UniPlug: A Framework For Ad-hoc *Invention* Sharing Over A Campus Network

by

Durga Prasad Pandey

B.Tech., Indian School of Mines Dhanbad (2004)

Submitted to the Program in Media Arts and Sciences,
School of Architecture and Planning,
in partial fulfillment of the requirements for the degree of

Master of Science in Media Arts and Sciences

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2007

Author_____
Program in Media Arts and Sciences
August 20, 2007

Certified by_____
Andrew B. Lippman
Senior Research Scientist
MIT Media Laboratory
Thesis Supervisor

Accepted by_____
Deb Roy
Chairperson, Departmental Committee on Graduate Students
MIT Media Laboratory

UniPlug: A Framework For Ad-hoc *Invention* Sharing Over A Campus Network

by

Durga Prasad Pandey

Submitted to the Program in Media Arts and Sciences,
School of Architecture and Planning,
on August 20, 2007, in partial fulfillment of the
requirements for the degree of
Master of Science in Media Arts and Sciences

Abstract

In the future, there will be a large number of devices, and most of us will own more than one(some of us already do). Many individual innovators write programs that exploit features on such devices for interesting, non-standard uses. Usually such inventions are lost over time. In this thesis, we propose a framework called UniPlug that encourages rapid and ad-hoc sharing of such inventions. It works by providing distributed repositories to make inventions publicly available, and providing an easy to use client that locates and fetches inventions for devices that a user owns. We begin by introducing the problem and related work. We then formulate the problem technically and design a solution. This is followed by the description of the implementation of a proof of concept. Further, we discuss its applicability to disseminating inventions for medical devices along with an example scenario. We conclude by summarizing this work, and briefly describing our planned future work.

Thesis Supervisor: Andrew B. Lippman
Title: Senior Research Scientist, MIT Media Laboratory

UniPlug: A Framework For Ad-hoc *Invention* Sharing Over A Campus Network

by

Durga Prasad Pandey

The following people served as readers for this thesis:

Thesis Reader_____

David P. Reed
Adjunct Professor of Media Arts and Sciences
MIT Media Laboratory

Thesis Reader_____

Chris Schmandt
Principal Research Scientist
MIT Media Laboratory

Acknowledgements

I have been fortunate to have Andy Lippman as my advisor. Andy provided me the freedom to pursue an independent line of research, while questioning my assumptions and pointing out possibilities to explore. He has managed to create a friendly yet challenging atmosphere in the group which provided a thoroughly enriching experience. David Reed, my co-advisor and thesis reader, has been an amazing source of wisdom and insight, and I have learnt a lot listening and discussing ideas with him. His curiosity is infectious, and he has deeply influenced my way of thinking. Chris Schmandt, my thesis reader, invited me to speak to his group last year, and I have become his fan ever since. Chris is highly knowledgeable, besides being a warm, affable person. Pattie Maes reviewed my thesis proposal and her sharp, critical comments helped me immensely in revising it into a much better form. Thank you all for being such great mentors.

Thanks to Rick Shrenker, Dr Julian Goldman and Susan Whitehead from the Medical Device Plug and Play program for all the enlightening discussions I have had about connecting medical devices with you. I am grateful to Mads Emborg and Sean Moore from Avaya from your help with providing equipment and software for the demonstration.

Thanks to all my friends and collaborators here at the lab, including Hector, Vyzo, Kwan, Fulu, Grace, Dawei, Paulina, Philipp, Isha, Manas, Sung, Alea, Seth, Mihir, Orkan, Elan, Pol, Nadav, Hoda, Ilia, Rachel, Akshay, Ashish, Carlos and many others. I owe extra thanks

to Hector and Kwan for always being willing to help. Thanks are also due to Jitendra, Shulabh, Ruchi, Yagnick, Priyank, Reena, Manish, Himanshu, Pallav, Birendra, Avesh, Sanjeev, Nidhi, Monosvita, Kuber, Rupam, Priya, Helen, Kristi, Lien, Kristin, David, Pankaj, Larry, Rosie, Anton, Andrea, Mithila, Anna, Zeb, Maurizio, Ambar, Anshuman and many others for making my life in graduate school easier. To Bill Thies I owe a special debt of gratitude for being a good friend and mentor.

Thanks to Yagnick for your suggestions on system design and for being my programming guru, and to Shulabh for your sharp analysis of my ideas.

My academic and research career has benefitted a lot from the mentoring and support I received from a number of people. I'd like to express my heartfelt thanks to Scott, Len, Vint, Chris, Adrian, Leigh, Achim, Vivek, Sandy, Andreas, Subrata, Mahesh, PJ, Grit, Jen and Keshav. Thanks are also due to my colleagues from the Digital Vision program, especially Mans, Margarita and Steve for being such great friends and mentors.

The Media Lab staff is just awesome and I'd like to thank you all. Special thanks to Deb, Sandy, Gigi, Tesha, Nikki and Linda for making my life so easy. Will and Tom from NecSys helped me many times and I thank you both.

Finally, my family's love and support has been my inspiration and strength. My mother taught me the value of education and compassion. My father has been a symbol of honesty and hard work. My sisters worked hard to not let humble roots come in the way of my getting a quality education. Thank you guys.

This thesis is yours, as much as it is mine.

To Pitaji and Ma

Contents

List of Figures

Chapter 1

Introduction

1.1 Motivation

In the future, there will be a large number of electronic devices around us, and each individual will own a number of them(some of us already do). There already has been a tremendous increase in the number of devices available of various types that can either be plugged in(USB, Ethernet, Serial) or connected wirelessly(Bluetooth, WiFi) to a computer. Most of these devices have identification by which they can be uniquely defined, such as Class ID and Serial number. On the other hand, there are a large number of high level programming languages today, and individuals routinely discover new device capabilities, write programs that exploit them, and if they are Internet-savvy, release them as open source code.

For instance, students at MIT write a wide variety of interesting programs during their stay here that usually get forgotten over time and are never visible to most of the MIT population. Some of them, for example, might relate to hacks that exploit features on an iPod[1], which is a popular gadget. However, there is no good way to look up for such software right now. So if one needs one of these inventions, one would end up writing it

herself, or else be deprived of it even though it exists on campus. There is a huge opportunity to make such inventions visible, in small networks such as university campuses like MIT, as well as networks of small organizations.

In this thesis, we propose a framework called the UniPlug that encourages the ad-hoc sharing of inventions that exploit different features on a device for specific, interesting uses and contexts. UniPlug makes it effortless for an inventor to share such inventions with the rest of the community. It makes it easy even for a computer layman to obtain inventions written for their devices by running software on their computer that goes and looks around at inventions posted by the community. Inventions can be computer programs or software, diagrams, websites, video tutorials or online papers.

UniPlug uses device identification information such as Class ID and Product ID to locate inventions for a device type. For example, when an iPod is plugged in, UniPlug software running on the computer, called the UniClient, would look around at UniServers on the network for iPod related inventions. The UniServer makes the invention available for download in an ad-hoc, local manner.

UniClient works by monitoring the Ethernet and USB ports to detect when a device is plugged in. It monitors the Bluetooth environment for proximity detection of Bluetooth devices. For network devices such as printers, it queries the UniServer for any information or programs it might have using the printers' address as an ID. When a device is plugged in, UniPlug does automatic lookups and downloads software from trusted UniServers. The UniServers are user populated, and inventions are safe tagged and in some cases the process of adding an invention is moderated by the administrator to prevent malicious code from being added.

We have formulated mechanisms such as Credit-based Election of UniServers, Safe Tagging and Trust Networks, Locally Relevant Storage and Context-Oriented Programming to ensure the efficient and successful operation of this system.

1.2 Thesis Structure

Chapter 2 gives background and an overview of work related to the UniPlug. Chapter 3 presents the design of the UniPlug as well as a practical demonstration of the idea. Chapter 4 discusses the applicability of UniPlug to the Medical Device domain. We conclude in chapter 5 with a summary and future work.

Chapter 2

Background

2.1 FluidVoice

The inspiration for UniPlug came from work on the FluidVoice[2] project. FluidVoice is an ad-hoc infrastructure-less voice conferencing system where each node has a local mixer controlled by a graphical user interface. FluidVoice exploits the broadcast nature of wireless, which is otherwise considered a weakness because of the shared spectrum. Nodes receive packets from everyone else and then mix them locally to listen to the desired mix.

FluidVoice is ad-hoc, so users join and leave the network at wish without causing any disruption to the system. A web based graphical user interface allows mixing to be controlled locally by representing each node as a named circle. A node's volume is increased or decreased by dragging its circular icon towards or away from the center of the semicircular screen(see Figure 2-1). The right and left halves of the semicircle represents the right and left channels on the speaker.

Communication networks till now have been designed for point to point communication. Communication nodes are built with an assumption of significant supporting infrastructure and are preprogrammed. With computing and communication capabilities continually

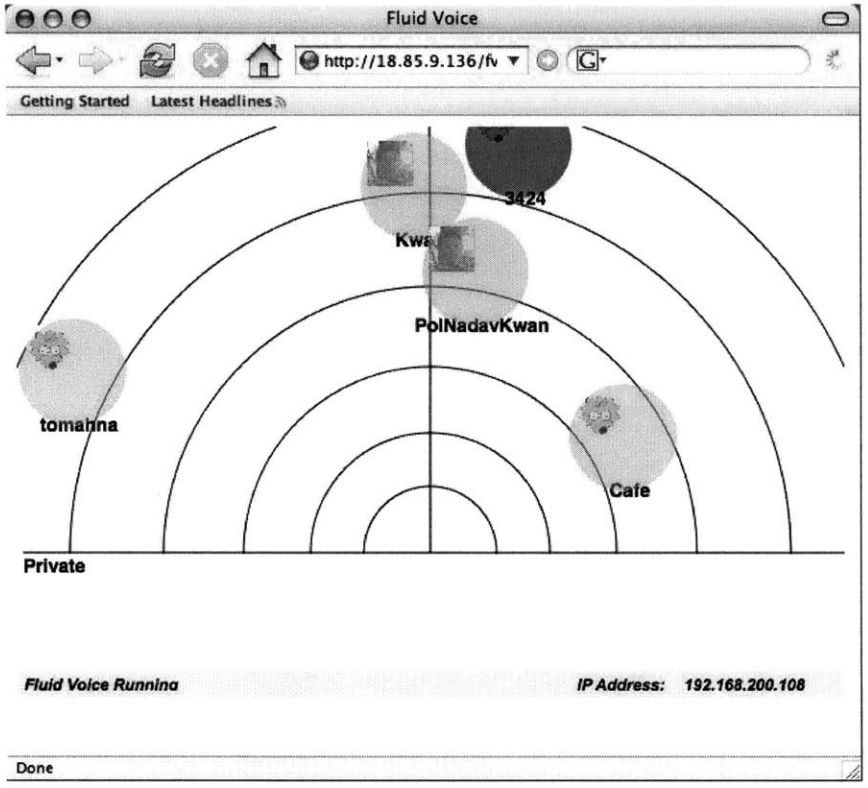

Figure 2-1: Screenshot of FluidVoice Web Based Interface

improving in the forseeable future, end user devices are increasingly becoming capable of being programmed to communicate with multiple nodes without the need for additional infrastructure. Unlike a traditional voice conference in which central conferencing equipment does the mixing and everyone listens to the same mix, FluidVoice allows different nodes to mix audio according to their own wish. The system is symmetric with respect to volume control, so that if user A pushes user B's icon towards the center thereby increasing B's volume, A's icon also gets pushed towards the center on B's GUI.

Apart from providing an interesting application over a new system architecture, FluidVoice also provides an opportunity to stress the wireless environment and reveal weaknesses. Voice is a compelling application for this purpose because it is a demanding application that makes the performance of the system visible through perceived degradations in voice

quality. Traditionally, sensors have typically been used to experiment on ad-hoc wireless networks. The issue with sensors is that they don't send a lot of data per unit of time and hence do not stress the network. On the other hand, video can be buffered and played without any perceivable loss of quality to the user. Real time audio thus provides an appropriate application as well as test environment.

While working on FluidVoice, we designed and implemented a gateway that allows Avaya's SIP-based OneX[3] peer-to-peer IP phones to call into the FluidVoice network. Then we observed that if someone else on the MIT campus were to use FluidVoice and wanted to interface an Avaya OneX phone, they'd end up either not using the phone or have to write up the code themselves. A good solution to this problem would be to design a system to make such ad-hoc inventions easily available to others. Then we could also design a subsystem that detected an Avaya phone being plugged into the FluidVoice network and fetch code from the server that would make the phone interoperate with FluidVoice.

The idea was generalized to detecting and sharing inventions for USB, Bluetooth and Ethernet based devices etc. The aim was to allow distributed invention sharing in a community where everyone benefited from each others inventions.

2.1.1 Living The Future(LTF) Project

One potentially interesting platform to test the UniPlug is the recently proposed Living the Future(LTF) program[4] at MIT. LTF proposes the establishment of a large-scale testbed at MIT to experiment with existing and novel application concepts based on existing, upcoming and not yet conceived networking concepts and technologies.

The scenario for discussion is a university campus, such as the MIT campus. The fundamental idea is to use the MIT community itself for pilot testing of new projects that are developed here. In this thesis, we will focus our design around such a campus, and show later how we can scale to larger administrative and geographical entities. A campus is

typically geographically limited, to a few miles. There could be tens of LANs spread across the campus.

To understand the context, the MIT campus has about 20,000 members[5] which includes students, faculty and staff. The MIT community isnt geographically limited to the MIT Campus alone; a number of its members live in the city of Cambridge outside the MIT campus. Still others live in surrounding towns such as Newton, Salem etc. It is the overarching aim of the Living The Future(LTF) program to include all members of the MIT community in this large scale experimentation. We need to therefore ensure that there are ways in which members in geographically diverse locations are able to participate actively in the program.

Using the MIT community as a testbed will allow quick validation of UniPlug as an easy to use, useful system and present a framework for the MIT community to share its creations. The UniPlug system can then be opened to public use and released as open source software.

2.2 Related Work

The UniPlug combines a number of ideas, some of which are similar to aspects of existing systems, to create a truly unique invention sharing system. There is thus a rich variety of scholarly work that relates to different aspects of the UniPlug. We briefly describe some of the most relevant related work below.

2.2.1 Personal Router

Personal Router [6] is a project at MIT that explores the technical challenges associated with creating open interfaces to wireless services. It enables users to dynamically and automatically choose between wireless services based upon requirements and prices.

The Personal Router seeks to create and enable a market where services are advertised and purchased. The UniPlug, on the other hand, openly shares inventions with everyone. Personal Router deals with purchasable items, while the UniPlug deals with free inventions, that is shared publicly in an open source way. Also, the Personal Router is targeted at wireless access only, while the UniPlug is targeted at devices that connect to a computer via a pluggable port, or through a wireless connection such as Bluetooth, or WiFi. Both have their benefits; the UniPlug might enable a device to offer a service by providing the right programs for it, but it doesnt act as a facilitator for any such service or transaction.

The Uniplug has similarities as well to the Personal Router. Both the UniPlug and Personal router are mobile device based, and involve service advertisement. They are both aimed at increasing the value offered by devices. We therefore view them as complementary to each other.

2.2.2 Peer-to-Peer Networking

A Peer-to-Peer(P2P) computer network is one in which network participants cumulatively provide value, instead of a few centralized servers providing core value to a service or application. Peer-to-Peer networks are typically used for connecting nodes via ad-hoc connections. Such networks are used for many useful purposes such as sharing files containing audio, video, data, and for voice telephony.

In a pure P2P network, each node acts as both a client as well as a server. This differs from the conventional client-server model where the server provides services that the client accesses and communication is typically to and from the server.

Some networks such as Napster, OpenNAP and IRC server channels use a client-server structure for some tasks (e.g. searching) and a peer-to-peer structure for others[7]. Gnutella and Freenet use a peer-to-peer structure for all purposes, and are sometimes referred to as

true peer-to-peer networks. Gnutella is however greatly facilitated by directory servers that inform peers of the network addresses of other peers.

2.2.3 Compact Disk DataBase(CDDB)

The Compact Disk DataBase(CDDB)[8] is in some ways a close analogue of the UniPlug. The CDDB is a database that allows software applications to look up audio CD information over the Internet. This is done by the client which calculates a nearly unique disc ID and then queries the database. Upon receipt of the reply, the client is able to display the artist name, CD title, track list and other information.

2.2.4 Advanced Packaging Tool(APT)

The Advanced Packaging Tool or APT[9] is a package management front-end used on Debian GNU/Linux[10] and its derivatives like Ubuntu[11]. APT simplifies the process of managing software on Unix-like operating systems, automating the retrieval, configuration and installation of software packages, either from binary files or by compiling source code. APT is often hailed as one of Debians best features[12].

In some ways, APT is similar to UniPlug. Both of them fetch programs from a repository, in a way convenient for the user. But while APT is highly structured in terms of how files are added and managed, UniPlug is intended to be fast and ad-hoc, something that would be more useful, for example, for people sharing files quickly within a university. UniPlug is likely to be used to fetch one or more program files, while APT deals with complete packages containing many files. UniPlug is less structured, more ad-hoc and is device targeted. APT is more structured, centrally controlled, and is targeted for computers in general.

The relevant question in considering APT as a candidate for replacement of the UniPlug is: Is APT the best way to share invention locally? We believe that APT, at least in the

current form, is not. And if we started modifying it to suit our needs, it would end up being something totally different from what it is today. Given that APT in its current form serves its intended needs pretty well, we believe that the creation of the UniPlug as a unique entity is well justified.

2.2.5 Service Discovery Protocols

UniPlug's mechanisms for seeking information are similar to the way service discovery protocols advertise and seek services. It is envisaged that service discovery would enable devices to automatically discover network services, including their properties, and services may advertise their existence in a dynamic way[13]. Some well known service discovery protocols include the Service Location Protocol(SLP)[14], Jini[15], Salutation[16], Universal Plug and Play(UPnP)[17], Bluetooth Service Discovery Protocol(SDP)[18], and Konark[19]. We present brief descriptions of these protocols below. Some of the descriptions has been taken from [13], which includes an excellent review section.

- **Service Location protocol(SLP)**: The Service Location Protocol(SLP) is designed for TCP/IP networks and is scalable up to large enterprise networks. It is being developed by the IETF. The SLP architecture consists of User Agents(UA) that perform service discovery on behalf of the client, Service Agents(SA) that advertise the location and characteristics of services and Directory Agents(DA) that collect service address and information from SAs and respond to service requests from UAs. Service discovery can also happen without the involvement of a DA.

- **Jini**: Jini, an extension of Java, addresses the issue of how devices connect with each other in order to form a simple ad-hoc network, and how these devices provide services to other devices in the network. Besides pointers to services, Jini can also store Java-based program code for these services. This means that services may upload device drivers, an interface, and other programs that help the user to access the service.

When a client wants to utilize the service, the object code is downloaded from the Lookup Table to the JVM of the client. Thus the Jini object code offers direct access to the service using an interface known to the client. This code mobility replaces the necessity of pre-installing drivers on the client.

Jini is tightly tied to the programming language Java, which makes it dependent on the programming environment. It also assumes that every device is running the Java Virtual Machine (JVM), which consumes memory and processing power. This can be a difficult requirement, which might especially not be fulfilled in embedded systems. Also Java does not include a mechanism for directly accessing memory or hardware registers. So there will always be a need for device drivers and other pieces of supporting software written in C/C++ or assembly[20]. Jini is also only designed to work with wired networks.

The concept of code mobility in Jini is interesting and somewhat similar to our interest in storing and retrieving programs in UniPlug. Also, Jini uses a concept called leasing, which makes it useful for ad-hoc networks. Leasing means that every time a device joins the network, it registers itself only for a particular period of time, called a lease, after which the lease expires unless it is renewed. Also, in the recent past, there has been work on possibly extending Jini to including a way for integrating legacy devices into the system[21]. Jini thus provides an interesting case study in the design of the UniPlug.

- **Salutation**: The Salutation architecture consists of SaLutation Managers(SLMs) that have the functionality of service brokers. Services register their capability with an SLM, and clients query the SLM when they need a service. After discovering a desired service, clients are able to request the utilization of the service through the SLM.

- **Universal Plug and Play(UPnP)**: Universal Plug and Play(UPnP) extends Microsoft's Plug and Play technology to the case where devices are reachable through a

TCP/IP network. Its usage is proposed for small office or home networks, where it enables peer-to-peer mechanisms for auto-configuration of devices, service discovery, and control of services. The Simple Service Discovery Protocol is used within UPnP to discover services. SSDP uses HTTP over UDP and is thus designed for usage in IP networks. UPnP works on the assumption that all the devices are based on a common protocol that is agreed to by vendors. So the vendors build compiant devices that adhere to the UPnP device architecture as defined by the UPnP forum working committee[17]. UniPlug also works by a *Plug and Play* mechanism to detect devices being plugged in and fetch inventions for them.

- **Bluetooth Service Discovery Protocol**: The Bluetooth Service Discovery Protocol(SDP) is used to locate services provided by or available via a Bluetooth device. Bluetooth addresses service discovery specifically for ad-hoc environments where it supports the following inquiries: search for services by service type; search for services by service attributes; and service browsing without a priori knowledge of the service characteristics. SDP doesnt include functionality for accessing services. However, the Bluetooth SDP, as the name implies, is only designed to work with Bluetooth based devices; which makes it inadequate for UniPlug needs.

- **Konark**: Konark is a service discovery and delivery protocol designed specifically for ad-hoc, peer-to-peer networks, and targeted towards device independent services in particular. It has two major aspects - service discovery and service delivery. For discovery, Konark uses a completely distributed, peer-to-peer mechanism that provides each device the ability to advertise and discover services in the network. The approach towards service description is XML based. It includes a description template that allows services to be described in a human and software understandable forms. A micro-HTTP server present on each device handles service delivery, which is based on SOAP. Konark provides a framework for connecting isolated services offered by proximal pervasive devices over a wireless medium.

Thus we can see that all of the protocols described above have strong features for UniPlug to draw from. Yet none of them, by itself, is flexible enough to address all our needs. It is also important to note that discovering a UniPlug repository is not the only aim of UniPlug. The aim of UniPlug is to provide a framework for developers to share programs using a seamless access mechanism that allows devices to obtain inventions that are most suitable for their form and context.

2.2.6 Device Detection Tools

There are a number of popular software tools that are used for detecting the presence or absence of devices for a particular port. Ifplugd[22] is a tool that is used to detect if something is connected to the Ethernet port, and it can be configured to run scripts when the port goes up or down. Ethereal[23] is a widely used program for sniffing packets and analyzing their details and statistics from an Ethernet port or wireless cards using 802.11 technology. Ifplugd and Ethereal used together can be used to monitor if the Ethernet port is up or down, and when it is up, to capture the packets and look for identifying information. In our program we use ngrep[24] to capture packets by calling it from within a shell script when ifplugd detects that the Ethernet port is turned on.

lsusb[25] can be used to scan for USB device and extract information about the device including its serial number, class ID, manufacturer and product ID, etc. This information reveals what kind of device it is, and the serial number in combination with the other information can be used to identify who the device belongs to, and subsequently to lookup for scripts instructing what the device is supposed to do, or what is supposed to be done with the device.

Bluetooth devices can be scanned and then probed using the BlueZ[26] utils tools. Various types of information about the device such as Bluetooth device address, Class ID, services offered, etc can be gleaned from using bluez.

Chapter 3

UniPlug Design

This chapter focusses on the design of the UniPlug. We begin by formulating the problem that the UniPlug intends to address and sketch out the key requirements that UniPlug must satisfy to act as a viable solution to this problem. We then describe the different possible models for the UniPlug architecture and compare their strengths and weaknesses. This is followed by a description of the various components of UniPlug and their functionality. We conclude the chapter with a discussion of security, context, fairness and scalability and how the system is designed to handle them in an effective way.

3.1 Problem Formulation

The problem can be stated as follows: to design a framework that enables invention sharing across MIT. Inventions could be applications that exploit features on the device to enable higher utilization. Or they could be programs that enhance a device's value by offering new affordances or services. The full range of inventions could include device/prototype, diagram, description, program/software, eprototype, website, etc. Inventions could be in

paper/device/electronics. However they must be represented in some electronic form to be sharable.

If someone builds a piece of software and is willing to share it openly, users who are not technically savvy should also be able to obtain and use it, easily. The software must come with a description that is easy to understand. It has to be short, succinct, and comprehensible to an average computer user, who we also call a computer layman. It must also specifically warn the user about the changes that the software will make to the system. So the challenge we are addressing is to take advantage of the presence of connectivity to make things easier for the user.

3.1.1 Requirements

A UniPlug system must satisfy the following requirements:

- **Ad-Hoc**: The system must be able to work well in an ad-hoc environment.
- **Distributed**: The system must allow for a distributed architecture. It must also be populatable in a distributed way, such as through a centralized webserver or by having UniServers in p2p/hybrid mode.
- **Cooperative**: There must exist a mechanism to query other servers about the availability of a program.
- **Safe**: The code should either be trusted or be verifiable. There must exist a mechanism to prevent or discourage malicious code submission in the first place, and control dissemination once its presence is detected.
- **Scalable**: The system should be able to start with minimum infrastructure, yet it should be able to scale.

3.2 UniPlug Components

The UniPlug architecture consists of the following components. These components are responsible for fetching, storing and providing inventions.

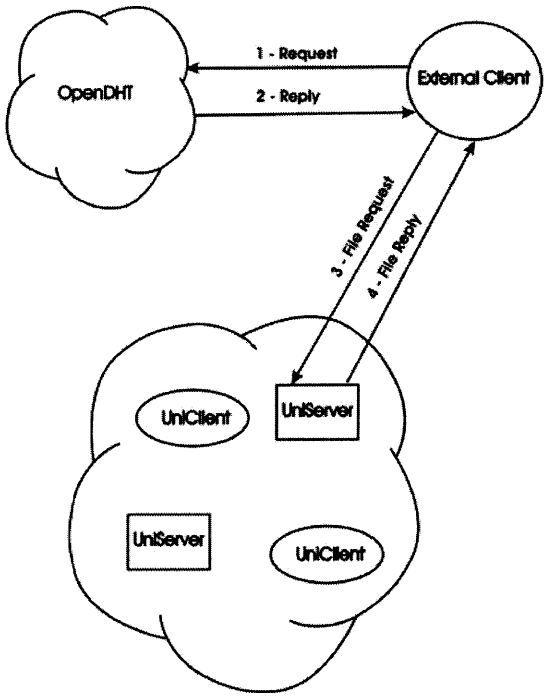

Figure 3-1: UniPlug Architecture

3.2.1 UniClient

The UniClient runs in the background on client machines. It monitors ports selected by the user such as Ethernet, Blutooth and USB etc. Upon detection of a device being plugged into the port(Ethernet, USB) or coming into proximity(Bluetooth), the UniClient performs various tasks that have been defined. Such tasks might be periodic, or event-driven. For example, the UniClient may contact the server on detecting a new device to see if the server has programs for the device. Or it might start another program on the machine that already exists locally.

The UniClient maintains a list of UniServers or obtains the list from OpenDHT[27](which is discussed next). It is a key UniPlug element and merges with UniServer in the P2P model. It maintains a list of IDs of devices that are owned or registered with that machine. It has the following port scanners:

- lsusb[25] for USB devices.
- bluez[26] for bluetooth devices.
- ifplugd[22] for Ethernet devices.

3.2.2 UniServer

The UniServer acts as a repository for UniClients. It provides a storage and directory service for inventions. Inventions stored on UniServers are accessible by UniClients both inside and outside the local network.

The UniServer runs as a daemon. It listens for requests from clients, and responds either with information or sends a *not-available* reply. The UniServer is also responsible for adding and deleting new entries. Depending on the mode, the UniServer might or might not contact other UniServers. In P2P mode it queries peers for programs that are relevant for devices owned by the user. In hybrid mode, it looks around for programs for the devices registered with it. The UniServer does periodic checks for programs for the devices it has registered. It also provides a registration mechanism for subscribing to alerts from other UniServers.

The functions of the UniServer can be listed as follows:

- Handle UniClient requests by listening for requests and replying to UniClient with content or error code or delay code if looking at other UniServers. Maintain a queue of requests.

- Query servers periodically and update information about relevant inventions.
- Allow Invention submission through a web server and administrator controlled moderation.

3.2.3 External Reference(OpenDHT)

OpenDHT[27] is a publicly accessible DHT service. Clients of OpenDHT can issue *get* and *put* instructions to any DHT node, which processes their request. They do not need to run a DHT node in order to be able to use the service. Thus OpenDHT is used as a highly-available naming and storage service for sophisticated applications. The OpenDHT server acts as an external reference for UniClients who may be outside the MIT network. It provides the IP address of UniServers in response to a request from an external UniClient.

3.3 UniPlug Models

There are 3 different models on which a UniPlug architecture might be based. They range from a fully infrastructure based (client-server) model, to a pure peer-to-peer model. In between lies the hybrid model, which is part peer-to-peer and part client-server.

The selection of a model for a UniPlug implementation depends on the context for which it is being built, taking into account the resources available and the best choice of advantages and disadvantages associated with each model. In the following sections, we describe each of these models and discuss their relative benefits and drawbacks.

3.3.1 Peer-to-Peer

The Peer-to-Peer model has the following properties:

- **Architecture**: In a pure P2P model, each node is both a client and a server. Peers can be discovered by a discovery mechanism, or by querying a directory server or supernode that keeps track of clients, like how Skype[28] works.

- **Content**: In P2P mode, each node only hosts content for devices that it owns. So different nodes might replicate content for a device if the owner of the node owns the relevant device for that content.

- **Suitability**: This is suitable for systems where users interact directly with each other as peers and act as servers for each other. There is no heirarchy.

3.3.2 Hybrid

The hybrid model has the following properties:

- **Architecture**: In this architecture, certain UniServers have the role of directory servers. They host information about the content hosted by each node in the network. They are elected dynamically and a particular UniServer might not always be up. A community chooses a node to act as a UniServer. This can be done in various ways such as random selection, round robin, credit based selection, or by simply asking for volunteers and choosing one between them. It is reasonable to expect that everyone who gets access to the service also contribute towards it by acting as a UniServer every once in a while. The non-UniServer nodes are then considered pure UniClients. The UniServers will also have their own UniClients.

- **Content**: In this model a community served by a UniServer has its own content, and only seeks when it doesn't find the content locally. Its like an interlibrary loan where a book is requested when it is not found locally, but finding it locally is the default. A UniServer uses its offline time to consult other UniServers and obtain their list of programs for devices registered by it so that the information is cached locally.

- **Suitability**: Suitable for an organization with many branches with each branch talking to each other when needed.

3.3.3 Client-Server

The Client-Server model has the following properties:

- **Architecture**: The server is always up and its address is well known, hence the availability is high. The architecture is centralized and therefore less resilient to failures. It requires regular maintenance and supervision.
- **Content**: The server has its own content which is limited to what the local community produces. A server may choose to check with outside servers about availability of files for a particular device type, as a special case.
- **Suitability**: Suitable for an organization or local community for inventions only expected to be produced or used locally.

3.3.4 An Appropriate Model For A Campus Network

For a campus network such as MIT, the hybrid model seems to be the most suitable. This is because the model allows for an architecture that is not centralized, yet gives the impression of a supporting architecture that provides services. It makes the ad-hoc community served by a UniServer look like its being served by a dedicated server, though actually the designated server keeps changing. The model is inherently more local and cooperative. In the following section, we will look at mechanisms to ensure fairness in supporting load, i.e. acting as the directory server. Given the dynamic nature of this model, there is a need to be able to designate UniServers dynamically, in a cooperative way. There must also exist mechanisms that allow the system to scale. We discuss each of these issues and our solution to them.

3.4 Mechanisms

3.4.1 Credit Based Directory Election For Fair Load Distribution

At any time, the UniPlug servers must coordinate to ensure there is at least one designated directory server for each LAN on campus, so that if there is a request being broadcast for information about looking for UniPlug server, then the designated directory server can provide that information. The directory servers are like floating points that are visible whenever the hosting computer is turned on, but might go invisible anytime the user decides to turn the computer off. Ideally, if the designated UniServer is about to go off, it informs the other computers on the LAN about its impending disappearance so that a different directory server can be selected and its details communicated to other servers on time. But that might not always happen, so that if a UniServer broadcast doesn't happen in a timeout period, a new UniServer must be elected and updated with the directory information held by the node who was the second to last UniServer. This information may be updated when the last UniServer comes online again. Another strategy is to provide redundancy so that at any times, at least one node other than the UniServer has the latest directory information.

The responsibility for acting as a directory server must be shared to ensure fairness. There are different ways of choosing UniServers to be a directory. These include random selection, round robin, credit based system, volunteering etc.

The issue with round robin selection is that a node might not be up when its turn comes. On the other hand, a random selection method will probabilistically ensure over a long period of time that the average number of times each node is selected is fairly equal. However, given that nodes availability is not regular, this approach might fail too. The volunteer method simply fails when some of the nodes never volunteer.

Our solution, which is to use a credit based system, overcomes these issues. In a credit based system, each node would get credit for the time it acted as a UniServer. The credit

values are stored with the current UniServer. This information is available for anyone to verify, along with a record of changes. When a UniServer selection is to be made, the lowest credit node will be attempted to be chosen first. If it is not up, the 2nd to lowest node is attempted and so on. A slight variation to this strategy is to group the nodes into low credit and high credit groups and randomly try to select one of the low credit nodes.

A node that is inside a local area network that has a directory server can simply ask it about an invention. However, a node that is not in the local area network needs to have a way to find where it can locate a UniServer. For this, we use OpenDHT[27], a publicly available DHT running on the PlanetLab[29] testbed for UniServers to advertise their addresses on. When an external node needs to find information, it does a query on the OpenDHT, which returns a value giving the IP address of a UniServer that holds information about the node holding the relevant file with configuration or program information. The external client then requests the machine holding the file for a copy and obtains it.

The following example shows how the UniServers' IP address could be added and retrieved from OpenDHT.

```
durga@fLuidMachine$ ./put.py UniPlugIP wolverine.media.mit.edu
Success
durga@fLuidMachine$ ./get.py UniPlugIP
wolverine.media.mit.edu
durga@fLuidMachine$ ./put.py UniPlugIP 18.85.9.170
Success
durga@fLuidMachine$ ./get.py UniPlugIP
wolverine.media.mit.edu
18.85.9.170
```

However, inside the local network, the client just broadcasts a request for the file information. If what the machine offers is what the client is seeking, it confirms its request, and

the machine sends back a reply with the file, otherwise it returns an error code.

3.4.2 Trust Networks and Safe Tagging For Local Security

A common concern with sharing programs and using a repository to distribute them is that it might be used maliciously. Its sometimes pointed out that the UniPlug could be used as a universal virus propagation machine. For example, someone could place a bluejacking program on a UniPlug server and thus be able to potentially use Bluetooth connection to hijack devices and then load them with programs to carry out attacks or steal information.

An interesting example in answering this concern is the open source community that has various contributors producing numerous programs. For the most part, it has been successful in not being misused to stage attacks through malicious code releases.

The possibility of malicious programs being released through UniPlug is however real, and we consider a combination of two solutions.

Trust Networks: Within a small community network, if it can be verified that a particular person submitted a piece of program, then if the code is found malicious, the person will be identified and will face social disapproval and might also be given punitive action. Such consequences will prevent most people from submitting malicious code because their reputation will be at stake and once they are identified, their inventions might be rejected by the community.

Safe Tagging: We propose the idea of tagging inventions so that people who have used an invention and can verify that its genuine or useful can vote for it, saying so. For example, a tag could indicate that 15 people have used a program so far and at least 7 of them have verified the code as being non-malicious. Safe Tagging has three benefits: Firstly, it creates a genuine/safe reputation for the program or invention. Secondly, the reputation contributes directly to the programmer's reputation who submitted the code. And thirdly,

it contributes to the reputation of the server that is the primary host for the program. The programmers reputation could be given a score from 0 to 100 to indicate the confidence level about their truthworthiness based on the reputation of all the programs they have submitted so far. Thus genuine tagging allows us to create reputations for programs, programmers and the nodes hosting them. In an active community, this creates a way for the community to collaboratively screen the safe and genuine from the unsafe and not genuine.

Issues

- **Attacks** : One way this system might fail is for a group of malicious users to join a network and then release malicious programs on their machines. The group could then go ahead and tag each others' inventions as genuine, perhaps many times over by assuming fake identities. For instance, a person could create 10 fake identities and use 9 of them for tagging malicious code as safe while maintaining one of them as a genuine persona to avoid being detected.

  Solutions to this problem might be proactive or reactive. A proactive approach might be to use a rating system that is slightly more complicated than what we have looked at earlier. For instance, a person's reputation could be allowed to mature so that they could have a high reputation only if they have been submitting genuine code for a reasonable period of time, say a year. Another solution is a strong authentication method that makes it difficult to impersonate someone else, and real-world credential checking to ensure that fake identities are not created.

  A reactive approach might be that if a program is tagged safe by 10 people and the 11th user recognizes it as being malicious, each of the 10 people must be penalized for tagging an unsafe program as being safe. Additionally, the programmer who created it must be heavily penalized. The issue though is, in a p2p or interlibrary system, if the program was borrowed by a number of people and then discovered to be malicious by someone, how do we inform the previous users that the program is unsafe. One

solution is to generate email alerts saying this program is unsafe and must be removed. Another might be for the UniCient to do a proactive check with the UniServer for malicious alerts associated with its devices every time it is turned on.

- **Reputation Consistency**: Another issue is that of carrying reputations around. If user A has a program which has acquired a reputation of 7.8, and user B fetches the program from A and starts hosting it himself, should be be able to say that the program has a 7.8 reputation? Would it contribute to his own reputation of being a safe server even though he did not really host it while the reputation was being built? Is it fair that someone hosts a program while its reputation is being built and then someone else is able to come and use the program and host it with the reputation it has acquired? Closely related is another issue. Without any coordination, how is the reputation acquired by a program through various different hosts going to be combined? An extreme example is this: if a program was being hosted by 100 different people, and each instance got a safe tag from 1 person each they are still each hosting a program with a pretty low reputation unless they combine all the reputation values in which case they'd have a reasonable reputable program depending on the actual number of average tags thats common in that system.

  Our solution to this issue involves uniquely identifying each safe tag and disseminating it so that new safe tags can be identified as such and be added locally so that everyone is able to calculate the reputation locally. The system might take some time to converge, but it works without any centralized architecture or the involvement of a third party.

Finally, If the source is trusted and there is proper authorization from the user, the system may install and configure high safe-tagged software transparently to the user. This might be necessary to do, for example, if a grandma with absolutely no technical knowledge is using the computer. However, in this case a more technically competent person(like her son) must be able to look at what changes occurred and which software got installed at

what time, and be able to potentially undo the installation and associated changes later.

3.4.3 Context-Oriented Programming For Device Efficiency

If measurements or events are interpreted differently based on an awareness of something, that something is context. For example, we respond differently to request for help from a friend doing her homework asking for water, compared to the same person lying in a hospital asking for water.

This discussion brings up two different meanings of context. The first meaning is the purpose or intention for which a program is written. For example, if an iPod was converted into a storage device, the context would be 'iPod as storage'. The second meaning refers to the variables whose value the program itself senses or measures. For example, the memory available on the iPod for the above mentioned program would be a context in that sense.

So context is the lens through which we look at a measurement or event. The context specifies a mode, and our reactions and interpretations are based on that mode. A mode is a set of conditions or constraints which are used to specify a context. It should be possible to customize our view based on the mode. And it should also be possible to create new modes.

Our intention is that UniPlug should be used to encourage writing programs that keep context in mind while writing programs and the context should be easy to specify, represent and interpret.

In the following sections, we formalize the notion of context and present a classification of various kinds of contexts.

Context Classification

In this section, we present an approach to classifying different kinds of contexts one might encounter when thinking about devices, and a way of representing them.

We believe that contexts can be broadly classified as belonging to one of the following three categories:

- **Device Based**

  Monitoring battery level or available memory in a phone is an example of a device based context. Another example is how much of insulin you have left in your injector. In FluidVoice for example, when the battery level of a cellphone node goes low, the context is conveyed, perhaps seeking nearby nodes that can act as relays for signals for this device, thus reducing the transmission power required and consequently reducing the rate at which battery is being consumed.

  The device based context might also be expressed in terms of a resource management question: how much of something my device requires does it have? How fast is it getting depleted/replenished and how long will it be before it gets used up at the current rate?

- **User Based**

  User based contexts are those that convey information about the user, such as blood pressure, oxygen level in blood, mood(good, sad, happy), conversation analysis(heated, pleasant, romantic), temperature, heart rate, status(busy, meeting, in a conference), special cases, such as conveying an emergency such as getting sick, in a fire, or medical emergency, in a flood, etc. A person's blood pressure might be interpreted differently based on their health status.

- **Surroundings Based**

Surroundings based context can be classified into proximity based context, and location base context. Proximity based context differs from location based context in that it does depend on the neighborhood of the device, but its not tied to a particular location. The affordances at a place might not necessarily be mapped to location. Such a mapping might not even be desirable.

Examples of proximity based contexts are temperature, humidity, brightness, the presence of other nodes, ESSID's available, number of willing relay devices, number of bluetooth devices, etc.

On the other hand, IP phones in separate departments of an enterprise might be governed by separate rules. For example, phones used by sales department could be allowed to make international calls, while phones belonging to the engineering department might only be allowed to make local and national calls. Here, the department is the relevant context. Different departments might not be geographically separate, but they are administratively separate entities. Or a device might grant access only in a corporate LAN environment. Our definition of location is broad enough to cover both geographic locations as well as administrative locations.

The importance of location is that certain services or functions may only be allowed in certain locations. Or certain services or functions may only be available in certain locations. So location-based context can be used both to find out if its permissible to do something because we are at a certain location, or to query if a certain service is available at a particular location. Examples of location based context include mall, house, work, gym, department, sensitive areas, etc.

There is an interesting example that involves an interaction of various context types. If a user make a lot of calls while she is in San Jose(surroundings), it can be interpreted in various ways. If the user lives in San Jose, she might be making a lot of calls from home(context: surroundings). Her making a lot of calls might reflect she is lonely(context: user). Or it might simply be because she has a lot of battery power(context: device) and

thats the primary reason that she calls a lot.

Context Representation

A context representation mechanism should make it easy to define new modes or contexts. Thus context representation reduces to providing the ability to a user to name a context and specify the constraints associated with it. Then the device can either be switched to that context or mode, or the relevant state of the device could be compared with the constraints specified by various contexts to see if any of them is matching candidate.

It is likely that part of the constraints are matched, maybe by more than one context, and then the program has to be designed to decide which of the contexts it should choose. Or it might follow a strict policy so that a context is selected only if all the constraints specified for it are met.

Context representation will the allow a programmer to leverage the power of different contexts to create programs that are aware and that perform optimally under a variety of contexts.

3.4.4 Locally Relevant Storage For Scalability

In UniPlug we only store locally relevant material, so that the system can scale. The system only stores programs for devices that are registered by the UniServer for the community it serves. By doing this, we ensure we have programs available for registered devices. At the same time, since the number of devices registered with a UniServer is expected to be limited, the number of programs stored for these devices would also be limited and it doesn't place a burden on the node. Thus storing locally needed programs on nodes is both useful and justified. When there is a request from an external node for a program designed for a locally registered device, the local server may serve a particular number of requests based on its load, and receive credit for it.

3.5 Practical Illustration

In this section, we describe the practical demonstration we built of UniPlug for Avaya one-X Quick Edition peer-to-peer phones[3]. This demonstration is a proof of concept and shows that the ideas outlined in this chapter are feasible.

The Avaya one-X Quick Edition is a smart communications system targeted at small businesses and small branches of enterprises. The one-X phones use SIP-based peer to peer technology. Once the phones are plugged into the local area network, the system configures itself using a proprietary discovery mechanism in minutes, and all the features such as voicemail, conferencing and call management become instantly available.

We have built a gateway to enable users to participate in a FluidVoice[2] conference using these Avaya one-X phones. Therefore, we decided to demonstrate the idea of UniPlug by openly sharing the Avaya gateway(invention) and fetching and running it automatically when an Avaya phone is discovered in the network.

3.5.1 Setup

The setup for this demonstration consists of an Avaya one-X QE peer-to-peer phone and three machines running Linux. The Avaya one-X phone is connected to a primary machine. The primary machine runs a version of UniClient that monitors the Ethernet port using the open source tool ifplugd[22]. It also has the software PABX Asterisk[30] installed on it.

The secondary machine has FluidVoice installed on it. It runs the gateway code that takes voice packets coming from the Avaya phone and converts them to FluidVoice format and broadcasts it. It also runs a mixer that mixes packets from all the other nodes in the FluidVoice conference and sends them back to the Avaya phone after encoding them. The secondary machine runs a version of UniClient that can connect to the UniServer and send requests for program files.

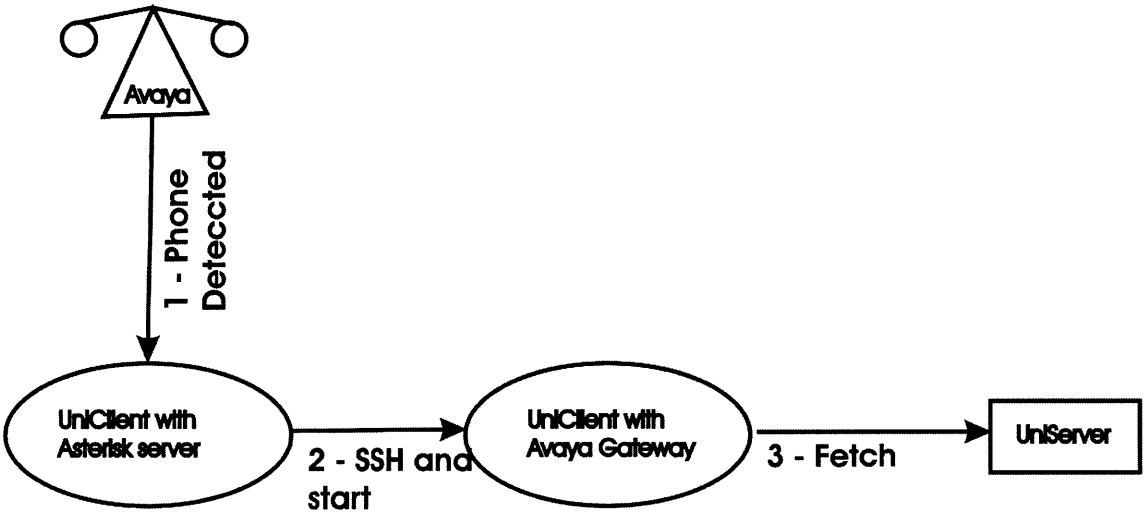

Figure 3-2: UniAvaya Module

Programs are stored on the server. UniServer runs as a daemon on the server. UniServer accepts requests from UniClient on secondary machine, queries a local MySQL[31] database to check if it has the gateway program, and replies with the program file, or sends an error code.

3.5.2 Working

To a user, the underlying system in this demonstration is totally transparent. The user plugs in an Avaya one-X QE phone into the primary computer. Within a few seconds the system sets itself up, so that the user can join the FluidVoice conference by dialing '230'. Behind the scene, a series of steps are involved in setting up the system, and obtaining and running programs if they aren't available.

The UniPlug demonstration consists of the following steps:

- **Step 1**: Initially, the Avaya phone is not plugged in. UniClient is running on the primary machine monitoring the Ethernet port using the open source tool ifplugd.

- **Step 2**: The Avaya phone is plugged into the primary machine. As soon as an Ethernet device is detected by ifplugd, it triggers a script in the UniPlug client module that starts monitoring if Avaya one-X phones are present in the local area network by capturing packets from the Ethernet port and analyzing it. We had discovered earlier from packet capture and analysis that Avaya one-X phones send out a packet to a multicast address every second. So by capturing packets on the Ethernet port and checking if the packets destination matches the multicast address that are used by one-X phones, we can tell if an Avaya one-X phone is present.

- **Step 3**: If a one-X phone is detected, the client module does the following:

  - **Step 3.1**: Starts the software PABX Asterisk on the local machine. Asterisk then registers itself as a SIP client with the Avaya one-X phone.

  - **Step 3.2**: ssh into secondary machine and runs the UniClient there. The UniClient checks if the one-X FluidVoice gateway is present on the machine. If it isnt, the UniClient connects to the UniPlug server and obtains the gateway code. The code retrieval is carried out by the UniClient establishing a TCP connection with the UniServer. Once the connection is established, the UniClient sends a request that contains the device identifier. On receiving this request, the UniServer connects to a local MySQL server[31] and runs a query to check if the device is listed. If it is, the UniServer sends the gateway program file back to the UniClient on the same connection. Once this is done, the UniClient terminates the connection with the UniServer.

    It then starts the gateway program, which registers with the Asterisk on the primary machine as a SIP client, and also waits, listening for calls. One-X is the SIP server for Asterisk, while Asterisk is the SIP server for Secondary machine. Thus the calls from One-X go through Asterisk and are eventually handled by the secondary machine.

- **Step 4**: At this stage, the system is ready to receive calls from one-X phones and

connect them to the FluidVoice conferencing system. On their part, the Avaya phone user has to simply dial 230, which we have chosen to be the standard extension to reach the FluidVoice network, and see if they get connected to FuidVoice. When a one-X user dials 230, the call gets forwarded to the local machine running Asterisk, since asterisk has registered itself as a SIP client with extension 230 on the one-X phones. When Asterisk receives the call, it forwards the call to the secondary machine which has registered itself as a SIP client with the Asterisk server for extension 230. The secondary machine answers the call and the caller gets connected to the FluidVoice conference.

We have thus taken advantage of a simple idea, that one-X phones send out periodic packets to a multicast address, to build a system that immediately detects the presence of one-X phones in a local area network and gets ready within seconds to receive calls from the one-X phones.

The objective of the abovementioned illustration is to illustrate the seamless sharing of innovation. The gateway system that allows one-X phones to talk to FluidVoice is shared in a way such that UniPlug clients are able to retrieve it whenever a one-X phone is plugged in. We can generalize this notion to say that whenever the UniPlug client detects a new device on the network, it could ask the UniPlug server if it has any inventions for the device.

The UniClient and UniServer are built using twisted[32], which is an event driven networking engine written in Python.

Chapter 4

UniPlug For Diffusion Of Medical Innovations

4.1 Motivation

In 1999, the Institute of Medicine reported[33] that errors in the delivery of medical care were leading to 98,000 deaths in the US annually, and recommended applying information technology to deliver safer care. The lack of appropriate context-based feedback that takes safety issues into account is a key factor in the occurrence of preventable medical accidents.

There is a compelling need to interconnect medical devices and equipment, and introduce automated safety checks. More specifically, the lack of control and monitoring in an operating room has been known to cause accidents. Consider the following case studies:

- A technician inadvertently turns a defibrillator off that was being used to externally pace a patients heart. The nurse had closed the privacy curtains around his bed because the patient needed to use the urinal while the defibrillator was sitting outside the curtain. The technician going on his rounds saw that the defibrillator was left

turned on after what he assumed was a manual testing so he turned it off. This resulted in the patient going mild bradycardia[34].

- An emergency medical services paramedic attempted to use a defibrillator on a 67-year-old man with ventricular tachycardia. Yet nothing happened. The defibrillator displayed an indication that it was in synchronized mode but provided no feedback to tell the user that it was not prepared to shock because of low QRS amplitude[34].

- In a fatal incident in 1984, a patient received 16,000 rads instead of the intended 180 rads when undergoing radiation treatment. This incident involved a combination of technical failures(software and possibly hardware), combined with human behavior resulting in catastrophic radiation overdoses[34].

- During a laparoscopic procedure, the surgeon and the anesthesiologist must carefully orchestrate monitoring with insufflation of the abdomen. It is a teamwork issue that requires clear communication with complex activities that are interdependentand if there's a lapse anywhere along the way, it could cause a very serious patient problem[35].

- A patient undergoing gallbladder surgery is under general anesthesia during the procedure. To avoid image blurring while taking X-ray images during the surgery, it is necessary to switch off the ventilator that is breathing for the anesthetized patient. Turning the ventilator off, taking the X-ray, and turning the ventilator back on again are all manual processes. If the team of caregivers is distracted, it is possible that the ventilator might not be turned back on. Although very unlikely, this tragedy occurred recently[36][37].

- A laser used for airway surgery ignites the oxygen-rich gas providing ventilation, burning the patient severely[38].

This raises some interesting questions. Could an anesthesiologist in an Operating Room(for example, at Massachusetts General Hospital) take advantage of network connectivity to

specify context aware high-level instructions and goals to make various operating room equipment serve her better? Better yet, could the networked or plugged-in appliance itself make an effort to find out what she wants it to do at a particular time and go on to execute those tasks, while learning and keeping track of what it learns in the process? Could a new device unknown to the system so far, be added seamlessly to such a system without introducing any safety issues? Could we ensure that the whole communication process is reliable to ensure that their failure doesnt cause accidents to occur? Could we make the entire process of adding, configuring and managing such devices extremely easy and intuitive for the technician or nurse?

There are a number of doctors and biomedical engineers who develop new software or build new systems by interconnecting software that have some of the features mentioned above. Such inventions are usually lost over time, after being used for a while perhaps, by the doctor. We propose that the UniPlug system be used both for sharing such inventions between doctors/engineers at a hospital or a group of hospitals. At the same time, context-oriented programming can be used to report the context being measured by a medical device and seek configuration information by the hospital server. By doing this, medical devices' behavior could be preprogrammed as well as monitored while it is running from a UniServer, resulting in smart and efficient utilization of medical devices.

4.2 Leveraging Networked Medical Devices

Networking medical devices is one approach to improving the reliability and safety provided by such systems. A medical system becomes inherently more reliable if networking enables it to acquire data from different devices and correlate them. For instance, if a patient has normal heart rate and temperature reading but a very low oxygen level based on pulse oximeter reading, the system could deduce that the pulse oximeter has probably slipped off the finger, and raise an alarm for a nurse to check the equipment. It also reduces the human

error factor that is involved in nurses manually taking readings and noting them down or feeding them into a computer. It could enable readiness assessment checks[23] that increase the efficiency of the operating room by reducing setup and checking time.

A networked operating room would have to provide high guarantees of reliability and safety. Messages sent from one machine to another much reach well in time with high probability, or else the destination machine must detect the delay and raise an alarm if appropriate. A major hurdle to such networking is that currently, it is not possible to buy the best of breed operating room equipment because equipment from different vendors isnt interoperable. Even custom vendor specific equipment systems do not offer adequate networking between the different operating room devices. To address this need, we propose building a class library as part of the UniPlug framework that allows a user to write a connecting block using a high level language such as Python to connect legacy devices into the system. The UniPlug block would provide basic classes and would allow extensions to be written by end users to provide advanced or custom functionality. The extensions would then be contributed to, and shared from, the open UniPlug repository.

The Medical Information Bus was the first well-known effort[39] to develop medical device-specific communication standard and supporting hardware. It was focused on intravenous infusion devices and RS-232 hardware. It wasnt adopted by the medical device manufacturers due to low clinical demand, complexity, and proprietary hardware requirements. The MIB concepts were later used to build a family of standards called IEEE 1073[40]. IEEE 1073 initiated efforts to standardize lower layers while adding work on upper layers in a 7-layer ISO communication model. IEEE 1073 has been standardized by the ISO as standard 11073[41]. However the adoption and promotion of 11073 by medical device manufacturers has been slow. Meanwhile HL7(Health Layer 7) a standards body[42] has formed standards for communicating patient data between clinical information systems at the application level. However HL7 lacks device connectivity, it as never intended for Point-of-Care devices and monitors. In early 2004, the Medical Device Plug-and-Play program[43]: a

multi-disciplinary, multi-institutional program was formed to support the development and adoption of clinically grounded solutions for medical device interoperability. The UniPlug work is being carried out in cooperation with the MD PnP program.

The Medical Devices PnP program is a first step in this area to interconnect various devices into a system in a standard way, thereby designing a basic open framework that can be used to connect a variety of devices. This would be an intermediate step that would facilitate the leap from the current scenario where there are no standards for medical device interconnection to a future scenario where there would be a universal standard for the same. In the medical device community, it is expected to eventually lead to a set of open standards for communication and control between devices as well as between a computer and various devices.

The UniPlug would enable sharing inventions for medical devices over a network, to enable the efficient utilization of the variety of medical devices attached to patients. Once inventions are tagged safe by an authority, they can diffuse seamlessly from one UniServer to another across a network of clinics and hospitals. The next section describes such a scenario.

4.3 A Patient-Centric UniPlug Implementation Scenario

Figure 4-1 describes a Patient-Centric UniPlug implementation scenario in a hospital. The UniPlug system has UniClients running on the operating room server. UniClients are responsible for monitoring inventions for the various devices attached to a patient such as pulse oximeter, dialysis machine, defibrillator, etc.

The operating room server is connected to various patient devices, as well as the Hospital A UniServer, which has peer-to-peer connections with hospital B and C UniServers. The hospital A WebServer interfaces to the hospital A UniServer. The WebServer provides a

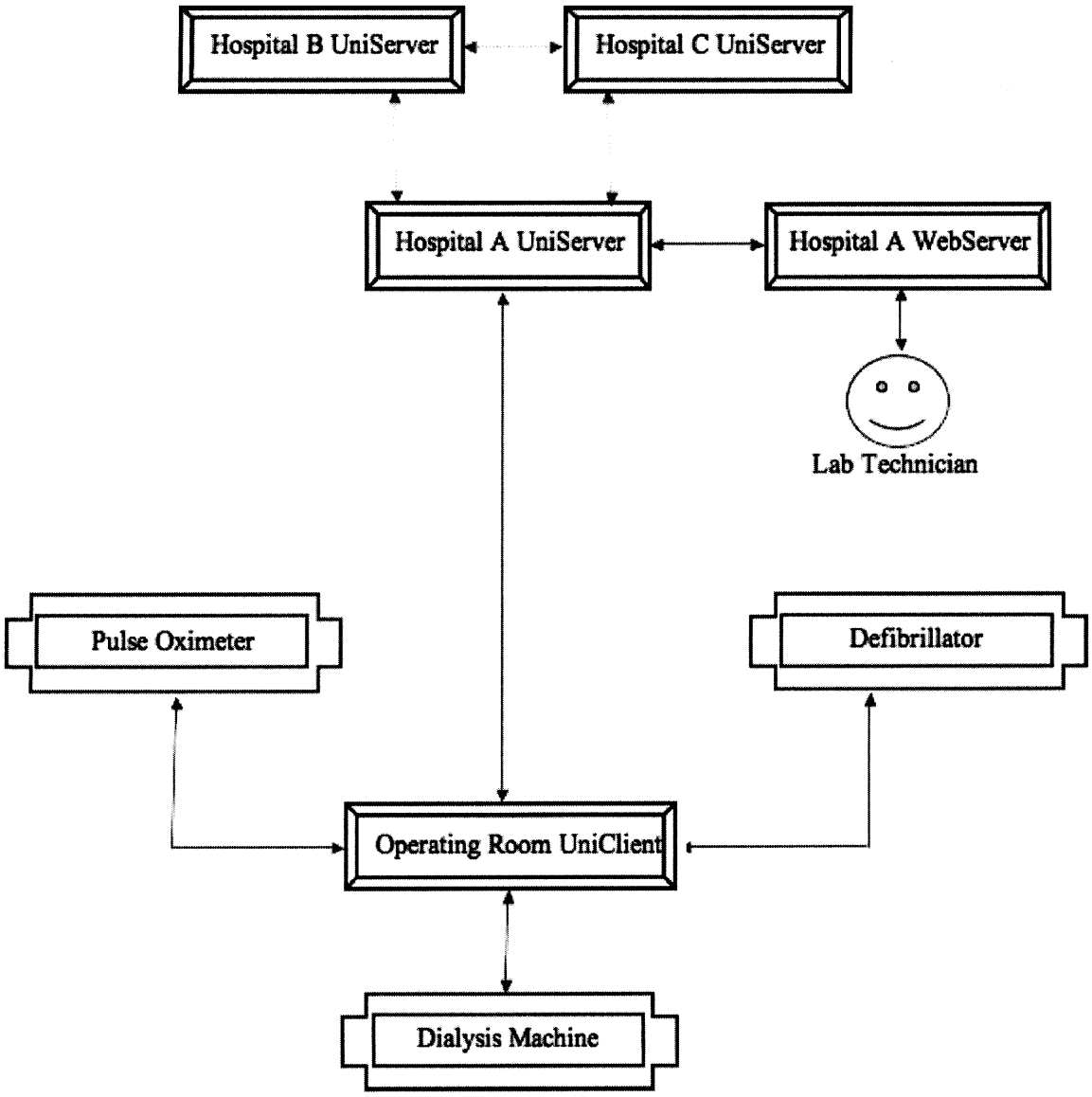

Figure 4-1: UniPlug Application Scenario with Medical Devices

web based interface for hospital staff to administer the UniServer, as well as monitor device states and send configurations for them.

4.4 Issues

Medical devices or software need to be approved by the Federal Food and Drug Administration(FDA) [44] as being safe. They are regulated under the Food, Drug and Cosmetics Act of 1932[45] via the Medical Device Amendments of 1976[46]. There is known set of processes for medical device approval. However the notion of doing a safety check on medical software, as well as the question of interoperability of medical devices is tricky, and even the FDA doesn't have a clear idea of how to standardize interoperability, and what would be an appropriate way to evaluate software and certify it as being patient-safe and reliable[47].

Additionally, a medical system presents special requirements compared to the general cases we have discussed above. The infrastructure for sharing inventions might be ad-hoc, but that for seeking and sending device configuration has to be stable, known and reliable. The system must be able to respond to the device in real time. Certain information such as the patient's temperature can be usually delayed by a few seconds without harm, but other information such as feedback from a heart equipment must reach within milliseconds. This a system designed for medical uses has to be extremely safe, reliable, easy to use, available and robust and presents additional challenges.

Chapter 5

Conclusion

5.1 Summary

In this thesis, we presented a framework for sharing inventions in an ad-hoc way over a campus network. We designed an architecture and associated mechanisms and sketched various architectural models to choose from. We went on to discuss the benefits and drawbacks of each model and the context for which one might be preferable over the other. Further, we discussed issues of scalability, security and context-oriented programming, and illustrated the proof of concept of this system.

Finally, we discussed the role that UniPlug can play in diffusing innovations for medical devices, especially given the current inclination on the part of clinicians to be able to connect medical devices from different vendors over a network.

5.2 Future Work

Future work on UniPlug will consist of building more elaborate and varied demonstrations, as well as refining the ideas and mechanisms we have proposed. For example, a demonstra-

tion that implements all the mechanisms proposed in this thesis. UniPlug holds promise of encouraging and sharing local inventions, particularly local language applications for devices. This could have a big impact on how devices are used in developing countries.

Actual experience with UniPlug will only come with a pilot deployment whose usage is observed and used to refine the idea and implementation. Towards that end, we plan to deploy UniPlug within the MIT community, perhaps starting with the Media Lab, under the Living The Future program. The pilot could then be made larger in scope by incorporating the rest of MIT and eventually being open to the public and being released as open source code. Such a deployment could be either pure P2P or hybrid, making the architecture resilient and grassroots level. The deployment would include all the features of UniPlug, such as safe tagging, leader election, etc.

A demonstration with medical devices is also planned to demonstrate the key value UniPlug can add in that area. One key area to explore will be the import and export of configurations for devices. Once that becomes feasible, a web based interface could be used to send periodic or event based configurations to devices in the UniServers' domain.

Bibliography

[1] iPod. http://www.apple.com/ipod/ipod.html.

[2] FluidVoice Project. http://viral.media.mit.edu/wiki/tiki-index.php?page=FluidVoice.

[3] Avaya One-X^{TM} Quick Edition. http://www.onexdirect.com.

[4] David Reed. An Open Mobile Applications Platform for MIT's Living the Future (OMAP-LTF), MIT-LTF Architecture Note 1, November 2006.

[5] MIT Population, Institutional Research, Office of the provost. http://web.mit.edu/ir/pop/index.html.

[6] David Clark and John Wroclawski. The Personal Router Whitepaper, Version 2.0, March 2001.

[7] Peer-to-Peer Wikipedia Page. http://en.wikipedia.org/wiki/Peer-to-peer.

[8] The CD Data Base. http://www.gracenote.com.

[9] Advanced Packaging Tool. http://www.debian.org/doc/manuals/apt-howto/.

[10] Debian - The Universal Operating System. http://www.debian.org.

[11] Ubuntu Home Page. http://www.ubuntu.com.

[12] Advanced Packaging Tool Wikipedia Page. `http://en.wikipedia.org/wiki/Advanced_Packaging_Tool`.

[13] C. Bettstetter and C. Renner. A Comparison of Service Discovery Protocols and Implementation of the Service Location Protocol. *Proceedings of the Sixth EUNICE Open European Summer School: Innovative Internet Applications, EUNICE 2000, Twente, Netherlands*, September 2000.

[14] Service Location Protocol Version 2, Internet Engineering Task Force (IETF), RFC 2608, June 1999.

[15] Ken Arnold and et al. The Jini Specification, V1.0 Addison-Wesley 1999. Latest version is 1.1 available from Sun.

[16] Salutation Architecture Specification, Version 2.0c, Salutation Consortium, June 1999.

[17] Universal Plug and Play Device Architecture, Version 1.0, Microsoft, June 2000.

[18] Specification of the Bluetooth System, Core, Volume 1, Version 1.1, the Bluetooth SIG, Inc., February 2001.

[19] Sumi Helal, Nitin Desai, Varun Verma, and Choonha Lee. Konark - a service discovery and delivery protocol for ad-hoc networks. In *Wireless Communications and Networking Conference*, 2003.

[20] Michael Barr and Jason Steinhorn. How to Implement a Java Virtual Machine. `http://www.netrino.com/Articles/KaffeAnyone/index.php`.

[21] Gerd Aschemann, Svetlana Domnitcheva, Peer Hasselmeyer, Roger Kehr, and Andreas Zeidler. A Framework for the Integration of Legacy Devices into a Jini Management Federation. In *Proceedings of Tenth IFIP/IEEE International Workshop on Distributed Systems: Operations and Management (DSOM'99)*, October 1999.

[22] ifplugd. `http://0pointer.de/lennart/projects/ifplugd/`.

[23] Ethereal: A Network Protocol Analyzer. http://www.ethereal.com/.

[24] ngrep - network grep. http://ngrep.sourceforge.net/.

[25] Linux-USB Project. http://sourceforge.net/projects/linux-usb/.

[26] BlueZ: Official Linux Bluetooth protocol stack. http://www.bluez.org/.

[27] OpenDHT: A Publicly Accessible DHT Service. http://opendht.org/.

[28] Skype. http://www.skype.com.

[29] PlanetLab — An open platform for developing, deploying, and accessing planetary-scale services. www.planet-lab.org/.

[30] Asterisk :: The Open Source Telephony Platform. http://www.asterisk.org/.

[31] The MySQL Open Source Database. http://www.mysql.com.

[32] Twisted. http://twistedmatrix.com/trac/.

[33] Linda Kohn, Janet Corrigan, and Molla Donaldson. *To Err is Human: Building a Safer Health System.* National Academies Press, 2000.

[34] John Gosbee and Laura Gosbee. *Using Human Factors Engineering to Improve Patient Safety.* Oakbrook Terrace, IL: Joint Commission Resources, 2005.

[35] AAMI News article: Plug-and-Play Connectivity Initiative Launched. *AAMI News*, 40(1):1, January 2005.

[36] Ann Lofsky. Turn Your Alarms On. *Anesthesia Patient Safety Foundation Newsletter*, 19(4), Winter 2004.

[37] Richard Shrenker. IEEE Computer "sidebar" on MD PnP program, in article "Software Engineering for Future Healthcare and Clinical Systems". *IEEE Computer*, page 30, April 2006.

[38] Adam Marcus. Once a Tech Fantasy, Plug-and-Play OR Edges Closer to Reality. *Anesthesiology News*, January 2007.

[39] Julian Goldman and Susan Whitehead. MDPnP Booklet, February 2007.

[40] David Franklin and David Ostler. The P1073 Medical Information Bus. *IEEE Micro*, 9(5), September 1989.

[41] ISO/IEEE 11073-00000, Health informatics - Point-of-care Medical Device Communication - Framework and Overview. `http://www.nap.edu/books/0309068371/html/`.

[42] Health Level Seven, Inc. `http://www.hl7.org/`.

[43] Medical Device "Plug-and-Play" Interoperability Program. `http://mdpnp.org/`.

[44] Food and Drug Administration Home Page. `www.fda.gov/`.

[45] Federal Food, Drug, and Cosmetic Act of 1932. `http://www.fda.gov/opacom/laws/fdcact/fdctoc.htm`.

[46] The Medical Device Amendments of 1976 to the Federal Food, Drug, and Cosmetic Act. `http://www.fda.gov/cdrh/pmapage.html`.

[47] Richard Shrenker. Personal Communication, April 2007.