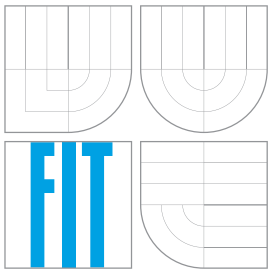
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

SYSTÉM PRO SPRÁVU VIRTUÁLNÍCH STROJŮ

VIRTUAL MACHINE MANAGEMENT SYSTEM

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

MILAN SKÁLA

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. LUDĚK DOLÍHAL

BRNO 2016

Zadání bakalářské práce/18450/2015/xskala09

Vysoké učení technické v Brně - Fakulta informačních technologií

Výzkumné centrum informačních technologií Akademický rok 2015/2016

Zadání bakalářské práce

Řešitel: **Skála Milan**
Obor: Informační technologie
Téma: **Systém pro správu virtuálních strojů**
Virtual Machine Management System
Kategorie: Počítačové sítě

Pokyny:

1. Seznamte se detailně s prostředím Virtual Box pro správu virtuálních strojů.
2. Dle pokynů vedoucího navrhněte aplikaci, která bude umožňovat vzdálenou správu virtuálních strojů.
3. Zaměřte se zejména na snadnou automatizaci navržené správy.
4. Proveďte implementaci navrženého řešení.
5. Zhodnoťte práci z hlediska reálné využitelnosti a rychlosti.

Literatura:

- *Oracle VM VirtualBox, User Manual,* [online]. [cit. 2015-09-15]. Dostupné z: https://www.virtualbox.org/wiki/End-user_documentation
- LUTZ, Mark. *Learning Python*. OReilly Media, 2013
- Codasip. *Codasip Studio User Guide.* Codasip s.r.o., 2015

Pro udělení zápočtu za první semestr je požadováno:

- *Body 1 a 2.*

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Dolíhal Luděk, Ing.**, VCIT FIT VUT
Datum zadání: 1. listopadu 2015
Datum odevzdání: 18. května 2016

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
L.S.
Výzkumné centrum informačních technologií
Božetěchova 2, 612 66 Brno

prof. Ing. Tomáš Hruška, CSc.
vedoucí ústavu

Abstrakt

Tato práce se zabývá vytvořením návrhu a implementací aplikace pro vzdálenou správu virtuálních strojů, která bude umožňovat tuto správu automatizovat. Jsou zde popsány důvody zavádění virtualizace ve firmách a korporacích, různé metody virtualizace včetně jejich zhodnocení z praktického hlediska. Dále jsou rozebrány existující řešení virtualizace, která jsou celosvětově rozšířená. V praktické části je proveden návrh a implementace aplikace, pomocí níž bude možné vzdáleně ovládat virtuální stroje. Na závěr jsou zhodnoceny možnosti dalšího rozšíření aplikace.

Abstract

This thesis focuses on design and implementation of the application for remote management of virtual machines that will be able to manage the virtual machines automatically. It describes a motivation for deployment of virtualization technology in companies and corporations, various virtualization methods altogether with their assessment from the practical point of view. The existing, globally widespread solutions, are also analyzes in the thesis. The application, which will be able to remotely control virtual machines, is designed and implemented in the practical part of this thesis. The final part describes possibilities of further extensions of the application.

Klíčová slova

virtualizace, VirtualBox, VBoxManage, VirtualBox API, virtuální stroj, automatizace, vzdálená správa, Python

Keywords

virtualization, VirtualBox, VBoxManage, VirtualBox API, virtual machine, automation, remote management, Python

Citace

SKÁLA, Milan. *Systém pro správu virtuálních strojů.* Brno, 2016. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Dolíhal Luděk.

Systém pro správu virtuálních strojů

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Luďka Dolíhala. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Milan Skála
16. května 2016

Poděkování

Chtěl bych poděkovat svému vedoucímu bakalářské práce Ing. Luďku Dolíhalovi za odborné vedení, za pomoc a rady při zpracování této práce.

Obsah

Kapitola 1

Úvod

Virtualizace je technologie, která si v dnešní době získává stále více příznivců z oblasti informačních technologií. Tento rozmach je způsoben širokou využitelností a flexibilitou virtualizačních technologií. Díky virtualizaci je možné snížit celkové náklady na provoz serverů, maximalizovat dostupnost služeb, které vyžadují nepřetržitý provoz nebo výrazně urychlit proces migrace serverů. Kvůli těmto vlastnostem je virtualizace používána spíše v prostředí firem a společností, jež potřebují spravovat velké množství fyzických strojů. Možnosti využití je možné najít i u jednotlivých uživatelů, zejména vývojářů softwaru, kteří tuto technologii využívají pro testování svého kódu na více operačních systémech zároveň, a tím výrazně urychlují celý vývoj produktu.

V první části práce je popsán vývoj virtualizace od doby, kdy byly v IT oblasti rozšířeny střediskové počítače až do současnosti, kdy tuto technologii používá velké množství lidí a její rozvoj rapidně roste. Dále je zde uvedena motivace pro nasazování virtualizační technologie, ať už v malých podnicích nebo celosvětových korporacích.

Druhá část se zabývá rozborem způsobů virtualizace podle míry, s jakou abstrahují fyzické komponenty stroje. Jsou zde tedy popsány způsoby, kterými se virtualizuje hardware i software. U každého způsobu je popsán princip, na jakém funguje, typická oblast využití a pro doplnění je uveden i reálný příklad softwaru, který daný způsob využívá.

Třetí částí práce je popis existujících řešení, které využívají virtualizaci. Popsány jsou produkty celosvětově známých korporací, které stojí za zmínku, neboť jsou velmi populární mezi jinými firmami nebo uživateli. Největší část je věnována virtualizačnímu nástroji VirtualBox od společnosti Oracle, neboť pro něj bude implementována aplikace, která je cílem této práce. Rozebrány jsou možnosti správy virtuálních strojů, ze kterých je následně vybrán ten, který nejvíce vyhovuje požadavkům výsledného programu.

Praktickou částí, jež je náplní práce, je návrh a implementace aplikace, která bude umožňovat vzdálenou správu virtuálních strojů. Velmi často nastává situace, kdy je potřeba provést stejnou operaci na více strojích. Uživatel se tedy musí připojit na všechny virtuální stroje zvlášť a operaci danou provést ručně. Hlavním cílem aplikace je tomuto přístupu zabránit a správu strojů urychlit pomocí automatizace, aby došlo k maximální úspoře času při provádění operací údržby strojů, mezi které patří např. aktualizace systému, spuštění skriptu na virtuálním stroji a další. Následně bude program otestován na platformách Microsoft Windows a jedné z distribucí Linuxu a bude zhodnocena jeho funkcionalita včetně rychlosti aplikace. Na závěr jsou popsány některé možnosti rozšíření, které by potenciálně vedly k lepší uživatelské přívětivosti či ke zdokonalení celého programu.

Kapitola 2

Virtualizace

Virtualizace je pojem, který se do povědomí široké veřejnosti dostává v průběhu tohoto tisíciletí. Přesto jde o mnohem starší technologii, jejíž vývoj způsobil revoluci v oblasti informačních technologií. V první části kapitoly bude popsán vývoj virtualizační technologie od jejího počátku. Další část bude shrnovat výhody této technologie a důvody nasazování v IT firmách a korporacích.

2.1 Historie

Za kořeny virtualizace je považováno období 60. let 20. století, kdy byly hlavními průkopníky v oblasti hardwaru a softwaru korporace IBM, Bell Labs a General Electric. V této době nebyly počítače schopny vykonávat více programů najednou. V případě potřeby zpracovávání vícero programů, muselo být použito tzv. dávkové zpracování, díky kterému bylo možné provádět programy bez účasti uživatele. IBM se tento přístup zdál být dostačující, neboť splňoval požadavky kladené vědeckými pracovníky, kteří tvořili převážnou část populace uživající počitač.

Dávkové zpracování nevyhovovalo americké univerzitě MIT[1], která v roce 1961 zahájila vývoj systému CTSS[2] [1]. Hlavním cílem bylo vytvořit operační systém, jenž by mohl interaktivně pracovat s uživatelem. MIT potřebovala pro svůj operační systém specializovaný hardware, který by podporoval ovládání více uživateli v jeden okamžik a požádala o podporu IBM. Ta tehdy nepovažovala projekt za perspektivní, neboť po podobných věcech neexistovala poptávka a vývoj platformy zamítla.

Později si IBM uvědomila využitelnost této technologie a v roce 1964 zahájila vývoj vlastního systému CP/CMS[3]. Záměrem bylo implementovat platformu, která by zvládala zpracovávat úkony s ohledem na co nejefektivnější sdílení počítačových zdrojů mezi velkými skupinami uživatelů. První model na této platformě byl CP-40, který se tak stal historicky prvním hypervizorem, tj. softwarovou mezivrstvou mezi fyzickým hardwarem a operačním systémem [5]. Tato vrstva umožňovala spustit více operačních systémů na jednom počítači. Poprvé se tak objevuje pojem virtuální stroj.

CP-40 byl využíván pouze v laboratořích Cambridgeské univerzity (která se podílela na jeho vývoji). V roce 1968 byl tento model nahrazen novější verzí CP-67/CMS, jež byla jako

[1] Massachusetts Institute of Technology – Soukromá výzkumná univerzita v USA

[2] Compatible Time Sharing System – jeden z prvních operačních systémů umožňující souběžnou práci více uživatelů

[3] Control Program / Conversational Monitor System – první systém umožňující vytváření nezávislých instancí virtuálních strojů

první použita pro komerční účely. Uvedení tohoto modelu na trh způsobilo revoluci v IT sféře, neboť na jednom fyzickém počítači mohlo pracovat více uživatelů nezávisle na sobě [14]. Tím byl vývoj počítačových programů do značné míry urychlen.

V průběhu 70. a 80. let se dále možnosti virtualizace rozrůstaly díky novým technologiím jako například virtualizace paměti, segmentace paměti a vícenásobná disková pole (RAID). Zároveň vznikal konkurenční boj mezi korporacemi, které se snažily přenést virtualizaci i na pole desktopových počítačů. V roce 1988 vydala firma Insignia Solutions program SoftPC, který umožňoval spustit systém DOS na platformách od Sun a Apple Macintosh. Dalším virtualizačním softwarem se v 90. letech stal VirtualPC od společnosti Connectix, který byl později odkoupen společností Microsoft a byl přejmenován na Windows Virtual PC.

V současnosti je virtualizace velmi rozšířená nejen v komerční, ale i v soukromé sféře, neboť virtualizační software je volně dostupný. Standardem v oblasti virtualizace jsou řešení firmy Oracle (VirtualBox), Microsoft (Hyper–V) a VMware (VMware Workstation).

2.2 Důvody nasazování

Důvodů pro nasazení virtualizační technologie existuje velmi mnoho. Virtualizace je nasazována zejména ve firmách, kde jsou kladeny požadavky na stálý provoz jistých služeb (např. webový server, mailový server a jiné), jednodušší operace údržby, snížení nákladů na provozování hardwarových serverů a mnoho dalších [11].

Mezi nejčastější důvody patří lepší využití hardwarových zdrojů fyzického serveru. Současné serverové počítače jsou několikanásobně výkonnější než jsou požadavky softwaru. To má za následek to, že servery bývají vytíženy na 5-10% výpočetní kapacity. Tím pádem firmy, které tyto servery provozují utrácejí za 90% výkonu serveru zbytečně. Při použití virtualizace je možné provozovat několik virtuálních serverů na jednom stroji, a tím zvýšit využití jeho zdrojů. S tím souvisí i úspora prostoru a energie. Dokud nebyla možnost virtualizace, musely firmy pro každou úlohu pořizovat nový server. Existují případy, kdy instituce vlastnila přes 120 fyzických serverů. Po uvedení virtuální infrastruktury do provozu byl jejich počet snížen na cca 40 [2]. Bylo tak docíleno nezanedbatelné úspory financí, neboť výrazně poklesla spotřeba elektrické energie. Zároveň bylo ušetřeno za klimatizační jednotky, které jsou v serverovnách z důvodů udržování konstantní teploty kolem 15–20 stupňů Celsia, což je optimální pracovní teplota serveru. Díky nižšímu počtu serverů nebylo vyprodukováno takové množství tepla a klimatizační jednotky spotřebovaly méně energie na udržení požadované teploty.

Dalším důležitým důvodem je zvýšení provozuschopnosti, na kterou má mimo jiné vliv také hardwarový výkon. Velké množství firem používá na svých serverech tzv. kritické aplikace, tzn. software, jehož činnost je pro firmu stěžejní a obvykle jsou takové aplikace v provozu 24 hodin denně. Mezi takový software patří například databáze, webový server, mailový server a další interní aplikace, které firma používá. Tyto služby běžně fungují na starším hardwaru, který již nemusí být podporovaný výrobcem nebo není možné provést upgrade potřebných komponent pro zvýšení výkonu. Postupem času dojde k situaci, kdy fyzický stroj nebude pro firmu dostačující, ať už z pohledu nedostatečné dostupnosti nebo nízkého výkonu, který by způsoboval přetěžování serveru. V takovém okamžiku bude nezbytný přesun systému včetně veškerých dat a konfigurace. Nejrychlejší a nejjednodušší způsob je použití virtualizace, která umožňuje snadnou přenositelnost systému na jakýkoliv hardware podporovaný virtualizačním prostředím bez nutnosti zásahu do přenášeného systému.

S využíváním kritických aplikací souvisí operace spojené s údržbou systému, ale také zotavení systému z havárie. Ve firmách velmi často nastává okamžik, kdy musí být provedena údržba nebo aktualizace právě kritické aplikace nebo dokonce celého systému. V takových případech musí být server na dobu údržby, která je obvykle několik hodin nebo dokonce celý den, odstaven a služba je během této doby nedostupná. To může mít za následek negativní dopad na výkonnost celé firmy. Aktualizovanou kritickou aplikací může být například systém pro správu verzí (CVS[4]). Suspendace takového systému může mít fatální následky. Během doby, kdy je systém odstaven, je všem vývojářům znemožněna jakákoliv práce s repozitáři, což do značné míry omezuje vývojářskou činnost. To ale není jediné omezení, které může být způsobené nedostupností CVS. Většina firem navíc provádí automatizované testování pomocí nástroje pro průběžnou integraci (angl. *continuous integration*[5]), který také využívá systém pro správu verzí pro vytvoření testovacího prostředí vyvíjeného softwaru. Na příkladu vídíme, že i běžná údržba může způsobit dočasnou paralyzaci firmy. Tomuto nežádoucímu důsledku je možné zabránit použitím virtualizace. Pokud je server, kde je nainstalována kritická aplikace, virtualizovaný, je možné za běhu zálohovat obraz současného stavu. Před provedením údržby tak lze v minimalním čase (v řádu jednotek minut) dočasně nahradit stávající virtuální server posledním zálohovaným obrazem, následně provést požadovanou operaci údržby a úvést původní server zpět do chodu. Podobný postup je možné aplikovat i v případě havárie virtuálního serveru. Během běžného provozu je průběžně zálohován stav systému ve formě *snapshotu*[6]. Pakliže dojde k nečekanému selhání virtuálního serveru s kritickou aplikací, je z tohoto obrazu automaticky vytvořen a spuštěn nový virtuální server bez nutnosti jakéhokoliv manuálního zásahu. Proces zotavení tak trvá několik minut a havárie může být analyzována příslušnými zaměstnanci bez dopadu na chod společnosti.

Výše popsané důvody zavádění virtualizace jsou prospěšné zejména pro provoz celé společnosti. Jde zejména o rychlejší zotavení po havárii, jednodušší migrace systémů na jiný fyzický hardware a zvýšení využití hardwarových prostředků. Virtualizace je ale užitečná i pro samotné vývojaře, kteří mohou mít na svém pracovním počítači několik virtuálních strojů s různými systémy vedle sebe. Vývojář tak může implementovat aplikaci například na operačním systému Windows a zároveň ji testovat na virtuálním počítači se systémem Linux. Vývojář může mít na jednom fyzickém stroji tolik virtuálních strojů, kolik mu dovolují jeho zdroje. V současnosti je doporučeno minimálně 2 GB operační paměti k běžnému provozování jednoho virtuálního stroje. Lze také jednoduše testovat síťové aplikace, neboť ve virtuálním prostředí je možné vytvořit virtuální LAN[7].

[4] Concurrent Versions System

[5] Vývojářská metoda, jejíž základem vytváření tzv. *buildů* (zkompilovaných verzí softwaru). Příkladem takového softwaru je např. Jenkins nebo TeamCity

[6] Kopie systému zachycující jeho stav v určitém časovém okamžiku

[7] Local Area Network

Kapitola 3

Techniky virtualizace počítačů

Způsoby, kterými je možné virtualizovat počítač existuje celá řada. Tyto způsoby se mezi sebou liší zejména v míře abstrakce od hardwaru, na kterém je virtuální stroj provozován. Každá z těchto technik má jisté kladné i záporné vlastnosti. Ty jsou relevantní, když se rozhodujeme, jakou metodu zvolit pro konkrétní aplikaci. V této kapitole budou popsány nejčastěji používané techniky virtualizace a to jak pro virtualizaci hardwaru, tak softwaru. U každého způsobu budou uvedeny výhody a nevýhody, společně s příkladem vhodné aplikace dané metody.

3.1 Emulace

Emulace je druh virtualizace, při které dochází k virtualizaci hardwarových komponent za účelem vytvoření jiné hardwarové platformy. Virtuální stroj tedy simuluje veškerý hardware pomocí softwaru. Princip této metody spočívá v překládání strojových instrukcí hostovaného[1] systému na strojové instrukce hostitelského[2] stroje. Tím se dostáváme k největšímu přínosu emulace, který spočívá v tom, že je možné uvnitř jednoho operačního systému spustit jiný operační systém, který není na hostující platformě podporován. Další, neméně důležitou vlastností emulace je skutečnost, že hostované systémy ani aplikace běžící ve virtuálním stroji není nutné nijak modifikovat. Emulace je využívána v situacích, kdy je požadována tvorba softwaru pro platformu, která již není dostupná. Tato technologie umožňuje dokonce emulovat vícejádrový procesor na jednoprocesorovém systému.

Kvůli faktu, že emulace virtualizuje rozdílnou platformu, není možné využít hardwarovou podporu virtualizace, kterou v sobě mají zakomponovanou novější řady procesorů od firem Intel a AMD. To má za následek vysokou režii pro hostitelský operační systém, neboť musí být každá instrukce dynamicky přeložena na instrukci (nebo posloupnost instrukcí) pro procesor na emulované platformě. Emulace se tak řadí mezi nejpomalejší metody pro virtualizaci. Další nevýhodou emulace může být nedostatečná dostupnost informací o emulované platformě, neboť ta může být součástí duševního vlastnictví firmy, která hardware navrhovala. Mezi nejrozšířenější emulační software patří DOSBox, QEMU, Bochs a jiné.

QEMU patří mezi nejrychlejší emulátory. Je vyvíjený pro systémy Linux, Windows a některé Unixové platformy [15]. Podporuje virtualizaci architektur x86, x64, MIPS, ARM, PowerPC, SPARC a dalších. Lze s ním pracovat ve dvou režimech. První, uživatelský režim, umožňuje spouštět v hostitelském operačním systému aplikace, které byly původně navr-

[1] Systém, který virtualizujeme (angl. *guest*)
[2] Systém, na kterém běží fyzický hardware (angl. *host*)

ženy pro jinou architekturu. Druhým režimem QEMU je plná virtualizace jádra operačního systému. Jak název napovídá, pomocí tohoto režimu lze spustit např. Android (navržený pro architekturu ARM) na procesoru řady x64.

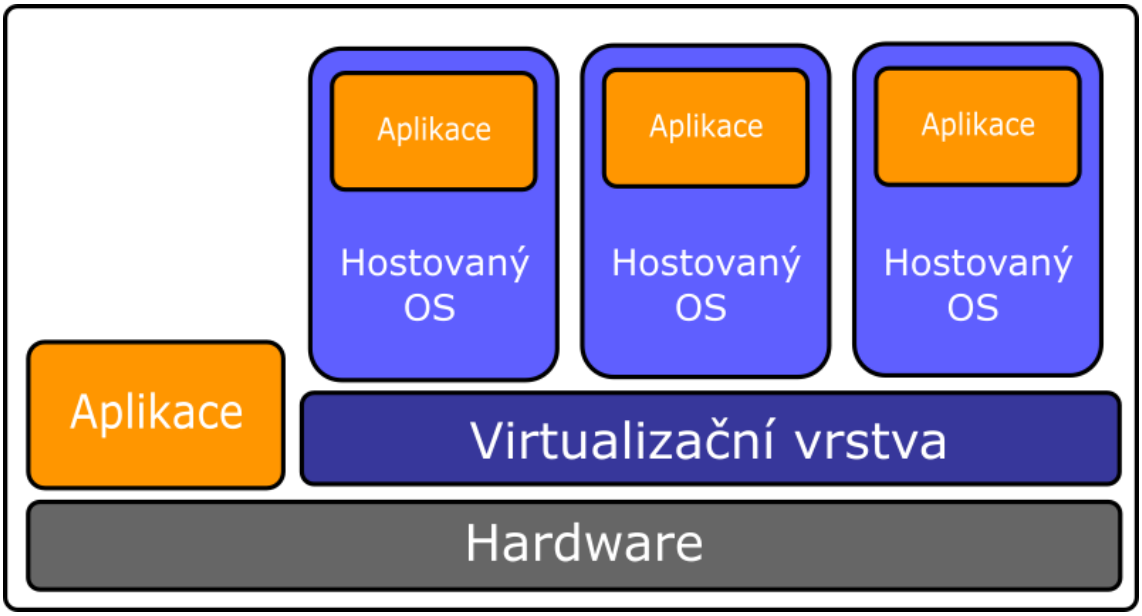
3.2 Plná virtualizace

Plná virtualizace umožňuje *nemodifikovanému* hostovanému operačnímu systému, včetně všech jeho nainstalovaných aplikací, běžet ve speciálním prostředí vytvořeném v hostitelském OS. Simulovány jsou pouze nezbytné hardwarové komponenty fyzického stroje. Většinu operací vykonávaných hostovaným OS není nutné modifikovat, neboť jsou prováděny přímo na fyzickém hardwaru. Tím je docíleno menší režie a vyššího výkonu než u emulace. Plná virtualizace na platformách x86 je poměrně nová technologie, která začala být reálně využívaná až s příchodem technologií AMD-V a Intel VT-x v roce 2006. Do té doby byla virtualizace systémů s architekturou x86 velice nefektivní, protože neexistovala dostatečná hardwarová podpora virtualizace v procesoru. Musely tak být implementovány složité programové algoritmy, které tento nedostatek kompenzovaly. Tím docházelo ke značnému zpomalení při provádění operací na hostovaném operačním systému. Tato rozšíření od Intelu a AMD implementují privilegovaný režim procesoru. V tomto režimu se procesor může nacházet ve dvou úrovních - privilegovaný a neprivilegovaný stav. V privilegovaném stavu mohou být prováděny strojové instrukce, které mohou ohrozit chod počítače a běží v něm pouze jádro operačního systému. Uživatelské programy jsou spouštěny v neprivilegovaném režimu, čili procesor nemůže modifikovat důležité části systému během vykonávání kódu.

Problém může nastat ve chvíli, kdy virtuální operační systém potřebuje provést privilegovanou operaci. Jak bylo napsáno výše, v privilegovaném režimu může běžet pouze operační systém, čili virtualizovaný OS běží na virtuálním stroji v privilegovaném režimu. Z pohledu fyzického stroje je ale virtuální stroj pouze běžící uživatelskou aplikací. Po provedení privilegované operace ve virtuálním stroji tedy musí zůstat stav fyzického stroje nezměněný a zároveň musí dojít ke změně stavu virtuálního stroje. K dosažení tohoto cíle je využíván hypervizor, který musí zachytit požadavek o provedení privilegované instrukce, přečíst aktuální stav virtualizovaného systému a tuto operaci softwarově emulovat. Následně jsou ve virtuálním operačním systému provedeny stejné změny, jaké by provedl reálný hardware. S tímto postupem je spojena vyšší režie, která má negativní dopad na výkon virtuálního stroje.

Výhodou plné virtualizace je možnost konfigurace zdrojů a některých vlastností hardwarových komponent virtuálního stroje. Můžeme např. nastavit velikost operační paměti, maximální využití fyzického procesoru virtuálním strojem, typ a kapacitu pevného disku, vytvořit libovolný počet vstupně/výstupních sběrnic a další. Se znalostí parametrů fyzického stroje je snadné vytvořit libovolný počet virtuálních strojů, ovšem s podmínkou, že součet přidělených zdrojů musí být menší než zdroje fyzického počítače. Jinak by byl stroj natolik vytížen, že by nezvládal zpracovávat požadavky od virtuálních strojů, a tím pádem ani od uživatelů operujících na nich. Neméně důležitou výhodou je možnost souběžné práce více uživatelů na jednom fyzickém stroji. Virtuální stroje jsou odděleny od fyzického hardwaru virtualizační vstvou, tudíž jsou pro sebe navzájem transparentní a nevznikají mezi nimi žádné konflikty.

Další pozitivní vlastnost této metody spočívá ve snadné přenositelnosti aplikací či dokonce celých operačních systémů. Stačí pouze vytvořit obraz (tzv. *snapshot*) virtuálního stroje, přenést jej na jiný počítač a tam jej převést zpět do virtuálního prostředí bez nutnosti jakékoliv úpravy. Nutnou prerekvizitou pro tento postup je nainstalovaný stejný ma-



Obrázek 3.1: Schéma plné virtualizace.

nažer virtuálních strojů na obou počítačích. Dále, aby bylo možné virtuální stroje takto přenášet, je nezbytné, aby měly oba fyzické stroje stejný procesor, resp. procesor se stejnou instrukční sadou. Při použití plné virtualizace totiž není možné emulovat jiné architektury, což je společně s pomalejšími vstupně/výstupními operacemi nevýhodami této technologie.

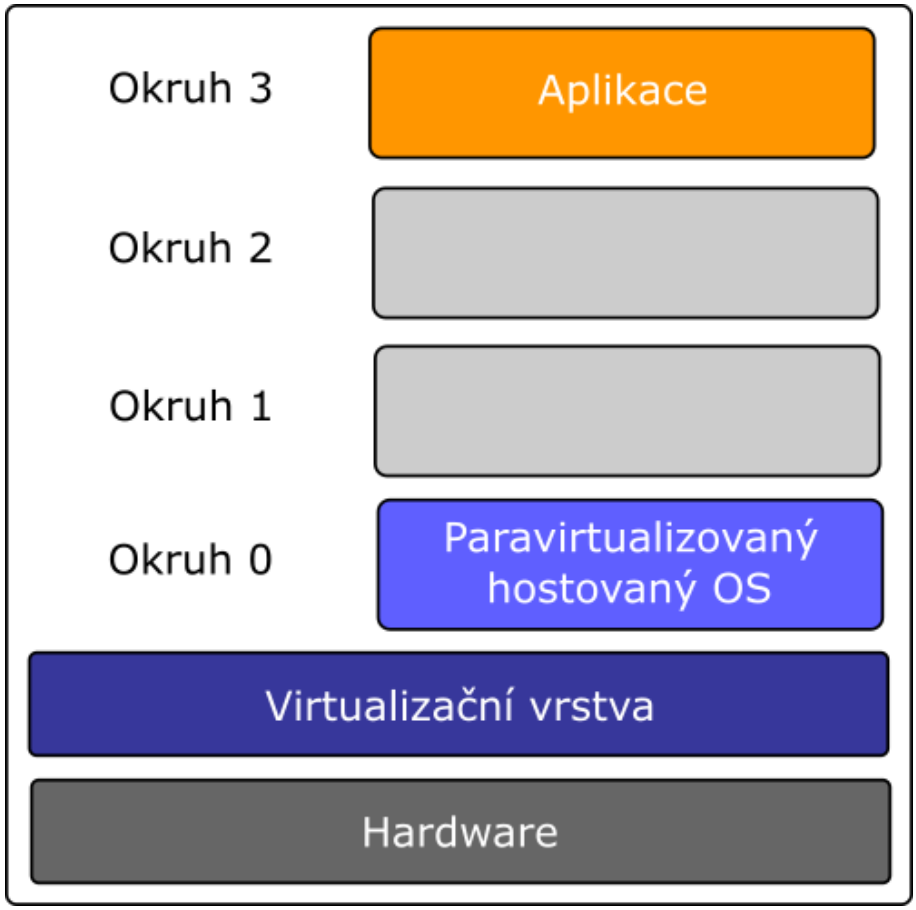
Podpora této technologie se nachází zejména v moderních virtualizačních nástrojích, které pro virtualizaci používají hypervizora. Nejznámějšími produkty jsou VirtualBox od firmy Oracle, VMware Workstation, Microsoft Virtual PC. Nejčastěji jsou využívány v komerční sféře kvůli lepší utilizaci výkonu, snadnější migraci systému na jiný hardware, rychlejší obnovení po havárii a snadnější zálohování. Je možné i použití pro soukromé účely, neboť jsou tyto produkty distribuované jako *freeware*[3].

3.3 Paravirtualizace

Paravirtualizace je v jistých aspektech metoda podobná plné virtualizaci. Virtuální stroj nemusí provádět emulaci veškerého hardwaru, což znamená, že z hardwarového pohledu si musejí být fyzický i virtuální stroj velmi podobné – tento požadavek je kladen zejména na procesor, který musí být shodný, nanejvýš smí pracovat na mírně nižší frekvenci. Virtualizace, která využívá tuto techniku není úplná, neboť operační systém ví o skutečnosti, že běží ve virtuálním prostředí. To ale vyžaduje úpravu jádra virtuálního operačního systému, která náhradí systémová volání za volání funkce hypervizora, tzv. *hypercall*, jenž komunikuje přímo s virtualizační vrstvou. Tato volání lze využít i pro operace spojené se správou paměti nebo obsluhou přerušení.

Kvůli faktu, že virtuální systém má povědomí o tom, že běží ve virtuálním prostředí, odpadá režie spojená s binárním překladem instrukcí[4] jako tomu bylo u plné virtualizace. Požadavky uživatelských aplikací ve virtuálním stroji jsou obsluhovány fyzickým hardwarem,

[3] Druh softwaru, který je dostupný zdarma
[4] Emulace instrukční sady procesoru jinou instrukční sadou



Obrázek 3.2: Schéma paravirtualizace.

čímž pádem může být výkon virtuálního stroje velmi blízký výkonu fyzického. Jedním z problémů, které musí paravirtualizace řešit je přístup do paměti. Uživatelské aplikace totiž nesmějí zapisovat do částí paměti, kde jsou uložena data, která jsou pro systém kritická. Pro tento účel je nedostatečné mít v procesoru podporu pouze pro privilegovaný a neprivilegovaný režim jako u plné virtualizace, neboť hypervizor musí běžet na vyšším stupni ochrany, aby nemohl operační systém ovlivnit jeho stav. Je tedy nezbytná podpora více úrovní ochran v procesoru. Procesory od firmy Intel podporují tzv. *ochranný režim*, který používá až 4 úrovně ochrany, tzv. okruhy (angl. *rings*). Každý okruh má přidělená jiná oprávnění, přičemž nejvyšší privilegia náleží programům běžícím v okruhu 0 a nejnižší programům běžícím v okruhu 3. Typicky je operační systém spuštěn v režimu s nejvyšším oprávněním, a naopak, uživatelské aplikace s nejnižším. Okruhy 1 a 2 zůstávají buď nevyužité, nebo jsou použity pro ovladače zařízení. Chráněný režim umožňuje souběžný běh více aplikací, neboť každá z nich má přidělenou jistou část operační paměti, která je s adresními prostory ostatních aplikací disjunktní.

Jak bylo řečeno výše, hypervizor musí běžet s vyššími oprávněními než hostující i hostovaný operační systém. To je možné zařidit dvěma způsoby. U prvního z nich běží hypervizor přímo na fyzickém vybavení počítače. Hostitelský i hostovaný operační systém pak operují s nižšími přístupovými právy než virtulizační vrstva. U druhého způsobu je hypervizor součástí virtuálního stroje. Aby měl hypervizor stále vyšší oprávnění než operační systém, je umístěn do chráněného okruhu 0, zatímco OS je posunut o jednu úroveň níže.

Velkou výhodou paravirutalizace je vlastnost, že požadavky uživatelských aplikací jsou vykonávány přímo na hostitelském hardwaru. To je umožněno kvůli tomu, že operační systém obsahuje speciální aplikační rozhraní (API[5]), které tuto funkci umožňuje. Tím odpadá nutnost použití binárního překladu instrukcí, který je využíván u plné virtualizace a virtuální počítač lépe využívá zdroje fyzického stroje. Aby mohlo být toto API použito, musí být upraveno jádro hostovaného OS. S tím také souvisí omezení této metody. Modifikace jádra s sebou nese komplikace s nasazením, zejména u komercializovaných operačních systémů, které není možné libovolně upravovat. Z tohoto důvodu je paravirtualizace rozšířená především na linuxových platformách.

Nejznámějšími produkty využívající paravirtualizaci jsou VMware Workstation a Xen. Podpora paravirtualizace taktéž přibyla v červenci roku 2015 ve virtualizačním nástroji VirtualBox od Oracle.

3.4 Částečná virtualizace

Při použití částečné virtualizace je vytvořeno několik instancí hardwaru, který je součástí fyzického stroje. Využívá se zejména pro virtualizaci adresních prostorů aplikací. Existuje zde podpora sdílených zdrojů počítače mezi procesy a vzájemná izolace těchto procesů. Pomocí této metody ale není možné virtualizovat celý stroj včetně operačního systému. Aby mohla částečná virtualizace fungovat, je zapotřebí relokační hardwarová komponenta, která převádí logickou adresu na fyzickou. Takovou komponentou může být např. MMU[6], která bývá součástí procesoru.

Částečná virtualizace byla významným mezníkem v době, kdy vznikala plná virtualizace. Poprvé byla použita v experimentálním systému IBM M44/44X v 60. letech 20. století. Přesto je stále využívána v moderních operačních systémech (Microsoft Windows nebo Linux). Její výhoda spočívá ve snadné implementaci a sdílení počítačových zdrojů vícero uživateli.

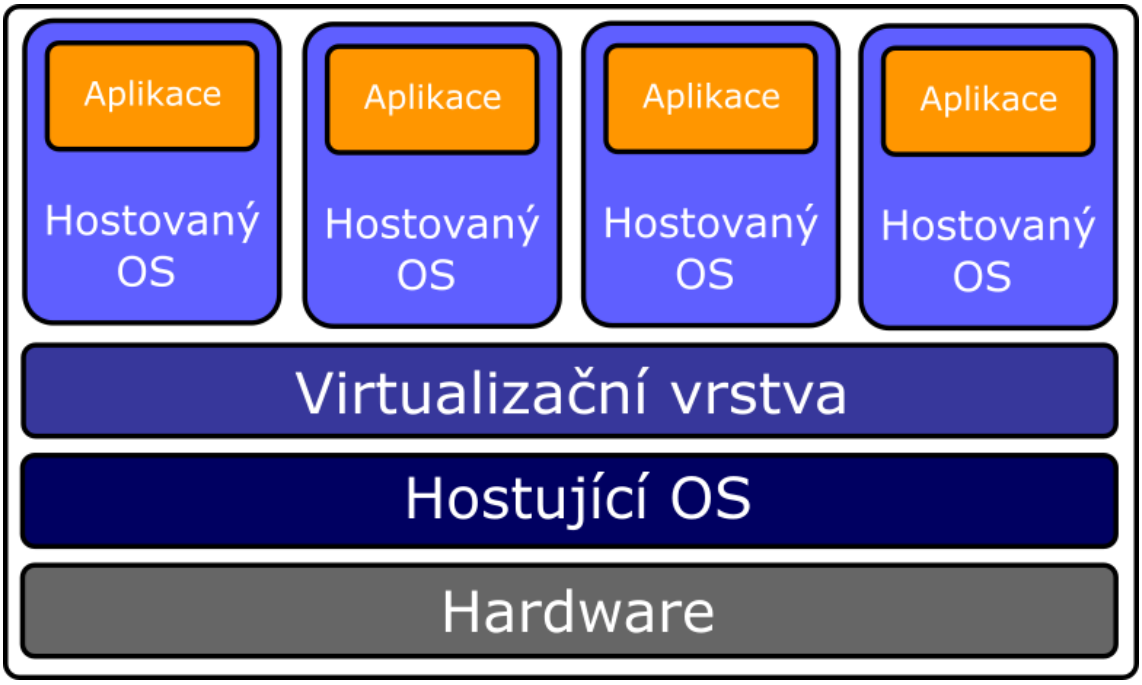
3.5 Virtualizace na úrovni operačního systému

Virtualizace na úrovni operačního systému je technika, při které jádro operačního systému umožňuje provozovat několik izolovaných instancí virtuálního prostředí [17]. Jinými slovy, je možné, aby více uživatelů pracovalo na jediném hostujícím operačním systému nezávisle na sobě. Tyto instance tedy nad sebou mají hostitelský operační systém a obsahují sadu rozhraní, se kterými aplikace uvnitř virtuálního operačního systému komunikují s fyzickým hardwarem. Toto rozhraní je implementováno tak, aby vytvářelo iluzi, že virtualizované OS běží na samostatném fyzickém stroji a mají tedy k dispozici veškeré jeho zdroje. Instance virtuálních operačních systémů bývají v literatuře označované jako tzv. kontejnery.

Virtualizační vrstva, která provádí obsluhu virtuálních strojů, je umístěna mezi operační systém na fyzickém stroji a virtuálními systémy. Její funkcionalita je ale oproti výše popsaným technikám velmi omezená, neboť jde o softwarovou virtualizaci a není potřeba emulovat žádnou hardwarovou komponentu. Většina režie tedy spadá pod správu hostujícího operačního systému. Tato režie je ale velmi malá, protože virtualizované operační systémy jsou stejné jako hostující OS. Díky tomu virtualizace na úrovni operačního systému patří

[5] Application Programming Interface – programové rozhraní umožňující komunikaci externího programu s programem, jenž API nabízí

[6] Memory Management Unit



Obrázek 3.3: Schéma virtualizace na úrovni operačního systému.

mezi nejrychlejší metody virtualizace. Mezi další výhodu patří velmi rychlá aktualizace systému. Pokud potřebujeme aktualizovat jádro operačního systému, stačí aktualizovat pouze hostitelský operační systém. Pakliže jsou virtuální stroje obrazy toho fyzického, využívají všechny jeho jádro a není potřeba aktualizovat každý zvlášť jako u plné virtualizace, či paravirtualizace, kde má každý virtuální stroj svou verzi operačního systému.

Mezi omezení této technologie patří zejména fakt, že hostitelský i hostované operační systémy musejí být shodné nebo podobné, co se týče verze jádra. V případě, že hostitelským systémem bude operační systém Linux, tak virtualizované operační systémy musí být taktéž Linuxové. To může způsobovat problém, pokud vyžadujeme běh různých aplikací ve virtuálních prostředích, neboť ty mohou být certifikovány pouze na určité verze operačních systémů.

Využití této technologie je ideální zejména pro firmy poskytující web hosting, neboť mohou mít více virtuálních serverů běžící na jednom fyzickém stroji. Tyto virtuální servery jsou na sobě nezávislé, čili nedochází ke kolizím. Mohou také velmi snadno aktualizovat všechny virtuální systémy, protože, jak bylo popsáno výše, při aktualizaci hostujícího operačního systému je změna okamžitě promítnuta i do kontejnerů.

3.6 Aplikační virtualizace

Aplikační virtualizace je druh virtualizace, která umožňuje spouštět aplikace bez nutnosti jejich instalace v klasickém smyslu slova. Aplikace tak běží ve vlastním virtuálním prostředí a nijak neovlivňují operační systém či ostatní aplikace. Pomocí této techniky je také možné mít spuštěno více instancí nějaké aplikace na jednom fyzickém stroji. Úkolem virtuálního prostředí je, aby vykonávání kódu aplikace vytvářelo dojem, že pracuje s reálnými soubory a systémovými částmi, např. registry.

Plná aplikační virtualizace vyžaduje speciální virtualizační vrstvu, která plně nahradí

běhové prostředí[7], které jinak poskytuje operační systém. Virtualizační vrstva odchytává požadavky aplikace, např. čtení či zápis do souboru, a přesměrovává je do virtualizovaného místa na disku, obvykle do jediného souboru. Tato skutečnost, že aplikace přistupuje k virtuálním zdrojům namísto fyzických, je před ní skryta a aplikace se tedy domnívá, že provádí změny v reálném systému. Kvůli faktu, že aplikace během své činnosti mění jediný virtualizovaný soubor na disku, je snadné tuto aplikaci spustit na jiném počítači, kde zároveň mohou běžet jiné aplikace, které by byly s onou přenášenou aplikací v konfliktu.

Virtualizace aplikací je nejčastěji realizována dvěma způsoby. U prvního z nich běží aplikace na vzdáleném serveru a koncoví uživatelé s ní operují pomocí RDP[8]. Vzdálené aplikace tak mohou být zcela integrovány v uživatelově počítači a jevit se tak jako lokálně nainstalovaná aplikace. K tomuto postupu je potřeba udržovat neustálé síťové spojení mezi serverem a uživatelským počítačem. Druhou možností je použití tzv. streamování. U této metody je na požadavek uživatele zahájeno stahování aplikace ze vzdáleného serveru. Po stažení součástí, které jsou nezbytné pro běh, je možné aplikaci spustit. Zbytek aplikace může být dostahován až když jsou potřeba (za běhu). Jakmile je stahování dokončeno, lze s aplikací pracovat i bez připojení k síti. Technologie pro izolaci aplikací zajistí, aby nebyla streamovaná aplikace v konfliktu s žádnými nainstalovanými aplikacemi. Po uzavření streamované aplikace jsou z disku smazány všechny její součásti. Obě tyto metody jsou výhodné v tom, že jsou spravovány centrálně. Virtuální aplikace stačí nainstalovat nebo aktualizovat pouze jednou na vzdáleném serveru a změny se ihned projeví i na stanicích koncových uživatelů. Je usnadněno i licencování aplikací, neboť aplikace je nainstalována pouze na jediném stroji a nemusí být pořizovány licence pro každého uživatele zvlášť.

[7] Běhové prostředí
[8] Remote Desktop Protocol

Kapitola 4

Nástroje pro virtualizaci

Virtualizace je v dnešní době velmi žhavým a diskutovaným tématem. V IT sektoru vzniká konkurenční boj společností, které se rozhodly do této oblasti přispět svými myšlenkami. Existuje tak mnoho nástrojů pro virtualizaci, které se liší v použité technologii virtualizace, otevřenosti kódu, podpoře různých funkcí a dalších vlastnostech. V této kapitole budou pro úplnost krátce zmíněny existující virtualizační nástroje. Nástroji VirtualBox bude věnována celá další kapitola, neboť jde o software, pro který bude napsána aplikace, jež je předmětem této práce.

4.1 VMware

VMware je společnost věnující se tvorbě softwaru v oblasti virtualizace již od roku 1998, kdy byla založena. Jako první se jí podařilo komerčně virtualizovat platformu x86, díky čemuž se stala středem pozornosti v komerčním sektoru a řadí se tak mezi největší poskytovatele virtualizačních nástrojů. VMware vydává své produkty na platformy Microsoft Windows, Linux a Mac OS X. Mezi její nejvýznamější produkty patří VMware Workstation, WMware Player, WMware Fusion a další.

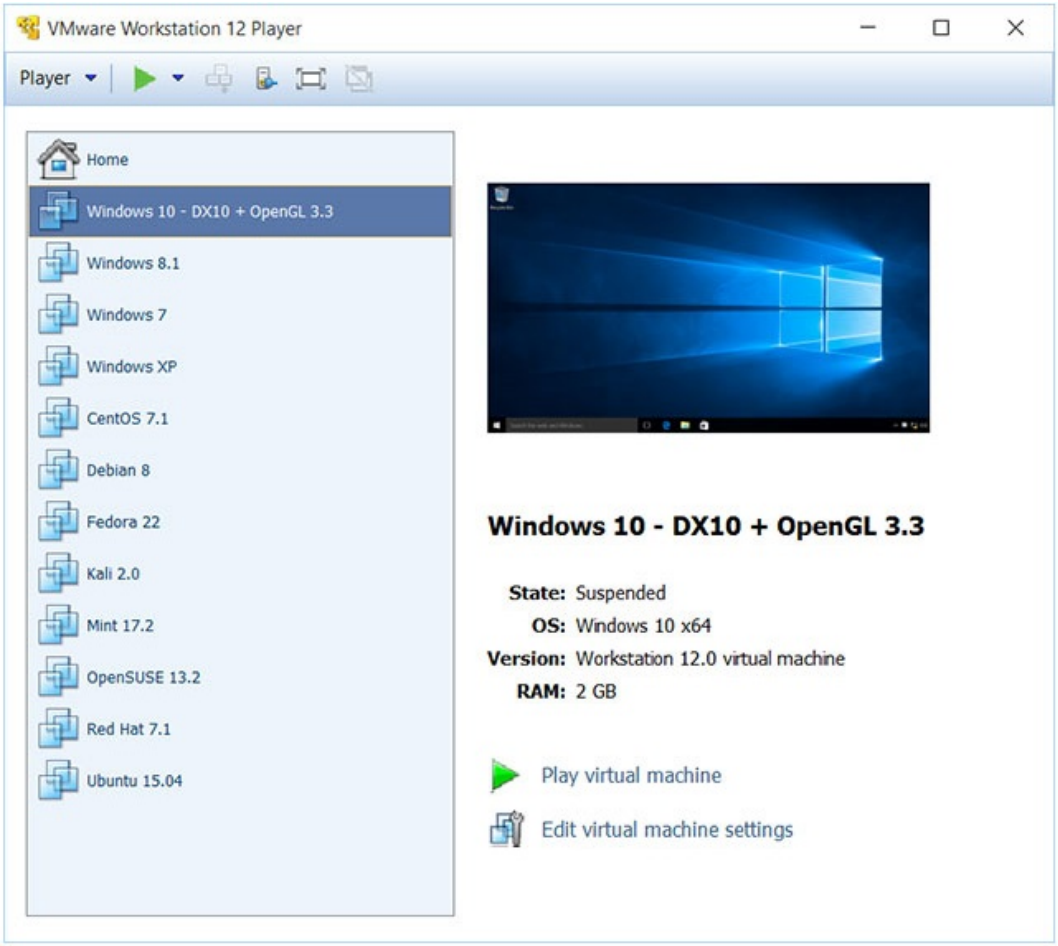
4.1.1 VMware Workstation

Workstation je nejčastěji využívaný produkt od firmy VMware v komerční sféře. Je určený k vytváření a provozování několika virtuálních strojů na jednom uživatelském počítači či serveru. Každý virtuální stroj má svůj operační systém, na kterém mohou být spuštěny podporované aplikace. VMware tento produkt vydává na systémy Microsoft Windows, Linux a Mac OS X.

4.1.2 VMware Workstation Player

Player je další hojně využívaný virtualizační software od firmy VMware. Využívá stejné virtualizační jádro jako VMware Workstation. Přes VMware Workstation Player je možné spouštět existující virtuální apliance[1] nebo vytvářet nové virtuální stroje, pro jejichž běh je nutné nejprve nainstalovat operační systém z instalačního média nebo obrazu. V případě, že je potřeba použít obraz již existujícího stroje, je možné tento obraz do aplikace nahrát, a tím vznikne nakonfigurovaný virtuální stroj. Zároveň je možné importovat obrazy virtuálních

[1] Předkonfigurovaný obraz virtuálního stroje, který je možné importovat do virtualizačního nástroje, který jej podporuje



Obrázek 4.1: Uživatelské rozhraní Workstation 12 Player, převzato z [12].

strojů, které nebyly vytvořené žádným produktem od VMware. Podporovány jsou formáty *.ova* a *.ovf* soubory, které jsou vytvořeny programem VirtualBox, nebo *.vmc* soubory, jež jsou vytvořeny produktem Microsoft Virtual PC.

Užitečnou funkcí je *Easy install*. Tato funkce dovoluje provádět instalaci operačního systému na virtuální stroj bez nutnosti přítomnosti uživatele. Tento způsob instalace musí být integrovaný v operačním systému. VMware Workstation Player pak vytvoří soubor, který obsahuje konfiguraci instalace (u systémů Windows jde o *.oem* soubor) a ten je následně použit pro automatické instalaci.

4.1.3 VMware vSphere

Tento produkt je velmi používaný společnostmi, které potřebují provozovat velké množství virtuálních serverů, datových center a virtuálních sítí pomocí cloudových technologii. Nabízí vysokou míru konfigurace a správy celé infrastruktury virtuálních strojů ve velkém měřítku. Mezi největší konkurenční produkt patří Hyper-V od společnosti Microsoft. Nástroj vSphere umožňuje provádět migraci strojů za běhu, a tím výrazně zvyšuje dostupnost služeb, které běží na virtuálních serverech. Tento fakt má pozitivní vliv na provozuschopnost firmy, neboť při migraci je nedostupnost služby minimalizována na jednotky minut. To je užitečné zejména při migraci serveru, na kterém běží nějáká kritická aplikace.

4.2 Microsoft

Microsoft je korporátní společnost, která se zabývá tvorbou hardwarových i softwarových produktů již od 70. let minulého století. Microsoft je známý především díky jejich produktům Microsoft Windows, Microsoft Office a herní konzoli Xbox.

Na poli virtualizace serverů a desktopových počítačů je Microsoft velkým hráčem a nabízí několik produktů. Mezi nejvýznamější patří Microsoft Hyper-V, Microsoft Virtual PC a Microsoft Virtual Server.

4.2.1 Microsoft Hyper-V

Microsoft Hyper-V je virtualizační nástroj určený pro platformy Microsoft Windows. Na trh byl uveden v roce 2008. S postupem času byly vydávány nové verze tohoto produktu, které jsou v současnosti celkem 4. Různé verze jsou integrovany v produktech Windows Server 2008, Windows Server 2008 R2, Windows Server 2012 a Windows Server 2012 R2.

Hyper-V umožňuje vytvářet celou virtuální infrastrukturu velkých podniků. Jeho použitím dochází ke zvýšení spolehlivosti, bezpečnosti a lepší dostupnosti a udržovatelnosti virtuálních serverů. Velkou výhodou Hyper-V je možnost přesunu virtuálních serverů na jiné fyzické stroje za běhu, a tím ušetřit nemalé množství času spojené s instalací a konfigurací nového fyzického serveru. Microsoft tuto funkci označuje jako *Live Migration*.

Jeden z velkých přínosů Microsoft Hyper-V je fakt, že je integrovanou součástí Windows Serveru, což firmám usetří náklady spojené s licenováním virtualizačních nástrojů jako je tomu u VMwaru nebo Citrixu.

4.2.2 Microsft Virtual PC

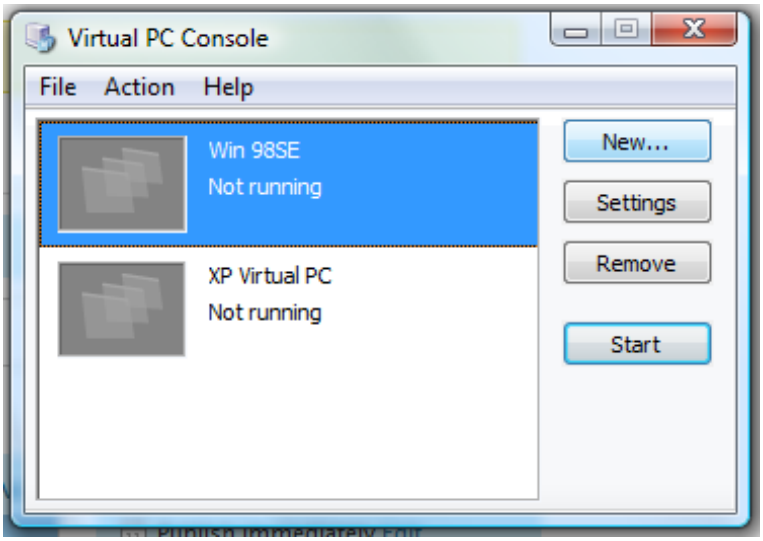
Jde o velmi jednoduchý virtualizační nástroj, který je bezplatnou součástí systémů Microsoft Windows od roku 2006. Nejnovější verzi tohoto produktu je možné spustit pouze na systémech Windows 7 a novějších, přičemž na těchto systémech není možné vytvářet virtuální počítače se systémem, které byly vydány dříve než Windows XP. Virtual PC je zaměřený pro virtualizace zejména operačních systémů od společnosti Microsoft, není tedy zajištěna podpora všech Linuxových distribucí. Pro spuštění stroje se starším systémem je možné využít starší verze tohoto produktu.

Díky faktu, že je Microsoft Virtual PC integrovaný přímo ve Windows, lze jej stále využívát pro zajištění zpětné kompatibility s aplikacemi, které byly vydány pro starší systémy od Microsoftu. Tato funkce ve Windows se nazývá *Režim zpětné kompatibility* a v tomto režimu je možné spustit jakýkoliv spustitelný soubor.

4.2.3 Microsoft Virtual Server

Microsoft Virtual Server je virtualizační nástroj určený zejména pro efektivní správu serverů. V době prvního vydání existovala podpora pouze 32 bitových systémů, později ale bylo toto omezení odstraněno a byla přidána podpora 64 bitových operačních systémů. Od roku 2006 byl vydáván jako *freeware* i pro komerční účely.

K manažerovi virtuálních strojů se přistupuje pomocí webového rozhraní, díky čemuž je správa serveru přístupná z jakéhokoliv počítače, který se nachází ve stejné síti jako fyzický server. Virtual Server podporuje skriptování, čímž je možné tento nástroj automatizovat. Poskytované API je podporované jazyky VBScript, Visual Basic .NET, C++ nebo C# [9]. Zajímavou funkcí tohoto nástroje je *Migration Toolkit*, který umožňuje převést fyzické



Obrázek 4.2: Uživatelské rozhraní Virtual PC, převzato z [13].

servery do virtuálního prostředí, čímž je usnadněna konfigurace virtuálního stroje, kterou by jinak musela provádět zodpovědná osoba.

Microsoft Virtual Server není spouštěn jako klasická aplikace, nýbrž služba v systému Microsoft Windows. Existuje tak možnost spouštění manažera virtuálních strojů hned po startu počítače.

4.3 Citrix

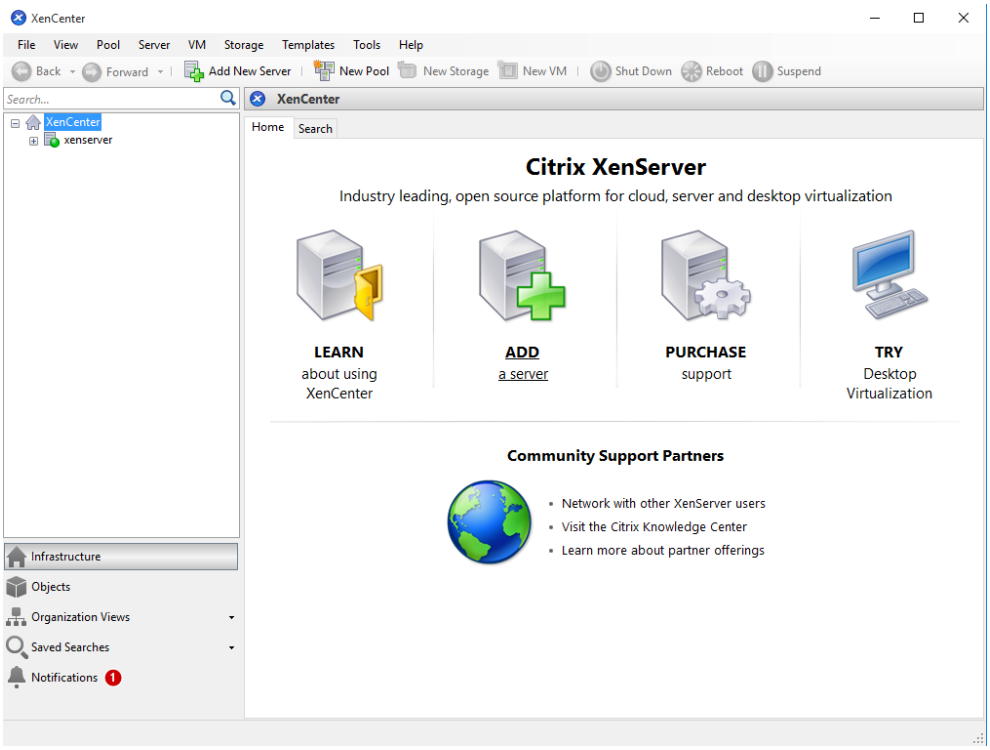
Citrix je nadnárodní společnost, která se věnuje virtualizaci serverů, virtualizací desktopových počítačů a počítačovým sítím od roku 1989. Její produkty jsou podporovány na systémech Windows, Mac OS X a Linux a v současné době jsou jejich produkty využívány více než 100 miliony uživateli celosvětově [3]. Také velice blízce spolupracuje s Microsoftem, kterému dodává pokročilé virtualiační nástroje integrované například do Windows Server 2008 Hyper-V.

Hlavní činností je poskytování licencí na mobilní aplikace a cloudových[2] služeb, díky kterým je možné pracovat odkudkoliv a dokonce na různých typech zařízení. Tato řešení jsou v dnešní době na vzestupu, protože poskytují centralizovaný přístup k aplikacím či desktopovým počítačům, a tím zvyšují jejich bezpečnost. Tyto služby jsou distribuovány zákazníkům skrze produkt XenCenter.

4.3.1 XenServer

XenServer je produkt od společnosti Citrix, který je používán ve velkých firmách a *cloudech* [16]. Jde o virtualizační platformu, která dodává všechny nezbytné součásti pro virtualizaci serverů a datových center. Jeho cílem je zlepšit dostupnost interních služeb a aplikací, které jsou používány ve firmách. Umožňuje spravovat, monitorovat a provádět operace údržby serverů. Administrátoři těchto serverů mohou spravovat stovky virtualizovaných serverů z jediné konzole, kterou lze nainstalovat na jakýkoliv stroj se systémem Windows.

[2]Způsob vývoje softwaru založený na Internetu [4]



Obrázek 4.3: Uživatelské rozhraní XenCenter, převzato z [6].

Důležitou podporovanou funkcí je stejně jako u ostatních virtualizačních nástrojů přesun virtuálních serverů na jiné fyzické stroje bez jakéhokoliv prostoje. Další vlastnosti XenServeru jsou:

4.3.2 XenDesktop

XenDesktop je další virtualizační nástroj od společnosti Citrix, který umožňuje několika uživatelům přistupovat k virtuálním počítačům, které jsou nainstalovány na centralizovaném místě. Uživatelé na tyto počítače mohou přistupovat pomocí různých zařízení a z míst, která nejsou ve stejné síti jako virtuální server. Při vytvoření spojení je instance operačního systému virtuálního stroje přenesena na počítač uživatele. Důležité je zmínit, že instance virtuálního stroj využívá zdroje cílového (uživatelova) zařízení, nikoli virtuálního serveru.

4.3.3 XenApp

Citrix XenApp umožňuje uživatelům využívat aplikace, které jsou nainstalované na vzdáleném počítači se systémem Microsoft Windows z jakéhokoliv místa či zařízení. Pokud je zařízení uživatele kompatibilní se systémem Windows, je virtualizovaná aplikace včetně její konfigurace a dat přeposlána do jeho zařízení. Uživatel pak může s aplikací pracovat i bez připojení k Internetu. V případě, že uživatel nevlastní zařízení kompatibilní s Windows, je instance virtualizované aplikace vytvořena na serveru a cílovému zařízení jsou pravidelně zasílány snímky aktuálního stavu. Uživateli se tedy aplikace jeví jako kdyby byla nainstalována v jeho zařízení. Při tomto použití nelze s aplikací pracovat bez Internetového připojení.

Kapitola 5

Oracle VM VirtualBox

VirtualBox patří mezi rozšířené virtualizační nástroje a je využívaný v komerčním i soukromém sektoru po celém světě. Jde o multiplatformní nástroj, který je vydáván společností Oracle jako *freeware*, tudíž jeho uživatelé nejsou při používání nijak omezováni. Hypervizor využívá metodu plné virtualizace, tím pádem není možné emulovat jiné hardwarové platformy. Umožňuje vytváření a správu virtuálních strojů s kterýmkoliv operačním systémem, jenž podporuje řadu procesorů x86 a x64. Je tedy možné provozovat např. virtuální stroje se systémy Windows a Linux na fyzickém stroji se systémem Mac OS X.

V této kapitole bude krátce představena společnost Oracle, která tento produkt vyvíjí. Dále budou detailně rozebrány různé možnosti, kterými lze vytvářet a spravovat virtuální stroje ve VirtualBoxu. Zároveň zde bude zvolen způsob, jakým bude realizována praktická část této práce.

5.1 Oracle

Společnost Oracle je americká nadnárodní společnost zabývající se vývojem databázových systémů, systémů pro správu cloudů a dalšího softwaru určeného pro podniky. Byla založena v roce 1977. Od té doby její význam na poli relačních databází rostl díky faktu, že byla první, kdo toto odvětví IT komercializoval. Na konci 70. let společnost vydala první verzi produktu Oracle (později přejmenovaného na Oracle Database).

V 80. a 90. letech docházelo ke značnému rozvoji produktu Oracle Database. Došlo k přepsání softwaru do jazyka C, integrování procedurálního jazyka PL/SQL[1] do databáze, přidání podpory paralelních dotazů nad databází a dalším změnám. V tomto období se začala společnost věnovat vývoji objektového jazyka Java a vydala integrované vývojové prostředí Oracle JDeveloper. Toto prostředí bylo později integrováno s databázemi od společnosti Oracle.

Na počátku tisíciletí význam společnosti Oracle ve světě dále rostl, neboť postupně odkupovala menší softwarové firmy, a tím pádem získávala větší podíl svých produktů na trhu. V roce 2010 došlo k odkoupení společnosti Sun, která stála za vývojem virtualizační platformy VirtualBox. VirtualBox byl nadále vyvíjen společností Oracle, která významně rozšířila jeho funkcionalitu a zvýšila stabilitu.

V současné době má společnost Oracle více než 420 000 zákazníků po celém světě, kterým poskytuje nástroje pro cloudové aplikace, hardwarové systémy a další služby spojené

[1] Rozšíření databázového jazyka SQL o procedurální paradigma

s hostováním serverů, odbornými konzultacemi či financováním [8]. Hlavním produktem na poli virtuálních nástrojů je stále VirtualBox, jehož funkcionalita je nadále rozšiřována.

5.2 Uživatelské rozhraní

VirtualBox nabízí pro správu virtuálních strojů grafické uživatelské rozhraní (GUI). V tomto rozhraní lze velice snadno vytvářet a provádět konfiguraci hardwarových vlastností, které mají mít jednotlivé virtuální stroje. Je tak možné přizpůsobit každý stroj potřebám uživatele. Některé vlastnosti virtuálního stroje lze měnit pouze ve vypnutém stavu, to je dáno i vlastnostmi virtualizovaného operačního systému. Mezi takové vlastnosti patří například počet procesorů, velikost operační paměti, velikost grafické paměti a jiné. Naopak jiné parametry lze měnit za běhu virtuálního stroje, například pravidla pro přesměrování portů.

Grafické uživatelské rozhraní umožňuje vykonávat základní operace nad virtuálním strojem, tj. spuštění, vypnutí, restartování či uložení stavu stroje. Taktéž je možné virtuální stroje importovat či exportovat, což je frekventovaně využívaná operace, neboť je velmi často potřeba virtuální stroj přemístit na jiný fyzický server.

Konfigurace virtuálního stroje obsahuje více kategorií. Každá kategorie v sobě zahrnuje několik položek, které je možné upravovat podle potřeby uživatele. Položky všech kategorií jsou následující:

1. *General* (Obecné) – v této kategorii lze změnit název nebo typ operačního systému virtuálního stroje. Typ je většinou nastaven při vytváření stroje a po dobu provozování zůstává nezměněný. Editovat jej má smysl v případě, že bychom chtěli stroji přiřadit jiný systémový disk. V takovém případě je vhodné tuto položku změnit, aby byl virtuální stroj lépe optimalizován,

2. *System* (Systém) – slouží k modifikaci hardwarových vlastností stroje. Nastavuje se zde mimojiné velikost operační paměti, pořadí při bootování, počet procesorů či možnost podpory hardwarové virtualizace. Zapnutí/vypnutí hardwarové vitrualizace má zásadní vliv na výkon celého virtuálního stroje. Některé operační systémy (Windows 8, FreeBSD a jiné) vyžadují mít tuto virtualizaci povolenou, jinak není možné systém spustit,

3. *Display* (Obrazovka) – v této skupině lze upravovat grafické nastavení virtuálního stroje. Mezi to patří velikost video paměti, počet připojených monitorů, povolení 2D a 3D akcelerace. Pokud bude stroj provozován v grafickém režimu, tj. bude pro něj vytvořeno speciální interaktivní okno, je doporučená minimální velikost grafické paměti 10 MB. Dále je zde možné povolit vzdálenou obrazovku, díky které je možné se na stroj připojit přes vzdálenou plochu a ovládat jej z jiného fyzického stroje. Poslední položkou v této kategorii je zachytávání videa. To umožňuje v reálném čase nahrávat obrazovku virtuálního stroje a tento záznam uložit do zvolené cesty na hostitelském počítači. Lze nastavit i vlastnosti zachytávaného obrazu, tj. rozlišení, počet snímků za sekundu, či bitový tok, který má vliv na kvalitu obrazu,

4. *Storage* (Úložiště) – slouží k přidání odebrání řadičů diskových jednotek, CD/DVD mechanik, disketových mechanik, či USB sběrnice. Kterékoliv diskové jednotce lze přiřadit existující virtuální disk a nastavit, které disky jsou systémové a které datové. Do virtuálních optických mechanik lze vložit obraz, který je tímto médiem podporován, např. soubory ve formátu *.iso*,

5. *Audio* (Zvuk) – v tomto nastavení lze povolit či zakázat zvukový vstup/výstup ve virtuálním stroji. Pokud je zvuk povolený, je možné zvolit virtuální zvukovou kartu, kterou bude stroj používat. Na výběr jsou některé zvukové karty implementované v programu VirtualBox a zvukové karty hostitelského stroje. Obvykle není potřeba toto nastavení měnit,

6. *Network* (Síť) – nastavení sítě umožňuje konfiguraci síťových karet virtuálního stroje. Je možné nastavit způsob připojení k síti, kterým může být například NAT, síťový most, síť pouze s hostem a jiné. Také je uživateli umožněno nastavit MAC adresu síťové karty nebo ji nechat náhodně vygenerovat, což je vhodné v případě, že virtuální stroj je importován z obrazu, který byl vytvořený na stejném počítači. Bez změny MAC adresy by ji oba stroje měly totožnou a docházelo by ke konfliktu při síťové komunikaci. Dalším důležitým je přesměrování portů. To umožňuje, aby byl stroj dosažitelný v rámci privátní sítě,

7. *Serial ports* (Sériové porty)– VirtualBox umožňuje připojení až 4 sériových portů (COM), přičemž u každého lze nastavit číslo portu a režim portu (hostitelská roura, přímý soubor, TCP a jiné),

8. *USB* (USB) – v tomto nastavení je možné vybrat verzi USB, kterou bude virtuální stroj podporovat. Nabízeny jsou verze 1.1, 2.0 a 3.0. USB sběrnice je zpětně kompatibilní, čili pro podporu všech tří verzí musí být zvolena podpora USB 3.0. Dále je možné vytvořit filtry USB zařízení, které umožňují automatické připojení zařízení k virtuálnímu stroji ihned po spuštění. Ostatní zařízení, pro které není filtr vytvořen, lze připojit manuálně z kontextové nabídky ve virtuálním stroji,

9. *Shared folders* (Sdílené složky) – sdílené složky umožňují uživateli snadno přesouvat soubory mezi virtuálním a fyzickým strojem. To může usnadnit práci se soubory v případě, že neexistuje nebo není dostupné žádné síťové úložiště,

10. *User Interface* (Uživatelské rozhraní) – Tato kategorie umožňuje uživateli si zvolit, které možnosti by měly být pro daný stroj dostupné v kontextové nabídce onoho stroje. Kontextová nabídka se nachází na horním okraji okna virtuálního stroje.

5.3 VBoxManage

Dalším způsobem, kterým je možné provádět správu virtuálních strojů v nástroji VirtualBox je program VBoxManage. Jedná se o program, který společnost Oracle dodává společně s každou distribucí VirtualBoxu. Není tedy nutné jej stahovat ani instalovat zvlášť. Spustitelný soubor se nachází v instalačním adresáři VirtualBoxu společně s jinými binárními soubory. VBoxManage je navržený pro správu virtuálních strojů přes příkazovou řádku či terminál a poskytuje více možností než grafické uživatelské rozhraní, neboť některé prvky nejsou v současné době v GUI implementované. Mezi případy, kdy je nezbytné použít VBoxManage namísto grafického rozhraní patří např. situace, kdy potřebujeme měřit metriky virtuálního stroje. Metrikami mohou být například využití jednotlivých jader procesoru, využívaná velikost operační paměti, aktuální stav virtuálního pevného disku a další.

Každý příkaz se skládá z podpříkazu a případně dalších argumentů, které jsou podpříkazem vyžadovány či podporovány. Fakt, že je VBoxManage spustitelný z příkazové řádky, jej činí snadno použitelným nástrojem pro automatizaci jakýchkoliv operací nad virtuálními

stroji. Je tedy vhodnější spíše pro vývojáře než pro běžné uživatele VirtualBoxu, pro které je grafické uživatelské rozhraní pohodlnější než použití příkazové řádky.

Jak bylo řečeno výše, VBoxManage je konzolová aplikace. To umožňuje snadné vytváření skriptů nebo dávkových souborů, a tím správu virtuálních strojů automatizovat narozdíl od grafického uživatelského rozhraní, kde tato možnost neexistuje. Ve skriptech je pak spouštěn příkaz VBoxManage s požadovanými argumenty. Obecný tvar příkazu je znázorněn na následujícím příkladu:

```
VBoxManage [general option] <subcommand> [arg [arg...]]
```

Jednotlivé komponenty příkazu mají následující sémantiku:

1. *General option* (Obecný přepínač) - umožňuje jednoduchou konfiguraci nebo získání základních informací o programu. Slouží například k vypsání verze nainstalovaného VirtualBoxu, zákazu vypisování výstupu na obrazovku či nastavení/zadání přístupového hesla, které je zapotřebí k používání některých příkazů, jež vyžadují vyšší míru zabezpečení,

2. *Subcommand* (Podpříkaz) je povinným parametrem každého příkazu, protože určuje činnost, kterou bude VBoxManage vykonávat. Takovou činností může být například vytvoření virtuálního stroje, spuštění, export a mnoho dalších,

3. *Arguments* (Argumenty) blíže specifikují prováděnou operaci. Většina podpříkazů musí mít jako jeden z argumentů virtuální stroj, nad kterým bude operace provedena. Tento stroj lze specifikovat buď celým názvem, nebo unikátním identifikátorem vytvořeným VirtualBoxem.

VBoxManage podporuje velmi mnoho různých podpříkazů. Jejich seznam včetně podporovaných argumentů lze získat spuštěním programu bez podpříkazu či použitím parametru −−help. Není tak nutné si podpříkazy pamatovat.

Na závěr této podkapitoly bude uveden příklad použití programu VBoxManage pro vytvoření čistého virtuálního stroje s operačním systémem Debian, 4 GiB operační paměti, dvěma procesorovými jádry a síťovým adaptérem připojeným jako NAT.

```
1. VBoxManage createvm --name Debian-8-64 --register
2. VBoxManage modifyvm Debian-8-64 --ostype Ubuntu_64
3. VBoxManage modifyvm Debian-8-64 --memory 4096 --cpus 2 -acpi on --boot1 dvd
4. VBoxManage modifyvm Debian-8-64 --nic1 nat --cableconnected1 on
5. VBoxManage createhd --filename ./Debian-8-64.vdi --size 12000
6. VBoxManage storagectl Debian-8-64 --name IDE --add ide
7. VBoxManage storageattach Debian-8-64 --storagectl IDE --port 0 --device 0
--type hdd --medium ./Debian-8-64.vdi
8. VBoxManage storageattach Debian-8-64 --storagectl IDE --port 1 --device 0
--type dvddrive --medium debian-8.4.0-amd64-DVD-1.iso
```

Rozbor příkladu:

1. Vytvoření virtuálního stroje a přidání do manažera virtuálních strojů.
2. Nastavení typu operačního systému. VirtualBox nabízí omezenou množinu typů operačních systémů. Typ *Debian* neexistuje, proto je zvolena možnost *Ubuntu_64*, která je na Debianu založená.
3. Přidělení zdrojů virtuálnímu stroji a nastavení pořadí bootování, na prvním místě je DVD mechanika.
4. Nastavení síťového rozhraní *nic1* na NAT a připojení (virtuální stroj bude mít připojený síťový kabel).
5. Vytvoření nového obrazu pevného disku s velikostí 12 GB.
6. Vytvoření řadiče pro rozhraní *IDE*, aby bylo možné vložit obraz DVD do virtuální mechaniky stroje .
7. Připojení vytvořeného virtuálního pevného disku (*Debian-8-64.vdi*) ke stroji na port č. 0.
8. Připojení instalačního obrazu, se jménem *debian-8.4.0-amd64-DVD-1.iso*, do virtuální mechaniky stroje na port č. 1.

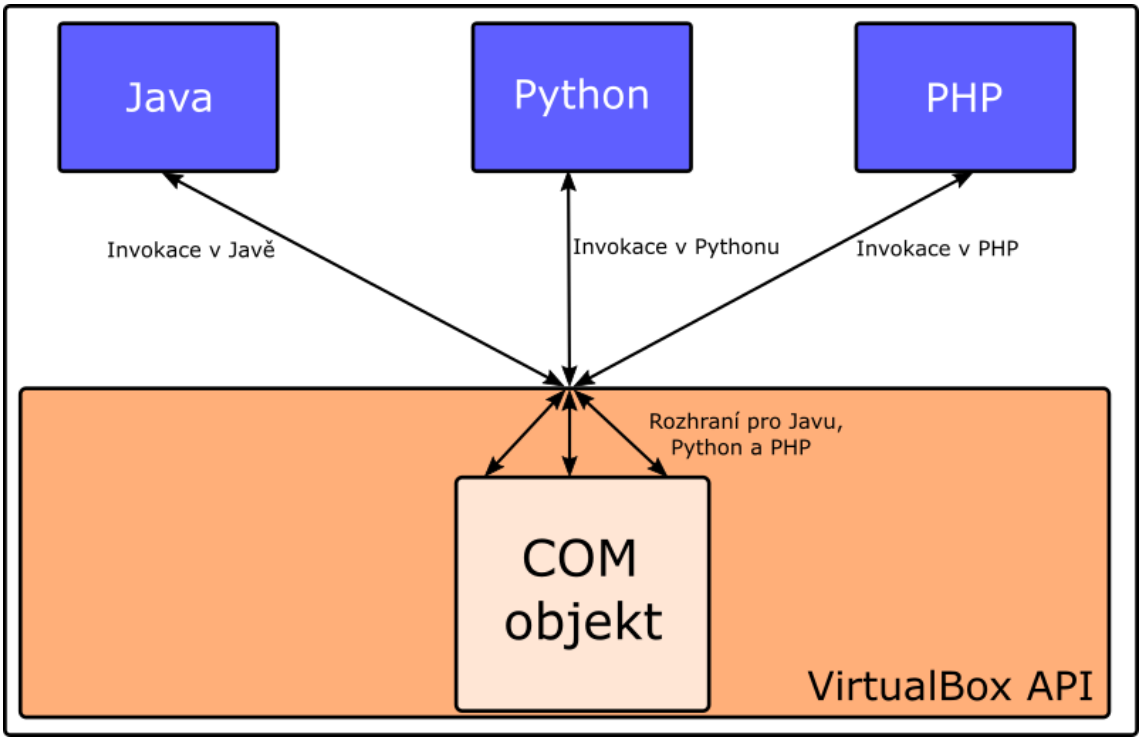
Po provedení všech výše uvedených příkazů bude v grafickém rozhraní zobrazen virtuální stroj s názvem *Debian-8-64* s parametry, které byly v příkazové řádce nastaveny. Při prvním spuštění bude zahájena instalace operačního systému Debian.

5.4 VirtualBox API

VirtualBox nabízí možnost ovládání virtuálních strojů skrze speciální aplikační rozhraní, které umožňuje přímou komunikaci s jádrem hypervizora, a tím pádem je přístupná veškerá funkcionalita, kterou hypervizor poskytuje. K používání VirtualBox API je nutné si nejdříve stáhnout a nainstalovat VirtualBox SDK, které je dostupné na webových stránkách VirtualBoxu. Určeno je především pro programátory, kterým nedostačuje funkcionalita již existujících nástrojů (*VBoxManage*, *VBoxHeadless* a jiné) a potřebují správu virtuálních strojů přizpůsobit svým potřebám. V současné době je SDK oficiálně podporováno v několika programovacích jazycích, mezi které patří Java, C++, Python, .NET, PHP a jiné. Pro některé jazyky existují i neoficiální portované moduly. Při používání VirtualBox API existují dva možné přístupy. Oba budou popsány v následujících podkapitolách.

5.4.1 Component Oriented Model

Component oriented model neboli COM, je „standard pro vytváření binárních softwarových komponent, které jsou objektově orientované a nezávislé na platformě“ [7]. Umožňuje sdílení dat mezi procesy a jejich vzájemnou komunikaci. Důležité je zmínit, že objekty, které jsou implementovány tímto modelem jsou jazykově nezávislé. To má za následek skutečnost, že vnitřní funkcionalita komponenty musí být implementačně oddělená od rozhraní přes které probíhá komunikace s danou komponentou. Pro každý podporovaný jazyk pak musí existovat rozhraní, které vývojáři využívají k propojení jednotlivých součástí svého softwaru.



Obrázek 5.1: Obecný příklad COM modelu

VirtualBox API tento model využívá pro vytvoření struktury objektů, které pak může programátor využívat. Jelikož byl COM model navržený společností Microsoft, je nativně podporovaný pouze na systémech Microsoft Windows. Na ostatních systémech je používána volně šířitelná implementace tohoto modelu zvaná XPCOM. Na obrázku 5.1 je uvedena zjednodušená architektura COM modelu, která je tímto aplikačním rozhraním využívána. Je z něj patrné, že každá komponenta hypervizora má několik rozhraní, každé příslušící jednomu programovacímu jazyku. Programátor si tak může zvolit libovolný programovací jazyk, pro který rozhraní existuje a k objektu přistupovat jednotným způsobem z kteréhokoliv z těchto jazyků.

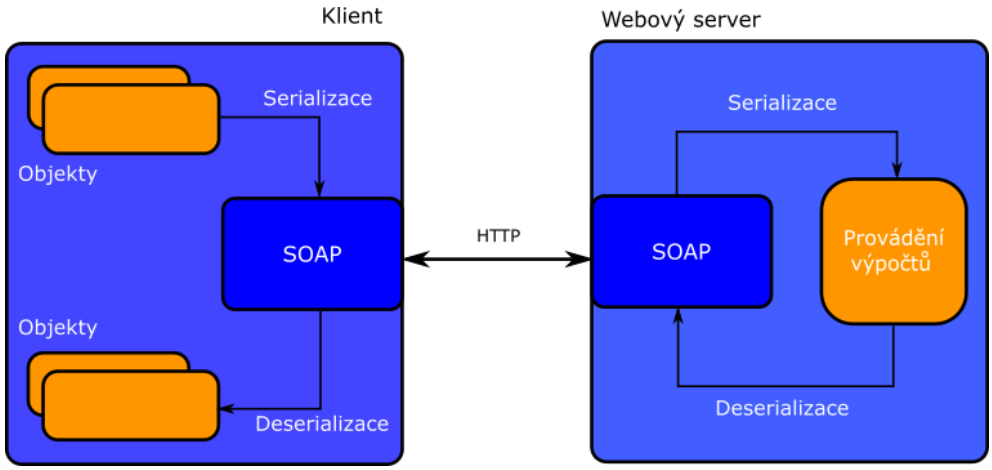
5.4.2 Webservice

Webová služba je služba, která umožňuje komunikaci dvou elektronických zařízení prostřednictvím WWW[2]. Používá architekturu typu klient-server, kde klient zasílá požadavky serveru, ten je zpracovává a zpracovaná data odešle zpět klientovi. Nejčastěji bývají využívány společně se standardy XML a SOAP, které definují formát zpráv mezi klientem a serverem. Princip komunikace je zobrazen na obrázku 5.2.

V případě, že chceme využít VirtualBox API společně s webovou službou, jsou zapotřebí 3 komponenty:

1. První z nich je webový server *VBoxWebsrv.exe*, který je součástí distribuce VirtualBoxu. Server přijímá HTTP požadavky od klienta a získává data od instance VirtualBoxu, která se nachází na stejném stroji jako server. Následně jsou požadovaná data zaslána zpět klientovi.

[2]World Wide Web



Obrázek 5.2: Princip komunikace přes webovou službu

2. Druhou nezbytnou komponentou jsou WSDL[3] soubory, které popisují funkcionalitu webové služby. Jsou v nich například popsány objekty, které API definuje včetně jejich metod. Pokud se tedy uživatel (programátor) pokouší provést zavolání funkce na vzdáleném serveru, je před odesláním dotazu zkontrolována existence této funkce ve WSDL souboru. Tyto soubory jsou zpravidla ve formátu XML a jsou dostupné v SDK, které Oracle zdarma poskytuje.

3. Poslední součástí je klient, který se připojuje k webové službě na vzdáleném počítači a provádí správu virtuálních strojů. Tento program vytváří progamátor podle vlastních potřeb.

Výhodou webové služby je fakt, že se dá považovat za jisté rozšíření COM modelu. Program (klient) implementovaný tímto způsobem přistupuje k objektům, které jsou definovány VirtualBox API stejným způsobem jako u COM modelu, ovšem s tím rozdílem, že manažerovi virtuálních strojů jsou předány speciální parametry. Tím je zajištěna znovupoužitelnost kódu při přechodu mezi COM a webovou službou nebo naopak. Mezi nevýhody patří naopak vysoká režie spojená se serializací dat do formátu XML. Ta musí být provedena při každém volání metody, což značně zpomaluje celou činnost programu.

[3]Web Service Description Language

Kapitola 6

Návrh aplikace

Cílem praktické části této práce je vytvoření programu, který bude umožňovat snadnou automatizaci správy virtuálních strojů. Tato kapitola se zabývá specifikací výsledného programu, výběru nejvhodnější metody ze způsobů, které byly popsané v předchozí kapitole a návrhem této aplikace.

6.1 Specifikace a cíle aplikace

Specifikace a požadavky na aplikaci byly konzultovány s vedoucím práce a došlo k ustanovení, že aplikace musí mít následující funkcionalitu:

- Interaktivní režim, ve kterém bude umožněno přímo spravovat virtuální stroj.
- Virtuální stroje se mohou nacházet na více fyzických serverech.
- Možnost shlukovat stroje do uživatelem definovaných skupin.
- Provádět operace nad skupinami strojů.
- Možnost uložení/načtení konfigurace do/ze souboru.
- Podpora více platforem.

Interaktivní režim by měl umožňovat snadné provádění správy strojů na základě vstupů přijímaných od uživatele v reálném čase. Virtuální stroje mohou být provozovány na několika fyzických serverech. Je nutné tuto skutečnost brát v potaz a aplikaci umožňovat dynamické přepínání mezi těmito servery.

Předpokládá se, že častým případem užití bude provedení stejného typu operace na několika strojích, např. zapnutí, vypnutí, restart, spuštění skriptu, zkopírování souborů a další. Virtuální stroje by tedy mělo být možné shlukovat do skupin, které jsou definované uživatelem, přičemž v jedné skupině mohou být virtuální stroje z různých fyzických serverů.

V rámci práce je největší důraz kladen na možnost automatizace správy virtuálních strojů. Proces údržby virtuálních strojů je pro člověka časově náročný, protože se musí na každý virtuální stroj připojit, provést operaci údržby a následně se odpojit. Aplikace by tedy měla umožňovat uživateli definovat operace, které chce na každém stroji provést a následně je spustit. Všechny definované operace by se měly provést bez dalšího nutného zásahu uživatele.

Další funkcí aplikace by měla být snadná konfigurace. Musí existovat způsob, jakým bude možné snadno přidávat nebo odebírat stroje do/z aplikace, aby uživatel nemusel provádět nastavení při každém spuštění. Vedoucí práce navrhl, že pro tento účel je vhodné používat soubory, které budou mít předem definovaný tvar.

Aplikace by měla být spustitelná na více platformách, konkrétně tedy v prostředí Microsoft Windows a Linux. Podpora systému Mac OS X, Solaris a dalších není vyžadována.

6.2 Výběr metody

V předchozí kapitole o VirtualBoxu byly popsány různé metody, jakými je možné stroje i celý VirtualBox ovládat. Mezi tyto způsoby patří grafické uživatelské rozhraní, program VBoxManage a VirtualBox API, které ještě poskytuje 2 různé přístupy - COM a webová služba. Výhody a zápory těchto metod jsou shrnuty v tabulce 6.1.

GUI	+ Jednoduché a snadno použitelné pro uživatele − Neposkytuje veškerou funkcionalitu, kterou VirtualBox nabízí − Neautomatizovatelné
VBoxManage	+ Poskytuje plnou funkcionalitu VirtualBoxu [10] − Automatizovatelné v jakémkoliv jazyce (je spouštěn jediný binární soubor s různými argumenty) − Virtuální stroj musí být na stejném stroji jako VBoxManage
API (COM)	+ Objektově orientovaný přístup, programovatelný v jazycích, které mají dostupnou podporu pro COM rozhraní + Relativně vysoká rychlost − Klient musí být na stejném fyzickém stroji jako virtuální stroje
API (*webservice*)	+ Objektově orientovaný přístup, programovatelný v jazycích Java, Python, C++ a dalších + Virtuální stroj se může nacházet na jakémkoliv dostupném serveru − Vysoká režie spojená se serializací do formátu XML

Tabulka 6.1: Srovnání metod pro správu virtuálních strojů.

Z tabulky je patrné, že nejvhodnější metodou, která vyhovuje požadavkům na aplikaci je použití VirtualBox API s webovou službou, neboť podporuje případy, kdy se virtuální stroje nacházejí na několika serverech. Kdyby tento požadavek neexistoval, bylo by vhodnější metodou využití VirtualBox API s COM, neboť by zde odpadala vysoká režie spojená se serializací objektů ve formátu XML, která má zásadní vliv na rychlost běhu celého programu. Jako implementační jazyk byl zvolen Python, protože Oracle pro tento jazyk poskytuje SDK, které obsahuje nezbytné náležitosti pro použití webové služby.

6.3 Návrh uživatelského rozhraní

Návrh uživatelského rozhraní je klíčovou součástí celé aplikace, neboť jde o součást aplikace, která má zásadní vliv na způsob používání celého programu. Při návrhu byl kladen největší důraz na co nejintuitivnější ovládání virtuálních strojů, které mohou být umístěny na několika fyzických serverech. Pro tento účel bylo účel bylo zvoleno textové uživatelské rozhraní (TUI), které bude implementováno formou interpretu. Tímto způsobem bude docíleno jednoduchého rozhraní, které bude mít minimální nároky z hlediska potřebných externích modulů pro jazyk Python. Grafické moduly totiž mají určité závislosti, které musí být před spuštěním aplikace nainstalované. U textového uživatelského rozhraní jsou ale plně dostačující moduly, které jsou součástí každé instalace jazyka Python.

Jak bylo řečeno výše, textové uživatelské rozhraní bude implementováno jako interpret. Uživatel tedy bude zadávat jednotlivé příkazy, které budou okamžitě vykonávány. Každý příkaz bude mít specifické argumenty, jež budou blíže specifikovat jeho činnost. Jako součást uživatelské přívětivosti by měl interpret ukládat (omezenou) historii zadaných příkazů, aby např. v případě překlepu nebyl uživatel nucen vypisovat celý příkaz znovu. S překlepy souvisí i obecné syntaktické či sémantické chyby příkazů, které bude třeba ošetřit. Aplikace by při měla chybu vstupu okamžitě rozpoznat, uživateli danou skutečnost oznámit a nechat jej zadat další příkaz. Nemělo by tedy docházet k situaci, kdy je program nečekaně ukončen.

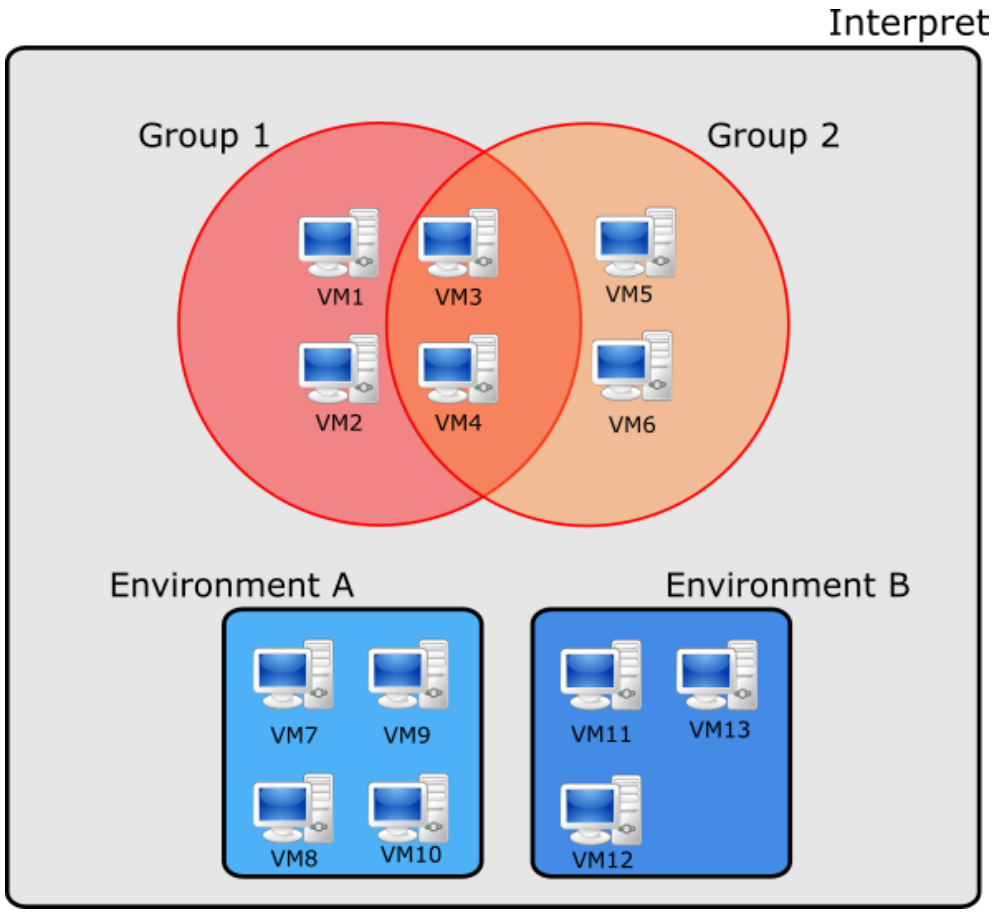
6.4 Objektový model

Další důležitou součástí je návrh struktury programu. Pro vývoj bylo zvoleno objektově orientované paradigma z toho důvodu, že tento přístup umožňuje snadnější spolupráci mezi jednotlivými komponentami aplikace a vyšší přehlednost napsaného kódu – v případě přidání nebo změny metody, je program modifikován na jediném místě. Během fáze návrhu byly navrženy celkem 3 třídy, které by měly být dostačující na požadovanou funkcionalitu. Těmito třídami jsou:

- *Group* (skupina) – třída, jež bude shlukovat jeden či více virtuálních strojů. Tyto stroje se mohou nacházet na více fyzických serverech. Bude obsahovat metody pro přidání či odebrání virtuálního stroje,
- *Environment* (prostředí) – třída obsahující informace o jednom fyzickém serveru. Tato třída je užitečná kvůli faktu, že každý fyzický server operuje v jiném prostředí. Každý fyzický server má unikátní IP adresu v rámci sítě, běží na něm určitá instance VirtualBoxu, jejíž verze se může lišit od verzí na jiných strojích, obsahuje jiné virtuální stroje apod. Tyto informace o různých serverech od sebe musí být izolovány,
- *Interpreter* (interpret) – hlavní třída aplikace, která bude mít v programu jedinou instanci. Po spuštění programu bude zavolána metoda, ve které budou přijímány příkazy od uživatele. Instance této třídy bude obsahovat veškeré odkazy na známé, tj. uživatelem definované, skupiny virtuálních strojů, fyzické servery a virtuální stroje na nich. Tato třída bude také definovat veškeré metody pro manipulaci se skupinami, stroji a v neposlední řadě metody pro vykonání samotných příkazů, které uživatel zadá.

Na obrázku 6.1 je znázorněn příklad, kdy interpret obsahuje 2 skupiny virtuálních strojů a 2 prostředí fyzických serverů. Lze zpozorovat, že skupiny strojů nemusí být nutně disjunktní, a tedy stroje mohou být v několika skupinách zároveň. To ale neplatí o prostředích.

Pakliže prostředí uchovává informace o jednom fyzickém serveru, není možné, aby se jeden virtuální počítač nacházel na více těchto serverech. Je nutno podotknout, že virtuální stroje, které jsou v nějakém prostředí, mohou být zároveň v nějaké skupině. Mohl by tedy nastat případ, že by se virtuální stroj *VM7* nacházel jak v prostředí *Environment A*, tak i ve skupině *Group 1*.



Obrázek 6.1: Návrh propojení objektů v interpretu.

6.5 Konfigurace interpretu

Konfigurace, obsahující seznam virtuálních strojů, popř. definici skupiny strojů bude uložena v souboru. Tento soubor bude při spuštění aplikace zadán jako argument. Každý řádek souboru bude definovat právě jeden virtuální stroj, popř. přiřadí virtuální stroj do skupiny. U každého stroje bude nezbytné uvést jeho celý název a na jakém serveru se nachází. Dále se na řádku mohou vyskytovat položky, které jsou nezbytné jen pro určité příkazy. Mezi tyto položky patří uživatelské jméno a heslo. Tyto údaje jsou potřebné například pro provedení jakékoliv operace přímo ve virtuálním stroji, tzn. spuštění skriptu, vytvoření adresáře a další. Jiné operace, jako např. spuštění, restartování či export virtuálního stroje tyto údaje nepotřebují. V takových případech by tyto údaje neměly být zadávány, protože jde o citlivá data. Navržený tvar jednoho řádku konfiguračního souboru má následující syntaxi:

```
<hostname|IP>(/|:)<machine_name> [property=value[,...] [...]]
```

Řádek bude vždy začínat doménovým jménem, či IP adresou fyzického serveru, na kterém se daný virtuální stroj nachází. Za ním bude následovat název virtuálního stroje, který musí být totožný s názvem stroje na serveru. Tyto dva údaje od sebe budou odděleny jedním lomítkem nebo dvojtečkou. Tyto údaje jsou nezbytné pro registraci virtuálního stroje do aplikace, avšak bude musí existovat možnost, jak jej bude možné přidat do nějaké skupiny. Pro tento účel lze v konfiguračním souboru specifikovat rozšiřující vlastnosti stroje. Mezi tyto vlastnosti patří:

- *group* – přiřadí stroj do pojmenované skupiny. V interpretu pak bude možné provádět operace nad celou skupinou virtuálních strojů. Jeden stroj může být ve více skupinách zároveň,
- *user* – uživatelské jméno, které bude použito pro přihlášení na daný virtuální stroj. Je vyžadováno pouze u některých operací. K virtuálnímu stroji může být přiřazen pouze jediný uživatelské jméno,
- *password* – heslo k uživatelskému účtu, jenž bude použit pro provedení operací. Stejně jako u uživatelského jména, je povoleno pouze jedno heslo k virtuálnímu stroji.

Výše popsané vlastnosti budou v souboru definovány způsobem, který je syntakticky shodný s přiřazením. Vlastnost a hodnota od sebe budou odděleny rovnítkem, přičemž název vlastnosti se bude nacházet na levé straně a hodnota na pravé straně. Pokud bude stroj přiřazován do více skupin, budou názvy těchto skupin odděleny čárkou. Dle návrhu by konfigurace virtuálního stroje mohla mít tvar, který je znázorněn na následujícím příkladu:

```
lissom1/Deb-8-64 group=debian,64bit user=taylor password=t0ps3cret
```

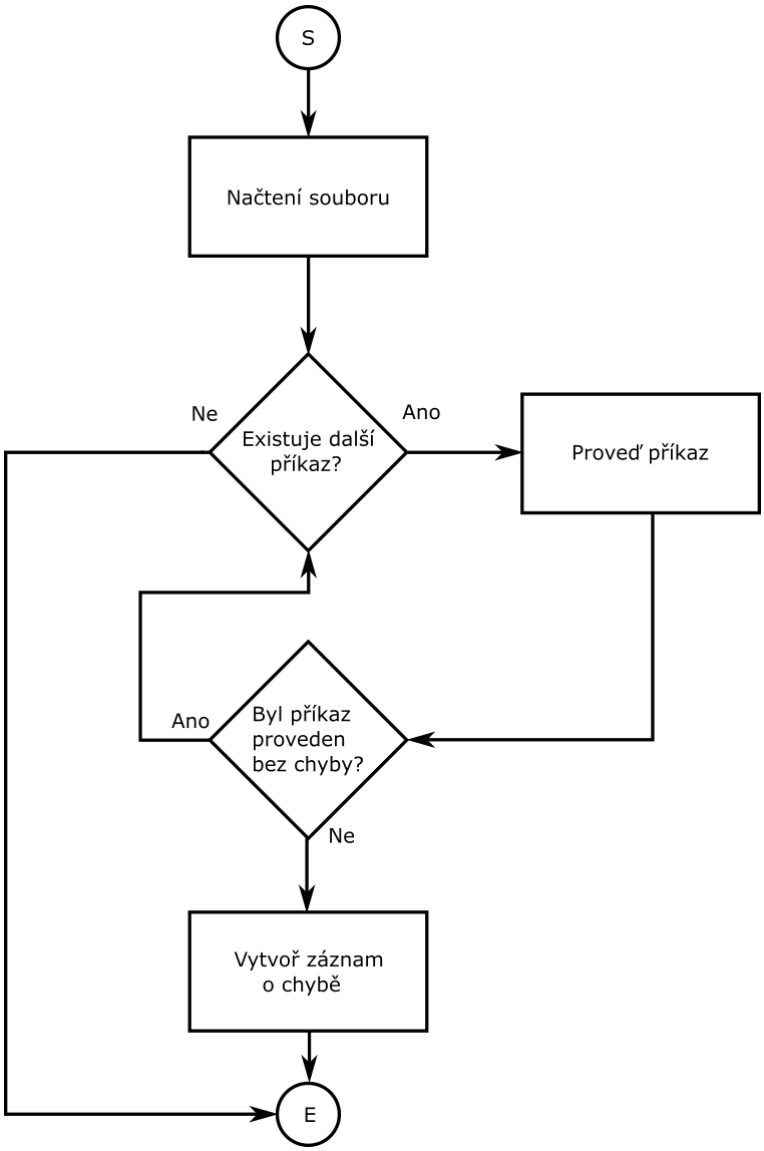
6.6 Automatizace

Automatizace správy virtuálních strojů je hlavním cílem této aplikace, je tedy nedílnou součástí návrhu. Jedná se o takový režim interpretu, kdy k provedení příkazů nad virtuálními stroji nebo celými skupinami nebude potřebný zásah člověka. V první řadě tedy musí být interpret nakonfigurován, aby měl uložené objekty, nad kterými budou příkazy vykonávány. Konfigurace bude probíhat způsobem, který byl popsán v předchozí podkapitole.

Operace, které budou interpretu předány k automatickému zpracování, budou uloženy ve zvláštním souboru. Jeho obsah budou tvořit jednotlivé příkazy, které jsou interpretem podporovány. Půjde tedy o dávkové zpracování, kdy budou sekvenčně vykonávány operace, které budou v tomto souboru uvedeny. Tento soubor bude aplikaci předán přes argument příkazové řádky.

Může nastat situace, že soubor bude obsahovat neznámé příkazy, neexistující (resp. neregistrované) virtuální stroje, či nedostatečné informace o virtuálních strojích, např. při konfiguraci stroje nebylo zadáno uživatelské jméno a heslo, ale aktuálně prováděná operace tyto údaje potřebuje. Je potřeba se zamyslet nad způsobem, jakým budou tyto chybové případy řešeny. Jsou možné 2 přístupy. Buď bude chybná operace přeskočena a bude se pokračovat dalším příkazem, nebo dojde k okamžitému přerušení vykonávání programu. Vhodným způsobem je možnost, kdy bude o uživatel o chybovém stavu informován prostřednictvím výpisu na obrazovku nebo přidáním záznamu do speciálního záznamu a vykonávání dávkového souboru bude ukončeno. Předpokládá se totiž, že většina operací v dávkovém souboru na sobě bude závislá, tzn. vykonání jistého příkazu se spoléhá na fakt, že některé

z předchozích příkazů byly provedeny dle očekávání. Typickým příkladem může být situace, kdy operace vyžaduje, aby byl jistý virtuální stroj zapnutý. Příkaz pro spuštění virtuálního stroje ale nemusel být vykonán dle očekávání, neboť obsahoval překlep. Od okamžiku selhání tohoto příkazu jistě selžou i ostatní příkazy, které tento virtuální stroj vyžadují ve spuštěném stavu. Provádění dalších příkazů tedy nemá význam. Diagram pro provádění automatizovaných operací je znázorněn na obrázku 6.2.



Obrázek 6.2: Diagram pro automatizované provádění příkazů.

Kapitola 7

Implementace a testování

Implementační část se zabývá popisem prvků, jež bude nezbytné vytvořit pro správnou funkcionalitu celé aplikace. Budou zde prakticky rozebrány části, které byly popsány v předchozí kapitole. První část kapitoly bude popisovat implementaci tříd, které aplikace využívá a jejich vzájemnému propojení. U implementace tříd jsem se zaměřoval na snadnou rozšiřitelnost a snadné použití.

Další část pak bude věnována zhodnocení vytvořené aplikace. Bude vykonáno několik měření rychlosti programu, které složitostí odpovídají očekávanému způsobu používání. Na základě výsledků bude zhodnocena použitelnost aplikace v praxi.

7.1 Implementace tříd

7.1.1 Třída Group

Třída *Group* je určena ke shromažďování virtuálních strojů do jedné entity. Obsahuje atributy *name* a *machines*. Atribut *name* je jednoznačný identifikátor instance této třídy, kterým bude instance adresována. Jedná se o povinný parametr konstruktoru třídy *Group*. Pro získání atributů *name* a *machines* jsou implementovány metody *GetName* a *getMachines*.

Atribut *machines* obsahuje virtuální stroje, které spadají do jedné skupiny. Je realizován pomocí vnořeného slovníku[1]. Klíči slovníku *machines* jsou názvy fyzických serverů nebo jejich IP adresy a hodnotami jsou slovníky, které obsahují informace o virtuálních strojích, tj. název a přihlašovací údaje. Název fyzického stroje je důležitý z toho důvodu, že aplikace je schopna pracovat s více servery a je nutné mít informaci o tom, kde se každý virtuální stroj nachází. Tím je navíc umožněno v aplikaci pracovat se dvěma stejně pojmenovanými virtuálními stroji (musí být ale na různých serverech). O každém stroji je nezbytné uchovávat i přihlašovací údaje, které mají být použity v případě, kdy je daná operace vyžaduje. Tyto údaje jsou implicitně nastaveny na *None*, neboť u většiny operací nejsou potřebné.

Pro přidávání či odebírání virtuálních strojů ze skupiny existují metody *addMachine*, resp. *removeMachine*. Metoda *addMachine* má jako povinné parametry jméno serveru a název virtuálního stroje, aby bylo možné jednoznačně identifikovat lokalizaci virtuálního stroje v rámci sítě. Nepovinnými parametry jsou pak uživatelské jméno a heslo k virtuálnímu stroji z důvodu, jenž byl popsán v předchozím odstavci. Na druhou stranu metoda *removeMachine* odebere virtuální stroj či celý fyzický server (včetně všech jeho virtuálních strojů) ze skupiny.

[1] Slovník – Datová struktura obsahující dvojice klíč:hodnota. V jiných programovacích jazycích je tento typ známý jako asociativní pole

7.1.2 Třída Environment

Třída *Environment* obsahuje informace o jednom fyzickém serveru. Na každém serveru běží unikátní instance HTTP serveru (program VBoxWebSrv) a VirtualBoxu, ve kterém jsou registrovány virtuální stroje, proto je nutné tyto údaje udržovat odděleně. Parametry se konstruktoru předávají ve slovníku, jehož klíči jsou názvy atributů. Třída obsahuje následující atributy:

- *host* - Doménové jméno nebo IPv4 adresa fyzického serveru. Slouží k nalezení serveru v rámci sítě.
- *port* - Port, na kterém naslouchá webový server. Implicitní hodnota je 18083.
- *user* - Jméno uživatele, pod kterým je spuštěný webový server. Implicitně prázdný řetězec.
- *password* - Heslo uživatele, pod kterým je spuštěný webový server. Implicitně prázdný řetězec.
- *name* - Uživatelem definované jméno instance třídy. Tento atribut je používán ve všech případech, kdy se adresuje nějaké prostředí, např. při změně aktivního prostředí. Implicitně je stejné jako host.
- *style* - Způsob připojení k manažerovi VirtualBoxu. Nabývá hodnot *None* nebo „*WEBSERVICE*“. Hodnota *None* znamená, že manažer je spuštěn na lokálním stroji a pro komunikaci s VirtualBox API bude použit model COM (viz kapitola 5.4.1). Hodnota „*WEBSERVICE*“ značí, že se aplikace pokusí připojit na webový server (který může běžet i na lokálním počítači).
- *remote* - Booleovská hodnota značící, zda je manažer připojen přes webový server.
- *mgr* - Manažer VirtualBoxu, na který je instance třídy *Environment* napojena.
- *vbox* - Instance VirtualBoxu, která je spuštěna na fyzickém serveru.
- *const* - Speciální konstanty, které jsou definovány ve VirtualBox API.
- *machines* - Virtuální stroje, které jsou registrovány v aplikaci. Každá instance třídy *Environment* obsahuje pouze množinu strojů, které jsou na daném serveru. Atribut je implementován stejným způsobem jako u třídy *Group*, tj. slovník, jehož klíči jsou názvy virtuálních strojů a hodnotami jsou přihlašovací jméno a heslo uživatele, pod kterým se mají operace na virtuálním stroji vykonávat. Účel tohoto atributu je uchovávat právě přihlašovací údaje, tento atribut proto neobsahuje nutně všechny virtuální stroje, ale pouze ty, u kterých jsou přihlašovací údaje známé.
- *prompt* - Řetězec, který je vypsán při čekání na zadání příkazu od uživatele. Má tvar *user@name>*, kde *user* a *name* jsou hodnoty stejnojmenných atributů.

Třída *Environment* implementuje metody pro přidání a odebrání virtuálního stroje. Pokus o přidání již existujícího nebo odebrání neznámého virtuálního stroje má za následek prázdnou operaci.

7.1.3 Třída Interpreter

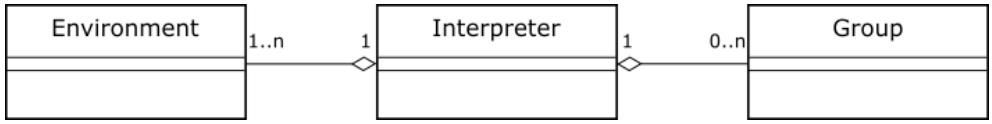
Tato třída je má hlavní roli v celé aplikaci, neboť obsahuje veškerou logiku a provádí příkazy, které uživatel zadá, tím pádem invokuje metody definované ve VirtualBox SDK, jež následně komunikují s hypervizorem. Třída má následující atributy:

- envs – slovník, uchovávající informace o registrovaných fyzických serverech. Klíč je tvořen názvem prostředí a hodnotou je instance třídy *Environment*.
- groups – slovník, uchovávající informace o uživatelem definovaných skupinách. Klíčem je název skupiny a hodnotou je instance třídy *Group*.
- isRemote – booleovská hodnota, značící, zda aplikace využívá webový server nebo bude využíván pouze lokální manažer VirtualBoxu. Tento atribut má vliv na množinu podporovaných příkazů z toho důvodu, že při použití lokálního manažera není možné používat příkazy pro přidání fyzického serveru, připojení na server a další. Konkrétněji bude toto chování popsáno dále.
- active – atribut uchovávající ukazatel na instanci třídy *Environment*, která je právě využívána. Určuje tedy, na kterém serveru se mají provádět zadané příkazy.
- commands – množina příkazů, které jsou interpretem podporovány. Tato množina se částečně odvíjí od atributu *isRemote*. Je implementována pomocí slovníku, jehož klíči jsou jména příkazu a hodnotami jsou dvojice (typ, metoda). Typ příkazu specifikuje, zda je operace prováděna bez využití VirtualBox API (např. výpis registrovaných skupin) nebo zda jej používá (např. spuštění virtuálního stroje). Druhou hodnotou ve dvojici je ukazatel na metodu, která příkaz provádí.

Třída obsahuje metody, které implementují všechny podporované příkazy. Každému příkazu odpovídá právě jedna metoda. To umožňuje snadnější testování správné funkcionality, protože jsou od sebe implementace příkazů odděleny a v případě zjištění programátorské chyby ji lze opravit na jediném místě bez ovlivnění dalších částí programu. Tyto metody mají unifikované rozhraní a jsou tedy invokovány jednotným způsobem. Díky tomu nemusí být v programu metody nijak rozlišovány podle počtu parametrů.

Jak bylo napsáno výše, atribut *isRemote* určuje množinu podporovaných příkazů. V případě, že tento atribut nabývá hodnoty *False*, nejsou povoleny příkazy, které slouží k přidání, odebrání či přepnutí aktivního prostředí. V takovém případě tyto příkazy nemají smysl, neboť komunikace s hypervizorem probíhá přes COM model, jenž ke své činnosti nevyužívá webový server, a tudíž není možné připojovat se na vzdálené servery.

Na obrázku 7.1 je znázorněno vzájemné propojení tříd *Group*, *Environment* a *Interpreter*. Je z něj patrné, že instance třídy *Interpreter* nemusí obsahovat žádnou skupinu strojů, ale musí existovat alespoň jedna instance třídy *Environment*, aby mohla být činnost programu započata.



Obrázek 7.1: Diagram tříd.

7.2 Zpracování argumentů

Zpracování argumentů aplikace je implementováno pomocí modulu *argparse*, který je součástí standardních knihoven Pythonu od verze 2.7 a provádí analýzu a načtení argumentů z příkazové řádky. Na parsování je vytvořena instance třídy *ArgumentParser*, následně jsou definovány podporované argumenty, včetně implicitních hodnot, a je invokována metoda, jež tyto argumenty zpracuje. Informace o parametrech jsou pak uloženy ve speciálním objektu třídy *Namespace*, jehož atributy jsou podporované argumenty a hodnotami těchto atributů jsou hodnoty převzaté z příkazové řádky.

7.3 Práce se soubory

7.3.1 Konfigurační soubor

Konfigurační soubor slouží k registrování virtuálních strojů a jejich případnému rozdělení do skupin bez účasti uživatele. Tento soubor je aplikaci předán buď pomocí parametru, nebo jej lze načíst i příkazem, který je v interpretu implementován.

Konfigurační soubor je načítán po řádcích. Na jednom řádku jsou tedy očekávány informace o jednom serveru nebo virtuálním stroji. Jednotlivé údaje jsou od sebe odděleny libovolným počtem mezer či jiných bílých znaků kromě znaku nového řádku. Oproti původnímu návrhu došlo k malým změnám formátu konfiguračního souboru, neboť při použití prvotního návrhu by nebylo možné do programu vkládat informace o fyzických serverech. Změna oproti návrhu je tedy taková. že prvním údajem je klíčové slovo *host* nebo *machine*, které určuje, zda následující informace budou interpretovány jako informace o serveru nebo virtuálním stroji. Následně byl přidán parametr host, který slouží k lokalizaci virtuálního serveru v rámci sítě. Ostatní parametry jsou oproti návrhu nezměněné.

Při implementaci bylo potřeba ošetřit chybové stavy, které mohou během načítání konfiguračního souboru nastat. Jedním z takových problémů je syntaktická chyba v konfiguračním souboru, např. chybějící znak = u parametru, chybějící klíčové slovo na začátku řádku a jiné. Při výskytu tohoto druhu chyby jsou všechny načtené informace zahozeny, aby byl zajištěn konzistentní stav aplikace. Před samotnou změnou je tedy zálohován aktuální stav a v případě chyby je tento stav obnoven.

7.3.2 Dávkový soubor

Dávkový soubor má za úkol automatizovat činnost interpretu. Obsahuje jednotlivé příkazy, které má interpret vykonávat. Ty jsou odděleny buď znakem nového řádku, nebo středníkem. Dávkový soubor je stejně jako konfigurační soubor možné předat přes parametr programu nebo zadání příkazu *batch*. Díky tomuto způsobu implementace je možné do jednoho dávkového souboru vložit příkaz pro zpracování jiného dávkového souboru, což umožňuje spojovat menší dávkové soubory do komplexnějších.

V souboru je možné psát jednořádkové komentáře, aby byla zvýšena přehlednost delších souborů. Komentář začíná znakem # a je ukončen znakem nového řádku. Blokové komentáře nejsou podporovány, neboť se nepředpokládá rozsáhlé použití komentářů v dávkových souborech.

7.4 Činnost aplikace

Po spuštění aplikace jsou zpracovány argumenty z příkazové řádky, které jsou následně uloženy do instance třídy *NameSpace*. Podporované argumenty je možné získat předáním parametru –h, který vypíše nápovědu programu a ukončí jej. Pokud je předán parametr –h, jsou ostatní argumenty ignorovány. Pro lepší pochopení následujícího textu jsou zde uvedeny podporované argumenty.

- –b BATCH_FILE – cesta k dávkovému souboru, který má být zpracován interpretem.
- –c CONFIG_FILE – konfigurační soubor obsahující informace o serverech a virtuálních strojích, se kterými má být interpret inicializován
- –w (webservice)– Pokud je tento parametr předán, tak interpret využívá webovou službu a je nutné, aby byl na všech spravovaných serverech spuštěný HTTP server VBoxWebSrv. V opačném případě je využíván COM model, který je rychlejší než webová služba, neboť komunikace s hypervizorem probíhá přímo a není nutné provádět serializaci dat do formátu XML. Bez použití webové služby je možné spravovat pouze ty virtuální stroje, které se nachází na stejném počítači jako aplikace.
- –o (opts) – Parametr sloužící k předání dalších argumentů, které jsou nezbytné pro připojení na webový server, tedy doménové jméno nebo IP adresa serveru, port a přihlašovací údaje uživatele, pod kterým je webová služba na vzdáleném serveru spuštěna. Tento parametr má tedy smysl pouze v případě, že je použita webová služba.

Poté, co jsou načteny argumenty, je vytvořena instance třídy *Interpreter*. Pokud byl zadán konfigurační soubor, je interpretem zpracován. V opačném případě je vytvořena instance třídy *Environment* a registrována do interpretu. Při vytváření tohoto prostředí mohou nastat dva případy:

1. Interpret má využívat webovou službu (parametr –w) – v takovém případě jsou pro určení serveru, který bude reprezentován ve vytvořeném prostředí, použity údaje, jež byly předány přes parametr –o. Pakliže tyto údaje nebyly uživatelem zadány, je proveden pokus o připojení na *localhost* s použitím implicitních hodnot. Jestliže pokus o připojení selže, není možné provádět běžné operace v interpretu – je nutné se nejprve připojit na běžící webový server.
2. Interpret má využívat COM model – U tohoto způsobu není nutné předávat jakékoliv ostatní parametry, protože VirtualBox API má přístup přímo k hypervizorovi na lokálním stroji. Během činnosti interpretu ale není možné spravovat vzdálené stroje.

Vytvořením instance třídy *Environment* a jejím načtením do interpretu, je možné zahájit činnost interpretu. K tomu slouží metoda run, jejímž nepovinným parametrem je dávkový soubor. Pokud je tento soubor zadán, nachází se interpret v automatickém módu a uživatel s ním nemůže provádět žádné akce. V tomto módu je zpracováván dávkový soubor s operacemi, které jsou postupně vykonávány. Po skončení této dávky je program ukončen.

Bylo nutné ošetřit chybové stavy, mezi které patří syntaktické chyby, neúplné argumenty příkazů, operování nad neznámými virtuálními stroji a další. V přístupu k chybovým stavům byla oproti původnímu návrhu provedena změna. Součástí návrhu bylo, že v případě výskytu chyby v dávkovém souboru, bude činnost ukončena. Tento přístup může být ale nevýhodný

v případě, že budou prováděny operace, které nejsou závislé, např. provedení příkazu na jedné skupině strojů a následně provedení jiného příkazu na druhé skupině strojů. Pokud by došlo k přerušení činnosti, kvůli tomu, že by první operace obsahovala chybu, nebyly by následující příkazy vůbec zpracovány. Tento důsledek je nežádoucí v případě časově náročných příkazů, neboť by uživatel musel po každé chybě spouštět dávku znova. Výsledný program tedy vytvoří záznam o chybě a pokračuje dalším příkazem.

Druhým režimem, který je interpretem podporován, je interaktivní režim. Ten je spuštěn v případě, že v argumentech programu nebyl zadán žádný dávkový soubor. V tomto režimu interpret běží v nekonečném cyklu a přijímá příkazy od uživatele. Po načtení vstupu je provedeno ověření, zda uživatel zadal validní příkaz. Jestliže příkaz neexistuje, je uživatel vyzván k zadání nového příkazu. V opačném případě je invokována metoda, jež daný příkaz implementuje. V každé metodě, která provádí nějakou operaci, je nejprve provedena kontrola správnosti jejích argumentů. Až když je toto ověření dokončeno, je příkaz vykonán. Algoritmus zpracovávání příkazů je zobrazen na obrázku 7.2.

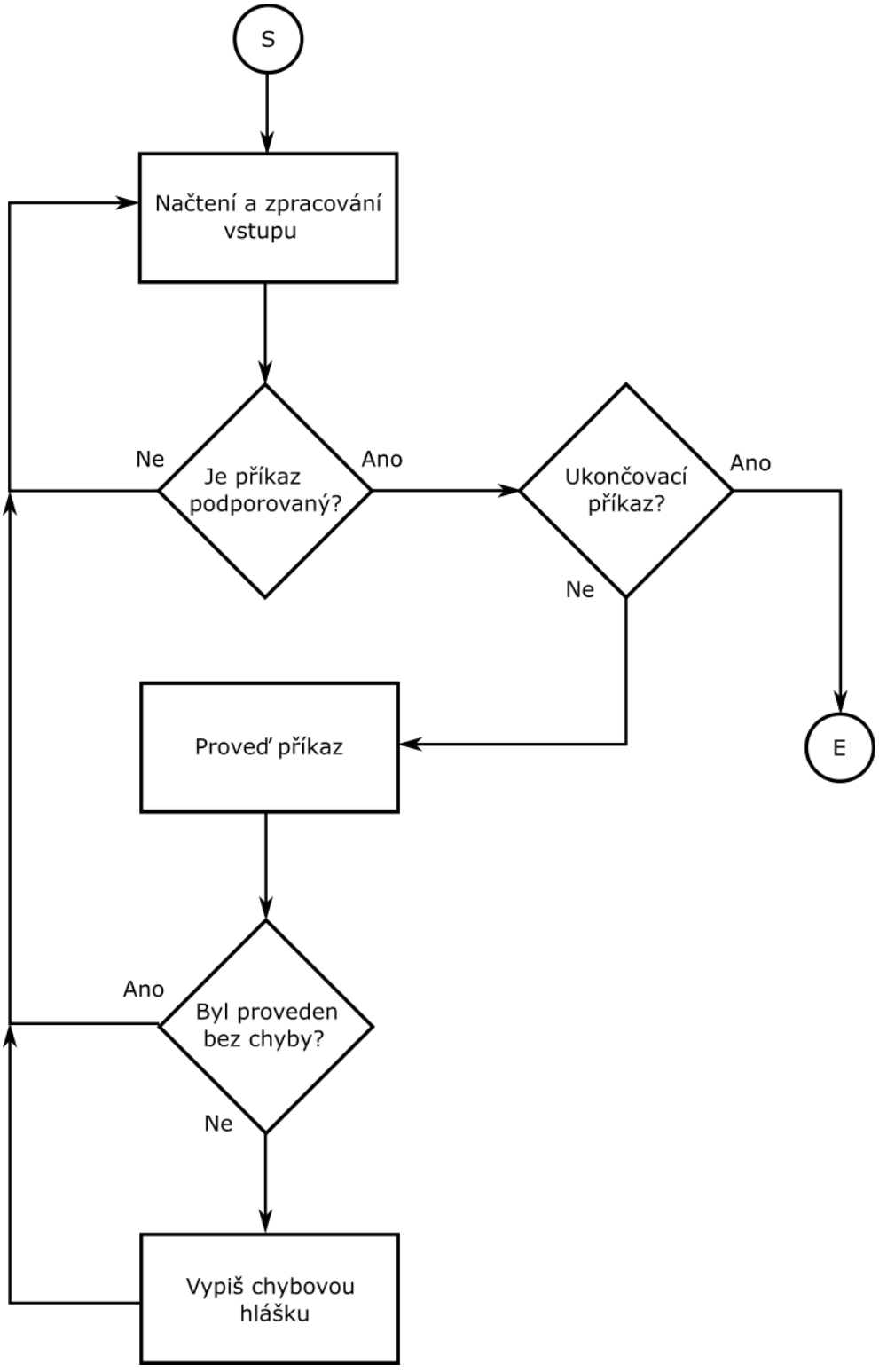
Program lze ukončit pouze příkazem *exit* nebo *quit*. Do doby, než je zadán jeden z ukončovacích příkazů, interpret očekává vstup od uživatele. V případě chyby během vykonávání příkazu tedy není ukončen celý program. Samotnou aplikaci není možné ukončit ani signálem *SIGINT* (klávesová zkratka *Ctrl + C*) z toho důvodu, že tento signál je využíván pro ukončení časově náročných operací, například importování nebo exportování virtuálních strojů. Při pokusu o přerušení programu tímto způsobem, je uživatel o této skutečnosti informován a jsou mu nabídnuty příkazy, kterými je možné interpret ukončit.

Pro časově náročné operace byla implementována metoda *ProgressBar*, která uživateli zobrazuje aktuální stav prováděné operace. Je možné programově nastavit interval, v jakém má být postup aktualizován. Uživatel tedy nemá možnost tento interval změnit. Tato metoda komunikuje s hypervizorem, neboť ten má povědomí a stavu prováděné operace. Hotová část této operace (v procentech) je součástí objektu, který je vrácen při zahájení. Tento objekt dále obsahuje odhadovaný zbývající čas, návratový kód operace a další vlastnosti, které nejsou v aplikaci využívány.

Pro zvýšení efektivity při užívání aplikace byla přidána podpora historie příkazů a automatické doplňování příkazů. K uchování historie příkazů je potřeba mít nainstalovaný modul *readline*[2], který ale bývá součástí distribucí Pythonu. V případě, že readline nenachází v nainstalovaných module, není zajištěna podpora tohoto rozšíření. Historie se ukládá před každým ukončením programu příkazem *exit* do souboru, jenž je umístěný v domovském adresáři uživatele, odkud je také načítán při spuštění aplikace. Maximální délka historie příkazů je nastavena na číslo 100. Tento počet je dostačující pro běžné používání aplikace.

Automatické doplňování příkazů je implementováno pomocí třídy *Completer*, jejíž definice se nachází v modulu *rlcompleter*. Konstruktoru této třídy je předán slovník s hodnotami, přes které je iterováno v případě, že probíhá vyhledávání řetězců, které začínají stejnými znaky jako řetězec zadaný uživatelem. V tomto případě jsou to všechny podporované příkazy. Vyhledávání možných příkazů je zahájeno klávesou tabelátor, která je pro tento účel používána v programech s textovým rozhraním, např. příkazová řádka nebo terminál. Pokud tedy uživatel zadá část nějakého příkazu, jsou mu po stisknutí tabelátoru nabídnuty varianty, kterými je možné doplnit příkaz. Pakliže existuje jediná možnost, je tento příkaz automaticky doplněn. Toto rozšíření může urychlit práci s interpretem, protože uživatel nemusí vypisovat celé příkazy ručně a není nutné si ani žádné pamatovat.

[2] readline – https://docs.python.org/2/library/readline.html



Obrázek 7.2: Algoritmus vykonávání příkazů.

7.5 Testování aplikace

V rámci testování aplikace byly vytvořeny soubory obsahující validní i nevalidní vstupy, což vedlo k odhalení několika programátorských chyb, zejména chybějící ošetření chybových vstupů. Testování aplikace proběhlo celkem na 4 serverech s následujícími konfiguracemi:

1. Server 1
    - Operační systém: Windows 8.1 64-bit
    - Počet jader procesoru: 4
    - Velikost operační paměti: 8 GiB
    - Verze VirtualBoxu (shodná s verzí SDK): 5.0.16
    - Rychlost síťového připojení: 100 Mbit
2. Server 2
    - Operační systém: Windows 10 64-bit
    - Počet jader procesoru: 8
    - Velikost operační paměti: 16 GiB
    - Verze VirtualBoxu (shodná s verzí SDK): 5.0.20
    - Rychlost síťového připojení: 100 Mbit
3. Server 3
    - Operační systém: Linux Mint 17 64-bit
    - Počet jader procesoru: 2
    - Velikost operační paměti: 4 GiB
    - Verze VirtualBoxu (shodná s verzí SDK): 5.0.2
    - Rychlost síťového připojení: 100 Mbit
4. Server 4
    - Operační systém: Debian 7.2 64-bit
    - Počet jader procesoru: 8
    - Velikost operační paměti: 4 GiB
    - Verze VirtualBoxu (shodná s verzí SDK): 4.3.xx
    - Rychlost síťového připojení: 100 Mbit

Servery 1–3 se nacházely ve stejné síti a testy probíhaly zejména na těchto serverech. Na serverech byly vytvořeny virtuální stroje s různými operačními systémy a hardwarovými konfiguracemi. Některým virtuálním strojům byly přiděleny stejné názvy a byly registrovány do jedné skupiny, aby došlo k otestování toho, že aplikace správně rozlišuje servery. Testování probíhalo tím způsobem, že na každý server byly nainstalovány prerekvizity, které aplikace vyžaduje ke svému běhu, následně byly na server zkopírovány zdrojové soubory, a poté byly spuštěny testy. Tím došlo k ověření funkčnosti na systémech Windows a Linux. Ověřování výsledků testů bylo prováděno ručně, neboť nebyl nalezen jiný způsob, jak provádět ověřování operací, které byly prováděny na virtuálním stroji, např. vytvoření složky,

či souboru, spuštění aplikace nebo skriptu a jiné. Při testování byl kladen důraz zejména na správné provádění příkazů pro skupiny strojů, dávkové soubory a stabilitu celé aplikace.

Server 4 sloužil primárně pro zjištění míry kompatibility se staršími verzemi VirtualBox SDK. Připojení na webový server proběhlo standardním způsobem, avšak během pokusů o vykonání příkazů bylo zjištěno, že v SDK ve verzi 4.x nejsou implementovány některé metody, jež aplikace využívá, tudíž nelze využívat veškerou funkcionalitu. Pro plnohodnotné používání je doporučena verze SDK 5.x, mimo verzi 5.0.14. U této verze byly nalezeny velmi vážné problémy při práci s VirtualBox API v jazyce Python. Nebylo možné použití webové služby, protože zřejmě docházelo ke špatně serializaci objektů nebo chyběla implementace tzv. *wrapperů*[3], které jsou používány pro odesílání a příjímání dat přes síť. Tyto chyby byly opraveny v následující verzi SDK, tedy 5.0.16.

Během testování byla nalezena závažná chyba, která způsobovala to, že při selhání příkazů, které k provedení zamykaly sezení (angl. *session*) stroje, nebyly stroje odemčeny, a tudíž bylo zabráněno jakékoliv další práci s virtuálním strojem. V takový okamžik pak musela být restartována běžící služba VirtualBoxu nebo muselo dojít k odhlášení a přihlášení uživatele. Po nalezení zdroje této chyby, byla chyba úspěšně odstraněna.

Součástí testování bylo i měření rychlosti aplikace, zejména časové intervaly mezi odesláním a přijetím zprávy od webového serveru, které měly na rychlost aplikace největší vliv. V tabulce č. 7.1 lze vidět porovnání rychlosti při využití COM modelu a webové služby. Zatímco průměrná doba odpovědi serveru byla 1325,1 milisekund, u COM modelu je průměrná doba vykonávání příkazu rovna 10,2 milisekundám. Je zde viditelný rozdíl způsobený režií během serializace objektů a přenosu po síti, který provádí webová služba.

Číslo měření	COM [ms]	Webová služba [ms]
1	1	1 015
2	17	1 016
3	18	1 034
4	17	1 046
5	1	1 016
6	4	1 015
7	14	3 050
8	13	1 015
9	16	2 032
10	1	1 012

Tabulka 7.1: Měření doby odezvy pri použití COM modelu a webové služby.

Při testování vzdáleného provádění příkazů na virtuálních strojích bylo zjištěno, že tato funkcionalita není podporována na některých Linuxových strojích, neboť není možné vytvořit sezení pro virtuální stroj, které je nezbytné pro provádění příkazů. Z diskusních fór VirtualBoxu bylo zjištěno, že tento problém má více lidí. Lze tedy předpokládat, že tato funkcionalita není na některých systémech v současné době podporována. Na testovaných virtuálních strojích se systémy Microsoft Windows se tyto problémy neprojevovaly.

[3]Funkce, jejímž hlavním účelem je volání jiné funkce

Kapitola 8

Závěr

Cílem práce bylo vytvořit aplikaci, která umožní vzdálenou správu virtuálních strojů a tuto správu urychlit pomocí automatizace. Tento cíl se mi podařilo splnit. Uživatel má možnost pracovat s aplikací jako s automatizačním nástrojem nebo interaktivní konzolovou aplikací. Práce s programem v interaktivním režimu je usnadněna automatickým doplňováním příkazů, které je běžné při používání terminálu na Linuxových systémech. V rámci práce proběhlo měření rychlosti programu, která je oproti manuální správě strojů lepší a program je tedy možné využít v praxi, ovšem s omezeními, jež jsou popsána v podkapitole zabývající se testováním. Aplikace umožňuje slučovat virtuální stroje do skupin a nad těmito skupinami vykonávat hromadné příkazy. Podporované jsou často využívané operace jako spuštění, restartování, vypnutí, import nebo export virtuálních strojů. Uživatel může spustit skript nebo jiný program na virtuálním stroji a s nově vytvořeným procesem interagovat.

Během vytváření aplikace došlo ke krátkodobému přerušení vývoje, neboť VirtualBox SDK 5.0.14, jež bylo aktuální na začátku vývoje, obsahovalo programátorské chyby, které zamezovaly jeho použití. Tento defekt se týkal zřejmě pouze jazyka Python. Komplikace byly v následující verzi odstraněny a vývoj byl obnoven. Doba, během které nebylo možné aplikaci implementovat, byla věnována teoretickému studiu SDK, ze kterého byly získány znalosti, jež byly použity při následném vytváření programu.

Aplikace má velké možnosti rozšíření, mezi které patří například vytvoření grafického uživatelského rozhraní, které bude přívětivější pro běžného uživatele. Při vytváření grafického rozhraní bude důležité myslet na responsivitu aplikace, aby nedocházelo k situacím, kdy bude aplikace zaneprázdněna vykonáváním správy virtuálního stroje a nebude reagovat na požadavky od uživatele. Dalším možným rozšířením je použití vláken v programu, aby mohlo být prováděno více operací zároveň. Toto rozšíření by mělo největší uplatnění v případech, kdy jsou vykonávány operace nad skupinami strojů, protože skupina může obsahovat velké množství strojů a provedení operace na všech může trvat velmi dlouho. V neposlední řadě je možné rozšíření množiny podporovaných příkazů podle požadavků uživatele. Přidání nových příkazů je velmi jednoduché, neboť při návrhu programu jsem s tuto možnost bral na vědomí.

Všechna výše popsaná rozšíření je plánováno realizovat v blízké budoucnosti, neboť aplikace bude prakticky využívána v komerční firmě, která používá spoustu virtuálních strojů pro různé, ať už pro interní (testování, interní aplikace) nebo veřejné (internetové stránky, poštovní server) potřeby. Tato aplikace pomůže ušetřit čas zaměstnanců, kteří mají na svědomí udržování těchto strojů v použitelném stavu.

Literatura

[1] BRODKIN, Jon : *With long history of virtualization behind it, IBM looks to the future* [online]. Network World, duben 2009, [cit. 2016-03-10]. Dostupné z: http://www.networkworld.com/article/2254433/virtualization/with-long-history-of-virtualization-behind-it--ibm-looks-to-the-future.html

[2] *Case Study: Wonkwang University* [online]. FUJITSU, 2015, [cit. 2016-03-15]. Dostupné z: http://www.fujitsu.com/global/documents/about/resources/case-studies/CS_2015Jan_Wonkwang_University_Eng_v01.pdf

[3] *Citrix: About* [online]. [cit. 2016-04-18]. Dostupné z: https://www.citrix.com/about.html

[4] DICTIONARY.COM. *Cloud computing: Define Cloud computing at Dictionary.com* [online]. [cit. 2016-05-03]. Dostupné z: http://www.dictionary.com/browse/cloud-computing

[5] *z/VM – A Brief Review of Its 40 Year History* [online]. IBM Corporation, červenec 2012, [cit. 2016-03-10]. Dostupné z: http://www.vm.ibm.com/vm40hist.pdf

[6] *Joseph Cravotta* [online], [cit. 2016-04-25]. Dostupné z: https://josephcravotta.com/2015/12/installing-xencenter/

[7] *Component Object Model (COM)* [online]. Microsoft Corporation, 2016, [cit. 2016-04-20]. Dostupné z: https://msdn.microsoft.com/cs-cz/library/windows/desktop/ms680573(v=vs.85).aspx

[8] *Oracle Fact Sheet: Speed Innovation, Streamline IT* [online]. Oracle Corporation, říjen 2015, [cit. 2016-04-18]. Dostupné z: http://www.oracle.com/us/corporate/oracle-fact-sheet-079219.pdf

[9] Oracle Corporation : *Oracle VM VirtualBox Programming Guide and Reference [online]*. 2016, [cit. 2016-04-20]. Dostupné z: http://download.virtualbox.org/virtualbox/5.0.16/SDKRef.pdf

[10] Oracle Corporation : *Oracle VM VirtualBox User Manual [online]*. 2016, [cit. 2016-04-20]. Dostupné z: http://download.virtualbox.org/virtualbox/5.0.16/UserManual.pdf

[11] RUEST, Danielle a Nelson RUEST. *Virtualizace: Podrobný průvodce*. Brno: Computer Press, 2010. 408 s. ISBN 978-80-251-2676-9.

[12] *SoftoCoupon* [online], [cit. 2016-04-25]. Dostupné z: http://www.softocoupon.com/comparisons/vmware-player-vs-vmware-workstation-12.php

[13] *That Net Site* [online], [cit. 2016-04-25]. Dostupné z: http://thatnetsite.com/tag/virtual-pc/

[14] WALDEN, D. : 50th Anniversary of MIT's Compatible Time-Sharing System. *IEEE Annals of the History of Computing [online]*, roč. 33, č. 4, říjen-prosinec 2011: s 84–85, ISSN 1058-6180, [cit. 2016-03-12]. Dostupné z: http://muse.jhu.edu/article/464605

[15] Wikibooks : *QEMU — Wikibooks, The Free Textbook Project*. 2015, [cit. 2016-04-03]. Dostupné z: https://en.wikibooks.org/w/index.php?title=QEMU&oldid=2955283

[16] *XenServer: Open Source Server Virtualization* [online]. 2015, [cit. 2016-04-23]. Dostupné z: http://xenserver.org/

[17] YU, Yang; State University of New York at Stony Brook. *OS-level Virtualization and Its Applications*. ProQuest, 2007. 134 s. ISBN 9780549914075.

Přílohy

Seznam příloh

Příloha A

Obsah DVD

Přiložené DVD obsahuje následující adresáře:

- */doc* - obsahuje technickou zprávu ve formátu pdf,
- */tex* - obsahuje soubory se zdrojovými kódy technické zprávy včetně použitých obrázků,
- */src* - obsahuje soubory se zdrojovými kódy výsledné aplikace. Struktura adresáře je detailněji popsána v souboru readme.txt, který se nachází v této složce.