Remote Access to Development Boards

Lonan Hugh Lardner
B.A.(Mod.) Computer Science
Final Year Project  April 2014
Supervisor: Dr. Jonathan Dukes

School of Computer Science and Statistics
O'Reilly Institute, Trinity College, Dublin 2, Ireland
Declaration

I hereby declare that this thesis is entirely my own work and that it has not been submitted as an exercise for a degree at any other university.

__________________________ 18th April 2014

Lonan Hugh Lardner
Permission to Lend

I agree that the Library and other agents of the College may lend or copy this thesis upon request.

_________________________ 18th April 2014

Lonan Hugh Lardner
I would like to thank my supervisor, Dr. Jonathan Dukes, for all of his support and advice while doing this project. I would also like to thank my proof readers, Wednesday Soper, Michael Clear and Tracey Farrelly. Any remaining mistakes are from me not listening to them. Lastly I would like to thank Michael Broadfoot, the owner of VirtualHere, for giving me a license to use in this project.
## Contents

1 Introduction  
   1.1 Motivation ...................................................... 1  
   1.2 Aims .......................................................... 2  
   1.3 Approach ........................................................ 2  
   1.4 Outline ......................................................... 2  

2 Background  
   2.1 Distance Learning ........................................... 4  
   2.2 Virtual Labs .................................................. 4  
      2.2.1 Trinity College Dublin .................................... 5  
      2.2.2 MIT iLabs ............................................... 5  
      2.2.3 Virtual Lab India ......................................... 5  
      2.2.4 Microsoft ................................................ 6  
   2.3 Technologies .................................................. 7  
      2.3.1 Virtual Machine Platforms ............................. 7  
         2.3.1.1 Virtual Box and VMWare .......................... 8  
         2.3.1.2 Proxmox ............................................. 8  
      2.3.2 USB to IP ................................................ 9  
         2.3.2.1 VirtualHere ....................................... 9  
         2.3.2.2 USB/IP .............................................. 9  
      2.3.3 Web Framework ......................................... 10  
      2.3.4 Access .................................................. 10  
         2.3.4.1 noVNC ............................................. 11  
         2.3.4.2 ThinVNC ........................................... 11
CONTENTS

2.4 Summary . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 11

3 Design and Implementation . . . 12
  3.1 Requirements . . . . . . . . . . . . . . . . . . . . . . . . . . 12
  3.2 Basic Problem . . . . . . . . . . . . . . . . . . . . . . . . . . 14
  3.3 Prototype Solution . . . . . . . . . . . . . . . . . . . . . . . 15
  3.4 Chosen Solution . . . . . . . . . . . . . . . . . . . . . . . . . 16
    3.4.1 Scenario one - Request for Board and Environment . 17
    3.4.2 Scenario two - Request for Just Board . . . . . . . . 18
  3.5 Board I/O . . . . . . . . . . . . . . . . . . . . . . . . . . . . 20
  3.6 Proof of Concept . . . . . . . . . . . . . . . . . . . . . . . . 20
    3.6.1 Management System . . . . . . . . . . . . . . . . . . 20
    3.6.2 Virtual Machine Host . . . . . . . . . . . . . . . . . . 23
    3.6.3 USB to IP Server . . . . . . . . . . . . . . . . . . . . . 23
  3.7 Summary . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 24

4 Evaluation . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 25
  4.1 Requirements testing . . . . . . . . . . . . . . . . . . . . . 25
  4.2 Summary . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 31

5 Conclusion . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 32
  5.1 Future work . . . . . . . . . . . . . . . . . . . . . . . . . . . 33
  5.2 Summary . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 33

Bibliography . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 33

A Management System Model Code . . . . . . . . . . . . . . 36

B Management System View Code . . . . . . . . . . . . . . . 38

C Management System Template Code . . . . . . . . . . . . 42
List of Figures

2.1 iLabs Shared Architecture[1] ................................. 6
2.2 USB/IP Design[2] .............................................. 10
3.1 Summation of the basic problem ............................. 14
3.2 A prototype solution to the problem ...................... 15
3.3 Second solution .............................................. 16
3.4 Chosen Solution - Physical Connections .................. 17
3.5 Request for Environment and Board ....................... 18
3.6 Request for just Board ....................................... 19
3.7 Board I/O ................................................... 21
4.1 The management tool .......................................... 27
4.2 Management tool - environment and board started .......... 28
4.3 Management tool - environment and board shutdown .... 29
Abstract

The goal of this project was to design and produce a proof of concept, a system to allow remote access to a number of development boards. The user needs to be able to flash code onto the board, debug it and execute it. There is a growing need for such a system due to demand for access to such boards and direct physical access is not always possible. The growth in distance learning furthers this need.

The basic approach was to create a virtual lab where the user could request a development environment and a development board via a web interface. A virtual machine would be started for the user and the board connected to it using USB to IP technology. Access and tracking would be handled through a dedicated management server.

A system was successfully put together showing how this system would work in a production environment. Users are able to request boards and environments or just boards directly. They are able to flash, debug and execute their code on them. However, there are modifications that would needed to be made to the system, such as showing input and output from the board, before the system could be used in a live environment.
Chapter 1

Introduction

The goal of this project was to design a system that would allow users access to development hardware without having to physically be at the hardware.

1.1 Motivation

There are a number of reasons why this is worth examining. There is currently a large growth in remote learning, both in terms of students studying purely online courses and modules or videos to support face to face traditional courses. It has also been observed that many students now prefer to use mobile computing options, such as laptops and tablets, rather than using fixed infrastructure, such as traditional computer labs that would be provided by a university.

There is also an issue with access to development hardware. Taking the situation in Trinity College Dublin as an example, there is a single undergraduate lab with development boards. From observation, the lab can hold thirty eight people at one time and there are around two hundred students studying modules that need access to this hardware. This does not include the students that need access to this lab for other modules. A virtual lab would allow the students, that need to use this hardware, access to it regardless of what the physical lab was being used for at the time.

While one option is to run on simulated or emulated hardware, there are
times when doing so does not match the behaviour of actual hardware. This is most easily observed when doing any work related to I/O or interrupts.

1.2 Aims

The objective of this project was to design and build a proof of concept for a virtual lab that will allow users to connect remotely to development hardware such as an LPC2648 ARM Development Board and allow them to program, debug and execute code on it. The system should be simple for the end user to connect to and use. It should be easy to scale. It should try and avoid specialized equipment and should not cost any more then a standard lab.

1.3 Approach

Following an examination of existing technologies, the approach taken in this project was to design a modular system to allow users access to the development hardware. The proposed solution has a central management system that tracks availability of resources and to connect up the resources requested by the user. Development environments are stored as virtual machine images on a number of servers. The boards connected via USB to a server that hosts a system to give access to the boards over IP. A proof of concept was built and evaluated for how well it met the specified requirements.

1.4 Outline

This report explains the background to the situation, discussing the growth in distance learning and previous and existing examples of remote access to tools and equipment. It goes on to examine the technologies that could be used to overcome this problem, including such areas as virtual machines and USB to IP systems.

Next it states the requirements that such a system would have. Following this, it proposes a design and analyzes architectural solutions. A solution is
proposed and two scenarios for its use are looked at. Then a proof of concept that was built is discussed and explanations for the chosen technologies are given.

The proof of concept is then evaluated. It is compared to the requirements and any shortcomings are discussed as are any problems or challenges that were encountered and needed to be overcome.

Finally the project is summarized and possibilities of where it should next be taken are given.
Chapter 2

Background

In this chapter the growth in distance learning is examined as one of the reasons for this project. It goes on to examine existing remote access setups. Next it discusses the technologies that were used or looked at in this project.

2.1 Distance Learning

Distance Learning is the act of providing education to students who are not physically present where the instructor is based. While it is not a new concept, it has experienced a large growth in recent years. As an example the Open University was founded 1969[3]. It opened to students in 1971 with twenty five thousand students. It currently has over two hundred and forty thousand students. This is expected to continue to grow and an increasing number of institutions are offering courses remotely.

2.2 Virtual Labs

In the physical sciences a virtual lab is a computer simulation which is used to perform experiments on. There are two main types: first where a computer model performs the test. The second, called a remote lab, is where the experiments are performed remotely on actual laboratory equipment controlled remotely via computers[4].
An increasing number of universities are now offering access to virtual labs including Trinity College Dublin. Others include such places as John Hopkins University and American University, Washington.

Sometimes this is done to allow users access to licensed software they may not personally have or the institution has a limited number of these licenses\cite{5}. In other cases, such as some of the case studies below, it is to give access to laboratory equipment.

### 2.2.1 Trinity College Dublin

Trinity College Dublin has set up a Windows Virtual Lab \cite{6} for use by students and staff of School of Computer Science and Statistics. It is currently labeled as in open testing. Options are limited to Windows 7. The system requires users to have a remote desktop client and to manually setup with the provided information. It does however demonstrate the concept.

### 2.2.2 MIT iLabs

MIT ran a project from 1999 to 2006. It was called the iLab Project and its stated goal was to:

> “create and demonstrate technologies with the potential for revolutionary change throughout the university curriculum”\cite{1}.

It provided access to real labs through an online interface. It had remote laboratories in microelectronics, chemical engineering, polymer crystallization, structural engineering, and signal processing.

### 2.2.3 Virtual Lab India

The Virtual Lab Project was set up in 2009 as an initiative of the Ministry of Human Resource Development, working with twelve educational institutions. The institutions are: seven IITs (Delhi, Bombay, Kanpur, Kharagpur, Madras, Roorkee and Guwahati), IIIT Hyderabad, Amrita University, Dayalbagh University, NIT Karnataka, and College of Engineering, Pune, \cite{7, 8} The project is based on MIT’s iLab.
The project covers physical sciences, chemical science and various branches of engineering like electronics and communications, computer science and engineering, electrical engineering, mechanical engineering, chemical engineering, biotechnology engineering and civil engineering. Experiments can be run virtually using logged data or run fresh on actual equipment. One of the reasons given for it is to reduce cost by sharing access to costly equipment and resources.

### 2.2.4 Microsoft

A commercial rather than educational system, Microsoft’s TechNet Virtual Labs was established for people to evaluate and test Microsoft products and technologies [9]. It is done in such a way as to make it easy for the user to set up and connect and requires no installation on their part.
2.3 Technologies

The following are some of the technologies examined and used in this project.

2.3.1 Virtual Machine Platforms

Through the use of virtual machine technology it is possible to run a number of separate logical instances of operating systems on the one physical machine. They can be started and stopped on demand and if there are problems with any individual one, its image can be rolled back without any impact on the other systems. There are a wide range of potential systems out there that provide this technology.

There are two types of virtualization technologies: hypervisor virtualization and containerization. Hypervisor virtualization is viewed as full virtualization, guest operating systems are isolated from each other and hardware is emulated. These guests do not share resources with each other. Performance generally takes a slight hit when running with a hypervisor, although with modern computer power, this tends to not be a major factor.

Container based virtualization operates differently. Instead of trying to run an entire guest OS, container virtualization isolates the guests, but doesn’t try to virtualize the hardware. Instead, there are containers (hence the name) for each virtual environment. With container-based technologies, a patched kernel and user tools to run the virtual environments are needed. The kernel provides process isolation and performs resource management. This means that even though all the virtual machines are running under the same kernel, they effectively have their own filesystem, processes, memory and devices. This does usually limit the system to a single operating system being in a container, Windows and Linux cannot be run together for example. While this can be limiting, there is a benefit to performance and scalability. Hypervisor virtualization usually has limits in terms of how many CPUs and how much memory a guest can address, whereas the container-based solutions should be able to address as many CPUs and as much RAM as the host kernel.

The following were the options looked at for this project.
2.3.1.1 Virtual Box and VMWare

Both of these are very similar technologies. Both run on either a Linux or Windows platform and are capable of running a range of virtual machine guests. Both provide web service based APIs to allow remote controlling of them. VirtualBox's web service uses SOAP and WSDL. The APIs for both systems allow for full control of the systems. The following code instantiates a VirtualBox and starts a named virtual machine. It is written in Python:

```python
from vboxapi import VirtualBoxManager
mgr = VirtualBoxManager(None, None)
vbox = mgr.vbox
name = "Linux"
mach = vbox.findMachine(name)
session = mgr.mgr.getSessionObject(vbox)
progress = mach.launchVMProcess(session, "gui", "")
progress.waitForCompletion(-1)
mgr.closeMachineSession(session)
```

The following code will print all registered machines and their log folders:

```python
from vboxapi import VirtualBoxManager
mgr = VirtualBoxManager(None, None)
vbox = mgr.vbox
for m in mgr.getArray(vbox, 'machines'):
    print "Machine '%s' logs in '%s'", %m.name, m.logFolder"
```

The main difference is VirtualBox is open source where VMWare is a paid solution. VirtualBox was the chosen solution for this project.

2.3.1.2 Proxmox

Proxmox was originally chosen to be used as the virtual machine platform as it had a number of advantages over the other two options looked at. The main advantages were that it installed on bare metal as a full operating system based on Linux rather than as an application and it uses containerization. This meant that there would be better sharing of system resources for any
Linux based operating system. However, during implementation of the proof of concept it was found that Proxmox does not allow systems direct access to physical memory. As VirtualHere (discussed in section 2.3.2.1) requires this access to function it was decided to abandon Proxmox and use one of the other virtualisation technologies.

2.3.2 USB to IP

USB to IP is the simple concept of making a USB connected device available over a network to a computer it is not physically connected to by USB cable. It was considered to write a system to do this, but while researching the writing of the system pre-existing technology which appeared suitable was found and used.

2.3.2.1 VirtualHere

VirtualHere[10] is a simple solution to this issue. Its server software can run on a range of devices, Synology, QNAP, ReadyNAS, Raspberry Pi, Beagle, OpenWRT, and Ubuntu (x86,amd64). Its client is limited to Windows and OS X. Its client can be set to run as a service and controlled from the command line. It does have the requirement of a paid license. A free license was provided, by the owner, for use in this project.

2.3.2.2 USB/IP

Built as an open source proof of concept for a paper written in 2005[2], USB/IP is explained in detail (figure 2.2). It has a number of similarities to VirtualHere in that it has a Linux server although the client will run on Linux or Windows. It also has a command line interface. While the paper describes the technology of the protocol in detail, it was decided not to use it for this project as there was no ongoing support if there were any issues encountered and the license had already been offered for VirtualHere.
2.3.3 Web Framework

A web application framework is a system designed to support the development of dynamic websites, web applications, web services and web resources. They usually tie in databases and have a simple to use interface to access and store data in them. The two that were looked at both use the model, view, controller philosophy although in Django[11] it is called model, template, view, with view actually being the controller. Django and web2py are both very similar in how they function. Both use python for the system logic and HTML5 for the front end. Both can also plug into most databases with Django using SQLite as standard.

2.3.4 Access

The area of how to access a development environment, regardless of if it is a virtual machine or just a remote server needed to be investigated. One of the simplest and most effective solutions was to run a virtual network computing (VNC) server on the remote environment. VNC allows for remote access
to a computer. Its desktop is displayed on the client computer and mouse movements and clicks as well as keyboard input is transferred to the remote computer. There are numerous server and client tools out there for doing this as well as other protocols such as the remote desktop protocol (RDP).

2.3.4.1 noVNC

noVNC is an open source, browser based VNC client implemented using WebSockets and HTML5. It is intended so that the end user does not need anything more then their web browser to access a machine running a VNC server. The machine running noVNC functions effectively as a proxy. It does require the VNC server to either have WebSocket support or to use a tool called websockify to act as a bridge. It was examined and time was spent trying to use it in this project so that the management system could function as a single control point so it could be tracked when a user finished using the system. However, issues were found getting a windows based VNC server with WebSocket support and getting websockify to support multiple connections at once due to port conflicts.

2.3.4.2 ThinVNC

ThinVNC is a VNC server with native support for accessing it through a web browser. It does not require any client to access. While the central control was lost, it was chosen as the simplest solution to implement for an area that was felt non critical for the demonstration of how the system operates.

2.4 Summary

Distance learning is continuing to grow and shows no evidence of slowing. Virtual labs have moved on from a demonstration to being used in production environments and actively in education. A number of existing technologies make them easier to setup and should provide additional capabilities.
Chapter 3

Design and Implementation

Here the requirements that the proposed solution need to meet are clearly laid out. The problem is examined, as are a number of prototype solutions. An improved solution is proposed and scenarios for its use are discussed. The next stage of board I/O is then looked at. The proof of concept that was built is examined in detail, as are the technologies used in it.

A note on terminology in this section:

Environment - This refers to any combination of operating system and IDE used for the writing and debugging of code.

Virtual Machine Host - The server on which a virtual machine runs. It provides the underlying hardware that supplies the computing power to the particular virtual machine guests

Virtual Machine Guest - This is an operating system running on a virtual machine host. It is often abbreviated to just virtual machine. In this instance all virtual machine guests are also environments.

3.1 Requirements

The system had a number of requirements that had to be met to achieve the aims of the project. These requirements were as follows:

- Users must be able to access development hardware remotely
Users should be able to select from a range of development environments

- This can be operating system (Windows, Linux, Mac)
- This can also be IDE (Keil µVision, Eclipse, etc.)

The system must support a large number of development boards being connected and used at once

Users should be able to select from a range of development boards (ARM, Arduino, IBM, etc.)

Users should be able to save and resume sessions

Users should be able to choose between using a virtual environment or directly connecting to a board

There were also a number of challenges that needed to be overcome:

- There is a hard limit to the number of USB devices that can be connected to one machine:
  - 60 per port including hubs
  - 127 per host controller

- There is a limit to the number of virtual machines that can be supported on a single host.

The number of virtual machine guests that can be supported on a host is not a hard number, but it can generally be assumed that one to two cores and two gigabytes of memory should be assigned to each virtual machine for the IDEs that will be run on them. This means the system will want to support a number of virtual machine hosts in a cluster. Also running with less guests per host results in better stability, as if any host goes down it brings down less guests, and should also be cheaper.


3.2 Basic Problem

Figure 3.1 outlines the basic problem. There are $x$ number of boards, $y$ number of environments and $z$ number of users. Normally boards are connected to a computer over USB. This raised a number of questions:

- What should the ratio of boards to environments be? One to one or one to many?
- If one to many, how would they be physically connected?
- How would the system change the environment a board was connected to?
- If a user just wanted a board without the environment how would that connection be forwarded?

The other side of the system, between the user and the server raised its own questions:

- How does the user connect?
- What, if any, software do they need?
- Is the environment run on their own computer?
- Do they use a standard remote desktop protocol or some custom protocol to connect?

- What manages what resources are available and what is in use?

- Are development environments running the whole time?

### 3.3 Prototype Solution

![Diagram]

Figure 3.2: A prototype solution to the problem

Figure 3.2 contains a solution to the problem. Each development board is connected directly by USB to one server that hosts a single development environment. Access and tracking availability is handled through a single management server. The users would connect either over a remote desktop connection to the environment, or the connection to the board would be fed directly to their machine over a USB to IP technology.

While this solution would be very easy to implement, only requiring the management server to maintain a list of available development systems and provide the correct connection information to a user when they request it, it
also has the most limitations and is very inefficient. There is no flexibility over what board can be connected to an environment. If a particular combination is either not set up or all are in use, a user cannot access it even if there are available boards and servers.

![Diagram of solution](image)

Figure 3.3: Second solution

Figure 3.3 offers a better solution. In it multiple development boards are connected to each environment server. These would be set up as virtual machine hosts with a number of different environments available. The user would connect to the management server and request the desired environment and board. There are still, however, issues with this setup. It would require the right virtual guest being installed on a host with the correct board being available. Also, while there is a noticeable improvement in the ratio of servers to boards, this can still be improved further.

### 3.4 Chosen Solution

The chosen solution (figure 3.4), while more complicated, was expected to result in a more flexible system that better met the requirements.
In this setup development boards are connected to USB to IP servers and then everything else connects to an Ethernet network. This results in a system that is far easier to scale as more servers and boards can be added independently of each other. The only part of the system that needs to know where everything is and what is available is the management system. It maintains a list of available boards, environments and if desired previous user behavior.

### 3.4.1 Scenario one - Request for Board and Environment

Figure 3.5 shows a use case of a user requesting both an environment and a development board. In this case user (i) wants environment (ii) and board (iii). Both environment and board are available in multiple locations.

1. User (i) contact management server. Server displays list of available environments and boards. User is allowed to select any combination.
2. Management server selects virtual machine host and USB to IP server with desired environment and board available.

   (a) Instruction sent to virtual machine host (iv) to start environment (ii) and connect to USB to IP server (v), board (iii).

   (b) Instruction sent to USB to IP server (v) to prepare for connection.

3. Board (iii) connected to environment (ii).

4. Environment (ii) connected to management server.

5. Management server proxies VNC session from (ii) back to user (i)

### 3.4.2 Scenario two - Request for Just Board

Figure 3.6 covers the scenario where a user (i) only wants a board (ii). The environment used in this case is the user's own computer. Again this type of
board could be available in numerous locations.

1. User (i) contact management server. Server displays list of available environments and boards. User is allowed to select any combination. This time user (i) only selects board (ii).

2. Management server USB to IP server with desired board available. Instruction sent to USB to IP server (iii) to prepare for connection.

3. Board (ii) connected directly to user (i) via USB to IP server (iii).

Scenario one has a number of advantages. The user does not need anything installed or set up on their own machine. Everything can be web based. Scenario two does require installation and configuration of a USB to IP client on the user’s machine, but does allow them use software installed locally.

Figure 3.6: Request for just Board
on their own machine, rather than software installed on a remote virtual machine. The proof of concept that was built was capable of performing both of these scenarios.

### 3.5 Board I/O

An area that was looked at but not pursued in practice due to time constraints, was getting and sending I/O signals from the board. These take the form of LEDs, push buttons or any other devices connected to the processors GPIO pins. It was considered building a device to capture and send data to these pins. This device would have been built from a raspberry pi. These signals could then be run through a system to virtualize these I/O devices. The concept is shown in figure 3.7.

### 3.6 Proof of Concept

A system was built to demonstrate how this design would work. The following is a list of technologies used for each part of the system and the problems and challenges found using them.

#### 3.6.1 Management System

The management system was built as a web service running on Django[11] web framework. It was intended to be straight forward for the end user to use and require no installation on their part. Its only requirement is that they have a web browser that can support HTML5. All major browsers now support this[12]. Users are presented with a list of available environments and boards. Once they have selected what they want, a signal is sent to start the appropriate virtual machine and a script is prepared to connect it to the board. Once the environment has started a scheduled task runs to execute the script to connect it to the board. It had been intended to proxy the VNC session through the management system, however due to problems discussed elsewhere with noVNC, this was not done. Instead the user is
redirected directly to the thinVNC server running on the environment. The management system maintains a list of running environments for the user and the link to connect to them. For convenience it also maintains a list of recently used environments and boards for easier resumption of a session.

**Django** Two major challenges were encountered using Django. The first was simply learning to use it. The theory of how a web services operates is not the same as knowing how to actually build one. Django, however, is well documented and relatively easy to learn what needs to be stored where and what calls to what. The second and more major problem was that the session key objects generated by the VirtualBox API were not serializable and so could not be stored in the provided database to store it when the user...
reloaded the web page either by an actual reload or clicking on any of the input fields on the site. This was resolved by having a dictionary of these session keys stored globally and then the database storing what dictionary key returned the correct session key. While this was not a suitable solution for a production environment, it was deemed sufficient for the proof of concept. A better solution would be to write a serializer for the session key, but that was considered outside the scope of what was needed for this project.

**VirtualBox** The setup of VirtualBox is covered in depth in the next section, but there was some configuration of its SDK to allow access to the API of the web service on the management system to allow it to connect and control the hosts. VirtualBox’s object-oriented web service for Python is dependent on the Python ZSI SOAP implementation. While the documentation for the API is extensive[13] and does cover some examples for Python, it is mostly written to cover all language implementations. This means method signatures can take some experimentation to work out correctly.

**Database** There is a database stored on the management system. It tracks:

- Board names to USB address for VirtualHere
- Environments to IP address for VNC connections
- Running environments, board they are connected to and VirtualBox session key location
- Recently used environments and board combinations

Tables table 3.1 table 3.2 table 3.3 table 3.4 show the database design.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Name</th>
<th>USBAddress</th>
<th>InUse</th>
</tr>
</thead>
<tbody>
<tr>
<td>Field Type</td>
<td>String</td>
<td>Comma Separated Integers</td>
<td>Boolean</td>
</tr>
<tr>
<td>Sample data</td>
<td>LPC2468</td>
<td>22,23</td>
<td>No</td>
</tr>
</tbody>
</table>

Table 3.1: Database - Board
CHAPTER 3. DESIGN AND IMPLEMENTATION

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Name</th>
<th>IP Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>Field Type</td>
<td>String</td>
<td>IP Address</td>
</tr>
<tr>
<td>Sample data</td>
<td>Windows 7 + uVision 4 Development Platform</td>
<td>192.168.1.2</td>
</tr>
</tbody>
</table>

Table 3.2: Database - IP Addresses

<table>
<thead>
<tr>
<th>Field Name</th>
<th>UUID</th>
<th>Name</th>
<th>Board</th>
</tr>
</thead>
<tbody>
<tr>
<td>Field Type</td>
<td>String</td>
<td>String</td>
<td>String</td>
</tr>
<tr>
<td>Sample data</td>
<td>9c9cd219-b91d-462e-9f03-f1108632281</td>
<td>Windows 7 + uVision 4 Development Platform</td>
<td>LPC2468</td>
</tr>
</tbody>
</table>

Table 3.3: Database - Recently used

<table>
<thead>
<tr>
<th>Field Name</th>
<th>UUID</th>
<th>Name</th>
<th>Session</th>
<th>Board</th>
</tr>
</thead>
<tbody>
<tr>
<td>Field Type</td>
<td>String</td>
<td>String</td>
<td>String</td>
<td>Board</td>
</tr>
<tr>
<td>Sample data</td>
<td>9c9cd219-b91d-462e-9f03-f1108632281</td>
<td>Windows 7 + uVision 4 Development Platform</td>
<td>aee7ccf675c8a26a-0000000000000053</td>
<td></td>
</tr>
</tbody>
</table>

Table 3.4: Database - Running Virtual Machines

3.6.2 Virtual Machine Host

Proxmox was originally chosen as the platform to use due to the benefits granted by containerization, but due to problems discussed earlier it had to be abandoned. VirtualBox was chosen to replace it. Three virtual machine guests were installed on the host, two Windows 7 and one Ubuntu. The VirtualBox web service requires some setup. For this project, a Linux platform was used. Settings needed to be set in /etc/default/virtualbox[14]. These were parameters such as user, host address, time out, etc. While, in this case, the web server was on the same machine as the VirtualBox itself, this does not have to be done. The web service and VirtualBox can be on separate machines. This would be a better solution in a production environment where logging and security like logging on is needed.

3.6.3 USB to IP Server

Early in the project it was expected that a USB to IP service would have to be written, but the two options, VirtualHere and USB/IP, discussed earlier were discovered during research. Of these it was decided to use VirtualHere, mainly, due to USB/IP no longer being supported and being a proof of
concept. If there were any issues with it they may be difficult to overcome. Michael Broadfoot, the owner of VirtualHere, also granted a free license to be used for the purpose of this project. Set up of VirtualHere was very straightforward.

The server version is started on the USB to IP server. It can be modified via a configuration file, but it was found that was not needed for the purpose of this project. The client is installed as a service on the virtual environments. The fact that it only has a Windows and OS X client limits some of the options for environments. The script set up by the management system then connects the client to the correct USB to IP server and the correct USB devices via address for the requested board. If the user wants the board connected directly to their machine, they must have the VirtualHere client installed as a service, located at $c:/virtualhere$.

3.7 Summary

This chapter sets out the requirements and the problem in detail. A number of proposed solutions were looked at before one was settled on. The proof of concept was then discussed and how it was built and fitted together was examined.
Chapter 4

Evaluation

The proof of concept is evaluated in relation to the requirements. Where the system does not fully meet the requirements explanations as to the reason are given, as are proposed solutions.

4.1 Requirements testing

The following is a list of the requirements and how the proof of concept compares to them:

1. Users must be able to access development hardware remotely
   - Users may access a development board, program it, debug and execute code on it without being located at it.

2. Users should be able to select from a range of development environments
   - Users can select any environment that is installed on the virtual machine host. However, to connect a board to it they are limited to Windows or OS X. This is a limitation of the system.

3. The system must support a large number of development boards being connected and used at once
While boards must be added to the database manually, there is no limit to how many can be connected. If USB port limits are met more USB to IP servers can be added.

4. Users should be able to select from a range of development boards (ARM, Arduino, IBM, etc.)

- There are no limitations on the system on the type of board connected. In practice the system is not even limited to development boards. Any device connected via USB can be shared.

5. Users should be able to save and resume sessions

- Sessions can be saved on the virtual machine and resumed later. The management tool also maintains a list of recently used virtual machines. However, it is not clear on what was done on what machine. This could be improved by allowing the user to write a note in relation to the machine.

6. Users should be able to choose between using a virtual environment or directly connecting to a board

- Users are able to use a provided virtual environment or connect the board to their own machine and use it. However, this does require specific configuration of their system. They must be using a windows based operating system and the VirtualHere client must be installed in the directory c:/virtualhere. It must also be set to run as a service.
Development System Management Tool

Create new development system:

Select Development environment:
- Windows 7+ uVision 4 Development Platform
- Windows 7+ uVision 4 Development Platform 2

Select Development board:
- LPC2468
- LPC2468b

Launch Environment
Prepare Board Only  Connect

Running development systems:

Running Development Platforms:

Recently used development systems:

Select Development Platforms:

Figure 4.1: The management tool

Figure 4.1 shows the interface for the management tool. In this set up no environments have been started and nothing has recently been used. Users select their desired environment and board from the list of available ones and then launch the environment.
Figure 4.2 shows how it looks following this. The selected options are removed from being available, however, the user can now connect to or shut-down the running environment. Connecting to it starts a VNC session in a new browser window.
Finally figure 4.3 shows the state of the system once an environment has been shut down and a board has been freed. The released environments are available for selection again or the user can just restart that combination from the list of recently used.

While the requirements (1 to 6) were met, the system was found to have some limitations and issues:
• Use of VirtualHere limited the operating systems the user could select from.

• The system seemed to lose Virtual Box sessions keys at random. This would result in the management system not being able to shutdown an environment. This could be overcome by shutting down the operating system through the VNC session. This however caused another problem.

• Following from the previous point, if a user did shutdown using the standard operating system option through the VNC session, it would result in a board not being freed up to be reused.

• The system could take up to five minutes for an environment to start and a board to connect to it. This is longer time then most users would be happy with.

• Even waiting this long sometimes a board would just fail to connect. This could be overcome by running the script installed on the desktop of the virtual machine.

• A large issue, discovered during the demonstration, was if a board froze or locked up there was no way to power cycle it remotely.

The loss of the session key can be explained by how storage of them was implemented. Django’s standard serializer was not able to serialize the session object returned from the VirtualBox web service. This meant it could not be stored in the management systems database. A work around was found to have a global variable of a dictionary object to store these session keys. The database then stored the key identity to the dictionary for the session key. However as a global variable there was no persistence in this object if the management system restarted for any reason. This could be overcome by writing a serializer for the session keys to allow them be stored in the database, but it was felt this was not needed just to prove the design concept overall worked. With regard to the boards failing to connect on occasion automatically via script, while no evidence could be found to explain why
this happened, it is assumed to be down to network issues and the task timing out before access to the remote drive where the script was stored was accessible.

An issue that was found with VirtualHere was that on occasion the address of shared USB devices would change when the client connected. While this does not matter for manual connections, the automated system is dependent on these addresses. While the address it changed to was consistent, a solution was to just instruct the client to connect to both addresses. VirtualHere’s server can be configured via a file, however, the contents of this file were ignored on occasion. No reason could be found for this behavior as there was no access to the source code for VirtualHere. This is the main reason it would be advised to avoid VirtualHere in any future development of this project.

4.2 Summary

The architecture of the design has been successfully demonstrated. While the proof of concept does not fully meet all of the requirements, this is a result of how it was implemented and the technologies chosen rather than of the architecture.
Chapter 5

Conclusion

The proof of concept, while not without its issues, demonstrates that the design concept for the remote lab works. The majority of the issues encountered were a result of technology choices or how things were implemented. The only exception to this is in regards to frozen boards. In terms of technologies and methods used, the following would be taken from this project:

**VirtualHere** While VirtualHere performed the required task in the proof of concept and appears to perform the basic task of sharing USB devices over IP, it felt limiting for the purpose of this project. In a production environment it would be advised to write a specific tool for this job. This could either be built off USB/IP or be created as completely new software.

**VirtualBox** VirtualBox performed the required task adequately. A better understanding of its API would allow for better control of the system and would remove the need for scripts to connect boards and so eliminate issues in that area.

**Django** Use of a framework to build the management tool is purely a preference issue for the creator. In this case it was found to reduce work, but had its own issues as have been discussed previously in regards to serializing the session keys.
ThinVNC ThinVNC provided quick and easy access to the development environments to the users. The only drawback is it meant the management system then had no oversight and so no way to track if a user was still using a system or not.

5.1 Future work

There are a number of areas the design could be improved on. Adding a login system would allow for user profiles setting which boards and environments they had access to. VirtualBox even supports different users having access to different virtual machine guests.

The ability to power cycle boards remotely needs to be added. Whether this is done over its standard USB interface or though a jumper on the board would need further research to determine the best solution.

A means of checking for user activity should be implemented so that resources can be freed up when no longer in use. Actual monitoring of activity, while more work to implement, would be better than just timeouts on their own. Users only having a system for a set amount of time could get frustrated.

The main area that future development should look at is getting input and output information from the board. This area was briefly mentioned in design.

5.2 Summary

As discussed above, there is room for further development of the remote access system. The basic concept is supported by this project. When refined and enhanced it has the potential to benefit both education and commercial environments.
Bibliography


Appendix A

Management System Model Code

```python
from django.db import models
from django.forms import ModelForm

# Create your models here.

class Board(models.Model):
    name = models.CharField(max_length=200)
    usbAddress = models.CommaSeparatedIntegerField(
        max_length=200)
    inUse = models.BooleanField()

    def __unicode__(self):
        return self.name

class Vm(models.Model):
    uuid = models.CharField(max_length=64)
    name = models.CharField(max_length=200)
    session = models.CharField(max_length=15)
    board = models.ForeignKey(Board, null=True)

    def __unicode__(self):
```

36
```
class RecentVm(models.Model):
    uuid = models.CharField(max_length=64)
    name = models.CharField(max_length=200)
    board = models.CharField(max_length=200)

def __unicode__(self):
    return self.name

class VmForm(ModelForm):
    class Meta:
        model = Vm
        fields = ['uuid', 'name', 'session', 'board']

class RecentVmForm(ModelForm):
    class Meta:
        model = RecentVm
        fields = ['uuid', 'name', 'board']

class Ip(models.Model):
    name = models.CharField(max_length=200)
    ip = models.GenericIPAddressField()
Appendix B

Management System View Code

```python
from django.shortcuts import render
from django.http import HttpResponse
import vboxapi
from vm_man.models import Board, Vm, VmForm, RecentVm,
    Ip
import subprocess
import os
import time

import mimetypes
import urllib2
from urlparse import urlparse

store = {}

def index(request):
    boards = Board.objects.all()
    vm = Vm.objects.all()
    mgr = vboxapi.VirtualBoxManager("WEBSERVICE", {
        'url
        ': 'http://localhost:18083', 'user': 'hugh',
```
password': '123456'}
20 vbox = mgr.vbox
21 machine = mgr.getArray(vbox, 'machines')
22 connectURL = Ip.objects.all()
23
24 if(request.GET.get('stopVM')):
25    mach = vbox.findMachine(str(request.GET.get('vm')))
26    stopVM(mach)
27
28 if(request.GET.get('launchVM')):
29    mach = vbox.findMachine(str(request.GET.get('vm')))
30    boardsfil = Board.objects.filter(name=str(request.GET.get('bd')))[0]
31    startVM(vbox,mgr,mach,boardsfil, request)
32
33 if(request.GET.get('startVM')):
34    mach = vbox.findMachine(str(request.GET.get('vm')))
35    boardsfil = Board.objects.filter(name=str(request.GET.get('board')))[0]
36    startVM(vbox,mgr,mach,boardsfil, request)
37
38 if(request.GET.get('launch Board')):
39    boardsfil = Board.objects.filter(name=str(request.GET.get('bd')))[0]
40    connect = open("board.bat", 'w')
41    connect.write('@ECHO OFF
')
42    connect.write('START c:\\virtualhere\\vhui64.exe -t "manual hub add,192.168.1.100:7575"
')
43    for x in boardsfil.usbAddress.split(','): 

44    connect.write('START c:\\ virtualhere\\vhui64.exe -t "use,\')
45    connect.write(x)
46    connect.write('"
')
47    boardsfil.inUse = True
48    boardsfil.save()
49
50    return render(request, 'vm_man/index.html', {"mach": machine, "boards": boards, "vm": vm, 'vbox': vbox, "recent": RecentVm.objects.all(), "connectURL": connectURL})
51
52 def startVM(vbox,mgr, mach, board, request):
53    connect = open(os.path.abspath("/home/hugh/virtualhere/connect.bat"), 'w')
54    connect.write('@ECHO OFF\n')
55    for x in board.usbAddress.split(', '):
56        connect.write('START c:\\ virtualhere\\vhui64.exe -t "use,\')
57        connect.write(x)
58        connect.write('"
')
59    board.inUse = True
60    board.save()
61    session = mgr.getSessionObject(vbox)
62    vm = Vm(uuid=mach.id, name=mach.name, session=
63             session, board=board)
64    vm.save()
65    recent = RecentVm(uuid=mach.id, name=mach.name,
66                       board=board.name)
67    recent.save()
68    store[str(session)] = session
APPENDIX B. MANAGEMENT SYSTEM VIEW CODE

```python
68    progress = mach.launchVMProcess(session, "headless", ", "")
69    progress.waitForCompletion(10000)
70
71    def stopVM(mach):
72        vm = None
73        for v in Vm.objects.all():
74            if v.uuid == mach.id:
75                vm = v
76        vm.board.inUse = False
77        vm.board.save()
78        console = store[str(vm.session)].console
79        progress = console.powerButton()
80        store[str(vm.session)].unlockMachine()
81        vm.delete()
82
83    
84    def boardOnly(request):
85        from django.views.static import serve
86        filepath = 'board.bat'
87        return serve(request, os.path.basename(filepath),
88                       os.path.dirname(filepath))
```
Appendix C

Management System Template
Code

1. `<!DOCTYPE HTML PUBLIC "−/W3C//DTD HTML 4.01
   Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
2. `<html lang="en">
3.   `<head>
4.     `<title>Development System Management Tool − {%block
         pagetitle %}{% endblock %}</title>
5.   `</head>
6. `<body>
7.   `<div id="header"><h1>Development System Management
      Tool</h1></div>
8.   `<div id="nav">
9.   `</div>
10. `<div id="content">
11.   `<div id="new">
12.     `<h3>Create new development system:</h3>
Select Development environment:

```
<form action="#" method="get">
  {% for m in mach %}
    {% if m.getState != "FirstOnline" %}
      <input type="radio" class="btn" value="{{ m.id }}" name="vm" checked='checked'>{{ m.name }}
    {% endif %}
  {% endfor %}
</form>
```

Select Development board:

```
{% for b in boards %}
  {% if not b.inUse %}
    <input type="radio" class="btn" value="{{ b.name }}" name="bd" checked='checked'>{{ b.name }}
  {% endif %}
{% endfor %}
```

```html
<input type="submit" class="btn" value="Launch Environment" name="launchVM">
<input type="submit" class="btn" value="Prepare Board Only" name="launchBoard">
```
APPENDIX C. MANAGEMENT SYSTEM TEMPLATE CODE

```
39   <input type="button" value="Connect"
   onClick="window.open('{{ url 'boardOnly' }}','{{ v.name }}')">

40   </div>
41
42   </form>
43
44
45   <br>
46
47   <div id="running">
48       <h3>Running development systems:</h3>
49       Running Development Platforms:
50   <br>
51   {% for v in vm %}
52       {% for m in mach %}
53           {% if v.name == m.name and m.get_state == "FirstOnline" %}
54               <form action="#" method="get">
55                   {{ v.name }} + {{ v.board }}
56                   <input type="hidden" name="vm" value="{{ m.id }}">
57                   <input type="submit" class="btn" value="Stop" name="stopVM">
58               </form>
59           {% endfor %}
60       {% endfor %}
61   {% endfor %}
62   {% for x in connectURL %}
63       {% if x.name == v.name %}
64           <input type="button" value="Connect"
65              onClick="window.open('http://{{ x.ip }}:8081','{{ v.}}
```
APPENDIX C. MANAGEMENT SYSTEM TEMPLATE CODE

```html
    name\}'}}>>
  60  { % endif % }
  61  { % endfor % }
  62  </form>
  63  { % endif % }
  64  { % endfor % }
  65  { % endfor % }
  66  </div>
  67  <div id="recent">
    68  <h3>Recently used development systems</h3>
    69  Select Development Platforms:
    70  <br>
    71  { % for r in recent %}
    72  { % for m in mach %}
    73  { % if r.name == m.name and m.get_state != "FirstOnline" %}
      74  <form action="#" method="get">
        75  {{ r.name }} + {{ r.board }}
        76  <input type="hidden" name="vm" value="{{ r.uuid }}"/>
        77  <input type="hidden" name="board" value="{{ r.board }}"/>
        78  <input type="submit" class="btn" value="Start" name="startVM"/>
      79  </form>
    80  { % endif % }
    81  { % endfor % }
    82  { % endfor % }
    83  </div>
  84  </div>
```
APPENDIX C. MANAGEMENT SYSTEM TEMPLATE CODE

86  </body>

87  </html>