick client architecture, thin clients have shown promising performance results over broadband networks. Sun Ray in particular has shown to deliver excellent performance on application benchmarks. This demands the exploration of different types of application and devices to be tested on this system.

Sun Ray architecture is highly suitable for schools, and basic PC user market segment where the users can be freed from day to day chores of maintenance and upgradation. Smarter viruses and complicated maintenance procedures quite often over power the users and while the real computing tasks are sidetracked.

Although Sun Rays are originally meant to work on Local Area Networks, with little innovation and tweaking, it has been possible to deploy these ultra thin clients over broadband networks. To cut down the cost, it is recommended to upgrade the Sun Ray's firmware with a PPPoE client capable of routing the boot up sequence to the correct server on the Internet.

With the use of Sun Ray's LibUSB, it has been possible to port Linux GPhoto 2 digital camera drivers on to Sun Rays. It is anticipated that SANE and other applications can also be utilised through similar processes.

Performance of Sun Rays compared with other thin clients have shown to deliver excellent user perceived performance.. Even on a broadband network, when server and clients are virtually on the other sides of the earth, the results have shown acceptable performance. However there may be a need to test the appliance over TransACT's broadband network to tweak the bandwidth for optimum performances.

2.0 Introduction

The aim of this report is to study and understand the feasibility of utilising Sun Ray thin clients over a broadband network. The report highlights the efficiency of Sun Rays over a broadband network and also suggests ways of utilising USB devices primarily built for non-Solaris based platforms. We also compare the data transfer rate and latencies of other thin clients with Sun Ray and deduce the areas of use for Sun Rays on a Wide Area Network. Focusing on the target user-segment, we propound the nature of applications that would be deemed fit and look at the functional requirements for successful installation of such applications.

With decreasing prices and bottlenecks in the broadband Internet access, the Sun Ray architecture exploits this opportunity of utilising broadband capabilities and cutting down costs incurred in maintenance and house keeping of computer equipment. Home computing can now be as simple as using a utility service such as a telephone, or electrical appliance, requiring virtually no maintenance at the client end. This is possible through centralised and remote administration of Sun Ray thin clients.

In order to make this service viable, it demanded a need to evaluate home-computing applications over a broadband network. Plug and play USB devices such as digital cameras and scanners have become a necessary part of computing and their functionality needs to be tested on the Sun Ray architecture.

The study also reports on the architecture of Solaris Operating System and how Sun Ray thin clients' USB ports interact with application software. Two common USB device drivers are discussed: Open Sourced GPhoto 2 and USB Mass Storage drivers. It also suggests ways of porting GPhoto 2 drivers to Solaris for use with Sun Rays.

3.0 Prospective Users

Sun Ray architecture allows remote administration leaving the users trouble free. All applications on the client are processed on the server and every user action is sent back to the server for processing. Such a setup is ideal for novice users who can not be bothered with day to day house keeping of a PC. Guarding a personal computer against smarter viruses and managing complicated applications is a daunting task for most users. The chore of a computer management quite often overpowers the actual computing needs.

In a typical scenario, a school has a Sun Ray set up at the student labs and students also have a Sun Ray client at their home connected over an ADSL connection to the same server as the school. A student can work in the school labs and use the same smart card and logs in the home Sun Ray. This allows transfer of student's session from the school terminal to the home terminal with just a swipe of a card, allowing the student to continue the classroom work at home. More over, the school staff or the students are free from the responsibility of housekeeping of their systems.

In another scenario, a home user requiring word processing, email and internet browsing can work on a Sun Ray at home connected over a broadband network to a remote server managed by an application service provider. The user pays a minimal monthly charge for the applications, which are subscribed. The terminal works similar to any electrical appliance, which is ready to operate as soon as it is plugged into the socket on the wall. Once again, the user is free from all system administration and maintenances.

Sun Rays are already used extensively in the corporate world and have significantly improved commercial services. An example is the Perth based construction company BGC, which had ousted half of its fleet of Wintel PCs in favour of thin clients running Solaris in order to "go forward" with service delivery to the company's employees. [1]

4.0 Thin Clients versus Fat Clients

Thin and fat clients in Client Server architecture may be defined by the use of tires. Tiers are used to describe the logical partitioning of an application across clients and servers. In 2 tier architecture, processing load is split into two. Majority of the application logic runs on the client, which typically sends SQL requests to a server – resident database. This is fat client architecture, because a chunk of the application runs on the client side. 3 tier splits the processing load between the clients that run the Graphical User Interface logic, application server (running logic) and the database or legacy application. 3 tier moves the application logic to the server – called the fat server or thin client. [2]

Thin client is the hardware at the client end in a client - server model. This client is designed to be small and light so that bulk of the data processing is carried out at the server. The term 'thin client' is becoming increasingly popular. It inhabits the territory owned by Netscape and Sun Microsystems promoting Java based applications running on thin clients on networked computers. On the other side, the territory owned by Microsoft and Intel, is building even larger applications running locally on the desktop computers.

The term 'thin client' is also used for software, but is more often used for computers such as networked computers, which are designed to serve as the client for client / server architecture. A thin client is a computer without a hard disk, and even memory and a microprocessor, whereas a fat client includes disk drive and has all the hardware needed to do bulk processing locally. However, the data itself is stored on the servers dedicated to provide file storage services. [3]

Most enterprises deploy thin client architecture to reduce the Total Cost of Ownership (TCO). By centralising services at a remote server, administrative tasks are accessible locally and are carried out more efficiently.

5.0 The Sun Ray Architecture

Sun Ray ultra thin client consists of monitor, keyboard, mouse, a built-in smart card reader, 10/100 BaseT network port, an embedded audio speakers and USB ports to support keyboard, mouse and other peripherals. These provide users with access to all the applications and utilities they normally use on their workstations or PCs. The actual computing, however, is performed on one or more remote servers. The physical desktop unit - the thin client itself - needs only enough memory and computing power to recognise keystrokes and mouse events and to display pixel data received from the server. No computing is performed locally.

Sun Ray is simply an input device, taking keyboard and mouse actions and transmitting them across the network to the Solaris server. The server then transmits back the screen pixels, including mouse movements, characters typed, and any audio back to the Sun Ray.

Performing all CPU intensive computing tasks on a large server allows a great deal of work to be done with considerably greater efficiency - in terms of time, money and hardware vis-à-vis on a multiplicity of individual workstations or PCs at the same time. [4]

Absence of application files, operating system, and storage media on the client defines the stateless nature – which also means that the desktop does not have to be administered. Users are administered centrally at the server. Administrators are freed from physically maintaining individual clients and performing repetitive task of setting up each client machine. [4]

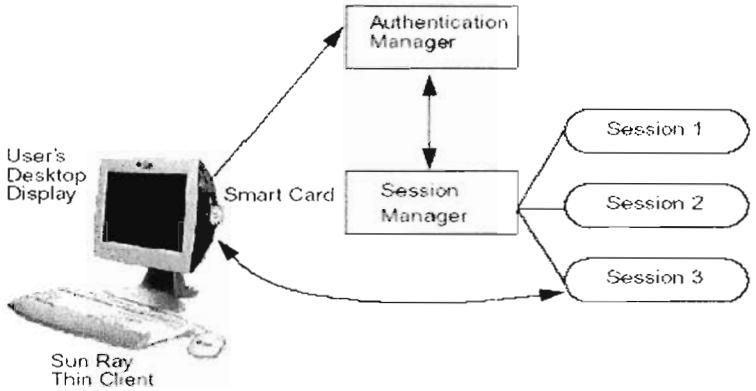


Figure 1: Sun Ray Session Management

Remote processing allows all sessions to be run at a central server, which maintains and authenticates session logins. Smart card allows hot swapping of sessions between terminals. Session Manager simply switches the session from one terminal to another and provides true mobility to users (Figure 1). Sun Rays on a Wide Area Network can totally eradicate the necessity of laptops and users may now carry only a smart card for hot decking! When combined with a separate Microsoft Terminal Server with Citrix MetaFrame, along with Sun Ray's support of the Citrix ICA software client, the Sun Ray can also run Windows applications [8], [9].

5.1 Sun Ray and Solaris

The key of this thin client architecture lies in the Sun Ray Server Software (SRSS). This layer of software runs on the Solaris OS and is responsible for transmitting screen refreshes, keyboard and mouse movements to the clients (See Figure 2). The client sends back the mouse clicks and keyboard strokes to the Server, which is then processed at the server side.

The Sun Ray Server Software 1.2 is compatible with both 32 and 64 bit applications of Solaris 8 operating environments. Common Desktop Environment (CDE) is pre-bundled with the system; however it also supports KDE, Java Desktop and other common desktops.

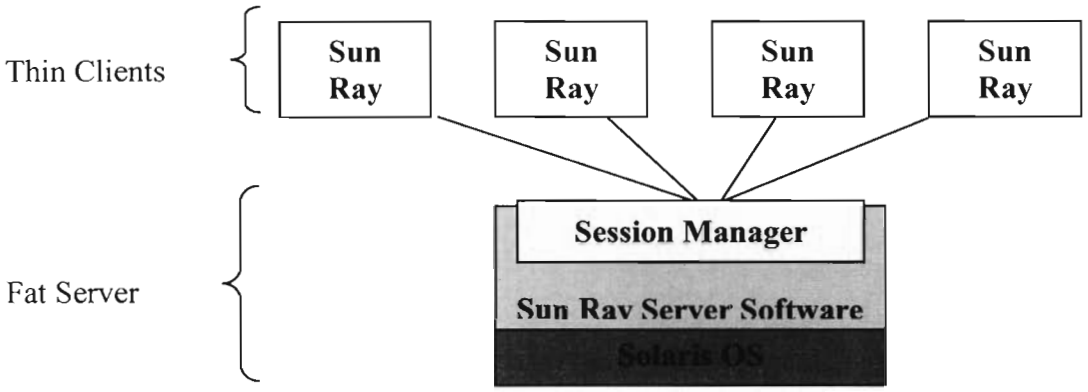


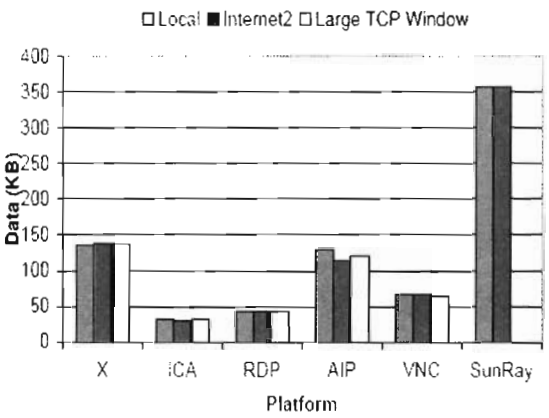
Figure 2: SRSS and Solaris

5.2 Solaris Application and Sun Ray

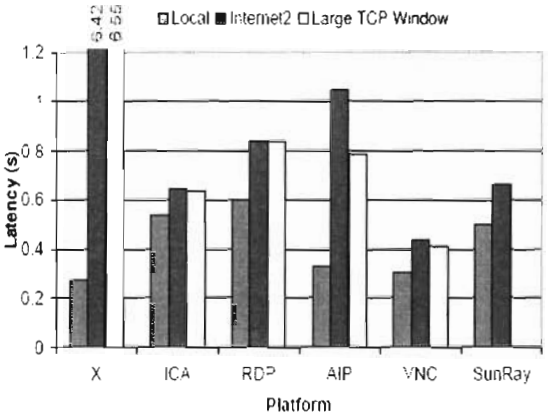
Sun Rays running are capable of running any application supported by Solaris. Sun Ray Server Software does the interpretation of resolving the applications "screen delta" (screen changes) and transmitting to the client. This saves on the bandwidth by sending only the changes instead of entire screen. However, graphic intensive and multimedia applications requiring high screen refresh rates will demand larger screen deltas and hog on network traffic.

6.0 Sun Ray on Wide Area Network

Sun Rays are designed to work best on a Local Area Network; however, with little innovation and tweaking, they have been tested to work efficiently on a Wide Area Network too.



Web Data Transfer [5]



Web Latency [5]

Figure 3 : Comparative Analysis of Thin Client performance

The results in Figure 3 plot the web data transferred and latencies of X, ICA, RDP AIP, VNC and Sun Ray thin clients. These results were conducted using the Slow Motion Bench Marking [9]. This research conducted by University of Columbia shows that using thin client computing in a WAN environment can deliver acceptable performance, even when client and server are located thousands of miles apart on opposite ends of the country. Sun Rays have shown to deliver excellent performance on all application benchmarks. Although, performance varies widely among thin-client platforms and not all platforms are suitable for WAN environment.

Measurements have shown that the bandwidth efficiency of a thin-client system is not a good predictor of performance over broadband network. It is shown that Citrix ICA and Microsoft RDP usually transfer less data overall for each benchmark compared to the other systems while Sun Ray typically transferred the most amount of data overall for each benchmark. However, in terms of user-perceived performance, Sun Ray significantly outperformed both ICA and RDP over broadband. [5], [7], [8]. [9], [11]

We had also conducted a test of "Proof of Technology" outside laboratory environments by setting up a Sun V20 server at TransACT House telecommunication facility which was given a static IP address. Sun Ray Server Software was installed on the server and clients were connected at ActewAGL House through an ADSL connection over 1 Meg connection. (Figure: 4)

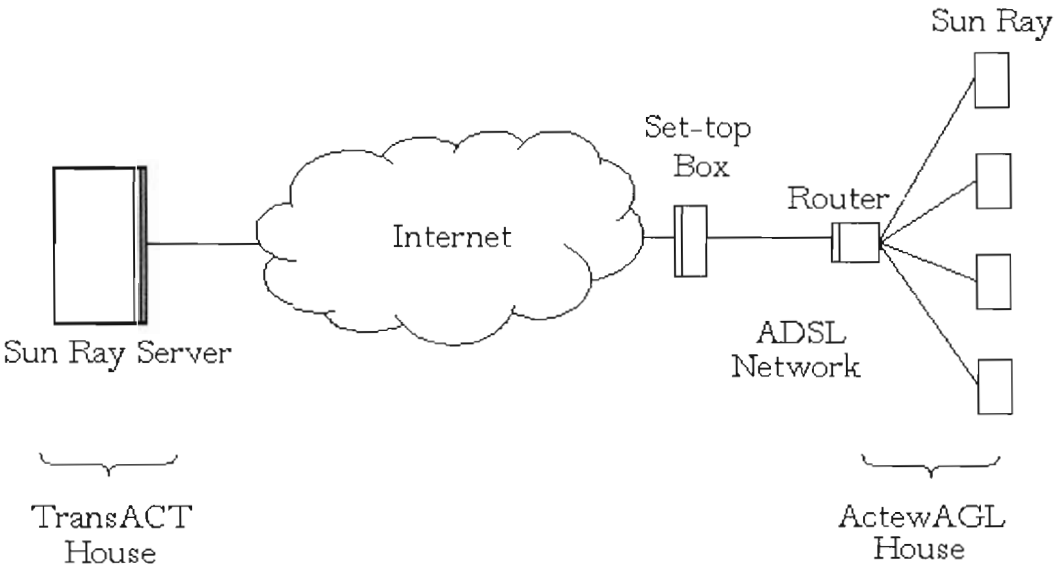


Figure 4: Sun Rays on Wide Area Network

The main concern on a broadband network is for graphic intensive applications which require large screen refreshes sent to the clients. On a slow network, (256Mbps or lower) the efficiency deteriorates, while on faster nets (1000 Mbps or higher) the performance shoots up. Since clients have no compression software in their firmware, heavy applications dominate the network bandwidth forfeiting the performance.

Use of Sun Rays over broadband network is currently not designed for CAD/CAM or Multimedia applications. Most home users and schools however would be most suited for this architecture.

6.1 The Missing Link (PPPoE client)

Sun Rays "listen" for a local Sun Ray server, and attempts to retrieve its IP. On a WAN, however, where the server is not available locally, Sun Rays will have to be routed to the correct server by telling them the IP address. This can be currently done by adding a router in series, which routes the packets to the Sun Ray server. This addition in hardware is a major cost hike to this architecture. To bring the cost down, there could also be a firmware upgraded with a (Point to Point Protocol over Ethernet) PPPoE client which can route the terminal to its server. This firmware could be installed in the Sun Ray PROM. Although, this routing is required only for the initial handshaking between the client and the server, it is a critical step to start the Sun Rays. Figure 5 depicts the network diagram post PPPoE client installation, eradicating Routers.

Sun Microsystems have engaged resources towards these requirements and Engineers are expected to demonstrate a working version within the next month. Performance of Sun Rays with this upgrade will be exciting to monitor.

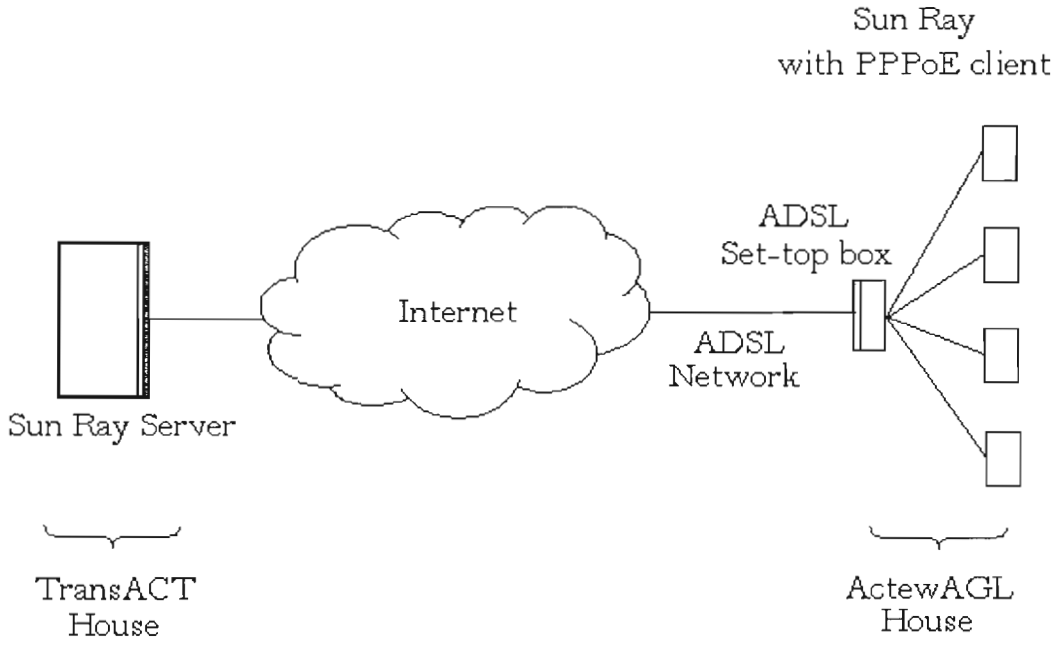


Figure 5: Sun Rays with PPPoE client on Wide Area Network

7.0 Sun Ray USB Architecture

One of the unique things about Unix operating system is that it regards everything as a file. These files are of three categories; ordinary or plain files, directories, and special or device files. Device files include things such as disks, cdroms, tapes, terminals, serial ports, and sound cards. All files contain data of some kind.

In Sun Ray architecture, each terminal's file systems are mounted under a separate node. Usually this node is denoted by the MAC (physical) address of the Sun Ray terminal. All USB ports are mounted under this node for each Sun Ray.

Currently, there are two ways to access a USB device on a Sun Ray; through Kernel level USB drivers or by Application level drivers. Kernel drivers provide basic I/O operations to devices and are generic. Application drivers may be written and designed for specific device and are more specific.

One of my research efforts was to port Linux GPhoto 2.0 drivers to Sun Rays using Sun's LibUSB interface. (Discussed on the next section)

7.1 LibUsb Applications

LibUsb is an open sourced software project, maintained at libusb.sourceforge.net. The aim of this library is to provide user level applications access to USB devices regardless of the Operating system [6]. Most application level USB drivers on Linux are written utilising LibUSB and it seems to be becoming a de-facto standard in the Linux world. This API is the lowest level of software interacting with the OS and provides device handles to applications / drivers running above this layer. Figure 6 illustrates the different layers of software interacting at different levels. LibUSB works for both USB and parallel interfaces, however, in Sun Ray architecture we are only interested in USB ports. [6]

Sun Rays having a significantly different architecture require a separate version of LibUSB, which is currently being developed by Sun Microsystems. The pre-release version of LibUSB for Sun Ray was used in successfully porting GPhoto drivers to Sun Ray.

Since LibUSB for Sun Rays have same interface as that of Linux and open sourced operating systems, most applications running on Linux utilising LibUSB, in theory, can also be ported to Sun Rays.

Figure 6 shows different layers of applications compiled over LibUSB for Sun Rays. GPhoto 2 is a well-known open sourced digital camera driver and has been shipped with various flavors of Linux.

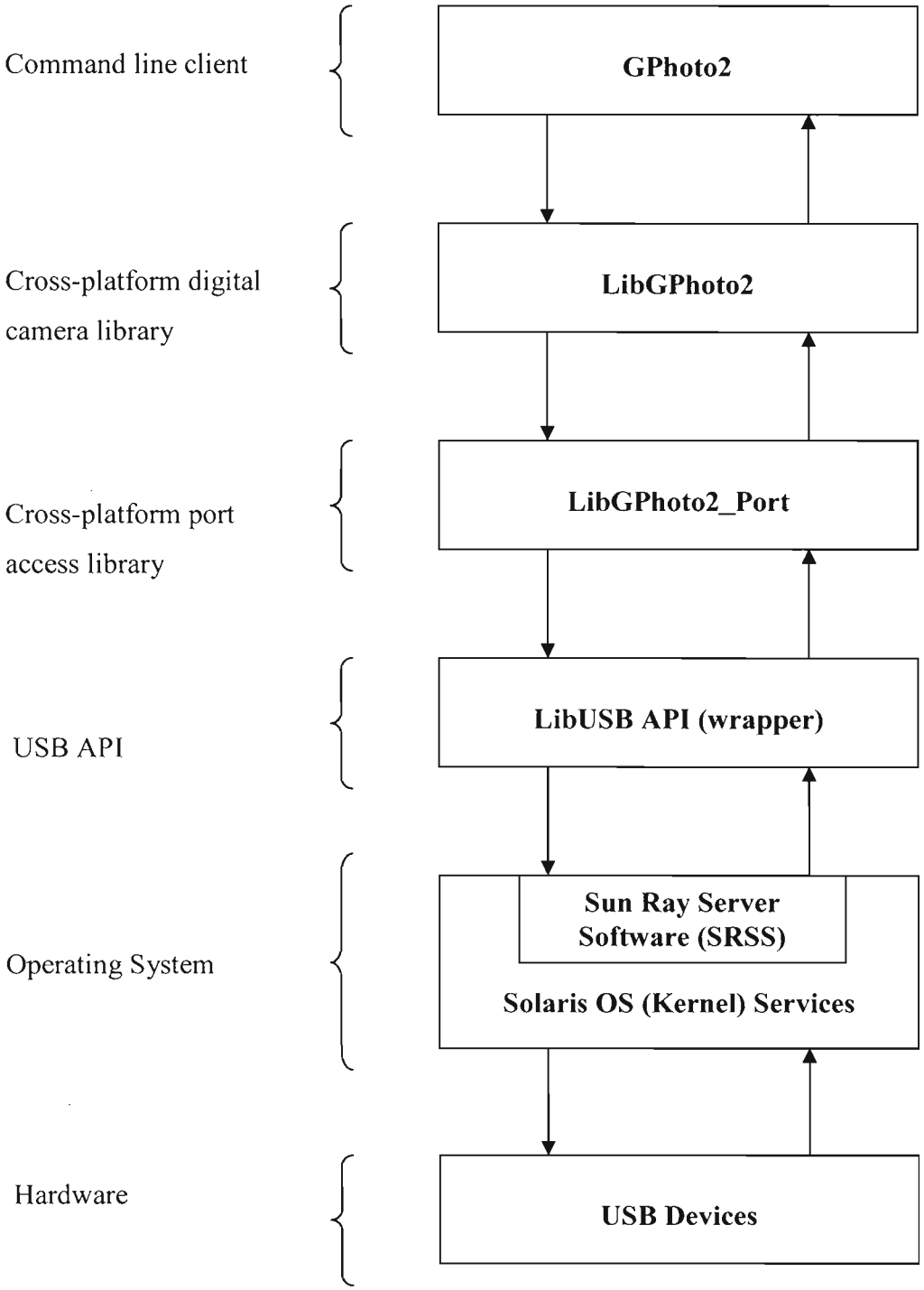


Figure 6: LibUsb and GPhoto architecture

8.0 Future Work

One of the main task which could significantly enhance the usability of Sun Rays in home computing is a Media Viewer application. We could utilise the LibUsb / GPhoto applications ported from Linux or use the USB Mass Storage drivers for its implementation. There is a dire need of a single application capable of reading picture, sound and movie clips from a device.

Compatibility with famous MP3 players such as Apple's iPOD or iTunes may be yet another project.

On a different note, it is also required to test the efficiency of this thin client on TransACT's broadband network to capture the real performance on varying bandwidth and latencies. It is anticipated that the results may be similar to the experiments conducted in simulated environment by University of Columbia. However, to tweak the bandwidth issues for optimum performance, conducting test of efficiency in a laboratory environment is necessary.

9.0 Conclusion

Sun Rays have a promising career in the basic user PC market, as they eliminate the housekeeping and administrative needs by a PC. True session mobility through the use of smart cards demonstrates suitability of this architecture in schools and offices.

Performance of Sun Rays when compared with other thin clients have shown to deliver excellent performance on all application benchmarks. Even on a broad band network, when server and clients are virtually on the other sides of the earth, the results have shown acceptable performance.

A vital component for a successful pilot on broadband networks is a PPPoE client. A firmware upgrade with this client would significantly reduce the cost by eliminating the need for a router for each Sun Ray.

GPhoto 2, a digital camera driver written originally for Linux, have been successfully ported on Sun Rays by utilising LibUsb API for Sun Rays. It is anticipated that similar processes may also utilize other such applications

10.0 Bibliography

[1] Gedda R, Switch to thin clients boosts builder's productivity, Computerworld 21/05/2004 08:12:00 as sighted on 1st Jun 2004 at http://www.computerworld.idg.com.au/index.php/id;77507880;relcomp;1

[2] Sharma D, University of Canberra, Client Server Computing Lecture Notes. As sighted on 2 June 2004 at http://www.ise.canberra.edu.au/u4349/lect41.htm

[3] Webopedia, definition. As sighted on 2 June 2004 at www.webopedia.com/TERM/T/thin_client.html

[4] Sun Microsystems; Sun Ray Overview, April 2003. As sighted on 28 August 2004 at wwws.sun.com/sunray/whitepapers/SunRay_WP042403.pdf

[5] Lai A, Nieh J; Limits of Wide-Area Thin-Client Computing June 2002. As sighted on 4 April 2004 at www.ncl.cs.columbia.edu/publications/sigmetrics2002_i2thin.pdf

[6] LibUSB Documentation. As sighted on 15 January 2004 at http://libusb.sourceforge.net/documentation.html

[7] GPhoto 2 Documentation. As sighted on 15 January 2004 at http://www.gphoto.org/doc/

[8] Jae S, Nieh J, Selsky M, Tiwari N; Columbia University. The Performance of Remote Display Mechanisms for Thin-Client Computing. June 2002. As sighted on 18 January 2004 at ww.ncl.cs.columbia.edu/publications/usenix2002_fordist.pdf

[9] Yang S, Nieh J, Novik N; Measuring Thin-Client Performance Using Slow-Motion Benchmarking, Columbia University. June 2001. As sighted on 18 January 2004 at www.ncl.cs.columbia.edu/publications/usenix2001_slowmotion.pdf

[10] Jason N, Yang S, Novik N; A Comparison of Thin-Client Computing Architectures. November 2000. As sighted on 02 Februaru 2004 at www.ncl.cs.columbia.edu/publications/cucs-022-00.pdf

[11] Nieh J, Yang S; Measuring the Multimedia Performance of Server-Based Computing. As sighted on 18 January 2004 at www.ncl.cs.columbia.edu/publications/nossdav2000_fordist.pdf