Remote Access to Virtual Machines in MiG

Bachelor Thesis June 2009

Simon Andreas Frimann Lund

Department of Computer Science, University of Copenhagen.
Advisor: Brian Vinter

Abstract

This thesis presents a design and implementation of a solution model for providing remote access to virtual machines in MiG. The strict requirements for expanding MiG is covered in detail and summarized into a requirement specification. The design and implementation of the solution model is based on the requirement specification and fulfills every requirement. The process of design and implementation is covered and documents other solution models that may be used in less strict environments.

The key requirements for the solution are the demand for anonymization, no use of MiG specific software and firewall compliance. The solution model is based on expanding MiG with a fault tolerant, anonymizing, firewall compliant, packet inspecting proxy and proxy agent. The RFB protocol has been used for providing remote access and any RFB compliant client can gain remote access to virtual machines in MiG.

Solutions are also provided for removing dependencies in the original work on virtual machines in MiG. The utilization of virtual machines no longer rely on: a MiG specific customization of VirtualBox, the slax Linux distribution, and the availability of VirtualBox on the users machine.

Usability enhancements of the web-interface has been implemented and the use of remote access has been implemented with complete transparency for the user.

Contents

1 Introduction

The goal of *the Grid[17]* and *grid computing[17]* is to obtain a system where computing resources can be accessed with the same simplicity as we get power from the power grid. Many models to creating *the Grid* has been developed since what is widely accepted as the first mention of *grid computing* by Ian Foster in 1998.

One such model for *grid computing* is the *Minimum Intrusion Grid (MiG)*[7] started in 2005 by Brian Vinter which proposes a model that addresses issues in other approaches to *grid computing*; allowing slimmer installations on the user and resource side by providing a fatter grid infrastructure.

The MiG model strive to remove fat not only from the installations on the users machine, but also by minimizing requirements on the human users themselves. This has so far been achieved by providing a minimum resource specification language[8] which is simpler and easier to learn in relation to the Globus resource job description language[19].

Learning a language (although simpler) is however still an intrusive operation for the human user, therefore *Tomas Grothe Christensen (TGC)* sought to ease the use of *grid computing* by giving the user access to a familiar *desktop environment[71]* installed in a *virtual machine* on the users computer and migrating the *virtual machine* out into *the grid* when he or she needed the computational resources provided by *the grid*.

In [11] TGC describes how he has implemented this in MiG with the virtualization software VirtualBox[33], when the *virtual machine* is migrated to MiG the user is disconnected from the *desktop environment*.

What this thesis proposes is a way of obtaining *remote access* to the *desktop environment* while the *virtual machine* is executing in MiG without violating the *MiG rules[6]*.

1.1 Problem Definition

This thesis seeks to explore the possibilities for implementing remote access to the *desktop environment* when the *virtual machine* is running in MiG by answering the questions: What is required to provide remote access to virtual machines in MiG? Is it possible to meet the requirements without violating the MiG rules and design criteria?

1.2 Related Work

Remote access has its roots from text terminals, graphical terminals and to what is today known a *thin clients*, with the advent of personal computers the trend shifted towards having the needed computing power locally. This trend in computing seems to continuously shift back and forth.

Virtual Network Computing (VNC)[58] is an ultra thin client system based on the *Remote Frame-Buffer Protocol (RFB)[65]*. VNC is because of the simplicity the RFB protocol totally platform

independent and can be implemented in hardware in thin-clients or as a software solution. A similar approach to remote access, with more features, but with high dependency on the operating system are the Microsoft Terminal Services based on the *Remote Desktop Protocol (RDP)[23]* and Citrix XenAPP based on the ICA protocol. None of these systems however are integrated with *grid computing*.

Work has been made in *virtual machine grid computing[63]* to provide a layer two network tool that connects a virtual machine based on VMWare to the local network that the user resides on. The work in this paper provides a method to provide remote access without installing a network tool on the user machine and without dependency on the hypervisor chosen for virtualization.

Remote access to *desktop environments* is thus not a new need, the *desktop environment* is a very popular and widespread graphical user interface. This was also why TGC chose to use the *desktop environment* as a way to ease the utilization of MiG. The new problem is to close the gap between *remote access*, *desktop environments* and *grid computing*.

The problem definition is related to the latest buzzword *cloud computing[69]* one key aspect to *cloud computing* vs *grid computing* is that it focuses on delivering services to its users rather than general purpose computing resources. Three groups of services in cloud terminology are: Platforms as a Service (PaaS), Infrastructure as a Service (IaaS) and Software as a service (SaaS).

The work in this thesis can in terms of cloud computing be described as Desktop as a Service (DaaS).

1.3 Delimitation

The interaction with the *desktop environment* is limited to being screen output, keyboard and mouse input and leaving features such as embedded file transfer, high definition graphics and other features for future work.

This thesis provides remote access to virtual machines as described in [11], a delimitation is needed here since other work in MiG uses virtual machines for public resource computing.

1.4 Source and Documentation

This thesis and all material related to it is available on-line on the MiG google code project site: http://code.google.com/p/migrid/source/browse/#svn/branches/vm-job-vnc

1.5 Terminology

In MiG the term *fat* is used as a measurement to describe where grid-enabling technology is added. Often distinguished between adding fat to resources, users or the MiG server.

2 Analysis and Requirements Specification

This goal of this project is to answer the questions:

1. What is required to provide remote access to virtual machines in MiG?
2. Is it possible to meet the requirements without violating the MiG rules and design criteria?

These questions will be answered in an analysis of the MiG rules and design criteria, an analysis of the available protocols and software stacks for providing remote access and an analysis of the architectural and infrastructural requirements. Each subject will be discussed in its own subsection and a summary of the requirements will be individually provided in each subsection. This current section is ended with a requirements specification summarizing the individual requirements of each subject. And thus an answer to the first question is provided in the form of a list of requirements.

An overview of the final solution model in relation to the requirement specification is provided in Section 3 with detailed design considerations and implementation issues described in Section 4. A short answer to question two is thus provided in Section 3 and a long detailed answer is given in Section 4.

Before continuing into coverage of requirements then it must be made clear what is meant by *remote access*. Remote access has historically meant different things, a software product with the name *RemoteAccess* has existed, multiple protocols using the name *remote access protocol (RAP)* exists. In this project the term remote access will be used to describe a means for providing remote access to a desktop environment. The remote access must enable a user of MiG to:

- See the graphical desktop environment.
- Interact with the desktop environment and applications available in the virtual machine by using mouse and keyboard.

The motivation in the original work for providing remote access is that visual job-status can be provided. Remote access however also provide all new ways of utilizing the computing resources available via MiG. The interaction provides the possibility for users to continuously monitor a simulation and change parameters as needed. Remote access will provide a means for using MiG as middleware for Virtual Desktop Computing / Desktop Virtualization. The motivation for providing remote access is high and the following section starts the uncovering of the requirements for doing so.

2.1 Minimum Intrusion Grid

I have briefly commented on what the goal of *grid computing* is in general and how MiG is different model. From an abstract top down view MiG is the grid glue that connects the entities of users and resources an abstract illustration of this is given in Figure 1 on page 8.

The MiG server is placed somewhere on the Internet, users and resources do not communicate with each other but use MiG as the middleware.

I will describe the technical details of MiG that has an impact on this project and the rules and goals of MiG that direct the design and implementation. Further details are available on the MiG websites[34][35].

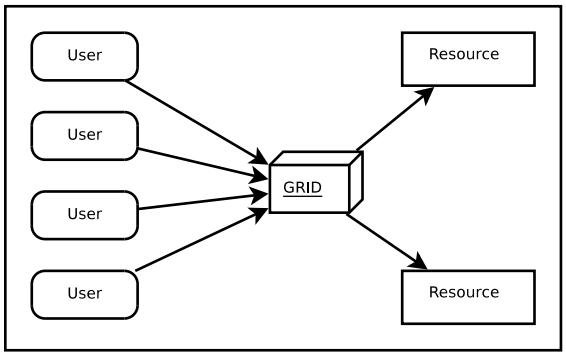


Fig. 1: The simple MiG model [8].

2.1.1 Rules

- Nothing produced by MiG can be required to be installed on either the resource or client end
- Everything within MiG must be implemented in Python unless another language is absolutely required
- Any design and implementation decision must optimize towards transparency for the users
- Anything that is not right must be thrown away

These are the rules of MiG as described in *MiG rules* [6], in addition specific design criteria must be met to ensure that MiG is and continue to be: non-intrusive, scalable, autonomous, anonymous, fault tolerant, firewall compliant, providing strong scheduling and cooperative support.

2.1.2 Users

In MiG a user requests access to computing resources by formulating jobs in the mRSL language, the following commands exist for job manipulation: job submission, job status of one or more jobs and job cancellation. Having access to computing resource is quite useful, but it is usually a lot more useful when the jobs compute something based on user given input and produces some output. MiG caters for this need providing access to files in the methodology of a home directory that is well known to UNIX/Linux users in its layout.

The commands for manipulating files in MiG are: download and upload files to and from MiG and shell-like commands such as cat/head/tail/wc/stat/touch/truncate/rm and rmdir. Additionally a command to show on-line documentation exist. I will refer to this set of commands/interactions as *core interaction*.

The *core interaction* is provided to users via different interfaces; web-interface, bash scripts, python scripts, XML-RPC[72], c++ library and in current development JSON[24]. The first three are focused on human interaction and the three others are used for machine interaction with MiG. The interfaces serve different purposes but they share the common feature that all interaction is only allowed when the user possesses and presents a valid x.509 certificate.

MiG distinguishes itself from other Grid systems by providing user experience enhancing tools for the *core interaction* such as the scripts/XML-RPC/c++ library and JSON but these are enhancements to the interaction and not required for the user. Thus the only requirement on users is to learn the mRSL language, request an x.509 from MiG, a HTTPS capable browser and Internet access.

In this project I continue the work of TGC who has created a tool for *high-level interaction* with MiG. The high-level interaction is based on interaction with a desktop environment, the desktop environment is running inside a virtual machine based on the virtualization software VirtualBox. The frontend of VirtualBox has been expanded to integrate the *core interaction* so the user does not need to. Instead they install a MiG specific customization of VirtualBox on their machine and interact with their applications and data inside the desktop environment of the virtual machine. When needing the computational resources of MiG they instruct VirtualBox by clicking a button to deploy the virtual machine out into MiG. VirtualBox accomplishes this by utilizing the *core interaction* wrapped in a c++ library.

2.1.3 MiG Servers

MiG is currently based on the Apache webserver, an SSH daemon and the actual application code written in python and some parts in Java (One-Click). MiG uses Debian stable as the operating system platform and thus provides the mentioned software in the version available in Debian stable. The MiG server handles requests from the users via *core interaction* and schedules jobs to be executed on the resources available to the user. MiG stores data for the user and serves the data for resources when they need it in the job execution process. Jobs are given unique identifiers, the identifiers are private meaning that only the owner of a job knows the job-identifier.

Resources are informed of the job-identifier but are unaware of the mapping between user and job identifier only the MiG server and the owner is aware of the mapping.

2.1.4 Resources

Four different types of computing resources are available in MiG, in the *Getting Started Guide[3]* they are called *Full scale*, *One-Click*, *Screen-Saver-Science (SSS)* and one not described in [3] but in [32] is the *PS3 live-CD*.

	HTTPS out	SSH out	SSH in
Full scale	+	+	+
Sandbox	+	+	-

Tab. 1: Firewall resource requirements.

The computational power and expected availability vary greatly, the *Full scale* are dedicated machines, with the sole purpose in life being to deliver computing resources to MiG, their availability can be expected to be high, other resources such as the *SSS* are more limited since they bring computation to MiG only when the machine they are sandboxed in is not being used, and can at any time be aborted.

The availability is one distinguishing characteristic of the resource types, another is availability of network access, the *Full Scale* offer both SSH and HTTPS in and out where the rest only offer outgoing traffic as illustrated in figure 1.

The VirtualBox virtual machines run on *Full Scale* resources because of the deployment times in migration from user machine to resource, then the *full scale* resources are the types which are best fitted for the purpose.

Communication with *full scale* resources utilize SSH and HTTPS. Even if an organization has multiple resources behind one firewall then only only one port needs to be opened in the firewall. This is because resources are assigned roles of *frontend node* or *execution node*, or both, as specified in the *job flow description [5].*

The purpose of a frontend node is to communicate with MiG and deliver jobs to execution nodes. If only one resource exist in an organization then it acts as both frontend and backend node, an organization with multiple resources configures one resource acting as the frontend and forwards the SSH port to it. The remaining resources are configured as execution nodes and do not communicate with MiG but with the frontend node.

2.1.5 Virtual Machines

VirtualBox is a user space application using a kernel-module to map into the kernel of the host operating system. For reference a figure of the layers in relation to the desktop environment in the virtual machine is provided in Figure 2 on page 10. As described in section 2.1.2 *core interaction* has been integrated into VirtualBox this means that as long as the user has a valid certificate she only needs VirtualBox to use MiG. It is also the only way for the user to use Virtual Machines thus it is a requirement for the users to install VirtualBox to get the facilities of a desktop environment.

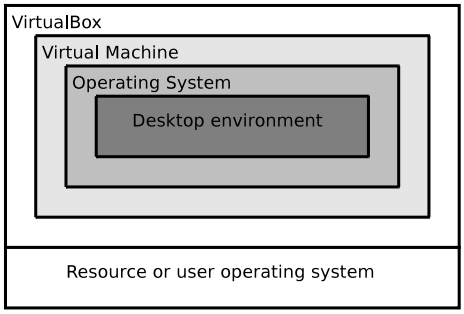


Fig. 2: Organization of layers.

It is worth noting that the virtual machine is configured with networking enabled via the hosts (user or resource machine) interface but ingoing traffic is denied in the operating system, this is done to restrict network access to the virtual machine from the resource owner and from the network that the resource is installed in. However no restrictions are applied to outgoing network traffic in the virtual machine or operating system, thus Internet access can be expected to be available to the same extent that Internet access is available to the resource.

2.1.6 Summary

MiG enforces strong rules and design criteria that must be strictly followed, the present project is firmly focused on maintained anonymity for both users and resources, no installation of MiG

specific software, firewall compliance and transparency for users. Remote access must thus be provided by transport via HTTPS or SSH in a way that does not require any software installed neither on the user machine or resource.

2.2 Protocols and Software

The MiG rules state that nothing produced by MiG can be required to be installed on neither user nor resource end. Thus an existing protocol and software suite should be chosen instead of developing a MiG specific remote access protocol and software stack. This section seeks to find an existing protocol and software suite fulfilling the following requirements:

1. Software must provide visual output of the desktop environment.
2. Software must accept keyboard and mouse input.
3. Software license should be compatible with GPL.
4. Protocol specification should be documented and available.

Requirements one and two are directly linked to the goals of the project, requirement three is stated to ensure that the chosen software can be be distributed with MIG.

The alternative to requirement four is to extract protocol information from the source code and reverse engineer network packages on live remote access sessions. This alternative is feasible but highly time consuming so choosing it must be based on significant gains in the other components.

2.2.1 Protocol

Below is a list of proven remote access protocols.

- RFB - Remote frame-buffer, specification available.
- RDP - Remote desktop protocol, specification available.
- X11 - X window system display protocol, specification available.
- NX - NoMachine NX, specification unknown.
- ICA - Independent computing architecture, specification unknown.
- AIP - Adaptive Internet Protocol, specification unknown.

Of the listed protocols RFB, RDP and X11 are the most interesting since they are properly documented, X11 however is very tightly linked with the X display server and it might prove hard to find software stacks implementing the protocol that are not tightly coupled with X display server and thus require the use of X. RDP has a similar constraint, it is developed for the Microsoft Windows operating system and the protocol specification is comprised of 9 documents

specifying various Windows specific features, the base specification is about 348 pages, RDP is well documented but also a very complex protocol. The RFB protocol however is completely decoupled from the operating system and has no operating system constrains in the specification.

2.2.2 Embedding Remote Access

The MiG Rules state that no software should be installed on neither client nor resource end. If no software is allowed to be installed on the client then the user must either already have the software installed or alternatively a Java client that can be run from the browser could be used. Most version of Windows comes with an RDP compatible client, most popular Linux distributions comes with VNC clients. However in various environments such as Internet cafes and libraries access to the software is often restricted and only access to games/book database and a web browser is allowed. The vision of grid computing is also about providing mobility by providing access from all types of heterogeneous environments only providing a browser and the user bringing her certificate.

The use of browser based client via Java thus seem appealing, since it allows access to computing resources in these restricted environments. But a Java enabled browser could be regarded as being intrusive, they are however widespread and the One-Click in MiG already utilize Java and it can therefore be regarded as an acceptable intrusion.

An entirely different approach would be to use *comet[13]* or *web-sockets*[68], this however would require that the remote access client must be implemented from scratch in ECMAScript/-JavaScript, this approach would not pose any restrictions on the user and no software would need to be installed. But it would violate my delimination of not implementing the software stacks from scratch. The technology is however very interesting and might prove very useful to MiG in other projects.

This still leaves the issue of installing software on the resource end, this can however be avoided by installing the software inside the virtual machine and not on the resource. Because of this a software approach to providing remote access is feasible.

Remote access on the server side can also be provided by embedding it into VirtualBox. The commercial version has integrated RDP support but the open source edition that MIG depends on does not. In section 4.1.2 an evaluation of re-embedding remote access in VirtualBox is provided.

2.2.3 Software Comparison

In the *comparison chart[70]* 48 different software stacks for remote desktop are compared, 32 of them being proprietary an hereby ill-fitted for distribution with MiG. Client and server software need not be provided from the same software stack as long as a client and server use the same protocol and Java client exists. Six such clients are available (jrdesktop (proprietary), rdesktop(RDP), RealVNC Free(RFB), TightVNC(RFB), UltraVNC(RFB) and x11vnc(RFB)). The

choice of server software must thus use either RDP, RFB or the proprietary jrdesktop protocol. There are not any server software that runs on all platforms, nor is this a direct requirement. But sound choices for server software would be FreeNX, iTALC, jrdesktop, RealVNC Free, TightVNC, UltraVNC, xrdp or x11vnc.

jrdesktop however is the only software stack using its own proprietary protocol. Using jrdesktop would mean that MiG would rely heavily on the continued development of the jrdesktop project I therefore do not see it well-fitted for this project.

FreeNX is a very interesting project they claim technical superiority over all existing remote access solutions, they do this by accelerating and compressing X11 traffic via nxproxies (see Figure 3 on page 13) support is also provided for translating RFB and RDP based servers into FreeNX/X11 protocol. On the client side an X11 client or commercial NoMachine client must be used. However for this project the accelerating architecture cannot be used since a FreeNX proxy cannot be installed on the users computer, without the nxproxy locally available to the user then none of the advantages of the FreeNX system can be used.

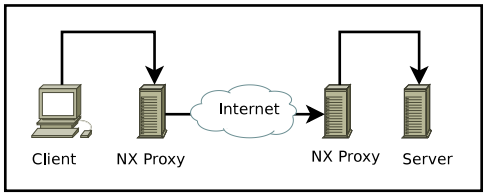


Fig. 3: FreeNX Architecture

This leaves the best fitted software choices as listed in Table 2 on page 13.

		Client	Server	Java Client	FreeBSD	Linux	MacOSX	Windows
RDP	rdesktop	1	0	1	1	1	1	1
	xrdp	0	1	1	1	1	0	0
RFB	iTALC	1	1	0	0	1	0	1
	RealVNC	1	1	1	1	1	0	1
	TightVNC	1	1	1	1	1	0	1
	UltraVNC	1	1	1	0	0	0	1
	x11vnc	0	1	1	1	1	1	0

Tab. 2: Best fitted remote access software.

In the case of RFB the software stacks are lacking a standardized encryption scheme for communication. This is due to the fact that the *RFB specification*[65] only specify simple three-way handshake, more advanced types of authentication and encryption of traffic is left non-standardized, no common ground for these task are available therefore other means for providing such features must be found.

An interesting software stack is x11vnc. The interesting part is that the development team has organized the development around creating a portable c library for all of the VNC functionality. The RFB protocol is highly portable and with the libvncserver a highly portable c library is available.

2.2.4 Summary

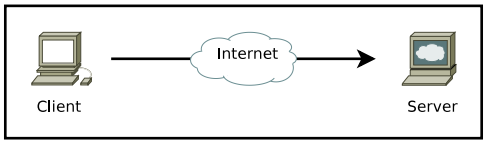
Table 2 on page 13 shows that the best supported protocol is RFB, RDP only has one server and one client available. The RFB protocol is a good choice; it is simple, well documented, well supported by Java clients and server software is available for all platforms.

2.3 Architecture

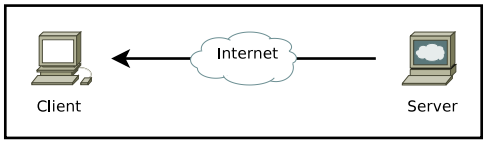
Architectural requirements for supporting anonymous remote access to virtual machines based on the RFB protocol must be defined.

RFB is an application level protocol implemented on top of TCP/IP in the traditional client/server architecture. When access to the server is needed the user enters the IP address and port in the client and connects to the server as illustrated in Figure 4a on page 15. This method of connection initiation is the regular way of establishing connections in a client/server architecture. It is quite natural that users initiate connections since the job of a server is to service users, the server normally does not know who to service at a given time.

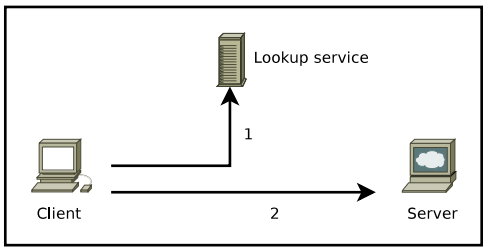
Some servers however also implement a reverse initiation method as illustrated in Figure 4b on page 15 where the server initiates the connection to the user. The use of reverse connections are not that widespread in the client/server architecture but it is useful in environments where the server for various reasons are not allowed incoming traffic.



(a) Regular connect.



(b) Reverse connect.



(c) Look-up service.

Fig. 4: Initiation methods.

Either way the initiating party must know the address and port of the other end, in MiG neither end-point knows the address of the other. A simple solution to this could be to provide a look-up service as illustrated in Figure 4c on page 15 where the connection initiation is based on a connection profile retrieved from a look-up service. Such an architecture could be integrated with regular DNS name look-ups, MiG could use dynamic DNS to push DNS entries of remote desktops to DNS servers and DNS naming convention such as desktop47.migrid.org could be used.

However trying to facilitate basic end to end connections in this manner, even when using a smart look-up middleware still pose a very big problem for integration with MiG. According to the design criteria anonymity must be maintained, traffic should be restricted to SSH or HTTPS and be firewall compliant. These traditional architectures violate all design criteria of MiG, the end-points are directly connected and thus anonymity is lost, traffic is plain RFB not SSH or HTTPS and this breaks firewall compliance.

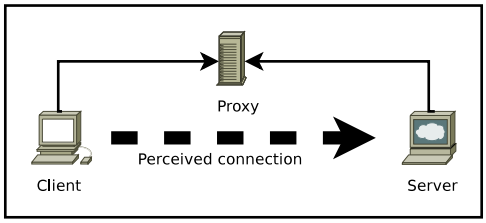


Fig. 5: Proxy-based architecture.

A proper architecture should maintain anonymity for both parties (users and resources), be firewall compliant and somehow utilize either SSH or HTTPS for the transport. It should also provide a means for packet inspection and content

filtering to provide features such as content rewrite. This is needed since most protocols and also in the RFB protocol, the hostname of the server is send in the handshake, this would reveal the identity of the server and compromise anonymity. If the architecture can not filter out such information then the chosen software stack should be able to disable sending that information. From the set of best-fitted software only the x11vnc server is able to parametrize the hostname that is being send.

Various solutions for providing anonymity on the Internet exist; open HTTP proxies, SOCKS proxies and advanced daisy-chained anonymous proxies such as the TOR network. In MiG anonymity is sought to hide users and resources from each other, complete anonymity is not sought after, MiG in fact must know about the participating parties, thus complete traceless anonymity is not the goal.

The use of a proxy-based architecture should also be completely transparent to the user, the proxy should be able to create a perceived connection from the client to the server from a users perspective. Since only outgoing traffic is available from the virtual machines OS then the proxy should be able to receive connections from both clients and servers. Such an architecture is illustrated in Figure 5 on page 15.

2.3.1 Summary

The expansion of MiG with an anonymizing proxy with packet inspection and content manipulation is the best-fitting architecture for enabling the needed communication without violating the *MiG rules*. SSH or HTTPS must be used for firewall compliance.

2.4 Requirements Specification

As promised in the start of section 2 then the requirements gathered will be formed into a requirements specification. The requirements specification is provided below as a numbered list of requirements. The design of a model to provide remote access to virtual machines in MiG must satisfy this list of requirements.

1. Be compatible with VirtualBox Open Source Edition.
2. Require no MiG specific software to be installed on user nor resource machine.
3. Maintain anonymity of users and resources.
4. Use Python as the implementation language.
5. Design decisions must strive towards total transparency for users.
6. Use HTTPS or SSH for transport as basis for communication to be firewall compliant.
7. Use the RFB protocol.
8. Be able to use both Java and external RFB compatible clients from the list of best-fitted software.
9. Use an infrastructural architecture expansion such as a proxy.
10. The proxy must be able to provide anonymization of communication endpoints.
11. The proxy must be able to perform content manipulation to ensure anonymization of identifiable data carried in protocol.

The list of requirements has some conflicts with the existing work on virtual machines in MiG. A conflict with requirement two occurs since users and resource owners are required to install a customized version of VirtualBox on their machines to be able to migrate virtual machines to resources. VirtualBox is currently the only way to interface and manage virtual machines in MiG, thus to resolve the conflicts these additional requirements must be met:

12. Virtual Machines must be migrated from the MiG server.
13. Access to Virtual Machines must not depend on a MiG specific version of VirtualBox.

In the following section these requirements will form the base for the design and implementation of a model to provide remote access to virtual machines in MiG.

3 Solution Model

The final solution model is comprised of a proxy, proxy agent and a protocol for communication between the two. The web-interface has been expanded to provide management and interaction with virtual machines. Support for migrating virtual machines via the web-interface and without installing VirtualBox on the virtual machine has been added.

How this solution model satisfies the requirement specification is covered by examining the requirements below.

1,12,13 Compatibility with VirtualBox OSE has been maintained additionally dependency on VirtualBox has been removed. Migration of virtual machines are provided by an expansion of the web-interface with a subsystem for managing virtual machines and by moving logic for importing MiG virtual machines and monitoring their execution time out of the customized VirtualBox and into the job encapsulation.

2 Remote access has been embedded without requiring the user to install MiG specific software, the user can utilize any RFB/VNC client they already have or they can choose to use a Java client integrated into the web-interface. The dependency on MiG specific customization of VirtualBox has also been removed as described above.

3 Anonymity is maintained for users and resources with the introduction of an anonymizing proxy and as described in Section 4.3.2 then care has been taken to utilize a shared-secret without revealing identity.

4 All parts of the solution: Proxy, Proxy Agent, Interface and Virtual Machine builder are implemented in pure python.

5 By moving management of Virtual Machines into the web-interface and by designing and implementing web-interface enhancements by user driven design and direct manipulation then complete transparency for users has been achieved. The considerations for which are available in Section 4.5.

6 Firewall compliance has been achieved by implementing *MiP/RFB over TLS* as described in Section 4.4.1, effectively traversing firewalls and encrypting traffic.

7,8 The RFB protocol has been used without depending on vendor specific registered extensions or VNC implementations. The issues are discussed in further detail in Section 4.1.1.

9,10,11 A proxy and proxy agent has been implemented that anonymize traffic as described in Sections 4.2-4. The proxy is able to perform packet inspection and content manipulation which enables the use of any RFB, by implementing identification via packet inspection as described in Section 4.2.1 and 4.3.2.

3.1 Implementation Overview

The solution is available on the MiG project site[34] as a branch of the MiG code-base with the name *vm-job-vnc*. The detailed modifications of the code-base can be inspected by reading the changelog, links are provided in appendix A.1. The location of the changes are briefly covered in Table 3 on page 19. A figure of the organization of the proxy code is given in Figure 23 on page 56.

Path	Purpose
/proxy	Contains the proxy and proxy agent
/mig/cgi-bin/images	Added graphics for web-interface and changed style sheet.
/mig/cgi-bin/vmachines_*	Contains the interface changes for virtual machine management and remote desktop interaction.
/mig/cgi-bin/shared/vms.py	Contains a library for the virtual machine management.
/state/server_home/vms	Contains skeleton files for the creation of virtual machines.
/builder	Contains the builder environment for python-vm-builder.

Tab. 3: Location of changes in repository.

3.2 Summary

The MiG rules and design criteria exist to ensure the distribution of fat in the grid model, fat should be removed from users and resources and placed in control of the grid middleware. The final solution model shows improvements in the distribution of fat as can be seen in Figure 6 on page 19 .

	User	MiG	Resource
VirtualBox	+		+
Virtual Machine	+	+	+
Data	+	+	+
Proxy			
Java in Browser			
VM Management	+		

(a) Original Work

	User	MiG	Resource
VirtualBox			+
Virtual Machine		+	+
Data		+	+
Proxy		+	
Java in Browser	+		
VM Management		+	

(b) Solution Model

Fig. 6: Distribution of Fat in Solutions.

The design and implementation of the solution model is covered in detail in the following section.

4 Design and Implementation

This section documents the process of designing and implementing a solution and thus answering the question of whether remote access can be provided to virtual machines in MiG under the requirement specification formed in the previous section.

The design and implementation is based on initial research and experiments to uncover the facilities provided by the RFB protocol, evaluate potential for re-embedding remote access into VirtualBox, researching proxy design and potential for reusing existing proxies.

The initial research is used to implement a standalone proxy in the first iteration proxy design, the issues are evaluated to provide the knowledge for integrating the standalone proxy with MiG.

4.1 Initial Research and Experiments

4.1.1 RFB vs VNC

The *remote frame-buffer protocol (RFB)[65]* protocol was developed for use with a hardware based thin-client called a *VideoTile* developed by *Olivetti Research Laboratory (ORL)*. As briefly covered earlier RFB takes a different approach to remote access decoupling it from the high-level systems by injecting directly into the frame-buffer layer. This approach makes RFB a very simple protocol since it is only purpose is to serve frame-buffers to clients as illustrated in Figure 7 on page 20 and to handle input from keyboard and mouse from the user. ORL later introduced *Virtual Network Computing (VNC)[58]* which could be called a software version of the VideoTile. VNC is based on the RFB protocol. However with the advent of VNC then new features where needed. But instead of standardizing the features then *registered extensions* where added to the RFB specification.

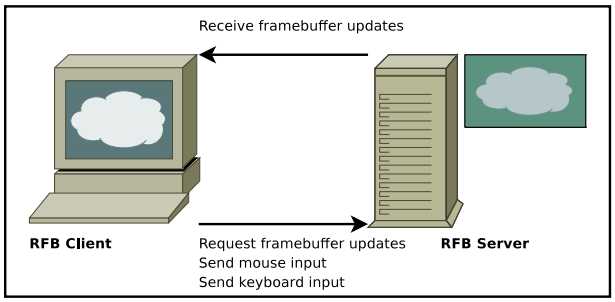


Fig. 7: Fundamentals of the RFB protocol.

In literature references are often made to the VNC Protocol, however no such protocol definition exist. When references to the VNC Protocol is made the reference is often meant as a reference to the RFB protocol with some vendor specific registered extension. This means that there is no common ground for providing proxy support, secure authentication or more efficient encoding schemes.

This means that the integration of remote access based on RFB/VNC can only rely on the features specified in *[65]* but should support the advances made by VNC. This covers the limitations of the protocol now a closer look at the features available will be covered. It will be uncovered whether the protocol reveals any sensitive information in the protocol messages and what the interconnection requirements are.

The flow of the RFB protocol is illustrated in Figure 8 on page 21, RFB performs a handshake and initialization before sending frame-buffers. In the handshake one of the supported authentication methods is used. Only two methods are standardized *None* and *VNC Authentication*. VNC Authentication uses a basic three way handshake to authenticate the user. It is noted that VNC Authentication might reveal the identity of the user and the proxy should filter out such a leak of information.

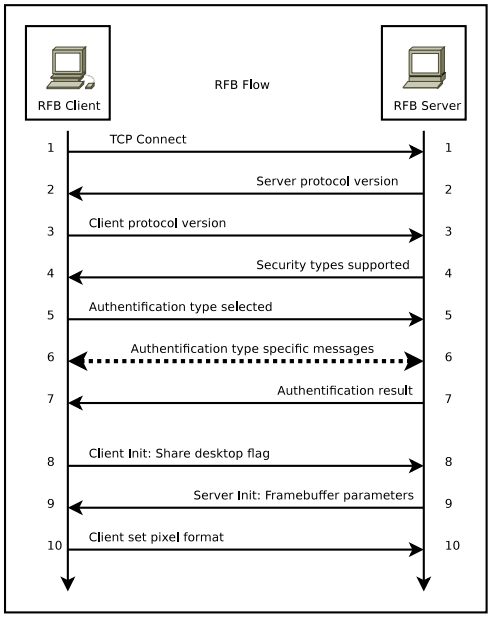


Fig. 8: RFB Flow

After the handshake phase initialization is performed to negotiate the set of supported frame-buffer encodings, no sensitive information is carried in the client initialization but the server initialization reveals the hostname of the server, it is noted that the proxy should filter out this information.

The performance of using RFB for remote access depends on the supported encodings, the resolution and color depth of the remote display. As an example, take a display of 1024x768 with a color depth of 24bits encode it with the RAW scheme and deliver it via 10Mbit/s interconnects. This will require 1024x768x24 bits / 10Mbit = ~1.88 seconds to transfer one frame-buffer and this is without counting latency and transport layer overhead. This means that the usage is not quite like sitting in front of the physical machine since the user has to wait ~1.88 seconds to see the result of the interaction. RAW encoding however does not perform any manipulations of the frame-buffer. Other encoding schemes significantly lowers the amount of data for transmission by compression and techniques such as *copy-rect* where the server sends information that the client should simply copy a rectangle of data in its buffer to another location thus only transferring coordinates and not pixel information. Details on the standardized encoding schemes are available in Section 6.6 of the protocol specification[65].

4.1.2 Embedding RFB into VirtualBox

Exploring the possibilities for embedding RFB into VirtualBox requires experimentation with the source code and compiling the results. I have for this purpose put together a set of instructions to follow for building the MiG custom version of VirtualBox, they are available in A.8. VirtualBox already provide *build instructions*[66] for building the vanilla VirtualBox but the changes made in [11] to the vanilla version requires some extra prerequisites and the instructions in the appendix also provide output of the commands which are helpful when setting up and environment and trying to build VirtualBox for the first time.

Adding RFB to VirtualBox can be done by cloning the implementations of the graphics, keyboard and mouse devices made available to a the virtual machine, the code is available in the directories:

- /src/VBox/Devices/Input/*
- /src/VBox/Devices/Graphics/*
- /src/VBox/Devices/Graphics/BIOS/*

In total it is about 15.000 lines of C++ code, the entire code-base of the MiG Server is currently about 15.000 lines of Python code. Doubling the size of the code-base for adding one feature to MiG did not seem like an optimal approach I therefore searched for another way to embed remote access.

As previously mentioned then RDP support is available in the closed source edition of VirtualBox. I found traces of the RDP implementation and I found that remote access can be embedded by implementing a VirtualBox frontend frame-buffer. So instead of implementing the devices made available to the virtual machine it would be possible to extract frame-buffer data from VirtualBox frontends. I've made an attempt to so and the result of my efforts can be inspected in A.9, the code is based on the VBoxSDL frontend. When pursuing this approach some issues became quite clear.

Customizing VirtualBox in this way has the effect that it must be maintained, never versions of VirtualBox must be tracked and the code must be tested and reimplemented for each new version of VirtualBox thus requiring quite a lot of human resources to continuously provide remote access to Virtual Machines in MiG. Additionally providing frame-buffer access only via VirtualBox restricts the remote access to only be available with VirtualBox. These two situations almost match the definition of a *vendor lock-in*, an unfruitful situation especially for a middleware such as MiG that seeks to glue heterogeneous environments together.

The motivation for choosing VirtualBox is that it can provide low-level access to the virtual machines frame-buffer and thus provide remote access to the virtual machine as soon as the virtual machine is booted. This provides access to bios information, ability to change bios properties and see the entire boot of the operating system. This project however aims to transparently provide access to computing resources via a desktop environment, access to bios, boot information and system start-up is simply not something the user is interested in, all they need is a desktop environment to work in.

Based on the issues above I do not see embedding RFB into VirtuaBox as a good approach for this project.

4.1.3 Proxying

Proxies are among other things used for these four purposes:

Anonymizing traffic between endpoints to anonymize client activities.

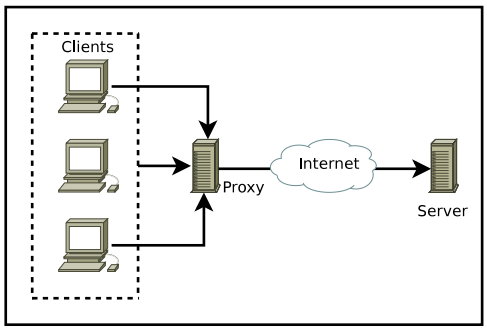
Caching requests from clients to minimize resource consumption such as bandwidth and computational power on servers.

Content-filtering responses from servers, often used in HTTP proxies to filter out malicious content, such as virus or in SMTP proxies to filter out and detect spam emails.

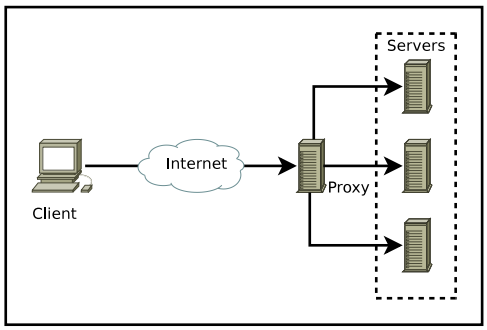
Logging requests and responses for various reasons some being monitoring others accountability of actions performed when using services.

Proxy architectures are usually designed as *forward proxies* or *reverse proxies*. Forward proxies as illustrated in Figure 9a on page 23 are used in environments where administrators wants to govern the use of external resources. A typical example are HTTP proxies where a forward proxy can be installed in an organization to provide caching of web content to reduce traffic, filter content such as pop-up adds or in other ways filter content to protect the clients.

Reverse proxies as illustrated in Figure 9b on page 23 are used to govern the access to servers, an example related to HTTP traffic is to distribute the load from clients between a cluster of servers. The naming *reverse proxy* does not refer to the use of reverse connection initiation, the traditional connection initiation is used and as the figures show these typical proxy design does not fit with the abstract architecture as illustrated in Figure 5 on page 15.



(a) Forward proxy.



(b) Reverse proxy.

Fig. 9: Proxy architectures.

When designing proxies a notion of *proxy awareness [73]* should be considered, it is repeated here for discussion of the trade-offs between the different types and their fitness for MiG. The question to be determined is at what level changes to must be made, the described levels are: application software, user procedures, router (hardware) or operating system software. The various levels are best described by examples.

Proxy-aware application software, a common feature in web-browsers are the ability to specify a HTTP proxy. Doing so lets the browser perform all HTTP requests to the proxy server instead of connecting to the server. Using this type of proxy awareness, once configured in the application software provides transparency for the user. They utilize the browser as they normally would and can't see the difference.

Proxy-aware user procedures, usefully applied when the application software is not proxy-aware. The user must then be aware of a procedure. E.g. a user uses an ftp client always connect to the same host (proxy host), but in her login credentials add user@thehostshereallywants.com then the proxy fetches this from the handshake and forwards the connection.

Proxy-aware router (hardware), can be applied for creating transparent access for both application software and users by translation of packages based on packet inspection.

Proxy-aware *operating system software*, can be approached by adding dynamically loaded libraries and hereby catching function calls and providing a proxy aware variant of the library function.

Another method not covered in [73] is to add proxy awareness to the virtualization layer by implementing proxy-aware virtualized network interfaces as a plug-in to VirtualBox. Doing this would provide a solution similar to router based awareness utilizing packet inspection and translation at the network layer. But I have already pursued VirtualBox specific solutions and uncovered the issues of a VirtualBox only solution and also in this case the issues still apply and I therefore will not pursue this approach further.

The RFB protocol does not support proxying, thus no standardized proxy protocol for RFB is available and only one (TightVNC client) of the best-fitted software stacks support proxying but not in an way compatible with the abstract architecture and only little documentation as to how it works is available. This approach could however still be used but the trade-off would be that custom software stacks must be implemented and maintained and the use of an already installed VNC client would not be possible. Using this approach should only be used if no other means of proxy-awareness are feasible. Adding awareness via the user and resource routers or via modified operating system software will be a very intrusive approach requiring access and control of routers and the users operating system. This would be a very cumbersome approach since the use of grid computing is provided for many different heterogeneous environments and to support them many different operating system libraries must be implemented and maintained.

Enabling proxy-awareness by user procedures is a more appealing approach since no software stacks must be customized and maintained, and it might be feasible if it is possible to use parts of the RFB protocols handshake like in the example of the ftp client. However doing so would not create the transparency expected for the users. Transparency can be regained by letting the MiG interface provide *connection profiles* containing the parameters needed. This approach is highly user-friendly since all the technical details of job encapsulation, protocols, client software and connection parameters are hidden from the user. Thus complete transparency for the user would be achieved.

4.1.4 Existing Proxies

Many proxies exist capable of both forward and reverse scenarios, the most widespread are based on SOCKS, HTTP, onion routing or a combination thereof and few are specific VNC proxies. Some are provided as services on the Internet others as installable software to mention a few I've collected the list below:

proxify[40], A service that anonymizes surfing habits of the user by browsing through their website.

socksify[60], A service hosting a SOCKS server.

anonymizer[1], Provides a software bundle that must be installed on the users computer and as

a service provide a set proxies with the purpose of anonymizing the surfing habits of the user.

privoxy[14], an open-source non-caching web proxy with advanced filtering capabilities for enhancing privacy. It is not provided as a service as the above but actual software is provided for installation in the environment needed.

proxifier[61], generic proxy server supporting both SOCKS and HTTPS via client installed software.

Squid[62]/mod_proxy[18]/Varnish[28], the HTTP proxy Squid is primarily used for caching web-content for users whereas Varnish is focused on load-balancing for web-servers whereas mod_proxy serves both purposes.

OpenSSH[38], client side support for SOCKS proxying and tunneling.

TOR[15], the second generation onion router is highly focused on providing a high-level of anonymization for its users and is also capable of what they call hidden services. A feature none of the other proxies are capable of.

UltraVNC-Repeater[56], is a VNC proxy server supporting both forward proxy configuration and reverse connections. But it relies on the use of the UltraVNC client and server.

VNC-Reflector[55], is a generic VNC proxy server.

VNC-Proxy[41], is despite its name not a generic VNC proxy but a specialized proxy made for a project named Chromium Renderserver that utilizes the RFB protocol in a very application specific way.

Three core issues with reusing any of these existing proxy servers are:

Architecture: most of the proxies are made for HTTP and are made for supporting either regular forward or reverse proxies since their purpose is to serve web-clients or balance access to web-servers. There is no support for managing the RFB protocol or reverse initiated connections.

Proxy-awareness: as covered in the previous section the best way to approach proxying is to find a way to support proxy awareness through user-procedures since none of the existing proxies supports this approach then the proxy must be extended to understand the user-procedures.

Implementation-language: the VNC proxies are all implemented in C or C++ this violates the MiG rule: *everything must be implemented in Python unless another language is strictly necessary.*

The issues of implementation language would not be a problem if the proxies had the exact functionality needed but there is a need for expanding the proxies to live up to the requirements established in 2.4 . The most promising proxy is the VNC-Reflector project. The VNC-Reflector is designed such that all features of the VNC server must be implemented in the proxy in order

for the client to utilize them and it does not support proxying in the architecture needed. Not much code can be reused and a lot of code must be added to obtain the functionality needed. Since the VNC-Reflector must be considerable modified then not much is gained from reusing it and it would be hard to argue against the Python only MiG rule.

4.1.5 Sockets and Asynchrony

Regular TCP based network services are implemented with standard socket libraries. When implementing a network service a choice between using blocking or non-blocking sockets must be made. Blocking sockets creates synchronous access to the service which means that the service processes one request at a time, such behavior is not attractive since only one user can be serviced at a time. Asynchrony must be obtained and there are several well-known ways of doing so, one approach is to use *threading* or *forking* by servicing each user in a thread or process of its own.

Another approach is to use non-blocking sockets and use the *select* system call to switch between processing the sockets. The advantages of the select call and non-blocking sockets is that the overhead of spawning threads and processes is removed. The trade-off is that non-blocking network services are regarded as being more complex to implement since they require the programmer to implement the switching and management of the input/output on the sockets whereas a threading or forking approach let's the operating system manage the complexity of switching.

Encapsulations of these methods are available in python via standard library and third party libraries, the most popular are:

- Threading: thread[49], threading[50], SocketServer[47] with mix-ins.
- Forking: subprocess[48], multiprocessing[44], POSH (Python Object Sharing)[45].
- Select: Twisted(3rd party)[25], asyncore[43], select[46].

The forking / subprocessing encapsulations are mainly focused on providing ways to perform cpubound computations in python, this is not easily achieved by threading because of Pythons global interpreter lock, thus various approach to solve this exist. This project however will implement a network service that is I/O bound not CPU bound no real advantages for this project is thus achieved by using a forking approach for the obtaining asynchrony. Using the low-level select module would add switching complexity to the code without providing any significant gains, asyncore removes the complexity by providing a framework for these challenges so it might be an interesting tool-set.

Using frameworks in general provides the programmer with a lot work already done and guidelines and conventions that eases the implementation process. The trade-off however is that if the framework does not exactly match the problem that needs to be solved then the conventions of the framework must be broken which results in a code-base that is even harder to maintain than a solution implemented from scratch.

A classical threading approach is still feasible and the twisted framework also shows promise. Twisted provides a big set of pre-built applications, it has on-line documentation, a printed book and a lot of commercial interest has been shown for the project. Work has been made to try and evaluate the applicability of twisted in *twisted vs threads benchmark[52]*. It seems hard to determine the applicability of twisted for this project, the framework is big and the simplicity it solves by managing select is replaced by introducing twisted terminology and concepts for implementing network services.

A classical threading approach seems sufficient to obtain asyncrony.

4.1.6 Summary

Embedding remote access into VirtualBox leads to a MiG specific customization of VirtualBox which resources are required to install in order to provide remote access to virtual machines. This is a direct violation of the MiG rules. The rules are however already broken by the customization for enabling migration from user machine to resource but this is not a justification for tightening the dependency on VirtualBox further. The design and implementation process should instead strive to find a solution that can rid MiG of the dependency.

The considerations for implementing a proxy has been covered and it has been established that no existing proxies are well-suited for providing remote access to virtual machines in MiG.

To support RFB then only the functionality standardized can be relied upon to be available. This will ensure interoperability and provide support for even the most primitive VNC clients. Improvements made by various VNC vendors should however be possible utilize when available.

4.2 First Iteration Proxy Design

With the first iteration of proxy design and implementation focus is laid upon creating a standalone proxy capable of establishing a connection between a vnc client and server. The architecture of the proxy is illustrated in Figure 10 on page 28 and the figure shows then the design relies on the software stacks ability to perform reverse connection initiation but instead of connecting to the client directly it instead connects to the proxy server. The client connects directly to the proxy server. The two issues to solve in this architecture are how to implement the management of sockets to obtain anonymization and how to *pair* client and server connections. The latter requires that the proxy must be able to make a pairing decision based on an identity provided by the client and server I considered the following three approaches.

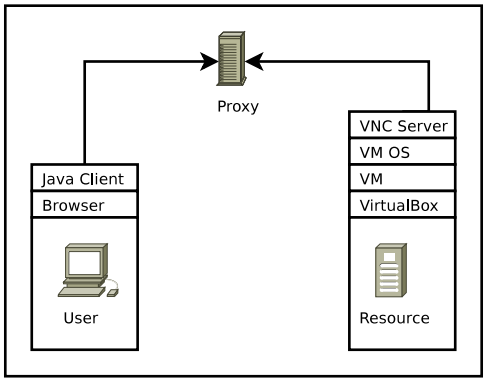


Fig. 10: First architecture.

Naively let the proxy provide a pair of ports, one port for the client another for the server and simply connect the first client with the first server that connects to the ports. Naive is a suiting label for this approach since it provides no guaranties for creating a correct match between client and server in a multi-user environment. A slight improvement to this approach would be to dynamically allocate port pairs and thus provide a short guarantee for the user that a connection to a certain port will provide a connection to the correct desktop. The trade-off is that the port pairs must be communicated to the client and server. This can be achieved for the user via the web-interface but it would require quite a lot more to automatically communicate the port to the vnc server and instruct it to initiate the outgoing connection.

port-knocking is normally used to dynamically open a port in a firewall by connection setups in a knock-sequence to a set of closed ports. This technique could be used in a another way by using the knock-sequence as the client and servers identity. This would be possible by providing a script for the VNC server parametrize reverse connection attempts with the knock sequence. The client can via the web-interface be instructed in the procedure required to connect to the proxy. The single-port port knocking technique can't be used since that would require the client to install a port knock client. The trade-off is that the the user procedure would be quite tedious for the users and the connection setup themselves would require a great deal more bandwidth than regular connection initiation.

packet-inspection could be used to extract identification information from the RFB protocol. Providing still a port for clients and another for servers such that the proxy knows the type of the incoming connection and know what data to extract.

Given that the packet inspection approach has most potential for producing a solution with least added annoyance for the user and without traffic overhead I chose to investigate the method

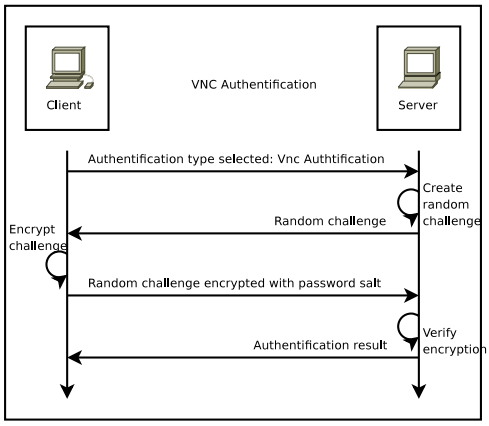
further.

4.2.1 Proxy Awareness by Packet Inspection

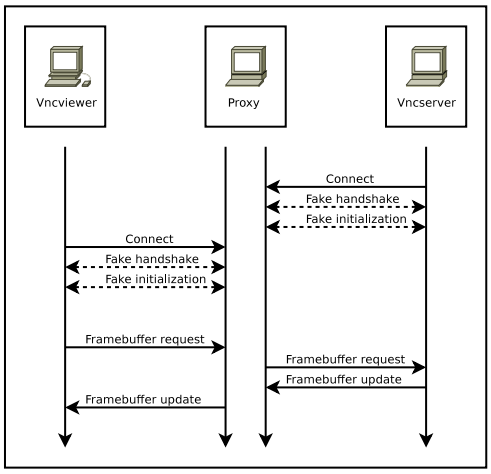
The proxy must be able to identify the client and server but at the same time ensure that neither end obtain sensitive about the other. As described in 4.1.1 then the handshake phase carries information identifying the client and the initialization carries information identifying the server.

Client-identification can be obtained by the proxy intercepting the data send in the VNC Authentication, a detailed illustration of the VNC Authentication method is illustrated in Figure 11a on page 29. The client however will not start the VNC Authentication before it has reached stage five in the handshake, to trigger the client to get to this stage the proxy must inject a fake handshake.

Server-identification can be obtained by intercepting the data send in server initialization message. Another issue is that the server won't send the server initialization message until it has reached stage eight, to trigger it the proxy needs to inject both a fake handshake and a fake initialization.



(a) VNC Authentication



(b) Proxy flow

This approach is illustrated in Figure 11b on page 29 showing how the fake initialization is also performed with the client to ensure that client and server are in the same stage. The information from the server is an almost unbounded plaintext representation of the servers hostname. The information extracted the client is a 16-byte DES encryption of the random challenge send from the proxy in the fake handshake. The encryption uses eight bytes of user-supplied input as salt for the DES encryption. The extracted client and server information can therefore not be used for pairing clients and servers without changing representation. Two approaches for changing representation are

reversing-client-identification: to obtain the user-supplied data the DES encryption needs to be reversed, since the proxy knows the challenge that has been encrypted the proxy can perform a *known-plaintext-attack*. Doing so is computationally intensive but it would

provide the eight bytes salt which is the user-supplied data. These eight bytes can then be used to match with the first eight bytes of the servers hostname.

encrypting-server-identification: since the content of the identifier is not needed a much less computationally expensive approach is to simply DES encrypt the random challenge using the first eight bytes of the servers hostname as salt. Thus pairing can be provided by matching this recently encrypted challenge with the challenge response from the client.

I've chosen the latter since it obtains the same goal but with the least computational overhead.

4.2.2 Anonymization

To maintain the anonymity in MiG the proxy must ensure anonymous operation without revealing sensitive information about the end-points. Anonymity is maintained in the proxy by only transferring application level data, the TCP/IP level is thus indirectly removed. By the methods of fake-handshake and fake-initialization the sensitive data at the application-level is also anonymized.

4.2.3 Implementation

When experimenting with the design considerations as previously mentioned then some practical issues where uncovered related to the code-base. In my first implementation of the proxy I chose to weave my own asynchronous sockets based on the threading module available in standard library. I was quite pleased with the implementation and I knew all the ins and outs of the code and had full control of everything. The downside was that I had reinvented the wheel, the SocketServer framework provided in the standard library did exactly what I had done. The SocketServer framework provides a generic yet minimalistic approach to sockets and can with a very simple change in the class definition switch the server being threading to forking. I chose to re-factor the code to use SocketServer with threading mix-ins. A lot of time can be saved by harvesting the powers of the standard library. I wish that I would have realized this much earlier on.

4.2.4 Issues

The first-iteration design discussed in this section forces sub-optimal operation because the proxy must make assumptions on the capabilities of VNC clients and servers and handle state bookkeeping.

This is due to the fake initialization performed to obtain proxy awareness and pairing connections. At the time where the proxy performs the fake handshake with the client it does not yet know the counterpart of the connection it therefore cannot send the correct set of supported encodings to the client, nor can it inform the client of the correct resolution of the remote display. It therefore has to make an assumption of the supported features, the proxy can either choose to

only announce the encoding schemes defined in the RFB specification this would ensure safe operation since it could be expected that a VNC server and client implements the encoding schemes as defined in the specification. The specification however states that only the RAW encoding is sufficient to implement for a client to label itself as standard compliant, thus RAW effectively becomes the lowest common denominator and to ensure correct operation then the proxy should only announce RAW encoding in the fake initialization this would impose great bandwidth requirements as previously described. Alternatively the proxy could announce a set of encodings equal to the set of encodings that all clients in the list of best-fitted software supports.

Another issue presents itself regarding the state of the client and server connections. The situation illustrated in Figure 11b on page 29 presents the best case: the server connects to the proxy before the client. This situation is the best since the proxy then knows the resolution of the frame-buffer and the capabilities of the server but in practice then the client could easily be the first to connect to the server. If the client does then the proxy would not know which encodings to announce to the client and the assumptions above must be used another problem is that it does not have a VNC server to pass frame-buffer request to. A solution would be to simply drop client connections if no matching server exist or the proxy could implement a fake frame-buffer that it can send to the client.

Another issue regarding connection state is that the implementations of reverse connection initiation in x11vnc and TightVNC, which are the only of the best-fitted VNC servers that support reverse connections, is that they only expect to handle one client per connection and their internal state relies on this, this means that the proxy must maintain the state bookkeeping in case of connection failure from the client. If the client disconnects while receiving a FrameBufferUpdate then the proxy must handle this properly. Multiple scenarios leads to connection state errors which must be handled by the proxy.

An alternate solution instead of basing the implementation on assumptions would be to implement message translation in the proxy for example if a client only supports RAW encoding but the server supports a more efficient encoding such as ZRLE (see Section 6.6.5 in [65]) then the proxy could transform the representation of the frame-buffer when it receives the frame-buffer update from the server from ZRLE to RAW. And send the transformed frame-buffer update to the client. The trade-off is added complexity to the proxy and higher requirements to the proxies computing resources.

The bottom-line is that the implementation of the design consideration at this stage requires state bookkeeping and using sub-optimal assumptions of the client and server capabilities and high complexity in the proxy logic, an implementation based on these findings can be used but in the second iteration design I will seek to improve the design by minimizing complexity.

4.3 Second Iteration Proxy Design

The previous section discussed some design issues which could be solved but with a trade-off of added complexity to the proxy this section discusses an approach that solves the design issues without adding complexity to the proxy.

Lets address these issues by doing like the coyote and the roadrunner and going back to the drawing board. The main sources of the issues are listed below and reevaluated.

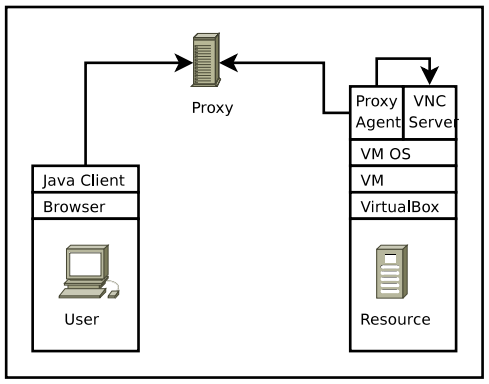


Fig. 11: Second architecture

Fake-initialization is needed for adding proxy awareness by user procedures and extracting the identification with packet inspection. It was established that the identification of the client could be provided by inspecting the handshake but fake initialization was added to provide identification of the server. If another method for identifying the server could be provided then fake-initialization could be removed.

Reverse-connection-initiation Reverse connection initiation from the server is needed since ingoing access to the Virtual Machine is not available. But could reverse connection initiation be provided by other means than relying on the servers ability for doing so?

The issues with the previous design lies within the servers identification and connection initiation. The server software is installed in an environment controlled by MiG, the virtual machine is made available by MiG and MiG controls the available software and configuration of the virtual machines operating system. An approach could be to install a piece of MiG software in the virtual machine that provides server identification and provides the connection setup needed, this can be done without violating any MiG rules or design criteria.

This leads me to the second architecture as illustrated in Figure 11 on page 32. The architecture is expanded to include a *proxy agent* the proxy agent takes care of the issues of creating reverse connections to the proxy and thus removing this requirement from the VNC server.

4.3.1 Proxy Agent and Protocol

The proxy agent must be able to communicate with the proxy to inform the proxy of the identity of the server that it is proxy-enabling and to establish connections to the server on demand. This is done by three messages *handshake*, *setup request* and *setup response*. Described below.

Handshake Send from the proxy agent to proxy after connection initiation to start a control connection and establish identification of the proxy agent. After the handshake the proxy can then send multiple setup request messages.

Setup-request Send from the proxy to the proxy agent over the control connection to instruct the proxy agent to establish a connection.

Setup-response After the connection setup has been performed by the proxy agent it sends a status message back to the proxy server over the control connection to communicate success or failure of the setup-request.

The above messages are encapsulated in a protocol that I have chosen to call *Minimum Intrusion Proxy (MiP)* protocol. MiP is a simple byte-oriented protocol, the byte-sequences representing the above messages are documented in A.5. The functionality of this design is best described by providing an illustration of the flow when handling the RFB connections such is provided in Figure 12 on page 33.

As the figure shows then the fake initialization is removed, reverse connection initiation is handled by the proxy agent as well as informing the proxy of the servers identity via the proxy agent handshake. The only state to handle for the proxy is when the client connects before the proxy agent has performed the identification handshake. Choosing to maintain the connection with the client before the proxy agent handshake would require that fake initialization must be used to avoid the client to timeout the effort of a proxy agent would thus be useless. Even if a timeout is not implemented in the client then maintaining a connection with the client could result in the client waiting forever if the user supplied identification is invalid since no proxy agent would ever connect and match the identification. It is thus a sound choice to simply close the connection with the client if no the corresponding proxy agent has not yet connected.

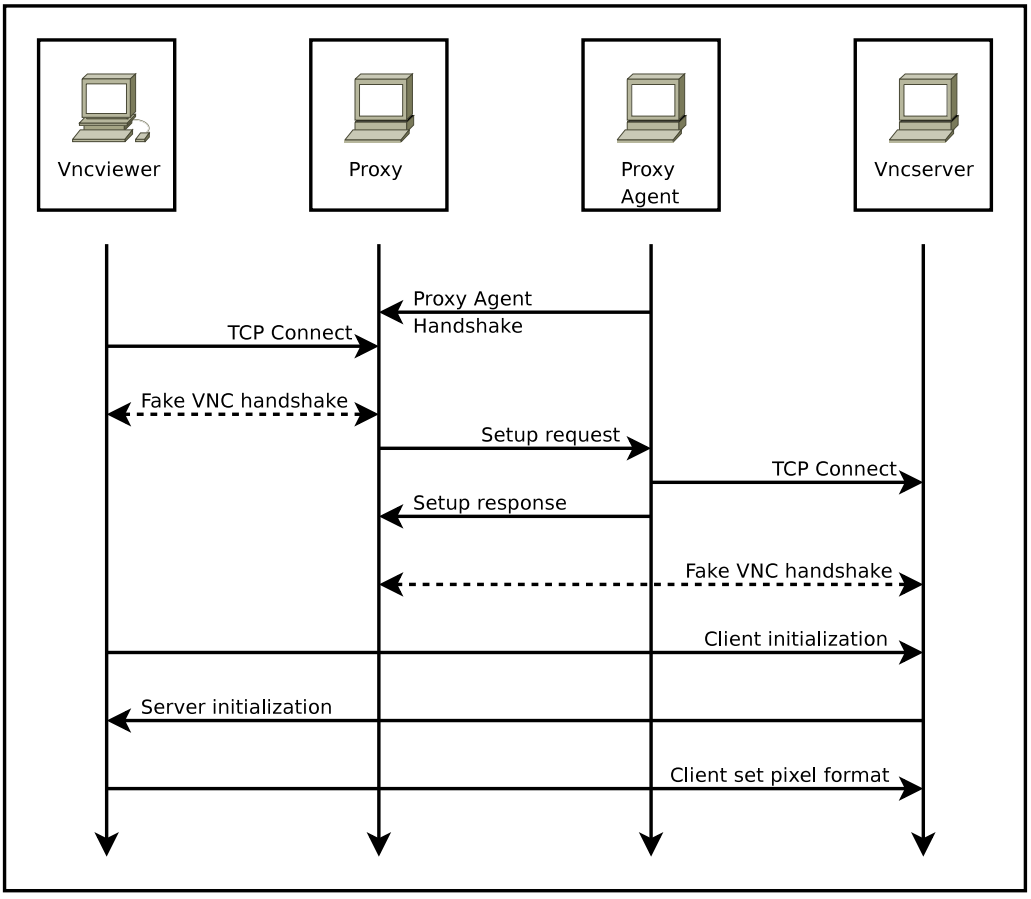


Fig. 12: Connection flow with proxy agent.

Having definined a way to achieve identification then proxy awareness will be revisited to discuss the content of the identifiers.

4.3.2 Proxy Awareness Revisited

The hostname of the VNC server has so far been used as content for the identifier. The use of hostnames for identification is an issue. Not with anonymity as previously considered, even though it seems like revealing the hostname of the VNC server would pose an issue. But the fact is that the hostname is not the hostname of the resource but the hostname of the virtual machine. Virtual machine hostnames are simple names such as *mig_scilab*, *mig_vanilla* and the user already knows the identity of the virtual machine that they are accessing. The issue regards uniqueness of the identifier, when multiple users are using *mig_scilab* as content for the identifier then collisions will occur as soon as more than one user wants to remote access the remote desktop environment. Thus an identifier with higher uniqueness must be obtained.

It would be ideal if a shared secret between the client and server could be established, thus would pairing connections be a trivial task of matching identifiers that would provide accurate access to the desktop environment without collisions. A shared secret is already in use in MiG; the job-identifier. Only the user submitting the job knows the unique job-identifier and since the migration of virtual machines to MiG is implemented by encapsulating the migration as a job then a one-to-one mapping exist between job-identifier and virtual machine/desktop environment. The job-identifier is already made available to the user via the web-interface, the challenge is to communicate the job identifier to the proxy agent since the virtual machine has no knowledge of the job that it is encapsulated in. Another challenge is that job-identifiers are strings such as *354887_1_20_2009__18_4_26_mig-1.imada.sdu.dk.0* far greater than eight bytes in size. To use the job-identifier two problems must be solved: communicating the job-identifier to the proxy agent, transforming the job-identifier into a representation of eight bytes in size that the user can input into the VNC client.

representation: transforming the representation involves taking a job-identifier as input and producing an eight byte user-input-able output. In this context user-input-able means an eight byte string with ASCII[2] representation. An issue to take into consideration is that ASCII contains 256 chars but only 94 of them are actually user-input-able. The transformation must thus map into a subset of the ASCII table. The function for performing the transformation is listed in Figure 13 on page 35 and an example of the tranformation is provided in Figure 14 on page 35. The transformation will provide 94^8 unique identifiers.

shared-secret-communication: can be achieved by utilizing features available in MiG and VirtualBox. VirtualBox provides a means for communication between hosts and guests, these are called *guest properties.* MiG provides a meta attribute in job descriptions that lets it utilize job identifers as parameters for commands in the job description. Combining these two features communicates the job-identifier to the guest operating system the encapsulation can be inspected in A.7.1. The identifer is then extracted inside the guest-operating system by the system user and passed as a parameter for the proxy-agent.

The above methods provides a feasible solution for providing shared-secrets with a much better unique-ness than using the hostname of the guest operating system.

```
def transform_identifier(job_id='Unknown'):
  job_id_digest = md5.new(job_id).hexdigest()[:16]  # eight byte md5 sum
  identifier = ''
  for i in range(0, len(job_id_digest), 2):
    char = int(job_id_digest[i:i+2], 16)            # get "hex-char"
    char_domain = char % 94                         # map to sub-set
      identifier += chr(32 + char_domain)           # shift into range
  return identifier
```

Fig. 13: Tranformation function in Python.

```
transform_identifier('354887_1_20_2009__18_4_26_mig-1.imada.sdu.dk.0')
-->
^HbD)GJo
```

Fig. 14: Example of transformation.

4.4 Third Iteration Proxy Design

So far an architecture has been designed and implemented that can provide remote access to virtual machines in MiG, but there are still requirements left to be solved, those are to provide firewall compliance and providing client software without intrusively installing it on the users machine. These requirements are discussed in each of their own subsections.

4.4.1 Firewall Compliance

To obtain firewall compliance then the link between the proxy and the proxy agent must comply to the firewall requirements specified in Section 2.4. Obtaining firewall compliance can be done by either encapsulating RFB messages in HTTP request/response messages, using the *HTTP connect*[21], *TLS/SSL over HTTP*[22] or *SSH tunneling*[20]. SSH tunneling is a bit cumbersome to setup and maintain since it requires the creation and existence of a user on the proxy server for the purpose of tunneling, the SSH daemon must be configured to lock the user down so he cant use the tunneling features for anything other than localhost and also restrict the usually available shell features of SSH. I have therefore pursued the HTTP-based approach since it does not pose any host requirements such as those just described.

HTTP-encapsulation can be achieved by embedding RFB protocol messages into the body of a POST request or the header of a GET request. Responses can then be encapsulated in HTTP responses. An example of this encapsulation approach is given in Figure 24 on page 57. The trade-off for this approach is that significant overhead in message sizes is introduced and encapsulation/de-encapsulation must be performed for each request.

HTTP-connect is a very lightweight approach to punching holes in firewall policies, the details of how the HTTP connect works is described in [29]. The essence is illustrated in Figure 25 on page 57. It is a very simple way to squeeze any type of traffic through a firewall that allows HTTP. And in relation to HTTP-encapsulation then it only requires encapsulation in the connection setup and not in each each request/response.

HTTPS is also called *HTTP over TLS/SSL*. HTTPS works by the client initiating a connection the server, then a TLS/SSL handshake starts and a *secure layer* is established encrypting HTTP messages send *over* TLS/SSL. HTTPS can be used to provide firewall compliance to RFB by implementing *MiP/RFB over TLS/SSL* this is possible since the firewall can't inspect the packages and identify whether the packages are HTTP, MiP or RFB since the application level packets are encrypted. Very complex firewalls might be able to identity the application level protocol by analyzing traffic patterns.

Of the above mentioned methods then utilizing TLS is the strongest option since it provides a means for encrypting the transmitted data and as mentioned in 2.2.2 then no standardized way of obtaining encryption is available in the RFB specification. The introduction of a proxy agent shows significant advantages at this point since all the non-standardized features as previously

mentioned can be added to the proxy agent and thus to the middleware and hereby providing the non-standardized features to any VNC server.

4.4.2 Sockets and Asynchrony Revisited

Adding TLS to the proxy and proxy agent requires reconsideration of some implementation choices regarding sockets and asyncrony. A library should be chosen to manage the complexity of TLS since nothing is gained by implementing the SSL/TLS from scratch.

From python 2.6.2 SSL is part of the standard library the version of python available to MiG however depends on the version available in base system of Debian stable, currently that is python 2.5. Therefore a 3rd party library must be used. The *pyOpenSSL[42]* library is available on the MiG server, other alternatives include but are not limited to: *M2Crypto[53]*, *ChilKat[10]* and *TLSLite[51]*. I've chosen to use *pyOpenSSL* simply because it was the best SSL implementation available in Python.

When covering sockets and asyncrony I described the advantages of the SocketServer framework. However not all issues can be solved with SocketServer and a threaded approach, in the core of the proxy is a module named Plumber whose purpose is to tunnel traffic between two sockets. This is the part of the proxy that anonymizes the network layer, by only copying the application level data between the two sockets. The plumber performed a blocking read and in another thread a concurrent write to the same socket. This is not a problem for regular sockets as they are thread-safe, however when adding encryption via pyOpenSSL or more specifically OpenSSL then the thread-safety is lost. OpenSSL does not support a blocking read and a concurrent write to the same socket, trying to do so results in a "PyEval_RestoreThread: NULL tstate". I learned this the hard way.

Thus TLS wrapped sockets must be accessed sequentially, this is not easily done with blocking sockets, since trying to govern a blocking read with a lock will result in a deadlock.

This is where *select* shines, a select call can check if an I/O stream has data in its buffer. This property can be used to avoid the deadlock by checking with a select call whether there is data in the buffer and only take the lock in case there is. This approach is feasible but adds a significant overhead of busy select calls and locking.

I instead chose to rewrite the core of the Plumber by switching the sockets dynamically from blocking to non-blocking and replace it with select based I/O handling. I thus achieved to provide safe access to the TLS based socket without excessive use of threading and locking. The complexity of the I/O handling is reduced to a very small core area so the simplicity of the code-base is not sacrificed.

4.4.3 Client Software

As previously mentioned then installing software on the users machine is considered intrusive and a violation of the MiG rules. A minimally intrusive approach is thus to facilitate the use

of a Java-based client. Java applications can be distributed by either using Java web-start or embedding the Java application into the browser with Java applets both methods require the installation of a Java run-time environment.

Java-Web-Start encapsulates the application into a self-contained sandbox environment with the needed requirements bundled into the web-start environment. Java Web-start is however mainly focused on *deployment* of the application via a browser and not to run inside the browser window.

Java-Applets are *embedded* into the browser by inserting the <applet> tag into the HTML document, the Java application is then running inside the browser and can thus be integrated into the existing web-interface.

Java-Web-start has the advantage that it provides a less restrictive environment than an applet based approach. Java-web-start is as mentioned not integrated into the browser and it is thus harder to implement a seamless integration of a web-start based application. I have therefore chosen to use a Java-applet, the restriction to comply with is that the run-time environment per default is configured to only allow outgoing network access to the server that the applet is downloaded from. This forces the proxy and webserver to be available to the client from the same IP address, this causes a conflict since the firewall compliance as previously described required the use of port 443. One solution to this problem is to change the configuration of the run-time environment for the user this however is not a very transparent approach and users are forced to maintain a list of proxy servers that they are allowed to connect to. Instead of imposing this burden on the users I have chosen to implement a simple applet-server in the proxy, a minimalistic webserver with the sole purpose of serving Java clients to its users. This also has the added value that the proxy can always provide a compatible VNC client for the user.

4.4.4 Summary

This covers the design and implementation of expanding the backend of the middleware that lets MiG capable of providing *remote access* to virtual machines in MiG which is the primary objective of this project. The key enablers for the solution are:

proxy the expansion of the MiG backend with a proxy capable of performing packet inspection and anonymizing users and resources.

proxy-awareness by user procedures and utilization of packet inspection obtained proxy support for even the most primitive VNC client.

proxy-agent maintained firewall compliance by examining the features available in the HTTP and HTTPS protocols and hereby adding transport layer security between the proxy and proxy agent and thus enable identification and secure communication for even the most primitive VNC servers without removing capabilities for the most feature-rich.

shared-secret identification without revealing the identity of users or resources by utilizing job identifiers and transforming them into a user input-able representation.

During the process it has been discovered that to successfully provide remote access to virtual machines in MiG then the existing work of migrating machines must be improved by removing the dependency of a customized VirtualBox to successfully let the complete solution comply with the MiG rules and design criteria. The migration of virtual machines also depends on the availability of VirtualBox on the users machine this requirement causes a conflict with the MiG rules and therefore must the management and migration of virtual machines functionality be provided via the web-interface in MiG.

4.5 Virtual Machines in MiG

In this section I address some design decisions in the original work on virtual machines in MiG.

Hypervisor-dependency: The choice of using VirtualBox as the virtualization tool based on a requirement that the source code should be freely available such that the virtualization tool could be modified to integrate it with MiG. I've implemented a solution that removes this requirement by encapsulating the MiG integration into the job description utilizing the features made available by the virtualization tool instead of modifying it. The solution however has a trade-off in relation to the original work.

Transfer-time: Issues were raised with the transfer times of virtual machines, the virtual machine must be transferred from the user to MiG and from MiG to a resource. I introduce a solution that effectively lowers the transfer times by simply removing the transfer from the users machine to MiG in compliance with the MiG rules and design criteria of fat grid middleware and slim clients.

Operating-system: Regular installation of an arbitrary Linux distribution was discarded since tests had shown that it was too cumbersome to implement virtual machines based on regular installation of the guest operating system thus the ISO based slax distribution was chosen. I provide a solution that removes the dependency on the slax distribution by introducing virtual machine builders which provides a simple approach to regular operating system installation.

4.5.1 Hypervisor Dependency and Migration Issues

The original work of virtual machines in MiG depends on a customized version of VirtualBox, the customizations are made to VirtualBox to be able to integrate VirtualBox with MiG. The biggest challenge was to maintain the state of execution of the virtual machine when the machine is migrated from the users machine to the resource. VirtualBox does not support migration and when trying to resume a saved state of a virtual machine on another host with a different CPU then VirtualBox complains that the CPU is different. Thus to provide migration the CPUID check where removed from VirtualBox.

I discovered another issue with the migration, it is not possible to change the amount of memory available to the virtual machine when migrating, trying to do so will result in a VERR_SSM_LOAD_MEMORY_SIZE_MISMATCH error from VirtualBox. The reason for this error is quite natural but it does however describe another issue with the migration approach to providing remote virtual machines in MiG.

To provide migration of virtual machines from the users machine to a resource in MiG, then the following compromises must be made:

- No CPU extensions can be used, due to the removed CPUID check.
- Only homogeneous migrating since migrating from 32bit hosts to 64bit hosts is not possible.

- Only 32bit guests are supported.
- More memory cannot be provided to the virtual machine.

A general motivation for using grid computing is that a user can get access to more computing resources that those available locally. The issue with the above compromises is that the only resource advantage is that access to a higher clock frequency can be provided. The same memory and architecture as the one available locally is provided. Thus the main advantage of the current virtualization approach is concerned with the usability of MiG.

The compromises mentioned above can be avoided and the dependency of a customized VirtualBox can be removed if the execution state of the Virtual Machine does not need to be migrated from the user to the resource. If the virtual machine is not migrated from the user another dependency of installing a MiG specific software on the users machine can also be removed. Thus removing VirtualBox from the users machine seems like a good choice, how virtual machines are managed without VirtualBox installed on the users machine is covered in the next section.

4.5.2 Transfer Times and Virtual Machine Management

Without VirtualBox on the users machine then the functionality for starting and creating virtual machines must be made available via the web-interface and storage of the virtual machines must be provided outside of the users machine.

This is implemented by adding a directory for virtual machines to the users homedir with the name *vms*. Inside all files related to a virtual machine is stored in a directory with the name of of the virtual machine. The layout is exemplified in Figure 15 on page 41. An option is provided for the file-extension of the system disk, the system disk extension can be either *.vmdk* or *.remote*. If the extension is *.vmdk* then the system disk is physically available in the homedir, if the extension is *.remote* then the system disk is expected to be available on the resource. An example of the storage of system disks on resources is illustrated in Figure 16 on page 41.

```
~/vms/runvm.sh
~/vms/MachineName/
~/vms/MachineName/machine.cfg
~/vms/MachineName/data.vmdk
~/vms/MachineName/sys_systemname.[vmdk|remote]
```

Fig. 15: Layout of users virtual machine storage.

```
~/vbox_images/plain.vmdk
~/vbox_images/scilab.vmdk
```

Fig. 16: Layout of resources image storage.

By providing the choice of the system disk being available on the MiG server or on the resource provides a means for evaluating the trade-off of waiting for transferring the system disk for different use-cases.

The job encapsulation with system disk on the MiG server is available in appendix A.7.1 and for system disk on the resource is available in appendix A.7.2.

4.5.3 Operating System Dependency / Introducing Virtual Machine Builders

Slax has been chosen since it provides an easy separation of system and user data, the trade-off with using Slax is that in relation to other Linux distributions such as SUSE, Fedora, Debian and Ubuntu then not much software is available and the installation of 3rd party software such as MATLAB is quite cumbersome.

I wanted to provide an approach as easy as using slax but with the rich availability of software that other Linux distributions provide. First step is to find a way to separate application and system files. Separation of system and application files from user data is already accomplished by the fact that most Linux distributions follow the suggestions part of the *File system Hierarchy Standard (FHS)*, the only location for storing user data in a FHS compliant Linux distribution is in /home. By mounting a static disk for "/" and a dynamic disk for "/home" then the separation of data and system is accomplished. However automatically installing operating systems require more work.

Installing virtual machines is part of a hot topic of virtual machine management. A recent trend within vitual machine management is the development of the *Open Virtualization Format (OVF)*[67]. An open standard seeking to provide mobility of a virtual machine between different hypervisors and a standardization of the virtual machine definition. The standard shows great promise and a tool exist for utilizing the standard it does however not yet support installation of guest operating system in the virtual machine.

Tools however do exists for doing what the OVF standard seeks to standardize. Two such tools are *python-vm-builder (VMBuilder)[9]* and the *virtualization api (libvirt)*[26]. In relation to OVF then these builders work by building a machine to a specific hypervisor in the hypervisors disk format and machine definition. This does not provide mobility but VMBuilder has the strong feature of being able to build a virtual machine with a guest operating system installed. On modern computers the process takes about 10 minutes to complete.

I've chosen to use VMBuilder since it is implemented entirely in python and is therefore well-suited for integration with MiG. However when more features are added to the ovftool then it might prove a better solution. The technical details of how python-vm-builder has been integrated with MiG is documented in appendix A.6.

4.6 Interface

Designing and implementing the user interface is split in three parts the first dealing with general usability issues, secondly with providing an interface for the remote access to virtual machines and third providing an area for managing virtual machines.

4.6.1 Usability Enhancements

The topic of usability of the web-interface is a area of research in its own in the field *human computer interaction (HCI)*. Most of the work done in MiG have been done by research in grid computing, distributed systems and high performance computing. The design of the web-interface is thus not developed with the focus of research in HCI.

Since the motivation for providing users with a desktop environment was that it should improve the usability of MiG then I've approached the design and implementation by using HCI techniques such as user driven design, direct manipulation.

To utilize user driven design an informal survey was conducted of the participants of a course held at DIKU on cluster computing in 2008 that used MiG and a couple of my own observations. The main issues reported by the users are listed below.

Centering: All text is centered including headers, for most users this is not optimal and decreases readability since they are used to read from left to right and top to bottom creates uneven lines that are not easily processed.

Space-utilization: As screen-shot shows can see in Figure 17 on page 44 then a lot of space is used for showing the MiG logo and a navigation menu and thus pushing the content below the bottom of the page.

Buttons: The navigation menu is based on text-only it is not easy to identify and recognize the various subsystems available.

Colors: The colors used are mostly gray in gray with little contrast.

Based on this input I implemented a prototype that I presented to the MiG developers the list of potential improvements was increased and improvements to the graphical design was implemented. The above mentioned issues where solved by left-justification of text, utilizing unused horizontal space by moving the navigation menu from to top to the left-hand side and minimizing the MiG logo and thus providing more screen real-estate. Brighter colors was added to the interface providing more contrast and providing a sharper look. Buttons where introduced with a feature of *stickyness*; the color change applied when hovering the button was made fixed when pressing the button, this feature made it easier to identify the subsystem of the interface that the user is currently using. Figure 18 on page 44.

Fig. 17: Original interface, downloads area.

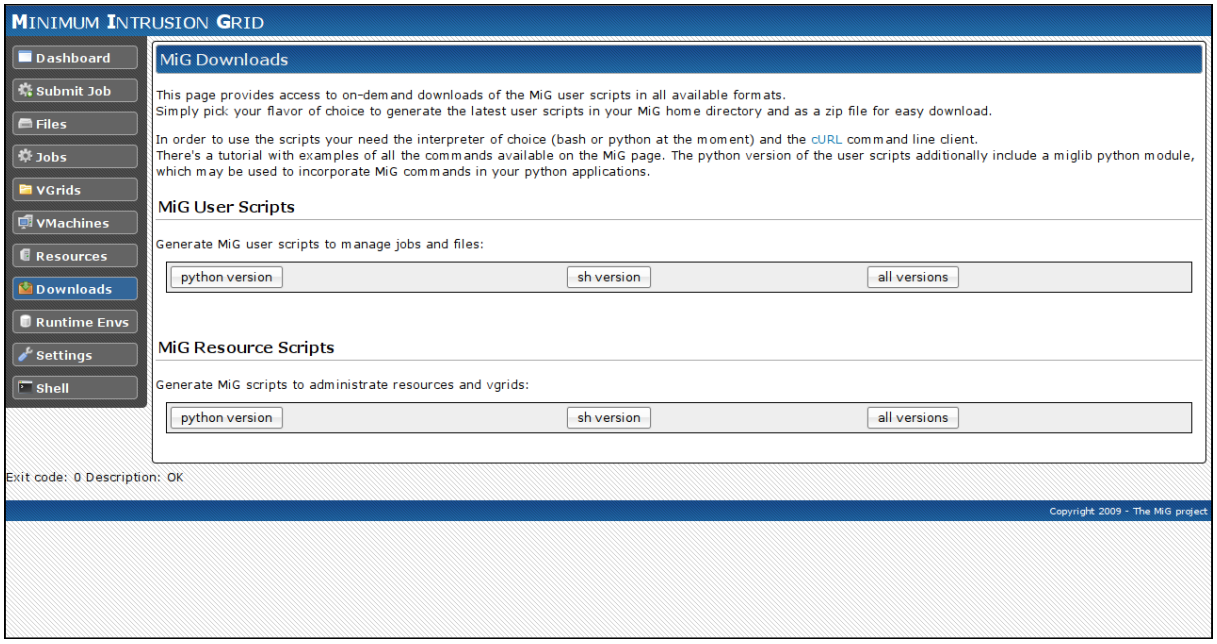


Fig. 18: Improved interface, downloads section.

4.6.2 Remote Access

Choosing the desktop environment metaphor as a means for interaction provides a familiar environment for the user to operate in, providing remote access to the desktop environment must thus support this familiar interaction. Desktop environments in a non-remote context are visually communicated to the user via the physical monitor that they are sitting in-front. Thus to support familiarity then a *virtual monitor* is introduced. The state of virtual machine and the

availability of the desktop environment is visually provided by informing the user via the *virtual monitor* thus all the backend details of job encapsulation is hidden from the user.

When the user clicks on the virtual monitor the state is changed to *booting* and the user is instructed to wait, when the desktop is ready for interaction the *virtual monitor* changes state to running and informs the user that she can "click to connect". This interaction supports the HCI technique of *direct manipulation*. The virtual machine overview interface can be seen in 19 the state changes can be seen in 20 and when actual remote access to the virtual machine is achieved can be seen in 21.

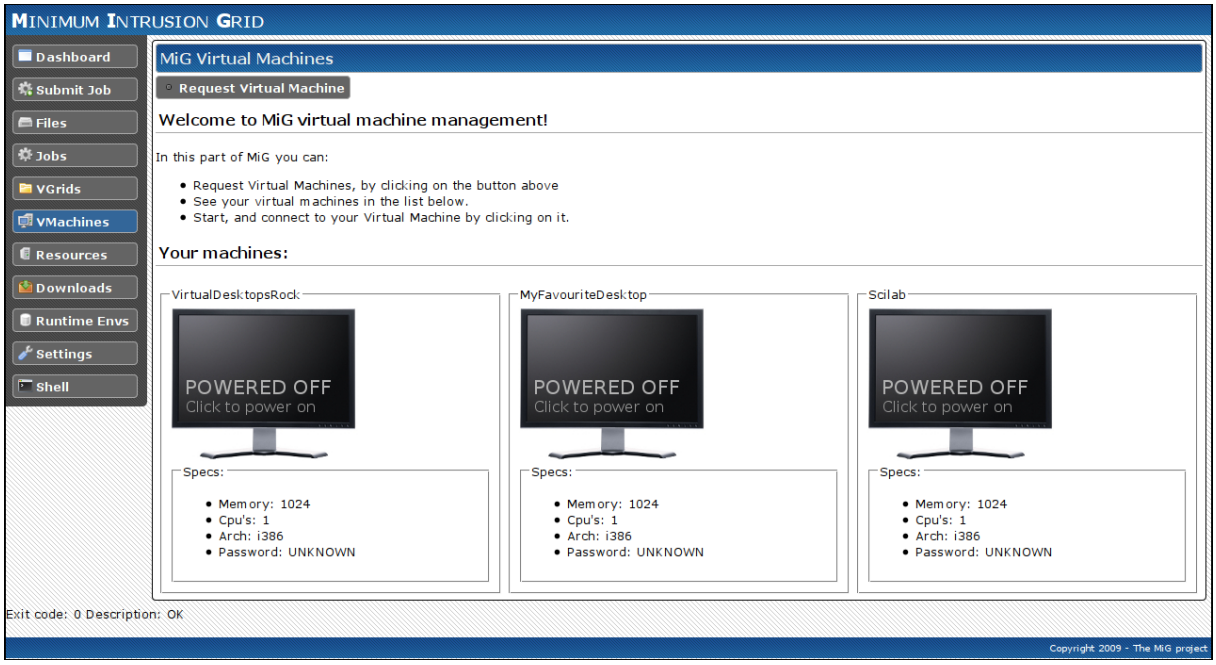


Fig. 19: Virtual Machines area of web-interface.

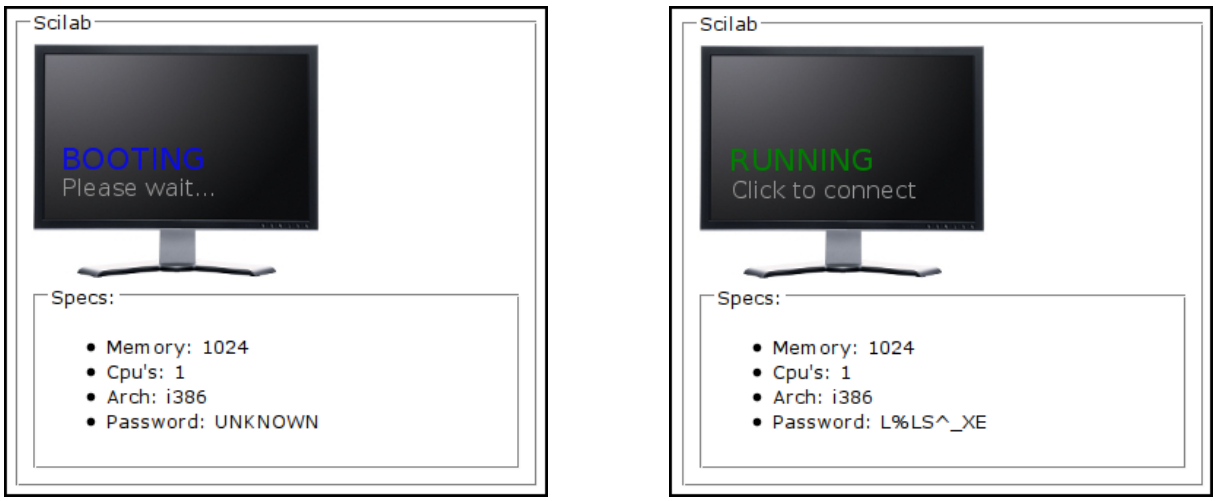


Fig. 20: Machine state change.

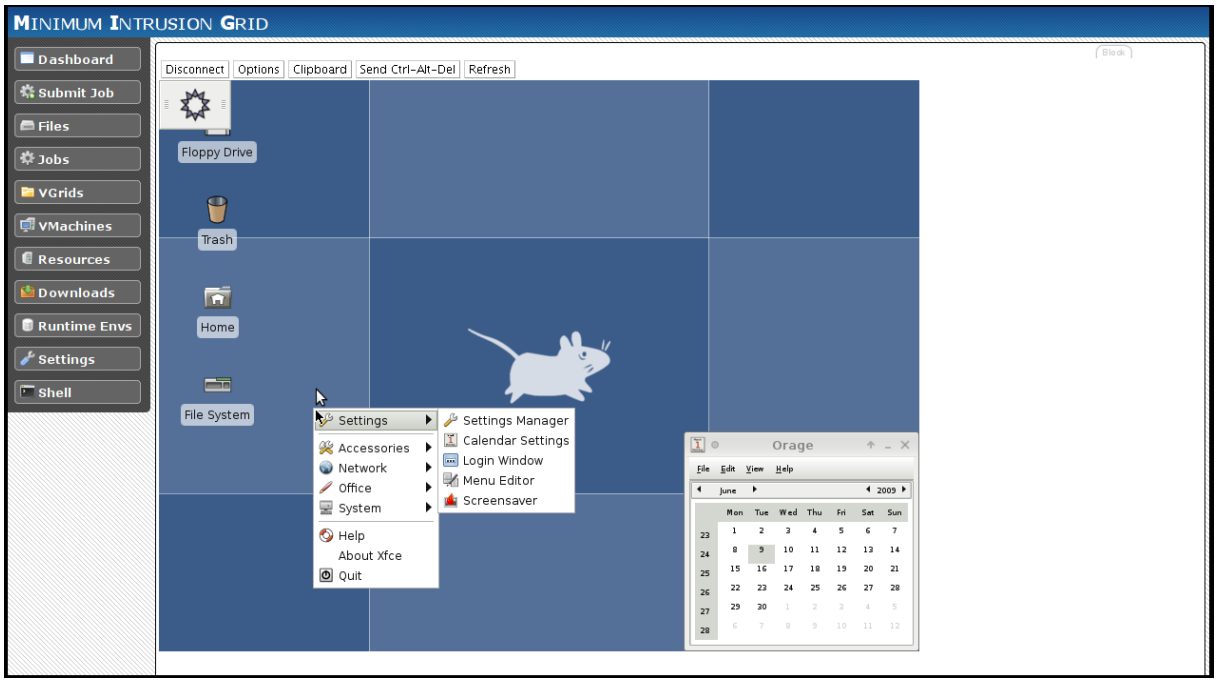


Fig. 21: Connected to remote desktop environment.

4.6.3 Request Virtual Machine

Functionality for creating virtual machines is provided by filling out a formula and either choosing a pre-built machine or specifying the machine properties and hereby request a personal virtual machine. However the implementation of personal virtual machines is left for future work. The virtual machine request formula can be seen in 22.

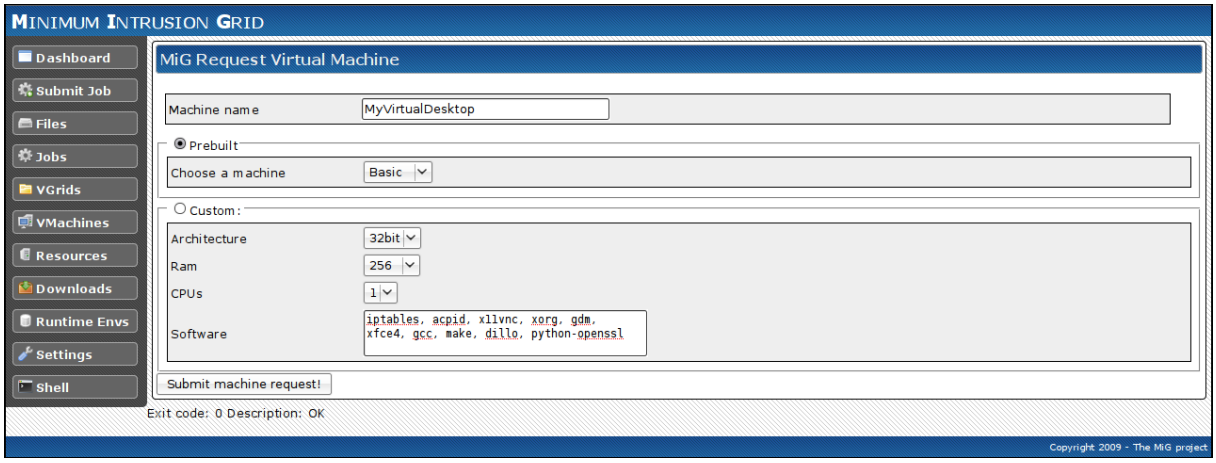


Fig. 22: Request Virtual Machine Formula.

5 Test

I have performed a series of tests to verify the correct functionality of the proxy, proxy agent, the web-interface interaction and the creation of virtual machines. The tests where performed with a MiG development server, a resource and a laptop.

NR.	Description	Parameters
1	Virtual machine listing	With no machines available
2	Virtual machine listing	With one machine available
3	Virtual machine listing	With more than one machine available
4	Virtual machine state	When deploying
5	Virtual machine state	When when booting operating system
6	Virtual machine state	When displaying desktop environment
7	Virtual machine state	When shutting down operating system
10	Request a virtual machine	Machine does not exist
11	Request a virtual machine	Machine with same name already exist
20	Get user input-able identifier	
30	Deploy virtual machine	With system disk on resource
31	Deploy virtual machine	With system disk from MiG server
32	Deploy virtual machine	With changes to data disk
40	Connect to RDE	With Java client and connection profile
41	Connect to RDE	With external client and valid identifier
42	Connect to RDE	With external client and invalid identifier
51	Connect to RDE	With both external and Java client
52	Connect to RDE	Disconnect then reconnect with Java client
53	Connect to RDE	Disconnect then reconnect with external client
54	Connect to RDE	Disconnect then reconnect with Java and external client
61	Connect to RDE	With network failure in virtual machine
62	Connect to RDE	With network failure in connection on MiG server

5.1 Observations

When testing the final solution model it was discovered that test 61 failed, some exceptions where not handled correctly in the proxy agent which caused it to exit ungracefully instead of trying to reconnect to the proxy. The bug was corrected and the current solution model works correctly to the extend that it was tested.

6 Future Work

proxy: means for locking down the use of the proxy should added, this can be achieved by enhancing the white-listing in the MiGTCPServer such that only connections from trusted peers are allowed. Additionally should functionality be added for identifier verification to prevent unauthorized use of the proxy with fake matching identifiers.

virtual-machines: the OVF standard promises to provide solutions for mobility of virtual machines between hypervisors with free mobility of virtual machines then MiG can much easier utilize heterogeneous resources with different hypervisors. It can provide a firm ground for implementing personal virtual machines.

Interface: the *virtual monitor* can be expanded to provide screen-shots of the desktop environment directly in the *virtual monitor,* doing so would provide instant status of the desktop environment without connecting directly to the virtual machine. Such a feature can prove useful for users who utilize the remote desktop environment for providing visual output from simulations but who has no need for interacting with the simulation.

Perspectives for using the work in this thesis is to utilize the flexibility of the proxy design. The only thing restricting the use of the proxy to the RFB protocol is the implementations of fake-handshake handling. This part of the proxy code-base can be expanded with handlers for other protocols such as HTTP, SMTP and SSH. Such an addition of handlers would provide MiG with remote access to web-servers and interaction with SSH. The complexity of the remote access will depend on the complexity of the protocol.

7 Conclusion

The initial goals of the project was to examine the requirements of providing remote access and whether it was possible to implement a model that satisfies these requirements in MiG. Remote access to Virtual Machines in MiG has been achieved and additionally the utilization of Virtual Machines in MiG has been substantially improved by:

- Removing the dependency of a MiG specific customization of VirtualBox.
- The choice of hypervisor for virtual machines in MiG is no longer restricted to VirtualBox, the broad suite of KVM, QEMU, Xen, VMWare Workstation, VMWare ESXi can also be used.

Usability improvements of the web-interface has been designed and implemented with success. I consider the changes a success since they have already been integrated into the stable release of MiG and put into production.

Remote access has been implemented with a secure and fault tolerant middle-tier that does not require installation of any additional software on the user or resource end. This sounds simple but as the documentation of the design and implementation process shows many obstacles and strict requirements had to be overcome. Existing solutions for proxying could not be reused because of the strict rules and design criteria of MiG.

By obeying the rules and design criteria it was revealed that the management of virtual machines could and should be removed from the user end. This was accomplished. To successfully provide the utilization of virtual machines then an interface for this purpose had to be implemented with focus on usability that was unseen in MiG. This was accomplished.

The section on future work describes the potential for the utilization of the proxy and proxy agent. Hopefully the work in this project will be expanded on to provide remote interaction with secure shell and other protocols. For now basic Virtual Desktop Computing in the Minimum Intrusion Grid has been achieved.

List of Figures

List of Tables

References

[1] Inc. Anonymizer. Anonymizer. `http://anonymizer.com/`, 02 2009. Visited 18. Feb 2009.

[2] asciitable.com. Ascii table. http://www.asciitable.com/, 18 February 2009. Visited 18. Feb 2009.

[3] Jonas Bardino. Getting started guide with MiG. http://code.google.com/p/migrid/wiki/GettingStarted, 18 February 2009. Visited 18. Feb 2009.

[4] Jonas Bardino. MiG Interfaces. http://code.google.com/p/migrid/wiki/MiGInterfaces, 18 February 2009. Visited 18. Feb 2009.

[5] Jonas Bardino. MiG Job Flow. http://code.google.com/p/migrid/wiki/MiGJobFlow, 18 February 2009. Visited 18. Feb 2009.

[6] Jonas Bardino. MiG Rules. http://code.google.com/p/migrid/wiki/MiGRules, 3 March 2009. Visited 3. Mar 2009.

[7] Brian Vinter. The Architecture of the Minimum intrusion Grid: MiG. In *Proceedings of Communicating Process Architectures*, pages 189–201. IOS Press, 2005.

[8] Brian Vinter and Henrik Hoey Karlsen. Minimum intrusion Grid - The Simple Model. http://www.imada.sdu.dk/Courses/DM75/Misc/MiG_Simple_Model_ETNgrid05.pdf, 2005.

[9] Canonical. Virtual machine builder. https://launchpad.net/vmbuilder, 18 February 2009. Visited 18. Feb 2009.

[10] Chilkat. Python chilkat. http://www.chilkatsoft.com/python.asp, 02 2009. Visited 18. Feb 2009.

[11] Tomas Groth Christensen. Migrerende virtuelle maskiner i minimum intrusion grid. Master's thesis, Department of Computer Science, University of Copenhagen, April 2009.

[12] Virtual Box OSE Community. Forum. http://forums.virtualbox.org/viewtopic.php?f=10&t=14948, 02 2009. Visited 18. Feb 2009.

[13] Wikipedia Community. Comet. http://en.wikipedia.org/wiki/Comet_(programming), 04 2009. Visited 19. Apr 2009.

[14] Privoxy Developers. Privoxy. http://www.privoxy.org/, 02 2009. Visited 18. Feb 2009.

[15] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. In *In Proceedings of the 13th USENIX Security Symposium*, pages 303–320, 2004.

[16] Ian Foster. What is the Grid? A Three Point Checklist. http://www-fp.mcs.anl.gov/~foster/Articles/WhatIsTheGrid.pdf, 2002.

[17] Ian Foster and C. Kesselman. *The Grid: Blueprint for a New Computing Infrastructure*, chapter 2. Morgan Kaufmann Publishers, 1998. http://www.globus.org/alliance/publications/papers/chapter2.pdf.

[18] Apache Foundation. Apache - mod_proxy. http://httpd.apache.org/docs/2.0/mod/mod_proxy.html, 02 2009. Visited 18. Feb 2009.

[19] Globus. The Globus Resource Specification Language RSL. http://www.globus.org/toolkit/docs/2.4/gram/rsl_spec1.html, 18 January 2009.

[20] Network Working Group. RFC 4251. http://www.ietf.org/rfc/rfc4251.txt, 01 2006.

[21] Network Working Group. RFC 2616. http://tools.ietf.org/html/rfc2616, 04 2009. Visited 19. Apr 2009.

[22] Network Working Group. RFC 2818. http://tools.ietf.org/html/rfc2818, 04 2009. Visited 19. Apr 2009.

[23] Microsoft Help and Support. Understanding the Remote Desktop Protocol (RDP). http://support.microsoft.com/kb/186607.

[24] JSON. Website. http://www.json.org/, 04 2009. Visited 19. Apr 2009.

[25] Twisted Matrix Labs. Twisted. http://twistedmatrix.com/trac/, 18 February 2009. Visited 18. Feb 2009.

[26] libvirt. The virtualization api. http://libvirt.org/, 18 February 2009. Visited 18. Feb 2009.

[27] libvirt the virtualization api. Website. http://libvirt.org/, 04 2009. Visited 19. Apr 2009.

[28] Redpill Linpro. Varnish. http://varnish.projects.linpro.no/, 02 2009. Visited 18. Feb 2009.

[29] Ari Luotonen. Tunneling tcp based protocols through web proxy servers. http://tools.ietf.org/id/draft-luotonen-web-proxy-tunneling-00.txt, 01 1998.

[30] David Ascher Mark Lutz. *Learning Python*. O'Reilly Media, Incorporated, 3rd edition, July 2008.

[31] Alex Martelli, Anna Martelli Ravenscroft, and David Ascher. *Python Cookbook*. O'Reilly Media, Incorporated, 2nd edition, March 2005.

[32] Martin. MiG sandboxes. http://www.migrid.org/MiG/Mig_english/sandboxes_html, 04 January 2008. Visited 18. Feb 2009.

[33] Sun Microsystems, VirtualBox Community, and Previously InnoTek. VirtualBox. http://www.virtualbox.org/, 18 February 2009. Visited 18. Feb 2009.

[34] MiG. Google Code / Project Site. http://code.google.com/p/migrid/, 02 2009. Visited 18. Feb 2009.

[35] MiG. Website. http://www.migrid.org, 02 2009. Visited 18. Feb 2009.

[36] Microsoft Developer Network. Microsoft Communication Protocols. http://msdn.microsoft.com/en-us/library/cc216513(PROT.10).aspx, 18 February 2009.

[37] Microsoft Developer Network. Remote Desktop Protocol: Basic Connectivity and Graphics Remoting Specification. http://msdn.microsoft.com/en-us/library/cc240445(PROT.10).aspx, 2 August 2009.

[38] OpenBSD. Openssh. http://www.openssh.com/, 02 2009. Visited 18. Feb 2009.

[39] The OpenSSL Project. Openssh. http://www.openssl.org/, 02 2009. Visited 18. Feb 2009.

[40] Proxify. Proxify. http://proxify.co.uk/, 02 2009. Visited 18. Feb 2009.

[41] VNC Proxy. Vnc proxy. http://vncproxy.sourceforge.net/, 02 2009. Visited 18. Feb 2009.

[42] pyOpenSSL. Python interface to the openssl library. http://pyopenssl.sourceforge.net/, 02 2009. Visited 18. Feb 2009.

[43] Python. asyncore, asynchronous socket handler. http://docs.python.org/library/asyncore.html, 18 February 2009. Visited 18. Feb 2009.

[44] Python. multiprocessing, process-based threading interface. http://docs.python.org/library/multiprocessing.html, 18 February 2009. Visited 18. Feb 2009.

[45] Python. Python object sharing. http://poshmodule.sourceforge.net/, 18 February 2009. Visited 18. Feb 2009.

[46] Python. select, waiting for io completion. http://docs.python.org/library/select.html, 18 February 2009. Visited 18. Feb 2009.

[47] Python. Socketserver, a framework for network servers. http://docs.python.org/library/socketserver.html, 18 February 2009. Visited 18. Feb 2009.

[48] Python. subprocess, subprocess management. http://docs.python.org/library/subprocess.html, 18 February 2009. Visited 18. Feb 2009.

[49] Python. thread, multiple threads of control. http://docs.python.org/library/thread.html, 18 February 2009. Visited 18. Feb 2009.

[50] Python. threading, higher-level threading interface. http://docs.python.org/library/threading.html, 18 February 2009. Visited 18. Feb 2009.

[51] Python. Tls lite. http://trevp.net/tlslite/, 02 2009. Visited 18. Feb 2009.

[52] Python. Twisted vs threads benchmark. http://kaishaku.org/twisted-vs-threads/, 18 February 2009. Visited 18. Feb 2009.

[53] Python MeTooCrypto. M2crypto. http://chandlerproject.org/bin/view/Projects/MeTooCrypto, 02 2009. Visited 18. Feb 2009.

[54] python-vm builder. Website. https://launchpad.net/vmbuilder, 04 2009. Visited 19. Apr 2009.

[55] VNC Reflector. Vnc reflector. http://sourceforge.net/projects/vnc-reflector/, 02 2009. Visited 18. Feb 2009.

[56] UltraVNC Repeater. UltraVnc repeater. http://www.uvnc.com/addons/repeater.html, 02 2009. Visited 18. Feb 2009.

[57] Matteo Ricchetti. Ss5. http://ss5.sourceforge.net/, 02 2009. Visited 18. Feb 2009.

[58] Tristan Richardson, Quentin Stafford-fraser, Kenneth R. Wood, and Andy Hopper. Virtual network computing. *IEEE Internet Computing*, 2:33–38, 1998.

[59] Constantine Sapuntzakis and Monica S. Lam. Virtual Appliances in the Collective: A Road to Hassle-Free Computing. http://suif.stanford.edu//papers/hotos03-virtual-app.pdf, 2003.

[60] Socksify. Socksify. http://socksify.com/, 02 2009. Visited 18. Feb 2009.

[61] Initex Software. Proxifier. http://www.proxifier.com/, 02 2009. Visited 18. Feb 2009.

[62] Squid. Squid cache. http://www.squid-cache.org/, 02 2009. Visited 18. Feb 2009.

[63] Ananth I. Sundararaj and Peter A. Dinda. Towards Virtual Networks for Virtual Machine Grid Computing. In *Proceedings of the 3rd USENIX Virtual Machine Research And Technology Symposium (VM)*, pages 177–190, 2003.

[64] TeamViewer. Teamviewer. http://www.teamviewer.com/, 18 February 2009. Visited 18. Feb 2009.

[65] RealVNC Ltd (formerly of Olivetti Research Ltd / AT&T Labs Cambridge) Tristan Richardson. The RFB Protocol Version 3.8. http://www.realvnc.com/docs/rfbproto.pdf, 21 April 2009.

[66] VirtualBox. Virtualbox build instructions. http://www.virtualbox.org/wiki/Build_instructions, 18 February 2009. Visited 18. Feb 2009.

[67] VMWare. Open virtualization format. http://www.vmware.com/appliances/learn/ovf.html, 18 February 2009. Visited 18. Feb 2009.

[68] W3C. Html 5 specification - editors draft. http://dev.w3.org/html5/spec/Overview.html#channel-mess 04 2009. Visited 19. Apr 2009.

[69] Wikipedia Community. Cloud Computing. http://en.wikipedia.org/wiki/Cloud_computing, 18 February 2009.

[70] Wikipedia Community. Comparison of remote desktop software. http://en.wikipedia.org/wiki/Comparison_of_remote_desktop_software visited, 18 February 2009.

[71] Wikipedia Community. Desktop Environment. http://en.wikipedia.org/wiki/Desktop_environment, 18 February 2009.

[72] XML-RPC. Website. http://www.xmlrpc.com/, 04 2009. Visited 19. Apr 2009.

[73] Elizabeth D. Zwicky, Simon Cooper, and D. Brent Chapman. *Building Internet Firewalls*, chapter 9, pages 224–239. O'Reilly & Associates, June 2000.

A Appendix

A.1 Changelogs

- http://code.google.com/p/migrid/source/detail?r=275
- http://code.google.com/p/migrid/source/detail?r=311
- http://code.google.com/p/migrid/source/detail?r=344
- http://code.google.com/p/migrid/source/detail?r=364
- http://code.google.com/p/migrid/source/detail?r=372
- http://code.google.com/p/migrid/source/detail?r=378

A.2 Proxy and Proxy Agent Code-base

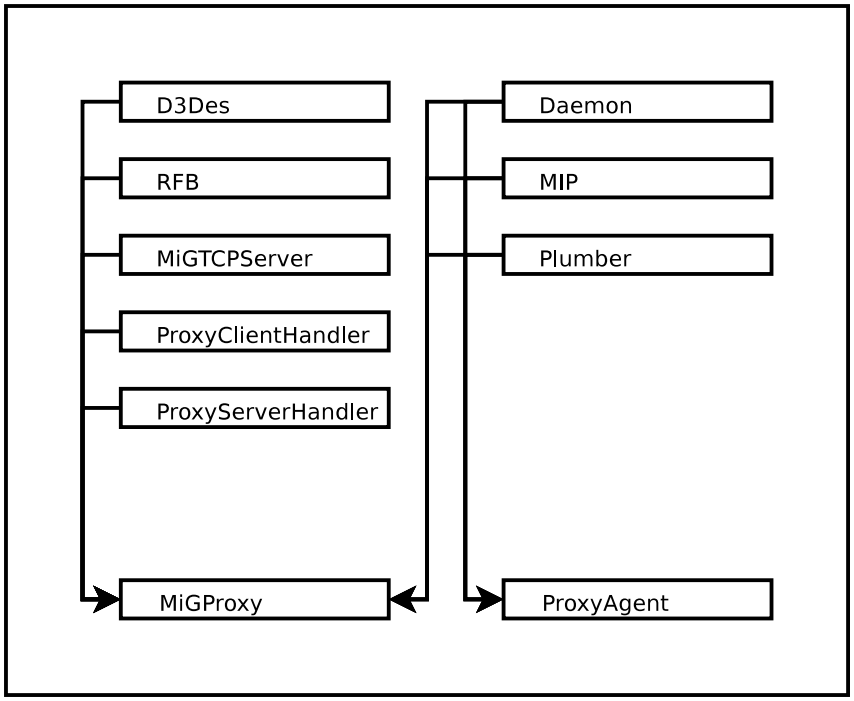


Fig. 23: Proxy and Proxy Agent code-base.

A.3 HTTP Encapsulation of RFB Messages

Frame-buffer update request:

```
POST /framebufer_update_request HTTP/1.1
Host: proxy.migrid.org
User-Agent: MigProxyAgent/0.1
Content-Length: XX

...RFB FRAMEBUFFER REQUEST...
```

Frame-buffer update response:

```
HTTP/1.1 200 OK
Content-Type: text/plain
Content-Length: YY
Server: MiGProxy/0.1

...RFB FRAMEBUFFER RESPONSE...
```

Fig. 24: HTTP Encapsulation.

A.4 HTTP Connect Method Encapsulation

Client sends:

```
CONNECT proxy.migrid.org:443 HTTP/1.0
User-agent: MiGProxyAgent/0.1

...data to be tunnelled TO the server...
```

Server Responds:

```
HTTP/1.0 200 Connection established
Proxy-agent: MiGProxyServer/0.1

...data tunnelled FROM the server...
```

Fig. 25: HTTP CONNECT method.

A.5 MiG Inter-proxy Protocol (MiP) Specification

Number	Name
0	*Proxy*
1	*Virtual machine*
2	*Resource*

Fig. 26: MiP Identifier types

Number	Name
0	*Handshake*
1	*Setup Request*
2	*Setup Response*

Fig. 27: MiP message types

No. of bytes	Type	Value	Description
1	U8	0	*message-type*
1	U8	{0,1,2}	*identifier-type*
4	U32	-	*identifier-length*
identity-length	-	-	*identifier*

Fig. 28: MiP handshake

No. of bytes	Type	Value	Description
1	U8	1	*message-type*
4	U32	0-65355	ticket
4	U32	-	*proxy-host-length*
proxy-host-length	-	-	*proxy-host*
4	U32	0-65355	*proxy-port*
4	U32	-	*endpoint-host-length*
endpoint-host-length	-	-	endpoint-host
4	U32	0-65355	endpoint-port

Fig. 29: MiP setup request

No. of bytes	Type	Value	Description
1	U8	2	*message-type*
4	U32	0-65355	*ticket*
1	U8	0-127	*status*

Fig. 30: MiP setup response

A.6 Integrating python-vm-builder with MIG

```
./hacks
./hacks/interfaces
./boot.sh
./login.sh
./guest_additions
./guest_additions/VBoxWindowsAdditions-amd64.exe
./guest_additions/VBoxSolarisAdditions.pkg
./guest_additions/VBoxWindowsAdditions.exe
./guest_additions/VBoxLinuxAdditions-x86.run
./guest_additions/VBoxLinuxAdditions-amd64.run
./guest_additions/VBoxWindowsAdditions-x86.exe
./machine.cfg
./machine.partition
./bundle
```

Fig. 31: Layout of build environment

A.7 Job Encapsulation of Virtual Machine Migration

The customized VirtualBox implemented a method *MigServerImport*, this is now handled by commands in the two job descriptions below. An additional VirtualBox method *MiGServerWait* monitored the execution time of the virtual machine in order to shut it down before the jobs execution time was exceeded this is now handled by the run-time wrapper script.

A.7.1 Job Description with System Disk on MiG server

```
1  ::EXECUTE::
2  rm -rf ~/.VirtualBox
3  mkdir ~/.VirtualBox
4  mkdir ~/.VirtualBox/Machines
5  mkdir ~/.VirtualBox/HardDisks
6  mv plain.vmdk ~/.VirtualBox/HardDisks/plain.vmdk
7  mv data.vmdk ~/.VirtualBox/HardDisks/+JOBID+_data.vmdk
8  VBoxManage openmedium disk +JOBID+_data.vmdk
9  VBoxManage openmedium disk plain.vmdk
10 VBoxManage createvm -name "$NAME" -register
11 VBoxManage modifyvm "$NAME" -nic1 nat
12 VBoxManage modifyvm "$NAME" -memory $MEMORY
13 VBoxManage modifyvm "$NAME" -pae on
14 VBoxManage modifyvm "$NAME" -hwvirtex on
15 VBoxManage modifyvm "$NAME" -ioapic off
16 VBoxManage modifyvm "$NAME" -hda "plain.vmdk"
17 VBoxManage modifyvm "$NAME" -hdb "+JOBID+_data.vmdk"
18 VBoxManage guestproperty set "$NAME" job_id +JOBID+
19 ./runvm.sh $NAME 780
20 VBoxManage modifyvm $NAME -hda none
21 VBoxManage modifyvm $NAME -hdb none
22 VBoxManage closemedium disk +JOBID+_data.vmdk
23 VBoxManage unregistervm $NAME -delete mv ~/.VirtualBox/HardDisks/+JOBID+_data.vmdk data.
       vmdk
24
25 ::INPUTFILES::
```

```
vms/$NAME/plain.vmdk plain.vmdk
vms/$NAME/data.vmdk data.vmdk

::OUTPUTFILES::
data.vmdk vms/$NAME/data.vmdk

::EXECUTABLES::
vms/runvm.sh runvm.sh

::MEMORY::
$MEMORY

::CPUTIME::
$CPUTIME

::ARCHITECTURE::
AMD64

::VGRID::
Generic

::NOTIFY::
jabber: SETTINGS
```

A.7.2 Job Description with System Disk on Resource

```
::EXECUTE::
rm -rf ~/.VirtualBox
mkdir ~/.VirtualBox
mkdir ~/.VirtualBox/Machines
mkdir ~/.VirtualBox/HardDisks
cp ~/vbox_disks/plain.vmdk ~/.VirtualBox/HardDisks/plain.vmdk
mv data.vmdk ~/.VirtualBox/HardDisks/+JOBID+_data.vmdk
VBoxManage openmedium disk +JOBID+_data.vmdk
VBoxManage openmedium disk plain.vmdk
VBoxManage createvm -name "$NAME" -register
VBoxManage modifyvm "$NAME" -nic1 nat
VBoxManage modifyvm "$NAME" -memory $MEMORY
VBoxManage modifyvm "$NAME" -pae on
VBoxManage modifyvm "$NAME" -hwvirtex on
VBoxManage modifyvm "$NAME" -ioapic off
VBoxManage modifyvm "$NAME" -hda "plain.vmdk"
VBoxManage modifyvm "$NAME" -hdb "+JOBID+_data.vmdk"
VBoxManage guestproperty set "$NAME" job_id +JOBID+
./runvm.sh $NAME 780
VBoxManage modifyvm $NAME -hda none
VBoxManage modifyvm $NAME -hdb none
VBoxManage closemedium disk +JOBID+_data.vmdk
VBoxManage unregistervm $NAME -delete mv ~/.VirtualBox/HardDisks/+JOBID+_data.vmdk data.
    vmdk

::INPUTFILES::
vms/$NAME/data.vmdk data.vmdk

::OUTPUTFILES::
data.vmdk vms/$NAME/data.vmdk

::EXECUTABLES::
vms/runvm.sh runvm.sh

```

```
::MEMORY::
$MEMORY

::CPUTIME::
$CPUTIME

::ARCHITECTURE::
AMD64

::VGRID::
Generic

::NOTIFY::
jabber: SETTINGS
```

A.7.3 Run-time Wrapper

```
#!/bin/bash
#
# Script to manage the execution time of the virtual machine.
#
# Arguments:
# 1 = Virtual Machine Name
# 2 = Execution time
#
VBOX_STATE=0
VM_NAME=$1
EXEC_TIME=$2
VBoxHeadless -startvm "$VM_NAME" & VBOX_PID=$!
while [[ $VBOX_STATE -eq 0 && $EXEC_TIME -gt 0 ]] do
  if kill -0 $VBOX_PID       # Is the process still alive?
  then
    VBOX_STATE=0 # Yes
  else
    VBOX_STATE=1 # No
  fi

  # Decrease exec time
  ((EXEC_TIME--))
  sleep 1

done
# If still running then turn it off
if [ $VBOX_STATE -eq 0 ]; then
VBoxManage controlvm "$VM_NAME" acpipowerbutton
fi

echo "CP: $VBOX_PID CS: $VBOX_STATE ET: $EXEC_TIME"
```

A.8 Building Customized VirtualBox

This is a walk-through of the Linux build instructions. on a Lenovo Thinkpad x200 running 64bit Xubuntu 8.10.

There are some additional information on the MIG prerequisites and a lot of output which might be useful if you want to compile VirtualBox yourself.

- Vanilla Prerequisites

```
sudo apt-get install gcc g++ bcc iasl xsltproc uuid-dev zlib1g-dev libidl-dev \
                libsdl1.2-dev libxcursor-dev libqt3-headers libqt3-mt-dev \
                libasound2-dev libstdc++5 libhal-dev libpulse-dev libxml2-dev \
                libxslt1-dev python2.5-dev libqt4-dev qt4-dev-tools libcap-dev
sudo apt-get install ia32-libs libc6-dev-i386 lib32gcc1 gcc-multilib \
                lib32stdc++6 g++-multilib
sudo ln -s libX11.so.6    /usr/lib32/libX11.so
sudo ln -s libXTrap.so.6  /usr/lib32/libXTrap.so
sudo ln -s libXt.so.6     /usr/lib32/libXt.so
sudo ln -s libXtst.so.6   /usr/lib32/libXtst.so
sudo ln -s libXmu.so.6    /usr/lib32/libXmu.so
sudo ln -s libXext.so.6   /usr/lib32/libXext.so
```

- MIG Prerequisites

Grab the latest and greatest "MIGified" VirtualBox unpack it and cd into it. Then install these packages:

```
sudo apt-get install libcurl3 libcurl3-gnutls libcurl4-openssl-dev libbz2-dev
```

- Start building

Execute the following command:

```
./configure --disable-hardening
```

This should give you an output like:

```
./configure --disable-hardening
Checking for environment: Determined build machine: linux.amd64, target machine:
    linux.amd64, OK.
Checking for kBuild: found, OK.
Checking for gcc: found version 4.3.2, OK.
Checking for as86: found version 0.16.17, OK.
Checking for bcc: found version 0.16.17, OK.
Checking for iasl: found version 20061109, OK.
Checking for xslt: found, OK.
Checking for pthread: found, OK.
Checking for libxml2: found version 2.6.32, OK.
Checking for libxslt: found version 1.1.24, OK.
Checking for libIDL: found version 0.8.10, OK.
Checking for zlib: found version 1.2.3.3, OK.
Checking for libpng: found version 1.2.27, OK.
Checking for SDL: found version 1.2.12, OK.
Checking for X libraries: found, OK.
Checking for Xcursor: found, OK.
Checking for Qt3: found version 3.3.8b, OK.
Checking for Qt3 devtools: found version 3.3.8b, OK.
Checking for Qt4: found version 4.4.3, OK.
Checking for Qt4 devtools: found version 4.4.3, OK.
Checking for python support: found version 2.5.2, OK.
Checking for static stc++ library: found, OK.
```

```
24 Checking for Linux kernel sources: found version 2.6.27, OK.
25 Checking for ALSA: found version 1.0.17, OK.
26 Checking for PulseAudio: found version 0.9.10 API version 11, OK.
27 Checking for libcap library: found, OK.
28 Checking for compiler.h: compiler.h not found, OK.
29 Checking for 32-bit support: OK.
30
31 Successfully generated '/home/safl/Desktop/bach/virtual/VirtualBox-2.1.4_OSE/
       AutoConfig.kmk' and '/home/safl/Desktop/bach/virtual/VirtualBox-2.1.4_OSE/env.
       sh'.
32 Source '/home/safl/Desktop/bach/virtual/VirtualBox-2.1.4_OSE/env.sh' once before
       you start to build VBox:
33
34   source /home/safl/Desktop/bach/virtual/VirtualBox-2.1.4_OSE/env.sh
35   kmk
36
37 To compile the kernel module, do:
38
39   cd ./out/linux.amd64/release/bin/src/vboxdrv
40   make
41
42
43   +++ WARNING +++ WARNING +++ WARNING +++ WARNING +++ WARNING +++ WARNING +++
44   Hardening is disabled. Please do NOT build packages for distribution with
45   disabled hardening!
46   +++ WARNING +++ WARNING +++ WARNING +++ WARNING +++ WARNING +++ WARNING +++
47
48 Enjoy!
```

Your environment should now be configured, so you can go ahead and start compiling, the argument -j to kmk is to specify the amount of cores in your machine. Set -j to the number of cores in your machine + 1.

```
1 source /home/safl/Desktop/bach/virtual/VirtualBox-2.1.4_OSE/env.sh
2 kmk -j3
```

The build took about on 14minutes on the x200, the build of the kernel module takes about 30 seconds. Build the kernel module by

```
1 cd out/linux.amd64/release/bin/src/
2 make sudo
3 make install
```

To be able to run it non-root you need to be part of the vboxusers group.

```
1 sudo groupadd vboxusers
2 sudo usermod -G vboxusers -a safl
```

Then load the kernel module

```
1 sudo modprobe vboxdrv
2 sudo chmod 660 /dev/vboxdrv
3 sudo chgrp vboxusers /dev/vboxdrv
```

Then you probably also need to disable KVM

```
sudo invoke-rc.d kvm stop
```

If you did not get any errors in the previous then you should now be able to run VirtualBox by:

```
cd ..
./VirtualBox &
```

A.9 Adding VNC to VirtualBox

A.9.1 MiGFramebuffer.h

```
/** @file
 *
 * Declaration of MigFramebuffer class
 */
#ifndef __H_FRAMEBUFFER
#define __H_FRAMEBUFFER

#include <iprt/thread.h>
#include <iprt/critsect.h>

//class MigFramebufferOverlay;

class MigFramebuffer :
    public IFramebuffer
{
public:
    MigFramebuffer(bool fFullscreen = false, bool fResizable = true, bool
        fShowSDLConfig = false,
              bool fKeepHostRes = false, uint32_t u32FixedWidth = ~(uint32_t)0,
              uint32_t u32FixedHeight = ~(uint32_t)0, uint32_t u32FixedBPP = ~(
                  uint32_t)0);
    virtual ~MigFramebuffer();

    NS_DECL_ISUPPORTS

    STDMETHOD(COMGETTER(Width))(ULONG *width);
    STDMETHOD(COMGETTER(Height))(ULONG *height);
    STDMETHOD(Lock)();
    STDMETHOD(Unlock)();
    STDMETHOD(COMGETTER(Address))(BYTE **address);
    STDMETHOD(COMGETTER(BitsPerPixel))(ULONG *bitsPerPixel);
    STDMETHOD(COMGETTER(BytesPerLine))(ULONG *bytesPerLine);
    STDMETHOD(COMGETTER(PixelFormat)) (ULONG *pixelFormat);
    STDMETHOD(COMGETTER(UsesGuestVRAM)) (BOOL *usesGuestVRAM);
    STDMETHOD(COMGETTER(HeightReduction)) (ULONG *heightReduction);
    STDMETHOD(COMGETTER(Overlay)) (IFramebufferOverlay **aOverlay);
    STDMETHOD(COMGETTER(WinId)) (uint64_t *winId);

    STDMETHOD(NotifyUpdate)(ULONG x, ULONG y,
                            ULONG w, ULONG h, BOOL *finished);
    STDMETHOD(RequestResize)(ULONG aScreenId, ULONG pixelFormat, BYTE *vram,
```

```
40                                  ULONG bitsPerPixel, ULONG bytesPerLine,
41                                  ULONG w, ULONG h, BOOL *finished);
42      STDMETHOD(OperationSupported)(FramebufferAccelerationOperation_T operation,
            BOOL *supported);
43      STDMETHOD(VideoModeSupported)(ULONG width, ULONG height, ULONG bpp, BOOL *
            supported);
44      STDMETHOD(SolidFill)(ULONG x, ULONG y, ULONG width, ULONG height,
45                           ULONG color, BOOL *handled);
46      STDMETHOD(CopyScreenBits)(ULONG xDst, ULONG yDst, ULONG xSrc, ULONG ySrc,
47                                ULONG width, ULONG height, BOOL *handled);
48
49      STDMETHOD(GetVisibleRegion)(BYTE *aRectangles, ULONG aCount, ULONG *
            aCountCopied);
50      STDMETHOD(SetVisibleRegion)(BYTE *aRectangles, ULONG aCount);
51
52      // internal public methods
53      bool initialized() { return mfInitialized; }
54      void resizeGuest();
55      void resizeSDL();
56      void update(int x, int y, int w, int h, bool fGuestRelative);
57      void repaint();
58      bool getFullscreen();
59      void setFullscreen(bool fFullscreen);
60      int  getXOffset();
61      int  getYOffset();
62      void getFullscreenGeometry(uint32_t *width, uint32_t *height);
63      uint32_t getGuestXRes() { return mGuestXRes; }
64      uint32_t getGuestYRes() { return mGuestYRes; }
65      void uninit();
66      void setWinId(uint64_t winId) { mWinId = winId; }
67
68  private:
69      /** the sdl thread */
70  //    RTNATIVETHREAD mSdlNativeThread;
71      /** current SDL framebuffer pointer (also includes screen width/height) */
72    //  SDL_Surface *mScreen;
73      /** false if constructor failed */
74      bool mfInitialized;
75      /** maximum possible screen width in pixels (~0 = no restriction) */
76      uint32_t mMaxScreenWidth;
77      /** maximum possible screen height in pixels (~0 = no restriction) */
78      uint32_t mMaxScreenHeight;
79      /** current guest screen width in pixels */
80      ULONG mGuestXRes;
81      /** current guest screen height in pixels */
82      ULONG mGuestYRes;
83      /** fixed SDL screen width (~0 = not set) */
84      uint32_t mFixedSDLWidth;
85      /** fixed SDL screen height (~0 = not set) */
86      uint32_t mFixedSDLHeight;
87      /** fixed SDL bits per pixel (~0 = not set) */
88      uint32_t mFixedSDLBPP;
```

```
89      /** default BPP */
90      uint32_t mDefaultSDLBPP;
91      /** Y offset in pixels, i.e. guest-nondrawable area at the top */
92      uint32_t mTopOffset;
93      /** X offset for guest screen centering */
94      uint32_t mCenterXOffset;
95      /** Y offset for guest screen centering */
96      uint32_t mCenterYOffset;
97      /** flag whether we're in fullscreen mode */
98      bool  mfFullscreen;
99      /** flag wheter we keep the host screen resolution when switching to
100      *  fullscreen or not */
101     bool  mfKeepHostRes;
102     /** framebuffer update semaphore */
103     RTCRITSECT mUpdateLock;
104     /** flag whether the SDL window should be resizable */
105     bool mfResizable;
106     /** flag whether we print out SDL information */
107     bool mfShowSDLConfig;
108     /** handle to window where framebuffer context is being drawn*/
109     uint64_t mWinId;
110
111 //    SDL_Surface *mSurfVRAM;
112
113     BYTE *mPtrVRAM;
114     ULONG mBitsPerPixel;
115     ULONG mBytesPerLine;
116     ULONG mPixelFormat;
117     BOOL mUsesGuestVRAM;
118     BOOL mfSameSizeRequested;
119
120     /** the application Icon */
121  //   SDL_Surface *mWMIcon;
122 };
123
124
125 #ifdef hejsa
126 class MigFramebufferOverlay :
127     public IFramebufferOverlay
128 {
129 public:
130     MigFramebufferOverlay(ULONG x, ULONG y, ULONG width, ULONG height, BOOL
            visible,
131                           MigFramebuffer *aParent);
132     virtual ~MigFramebufferOverlay();
133
134     NS_DECL_ISUPPORTS
135
136     STDMETHOD(COMGETTER(X))(ULONG *x);
137     STDMETHOD(COMGETTER(Y))(ULONG *y);
138     STDMETHOD(COMGETTER(Width))(ULONG *width);
139     STDMETHOD(COMGETTER(Height))(ULONG *height);
```

```
140     STDMETHOD(COMGETTER(Visible))(BOOL *visible);
141     STDMETHOD(COMSETTER(Visible))(BOOL visible);
142     STDMETHOD(COMGETTER(Alpha))(ULONG *alpha);
143     STDMETHOD(COMSETTER(Alpha))(ULONG alpha);
144     STDMETHOD(COMGETTER(Address))(ULONG *address);
145     STDMETHOD(COMGETTER(BytesPerLine))(ULONG *bytesPerLine);
146
147     /* These are not used, or return standard values. */
148     STDMETHOD(COMGETTER(BitsPerPixel))(ULONG *bitsPerPixel);
149     STDMETHOD(COMGETTER(PixelFormat)) (ULONG *pixelFormat);
150     STDMETHOD(COMGETTER(UsesGuestVRAM)) (BOOL *usesGuestVRAM);
151     STDMETHOD(COMGETTER(HeightReduction)) (ULONG *heightReduction);
152     STDMETHOD(COMGETTER(Overlay)) (IFramebufferOverlay **aOverlay);
153     STDMETHOD(COMGETTER(WinId)) (ULONG64 *winId);
154
155     STDMETHOD(Lock)();
156     STDMETHOD(Unlock)();
157     STDMETHOD(Move)(ULONG x, ULONG y);
158     STDMETHOD(NotifyUpdate)(ULONG x, ULONG y,
159                             ULONG w, ULONG h, BOOL *finished);
160     STDMETHOD(RequestResize)(ULONG aScreenId, ULONG pixelFormat, ULONG vram,
161                              ULONG bitsPerPixel, ULONG bytesPerLine,
162                              ULONG w, ULONG h, BOOL *finished);
163     STDMETHOD(OperationSupported)(FramebufferAccelerationOperation_T operation,
164                                   BOOL *supported);
165     STDMETHOD(VideoModeSupported)(ULONG width, ULONG height, ULONG bpp, BOOL *
            supported);
166     STDMETHOD(SolidFill)(ULONG x, ULONG y, ULONG width, ULONG height,
167                          ULONG color, BOOL *handled);
168     STDMETHOD(CopyScreenBits)(ULONG xDst, ULONG yDst, ULONG xSrc, ULONG ySrc,
169                               ULONG width, ULONG height, BOOL *handled);
170
171     // internal public methods
172     HRESULT init();
173
174 private:
175     /** Overlay X offset */
176     ULONG mOverlayX;
177     /** Overlay Y offset */
178     ULONG mOverlayY;
179     /** Overlay width */
180     ULONG mOverlayWidth;
181     /** Overlay height */
182     ULONG mOverlayHeight;
183     /** Whether the overlay is currently active */
184     BOOL mOverlayVisible;
185     /** The parent IFramebuffer */
186     MigFramebuffer *mParent;
187     /** SDL surface containing the actual framebuffer bits */
188     SDL_Surface *mOverlayBits;
189     /** Additional SDL surface used for combining the framebuffer and the overlay
            */
```

```
190     SDL_Surface *mBlendedBits;
191
192 };
193 #endif
194
195 #endif // __H_FRAMEBUFFER
```

A.9.2 MiGFramebuffer.cpp

```
1  /** @file
2   *
3   * VBox frontends: VBoxSDL (simple frontend based on SDL):
4   * Implementation of MigFramebuffer (SDL framebuffer) class
5   */
6  #include <VBox/com/com.h>
7  #include <VBox/com/string.h>
8  #include <VBox/com/Guid.h>
9  #include <VBox/com/ErrorInfo.h>
10 #include <VBox/com/EventQueue.h>
11 #include <VBox/com/VirtualBox.h>
12
13 #include <iprt/stream.h>
14 #include <iprt/env.h>
15
16 #ifdef RT_OS_OS2
17 # undef RT_MAX
18 // from <iprt/cdefs.h>
19 # define RT_MAX(Value1, Value2)  ((Value1) >= (Value2) ? (Value1) : (Value2))
20 #endif
21 #include "MigFramebuffer.h"
22
23 using namespace com;
24
25 #define LOG_GROUP LOG_GROUP_GUI
26 #include <VBox/err.h>
27 #include <VBox/log.h>
28 #include <stdio.h>
29
30 #if defined(VBOX_WITH_XPCOM)
31 NS_IMPL_ISUPPORTS1_CI(MigFramebuffer, IFramebuffer)
32 NS_DECL_CLASSINFO(MigFramebuffer)
33 //NS_IMPL_ISUPPORTS1_CI(MigFramebufferOverlay, IFramebufferOverlay)
34 //NS_DECL_CLASSINFO(MigFramebufferOverlay)
35 #endif
36
37 //
38 // Constructor / destructor
39 //
40
41 /**
42  * SDL framebuffer constructor. It is called from the main
43  * (i.e. SDL) thread. Therefore it is safe to use SDL calls
```

```
44   * here.
45   * @param fFullscreen     flag whether we start in fullscreen mode
46   * @param fResizable      flag whether the SDL window should be resizable
47   * @param fShowSDLConfig  flag whether we print out SDL settings
48   * @param fKeepHostRes    flag whether we switch the host screen resolution
49   *                        when switching to fullscreen or not
50   * @param iFixedWidth     fixed SDL width (-1 means not set)
51   * @param iFixedHeight    fixed SDL height (-1 means not set)
52   */
53  MigFramebuffer::MigFramebuffer(bool fFullscreen, bool fResizable, bool
        fShowSDLConfig,
54                                 bool fKeepHostRes, uint32_t u32FixedWidth,
55                                 uint32_t u32FixedHeight, uint32_t u32FixedBPP)
56  {
57      int rc;
58      LogFlow(("MigFramebuffer::MigFramebuffer\n"));
59
60      ////mSurfVRAM        = NULL;
61      mfInitialized    = false;
62      mfFullscreen     = fFullscreen;
63      mfKeepHostRes    = fKeepHostRes;
64      mTopOffset       = 0;
65      mfResizable      = fResizable;
66      mfShowSDLConfig  = fShowSDLConfig;
67      mFixedSDLWidth   = u32FixedWidth;
68      mFixedSDLHeight  = u32FixedHeight;
69      mFixedSDLBPP     = u32FixedBPP;
70      mDefaultSDLBPP   = 32;
71      mCenterXOffset   = 0;
72      mCenterYOffset   = 0;
73      /* Start with standard screen dimensions. */
74      mGuestXRes       = 640;
75      mGuestYRes       = 480;
76      mPixelFormat     = FramebufferPixelFormat_Opaque;
77      mUsesGuestVRAM   = FALSE;
78      mPtrVRAM         = NULL;
79      mBitsPerPixel    = 0;
80      mBytesPerLine    = 0;
81      mfSameSizeRequested = false;
82      //mWMIcon          = NULL;
83
84      rc = RTCritSectInit(&mUpdateLock);
85      AssertMsg(rc == VINF_SUCCESS, ("Error from RTCritSectInit!\n"));
86
87      RTPrintf("MIG:: Hey i was just started!\n");
88  }
89
90  MigFramebuffer::~MigFramebuffer()
91  {
92      RTPrintf("MIG:: Hey i was just killed!\n");
93      LogFlow(("MigFramebuffer::~MigFramebuffer\n"));
94      RTCritSectDelete(&mUpdateLock);
```

```
95  }
96
97  /**
98   * Returns the current framebuffer width in pixels.
99   *
100  * @returns COM status code
101  * @param    width Address of result buffer.
102  */
103 STDMETHODIMP MigFramebuffer::COMGETTER(Width)(ULONG *width)
104 {
105     LogFlow(("MigFramebuffer::GetWidth\n"));
106     if (!width)
107         return E_INVALIDARG;
108     *width = mGuestXRes;
109     return S_OK;
110 }
111
112 /**
113  * Returns the current framebuffer height in pixels.
114  *
115  * @returns COM status code
116  * @param    height Address of result buffer.
117  */
118 STDMETHODIMP MigFramebuffer::COMGETTER(Height)(ULONG *height)
119 {
120     LogFlow(("MigFramebuffer::GetHeight\n"));
121     if (!height)
122         return E_INVALIDARG;
123     *height = mGuestYRes;
124     return S_OK;
125 }
126
127 /**
128  * Lock the framebuffer (make its address immutable).
129  *
130  * @returns COM status code
131  */
132 STDMETHODIMP MigFramebuffer::Lock()
133 {
134     LogFlow(("MigFramebuffer::Lock\n"));
135     RTCritSectEnter(&mUpdateLock);
136     return S_OK;
137 }
138
139 /**
140  * Unlock the framebuffer.
141  *
142  * @returns COM status code
143  */
144 STDMETHODIMP MigFramebuffer::Unlock()
145 {
146     LogFlow(("MigFramebuffer::Unlock\n"));
```

```
147     RTCritSectLeave(&mUpdateLock);
148     return S_OK;
149 }
150
151 /**
152  * Return the framebuffer start address.
153  *
154  * @returns COM status code.
155  * @param   address Pointer to result variable.
156  * @TODO: implement
157  */
158 STDMETHODIMP MigFramebuffer::COMGETTER(Address)(BYTE **address)
159 {
160     LogFlow(("MigFramebuffer::GetAddress\n"));
161     if (!address)
162         return E_INVALIDARG;
163
164     LogFlow(("VBoxSDL::GetAddress returning %p\n", *address));
165     return S_OK;
166 }
167
168 /**
169  * Return the current framebuffer color depth.
170  *
171  * @returns COM status code
172  * @param   bitsPerPixel Address of result variable
173  */
174 STDMETHODIMP MigFramebuffer::COMGETTER(BitsPerPixel)(ULONG *bitsPerPixel)
175 {
176     LogFlow(("MigFramebuffer::GetBitsPerPixel\n"));
177     if (!bitsPerPixel)
178         return E_INVALIDARG;
179
180     *bitsPerPixel = (ULONG)(16);
181     return S_OK;
182 }
183
184 /**
185  * Return the current framebuffer line size in bytes.
186  *
187  * @returns COM status code.
188  * @param   lineSize Address of result variable.
189  */
190 STDMETHODIMP MigFramebuffer::COMGETTER(BytesPerLine)(ULONG *bytesPerLine)
191 {
192     LogFlow(("MigFramebuffer::GetBytesPerLine\n"));
193     if (!bytesPerLine)
194         return E_INVALIDARG;
195
196     *bytesPerLine = (ULONG)(1);
197     return S_OK;
198 }
```

```
199
200 STDMETHODIMP MigFramebuffer::COMGETTER(PixelFormat) (ULONG *pixelFormat)
201 {
202     if (!pixelFormat)
203         return E_POINTER;
204     *pixelFormat = mPixelFormat;
205     return S_OK;
206 }
207
208 STDMETHODIMP MigFramebuffer::COMGETTER(UsesGuestVRAM) (BOOL *usesGuestVRAM)
209 {
210     if (!usesGuestVRAM)
211         return E_POINTER;
212     *usesGuestVRAM = mUsesGuestVRAM;
213     return S_OK;
214 }
215
216 /**
217  * Returns by how many pixels the guest should shrink its
218  * video mode height values.
219  *
220  * @returns COM status code.
221  * @param   heightReduction Address of result variable.
222  */
223 STDMETHODIMP MigFramebuffer::COMGETTER(HeightReduction)(ULONG *heightReduction)
224 {
225     if (!heightReduction)
226         return E_POINTER;
227     *heightReduction = 0;
228     return S_OK;
229 }
230
231 /**
232  * Returns a pointer to an alpha-blended overlay used for displaying status
233  * icons above the framebuffer.
234  *
235  * @returns COM status code.
236  * @param   aOverlay The overlay framebuffer.
237  */
238 STDMETHODIMP MigFramebuffer::COMGETTER(Overlay)(IFramebufferOverlay **aOverlay)
239 {
240     if (!aOverlay)
241         return E_POINTER;
242     /* Not yet implemented */
243     *aOverlay = 0;
244     return S_OK;
245 }
246
247 /**
248  * Returns handle of window where framebuffer context is being drawn
249  *
250  * @returns COM status code.
```

```
251  * @param    winId Handle of associated window.
252  */
253 STDMETHODIMP MigFramebuffer::COMGETTER(WinId)(uint64_t *winId)
254 {
255     if (!winId)
256         return E_POINTER;
257     *winId = mWinId;
258     return S_OK;
259 }
260
261 /**
262  * Notify framebuffer of an update.
263  *
264  * @returns COM status code
265  * @param    x         Update region upper left corner x value.
266  * @param    y         Update region upper left corner y value.
267  * @param    w         Update region width in pixels.
268  * @param    h         Update region height in pixels.
269  * @param    finished  Address of output flag whether the update
270  *                     could be fully processed in this call (which
271  *                     has to return immediately) or VBox should wait
272  *                     for a call to the update complete API before
273  *                     continuing with display updates.
274  */
275 STDMETHODIMP MigFramebuffer::NotifyUpdate(ULONG x, ULONG y,
276                                           ULONG w, ULONG h, BOOL *finished)
277 {
278     /*
279      * The input values are in guest screen coordinates.
280      */
281 //     LogFlow(("MigFramebuffer::NotifyUpdate: x = %d, y = %d, w = %d, h = %d\n",
            x, y, w, h));
282     RTPrintf("MigFramebuffer::NotifyUpdate: x = %d, y = %d, w = %d, h = %d\n", x,
            y, w, h);
283     /*
284      * The Display thread can continue as we will lock the framebuffer
285      * from the SDL thread when we get to actually doing the update.
286      */
287     if (finished)
288         *finished = TRUE;
289     return S_OK;
290 }
291
292 /**
293  * Request a display resize from the framebuffer.
294  *
295  * @returns COM status code.
296  * @param    pixelFormat The requested pixel format.
297  * @param    vram        Pointer to the guest VRAM buffer (can be NULL).
298  * @param    bitsPerPixel Color depth in bits.
299  * @param    bytesPerLine Size of a scanline in bytes.
300  * @param    w           New display width in pixels.
```

```
301  * @param    h             New display height in pixels.
302  * @param    finished      Address of output flag whether the update
303  *                         could be fully processed in this call (which
304  *                         has to return immediately) or VBox should wait
305  *                         for all call to the resize complete API before
306  *                         continuing with display updates.
307  */
308 STDMETHODIMP MigFramebuffer::RequestResize(ULONG aScreenId, ULONG pixelFormat,
         BYTE *vram,
309                                            ULONG bitsPerPixel, ULONG bytesPerLine,
310                                            ULONG w, ULONG h, BOOL *finished)
311 {
312     LogFlowFunc (("w=%d, h=%d, pixelFormat=0x%08lX, vram=%p, "
313                   "bpp=%d, bpl=%d\n",
314                   w, h, pixelFormat, vram, bitsPerPixel, bytesPerLine));
315     RTPrintf("w=%d, h=%d, pixelFormat=0x%08lX, vram=%p, "
316                   "bpp=%d, bpl=%d\n",
317                   w, h, pixelFormat, vram, bitsPerPixel, bytesPerLine);
318
319     /*
320      * SDL does not allow us to make this call from any other thread than
321      * the main thread (the one which initialized the video mode). So we
322      * have to send an event to the main SDL thread and tell VBox to wait.
323      */
324     if (!finished)
325     {
326         AssertMsgFailed(("RequestResize requires the finished flag!\n"));
327         return E_FAIL;
328     }
329
330     /*
331      * Optimize the case when the guest has changed only the VRAM ptr
332      * and the framebuffer uses the guest VRAM as the source bitmap.
333      */
334     if (    mGuestXRes     == w
335         && mGuestYRes     == h
336         && mPixelFormat   == pixelFormat
337         && mBitsPerPixel  == bitsPerPixel
338         && mBytesPerLine  == bytesPerLine
339         && mUsesGuestVRAM
340        )
341     {
342         mfSameSizeRequested = true;
343     }
344     else
345     {
346         mfSameSizeRequested = false;
347     }
348
349     mGuestXRes    = w;
350     mGuestYRes    = h;
351     mPixelFormat = pixelFormat;
```

```
352     mPtrVRAM      = vram;
353     mBitsPerPixel = bitsPerPixel;
354     mBytesPerLine = bytesPerLine;
355     mUsesGuestVRAM = FALSE; /* yet */
356
357     /* we want this request to be processed quickly, so yield the CPU */
358     RTThreadYield();
359
360     *finished = false;
361
362     return S_OK;
363 }
364
365 /**
366  * Returns which acceleration operations are supported
367  *
368  * @returns   COM status code
369  * @param     operation acceleration operation code
370  * @supported result
371  */
372 STDMETHODIMP MigFramebuffer::OperationSupported(FramebufferAccelerationOperation_T
        operation, BOOL *supported)
373 {
374     if (!supported)
375         return E_POINTER;
376
377     *supported = false;
378
379     return S_OK;
380 }
381
382 /**
383  * Returns whether we like the given video mode.
384  *
385  * @returns COM status code
386  * @param   width       video mode width in pixels
387  * @param   height      video mode height in pixels
388  * @param   bpp         video mode bit depth in bits per pixel
389  * @param   supported   pointer to result variable
390  */
391 STDMETHODIMP MigFramebuffer::VideoModeSupported(ULONG width, ULONG height, ULONG
        bpp, BOOL *supported)
392 {
393     if (!supported)
394         return E_POINTER;
395
396     /* are constraints set? */
397     if (    (    (mMaxScreenWidth != ~(uint32_t)0)
398               && (width > mMaxScreenWidth))
399         || (    (mMaxScreenHeight != ~(uint32_t)0)
400               && (height > mMaxScreenHeight)))
401     {
```

```
402         /* nope, we don't want that (but still don't freak out if it is set) */
403         *supported = false;
404     }
405     else
406     {
407         /* anything will do */
408         *supported = true;
409     }
410     return S_OK;
411 }
412
413 STDMETHODIMP MigFramebuffer::SolidFill(ULONG x, ULONG y, ULONG width, ULONG height
        ,
414                                        ULONG color, BOOL *handled)
415 {
416     if (!handled)
417         return E_POINTER;
418     RTPrintf("SolidFill: x: %d, y: %d, w: %d, h: %d, color: %d\n", x, y, width,
        height, color);
419
420     return S_OK;
421 }
422
423 STDMETHODIMP MigFramebuffer::CopyScreenBits(ULONG xDst, ULONG yDst, ULONG xSrc,
    ULONG ySrc,
424                                             ULONG width, ULONG height, BOOL *handled)
425 {
426     if (!handled)
427         return E_POINTER;
428     return S_OK;
429 }
430
431 STDMETHODIMP MigFramebuffer::GetVisibleRegion(BYTE *aRectangles, ULONG aCount,
432                                               ULONG *aCountCopied)
433 {
434   PRTRECT rects = (PRTRECT)aRectangles;
435
436     if (!rects)
437         return E_POINTER;
438
439     /// @todo
440
441   NOREF(aCount);
442   NOREF(aCountCopied);
443
444     return S_OK;
445 }
446
447 STDMETHODIMP MigFramebuffer::SetVisibleRegion(BYTE *aRectangles, ULONG aCount)
448 {
449     PRTRECT rects = (PRTRECT)aRectangles;
450
```

```
451     if (!rects)
452         return E_POINTER;
453
454     /// @todo
455
456   NOREF(aCount);
457
458     return S_OK;
459 }
460
461 //
462 // Internal public methods
463 //
464
465 /**
466  * Method that does the actual resize of the guest framebuffer and
467  * then changes the SDL framebuffer setup.
468  */
469 void MigFramebuffer::resizeGuest()
470 {
471     LogFlowFunc (("mGuestXRes: %d, mGuestYRes: %d\n", mGuestXRes, mGuestYRes));
472     AssertMsg(mSdlNativeThread == RTThreadNativeSelf(),
473               ("Wrong thread! SDL is not threadsafe!\n"));
474
475     uint32_t Rmask, Gmask, Bmask, Amask = 0;
476
477     mUsesGuestVRAM = FALSE;
478
479     /* pixel characteristics. if we don't support the format directly, we will
480      * fallback to the indirect 32bpp buffer (mUsesGuestVRAM will remain
481      * FALSE) */
482     if (mPixelFormat == FramebufferPixelFormat_FOURCC_RGB)
483     {
484         switch (mBitsPerPixel)
485         {
486             case 16:
487             case 24:
488             case 32:
489                 mUsesGuestVRAM = TRUE;
490                 break;
491             default:
492                 /* the fallback buffer is always 32bpp */
493                 mBitsPerPixel = 32;
494                 mBytesPerLine = mGuestXRes * 4;
495                 break;
496         }
497     }
498     else
499     {
500         /* the fallback buffer is always RGB, 32bpp */
501         mPixelFormat = FramebufferPixelFormat_FOURCC_RGB;
502         mBitsPerPixel = 32;
```

```
503             mBytesPerLine = mGuestXRes * 4;
504         }
505
506         switch (mBitsPerPixel)
507         {
508             case 16: Rmask = 0x0000F800; Gmask = 0x000007E0; Bmask = 0x0000001F; break
                    ;
509             default: Rmask = 0x00FF0000; Gmask = 0x0000FF00; Bmask = 0x000000FF; break
                    ;
510         }
511
512         LogFlow(("VBoxSDL:: created VRAM surface %p\n", mSurfVRAM));
513
514     }
515
516     /**
517      * Sets SDL video mode. This is independent from guest video
518      * mode changes.
519      *
520      * @remarks Must be called from the SDL thread!
521      */
522     void MigFramebuffer::resizeSDL(void)
523     {
524         LogFlow(("VBoxSDL:resizeSDL\n"));
525
526     }
527
528     /**
529      * Update specified framebuffer area. The coordinates can either be
530      * relative to the guest framebuffer or relative to the screen.
531      *
532      * @remarks Must be called from the SDL thread on Linux!
533      * @param    x               left column
534      * @param    y               top row
535      * @param    w               width in pixels
536      * @param    h               height in pixels
537      * @param    fGuestRelative  flag whether the above values are guest relative or
            screen relative;
538      */
539     void MigFramebuffer::update(int x, int y, int w, int h, bool fGuestRelative)
540     {
541       RTPrintf("VBoxSDL::update: %dx %dy %dw, %dh \n", x, y, w, h);
542     }
543
544     /**
545      * Repaint the whole framebuffer
546      *
547      * @remarks Must be called from the SDL thread!
548      */
549     void MigFramebuffer::repaint()
550     {
551         LogFlow(("MigFramebuffer::repaint\n"));
```

```
552     RTPrintf("MigFramebuffer::repaint\n");
553     //update(0, 0, mScreen->w, mScreen->h, false /* fGuestRelative */);
554     update(0, 0, 0, 0, false /* fGuestRelative */);
555 }
556
557 bool MigFramebuffer::getFullscreen()
558 {
559     LogFlow(("MigFramebuffer::getFullscreen\n"));
560     return mfFullscreen;
561 }
562
563 /**
564  * Toggle fullscreen mode
565  *
566  * @remarks Must be called from the SDL thread!
567  */
568 void MigFramebuffer::setFullscreen(bool fFullscreen)
569 {
570     LogFlow(("MigFramebuffer::SetFullscreen: fullscreen: %d\n", fFullscreen));
571     mfFullscreen = fFullscreen;
572 }
573
574 /**
575  * Return the geometry of the host. This isn't very well tested but it seems
576  * to work at least on Linux hosts.
577  */
578 void MigFramebuffer::getFullscreenGeometry(uint32_t *width, uint32_t *height)
579 {
580
581
582 }
583
584 /**
585  * Returns the current x offset of the start of the guest screen
586  *
587  * @returns current x offset in pixels
588  */
589 int MigFramebuffer::getXOffset()
590 {
591     /* there can only be an offset for centering */
592     return mCenterXOffset;
593 }
594
595 /**
596  * Returns the current y offset of the start of the guest screen
597  *
598  * @returns current y offset in pixels
599  */
600 int MigFramebuffer::getYOffset()
601 {
602     /* we might have a top offset and a center offset */
603     return mTopOffset + mCenterYOffset;
```

```
604 }
605
606 /**
607  * Terminate SDL
608  *
609  * @remarks must be called from the SDL thread!
610  */
611 void MigFramebuffer::uninit()
612 {
613
614 }
```