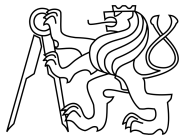
ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ



ZADÁNÍ DIPLOMOVÉ PRÁCE

Název:	Návrh platformy pro testování hotelových sítí
Student:	Bc. Tomáš Pokorný
Vedoucí:	Ing. Michal Valenta, Ph.D.
Studijní program:	Informatika
Studijní obor:	Webové a softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce zimního semestru 2016/17

Pokyny pro vypracování

Hotelové sítě se skládají z různých hw prvků a služeb. Služby nejsou vzájemně nezávislé. Odsud pochází potřeba testovací platformy realizace konkrétní hotelové sítě.

1. Analyzujte požadavky na platformu pro testování hotelových sítí. Zpracujte konceptuální návrh této platformy.
2. Proveďte diskusi technických prostředků pro realizaci této platformy. Pro jedno možné řešení rozpracujte návrh z bodu 1.
3. Navrhněte a implementujte prototyp dvou klíčových modulů zamýšlené testovací platformy - VPC (virtual PC) a VPMC (virtual PC manager) a jejich komunikační rozhraní (API).
4. Na vytvořeném prototypu otestujte a zdokumentujte několik testovacích scénářů. Scénáře k otestování prototypu konzultujte s vedoucím práce.

Další funkční a nefunkční požadavky:
- při návrhu se zaměřte na rozšiřitelnost a srozumitelnost API, bezpečnost prioritou není,
- pro implementaci uvažujte linuxovou platformu.

Seznam odborné literatury

Dodá vedoucí práce.

L.S.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

prof. Ing. Pavel Tvrdík, CSc.
děkan

V Praze dne 30. září 2015

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra softwarového inženýrství

Diplomová práce

Návrh platformy pro testování hotelových sítí

Bc. Tomáš Pokorný

Vedoucí práce: Ing. Michal Valenta, Ph.D.

1. července 2016

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 1. července 2016 .....

České vysoké učení technické v Praze
Fakulta informačních technologií
© 2016 Tomáš Pokorný. Všechna práva vyhrazena.
Tato práce vznikla jako školní dílo na Českém vysokém učení technickém v Praze, Fakultě informačních technologií. Práce je chráněna právními předpisy a mezinárodními úmluvami o právu autorském a právech souvisejících s právem autorským. K jejímu užití, s výjimkou bezúplatných zákonných licencí, je nezbytný souhlas autora.

Odkaz na tuto práci

Pokorný, Tomáš. *Návrh platformy pro testování hotelových sítí.* Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2016.

Abstrakt

Projekt si klade za cíl navrhnout a vytvořit prototyp platformy, která umožní automatizovat procedury spojené s testováním nově vytvářené sítě. Hlavním funkcí této platformy bude testování konfigurace sítě pomocí virtualizovaných počítačů, které mezi sebou budou komunikovat přez testovanou síť. V úvodu práce se budu zabývat platformou z konceptuálního pohledu. Dále prozkoumám možnosti virtualizace počítačů a síťových prvků. Třetí kapitola bude věnována realizaci platformy pomocí vybraných technologií. V poslední kapitole představím vytvořenou platformu z uživatelského hlediska a vytvořím ukázkové testy.

Klíčová slova virtualizace, síťování, testování, php, linux

Abstract

Goal of the project is to create a platform for automatic testing of network configuration. Platfrom will use virtualization computers on host machine. At the beginning, I will deal with the platform from a conceptual point of view. Further explore the possibilities of PC virtualization and software networking. The third chapter is devoted to the implementation of the platform with the

selected technologies. In the last chapter I will present a platform created from the user's point of view and create sample tests.

Keywords virtualisation, networking, testing, php, linux

Obsah

Seznam obrázků

Úvod

Zadavatelem práce je společnost Mikenopa, která dodává technologie a související služby pro hotely, business centra a rezidenční projekty. Jedna ze služeb je i zasíťování objektů. V hotelech jsou kromě připojení hostů k internetu používány sítě i v dalších případech. Od bezpečnostních IP kamer, IP telefony, televize, nebo i zámky na dveřích. Společnost dále poskytuje technické zázemí pro konference a přednášky. Zde se z pohledu síťování musí zajistit dostupnost připojení do internetu pro velké množství lidí najednou.

Tento projekt má do budoucna poskytnout možnost vytvořeni komplexních a automatizovatelných testů, ktere bude možné používat univerzálně v libovolné síti. Testy budou mít za cíl ověřit jak dostupnost jednotlivých síťových prvků, tak i funkčnost systémů jako přihlašování do sítě, omezení šířky pásma, a dalších, které v tuto chvíli nejsou známé. Platforma bude pomáhat technikům ověřit funkčnost sítě po její instalaci.

Cílem této práce je ananlyzovat požadavky na tuto platformu, prozkoumat možnosti virtualizace, implementovat prototyp platformy, a ten otestovat. Tomu odpovídá i rozvržení textu. První kapitola se věnuje analýze a konceptuální úrovni platformy. Drhuhá kapitola je zaměřena na výběr technologií identifikace jejich použití v platformě. Následující kapitola popisuje implementaci, a v poslední je popsáno použití a ukázáno na testech.

Kapitola 1

Analýza

Tato napitola je zaměřena na analýzu a konceptuální pohled na platformu.

V první sekci bude nejprve stručně popsána vize celého projektu 1.1. V sekci 1.2 jsou formalizovány požadavky. Je zmíněno nekolik nefunkčních požadavků, a další požadavky jsout přiřazeny k jednotlivým modulům. Protože se jedná o platformu na testování, bude v kapitole 1.3 rozvedeno, co v tomto kontextu znamená test. V následující kapitole bude stručně popsána platforma ze síťového pohledu 1.4.

Další kapitoly budou věnovány jednotlivým modulům, jak byly rozděleny ve vizi. Budou diskutovány jejich zodpovědnosti, propojení, a u některých i logika fungování.

První bude zmíněn modul VPCM 1.5.1, dále knihovna pro práci s VPCM nazvaná TestManager 1.5.2. Bude představeno Api 1.5.3 mezi těmito dvěma komponentami. Dále bude rozebráno testování z pohledu procesů 1.5.4 a jejich komunikace, v sekci 1.5.5 bude nastíněn způsob spoštění příkazů v prostředí jednotlivých VPC. Sekce Loader je věnována načítání testu, a v poslední sekci budou znázorněny moduly z pohledu rozložení na reálných strojích.

1.1 Vize

Zadavatel má určitou představu, jakou podobu by platforma měla mít, a jaké použití od ní očekává. Tato předtava byla dále rozvíjena na konzultacích a konečná verze je popsána zde.

Platforma bude spuštěná na testovacím počítači s více ethernetovými porty. Ty budou zapojeny do switchů v testované sítí. Na testovacím počítači budou pomocí virtualizace simulováni klienti sítě. Platforma bude testerům poskytovat api, které bude moci být použito v testovacích skriptech.

Projekt je logicky rozdělen na následující části.

virtual PC manager Virtual PC Manager bude spravovat virtuální počítače na jednom fyzickém stroji. Tyto virtuální počítače budou označo-

vány zkratkou VPC. VPC budou zapojeny přes síťové interfacy fyzického stroje do nějaké sítě. Přes tuto síť budou moci komunikovat s okolním, a tím simulovat komunikaci reálných klientů. Pro okolní síť se budou jevit jako další reální klienti, což je základní myšlenka celého toho projektu.

knihovna pro práci s VPCM Bude obalovat moduly VPCM a zpřístupňovat jejich funkce testovacím skriptům. Bude moci obsluhovat více VPCM najednou. Pro testovací skript bude prezentovat rozhraní, se kterým bude tvůrce testu pracovat přímo.

testovací framework Úkolem je zavedení testu při startu a distribuce testovacích souborů v případě spuštění na vzdálených počítačích.

procesní komunikace Pro případ, že test bude používat více procesů najednou, bude tento modul nabízet možnost komunikace a synchronizace procesů.

1.2 Požadavky

Nefunkční požadavky na platformu jsou následující.

- platforma bude implementována v jazyce PHP
- platforma bude provozována na systému Linux
- použitá virtualizace umožní vytvořit větší počet VPC
- platforma bude připravena na budoucí rozšíření o komunikaci s více HW jednotkami

V další části jsou shrnuty požadavky podle jednotlivých komponent.

1.2.1 Virtual PC Manager

- VPC bude možné přidělit více síťových karet (alespoň 2)
- VPC umožňuje síťové karty konfigurovat (přidělení IP, MAC,...)
- VPC budou viditelné z okolní sítě (nebude za NAT)
- VPC umožní simulaci webového prohlížeče pomocí Curl a wget
- VPC umožňuje spouštění skriptů nebo příkazů v prostředí virtuálního PC
- VPCM umožní vytvoření a odstranění VPC
- VPCM bude spravovat ethernetové porty fyzického počítače, na kterém je spuštěn

- VPCM umožňuje přidávání a odebírání vlanů na fyzickém počítači
- VPCM umožňuje přidávání a a odpojování síťových interfaců, které spravuje, s interfacy VPC
- VPCM umožňuje přidání dalších operací na VPC
- VPCM umožní spouštění procesů
- VPCM umožní běh více procesů a komunikaci mezi nimi

1.2.2 Knihovna pro prácí s VPCM

- poskytuje vnější rozhraní pro práci s VPCM modulem
- umožňuje práci s více VPCM najednou
- umožňuje spuštění procesu na libovolném VPCM
- umožňuje práci s VPC nezávisle na jejich umístění
- realizuje komunikaci více VPCM

1.2.3 Testovací framework

- umožňuje spuštění testu
- distribuje test na případné další VPCM
- inicializuje platformu

1.2.4 Procesní komunikace

- umožňuje zaslání zprávy mezi dvěma procesy
- umožňuje synchronizaci dvou procesů
- inicializuje platformu

1.3 Testy

Test je php skript, který používá Knihovnu pro prácí s VPCM, a je spouštěn modulem Testovací framework. Vlastní výstup z testu je generovaný samotným testovacím skript, a to například výpisy do souboru.

Testy je možné rozlišit na dva typy. Jsou to testy konfigurace sítě, které ověřují například dostupnost prvků sítě. Dále testy zátěžové, které mají prověřit správnost chování sítě při vysoké zátěži, nebo i při jejím přetížení.

V této sekci jsou podrobně rozvedeny některé příklady testů, které by do budoucna mělo být na platformě možno vytvořit. Některé z těchto testů

jsou příklady budoucího použití. Cílem této práce není je realizovat. Cílem je platformu navrhnout tak, aby do budoucna mohla být o tyto testy rozšířena.

test VLANu Test ověřuje, že 2 počítače na stejné síti ale v rozdílných VLANech jsou navzájem nedostupné.

1. Vytvoření 2 VPC
2. Vytvoření vlan 10 a vlan 20 na interfacu fyzického počítače
3. Připojení VPC1 na vlan 10 a VPC2 na vlan 20
4. Provedení DHCP requestu na VPC1 a VPC2
5. Počkání na dynamické přidělení IP adresy
6. Provedení ping příkazy z VPC1 na IP adresy síťového interfacu VPC2
7. Ověření že VPC2 je nedostupné

test přihlášení do sítě Test ověřuje funkci přihlášení do sítě. Dále ověřuje, zda po připojení klienta z jiného síťového interfacu je klient stále autorizován.

1. Vytvoření VPC
2. Připojeni síťového interfacu VPC na vlan 10 interface fyzického počítače
3. Přidělení IP adresy VPC interfacu pomocí DHCP žádosti
4. Dotaz na vebovou stránku přez webový prohlížeč
5. Kontrola, že došlo k přesměrování na captive portál
6. Odeslání vyplněného formuláře s platnými přihlašovacími údaji na captive portál
7. Kontrola, že došlo k přesměrování na původně dotazovanou webovou stránku
8. Odpojení síťového interfacu VPC
9. Připojení 2. síťového interfacu VPC
10. Přidělení IP adresy VPC interfacu pomocí DHCP žádosti
11. Dotaz na webovou stránku
12. Kontrola, že nedošlo k přesměrování na captive portál (Účet klienta nalezen pomocí cookies. Přestože klient přistupuje z jiné síťové karty, další přihlášení není nutné.)

test omezení šířky pásma Klient hotelu má zakoupenou rychlost připojení 100 mbit/s. Test má za úkol ověřit, zda se toto nastavení chová správně i v případě, kdy jsou data klienta posílána do více zařízení rozdílnými cestami.

1. Vytvoření N VPC
2. Připojení všech VPC do sítě a získání dynamické IP adresy
3. Přihlášení se pod stejného klienta
4. Generování provozu na klientech stažením souboru
5. Ověření, zda součet rychlostí přenosu dat odpovídá objednané šířce pásma

zátěžový test Test zjišťuje chování sítě v případě, kdy se v jednom časovém úseku pokusí o připojení větší množství klientů.

1. Vytvoření 100 VPC
2. Připojení všech VPC do sítě a získání dynamické IP adresy
3. Provedení přihlášení na všech VPC pod jinými klienty

1.4 Platforma ze síťového pohledu

Sítě mohou mít rozdílné struktry. Například jednoduché síťě v domácnostech se často skládají z jednoho routeru a více klienských zařízení. Všichni klienti tohoto routeru jsou v jedné podsíti. Pokud komunikují s čímkoli vně sítě, provádí se routování, tedy směrování na vrstvě L3 a překlad adres pomocí NAT.

V hotelových sítích bývá více prvků pracujících na linkové vrstvě L2. K rozdělení sítě na podsítě se hojně využívá technologie VLAN. Technologie VLAN je pro testovací platformu stěžejní a veškeré testy v reálném prostředí ji musejí podporovat. Tato technologie bude dále zmíněna v sekci 2.2.3.

Na obrázku 1.1 je znázorněna ukázka zapojení testovacího stroje, spuštěných virtuálních PC, a jejich zapojení do nějaké sítě.

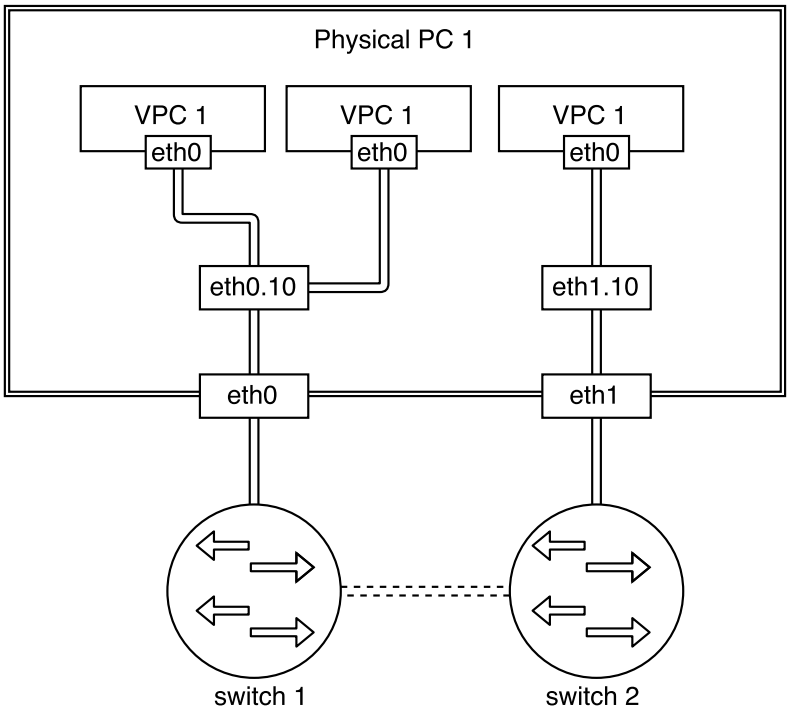
Virtuální PC mají alespoň jeden síťový interface. Přes tento interface jsou zapojeny do okolní sítě. Při pohledu z prostředí virtuálního počítače by okolní síť měla být dostupná stejně, jako by pc bylo reálné PC a místo přes softwarové směrovače hostovacího PC by bylo zapojeno do reálného switche.

Dále je zde znázorněn fyzický stroj. Ten může mít větší množství síťových karet, a tedy i síťových interfaců. Protože všechen provoz má být tagovaný, na techto interfacech budou vytvářeny VLANy. Do nich budou zapojovány jednotlive virtuální PC. Veškerou softwarovou konfiguraci síťových interfaců, vlanů, irtuálních PC a jejich propojení má na starosti modul VPCM.

Síťové interfacy fyzického stroje jsou zapojeny do reálné sítě, která je v tomto případě znázorněná dvěma switchi, a nějakým nedefinovaným spojením mezi nimi.

1.5 Části platformy

V této sekci budou podrobněji předtaveny komponenty platformy.



Obrázek 1.1: Ukázka prostředí testovacího stroje

1.5.1 VPCM

Modul VPCM bude spravovat reálný hardware. Bude na něm přímo provádět operace a měnit jeho nastavení.

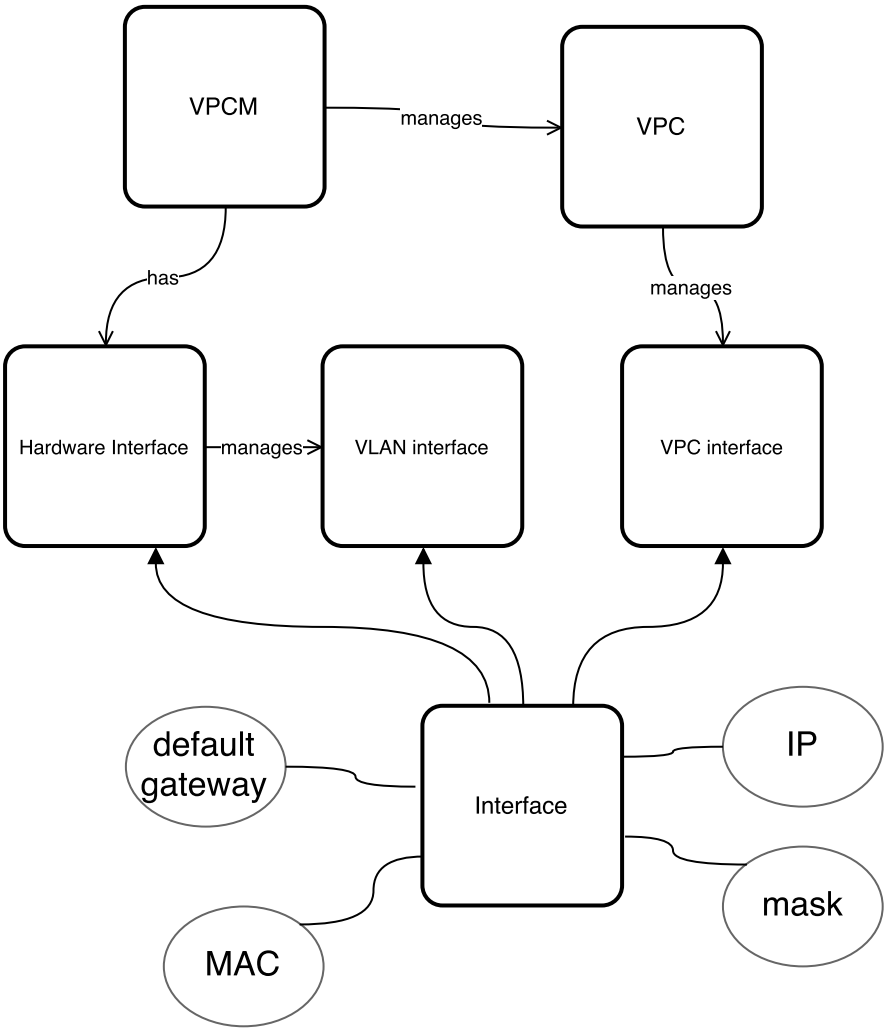
Jeho entity jsou zachyceny na obrázku 1.2. Tento model se může mírně změnit na základě zvolených technologií, které jsou v tuto chvíli neuvažovány. Je zde VPCM, což je reprezentace nějakého virtuálního stroje, se kterým se pracuje. VPC má nějaká síťová rozhraní. Přístup k objektům VPC je umožněn entitou VPCM. Ta dále spravuje síťová zařízení na zařízení.

Důležité informace ponese entita Interface, která je rozdělena na 3 typy objektů. První je interface uvnitř VPC, dalším je reprezentace rálného interfacu na PC, a posledním je reprezentace VLAN interfacu. Kardinalita vztahů mezi objekty jse stejná jako mezi příslušnými objekty hardwarového modelu.

Modul bude zpřístupňovat práci se svymi entitami přez API, které je zmíněno dále.

1.5.2 Test Manager

Modul knihovny pro práci s VPCM byl kratším názvem pojmenován TestManager. Zobrazen je na obrázku 1.3.



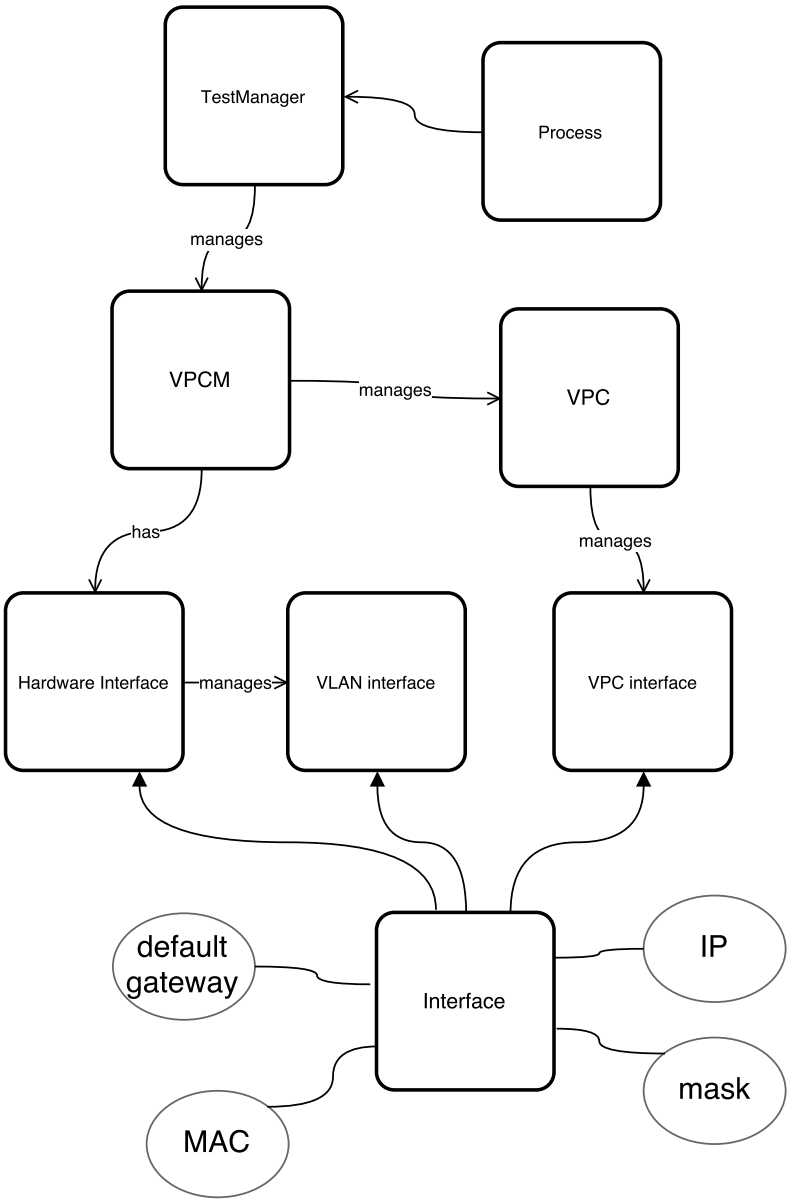
Obrázek 1.2: Analytický model součástí VPCM modulu

Je zde nová entita TestManager, která má za úkol umožnit přístup ke všem dostupným VPCM, a nějak s nimi komunikovat. Modul TestManager nabízí objekty pro použití v testovacích skriptech. Objekty mohou reprezentovat i entity na jiných strojích. Úkolem TestManageru je transformovat volání operací na svých objektech na volání metod správného VPCM.

Modul zpřístupňuje funkce modulu VPCM, bude tedy muset nějak reprezentovat i jeho objekty. Nejintuitivnější přístup je pro každou entitu VPCM vytvořit entitu realizující její obraz v TestManageru.

Hierarchie entit je následující.

- Nejvýše postavená je entita TestManager. Ta zpřístupňuje testu objekty PCM a provádí jejich inicializaci.



Obrázek 1.3: Analytický model součástí TestManageru

- Další je entita PCM. Ta reprezentuje jednu hardwarovou jednotku, na které umožňuje testu nějak pracovat s PC a konfigurovat své síťové interfacy.
- Podřazená PCM je entita PC. Ta reprezentuje jedno virtualizované PC. Testu umožňuje provádět operace na PC a souštět procesy uvnitř virtuálního stroje.

Entita Process bude muset používat modul TestManager. Pouze ten uchovává informace o dalších VPCM. Pokud bude proces vyžadovat spuštění jiného procesu nebo komunikaci s jiným procesem na jiném VPCM, bude muset přistupovat přes TestManager. Entita Process je rozvedena v sekci 1.5.4.

1.5.3 Pcm Api

Pcm Api je rozhraní, které definuje způsob komunikace mezi VPCM a TestManagerem. Důvodem zavedení tohoto rozhraní je sjednotit přístup TestManageru k VPCM implementacím, kterých může být více.

V první fázi projektu bude vytvořena pouze lokální implementace. Bude na stejném stroji, a TestManager k ní může přistupovat přímo.

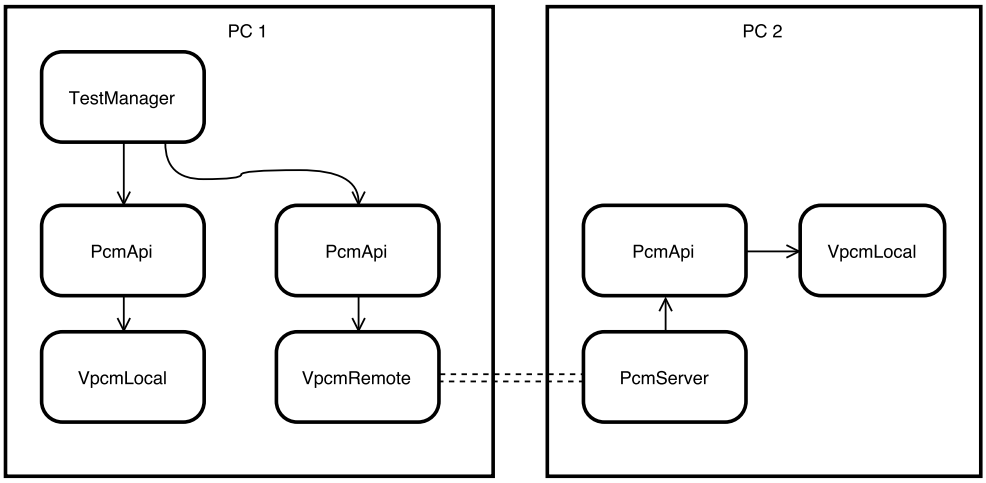
Do budoucna se počítá se zavedením možnosti ovládat jedním TestManagerem více vzdálených VPCM. PcmApi má za úkol definovat metody, které by nová síťová implementace přístupu k VPCM respektovala.

Výsledný návrh závislosti modulů TestManager a VPCM je zobrazen na obrázku 1.4. Je zde znázorněno použití modulu PcmAPi se dvěma rozdílnými implementacemi VPCM. První je lokální implementací, která přímo implementuje rozhraní PcmApi. Druhá VPCM je síťové a je rozděleno na klientskou a serverovou část. Klientská část má za úkol se připojit k serverové, předat jí volání funkcí z api, a vrátit výsledek. Jak to bude dělat je z pohledu TestManageru nepodstatné. Serverová část obdrží volání, zpracuje ho na volání funce na svém lokálním api, pošle výsledek zpět klientovi.

Takováto architektura má několik důsledků.

1. TestManager nemusí rozlišovat, s jakou implementací pracuje. O VPCM musí mít pouze informaci, jak se připojit, a jakou z implementaci rozhraní má použít.
2. Veškeré operace TestManageru, které pracují s VPCM, musí být součástí PcmApi. TestManager musí počítat s tím, že implementace, se kterou pracuje, může být spuštěna v jiném prostředí, které je pro něj jinak než přes api nedostupné.

Toto rozhraní musí použít i ostatní moduly, které potřebují přistupovat do objektů pod jiným VPCM. Modul meziprocesní komunikace jej použije pro komunikaci s procesy, a modul Loader jej musí použít k distribuci případných souborů, které je třeba ke spuštění testu.



Obrázek 1.4: Použití PCM API

1.5.4 Procesy

Je požadováno, aby platforma podporovala běh více procesů současně. Tyto procesy mohou simulovat práci uživatelů sítě. Procesy mohou provádět operace s více VPC najednou, takže budou muset pracovat v hostovacím PC a ne uvnitř některého z virtuálních.

Každý spuštěný testovací skript je procesem. Aby byl proces v systému adresovatelný, bude mu přidělen nějaký identifikátor. Po vytvožení dalšího procesu by měly mít procesy možnost se sebou komunikovat. To znamená, aby vytvářený proces měl kontakt na svého rodiče, a rodič by měl mít kontakt na svého potomka.

Pro prototyp bude uvažována komunikace pouze lokální, nicméně do budoucna by měla být platforma rozšířitelná o vzdálené počítače, takže bude třeba i vytváření procesů a komunikace mezi procesy přes síť.

Z pohledu vytvářeného API byly navrženy následující funkce:

run process tato operace musí zajistit:

- předání identifikátoru rodičovského procesu nově vytvořenému procesu
- předání identifikátoru nového procesu rodičovskému procesu
- lokalizaci správného vpcm, kde má být proces spuštěn
- spuštění zadaného skriptu jako nový proces
- předání parametrů procesu

send odešle zprávu procesu specifikovém pomocí id

recieve přijme zprávu od nespecifikovaného procesu

Kvůli požadavku na možnost spuštění a práci s procesy na cizích VPCM musí být operace vázány na modul PcmApi.

Data přenášená těmito metodami jsou textová, nicméně nic nebrání do textu zakódovat složitější objekty nebo pole pomocí serializace, převedení do JSON, nebo libovolné jiné metody.

Pro synchronizaci procesů můžeme zavést další operace, jako synchronizace pomocí bariery, nebo se inspirovat knihovnami jako pthread, a metodami jako `join()`, která čeká na ukončení threadu.

Základní synchronizaci je možné zavést už pomocí operací `send` a `recieve`. Obě čekají, až bude přenos zprávy dokončen, a až potom se ukončí. Mohou nastat dvě situace:

1. Proces A zavolá operaci `send`, ale adresát, proces B, je nedostupný. Operace `send` se zastaví a čeká odpověď. Tu dostane až v okamžiku, kdy proces B zavolá operaci `recieve`, a zpráva pak může být doručena.

2. Proces A zavolá `recieve`. Pokud už nějaký proces zavolal `send` s cílem procesu A, tak předá zprávu, a obě operace se ukončí. Pokud žádný odesílatel není, `recieve()` blokuje, dokud se nějaký neobjeví.

1.5.5 Atomické operace

Název atomická operace byl zvolen pro spouštění operací na platformě. První způsob spouštění externích externích skriptů je runProcess(), který je omezen naspouštění php skriptů s nějakým testem používajícím TestManager. Atomická operace je univerzálnější.

Má za úkol umožnit zadávání příkazů do příkazové řádky, a to buď hostitelského počítače, nebo do příkazové řádky virtuálních počítačů. Tím bude docíleno provádění operací z prostředí virtuálních PC. Na rozdíl od `run process` je synchronní a čeká na vrácení výsledku operace.

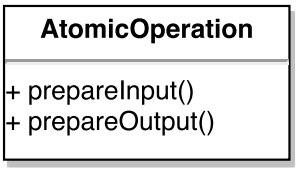
Všestranost této metody spočívá v tom, že pomocí ní lze provést jak jednoduchý příkaz vracející nějakou hodnotu, tak i spustit operactrvající delší dobu, nebo spustit nějaký proces na pozadí.

Jádrem této operace je nějaký příkaz. Pro jednoduší práci s ním je obalený objektem, který má za úkol zpracovat parametry operace, a po jejím provedení zpracuje výstup operace do nějaké požadované podoby.

Základní rozhraní atomické operace lze shrnout jako třídu s dvěmi funkcemi z obrázku 1.5.

Funkce `prepareInput()` přijímá proměnný počet argumenů, ze kterých sestaví příkaz, a ten vrátí. Příkaz je předán do VPC, který ho provede. Výstup po provedení příkazu je předán zpět do `prepareOutput()`. Ten jej zpracuje a vrátí zpět volajícímu skriptu.

Operace zahrnují spouštění na nejnižší vrstvě. Proto budou muset být prováděny modulem VPCM.



Obrázek 1.5: Rozhraní atomické operace

1.5.6 Loader

Posledním nezmíněným je modul původně označený jako testovací framework. Jeho úkolem je nastartování testu. Testovací skript bude používat výše zmíněné moduly jako Testmanager a Process. Jejich inicializace je prací modulu Loader.

Dále by mohl modul sloužit k vyčištění konfigurace a ukončení všech běžízích procesů v momentě, kdy skript skončí.

1.5.7 Deployment

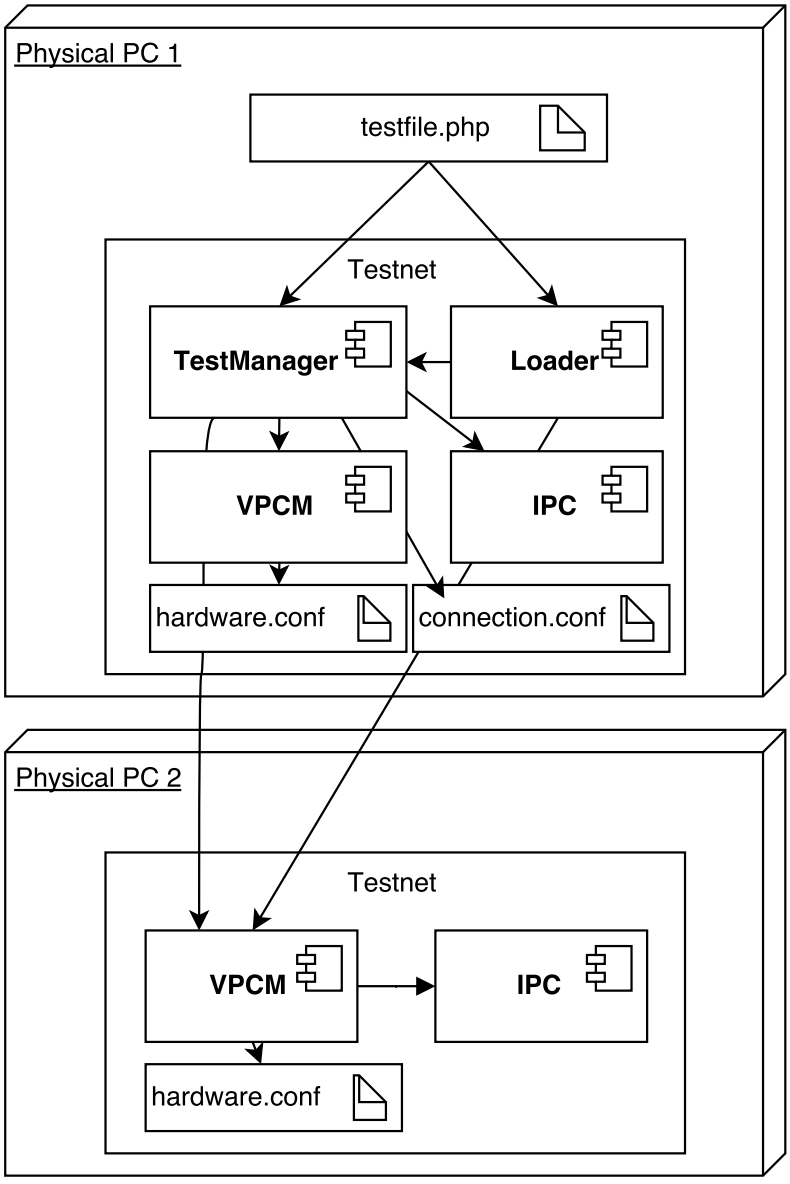
Konkrétní příklad rozložení a vazby komponent pri běhu na dvou HW zarízeních znázornuje obrázek 1.6.

Přibyla zde nutnost komukace s dalším VPCM, který se nachází na jiném fyzickém stroji. Tento stroj je třeba nejak identifikovat a navázat s ním spojení. Proto zde vznikl další konfiguracní soubor.

Konfigurační soubory jsou tedy 2.

- První je zde pojmenován `hardware.conf`. Bude obsahovat data pro lokální VPCM. Položky pro konfiguraci budou identifikovány v sekci zabývající se technologiemi. Každá instance VPCM potřebuje svůj konfigurační soubor.
- Druhý soubor zde označený `connection.conf` používá TestManager. Jeho účelem je umožnit TestManageru identifokovat a připojit se k instancím VPCM. Bode tedy obsahovat seznam dostupných VPCM.

Z grafu dále vyplývá, že moduly Loader a TestManager jsou potřebné pouze na stroji, kde pracuje test. Vzdálené implementace potřebuje pouze modul VPCM a Process.



Obrázek 1.6: Ukázka rozložení komponent pro 2 stroje

Kapitola 2

Technologie

V této kapitole nejprve zmíním několik technologií přímo ze zadnání. Jedná se IP, MAC, a VLANy.

Ve druhé části této kapitoly bude rozebráno několik možností realizace VPC modulu. U vybrané technologie bude podrobně popsáno, jak ji bude platforma používat.

2.1 PHP jako základ platformy

PHP je skriptovací jazyk, určený především pro tvorbu dynamicky generovaných webových stránek. Lze jej ale použít jako plnohodnotný skriptovací jazyk nejen pro účely webů. V tomto případě bude použit pro vytváření skriptů pro příkazovou řádku. Ve skriptech je dostupná všechna funkcionalia jako u webového použití.

Pro spouštění shellových příkazů je k dispozici několik metod [1]. Nejvhodnější jsou metody `exec()` a `shell_exec()`. Druhý jmenovaný vrací celý výstup příkazu jako string a `exec()` vrací pouze poslední řádek výstupu. Nicméně u `exec()` lze použít parametry, které budou použity pro vrácení dalších hodnot. Druhý parametr je použit pro vrácení celého výstupu jako pole řádek, a třetí pro návratovou hodnotu příkazu. Příkaz `exec()` umí tedy to samé a více než `shell_exec()`.

Obě tyto funkce čekají na dokončení příkazu, aby mohly vrátit výstup. To lze obejít na úrovni bashového skriptu. Výstup lze odstranit pomocí `/dev/null` a pokud je třeba skript spustit v pozadí, stačí použít `&` na konci příkazu.

V tomto projektu je `exec()` základem pro práci se síťovými zařízeními a ovládáním VPC, protože skriptu zpřístupňuje veškeré nástroje a programy, které mají commandline rozhraní.

[1] PHP: Program execution Functions - manual. 2016. URL: http://php.net/manual/en/ref.exec.php

Některé nástroje a příkazy využadují root oprávnění. To lze získat buď zapnutím samotného skriptu s root oprávněními, nebo pro daného uživatele nastavit bezheslové spouštění pod rootem pro vybrané příkazy. To se nastavuje v souboru `/etc/sudoers`. Pro zjednodušení bylo řečeno, že skripty budou spuštěny vždy pod rootem.

2.2 Síťové technologie

Zde bude představeno několik síťových pojmů, které byly zmíněny v zadání projektu.

2.2.1 MAC adresa

MAC adresa, někdy nazývaná fyzická adresa, je identifikátor síťového interfacu složený z 12 hezadecimálních číslic, sloužící 2 vrstvě modelu OSY. Používá ji nejen ethernet, ale Wi-Fi, Bluetoth, Token-Ring a další.

Síťové interfacy mají již od vytvoření přidělenou nějakou MAC adresu [2]. Ta je přidělena výrobcem, a je globálně unikátní. V síťových kartách ji lze změnit, a tak může nechtěně vzniknout kolize.

Další způsob přidělení MAC adresy je při vytvoření virtuálního síťového interface, kde se o přidělení stará systém. Ten by se měl o unikátnost postarat.

Délka adresy je 48 bitů. V desítkové soustavě je tedy možných MAC adres řádově `10^14`. I když některé sítě mohou být rozsáhlé, pravděpodobnost vzniku kolize je minimální. V případě, že by kolize nastala, chování sítě je nedefinované.

Pro práci s MAC adresami lze použít příkazy z package `iproute2` [3]. Pro zjištění MAC adresy vybraného síťového interfacu použijeme přikaz

```
ip link show [interface]
```

jehož výsledkem je výpis s hledanou informací podobný

```
link/ether 08:00:27:cd:ad:b2 brd ff:ff:ff:ff:ff:ff
```

Pro změnu MAC adresy musí být zařízení vypnuté. Přepneme ho tedy do stavu `down`, změníme adresu, a zařízení znovu zapneme. Docílíme toho sekvencí příkazů

```
ip link set dev [interface] down
ip link set dev [interface] address [MAC address]
ip link set dev [interface] up
```

[2] Ethernet MAC address Assigments. 2016. URL: http://www.erg.abdn.ac.uk/users/gorry/course/lan-pages/mac-vendor-codes.html

[3] Iproute2. 2016. URL: http://baturin.org/docs/iproute2/

2.2.2 IP adresa

Pro zjištění Ip adresy interfacu můžeme použít množství příkazů, kde nejužívanější je `ifconfig [interface name]`, které vrátí detailní informace o nastavení interfacu.

Od vyvíjené platformy je požadována možnost nastavenní IP adresy staticky i dynamicky.

2.2.2.1 Statické přidělení

Pro statické přidělení a odebrání adresy můžeme znovu použít příkaz `ifconfig`

```
ifconfig [interface name] [ip address]
ifconfig [interface name] delete [ip address]
```

nebo pomocí novějšího příkazu z balíčku `iproute2`

```
ip addr add [ip address] dev [interface name]
ip addr del [ip address] dev [interface name]
```

2.2.2.2 Dynamické přidělení

Dynamické pridělení adresy je realizováno pomocí protokolu DHCP [4], [5]. Protokol používá architekturu klient - server. Server je správce adres na lokální síti a klient se dotazuje na přidělení adresy. DHCP poskytuje klientovi lokální sítě IP adresu, defaultní gateway, adresy DNS serveru a další případné nastavení.

Komunikace mezi nimi probíhá za použití zpráv

1. DHCP discovery – klient pošle broadcast zprávy, aby objevil dostupné DHCP servery
2. DHCP offer – server po obdržení zprávy odpoví nabízenopou adresou a dalším nastavením
3. DHCP request – klient potvrdí serveru zájem o nabízené nastavení
4. DHCP ack – server klientovi potvrdí přidělení IP spolu s dalším nastavením

Pro účely použití z příkazové řádky je v linuxu dostupný nástroj `dhclient`. Pro získání adresy stačí zapnout tento příkaz bez parametrů.

Pokud je tento příkaz zapnut na počítači s více ethernetovými porty, `dhclilent` se pokusí získat adresy pro všechny interfacy. Specifikovat konkrétní interface pro dynamické přidělení adresy lze zadáním jména interfacu

[4] dhclient(8) - Linux man page. 2016. URL: http://linux.die.net/man/8/dhclient

[5] Dynamic Host Configuration Protocol (DHCP). 2016. URL: https://help.ubuntu.com/lts/serverguide/dhcp.html

nebo v konfiguračním souboru. Používá se `/etc/dhcp/dhclient.conf`, nebo lze specifikovat cestu k jinému konfiguračnímu souboru pomocí příkazu Tenlze pomocí parametru `-cf`.

Další užitečné parametry jsou `-r` (release) pro odebrání přidělené adresy, a podrobnější informace o běhu získáme pomocí `-v`. Zde je ukázka použití z příkazové řádky.

```
dhclient -v enp0s3
Listening on LPF/enp0s3/08:00:27:cd:ad:b2
Sending on  LPF/enp0s3/08:00:27:cd:ad:b2
Sending on  Socket/fallback
DHCPDISCOVER on enp0s3 to 255.255.255.255 port 67 interval 3 (xid=0
    x8092eb1a)
DHCPREQUEST of 10.0.2.15 on enp0s3 to 255.255.255.255 port 67 (xid=0
    x1aeb9280)
DHCPOFFER of 10.0.2.15 from 10.0.2.2
DHCPACK of 10.0.2.15 from 10.0.2.2
bound to 10.0.2.15 -- renewal in 36184 seconds.
```

2.2.3 VLAN

Další technologií, která je zahrnuta přímo v základním požadavku na síťovou stránku platformy, jsou VLANy.

VLAN je zkratka pro Virtual Local Area Network. Je to technologie umožňující rozdělení klientů lokální sítě na skupiny. Klienti spolu mohou komunikovat, jako by byli v lokální síti, a přitom jsou jako skupina odděleni od ostatních skupin. Skupiny jsou identifikovány VLAN ID.

Z pohledu platformy jako koncového zařízení znamená připojení do sítě s VLANy nutnost pracovat s tagovanými ethernetovými packety definovanými standardem IEEE 802.1Q [6]. [7].

Linux umožňuje na jednom portu pracovat s více VLANy najednou. Zde je příklad vytvoření interfacu na eth0.2, který přijímá packety s VLAN ID 2 poslané na interface eth0. Druhý příklad je na odstranění tohoto interfacu.

```
ip link add link eth0 name eth0.2 type vlan id 2
ip link delete eth0.2);
```

S těmito interfacy je možné pracovat stejně, jako s obyčejnými interfacy. Lze je přiřazovat bridgi a podobně.

2.3 Virtualizace PC

V této sekci jsou vypsány uvažované technologie na realizaci VPC. U zvolené technologie je rozpracováno zapojení do sítě tak, jak bylo požadováno v 1.4.

[6] VLAN (4) – standard 802.1Q. 2003. URL: http://www.svetsiti.cz/clanek.asp?cid=VLAN-4–standard-8021Q-2242003

[7] IEEE 802.1Q VLAN Tutorial. 2006. URL: http://www.microhowto.info/tutorials/802.1q.html

2.3.1 Plná virtualizace

První z možností virtualizace PC a simulace klientů je plná neboli nativní virtualizace [8]. Hostovaný virtuální systém v tomto případě nemá informaci, že běží ve virtuálním prostředí. Pracuje se simulovanými hardwarovými zařízeními, které mu poskytuje hostitelský operační systém. Tato virtualizace je použita systémy jako Oracle VirtualBox nebo VMware Player. Pro realizaci řešení byl uvažován Oracle VirtualBox

Kromě grafického prostředí nabízí VBox také další api. Pro užití v platformě bylo pro nejpraktičtější užití uvažováno commandline interface [9]. Pomocí tohoto interface je možné konfigorovat vše, co nalezneme v GUI verzi. Zde je ukázka příkazů, které by používala platforma.

```
VBoxManage createvm --name [vpc name] --ostype [os] --register
VBoxManage modifyvm [vpc name] --nic1 bridged --bridgeadapter1 eth0
VBoxManage storageattach [vpc name] --storagectl "SATA Controller" --
    port 0 --device 0 --type hdd --medium [hard_drive].vdi
VBoxManage startvm [vpc name]
```

Pro vytvoření VPC je třeba více kroků. V ukázce je pouze vlastní vytvoření stroje pro VirtualBox, zapojení do sítě přes bridge do `eth0` hostitele, a přiřazení virtuálního disku.

Předpokládá se, že na disku `[hard_drive].vdi` je nainstalovaný a nakonfigurvaný operační systém. Po zapnutí by se do tohoto systému připojovalo.

V úvahy připadaly možnosti:

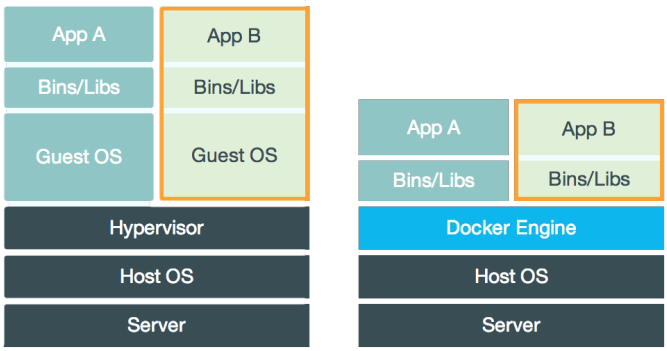
- Připojení prostřednictvím `ssh`
- Zapnutí webového serveru, na který by VPCM posílal samostatné požadavky. Server by byl na disku předkonfigurovaný a zapínal by se se startem systému.

Řešení použitím VirtualBoxu přináší řadu problémů.

- Pro změnu připojení VPC je třeba, aby byl VPC vypnut.
- Spuštění stroje probíha delší dobu, protože se načítá kompletní linuxový systém, třebaže bez grafické nástavby.
- Každá instance používá disk. Pro sdílení jednoho disku jsou k dispozici Snapshoty, které vyžadují další správu.
- Řešení je těžkopádné a hardwarově náročné. Neumožňuje větší množství VPC, protože v každém je načtena instance operačního systému.

[8] Virtual machines. 2016. URL: https://www.virtualbox.org/wiki/Virtualization

[9] Controlling VirtualBox from the Command Line. 2014. URL: https://community.oracle.com/docs/DOC-890092



Obrázek 2.1: Prorovnání Dockeru a plné virtualizace

2.3.2 Konteinerova virtualizace

Jiným druhem virtualizace je virtualizace konteinerová. Jako zástupce nástrojů využívající tuto technologii byl zvolen Docker. Následující text se bude vztahovat k němu, nicméně existuje řada dalších nástrojů, které jsou podobné. Příkladem takových jsou OpenVZ, CoreOS nebo Rocket.

Na Docker můžeme pohlížet jako na nenáročný virtuální stroj. Můžeme se do něj připojit přez SSH. Z vnitřního pohledu máme k dispozici to samé, jako bychom měli klasický virtuální stroj jako VirtualBox. Má vlastní procesy a jejich správu, síťové interfacy a nástroje pro jejich správu, a můžeme na něm spouštět cokoli stejně jako na jiných virtuálních strojích.

Na druhou stranu se od VM liší. Používá hostitelův operační systém, a tedy neumožňuje spustit jiný operační systém. Skládá se v jádru jen z procesů s vlastním nastavením. Tyto procesy, a všechny ostatní běžící v kontainerech, jsou z hostitelského systému viditelné.

Na obrázku 2.1 převzatého je z [10] článku je znázorněn rozdíl mezi klasickými VM vlevo a Dockerem vpravo. V porovnání s klasickými virtuálními stroji je toto řešení neporovnatelně rychlejší a méně náročné na hardware. Zapnutí konteineru je srovnatelné se zapnutím procesu, zatímco zapnutí VM znamená bootování nového systému.

Díky tomu je Docker vhodný ke spouštění aplikací ve vlastních prostředích, které se neovlivňují. Další použití může být simulace produkčního prostředí pro testovací učely. Použitím Dockeru lze vytvořit i složitou infrastrukturu v jednom PC, která by bez použití kontejnerové virtualizace byla příliš náročná.

[10] Docker – A highly Portable and Lightweight software container. 2015. URL: http://www.shadowandy.net/2015/05/docker-a-highly-portable-and-lightweight-software-container.htm

2.3.2.1 Cgroups

Jednou ze základních technologií, na kterých je Docker postaven, jsou Control groups, zkráceně cgroups. Původní název „process containers“ [11] je podle mého názoru výstižnější.

Je to nástroj na sledování a nastavování limitů pro alokaci zdrojů procesy nebo jejich skupinami. Zdroji jsou v tomto případě využití CPU, paměti, vstupních nebo výstupních operací a dalších.

2.3.2.2 Namespaces

Další stěžejní technologií Dockeru jsou Namespaces [12]. Cgroups slouží ke sledování a limitaci zdrojů procesu, namespacy slouží ke zpřístupnění nebo odepření těchto zdrojů. Z pohledu procesů mění pohled na systém. Existuje několik namespaců, kdy každý proces má přiřazen vždy jednu instanci každého.

Zde je výčet namespaců a jejich význam.

pid Process IDs

: Procesy vidí pouze procesy ve stejném namespacu. PID mohou být vnořovány do sebe. V takovém případě má proces více PID.

net Network devices, stacks, ports, etc

: Poskytuje skupině procesů vlastní síťové nastavení. Týká se routovacích tabulek, iptabulek a síťových interfaců.

mnt Mount points

: Poskytuje procesům možnost mít vlastní `root`, podobně jako u `chroot`, a vlastní připojené disky.

uts Hostname and NIS domain name

: Možnost izolace identifikátorů hostname a domainname.

ipc System V IPC, POSIX message queues

: Izoluje prostředky pro meziprocesovou komunikaci, sdílenou paměť a semafory.

user User and group IDs

: Izoluje různé identifikátory a atributy uživatelů a uživatelských skupin.

Každý proces zná ID všech svých výše zmíněných namespaců. Pokud ID nemá, znamená to, že je v globálním namespacu. Procesy svůj namespace dědí od rodiče.

[11] Process containers. 2007. URL:http://lwn.net/Articles/236038/

[12] namespaces(7) - Linux manual pages. 2016. URL: http://man7.org/linux/man-pages/man7/namespaces.7.html

Docker je komplexní nástroj určený k izolaci aplikací v kontejnerovém prostředí. Ty pak mohou být zabaleny a distribuovány spolu s prostředím. Přináší mnoho možností a funkcí, které jsou pro platformu zbytečné. Z použitých nástrojů by pro platformu bylo dostačující použití síťového namespacu.

2.3.3 Network namespace

Síťové namespacy [13] slouží k vytvoření různých nastavení síťových interfaců a směrovacích tabulek pro různé procesy. Tato nastavení se vzájemně neovlivňují.

Některé nástroje používají konfiguraci, která je uložena v souborech, například v adresáři `/etc/`. Toto nastavení síťové namespacy sdílí, protože sdílí celý filesystém. Pokud chceme takovouto konfiguraci změnit pro konkrétní namespace, je konvence používat cestu `/etc/netns/[ns name]/`. Pokud hledaný soubor na této cestě není, použije se ten z `/etc/`. Například pro změnu DNS s defaultním konfiguračním souborem `/etc/resolv.conf` lze pro nastavení v konkrétním namespacu použít `/etc/netns/[ns name]/resolv.conf`.

Pro prácí se síťovými namespacy je jednoduché rozhraní:

```
ip netns add [ns name]
ip netns del [ns name]
ip netns exec [ns name] command...
ip netns set [ns name] NETNSID
ip netns identify PID
ip netns pids [ns name]
```

Provádět příkazy a spouštět procesy uvnitř namespacu lze pomocí `exec`. Dále jsou zde příkazy pro identifikaci procesů. `identify` vrací jméno namespacu pro zadané PID procesu a `pids` vrací výčet PID procesů, které pracují v zadaném namespacu. Vytváření a mazání namespaců je zpřístupněno funkcemi `add` a `del`, které přijímají pouze jméno.

Na základě zhodnocení vypsaných možností byly pro realizace VPC zvoleny síťovými namespacy. V následujících podsekcích je popsána realizace zapojení do sítě tak, jak byla požadována v zadání.

2.3.3.1 Přidání interfacu

Po vytvoření nemá namespace žádné síťové rozhraní. Je třeba jej nějak zapojit. Pro to je zde příkaz `set`, kterým lze přesunout síťový interface z defaultního do zadaného namespacu. Namespace se pak stává vlastníkem interface, a ten přestává být z globálního namespacu dostupný. V případě použití na nějaký síťový interface hostitele by to znamenalo, že by tento interface mohl být pouze

[13] namespaces(7) - Linux manual pages. 2016. URL: http://man7.org/linux/man-pages/man8/ip-netns.8.html

v jednom VPC, protože po jeho přiřazení by v defaultním namespacu přestal existovat, a nemohl by tedy být znovu přiřazen.

Takové chování by bylo použitelné pro testovací Wi-Fi interface. U Wi-Fi by VPC měly mít přístup přímo k interfacu. V první fázi má však VPC poskytovat pouze ethernetové připojení, pro které je toto zapojení nevhodné.

2.3.3.2 Veth

Pro propojení 2 síťových zařízení Linux nabízí kostrukt `veth` neboli virtual ethernet.

Virtual ethernet je z pohledu uživatele pár virtuálních síťových interfaců, který je vzájemně propojen. Co je posláno do jednoho, to se objeví v druhém. Pro vytvoření je dostupný příkaz

```
ip link add [tap0] type veth peer name [tap1]
```

který vytvoří pár interfaců se jmény `tap0` a `tap1`. S těmi se dá pracovat jako s klasickými interfacy, tedy i přiřadit do namespacu pomocí `set` nebo přidat do bridge.

Z pohledu namespaců ještě zmíním chování veth páru při zapojení do namespacu. Veth pár, který má jeden interface uvnitř namespacu, a tento namespace je odstraněn, bude automaticky odstraněn celý.

2.3.3.3 Bridge

Linux pro vnitřní směrování nabízí softwarovéhý bridge. Jeho možnosti se však podobají možnostem switche.

Tento softwarový bridge bude použit pro připojení interfacu hostitele do více VPC. Pro práci v linuxu lze použít nástroj `brctl` nebo balíček `iproute2` s příkazem `ip link`, z něhož jsou pro platformu významné tyto příkazy.

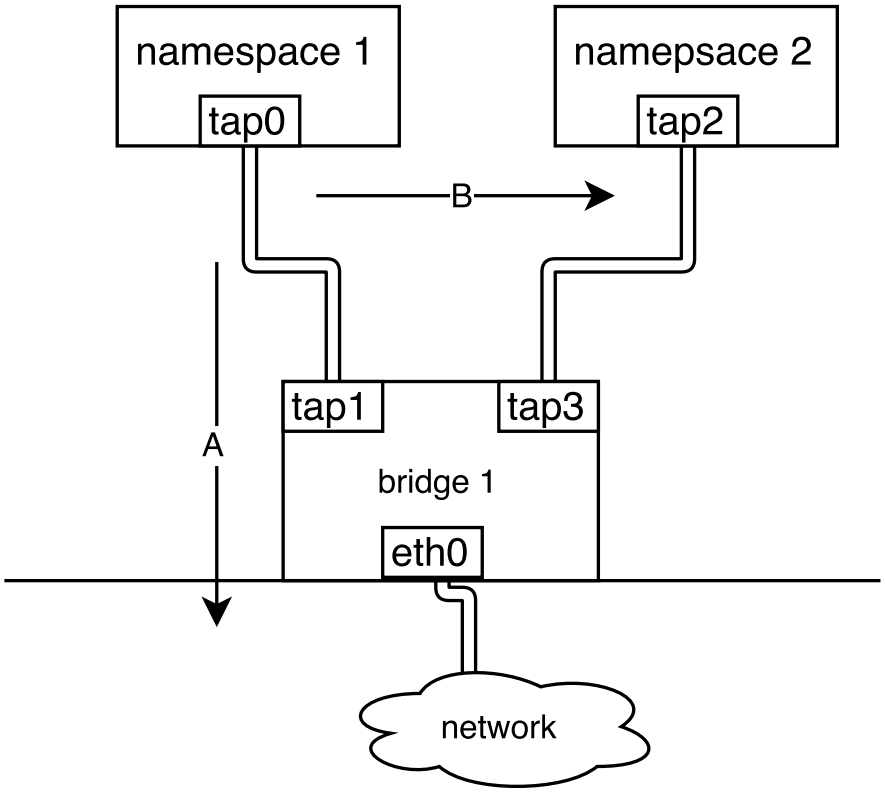
```
ip link add name [name] type bridge
ip link del name [name] type bridge
ip link set [interface] master [name]
ip link set [interface] nomaster
```

První dva příkazy vytváří respektive odstraňují bridge, další dva jej připojují respektive odpojují od interfacu, který je předaný parametrem.

Výsledné zapojení VPC do portu pomocí bridge je znázorněno na ilustrační situaci na 2.2.

Bridge a namespace je propojen pomocí veth páru. Veth pár má koncové porty označeny „tap“. Vždy jeden z páru je vložen do bridge, a druhý do namespacu. Pro připojení bridge do okolní sítě je do bridge přesunut port „eth0“.

Výsledná komunikace by měla probíhat z namespacu do okolní sítě, jak je znázorněno šipou A. Vlastnosti bridge však povolují komunikaci mezi všemi



Obrázek 2.2: Schéma zapojení bridge

dostupnými zařízeními. Takže je možná i komunikace z namespacu 1 do namespacu 2, jak znázornňuje šipka B.

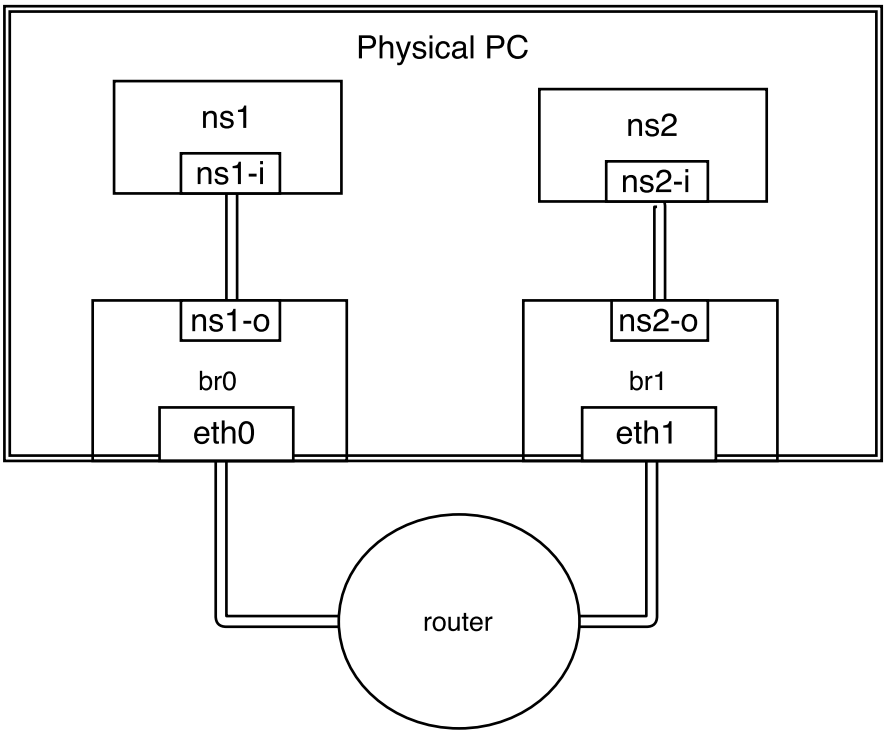
2.3.3.4 Nežádoucí komunikace

Tuto nežádoucí komunikaci z obrázku 2.2 lze zakázat použitím `iptables`, což je nástroj na filtraci packetů.

Nechtěná komunikace je taková, která jde z virtuálního PC do jiného virtuálního PC přímo přes softwarový bridge. Veškerá komunikace by měla probíhat přez port bridge, který je zapojen do fyzického interfacu.

Tento interace je známý, a proto například pro interace eth0 můžeme formulovat pravidla:

1. akceptuj packety se vstupním interfacem eth0
2. akceptuj packety se výstupním interfacem eth0
3. všechny ostatní packety zahoď



Obrázek 2.3: Ověření izolace namespaců

Pro specifikaci vstupního interfacu je parametr `-i, --in-interface`, a pro specifikaci výstupního je `-o, --out-interface`.

2.3.3.5 Ověření izolace

V této sekci bude vytvořena softwarová síť pomocí výše zmíněných nástrojů. Cílem je prakticky ověřit, zda spolu nástroje fungují, a zda splňují to, co říkají v dokumentaci. Zde se zapojení vytvoří manuálně. Po ověření se tyto nástroje použijí v implementaci platformy.

Model síťě, na které to bude ověřeno, je na obrázku 2.3. síť byla vytvořena na reálném PC se dvěma ethernetovými porty. Oba byly zapojeny do routeru.

Jako ukázka je uvodeno vytvoření a zapojení namepsacu 1 z obrázku.

```
ip netns add ns1
ip link add ns1-i type veth peer name ns1-o
ip link set ns1-i netns ns1
ip link add name br0 type bridge
ip link set br0 up
ip link set eth0 up
ip link set eth0 master br0
ip link set ns1-o up
```

```
ip link set ns1-o master br0
ip netns exec ns1 ifconfig ns1-i 10.0.0.28
```

Test spočíval v ověření dostupnosti síťového rozhraní namespacu 1 ze síťového rozhraní namespacu 2. Byly ověřeny 3 situace:

1. Oba ethernetové porty jsou připojeny. Test pingu z ns2 do ns1.
2. Jeden ethernetový port je odpojen. Test pingu z ns2 do ns1.
3. Oba ethernetové porty jsou připojeny. Test pingu z ns2 do internetu.

Síť se chovala, jak bylo požadováno. V prvním případě ping prošel, v druhém ne, a ve třetím ano.

2.4 Zvolené řešení

Při virtualizaci PC byly uvažovány technologie VirtualBox, Docker a Linuxové namepsacy. Jako nejvhodnější byly zvoleny síťové namespacy. Z uživatelského hlediska jsou jednoduché na použití, a z hlediska využívání zdrojů a rychlosti vyhovující. Z pohledu závislosti na softwaru toto řešení požaduje pouze nainstalovaná balíčky `bridge-utils` a `iproute2`.

V předchozí kapitole bylo řešení s namespacy rozvedeno a byly představeny a demonstrovány všechny síťové technologie, které bude implementace používat.

Kapitola 3

Navrh a implementace

Tato kapitola je věnována návrhu platformy a její implementaci. Návrh prošel mnoha iteracemi, během kterých byly provedeny změny. Důvody změn byly jak postupné aplikace technologií z předchozí kapitoly, tak i změny požadavků na API. Kapitola je rozdělena podle jednotlivých modulů.

3.1 Pcm Api

Pro univerzální přístup k různým implementacím VPCM bylo zvoleno rozhraní nazvané PcmApi3.1. Test jej nepoužívá přímo, ale používá jej modul TestManager. Každá implementace modulu VCPM musí toto rozhraní implementovat.

Účelem toho rozhraní je sjednotit přístup TestManageru k VPCM. Díky tomu je možné vytvořit různé verze VPCM, aniž by se TestManager musel měnit. Vyplývá z toho také požadavek, aby se přes API nepřenášely žádné objekty TestManageru ani implementace VPCM. PcmApi může pracovat pouze s primitivní typy, stringy a poli.

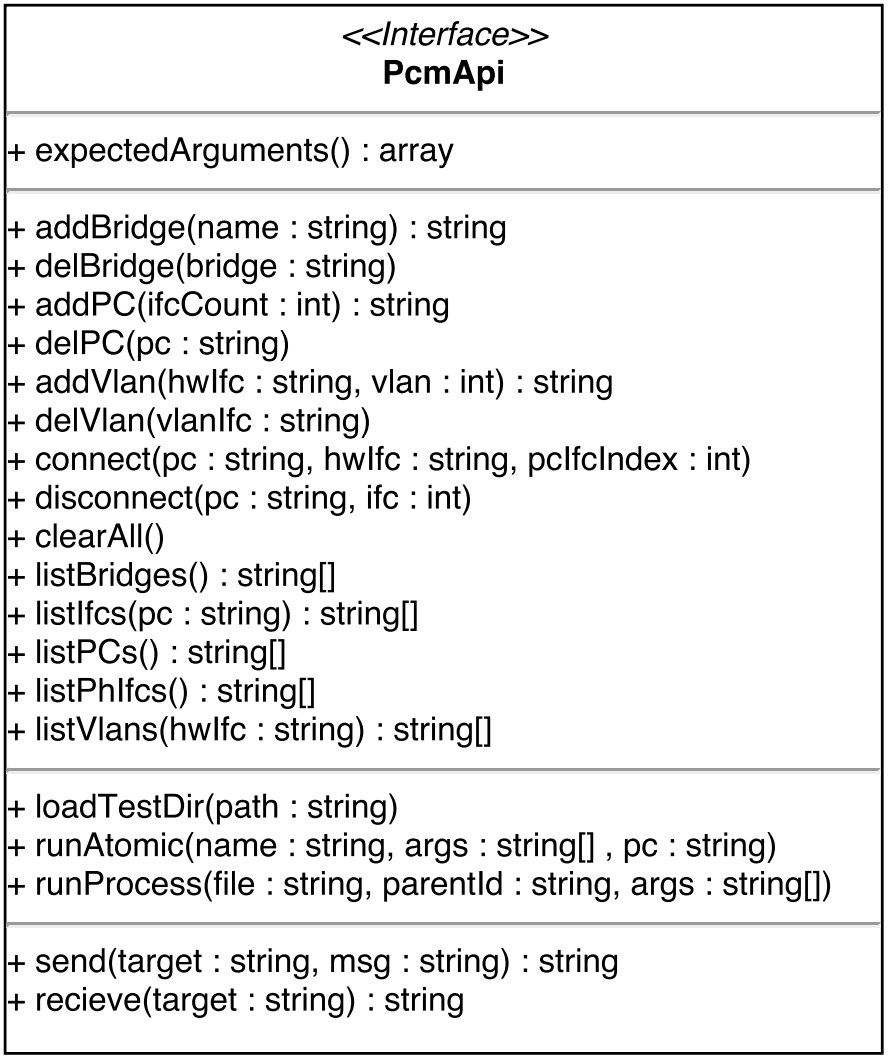
PcmApi bude zmíněno u komponent, které s ním pracují, a bude vyvětlen jejich vztah.

3.2 VPCM

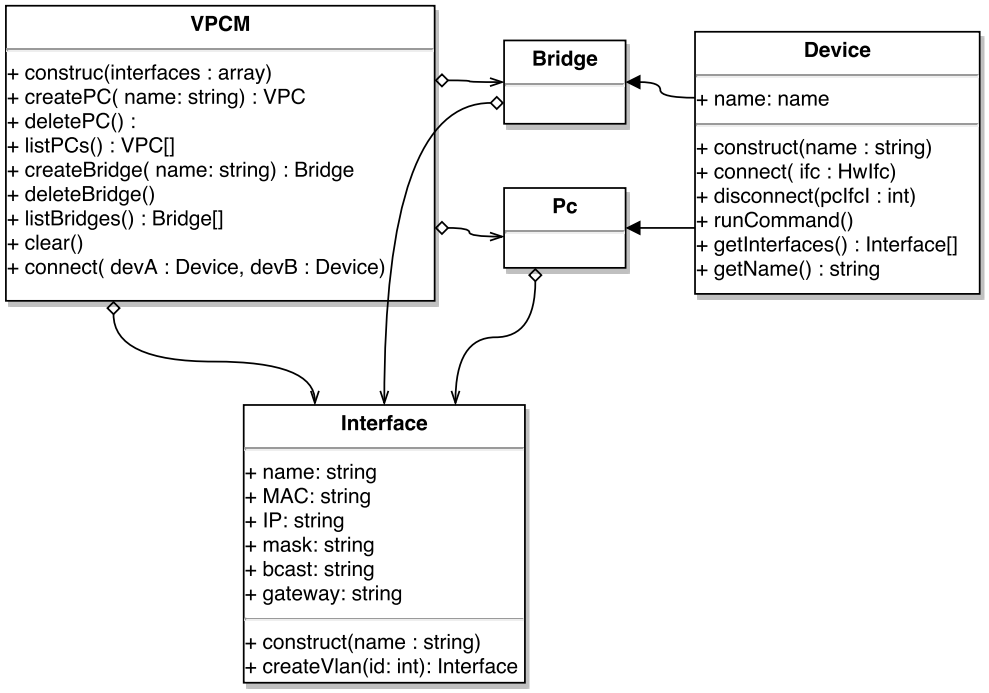
Tato sekce se zabývá implementací práce modulu VPCM, který přímo zadává instrukce operačnímu systému, a tím konfiguruje namespacy, připojení a další.

3.2.1 Prvotní Implementace VPCM

Diagram tříd 3.2 zobrazuje první verzi modulu VPCM. Sloužila jako nástroj pro ověření použitých technologií a základ pro další iterace při tvorbě platformy.



Obrázek 3.1: PcmApi class diagram



Obrázek 3.2: Class diagram první implementace VPCM

Přibyla zde navác jedna entita, se kterou původní návrh nepočítal. Je jí Bridge, který má za úkol propojovat VPC s síťovými interfacy hostitelského počítače. Důvody pro nutnost zavedení Bridge jsou vysvětleny v kapitole 2.3.3.3.

Práce s Bridgi a VPC je z pohledu jejich API podobná. Oba mají nějaké své interfacy, které lze přidávat a odebírat, oba jsou identifikovány jménem. Proto zde byla zavedena třída Device, která práci s jimi sjednocuje. Díky tomu je možné definovat metodu connect na PVCM tak, že jí lze propojovat Bridge a VPC libovolně mezi sebou. VPCM pak volá metodu connect, která je definovaná v předku Device, na obou objektech.

Existence entity Bridge dovoluje vytvořit i komplikovanou softwarově definovanou síť uvnitř hostitelského počítače. Podle zadání však po platformě není požadováno, aby umožňovala vytváření složitějších sítí. Je požadováno, aby z uživatelského hlediska co nejjednodužším způsobem umožnila vytváření takových zapojení, jaké je popsáno v 1.4.

VPCM má skrývat detaily vytváření sítě a zapojoví VPC, a poskytovat požadované rozhraní testu. VPCM musí bridge obsahovat, ale pro zapojení PCM do interfacu je nežádoucí, aby se s ním muselo pracovat explicitně.

3.2.2 Uchovávání dat modelu

První implementace dále předpokládá, že bude mít přístup k modelu sítě vytvořeném v paměti. Tento koncept je ale překážkou paralizace. VPCM má být dostupné z více procesů, které mezi sebou paměť nesdílí.

Řešením tohoto problému je nějak zamezenit paralelnímu přístupu nebo model umístit na do nějakého úložiště, které paralelní přístup umožňuje. Byla zde uvažována řešení:

- Vytvožení VPCM v samostatném proces běžícím na pozadí. Proces by poskytoval nějaký přístupový bod, například socket, na kterém by poslouchal jeho server. Volání metod na VPCM by bylo převedeno na požadavky na tento server, které by tyto operace prováděl na své instanci VPCM a vracel by výsledky. Server by byl jednovláknový.
- Pro uložení datového modelu použít databázi, ke které by mohlo být přistupováno paralelně.
- Ukládat data serializací do lokálních souborů. Při jejich čtení nebo změnách by se používaly dva zámky. Jeden pro čtecí operace, druhý pro zapisovací.

Při revizi těchto možností bylo zjištěno, že většina informací, které model uchovává, je dohledatelná v systému. Persistentním modelem pro VPCM je tedy systém samotný. Věci, které nemohou být zjištěny přimo ze systému, budou uloženy v souborech. Jsou použity soubory konfigurací a další pomocné soubory pro běh.

Pro ochranu těchto souborů při čtení a zápisu jsou použity zámky řešené pomocí souborových zámků `flock`.

3.2.2.1 Jmenné konvence

Pro zjednodušení zjišťování informací o síťovém nastavení byla zavedena jmenná konvence na vytvářené objekty. Vytvářeným prvkům jsou přidělována taková jména, aby je bylo možno později podle nich identifikovat. Používá se prefixů nastavených v konfigiračním souboru. Tento konfigurační soubor má každý VPCM vlastní. Lze tedy nastavit, aby každé VPCM používalo jiná jména, a navzájem se VPCM neovlivňovaly. V následujících odstavcích bude vysvětleno použití pojmenování pro některé entity.

Zde je výpis části konfiguračního souboru, kde jsou nastaveny prefixy.

```
"prefix_pc":"pc",
"prefix_br":"br_",
"prefix_vpcm":"A_",
```

Při vytvářžení se VPC se vytvoří se jménem složeným z prefixu VPCM, prefixu PC a identifikáoru.

VPC se vytváří s již předdefinovaným počtem interfaců. Pro zjednodušení jsou v nich již zapojeny virtuální ethernetové pary, kdy vnitřní část má jméno `[pcname]-[index]-i` a vnější `[pcname]-[index]-o`, kde index je pořadové číslo interfacu vrámci PC. Při dalším zapojování a odpojování těchto interfaců tedy víme jméno vnějšího rozhraní, které je třeba někam připojit či odpojit.

Pro zapojování je použito bridge, jak je popsáno v kapitole 2.3.3.3. Zapojení VPC do interfacu hostitele znamená vytvořit bridge před tímto interfacem, do kterého se budou zapojovat další VPC. Bridge není nutný v případech, kdy je zapojeno pouze jedno PC, ale ze síťového pohledu nijak nevadí.

Zapojení tedy probíhá tak, že pokud zapojujeme VPC do síťového interfacu, zapojujeme jej vlastně do bridge před tímto interfacem. Pro jeho identifikaci je tento bridge pojmenován řetězcem `[prefix_vpcm][prefix_br][ifc name]`. Příklad může být interface `eth0`, které se při ponechání defaultních hodnot prefixů zapojí do bridge `A_br_eth0`. Stejné řešení je použito i na VLANy. Výsledné jméno bridge s VLANy je například `A_br_eth0.10` pro vlan s id 10.

Prefixy by neměly být moc dlouhé. Zatímco délka jména namespacu a interfacu není omezená, maximální délka jména linuxového bridge je 15 znaků. Pokud je tato délka překročena, bridge nelze vytvořit. Dále některá jména síťových interfaců mohou být dlouhá, s ohledem na tvotbu bridgů je vhodné je před použitím přejmenovat.

3.2.2.2 Další nastavení

Další část konfiguračního souboru uchovává tyto údaje.

```
"hw_ifc":["enp0s3"],
"folder":"\/nettest"
```

Položka `hw_ifc` je zkratkou pro hardware interfaces. Je to výčet jmen interfaců, která může VPCM používat pro připojování VPC.

Poslední položkou konfigurace pro lokální implementaci je cesta k pomocnému adresáři, který je specifický pro VPCM. V první řadě slouží k ukládání testových soubotů. Dále k všem potřebným pomocným souborům pro běh. Používají ho procesy pro komunikaci a ukládání logů, a také VPCM pro ukládání informací generátoru jmen.

3.2.2.3 Generátor jmen

Generátor jmen má za úkol vytvářet jména pro bridge, VPC a procesy. Na ty jsou kladeny 2 požadavky: musí používat prefixy a musí být unikátní. Jméno je realizované jako prefix a číslé. Pro zajištění unikátnosti je použito souboru, kam ukládá generátor již přidělená čísla. Všechny jmenovaná zařízení lze vytvářet najednou z více procesů, a samotné vytváření je z pohledu vícevláknovosti kritická sekce. Proto jsou pomocné soubory generátoru při použití zamykány.

3.2.3 Implementace

V konečné implementaci je VPCM pouhá implementace rozhraní PcmApi. Objektový přístup z prvního návrhu se ukázal jako nepraktický. Jedním z důvodu je, že metody Api nepočítají s pamětí. Data v paměti jako v první implementaci nejsou, ale místo toho jsou metody PcmApi navrženy tak, aby vše potřebné dodaly v parametrech. VPCM bude používáno pouze přes api, a jinak se s ním pracovat nebude.

Výsledné VPCM se tedy skládá z třídy implementující metody PcmAPi výše popsaným způsobem a několika pomocných tříd. Jednotlivé metody Api volají už přímo příkazy v příkazové řádce. Používají pro to metodu `cmd()`, která ve svém těle volá php příkaz

```
exec($command . " 2>&1", $out, $rv);
```

který příkaz provede.

3.2.4 Remote implementace

Do budoucna je předpokládáno, že se do platformy přidá síťová vrstva. Tato vrststva umožní použití více VPCM na různých strojích. Pro to bude třeba vytvořit implementaci VpcApi, která bude fungovat přes síť. Tato síťová implementace PcmApi bude komunikovat se serverovou částí na jiném stroji. Serverová část bude mít za úkol pouze volat příkazy obdržené od klienta, na své lokální implementaci VPCM.

3.3 TestManager

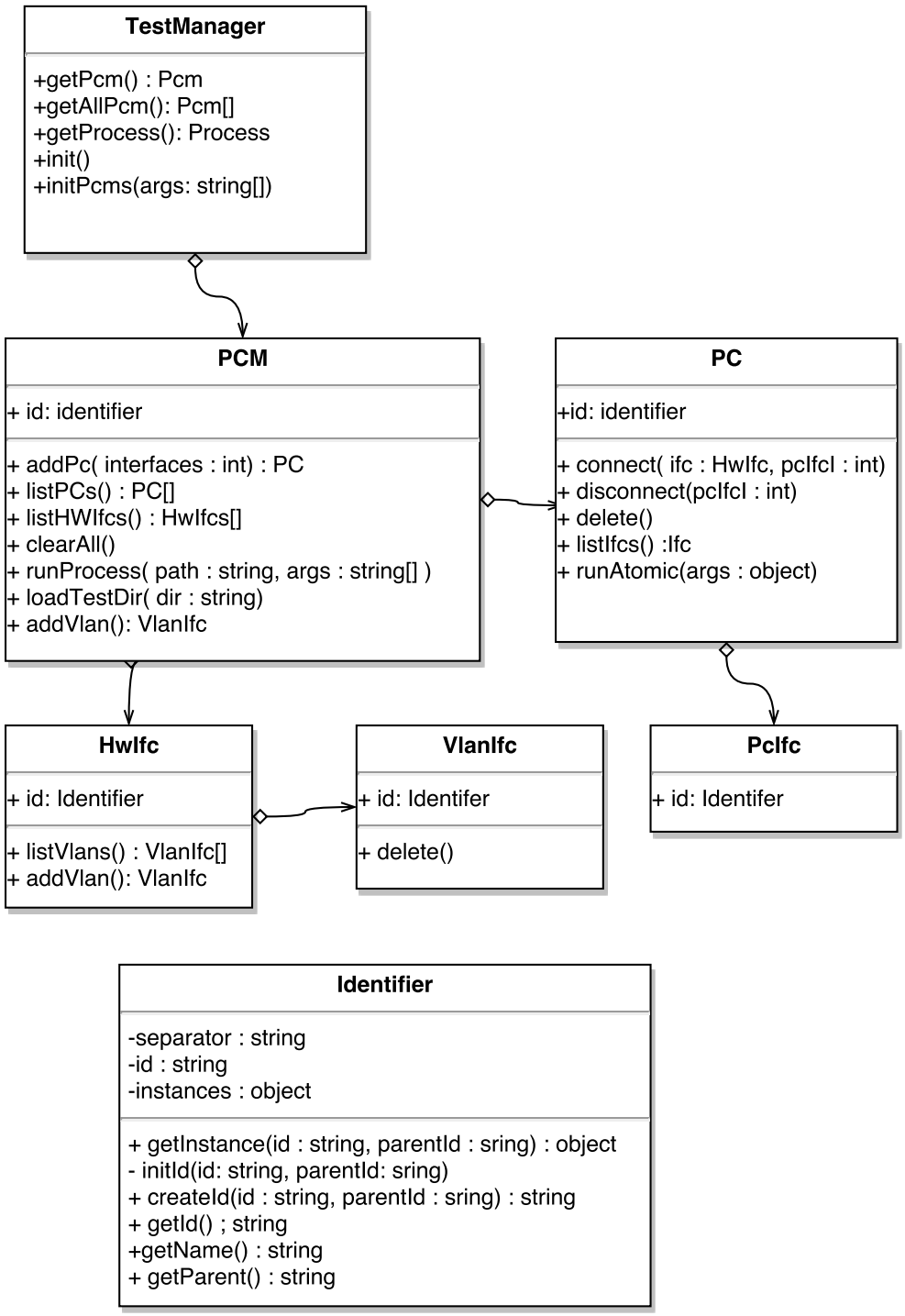
TestManager je komponenta, která bude prezentovat veškeré funkce VPCM tvůrci testovacích skriptů. Její rozhraní by mělo být jednoduché a logické. Zde je diagram výsledného API 3.3.

Objekty TestManagerureprezentují objekty jako PC a interface, které spravuje VPCM. S tímto VPCM se spojí prostřednictvím PcmApi. Jelikož Api je přístupné z více vláken, jsou veškeré persistentní informace ukládány na straně VPCM. Jediné informace na straně knihovny TestManager jsou identifikátory jednotlivách objektů. Objekty tedy uchovávájí pouze ID. Veškeré jejich metody se mapují na metody PcmApi.

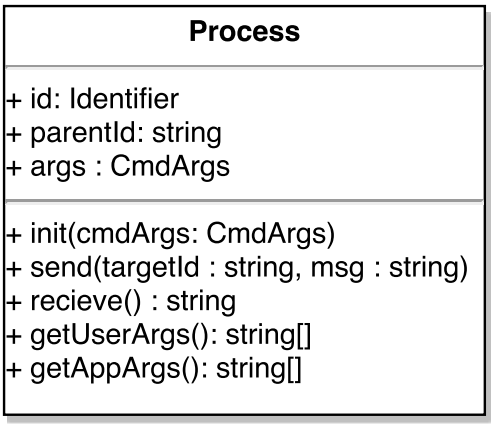
Vyjímku tvoří objekt TestManager. Ten má za úkol spravovat více instancí PCM. Uchovává informace potřebné k získání a vytvoření PCM. Pro jejich správu používa data z konfiguračního souboru.

3.3.0.1 Identifikátor

Třída Identifikátor realizuje ID pro veškeré objekty TestManageru. Je implementovná jako trait, který dané třídy použijí.



Obrázek 3.3: TestManager class diagram



Obrázek 3.4: Process API

ID je realizováno jako string složený z identifikátorů jednotlivých částí. Každý element modelu má své ID svého PCM. Pokud se nachází uvnitř PC, má i ID svého PC. ID je tedy string, který umožňuje jendoznačně rozlišit, ke kterému PCM případně PC objekt patří. Vychází to z modelu 1.5.2, kdy ID je složeno vždy z ID předchozích uzlů, přičemž CPM je kořen. Části ID jsou odděleny separátorem „:“. Například pro interface PC je ID složeno takto: `[pcm id]:[pc id]:[ifc name]`.

Další funkcí Identifikátoru je vytváření objektů. Je požadováno, aby TestManager nevracel více objektů reprezentujících stejný reálný objekt. Tento požadavek byl vyřešen použitím factory metody `init()` pro vytváření objektů. Zároveň má každá třída staticky instance všech jejích vytvořených objektů. Pokud přijde dotaz na nový, nejprve zkontroluje, zda již není vytvořen. Pokud ano, už jej nevytváří. Díky tomu není třeba porovnávat jen pomocí ID, ale pro zjištění, zda jsou objekty stejné, lze požít operátor `===`, který zjistí, zda jsou objekty na stejném místě v paměti.

3.4 Procesy

Třída Process v TestManageru poskytuje rozhraní pro práci s procesy. To umožňuje vzájemnou komunikaci procesů a jejich synchronizaci.

Instance procesu je dostupná vždy jedna, a to ta, co reprezentuje proces, ve kterém je skript spuštěn. Umožňuje posílat zprávy ostatním procesům, které jsou adresovány pomocí ID. Třída Process má rozhraní zobrazené na 3.4.

Proces je možné spustit dvěma způsoby:

- spuštěním testu z shellu přes loader,
- zavoláním metody `runProces()` na příslušném objektu VPC.

Každý proces potřebuje vlastní ID. Při vytváření je možno zadat ID jako commandline parametr. Pokud není zadáno, je vygenerováno další. K vytvoření ID je použito generátoru jmen, který je už použit např při vytváření VPC. Dalším uvažovaným řešením je použití PID jako identifikátoru v rámci jednoho PCM.

Z procesu lze vlastní PID získat například pomocí `echo \$BASHPID`. Při spuštění pomocí `runProces()` můžeme získat PID právě spuštěného procesu pomocí `echo $!`;

Proces dále uchovává argumenty zadané při spuštění. Ty jsou dodány buď přímo z shellu, nebo jako parametr při volání funkce `runProces()`. Parametry jsou rozděleny na aplikační, které používá platforma, a uživatelské, které jsou pro potřeby testu. Z procesu jsou dostupné metodami `getUserArgs()` `getAppArgs()`. Obě vrací asociativní pole, kde klíčem je jméno parametru. Z toho vyplývá omezení na zadávané parametry na formát `--[key] [value]`.

Metody `send()` a `recieve()` zprostředkovávají komunikaci a synchronizaci mezi procesy. Procesy mohou bežet na různých VPCM. Díky modulu TestManager a volání `send()` a `recieve()` přes PcmApi na správném VPCM je zaručeno, že v VPCM implementaci jsou oba procesy lokální, respektive ten vzdálený proces je realizován například vláknem serveru, které běží lokálně. `recieve()` je vždy voláno na vlastním VPCM. Naopak `send()` musí nejprve nalézt správné VPCM podle ID adresáta, a potom na jeho PCM reprezentaci zavolá `send()`.

3.4.1 Realizace komunikace

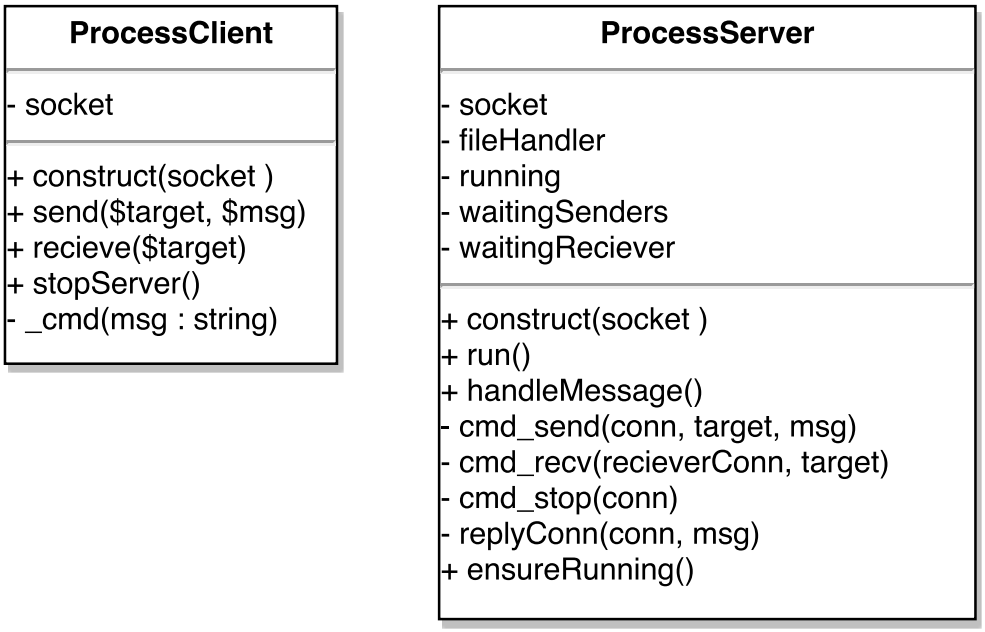
V předchozí části bylo řečeno, jak se volání převede z distribuovaného prostředí do jednoho VPCM. Zde bude rozvedeno, jak probíhá komunikace v rámci jednoho VPCM.

Pro komunikaci bylo použito řešení přes unix domain socket. Na každém VPCM se při inicializaci zapne ProcessServer, který zprostředkovává komunikaci. Na něj se připojují klienti ProcessClient s dotazy „send“ a „ricieve“. Rozhraní těchto dvou tříd je zobrazeno na 3.5.

Server se startuje pomoci metody `Server::ensureRunning()`. Ta zkontroluje, zda je server zapnutý. Pokud ne, zapne ho. Ukončuje se zprávou od klienta ClientProcess „stop“.

Pro odeslání a přijetí zprávy klient odesíla zprávy „send [target] [msg]“ a „recieve [target]“ Pokud zprávu nemůže server doručit hned, vnitřně si uloží zprávu, odesílatele, příjemce, a spojení mezi ním a klientem nechá otevřené. Když pak obdrží zprávu od druhé strany, odpoví oběma stranám. Tomu, kdo zavolal `recieve()` odešle data. A tomu, kdo poslal `send()`, odešle potrzení o odeslání zprávy.

Do budoucna se uvažovalo o rozšíření o další synchronizační způsob, a to o barierové synchronizaci. Ta by měla metodu `wait()`, která by jako parametr přijímala počet vláken, při kterých se bariera prolomí. Přidání by si vyžádalo



Obrázek 3.5: Process klient a server

zásah do logiky ProcessServeru. V tuto chvíli jsou dostačující dvě implementované metody.

3.5 Atomické operace

Atomická operace má umožnit jednoduše rozšiřovat platformu o nové operace na PC, jak je popsáno v kapitole 1.5.5. Operace vpodstatě obalují nějaký příkaz příkazové řádky. Příkazy mohou být parametrizované, a mají nějaký výstup, který operace zpracovává.

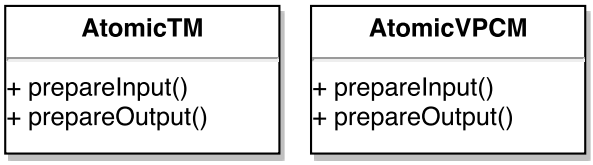
Obecně se atomická operace skládá z částí

- funkce vytvářející příkaz
- funkce zpracovávající výstup

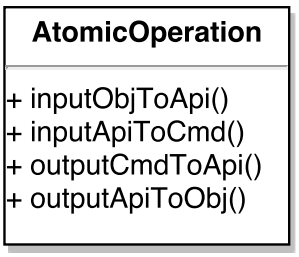
Z pohledu na komonenty je třeba brát v úvahu rozložení na moduly. Operace budou používané z testovacího skriptu a budou součástí modulu TestManager. Mohou používat jeho i objekty. Provedení operace je ale vázáno na modul VPCM a jeho lokální implementaci. Dále objekty TestManageru, které lze používat v testovacím skriptu, zde již použít nelze.

Kvůli respektování rozložení na moduly a jejich separaci by se atomická operace měla skládat ze dvou objektů, jak jsou znázorněny na 3.6.

Objekt AtomicTM je součásti TestManageru. Má za úkol přijmout parametry, zpracovat je, a poslat modulu VPCM přes Api.



Obrázek 3.6: Atomické operace



Obrázek 3.7: Atomické operace

Objekt AtomicVPCM je na úrovní lokálního VPCM. Přijme parametry a vytvoří příkaz. VPCM jej provede a vrátí výstup. Ten je dále zpracován objektem AtomicVPCM, a vrácen TestManageru. Tam jej zase může zpracovat objekt AtomicTM.

V konečné implementaci byly tyto tyto 2 objekty složeny do jednoho 3.7. Porušuje se tím pravidlo oddělení vrstev, kdy atomická operace pracuje jak na TestManageru, tak ve VPCM. Na druhou stranu se tím docílilo jednodušší rozšířitelnosti. Pro definici nové atomické operace je třeba přidat jednu třídu, takže veškeré změny jsou pouze na jednom místě.

3.5.1 Implementace

Nakonec je atomická operace realizována abstraktní třídou AtomicOperaton 3.7. Definuje 4 metody, které je možné v potomcích překrýt.

Při definování nové atomické operace je tuto třidu třeba podědit. Třída nové operace pak musí mít jméno `cmd_[name]`. Na objektech VPCM a VPC se pak volá pouze jménem.

Vlastní volání atomické operace je umožněno na PC pomocí magické PHP metody `__call`. Pokud se na PC zavolá metoda, která není definována, je zavolána metoda `__call()`, která se pokusí na PC provést atomickou operaci. Pro její dohledání je použito jméno volané metody s prefixem `cmd_`.

Například při zavolání metody `$pc->setIp("10.0.0.1");` se PC pokusí načíst třídu `cmd_setIp` implementující AtomicOperation.

Protože u atomických operací se očekává, že jich bude implementováno více, bude zde demonstrována implementace operace set ip.

```
class cmd_setIp extends AtomicOperation {
```

```
    public function inputObjToApi($ip, $mask, $ifc = 0) {
        if (is_numeric($mask)){
            $mask = NetUtils::maskToIp($mask);
        }
        $ifc = AtomicFunctions::getIfcName($ifc, $this->getPc());
        return array($ip, $mask, $ifc );
    }

    public function inputApiToCmd($ip, $mask, $ifc) {
        return "ifconfig␣$ifc␣$ip␣netmask␣$mask";
    }
}
```

V tomto případě metoda `inputObjToApi()` přijímá 3 parametry. IP adresu, masku sítě a síťový interface na PC. Ten může být zadán jako index nebo jménem, a maska může být zadána celou IP nebo jen číslem. Parametry mohou být dále transformovány. V ukázce tato metoda přistupuje i k objektu PC, a to zde `$this->getPc()`. Nakonec vrací pole s hodnotami, které budou použity jako vstupní parametry drufé funkce `inputApiToCmd()`, která už vrací konkrétní příkaz.

Výstupní operace `inputObjToApi()` a `inputObjToApi()` zde implementovány nejsou. Použije se jejich defaultní implementace z třídy AtomicOperation, která vrací výstup příkazu jako text.

Součástí platformy jsou implementované základní požadované operace jako statické a dynamické přidelení IP adresy, zjištení IP adresy a ping.

3.6 Loader

Loader slouží pro spuštení testovacího skriptu. Jeho úkoly jsou

- inicializace TestManageru
- načtení adresáře s testem
- nalezení testovacího souboru
- spuštění testovacího souboru

Loader dále kontroluje parametry pro načtení testu. Jsou to „testfile“ a „testdir“ . První určuje jméno testovacího souboru. Druhý má za úkol předat cestu k adresáři, kde je test.

3.6.1 Adresář s testem

Předpokládá se, že test nemusí obsahovat pouze jeden soubor. Proto se načítá celý adresář. Soubory mení potřeba kopírovat ručně.

Hlavním důvodem zavedení kopírování adresáře s testem je budoucí možnost rozšíření o další VPCM. Ty mohou běžet v jiném systému a nesdílí tak

ani stejnou paměť. Takže před použitím testových souborů na jiných VPCM tam musí být nejprve rozkopírovány.

Do budoucna je plánována tato distribuce testů v komponentě Loader, kdy při zadání testovacího adresáře zároveň rozešle tyto adresáře všem dostupným VPCM, které si vytvoří lokální kopii.

Vlastní kopírování bude probíhat přes metodu `loadTestDir($data)` PcmApi. Protože PcmAPi umí přenášet pouze jednoduché datové typy, bude muset být adresář nejprve zabalen do jednoho souboru, a ten pak zakódován do textu. Pro zabalení jsou uvažovány `zip` nebo `phar`, pro zakódování pak `base64`

Pokud by se nepoužila tato metoda, musely by testy být distribuovány například pomocí nástrojů jako `scp` nebo `rsync`. Takové řešení by však obcházelo VPCM a vytvářelo další přístupový bod.

3.6.2 Ukončení procesů

Do budoucna by měl řešit i správné dokončení všech zapnutých procesů. Nemělo by se stát, aby testovací skript za sebou nechal spuštěné další procesy, a sám se jako proces ukončil.

V případě, že by hlavní proces končil, a přitom vytvořené procesy stále existovaly, by měly být všechny tyto vytvořené procesy zabity. Tato funkce by musela být podporována modulem VPCM, protože procesy mohou být vytvářeny v jiných systémech, a ne pouze lokálně. Dále by to znamenalo mít přehled, jaké procesy který proces vytvořil, přičemž procesy lze vytvářet rekurzivně.

3.7 Souhrná struktura projektu

V této kapitole byly popsány jednotlivé moduly. V poslední sekci bude shrnuta struktura projektu z pohledu adresáře.

```
Testnet
├── config.json
└── src
    ├── api
    ├── atomic
    ├── local
    ├── misc
    ├── process
    ├── tm
    ├── test
    ├── autoload.php
    └── loadTestnet.php
```

V adresáři aplikace je konfigurační soubor `config.json`, který byl zmíněn například v sekci 3.2.

Dále obsahuje adresář `src` se zdrojovými kódy aplikace. Ty jsou členěny podle rozložení na komponenty. `api` obsahuje interface PcmApi, `atomic` obsahuje abstraktní třídu AtomicOperations a pak ukázkové definice několika operací, `local` obsahuje třídy lokální implementace VPCM, `misc` obsahuje pomocné metody nezávisle na vrstvě, `process` obsahuje třídy Process, ProcessClient a ProcessServer, `tm` obsahuje všechny třídy modulu TestManager a `test` obsahuje několik testů na demonstraci funkce platformy.

Obsahuje také dva soubory. `autoloader.php` a `loadTestnet.php`, který slouží ke startu aplikace.

Kapitola 4

Ověření prototypu

V této kapitole bude výsledněhý prototyp představen z hlediska použití. Pro její demonstraci budou ve druhé části této kapitoly provedeny dvě konfigurace sítě a na každé jednoduchý test.

4.1 Nastavení prostředí

Výsledná aplikace se skládá z několika balíčků tříd a konfiguračního souboru. V této části bude ukázáno, jak se aplikace spouští, budou uvedeny možné parametry a jejich význam, dále bude konfigurační soubor, a v poslední části nástin užití objektů v testu.

4.1.1 Spuštění a parametry

Vstupní bránou je skript `loadTestnet.php`. Používá se takto

```
php loadTestnet.php [app parameters] -- [user parameters]
```

Zde je výčet aplikačních parametrů, které jsou v tuto chvíli implementovány, a jejich význam. Dále bude jejich význam dovysvětlen.

- `testfile` – jméno testovacího skriptu
- `testdir` – adresář s testem
- `processId` – ID vlastního procesu
- `parentId` – ID rodičovského procesu

Parametr `testfile` specifikuje jméno souboru, který se má spustit jako test. Pokud není parametr zadán, použije se defaultní hodnota „test.php“.

Dalším parametrem je `testdir`. Pomocí něj lze zadat cestu k lokálnímu adresáři, který obsahuje testové soubory používané spouštěným testem. Pokud

není parametr zadán, použije se adresář specifikovaný v k-onfiguračním souboru pod názvem „folder“ a v něm podadresář „test“. Výsledná cesta adresáře je „[rootdit]/test/“.

Další parametry používané aplikací jsou `processId` a `parentId`, které se týkají procesu. Jsou používány při vytvoření procesu jiným procesem, a to metodou `runProcess()`. Při manuálním spuštění by se používat neměly.

Uživatelské parametry lze zadávat za ty aplikační. Je nutné je oddělit pomocí dvojpomlčky „ – “ a musí obsahovat vždy dvojici jméno parametru a jeho hodnotu.

Výsledné volání pak může vypadat takto.

```
php loadTestnet.php --testdir "path" --testfile "test1.php" -- --
    scriptparam "value"
```

4.1.2 Vytváření skriptu

Test je obyčejný php skript. Po zapnutí loaderem má k dispozici statické třídy TestManager a Process. Pomocí TestManageru může získat objekt PCM, který spravuje své virtuální PC. Api těchto objektů je znázorněno v 3.3 a jejich použití je ukázáno v testech níže.

Pro jakékoli operace, které mění konfiguraci systému, je třeba získat objekt PCM, který daný systém spravuje. To lze získat metodami

```
$pcm = TestManager::getMyVPC();
$pcm = TestManager::getVPC("id");
```

která vrací PCM shodné s PCM procesu, nebo jej získat pomocí ID.

S tímto objektem lze nyní vytvářet další procesy, vytvářet a mazat PC, získat seznam ethernetových portů, a tyto porty a PC propojovat.

Dále je zde objekt Process. Pomocí něho lze posílat zprávy ostatním procesů. Ty jsou adresovány ID. Nejčastější scénář použití je vytvoření procesu a poslání mu zprávy, nebo přijetí výsledku.

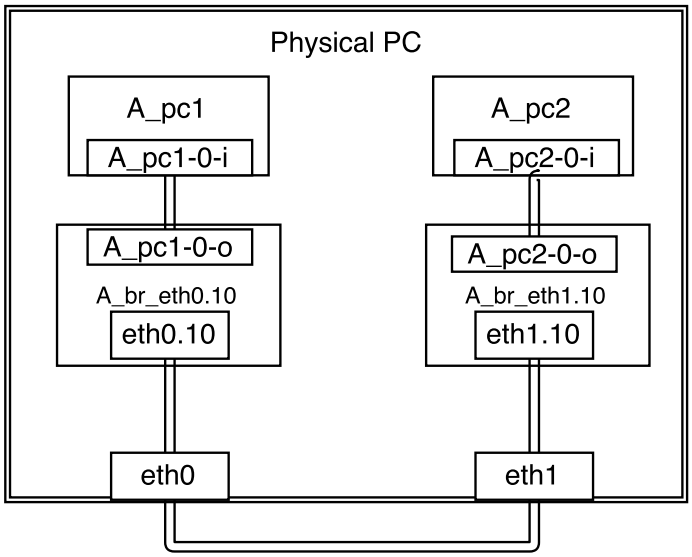
```
$childId = $pcm->runProcess("childScript.php");
$rv = Process::send($childId, $message);
```

4.1.3 Konfigurace

Platforma používá konfigurační soubor umístěný vedle adresáře se zdrojovými kódy. Pokud neexistuje, při prvním spuštění se vytvoří a vyplní defaultními hodotami.

Konfigurační soubor obsahuje pole, kde každý prvek je konfigurací pro jedno připojené VPCM. Pro vzdálené VPCM obsahuje informace k připojení. Pro lokální obsahuje nějaké nastavení, které potřebují k práci.

Při prvním spuštění bude nejspíše třeba změnit položku „hw_ifcs“, která nastavuje, jaké síťové interfacy na počítači mají být dostupné lokálnímu VPCM.



Obrázek 4.1: Konfigurace testu 1

Dále VPCM potřebuje nastavit nějaký pomocný adresář, se kterým může pracovat. Ukládají se do něj vlastní testy, logy, a další soubory, se kterými pracuje lokální implementace. V konfiguračním souboru je nastavován pod jménem „folder“ a jeho defaultní hodnota je „/testnet/“.

4.2 Testy

Součástí ověření je i provedení dvou testů. Uvedu zde síť, jaká je pro ně vytvořena. Dále zdrojový kód testovacího skriptu. A nějaký výstup. V tuto chvíli není definovano, jaké výstupy by skripty měly dávat. Definice výstupů je odpovědností samotných testovacích skriptů.

4.2.1 Test VLAN

Účelem toho testu je vyzkoušet správnou funkci izolace pomocí vlanů.

Je zde vytvořena síť znázorněná na obrázku 4.1. Test bude realizován na PC se dvěma fyzickými ethernetovými porty. Oběma těmito porty bude připojen do switche. Na obou portech se vytvoří VLANy a do každého z nich se zapojí přez bridge jedno VPC.

Vlstní kód testu, který síť vytvořil, je následující.

```
$pcm = TestManager::getMyPCM();
$pc1 = $pcm->addPC();
$pc2 = $pcm->addPC();

$listHWIfcs = $pcm->listHWIfcs();
```

```
$eth1 = $listHWIfcs[0];
$vlan1 = $eth1->addVlan(10);
$pc1->connect($vlan1);
$pc1->setIp("10.0.0.1", 24);

$eth2 = $listHWIfcs[1];
$vlan2 = $eth2->addVlan(10);
$pc2->connect($vlan2);
$pc2->setIp("10.0.0.2", 24);

$ip1 = $pc1->getIp();
print_r($pc2->ping($ip1));
```

Nejprve je získána instance lokálního VPC a na něm jsou vytvořeny dva PC. Dále jsou na dvou harwarových interfacech vytvořeny VLANy s id 10. Tyto vlany jsou zapojeny do PC1 a PC2. Konfigurace je dokončena přidělením IP oběma virtuálním počítačům.

Pro otestování se provede příkaz `ping` z jednoho PC do druhého. V této situaci ping prošel.

Jako protipříklad se provedlo to samé s rozdílným ID jednoho VLANu. Po této změne již ping neprocházel. Test tedy potvrdil funkčnost izolace.

4.2.2 Test přesměrování

V tomto testu bude částečně nastíněno budoucí použití platformy, kdy se předpokládá testování captive portálu. Vlastní test bude spočívat v dotazu na webovou stránku.

Captive portál by měl fungovat tak, že pokud je uživatel přihlášen, může přistupovat do internetu. Pokud není, dotaz na webovou stránku vrací přesměrování na stránku captive portálu.

Tento test domonstruje možnou metodu kontroly, zda došlo k přeměrování.

Pro test bude použito nástroje `curl` a vlastní dotaz na stránku realizován příkazem `curl -i [site]`.

Test bude proveden na takovéto 4.2 síti.

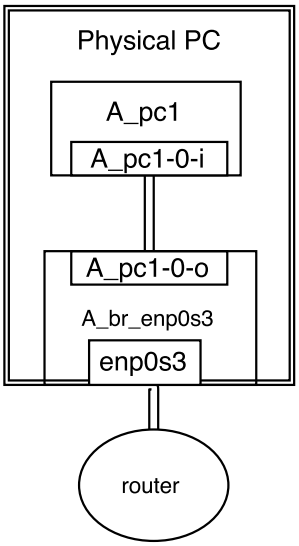
Pro vlastní vytvoření sítě a provedení dotazů je použito skriptu

```
$pcm = TestManager::getMyPCM();
$pc = $pcm->addPC();

$listHWIfcs = $pcm->listHWIfcs();
$eth = $listHWIfcs[0];

$pc->connect($eth);
$pc->dhcp();

print_r( $pc->cmd("curl␣-i␣https://www.google.com") );
```



Obrázek 4.2: Konfigurace testu 2

```
print_r( $pc->cmd("curl␣-i␣https://www.google.cz") );
```

Spuštění skriptu vede k odeslání následujících příkazů do systému. Jsou vybrány jen ty podstatné příkazy.

```
ip netns add A_pc1
ip link add A_pc1-0-i type veth peer name A_pc1-0-o
ip link set A_pc1-0-i netns A_pc1
ip link set A_pc1-0-o up
ip link set A_pc1-0-o master A_br_enp0s3
ip netns exec A_pc1 dhclient
```

První volání s dotazem

```
ip netns exec A_pc1 curl -i https://www.google.com
```

vrací

```
HTTP/1.1 302 Found
Location: https://www.google.cz/...
...
<TITLE>302 Moved</TITLE></HEAD><BODY>
<H1>302 Moved</H1>
The document has moved
<A HREF="https://www.google.cz/...">here</A>.
...
```

http status code 302 a přesměrování na jinou stránku.

Jako druhý dotaz jsme použili

```
ip netns exec A_pc1 curl -i https://www.google.cz
```

Odpověď na tento příkaz byla

```
HTTP/1.1 200 OK
...
```

Požadovaná stránka byla stažena a k žádnému přesměrování nedošlo.

Tento test slouží pouze jako příklad budoucího použití s výpisy spíše kontrolního charakteru. Co budou takovéto metody vracet v budoucnu není zatím jasné. Mohou například vracet jen http status code nebo `boolean` infromaci o úspěchu.

Závěr

V rámci této práce byl vytvořen prototyp platformy na testování hotelových sítí. Platforma byla rozložena na moduly. Mezi nimi byly popsány závislosti.

V následujích bodech připomenu cíle práce a zhodnotím jejich splnění.

1. „Analyzujte požadavky na platformu pro testování hotelových sítí. Zpracujte konceptuální návrh této platformy.“

   Požadavky na platformu byly definovány a rozvíjeny na konzultacích se zadavatelem. Jejich souhrn spolu s analýzou komponent a jejich významu v celku je obsahem v kapitole 1.

2. „Proveďte diskusi technických prostředků pro realizaci této platformy. Pro jedno možné rešení rozpracujte návrh z bodu 1.“

   Technickými prostředky umožňujícími zamýšlenou funkcionalitu se zabývá kapitola 2. Obsahuje informace o uvažovaných technologiích a možnosti použití pro platformu. Z virtualizačních technologií je zvolena nejvhodnější, a pro ni jsou detailně popsaná technická řešení síťové konfigurace, které jsou dále použity v implementaci.

3. „Navrhněte a implementujte prototyp dvou klícových modulů zamýšlené testovací platformy - VPC (virtual PC) a VPMC (virtual PC manager) a jejich komunikacní rozhraní (API).“

   V rámci analýzy došlo k částečné změně pohledu na moduly aplikace, a jejich význam. Aplikace nyní obsahuje VPCM modul se stejným významem, jako měly VPCM v zadání. Obsahuje 2 API, které se dále budou používat. Prvním je API modulu TestManager používané v testech, a druhé je API pro spolupráci modulu TestManager a VPCM.

   Kapitola 3 popisuje implementaci platformy. Zdrojové soubory výsledné implementace jsou na přiloženém CD.

4. „Na vytvořeném prototypu otestujte a zdokumentujte několik testovacích scénářů. Scénáře k otestování prototypu konzultujte s vedoucím práce.“

   Byly vytvořeny 2 testovací scénáře, které demonstrují použití platformy a její funkčnost. Tyto testy jsou popsány v kapitole 4.

Domnívám se, že jsem zadání práce splnil. Výsledkem není hotová aplikace ale prototyp, který je připraven na další testování. Z něj vyplynou požadavky na další věci, které mají být do platformy implementovány.

Uvažované rozšiřování platformy bude spočívat v implementaci práce se vzdálenými stroji a přidáváním dalších operací k virtuálním počítačům.

Příloha **A**

Seznam použitých zkratek

API Application programming interface

CPU Central processing unit

DHCP Dynamic host configuration protocol

DNS Domain name system

IP Internet protocol

IPC Inter process communication

JSON Javascript object notation

MAC Media access control

NAT Network address translation

PID Process identifier

POSIX Portable operating system interface

VLAN Virtuál local area network

VM Virtual machine

VPCM Virtual PC manager

Příloha **B**

Obsah přiloženého CD

- `src`
  - `impl` ..... zdrojové kódy implementace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `DP_Pokorny_Tomas_2016.pdf` ..... text práce ve formátu PDF