ŽILINSKÁ UNIVERZITA V ŽILINE

AUTOREFERÁT
DIZERTAČNEJ PRÁCE

Žilina, apríl 2018

Ing. Marek Moravčík

Žilinská univerzita v Žiline
Fakulta riadenia a informatiky

Ing. Marek Moravčík

Autoreferát dizertačnej práce

Migrácia služieb pre Cloud Computing

na získanie akademického titulu **„philosophiae doctor“ (v skratke PhD.)**
v študijnom programe doktorandského štúdia
aplikovaná informatika

v študijnom odbore:
9.2.9 aplikovaná informatika

Žilina, apríl 2018

Dizertačná práca bola vypracovaná v dennej forme doktorandského štúdia **na Katedre informačných sietí, Fakulte riadenia a informatiky Žilinskej univerzity v Žiline**

Predkladateľ: **Ing. Marek Moravčík**
Katedra informačných sietí
Fakulta riadenia a informatiky
Žilinská univerzita v Žiline

Školiteľ: **doc. Ing. Pavel Segeč, PhD.**
Katedra informačných sietí
Fakulta riadenia a informatiky
Žilinská univerzita v Žiline

Oponenti:

Autoreferát bol rozoslaný dňa: .....

Obhajoba dizertačnej práce sa koná dňa ..... o ..... h. pred komisiou pre obhajobu dizertačnej práce schválenou odborovou komisiou v študijnom odbore **9.2.9 aplikovaná informatika**, **v študijnom programe aplikovaná informatika**, vymenovanou dekanom Fakulty riadenia a informatiky Žilinskej univerzity v Žiline dňa .....

prof. Ing. Karol Matiaško, PhD.
predseda odborovej komisie
študijného programu **aplikovaná informatika**
v študijnom odbore **9.2.9 aplikovaná informatika**

Fakulta riadenia a informatiky
Žilinská univerzita
Univerzitná 8215/1
010 26 Žilina

Obsah

Úvod

V dnešnom svete informačných technológií (IT) sa môžeme vo viacerých ohľadoch stretnúť s problematikou virtualizácie. Či už ide o virtualizáciu operačných systémov (OS), sieťových prvkov, vývojových platforiem či aplikácií. Virtualizácia takéhoto typu je výhodná hlavne pre poskytovateľov služieb, pretože na jednom fyzickom zariadení môžu poskytovať služby niekoľkým používateľom bez toho, aby tieto služby navzájom kolidovali, resp. je vďaka virtualizácii možné optimálnejšie riešiť využívanie kapacity fyzických zdrojov so škálovaním a flexibilitou v ich prideľovaní. Z pohľadu koncového používateľa nemá virtualizácia na neho takmer žiadny vplyv, pretože virtuálna služba ktorú využíva má identické správanie, ako služba, ktorá by bola poskytovaná priamo na fyzickom zariadení. V mnohých ohľadoch si používateľ ani nevšimne, že využíva službu, ktorá je virtualizovaná.

Vývoj IT virtualizácie môžeme datovať od 50-tych rokov minulého storočia, odkedy prešla niekoľkými vývojovými fázami. Prvé náznaky virtualizácie sa objavili v roku 1957, kedy spoločnosť IBM navrhla systém časového zdieľania sálových počítačov. Ďalším významným krokom je koncept virtuálneho stoja (*virtual machine*), ktorý predstavila opäť spoločnosť IBM v roku 1972. Virtualizácia sa postupne vyvíjala, až dospela do dnešného stavu, ktorý nazývame Cloud Computing (CC) [1].

S pojmom „cloud“ sa stretli mnohí používatelia IT, mnohí dokonca služby CC aj nevedomky využili či využívajú. Pravdepodobne najznámejšími predstaviteľmi CC sú služby ponúkajúce online úložisko. Patria sem služby ako Dropbox, Google Drive, či Microsoft OneDrive. Ďalším známym predstaviteľom sú kancelárske prostredia s balíkmi používateľského online dostupného softvéru – Google Docs, či Office 365 od spoločnosti Microsoft.

CC prostredia sú využívané v čoraz väčšej miere, v neposlednom rade preto, že významným spôsobom šetria prostriedky, či otvárajú nové možnosti v poskytovaní IT služieb. S rozširovaním povedomia o CC sú si používatelia ako aj poskytovatelia týchto výhod CC vedomí. CC tak na jednej strane šetrí prostriedky poskytovateľom CC prostredí, pretože optimalizovaným využívaním fyzických zariadení dokážu pokryť požiadavky ďaleko väčšieho počtu používateľov ako tomu bolo pri fyzickom dedikovaní zariadení. Na druhej strane šetria prostriedky používateľom, ktorí nie sú nútení kupovať, prevádzkovať a spravovať mnohokrát finančne náročný hardvér alebo softvér. Poskytovatelia zväčša používajú tzv. „*pay-as-you-go*“ model, kde im používateľ zaplatí len za tie výpočtové zdroje, ktoré aj reálne spotreboval. Takéto riešenie je obzvlášť výhodné pre väčšie organizácie. Tie tak nemusia spravovať vlastné dátové centrum, a môžu si dynamicky zvyšovať a znižovať prenajatý výpočtový výkon podľa potreby, čo môže pri správnom manažmente v konečnom dôsledku ušetriť nemalé finančné prostriedky.

1 Súčasný stav riešenej problematiky

1.1 Špecifikácia pojmu CC

ITU-T v odporúčaní Y.3500 [6] definuje CC ako paradigmu, ktorá cez sieť povoľuje prístup k množine zdieľaných fyzických alebo virtuálnych zdrojov. Tieto zdroje sú podľa potreby škálovateľné a používatelia si ich sami dokážu prispôsobovať. Táto paradigma je zložená z kľúčových charakteristík, používateľských rolí, modelov nasadenia a prierezových CC aspektov.

V NIST SP-500-291 [7] je CC definovaný ako „Cloud Computing je model umožňujúci všadeprítomné, praktické, cez sieť prístupné a na požiadanie dostupné výpočtové zdroje (napríklad siete, servery, úložisko, aplikácie a služby) ktoré môžu byť rýchlo vytvorené s minimálnym úsilím a bez interakcie s poskytovateľom týchto zdrojov. Tento model je zložený z piatich základných charakteristík, troch modelov služieb a štyroch modeloch nasadenia.“

1.2 Infraštruktúra ako služba (Infrastructure as a Service)

Organizácia NIST vo svojom odporúčaní SP 500-291 definuje IaaS ako možnosť pre používateľa vytvoriť si základné výpočtové prostriedky (výpočtový výkon, úložisko, sieťové prepojenie, ...). Používateľ nespravuje infraštruktúru na ktorej sú spustené výpočtové prostriedky, no nad týmito prostriedkami má plnú kotnrolu. ITU-T v odporúčaní Y.3500 definuje IaaS ako kategóriu služby, kde používateľ má od poskytovateľa k dispozícii infraštruktúru.

Služba Infrastructure as a Service (IaaS) je určená pre skúsenejších používateľov, pretože si vyžaduje vedomosti z administrácie operačných systémov. Pri tomto type služby si používateľ kompletne spravuje celú IT infraštruktúru sám, počnúc servermi s ich OS, cez databázy až po sieťové prvky či sieťové prepojenia komponentov. Odporúčanie NIST definuje IaaS ako možnosť pre používateľa vytvoriť a prevádzkovať vlastnú infraštruktúru, na ktorej môže prevádzkovať vlastné aplikácie. Používateľ má k dispozícii základné výpočtové prostriedky (výpočtový výkon, úložisko a sieť), nad ktorými má zároveň administrátorskú kontrolu. Odporúčanie Y.3500 od ITU-T je v definícii strohejšie, hovorí len o tom, že používateľ má k dispozícii od poskytovateľa infraštruktúru.

SaaS, PaaS a IaaS sú tri základné rozdelenia modelu CC služby. V súčasnosti však vznikajú nové a nové modely CC služieb, ktoré poskytovatelia ponúkajú používateľom. Môže to byť napríklad preklad doménových mien - DNSaaS (*Domain Name System as a Service*), firewall - FWaaS (*Firewall as a Service*) či rozkladanie záťaže - LBaaS (*Load Balancer as a Service*). Vo všeobecnosti môžeme všetky tieto služby zhrnúť jednotným označením „*Čokoľvek ako služba*“ – **XaaS**, alebo **EaaS** (*Everything as a Service*).

1.3 Portabilita CC

Pojem ***portabilita*** všeobecne znamená prenositeľnosť produktov alebo komponentov. ITU-T Y.3500 v prierezových aspektoch CC špecifikuje portabilitu ako „Ability of cloud service customers to move their data or their applications between multiple cloud service providers at low cost and with minimal disruption. The amount of cost and disruption that is acceptable may vary based upon the type of cloud service that is being used. “. Ďalej ju rozširuje takto „Portability is significant in cloud computing since prospective cloud service customers are interested in avoiding lock-in when they choose to use cloud services. Cloud service customers need to know that they can move cloud service customer data or their applications between multiple cloud service providers at low cost and with minimal disruption. The amount of cost and disruption that is acceptable can vary based upon the type of cloud service that is being used. For example, if a cloud service customer organization is considering moving from one IaaS cloud service provider to another, the cloud service customer should be able to take its data and the virtual machine (VM) image and get it up and running on an equivalent IaaS service in a relatively straightforward manner. In an SaaS environment, when a cloud service customer organization wants to move an SaaS application

to a different cloud service provider (i.e., switch SaaS service providers), the cloud service customer needs to be able to take their data with them, but the rest of the switching cost will include exporting, mapping and importing the data into the new cloud service provider's SaaS application, and that cost is a function of how well the data models and formats of the two SaaS cloud service providers line up. Ideally, SaaS cloud service providers should adopt standard data interchange format(s) relevant to their application domain. Changing between SaaS applications can also involve the cloud service customer adapting to a new service interface (which relates to the interoperability of the service). However, since different cloud capabilities types can have different requirements related to portability, it is more useful to focus on specific types of portability such as cloud data portability and cloud application portability. Cloud service customer data is a class of data objects under the control of the cloud service customer. Cloud data portability allows the cloud service customers the ability to copy cloud service customer data into or out of a cloud service through network access or by physical transfer of storage devices. Cloud application portability allows the migration of items such as a fully-stopped virtual machine instance or a machine image (IaaS service) from one cloud service provider to another cloud service provider, or the migration of application components (PaaS service) from one cloud service provider to another. In both cases, there is a related aspect of the support of portability of metadata relating to the application components, providing information about the relationships of program components and about the required infrastructure for the program components (e.g., load balancing configuration, firewall settings). “

Ak je interoperabilita definovaná ako komunikácia medzi rôznymi systémami, portabilita je schopnosť používať systémy alebo ich komponenty na rôznom hardvéri alebo softvérových platformách [26]. Portabilita môže byť taktiež možnosť prenosu jednotlivej entity, či celého prostredia bez nutnosti ich modifikácie [33]. O portabilite môžeme uvažovať pri rôznych aplikáciách. Či už je to portabilita softvéru, portabilita elementov v sieti, portabilita telefónneho čísla medzi operátormi, či portabilita nariadení medzi rôznymi inštitúciami.

Používateľ určitej služby tak môže využívať rovnaké funkcie bez ohľadu na to, v akom systéme danú službu využíva. Takýto prístup je pre používateľov veľmi výhodný najmä pri zmene poskytovateľa služby, prípadne pri zmene koncepcie fungovania služby. V CC napríklad pri využití virtualizácie, či priamo CC platformy.

V [7] je portabilita v CC členená na dátovú a aplikačnú. V [34] a [35] portabilitu CC rozčleňujú na dátovú, aplikačnú a platformovú. Popis rôznych typov portability spracovaný ďalej je podľa [27].

Dátová portabilita umožňuje preniesť dáta aplikácií medzi rôznymi implementáciami CC prostredia, prípadne ich preniesť medzi fyzickým serverom a virtuálnym CC prostredím. Takýto prístup je možný, ak sú splnené dve podmienky. Prvá je, že používateľské dáta môžeme exportovať a importovať zo systémov. Export a import sa väčšinou vykonávajú pomocou štandardizovaných rozhraní a protokolov, ktoré väčšina poskytovateľov CC prostredí a operačných systémov ponúka. Druhá podmienka znie, že zdrojový aj cieľový systém musia poznať rovnaký zápis (sémantiku) dát, aby ich boli schopní správne interpretovať. V prípade, že systémy nedokážu spracovať rovnaký formát dát, malo by byť jednoduché dáta medzi formátmi konvertovať bežne dostupnými nástrojmi, bez straty informačnej hodnoty [27].

Aplikačná portabilita umožňuje preniesť aplikáciu, alebo jej časť medzi rôznymi CC implementáciami, prípadne spustiť aplikáciu virtualizovane aj v prípade, že nebola na takýto účel navrhnutá. Za portabilnú aplikáciu sa považuje aj taká, ktorá by mohla vyžadovať prekompilovanie, prípadne prelinkovanie niektorých knižníc, no v zásade nevyžaduje žiadne väčšie zásahy do svojej štruktúry [27].

Posledným typom je platformová portabilita. V tomto význame je CC platforma podľa [36] definovaná ako infraštruktúra CC služby, ktorá zahŕňa aplikácie umožňujúce používateľom vytvárať a manažovať ich virtuálne prostredia. Platformová portabilita je chápaná ako presun používateľských zdrojov medzi rôznymi CC prostrediami. Zdrojmi sú myslené používateľské dáta nie v zmysle užitočných dát, ako sú napríklad údaje o zamestnancoch či jeho pohľadávkach, ale podporné dáta, ktoré využíva pre fungovanie svojho virtuálneho prostredia. Najčastejším predstaviteľom takýchto dát sú obrazy virtuálnych strojov, ktoré spúšťa vo svojej topológii. Ako príklad môžeme uviesť operačný systém pre webový server. Ak používateľ používa Linux, môže mať pripravený systém s už nainštalovaným webovým serverom, PHP, pomocnými nástrojmi a potrebnými aktualizáciami. Takýto obraz si používateľ môže nahrať do svojho CC prostredia a pri vytváraní nového webového servera je po inštalácii všetko pripravené na spustenie [34] [35].

K platformovej portabilite patrí aj portabilita virtuálnych prostredí. Virtuálnym prostredím sa myslí popis prostredia formou skriptu. Každý významný poskytovateľ CC riešenia ponúka nejakú možnosť, ako skriptom popísať celkové prostredie a entity v ňom. Prostredím rozumieme logickú topológiu siete, IP adresovanie, nastavenie smerovania, DNS a pod. Entity sú spravidla virtuálne stanice (VM), ktoré vykonávajú užitočnú službu pre používateľa (aplikačné servery, firewally, ...). Týmto entitám je možné pomocou vytvorených skriptov nastaviť rôzne údaje, od veľkosti pamäte, jadier CPU, IP adries až po rôzne inicializačné skripty, ktoré sa spustia o prvom nabootovaní VM. Pri rozsiahlejších virtualizovaných prostrediach je práve skriptovanie veľmi často používaným nástrojom administrátorov.

1.4 Špecifikácia problému riešenia

Služby CC sú v súčasnosti na začiatku vývoja a majú mnoho nedostatkov. Jedným z problémov, ktorý označujú mnohí používatelia je „*vendor lock-in*“. Ide o stav, kedy používateľ použije určité rozhranie alebo službu, ktorá je proprietárna len pre daného poskytovateľa. Ak chce používateľ využívať službu iného poskytovateľa, potrebuje signifikantné zmeny svojej služby. Bez zmien musí využívať službu aktuálneho poskytovateľa aj napriek tomu, že iný poskytovateľ ponúka alternatívnu podobnú službu, mnohokrát aj za výhodnejších podmienok.

Riadeniu služieb CC sa venuje skupina štandardizačných organizácií, ktoré sa venujú implementácii služieb CC. Odporúčajú, ako by mala byť riešená spolupráca medzi zainteresovanými stranami, hlavne medzi poskytovateľom a používateľom CC služby. Problém „*vendor lock-in*“ odporúčajú riešiť poskytovaním funkcionalít interoperability a portability.

Interoperabilita, ktorá zabezpečuje vzájomnú komunikáciu platformovo rôznych systémov CC, je základným predpokladom pre využívanie portability. Problematika interoperability je v súčasnosti štandardizovaná a poskytovaná významnými poskytovateľmi CC. Existujúce štandardy sú v súčasnosti dostatočné na poskytovanie elementárnej interoperability medzi rôznymi prostrediami CC služieb, čo umožňuje vytváranie a využívanie CC služieb nad rôznymi prostrediami.

Portabilita znamená migráciu aplikácií, dát alebo systémov medzi platformovo rôznymi CC systémami, respektíve medzi rôznymi CC prostrediami. Mnohé organizácie sa sústreďujú aj na rôzne aspekty portability uvedené v kapitole 1.3. Ich snahou je prevažne portabilita dát a aplikácií, nie portabilita popisov prostredí. V súčasnosti neexistujú ani odporúčania, ktoré by riešili návrhy konkrétnych technických riešení ako migrovať služby CC

medzi rôznymi prostrediami. Čo znamená presun už existujúceho prostredia alebo inštancií služieb z/do iných prostredí bez nutnosti modifikácie týchto inštancií. Rovnako nie je riešené, ako má byť CC služba „rozprestretá“/implementovaná nad CC prostredím viacerých poskytovateľov.

Z analýzy súčasného stavu riešenej problematiky môžeme konštatovať, že v súčasnosti neexistuje unifikovaný spôsob transformácie pre migráciu popisu prostredia CC služby IaaS medzi poskytovateľmi rôznych CC prostredí. Nie je preto možné jednoducho riešiť poskytovanie portability IaaS služieb. V dostupnej literatúre nie je popísané žiadne riešenie, ktoré by umožňovalo automatizovane alebo poloautomatizovane pretransformovať popis implementácie prostredia z jazyka jedného špecifického poskytovateľa do jazyka iného poskytovateľa a tým poskytovať portabilitu CC systému alebo prostredia. Používateľ by si mohol vytvoriť vlastný nástroj, ktorý by dokázal transformovať popis jeho prostredia, no takýto nástroj by bol len jednoúčelový a viazaný na dvoch konkrétnych poskytovateľov. Preto snahou tejto práce je navrhnúť a vytvoriť všeobecný univerzálne použiteľný mechanizmus, využiteľný pre portovanie IaaS služieb medzi rôznymi CC prostrediami

Tento problém s portabilitou je kritický pre používateľov všetkých CC služieb, nielen IaaS. Existujúce riešenia portability neumožňujú plne využiť výhody rastúceho a meniaceho sa trhu CC a vedie v závislosti od hĺbky využívania služieb k prehlbujúcej sa závislosti na jednom riešení jedného dodávateľa.

2 Ciele práce

Z analýzy súčasného stavu riešenej problematiky a zo špecifikácie problému riešenia uvedeného v kapitole 1.4 môžeme konštatovať, že portabilita v CC službe IaaS je v súčasnosti považovaná za hlavnú prekážku vyriešenia problému *vendor lock-in.* Problém nie je iba štandardizačný alebo legislatívny, ale aj technický. Nie je dostupné univerzálne systémové riešenie popisu prostredia z jazyka jedného poskytovateľa do jazyka iného poskytovateľa. Existujúce jednoúčelové riešenia viazané na dvoch konkrétnych poskytovateľov sú časovo aj cenovo náročné.

Preto sme stanovili hlavný cieľ riešenia práce:

„**Návrh systémového prístupu k transformácii CC služby IaaS medzi rôznymi prostrediami a vytvorenie nástroja pre zabezpečenie portability služby IaaS“.**

Hlavný cieľ je rozčlenený do štyroch parciálnych cieľov:

1. Návrh transformačných množín v modelom riadenom vývoji - MDD.
2. Zostavenie všeobecného modelu IaaS služby.
3. Vytvorenie transformačných pravidiel medzi všeobecným modelom a najrozšírenejšími CC implementáciami.
4. Praktické overenie funkčnosti navrhnutých transformácií v nástroji pre implementáciu portability virtuálnych prostredí.

Riešenie bude zamerané výhradne na softvérovú portabilitu služby IaaS CC systémov.

3 Metodika a metódy práce

3.1 Metodika riešenia dizertačnej práce

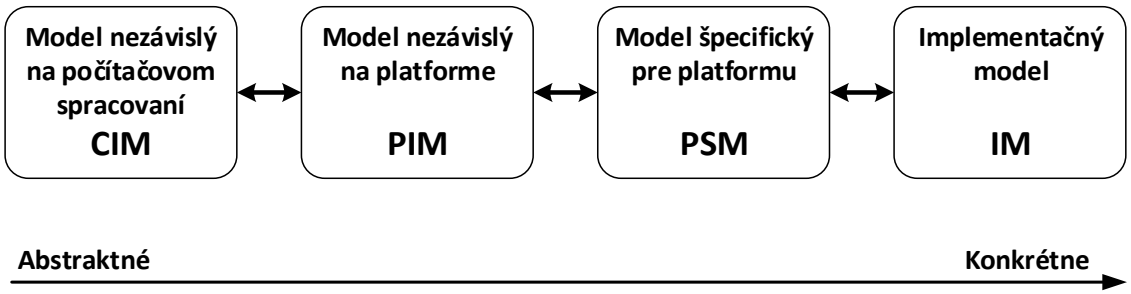
Metodika riešenia vychádza z parciálnych cieľov dizertačnej práce. Etapy riešenia sú zostavené nasledovne:

1. Spracovanie princípov transformácií vhodných k modelom riadenému vývoju použitému pre návrh transformačných množín. Transformácie budú základnými metódami pre návrh transformačných pravidiel.
2. Analyzovanie a vyhodnotenie niektorých prostredí systému CC za účelom získania informácií pre riešenie a overenie nástroja na zabezpečenie portability služby IaaS.
3. Riešenie všeobecného modelu pre popis virtuálnych prostredí. Vytvorenie transformačných pravidiel pre logické entity tvoriace virtuálne prostredie.
4. Pilotná implementácia všeobecného modelu a transformačných pravidiel v zvolenom programovacom jazyku.
5. Overenie implementácie transformačných pravidiel na reálnych CC systémoch.

3.2 Model Driven Architecture - MDA

Modelom riadená architektúra - MDA je jedným z prístupov vývoja softvérového IS, ktorý vyššie spomínané fakty MDD zohľadňuje a reálne uvádza takýto postup do praxe.

MDA je špecifikáciou konzorcia OMG (*Object Management Group*) [39]. Špecifikácia poskytuje návod, ako navrhovať štruktúrované špecifikácie - modely. MDA má špecifikované štyri vrstvy, znázornené na obrázku Obrázok 1.



Obrázok 1: MDA model - rôzne úrovne abstrakcie, zdroj [40]

Platformovo nezávislá vrstva PIM (*Platform Independent Model*) kompletne špecifikuje funkcionality softvérového systému, ktoré sa transformujú do konkrétnej implementačnej platformy. Má určitú mieru nezávislosti od konkrétneho riešenia, aby bolo možné použiť akúkoľvek vývojovú platformu. Popisuje algoritmy a štruktúru softvérového systému do takej miery, aby bola prenositeľná a implementovateľná na rôznych technických

riešeniach, spravidla v modelovacom jazyku UML (*Unified Modeling Language*). PIM obsahuje informácie, ktoré sú dôležité z hľadiska riešenia a vývoja softvérového systému. Môžu to byť algoritmy, rôzne druhy pravidiel či obmedzení a podobne. Transformácia CIM do PIM v súčasnosti neprebieha automaticky, je riešená podľa úvahy softvérového analytika. Softvérový analytik si z neho vyberie len tie špecifikácie, ktoré sú podľa neho zmysluplné z hľadiska počítačového spracovania daného riešenia. A tu vznikajú problémy, že používateľ má určité funkčné požiadavky, ale tie nie sú riešené ako funkcionality softvérového systému. Táto nepriaznivá situácia je problémom vývoja takmer všetkých IS. Ale každý problém je možné riešiť. Tvrdenie, že to nie je možné lebo je to procesný model a PIM je objektový model neobstojí. Jeden z príkladov je v [40] a je riešený nasledovne. OMG špecifikovala špeciálny grafický jazyk BPMN (*Business Process Modeling and Notation*). Modely BPMN je možné previesť do štandardizovaného platformovo nezávislého formátu XPDL (*XML Process Definition Language*). Platformová nezávislosť a rozšíriteľnosť XPDL formátu reprezentuje štandardizovaný sémantický zápis BPMN notácie, a špecifikácia UML modelov pomocou MOF XMI výmenného formátu sú predpokladom pre CIM – PIM transformáciu. Použitie XPDL a MOF XMI formátu v kombinácii s jazykom XSLT, ktorým sú vytvárané transformačné pravidlá aplikovateľné na popisné formalizmy založené na XML, zabezpečuje platformovú nezávislosť (Linux, Windows) pre automatizovaný prístup k CIM – PIM transformácii. Takouto transformáciou je možné vytvárať modely use case a konceptuálne diagramy tried. Iné riešenia sú uvedené v prehľadovom článku [41]. Platformová nezávislosť PIM umožňuje znovupoužiteľnosť pri rôznych zadaniach, pretože PIM nie je viazaný na žiadnu konkrétnu aplikáciu.

Platformovo špecifická vrstva PSM (*Platform Specific Model*) je už závislá na konkrétnej platforme, na ktorej sa bude systém vyvíjať. Pomocou tohto modelu vieme zo všeobecnej špecifikácie vo vrstve PIM vytvoriť konkrétne štruktúry vo výslednej implementačnej platforme (C++, Java, .Net). Tento model dostatočne odráža štruktúru kódu a je dostatočným podkladom pre implementáciu. Môžeme povedať, že ide o vizualizáciu kódu v konkrétnej vývojovej platforme. Vyskytujú sa v ňom totiž objekty, ktoré sa použijú aj vo výslednom procese programovania, ako napr. triedy, konštruktory, prístup k objektom a pod.

Poslednou vrstvou je implementačný model IM, ktorý predstavuje „kód“. V tejto vrstve sa programuje softvér v zvolenom implementačnom jazyku. Výsledkom tejto vrstvy je skompilovateľný kód, ktorý je pripravený na nasadenie a používanie používateľom. Pokiaľ by bolo potrebné upraviť vybrané funkcionality softvéru, zvyčajne sa spustí celý proces návrhu systému nanovo, teda cez úpravu požiadaviek v CIM vrstve, jej transformáciu do nižších vrstiev, až po zmenu implementačného kódu a jeho opätovné nasadenie u používateľa.

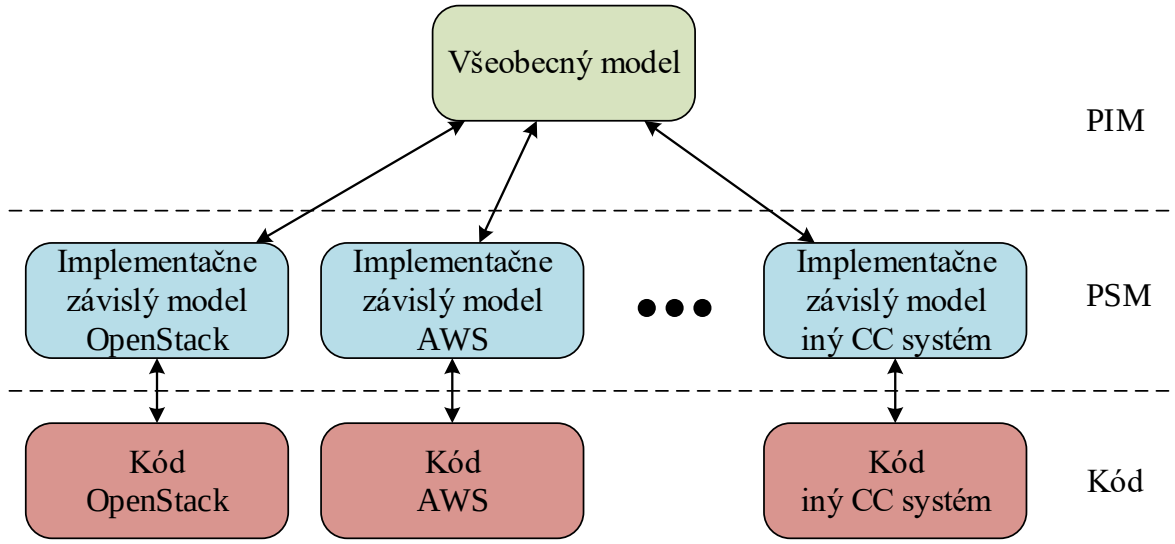
Automatizované transformácie medzi vrstvami PIM - CIM a CIM - PIM sú v súčasnosti funkcionalitami mnohých modelovacích nástrojov.

MDA predpokladá transformácie medzi týmito vrstvami, pričom tvorba vývoja prebieha smerom od najviac všeobecného modelu k modelom viac špecifickým pre platformu, kde sa navrhovaná aplikácia bude implementovať. MDA princípy môžu byť použité aj reverzne a podľa potrieb iba medzi niektorými vrstvami.

4 Návrh všeobecného modelu popisu virtuálneho prostredia

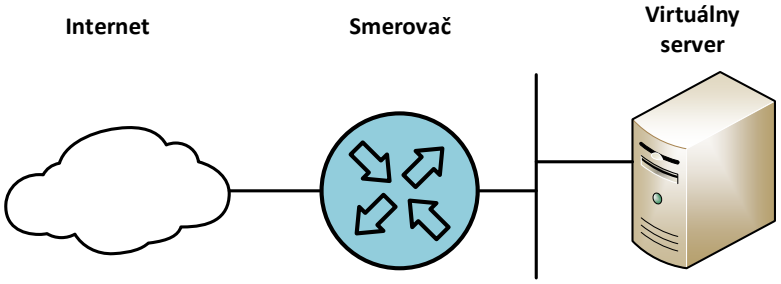
Pri návrhu všeobecného modelu sme sa inšpirovali existujúcimi implementáciami CC IaaS prostredí. Cieľom práce je vytvorenie všeobecného, nezávislého popisu prostredia, ktorý by mohol slúžiť pre akéhokoľvek poskytovateľa CC. Pri návrhu a tvorbe všeobecného modelu sme sa inšpirovali modelovým prístupom, presnejšie princípom modelom riadenej architektúry (MDA), ktorá sa používa pri vývoji softvéru.

Na obrázku Obrázok 2 je znázornený koncept nami navrhovaného modelu v rámci paradigmy MDA. Každý CC poskytovateľ ponúka možnosť svojim používateľom vytvoriť popis prostredí v jazyku špecifickom pre svoje prostredie. Tento popis radíme do roviny PSM. Náš model radíme do roviny PIM, pretože nie je zložený zo špecifických komponentov nejakej CC implementácie, no zachováva všetky atribúty a vlastnosti prostredia, z ktorých by sme mohli vytvoriť funkčný platformovo závislý popis.



Obrázok 2: Návrh všeobecného modelu popisu virtuálneho prostredia, zdroj autor

Virtuálna topológia sa môže skladať z desiatok logických entít IaaS služby, ktoré sú navzájom prepojené. Zámerom tejto práce nie je vytvoriť všeobecný model pre všetky z nich, ale navrhnúť základný model len pre niektoré entity, ktorý sa môže do budúcnosti rozvíjať. Pre demonštračné a testovacie účely sme preto navrhli jednoduchú vzorovú topológiu IaaS služby, ktorá sa skladá z jedného virtuálneho stroja (VM) v privátnej sieti za logickým smerovačom CC prostredia. Vychádzame z vlastnej skúsenosti a taktiež z domnienky, že takáto topológia býva základom aj pre rozsiahlejšie topológie virtuálnych prostredí riešených ako IaaS CC služby. Diagram takejto topológie môžeme vidieť na obrázku Obrázok 3.



Obrázok 3: Testovacia topológia, zdroj autor

Nižšie sú analyzované a uvedené popisy základných logických entít, pomocou ktorých je možné vytvoriť danú vzorovú topológiu. Ďalej sa tu nachádza analýza týchto entít v konkrétnych CC prostrediach, návrh entít vo všeobecnom modeli a následne návrh transformačných pravidiel.

4.1 Identifikácia entít IaaS služby

Pri návrhu modelu sme vychádzali z najrozšírenejších platforiem na trhu – Amazon AWS, OpenStack a Microsoft Azure. Vo všetkých týchto platformách je možné vytvoriť popis prostredia pomocou deklaratívneho jazyka, ktorý popisuje parametre prostredia, jednotlivé entity, ich atribúty a vzájomné prepojenia a závislosti týchto entít. Tohto konceptu sme sa pri návrhu držali aj my, a nami navrhovaný model je taktiež rozdelený na dve hlavné časti – parametre a zdroje (*resources*).

Parametre prostredia slúžia na definovanie hodnôt administrátorom, ktoré je možné jednoducho meniť a nachádzajú sa v popise viackrát. Administrátor takto môže zmenou jedného parametra ovplyvniť viac výskytov jednej hodnoty. Ako príklad môžeme uviesť obraz virtuálneho stroja, z ktorého sa budú vytvárané virtuálne stroje klonovať. Ak sa v topológii nachádza niekoľko identických serverov, v prípade že administrátor požaduje zmenu obrazu pre všetky výskyty, stačí zmeniť jednu hodnotu. Pri každej definícii virtuálneho stoja je definovaný odkaz na parameter, odkiaľ má získať hodnotu. Parametrizovať je možné rôzne hodnoty, či už spomínané obrazy virtuálnych strojov, rozsahy IP adries, názvy virtuálnych sietí či rozsah blokovaných portov pre firewall.

V časti zdrojov sú definované vlastné entity, ktoré sa budú v topológii vyskytovať. Tu prebieha samotná konfigurácia prostredia, definovanie atribútov, závislostí a prepojení medzi entitami. Pre náš návrh a overenie sme vytipovali tieto základné entity: sieť, podsieť, virtuálny server a bezpečnostná skupina.

Pre vytvorenie jednoduchej topológie, ako je znázornení na obrázku Obrázok 3 sme identifikovali päť základných entít, ktoré sú nevyhnutné pre jej funkčnosť. Popisy týchto entít, ako aj ich funkcie vo virtuálnych prostrediach sú uvedené nižšie.

4.1.1 Parametre prostredia

Väčšina popisných skriptov obsahuje parametre. Sú to hodnoty, ktoré definuje administrátor a ktoré mu uľahčujú prácu so skriptom popisujúcim virtuálne prostredie (vrstva PSM). Hodnoty týchto parametrov sú zvyčajne čísla a textové reťazce, no môžu nadobúdať aj hodnoty pravdivostnej logiky.

Na parametre je možné odvolávať sa v rámci definície logických entít, ako napríklad virtuálnych serverov, sietí, bezpečnostných skupín a pod. Analógiu môžeme nájsť v definovaní konštanty v rôznych programovacích jazykoch. Ak programátor potrebuje zmeniť viac výskytov jednej hodnoty, stačí zmeniť definovanú hodnotu konštanty, čo má za následok zmenu hodnoty v celom programe, kde sa konštanta použila.

Každý parameter obsahuje hodnotu, no môže obsahovať aj rôzne obmedzenia hodnoty, ktoré môže administrátor zadať. Sú to napríklad regulárne výrazy ktorými sú obmedzené znaky hodnoty, zoznam či interval povolených hodnôt. Konkrétne typy obmedzení budú rozobraté pri implementáciách CC prostredí.

4.1.2 Virtuálny server

Virtuálny server (VM) je väčšinou kľúčovým prvkom topológie, preto je na neho kladený dôraz pri popise, a taktiež je bohato konfigurovateľný v rámci popisu prostredia. V rôznych systémoch má rôzne parametre, no vo všetkých sú to najmä meno VM, obraz, z ktorého sa bude VM klonovať a určitá forma šablóny, na základe ktorej bude mať VM pridelené výpočtové prostriedky (CPU, RAM, HDD a pod.). Toto sú zvyčajne základné parametre, bez ktorých nie je možné VM v IaaS prostredí spustiť. Následne je možné VM prideliť rôzne dodatočné parametre, ako napríklad sieťové rozhrania, zabezpečenie na základe filtrovania paketov, pridelenie SSH kľúča, či dátové centrum, prípadne konkrétny server, kde

by sa mala VM vytvoriť. Viac sa k týmto parametrom venujeme nižšie, už pri konkrétnych implementáciách CC prostredí.

4.1.3 SSH kľúč

Ako napovedá názov, SSH kľúč slúži na prístup k VM pomocou protokolu SSH (*Secure Shell*). Tento protokol bol vyvinutý pre UNIX-ové systémy, no dnes ho je možné používať aj na pripájanie k iným OS, ako napríklad MS Windows. SSH poskytuje šifrovaný prístup k VM na báze príkazového riadku, cez ktorý je možné systém plnohodnotne ovládať. V CC prostrediach je SSH jedným zo základných protokolov na vzdialený prístup a ovládanie VM. Protokol SSH používa algoritmy kryptografie verejného kľúča (*PKI – Public Key Infrastructure*), kde je potrebný pár kľúčov. Verejný a privátny kľúč, ktoré tvoria neoddeliteľnú dvojicu. Aby bolo možné kľúče použiť, jeden z nich (zvyčajne verejný) musí byť nahratý na VM a druhý použije administrátor pre získanie prístupu k serveru. Entita SSH kľúča slúži na vygenerovanie dvojice kľúčov a importovanie verejného kľúča do VM, aby bolo možné sa na ňu pripojiť.

Pri vytváraní kľúča je potrebné zadať len meno kľúča, ktoré slúži ako jedinečný identifikátor. Voliteľné parametre sú *uloženie kľúča*, *vloženie verejného kľúča* a *používateľ*.

Parameter *uloženie kľúča* hovorí, či administrátor chce dvojicu kľúčov uložiť a v budúcnosti ešte tento pár použiť pri inej VM. Ak nie, vygenerovaný kľuč sa importuje do VM a zahodí, čím vznikne kľúč na jedno použitie, pri zmazaní VM bude zmazaná aj jedna polovica kľúčového páru, čím sa druhá polovica stane nepoužiteľnou.

Druhý voliteľný parameter *vloženie verejného kľúča* hovorí, či chce administrátor VM použiť už existujúci verejný kľúč pre prístup k VM. Ak áno, nebude generovaný nový pár kľúčov, ale zadaný kľúč sa nahrá do VM. Administrátor VM tým docieli situáciu, kde sa môže pomocou jedného privátneho kľúča pripájať k viacerým nezávislým inštanciám VM.

Posledný parameter je *používateľ*, pomocou ktorého je možné vygenerovaný kľúč priradiť konkrétnemu používateľovi. Takéto priradenie sa deje z bezpečnostného hľadiska, kde sa kľúč priradí jednému používateľovi, a ostatní používatelia v rámci CC prostredia nemajú ku kľúčovému páru prístup. V prípade, ak by bol kľúč zdieľaný, ostatní používatelia by mali prístup len k verejnému kľúču, cez ktorý by potencionálne mohli vypočítať druhú polovicu kľúča, čím by dokázali ovládať VM iných používateľov.

4.1.4 Bezpečnostná skupina

Bezpečnostná skupina (*Security Group*) funguje ako jednoduchý filter paketov. Každá VM má pridelenú aspoň jednu bezpečnostnú skupinu, ktorej pravidlá sa aplikujú na každý paket, ktorý prichádza alebo odchádza z VM. Každá bezpečnostná skupina obsahuje pravidlá, ktoré sú vyhodnocované sekvenčne. V CC prostrediach ktoré sme analyzovali dokážu tieto pravidlá len zakazovať pakety. Ak niektorý paket vyhovuje porovnávacej podmienke (pravidlo), je zahodený. Zvyčajne bez ICMP správy, ktorá informuje odosielateľa o zahodení paketu. Ak paket nesplní podmienku ani jedného pravidla, je prepustený ďalej. V pravidlách je možné len zakázať prechod paketu, na konci každej skupiny je implicitné povolenie prechodu. V prípade, že je na jednu VM aplikovaných niekoľko bezpečnostných skupín, pravidlá v nich obsiahnuté vytvoria logický súčet (*or*) a paket je porovnávaný s každým pravidlom. V pravidlách je typicky možné porovnávať hodnoty len v hlavičkách sieťovej a transportnej vrstvy ISO OSI modelu.

4.1.5 Sieť

V CC IaaS systémoch je entita „sieť“ chápaná ako kontajner pre podsiete. Väčšinou nemá vlastný adresný rozsah a slúži čisto ako logický identifikátor. Typicky sa jedna takáto entita pridelí jednému používateľovi, v ktorej si používateľ vytvára podsiete so špecifickými parametrami (adresa siete, adresa DNS servera, podpora DHCP a pod.).

Každá sieť obsahuje unikátny identifikátor, na základe ktorého sú identifikované aj všetky jej podsiete. Na základe tohto identifikátora potom môže poskytovateľ CC prostredia tarifikovať jednotlivých používateľov, za napríklad objem prenesených dát. Okrem identifikácie z pohľadu manažmentu sú identifikátory využívané aj na sieti, kde môžu byť reštrikcie pre používateľa na základe identifikátora. V CC prostrediach sa využívajú virtualizačné techniky siete, ako napr. VXLAN alebo NVGRE, ktoré vytvárajú tunely medzi rôznymi prvkami siete. Na identifikátory tunelov sa veľmi často používajú práve identifikátory sietí, a z toho je možné odvodiť, ktorý používateľ používa ktoré tunely.

Sieť nemá veľa parametrov. Zvyčajne je to len meno, prípadne CIDR (*Classless Inter-Domain Routing*) zápis rozsahu sietí, ktoré sa môžu použiť v jej podsieťach. Amazon AWS používa CIDR zápis, a podsiete môžu mať rozsahy len z tohto rozsahu.

Podsieť

Entita podsiete vždy patrí do nejakej siete. Sieť je v podstate kontajner, ktorý združuje niekoľko podsietí. Podsieť je už vlastnou sieťou, do ktorej sa pripájajú VM. Každá podsieť je definovaná prefixom, ktorý je jedinečný v rámci jednej siete. Ďalšie parametre, ktoré sa definujú v rámci podsiete často bývajú adresa brány, adresa DNS servera, verzia IP protokolu či meno, ktoré má len informačný charakter pre administrátorov. Podsiete sú navzájom nezávislé, a pre prechod dát medzi podsieťami je potrebné smerovanie.

Ako bolo spomenuté vyššie, vybrali sme len základné entity, pomocou ktorých vieme definovať našu vzorovú topológiu. Každý CC systém môže mať desiatky logických entít tvoriacich virtuálne prostredie. Ako príklad uvedieme, že systém OpenStack definuje okolo sto rôznych logických entít a systém AWS dokonca niečo cez dvestopäťdesiat.

4.2 Entity služby v existujúcich IaaS riešeniach

4.2.1 Amazon Web Service

Systém AWS je podľa [13] najpoužívanejšie verejné CC riešenie vo firemnom prostredí. V nasledujúcej časti sa nachádza prehľad vybratých komponentov vzorovej IaaS služby tak, ako ich definuje a používa AWS.

Parametre prostredia

Systém AWS taktiež ponúka využitie parametrov v popisných skriptoch, kde v jednom skripte môže byť najviac šesťdesiat parametrov. Každá entita *parametre prostredia* má vlastné parametre, kde jediným povinným parametrom je *Type*. Ten hovorí o type hodnoty, ktorú môže nadobudnúť. V systéme AWS to môže byť reťazec, číslo a zoznam reťazcov oddelených čiarkou. Na základe typu je potom možné definovať obmedzenia hodnoty, ktoré môže entita parametra nadobúdať. Z nepovinných parametrov je to popis logickej entity daného parametra a implicitná hodnota. Implicitná hodnota môže byť predefinovaná pri spustení skriptu, no ak administrátor nepovie inak, použije sa práve definovaná implicitná hodnota.

Ďalšie nepovinné parametre obmedzujú použiteľné hodnoty v entite parametra. Pri reťazci to môže byť minimálna alebo maximálna dĺžka, prípadne povolené znaky (*AllowedPattern*), ktoré sa definujú pomocou regulárnych výrazov. Pri číselných hodnotách je možné obmedziť interval, z ktorého môžu hodnoty pochádzať. Čísla môžu nadobúdať ako celé, tak aj desatinné hodnoty. Posledným parametrom je *AllowedVallues*, ktorý obsahuje zoznam predvolených hodnôt. Z tých si môže administrátor jednu vybrať pri aplikovaní skriptu. Benefit je v tom, že nemôže dôjsť k preklepu, prípadne k syntaktickej chybe.

Virtuálny server

Ako bolo spomenuté vyššie, virtuálny server (*VM – Virtual Machine*) je jedným zo základných kameňov tvorby popisných skriptov. V skriptoch sa označuje ako *AWS::EC2::Instance*, a je bohato konfigurovateľný. VM má v systéme AWS jeden povinný parameter. Je ním *ImageId*, čo je referencia na obraz operačného systému, z ktorého bude daná VM klonovaná. Je možné použiť obrazy ktoré nachystal poskytovateľ AWS, alebo je možné si vytvoriť vlastné obrazy a tie importovať do systému AWS. V každom prípade už musia existovať, aby bolo možné vytvoriť klonovanú VM.

Všetky ostatné parametre sú nepovinné. Parameter *InstanceType* je šablóna, podľa ktorej budú VM pridelené výpočtové prostriedky, ako napr. počet jadier CPU, pamäť RAM, či pevný disk. Na základe využívania týchto hodnôt je následne používateľ spoplatňovaný. Šablóny nie je možné vytvárať na požiadanie, teda je možné použiť len existujúce, ktoré ponúka AWS. Pokiaľ tento parameter nie je definovaný, použije sa predvolená hodnota. Ďalším parametrom sú bezpečnostné skupiny, ktoré fungujú ako paketové filtre. Viac o bezpečnostných skupinách sa nachádza v časti 4.4.4.

Parameter *AvailabilityZone* označuje dátové centrum spoločnosti Amazon, kde sa daná VM vytvorí. V súčasnosti má spoločnosť Amazon 16 dátových centier umiestnených po celom svete. Takto si môže používateľ vybrať, kde sa nová VM vytvorí. To môže mať dôležitý dopad na jeho dáta, pretože rôzne krajiny majú rôznu legislatívu na prístup k privátnym dátam. Ďalším parametrom je *SubnetId*, čo je referencia na podsieť, do ktorej bude VM pripojená. Týchto parametrov môže byť viac, čím administrátor docieli pripojenie jednej MV do viacerých podsietí. Viac o podiseťach sa nachádza v časti 4.4.6. Parameter *KeyName* odkazuje na SSH kľúč, ktorý sa použije na vzdialený prístup k VM. SSH kľúčom sa venujeme v časti 4.4.3.

Veľmi dôležitým parametrom je *UserData*. Pomocou tohto parametra je možné vo VM spustiť skript pri prvom bootovaní. Vlastný skript môže byť v akomkoľvek jazyku, ktorý je podporovaný operačným systémom, či už je to UNIX-ový Shell, alebo PowerShell pre systémy Microsoft Windows. Na základe týchto skriptov je možné modifikovať VM podľa požiadaviek. Či už sú to rôzne nastavenia alebo inštalácie programov, po použití takéhoto skriptu sa vytvorí VM, ktorú nie je potrebné ďalej modifikovať a môže ihneď slúžiť zámeru, s ktorým bola vytváraná. Posledným parametrom sú tagy, čo sú len jednoduché prívlastky, ktoré slúžia administrátorom pre jednoduchšiu identifikáciu a utrieďovanie virtuálnych inštancií.

SSH kľúč

V systéme AWS nie je možné vytvoriť SSH kľúč pomocou popisného skriptu. Jediná možnosť vytvorenia kľúča je cez webovú stránku nazývanú „*AWS Management Console*“. Tam si používateľ vytvorí kľúč, pomenuje, prípadne uloží jeho privátnu časť. V popisných skriptoch sa potom už len odvolá na meno existujúceho kľúča, ktorý bude importovaný

do vytváranej VM. Používatelia tak nemajú možnosť vytvorenia SSH kľúčov v skriptoch a musia si kľúče vytvoriť predtým, ako začnú vytvárať popisné skripty prostredia.

Bezpečnostná skupina

Bezpečnostná skupina môže existovať úplne autonómne. Jej názov je „AWS::EC2::SecurityGroup“ a je ju možné definovať cez webové rozhranie, ako aj cez popisný skript. V systéme AWS má každá bezpečnostná skupina jeden povinný parameter, popis danej skupiny. Meno skupiny je jedinečný identifikátor v rámci jedného projektu, ktorý ale nie je povinným parametrom. Ak meno bezpečnostnej skupiny nie je definované, systém AWS vygeneruje unikátne meno pre danú skupinu. Nasledujú ďalšie dva nepovinné parametre, referencia na skupiny vstupných a výstupných pravidiel. Posledným parametrom sú *Tagy*, ktoré slúžia opäť len na označenie a popis skupiny.

Skupiny vstupných a výstupných parametrov majú totožnú syntax. No ako napovedá názov, líšia sa len v smere nasadenia filtrovania paketov. Smer sa určuje z pohľadu VM, to znamená, že vstupné pravidlá budú aplikované na pakety, ktoré sú určené pre danú VM, a výstupné na tie, ktoré vygenerovala VM. Tieto skupiny môžu byť zdieľané pre viaceré bezpečnostné skupiny, nakoľko to sú samostatné logické entity, ktoré majú názov „*AWS::EC2::SecurityGroupIngress*“, resp. „*AWS::EC2::SecurityGroupEgress*“.

V týchto skupinách sa nachádzajú štyri povinné parametre. Prvým z nich je *IpProtocol*, ktorý trochu zavádzajúco označuje transportný protokol nesený v pakete. Môže nadobúdať číselné alebo slovné parametre. Zo slovných je to *TCP, UDP* a *ICMP*. Číselné hodnoty označujú čísla príslušných protokolov podľa IANA (*Internet Assigned Numbers Authority*), napríklad *TCP* má číselnú hodnotu „6“. Ak by administrátor chcel špecifikovať všetky transportné protokoly, nastaví hodnotu parametra na „-1“.

Ďalšie dva povinné parametre spolu súvisia. Sú to *FromPort* a *ToPort*. Tieto dva parametre určujú interval portov, ktorý bude porovnávaný v danom pravidle. Ak by administrátor potreboval definovať len jeden port, obidva tieto parametre nastaví na rovnakú hodnotu. V prípade transportného protokolu ktorý nepoužíva transportné porty (napr. ICMP), sú tieto hodnoty ignorované, no z hľadiska zrozumiteľnosti a čitateľnosti skriptu sa odporúča zadať hodnotu, ktorá je neprípustná, napríklad „-1“.

Posledným parametrom je *CidrIp*, ktorý určuje rozsah IP adries, ktoré budú v danom pravidle porovnávané. Ako napovedá názov, hodnota sa zapisuje v tzv. *CIDR* formáte. Tento parameter má rozdielne vnímanie vo vstupných a výstupných pravidlách. Pri vstupných pravidlách sa porovnáva so zdrojovými adresami, zatiaľ čo pri výstupných pravidlách s cieľovými IP adresami. Pokiaľ by administrátor nastavil hodnotu na „0.0.0.0/0“, označil by tým úplne všetky adresy, čo by malo za následok porovnávanie všetkých paketov s týmto pravidlom.

Z nepovinných parametrov tu môžeme nájsť názov skupiny, ktorý je opäť jedinečný v rámci jedného projektu. Na nasledujúcom obrázku môžeme vidieť diagram závislostí bezpečnostnej skupiny v CC systéme AWS.

Sieť

V systéme AWS má sieť jediný povinný parameter – „CidrBlock“, ktorý obsahuje CIDR záznam. Všetky podsiete z tejto siete musia mať adresy daného CIDR záznamu. Ďalej obsahuje nepovinné atribúty ako napríklad *tagy*. O sieť sa stará modul EC2, a v skriptoch sa zapisuje ako AWS::EC2::VPC, kde skratka VPC znamená „Virtual Private Cloud“.

Podsieť

V systéme AWS sú podsiete súčasťou entity sieť. Entita siete má ako povinný atribút aj IP prefix, a prefixy podsietí musia byť z rozsahu, ktorý je definovaný v sieti. Inými slovami, entita siete sumarizuje všetky prefixy zo svojich podsietí. Ďalší povinný atribút je referencia na entitu siete. Pomocou tejto referencie je podsieť jednoznačne pridelená k jednej sieti, čo má za následok možnosť vytvárať podsiete s rovnakými prefixami pre rôznych používateľov. Ako bolo spomenuté vyššie, entita siete má jednoznačný identifikátor, ktorý týmto prechádza aj na všetky jej podsiete. Spoločnosť Amazon tak môže jednoznačne monitorovať a prípadne spoplatňovať svojich používateľov na základe tohto identifikátora.

Niektoré hodnoty, ktoré je možné konfigurovať v iných CC systémoch sú určené poskytovateľom CC prostredia AWS. Napríklad adresy DNS serverov sú vopred zadané, a pre VM v tomto prostredí je možné používať výhradne DNS servery poskytované spoločnosťou Amazon. Taktiež nie je možné nastaviť IP adresu brány v danej podsieti. Tá bola opäť stanovaná poskytovateľom na pevnú hodnotu, ktorú používateľ dokonca nepozná. Všetky VM v tomto systéme získavajú IP adresy pomocou DHCP protokolu, cez ktorý sa dozvedia aj adresu svojej brány, ktorá sa môže týmto meniť bez toho, aby bola ovplyvnená dátová prevádzka.

4.2.2 OpenStack

Podľa [13] je systém OpenStack najrozšírenejším otvoreným (*open-source*) CC riešením vo firemnom prostredí. V nasledujúcej časti sa nachádza prehľad identifikovaných IaaS entít spolu s popismi povinných a nepovinných parametrov v tomto systéme.

Parametre prostredia

Podobne ako AWS, aj systém OpenStack povoľuje používanie parametrov v popisných skriptoch. Taktiež entity parametrov majú vlastné parametre, kde povinným je typ. Typ určuje hodnotu parametra, ktorými môže byť reťazec, číslo, zoznam reťazcov a hodnota Boolovej algebry. Na rozdiel od systému AWS OpenStack dovoľuje použitie logických výrazov pravda/nepravda, na základe ktorých je potom možné pri definovaní logických entít vetviť program a tak lepšie prispôsobiť výsledné riešenie.

Z nepovinných parametrov to je popis daného parametra, implicitná hodnota a obmedzenia vlastnej hodnoty parametra. Implicitná hodnota je použitá v prípade, ak administrátor pri aplikovaní skriptu nepredefinuje hodnotu parametra. *Constraints* obsahuje zoznam obmedzení pre vlastnú hodnotu parametra. Podobne ako v systéme AWS môže administrátor definovať minimálnu a maximálnu dĺžku reťazca, interval hodnôt pre číselné parametre, či regulárny výraz obmedzujúci znaky v textovom reťazci. Nechýba ani zoznam hodnôt, z ktorých má administrátor možnosť výberu pri aplikovaní skriptu.

Virtuálny server

V popisných skriptoch sa VM v systéme OpenStack označuje ako *OS::Nova::Server*. Tento názov naznačuje, že o VM sa stará modul *Nova*, ktorý je všeobecne zodpovedný za vytváranie, prevádzku a modifikáciu virtuálnych inštancií koncových staníc. V systéme OpenStack má VM jeden povinný parameter. Na rozdiel od systému AWS to nie je obraz z ktorého sa bude nová VM klonovať, ale je to šablóna, podľa ktorej budú VM pridelené výpočtové zdroje (CPU, RAM, HDD). V systéme OpenStack sa táto šablóna nazýva *flavor*. Z nepovinných parametrov to je meno VM, ktoré je jedinečné v rámci projektu a názov

obrazu operačného systému, z ktorého sa nová VM bude klonovať. OpenStack býva nasadzovaný väčšinou ako privátny CC systém, preto sa o obrazy (nazývané *image*) musí starať prevádzkovateľ systému. Existujú dve možnosti – buď si prevádzkovateľ bude vytvárať a aktualizovať vlastné obrazy, alebo si zadováži už existujúce obrazy. Väčšina Linuxových distribúcií ponúka zdarma svoje aktuálne obrazy systémov, ktoré stačí jednoducho importovať do CC systému.

Ďalší nepovinný parameter je bezpečnostná skupina (*SecurityGroup*), čo je jednoduchý filter paketov na úrovni CC systému. Viac o bezpečnostných skupinách sa nachádza v časti 4.4.4. Parameter *networks* slúži na pripojenie VM k sieti. V rámci tohto parametra je možné pripojiť VM do jednotlivej podsiete, alebo do všetkých podsietí v rámci jednej siete. Entita siete je rozobratá v časti 4.4.5 a podsiete v časti 4.4.6.

Ďalším nepovinným parametrom je referencia na SSH kľúč, ktorý slúži na vzdialený prístup a manažment VM. Viac o SSH kľúčoch pojednáva sekcia 4.4.3. Parameter *user_data* dovoľuje v novovytvorenej VM spustiť ľubovoľný skript. Tento sa spustí len pri prvom spustení VM a dokáže modifikovať VM podľa predstáv administrátora, či už v rôznych nastaveniach systému, alebo môže nainštalovať a nastaviť nové aplikácie podľa potreby. Skript môže byť napísaný v akomkoľvek jazyku, ktorý je spustiteľný v danom operačnom systéme. Posledný nepovinný parameter je *tags*, čo sú pomocné označenia VM. Tie slúžia len pre administrátora, na základe ktorých môže VM triediť a filtrovať v rôznych výpisoch. Tento parameter nemá žiadny vplyv na funkčnosť VM a systému ako celku.

SSH kľúč

V systéme OpenStack je možné kľúč vytvoriť aj cez webový prehliadač, prípadne cez príkazový riadok, aj priamo v rámci skriptu. Ak sa kľúč vytvára cez prehliadač, obdobne ako v systéme AWS kľúč musí existovať pred spustením skriptu a v rámci skriptu sa nachádza už iba referencia na meno kľúča, ktoré musí byť jedinečné.

Pri vytváraní kľúča v rámci popisného skriptu je postup veľmi podobný. Treba definovať meno, prípadne iné voliteľné parametre. Následne sa v rámci definovania VM nachádza referencia na vytvorený kľúč. O prístup k VM a manažment SSH kľúčov sa stará výpočtový modul Nova a entita má v skripte názov OS::Nova::KeyPair. Ak chce administrátor kľúč nielen uložiť do CC prostredia, ale aj ho zobraziť, je možné použiť tzv. výstupy, cez ktoré systém zobrazí privátnu časť dvojice kľúčov.

Bezpečnostná skupina

CC systém OpenStack používa obdobnú logiku bezpečnostnej skupiny ako systém AWS. Taktiež je to samostatná logická entita bez závislosti na iných entitách. Je ju možné definovať buď cez webové rozhranie, resp. cez príkazový riadok, alebo pomocou skriptu, kde jej názov znie „*OS::Neutron::SecurityGroup*“. Nemá žiadne povinné parametre, obsahuje len tri nepovinné. Prvým je *description*, ktorý slúži len ako popis danej skupiny. Druhým parametrom je jej meno, ktoré je jedinečné v rámci jedného projektu a tvorí akýsi identifikátor danej skupiny. Posledným parametrom je *rules*, čo sú vlastné bezpečnostné pravidlá. Na rozdiel od systému AWS bezpečnostné pravidlá nie sú ako samostatné logické entity. Pravidlá sú pevne integrované do bezpečnostnej skupiny. Na obrázku **Chyba! Nenašiel sa žiaden zdroj odkazov.**, kde sa nachádza diagram závislostí sú pravidlá vyčlenené akoby samostatná entita, no je to len z dôvodu jednoduchšej orientácie a pochopenia, pretože každé pravidlo má vlastné parametre, ktoré definujú jednotlivé pravidlá.

Každé pravidlo má päť parametrov. Prvý je *direction*, ktorý označuje smer, či bude pravidlo aplikované na prichádzajúce, či na odchádzajúce pakety Pokiaľ nie je definovaný,

implicitne sa porovnávajú prichádzajúce pakety. Ďalšie štyri parametre sú zhodné ako v systéme AWS. *Protocol* označuje transportný protokol nesený v tele paketu. Akceptované sú len tri hodnoty: *TCP, UDP* a *ICMP*. Číselné hodnoty nie sú prípustné. Ďalej dvojica parametrov *port_range_min* a *port_range_max* označujú interval portov, ktoré budú porovnávané v rámci pravidla. Posledným pravidlom je *remote_ip_prefix*, ktorý určuje rozsah IP adries porovnávaných v pravidle. Akceptovanou hodnotou je IP prefix v CIDR zápise.

Sieť

V systéme OpenStack sa všeobecne os siete stará modul Neutron. Ako vo všetkých moduloch, aj Neutron využíva pre názvy entít balíčkovaciu konvenciu. Sieť má názov „Net“ a zapisuje sa ako „OS::Neutron::Net“.

V tomto systéme sieť nemá žiadne povinné parametre, všetky parametre sú nepovinné. My sme zvolili pravdepodobne najpoužívanejšie parametre: meno a tagy. Na nasledujúcom obrázku sa nachádza model závislostí a používaní entity sieť v systéme OpenStack.

Podsieť

Oproti systému AWS je podsieť viac konfigurovateľná. Na jednej strane sú to dva zhodné povinné parametre (označené symbolom plus), na druhej strane sú tu nepovinné parametre, ktoré AWS neponúka. Z povinných parametrov je to referencia na entitu siete, pod ktorú daná podsieť patrí, a zhodne ako v AWS, aj tu môže byť každá podsieť jednoznačne identifikovateľná na základe siete, pod ktorú patrí. Druhý parameter je IP prefix, ktorý môže byť ľubovoľný. Entita siete v systéme OpenStack nemá parameter IP prefixu a z toho vyplýva, že podsiete nemusia nutne byť z nejakého pevne definovaného rozsahu.

Z nepovinných parametrov je to napríklad meno, ktoré má len informatívny charakter, ďalej adresu DNS servera, či adresu brány v danej podsieti. Na rozdiel od systému AWS je možné tieto adresy definovať pre každú podsieť zvlášť. Ďalším parametrom je verzia IP protokolu, kde si môže administrátor nastaviť, či bude daná podsieť používať IPv4 alebo IPv6. Každá podsieť môže používať len jednu verziu súčasne. Ak by administrátor chcel zapojiť VM do siete IPv4 aj IPv6, musí vytvoriť dve podsiete, každú z inou verziou IP protokolu, VM prideliť dva sieťové rozhrania a následne každé rozhranie pripojiť do inej siete. V systéme OpenStack nie je možné prevádzkovať jednu podsieť v tzv. „*dualstack*“ režime. Posledným nepovinným parametrom sú *tagy*, čo sú informácie pre administrátorov, na základe ktorých môže podsiete triediť do rôznych kategórií bez ohľadu na ich funkčnosť alebo príslušnosť k nadradenej sieti.

4.2.3 Microsoft Azure

Microsoft Azure je podľa [13] druhý najpoužívanejší verejný CC systém vo firemnom prostredí v roku 2017. V nasledujúcej časti sa nachádzajú identifikované IaaS entity v tomto systéme, spolu s ich povinnými a nepovinnými parametrami.

Parametre prostredia

Rovnako ako v systémoch AWS a OpenStack, aj v systéme Azure je možné definovať parametre prostredia, na ktoré sa je možné neskôr odvolávať v popisných skriptoch virtuálnych prostredí. Aj v systéme Azure sa v entite *parametre prostredia* nachádza jediný povinný parameter – *type*. Ten definuje typ parametra, či je to textový reťazec, číslo, pole hodnôt, či logická hodnota typu pravda/nepravda.

Ostatné parametre sú nepovinné. Jedným z nich je popis daného parametra (*description*), ktorý je na rozdiel od ostatných parametrov o jednu úroveň hlbšie, v časti *metadata*. Medzi ďalšie parametre patria *implicitná hodnota*, ktorá je použitá v prípade, ak administrátor nezadá vlastnú hodnotu, ohraničenie dĺžky reťazca (*minLength* a *maxLength*), interval povolených hodnôt pre číselné hodnoty (*minValue* a *maxValue*), alebo zoznam hodnôt, z ktorých má administrátor možnosť výberu pri aplikovaní skriptu.

Virtuálny server

Jednou zo základných entít je virtuálny server (VM). Microsoft Azure nepoužíva tzv. balíčkovaciu notáciu, ktorá oddeľuje jednotlivé časti hierarchie dvomi dvojbodkami. Namiesto dvoch dvojbodiek používa lomku. Virtuálny server sa nachádza v časti „Microsoft.Compute“, a celý názov definície VM je „Microsoft.Compute/virtualMachines“.

Podobne ako v systémoch AWS a OpenStack, aj v systéme Azure sa nachádzajú povinné a nepovinné parametre. Jediným povinným parametrom je názov VM - „name“. Všetky ostatné parametre sú nepovinné, a pokiaľ ich administrátor nenastaví v automatizačnom skripte, doplnia sa predvolenými hodnotami.

Na obrázku **Chyba! Nenašiel sa žiaden zdroj odkazov.** sú znázornené parametre VM v systéme Azure. Na rozdiel od systémov Amazon AWS a OpenStack sú parametre viac štruktúrované. Identifikátor obrazu, z ktorého sa bude VM klonovať má názov „storageProfile/imgeReference/id“, kde lomky oddeľujú jednotlivé úrovne. V tomto prípade sa samotné „id“ nachádza v časti „imageReference“, ktorá sa samotná nachádza v „storageProfile“. Šablóna pre VM, ktorá nastaví počet jadier CPU, veľkosť RAM a HDD má názov „vmSize“ a nachádza sa v časti „hardwareProfile“. Pri definícii VM je ju možné spustiť v bližšie špecifikovanej časti CC prostredia, ako napríklad definovať konkrétnu množinu, či jednotlivý fyzický server. Tieto servery sa nachádzajú v skupinách nazývaných zóny dostupnosti. Pre vytvorenie VM v špecifickej zóne je potrebné definovať direktívu „zones“.

V časti „networkProfile/networkInterfaces/id“ je možné definovať pripojenia VM do virtuálnych sietí. Vo väčšine CC IaaS prostredí je možné definovať inicializačný skript, ktorý sa vykoná pri prvom spustení VM. Aj v systéme Azure je možnosť spustiť takýto skript. Administrátor ho môže definovať v časti „OSProfile/customData“. Pre vzdialené prihlasovanie sa na VM je možné zadať hodnotu verejného kľúča. Tá sa definuje hlboko v štruktúre skriptu, konkrétne v „OSProfile/linuxConfiguration/ssh/publicKeys/keyData“. Posledným parametrom je „tags“, čo sú značky, ktoré môže administrátor priradiť konkrétnej VM. Použitie je identické ako v systémoch OpenStack a Amazon AWS.

SSH kľúč

V systéme Azure, podobne ako v systéme Amazon AWS nie je možné vytvoriť SSH kľúč pomocou automatizačného skriptu. Podľa dokumentácie je jediná možnosť vytvoriť kľúč vo webovom rozhraní a následne hodnotu verejného kľúča uviesť v skripte. Nie je možné sa odvolať na meno kľúča, vždy je nutné zadať hodnotu kľúča. Azure nepodporuje ani možnosť importovať do VM už existujúci SSH kľúč.

Bezpečnostná skupina

Bezpečnostná skupina funguje ako paketový filter, ktorým môže administrátor obmedzovať IP prevádzku VM na úrovni CC systému. Bezpečnostná skupina má rovnakú logickú štruktúru ako v systéme AWS, kde je definovaná entita skupiny, ktorá následne využíva ďalšie entity, v ktorých sú definované vlastné pravidlá. Entita bezpečnostnej skupiny sa definuje ako „Microsoft.Network/networkSecurityGroups“ a má jeden povinný parameter,

„name“, ktorý označuje meno danej skupiny. Nasledujú dva nepovinné parametre, jedným z nich je „securityRules“, v ktorom sa nachádza zoznam entít pravidiel bezpečnostných skupín. Druhý nepovinný parameter je „tags“, ktorý dovoľuje administrátorom prideliť bezpečnostnej skupine určité značky.

Entita pravidiel bezpečnostných skupín je definovaná ako „Microsoft.Network/networkSecurityGroups/securityRules“. Má tri povinné a tri nepovinné parametre. Prvý povinný parameter je „name“, ktorý nastavuje meno pravidiel, ďalej to je „direction“, ktorý definuje smer toku paketov. Môže obsahovať dve hodnoty – ingress a egress, kde obe hodnoty sú definované z pohľadu VM. To znamená, že ak je nastavená hodnota ingress, porovnávané sú všetky pakety, ktorých cieľová adresa je adresa VM. Posledným povinným parametrom je „protocol“. Tento parameter definuje transportný protokol nesený v IP, a môže nadobúdať hodnoty TCP, UDP a ICMP.

Z nepovinných parametrov môže administrátor definovať „description“, čo je popis definovaných pravidiel a dvojica „sourceAddressPrefix“ a „sourcePortPrefix“. Tie definujú zdrojovú IP adresu a zdrojový port. Zdrojová adresa a port sú dostačujúce, pretože ak by administrátor potreboval definovať cieľovú adresu a port, môže jednoducho zmeniť definovaný smer pravidiel.

Obdobne ako v systémoch AWS a OpenStack, ani systém Azude nedovoľuje zakazovať pravidlami prevádzku. Všetky pravidlá prevádzku povoľujú, a na konci pravidiel je vždy implicitný zákaz všetkého, čo nebolo pravidlami povolené.

Sieť

Systém Azure má odlišnú filozofiu prideľovania podsietí do sietí, ako systémy AWS a OpenStack. V AWS a OpenStack môže administrátor prideliť podsieť do siete. To znamená, že v definícii podsiete je špecifikované, do akej siete bude patriť. V systéme Azure je to opačne, v sieti sa nachádza definícia, ktoré podsiete do nej patria.

Názov definície siete v systéme Azure je „Microsoft.Network/virtualNetworks“. Sieť má jeden povinný parameter – meno siete nazvaný „name“. Okrem mena sa tu nachádzajú dva nepovinné parametre. Prvý je „subnets“, ktorý definuje zoznam podsietí patriacich do siete. Druhý parameter sú značky, ktoré môže administrátor prideliť ku sieti a nazýva sa „tags“.

Rovnako ako v systéme OpenStack, ani v systéme Azure nie je nutné, aby podsiete jednej siete patrili do definovaného rozsahu, ktorý je uvedený v definícii siete.

Podsieť

Podsieť sa v systéme Azure nachádza v rovnakej časti, „virtualNetworks“ a definuje ako „Microsoft.Network/virtualNetworks/subnets“. Má tri parametre, z toho je povinné len meno podsiete, nazvané „name“. Nepovinný parameter „addressPrefix“ definuje adresový rozsah podsiete vo formáte CIDR. Posledným parametrom je „tags“, ktorý obdobne ako pri všetkých ostatných entitách dovoľuje administrátorom pridať ku konkrétnej podsieti značky, ktoré môže administrátor využité pre jednoduchšie identifikovanie entity.

4.3 Návrh všeobecného modelu

Na základe analýzy existujúcich IaaS CC riešení tu v práci predkladáme návrh vlastného všeobecného modelu, ktorý je jednou z nosných tém ktoré práca rieši. V terminológii MDA tento model korešponduje s vrstvou PIM. Tu navrhovaný a predkladaný všeobecný model obsahuje entity IaaS služby, ich parametre a vzťahy a to spôsobom, ktorý je

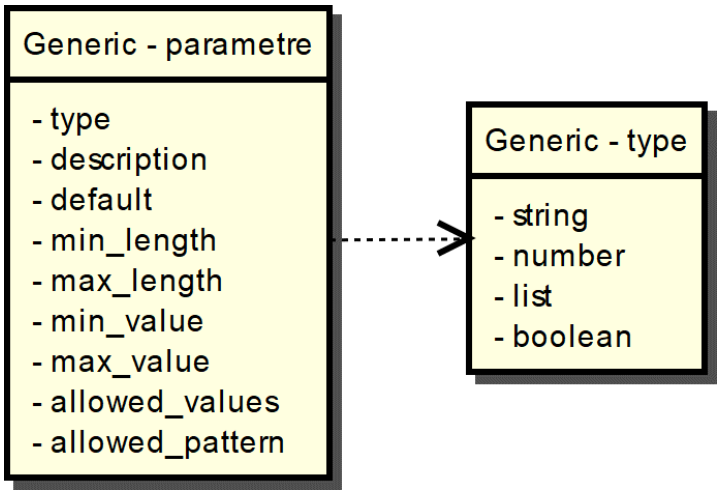
nezávislý od konkrétnej implementácie IaaS služby. Model je flexibilne rozšíriteľný. S implementáciou nástroja, ktorý obsahuje tu navrhovaný všeobecný model, platformovo závislé modely ako aj navrhované transformačné pravidlá, bude možné vykonávať transformácie popisných skriptov IaaS služby (entít, parametrov a ich vzťahov) medzi rôznymi CC systémami (medzi všeobecným a závislým modelom a naopak). Práca tak rieši identifikovaný problém portability IaaS služieb.

V nasledujúcej časti uvádzame vlastný návrh logických entít IaaS služby pre všeobecný model IaaS služby. Pri návrhu sme vychádzali z už existujúcich systémov aj z dôvodu jednoduchšej tvorby transformačných pravidiel.

4.3.1 Parametre prostredia

V navrhovanom všeobecnom modeli sa taktiež nachádza podpora pre parametre skriptov. Obdobne ako v systémoch AWS a OpenStack, aj v tomto modeli môžu mať logické entity parametrov vlastné parametre. Jedným z nich je typ, ktorý hovorí o type vlastnej hodnoty parametra. Ďalej sa tu nachádza popis entity parametra a implicitná hodnota, ktorá bude použitá v prípade, ak administrátor nedefinuje novú hodnotu pri aplikovaní skriptu. Opäť ako v predchádzajúcich systémoch, aj tu sa nachádzajú obmedzujúce faktory pre vlastnú hodnotu parametra. Môže to byť dĺžka reťazca, interval hodnôt pre číselné hodnoty, regulárny výraz obmedzujúci znaky v reťazci, alebo zoznam predvolených hodnôt.

Na nasledujúcom obrázku sa nachádza logické členenie parametra v navrhovanom modeli. Entita „Generic - type“ zobrazuje hodnoty, ktoré môžu byť v parametri *type*. Toto vyčlenenie je urobené z hľadiska názornosti a pochopenia konceptu parametra.



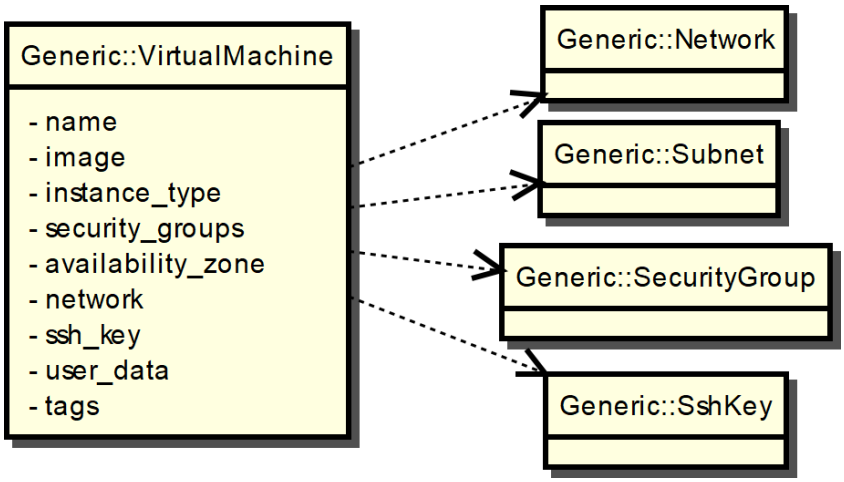
Obrázok 4: Parametre vo všeobecnom systéme, zdroj autor

4.3.2 Virtuálny server

Medzi CC systémami AWS a OpenStack prakticky nie je rozdiel. OpenStack má navyše názov VM, čo je možné do systému AWS jednoducho pretransformovať pomocou parametra *Tags*. Ďalšia odlišnosť je fakt, že v systéme AWS nie je možné priradiť VM do celej siete, a tým pádom do všetkých jej podsietí. VM môže byť priradená len do konkrétnych podsietí. Ak je v systéme OpenStack VM pripojená do celej siete, vieme toto

pripojenie interpretovať ako pripojenie do všetkých podsietí danej siete a následne jednoducho transformovať do systému AWS.

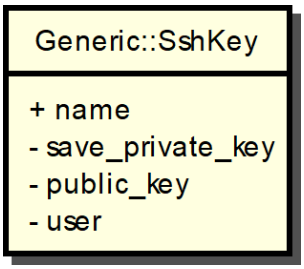
Na obrázku nižšie sa nachádza diagram závislostí v navrhovanom všeobecnom modeli. Diagram je identický ako diagram systému OpenStack. Keďže sa v ňom nenachádzajú žiadne povinné parametre, na rozdiel od systému OpenStack sa v ňom nenachádza logická entita šablóny výpočtových zdrojov pre VM. Ako bolo spomenuté, rozdiely v diagramoch medzi CC systémami AWS a OpenStack je možné pomerne jednoducho vyriešiť. Z toho dôvodu si môžeme dovoliť prevziať popis jedného systému, ktorý má v niektorých ohľadoch väčšie možnosti konfigurácie.



Obrázok 5: VM vo všeobecnom modeli, zdroj autor

4.3.3 SSH kľúč

V rámci všeobecného modelu sme sa nechali inšpirovať systémom OpenStack. Môžeme povedať, že tieto dve entity sú totožné, nakoľko sa domnievame, že funkcionalita je plne postačujúca pre akékoľvek IaaS prostredie. Na nasledujúcom obrázku sa nachádza diagram závislostí SSH kľúča v navrhovanom všeobecnom modeli. Rovnako ako v systéme OpenStack je jediným povinným parametrom meno kľúča.

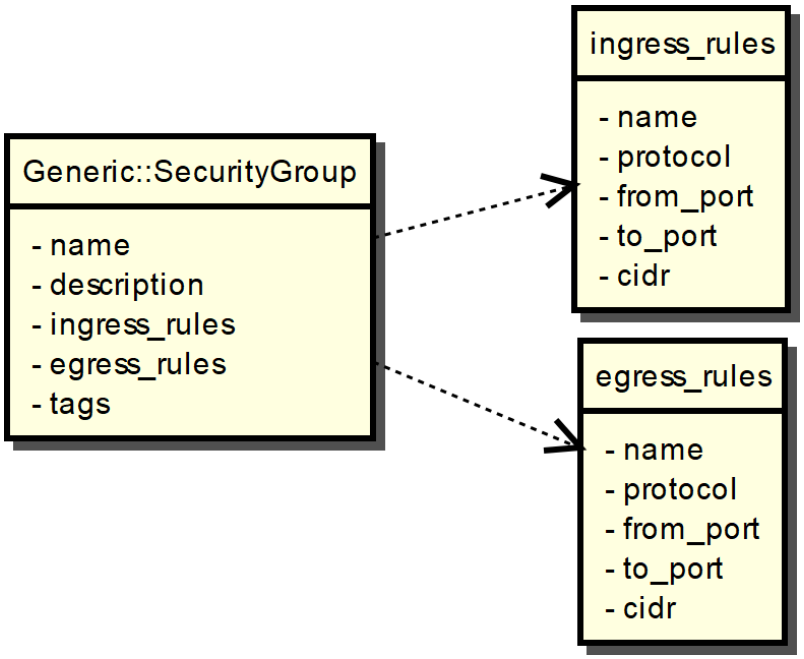


Obrázok 6: SSH kľúč vo všeobecnom modeli, zdroj autor

4.3.4 Bezpečnostná skupina

Vo všeobecnom modeli sme sa nechali inšpirovať systémom AWS. Myslíme si totiž, že vyčlenenie bezpečnostných pravidiel do samostatných logických entít dovoľuje väčšiu

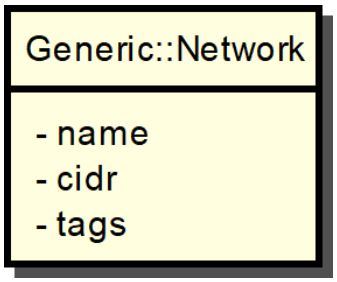
variabilitu konfigurácií ako priame integrovanie pravidiel do bezpečnostnej skupiny. Z toho dôvodu navrhujeme vyčleniť skupiny bezpečnostných pravidiel ako samostatné logické entity, čo je zobrazené aj na obrázku Obrázok 7. Bezpečnostná skupina má podobne ako v systémoch AWS a OpenStack parametre meno, popis, tagy a vlastné bezpečnostné pravidlá. Tie sú rozdelené do dvoch entít, samostatne pre prichádzajúce a samostatne pre odchádzajúce pakety. Skupiny pravidiel majú totožnú syntax, nachádza sa v nich meno, typ transportného protokolu neseného v pakete, dve hranice intervalu transportných portov a IP prefix v CIDR formáte. Diagram závislostí bezpečnostnej skupiny sa nachádza na obrázku nižšie, kde môžeme vidieť že skupina nemá žiadne externé závislosti a môže existovať ako úplne samostatná entita.



Obrázok 7: Bezpečnostná skupina vo všeobecnom modeli, zdroj autor

4.3.5 Sieť

Vo všeobecnom modeli sme sa snažili skĺbiť atribúty zo všetkých systémov tak, aby boli pokryté najčastejšie používané parametre v každom zo systémov V tomto konkrétnom prípade je jediný povinný parameter CIDR v systéme AWS. Na nasledujúcom obrázku sa nachádza model závislostí siete vo všeobecnom modeli.

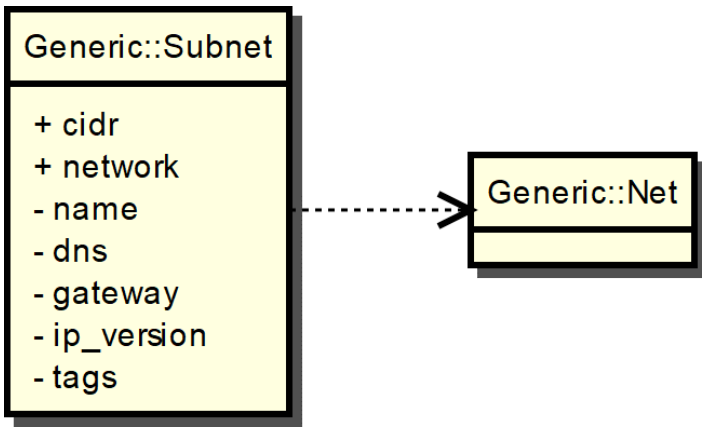


Obrázok 8: Sieť vo všeobecnom modeli, zdroj autor

4.3.6 Podsieť

Vo všeobecnom modeli sme sa opäť snažili navrhnúť entity tak, aby bola možná transformácia z/do ľubovoľného CC systému. Na základe vyššie uvedeného môžeme povedať, že systém AWS je podmnožinou systému OpenStack, čo sa parametrov týka. Z toho dôvodu sme zvolili parametre v podstate zhodné so systémom OpenStack.

Na obrázku nižšie môžeme vidieť diagram závislostí podsiete vo všeobecnom modeli. Znovu je tu jediná entita, na ktorej podsieť závisí – entita siete. Tá opäť musí existovať , aby mohla združovať podsiete.



Obrázok 9: Podsieť vo všeobecnom modeli, zdroj autor

4.4 Transformačné pravidlá

4.4.1 Parametre prostredia

V nasledujúcej tabuľke sa nachádzajú transformácie pre parametre prostredia. Môžeme vidieť, že systém AWS môžeme priamočiaro transformovať do všeobecného modelu. Systém OpenStack je trochu zložitejší, pretože obmedzenia ako dĺžka reťazca či interval číselných hodnôt sa nachádzajú zapuzdrené v časti *constraints*. Systém Azure je rovnako ako systém AWS priamo transformovateľný do všeobecného modelu. Výnimku tvorí parameter *description*, ktorý je o jednu úroveň nižšie v časti *metadata*. Parameter *allowed_pattern*, ktorý definuje regulárnym výrazom použiteľné hodnoty, nie je v systéme Azure implementovaný.

Parametre *length* a *range* v časti *constraints* obsahujú dve hodnoty – *min* a *max*. Tieto hodnoty tak vieme priamo transformovať do hodnôt *min_length* a *max_length*, resp. *min_value* a *max_value*. Hodnoty *allowed_values* a *allowed_pattern* sú transformovateľné priamo napriek tomu, že v CC systéme OpenStack sú o jednu úroveň nižšie – v rámci parametra *constraints*.

Tabuľka 1: Transformačné pravidlá: Parametre prostredia, zdroj autor

Všeobecný model	AWS	OpenStack	Azure
type	Type	type	type
description	Description	description	metadata -

			description
default	Default	default	defaultValue
min_length	MinLength	constraints - length	minLength
max_length	MaxLength	constraints - length	maxLength
min_value	MinValue	constraints - range	minValue
max_value	MaxValue	constraints - range	maxValue
allowed_values	AllowedValues	constraints - allowed_values	allowedValues
allowed_pattern	AllowedPattern	constraints - allowed_pattern	-

V tabuľke Tabuľka 2 sa nachádzajú transformácie špeciálne pre parameter *type*. Tento parameter je dôležitý, pretože na jeho základe sa môžu niektoré hodnoty ignorovať. Ak bude tento parameter obsahovať hodnotu *number*, hodnoty obmedzujúce dĺžku reťazca a regulárny výraz obmedzujúci znaky reťazca budú ignorované ako v systéme AWS, tak v systéme OpenStack.

Môžeme vidieť, že všetky hodnoty sú priamo transformovateľné. Jedinú výnimku tvorí hodnota *boolean*, ktorú systém AWS vôbec nepodporuje. Domnievame sa, že pri tejto hodnote nie je možné plnohodnotne automatizovať transformáciu, a je potrebné informovať administrátora, aby vykonal manuálnu kontrolu a prerobenie popisného skriptu.

Tabuľka 2: Transformačné pravidlá: Parametre prostredia - typ, zdroj autor

Všeobecný model	AWS	OpenStack	Azure
string	String	string	string
number	Number	number	int
list	List	comma_delimited_list	array
boolean	-	boolean	bool

4.4.2 Virtuálny server

Tabuľka 3 znázorňuje transformačné pravidlá virtuálnych strojov medzi CC systémami OpenStack, AWS, Azure a všeobecným modelom. CC systém AWS neobsahuje názov VM, ale ako sme avizovali vyššie, meno je možné nastaviť v parametri *Tags*.

Ďalší rozdiel môžeme vidieť v šiestom parametri, kde všeobecný model a systém OpenStack majú uvedenú sieť, systém AWS má podsieť a v systéme Azure je možné definovať priamo rozhranie, ktoré je možné následne pripojiť do konkrétnej siete, alebo podsiete. Ak v systéme OpenStack pripojíme VM do celej siete, znamená to pripojenie do všetkých jej podsietí. Taktiež je možné pripojiť VM len do jednej konkrétnej podsiete. Môžeme tak hovoriť o priamočiarom transformovaní, kde pripojenie do celej siete zameníme za niekoľkonásobné pripojenie do všetkých podsietí.

Ako môžeme vidieť v tabuľke nižšie, CC systém Azure má oproti systémom OpenStack a AWS značne zložitejšiu štruktúru. Napriek tomu, sú tieto parametre významovo identické, takže transformácia do všeobecného modelu je priamočiara, je len potrebné dbať na správne umiestnenie parametrov v štruktúre skriptu.

Tabuľka 3: Transformačné pravidlá: Virtuálny server, zdroj autor

Všeobecný model	AWS	OpenStack	Azure
name	-	name	name
properties - image	properties - ImageId	properties - image	properties - StorageProfile - imageReference - id
properties - instance_type	properties - InstanceType	properties - flavor	properties - hardwareProfile - vmSize
properties - security_groups	properties - SecurityGroupIds	properties - security_groups	
properties - availability_zone	properties - AvailabilityZone	properties - availability_zone	zones
properties - network	properties - SubnetId	properties - networks	properties - networkProfile - networkInterfaces - id
properties - ssh_key	properties - KeyName	properties - key_name	properties - OSProfile - linuxConfiguration - ssh - publicKeys - keyData
properties - user_data	properties - UserData	properties - user_data	properties - OSProfile - CustomData
properties - tags	properties - Tags	properties - tags	tags

4.4.3 SSH kľúč

V časti 4.2.1 môžeme vidieť, že systémy AWS a Azure úplne absentujú akékoľvek pravidlá. To je z dôvodu, že kľúč nie je možné vytvoriť v rámci skriptu, ale len cez webové rozhranie, či priamy import hodnoty kľúča. Keďže všeobecný model bol inšpirovaný systémom OpenStack, v zásade nie je nutné robiť žiadne transformácie medzi týmito modelmi. Pretože kľúčové slová sú rovnaké, stačí jednoducho skopírovať hodnoty.

Tabuľka 4: Transformačné pravidlá: SSH kľúč, zdroj autor

Všeobecný model	AWS	OpenStack	Azure
properties - name	-	properties - name	-
properties - save_private_key	-	properties - save_private_key	-
properties - public_key	-	properties - public_key	-
properties - user	-	properties - user	-

4.4.4 Bezpečnostná skupina

Na základe analýzy sme zistili rozdiel medzi systémami AWS, Azure a OpenStack. Systém OpenStack má pravidlá integrované do jednej logickej entity, zatiaľ čo AWS a Azure tieto pravidlá vyčlenili do samostatných logických entít. V tabuľke Tabuľka 5 môžeme vidieť, že pravidlá nemajú spoločný prienik vo všeobecnom modeli. Systém OpenStack používa jeden parameter *rules*, zatiaľ čo všeobecný model, systém AWS a Azure používajú oddelené parametre pre vstupné a výstupne pravidlá. AWS navyše rozdeľuje tieto entity pre vstupné a výstupné pravidlá, zatiaľ čo Azure používa jednu spoločnú logickú entitu pre obidva smery. Ako bolo spomenuté vyššie, vo všeobecnom modeli sme navrhli dva nezávislé parametre, podobne ako v systéme AWS. Z toho dôvodu vidíme priamočiare transformovanie parametrov medzi systémom AWS a všeobecným modelom. Pri transformácii medzi systémom OpenStack a všeobecným modelom bude potrebné vytvárať nové logické entity, resp. združiť viac entít do jednej. Mapovanie do systému Azure je pomerne priamočiare, je potrebné len rozdeliť pravidlá na vstupné a výstupné entity podľa parametra *direction*, ktorý sa nachádza v každej entite pravidiel.

Posledný parameter *tags* zatiaľ nie je v systéme OpenStack implementovaný. Ak by sa vývojári rozhodli implementovať tento parameter v nasledujúcich verziách, bude potrebné doplniť aj túto transformáciu, čo ale nepredstavuje výraznejší problém.

Tabuľka 5: Transformačné pravidlá: Bezpečnostná skupina, zdroj autor

Všeobecný model	AWS	OpenStack	Azure
properties - name	properties - GroupName	properties - name	name
properties - description	properties - GroupDescription	properties - description	-
-	-	properties - rules	securityRules
properties - ingress_rules	properties - SecurityGroupIngress	-	-
properties - egress_rules	properties - SecurityGroupEgress	-	-
properties - tags	properties - Tags	-	tags

Nasledujúca Tabuľka 6 je združenou tabuľkou. Pre systém AWS a pre bezpečnostnú skupinu v nej vidíme transformácie logických entít pravidiel pre prichádzajúce a odchádzajúce pakety, zatiaľ čo pre systém OpenStack to sú pravidlá integrované v tele bezpečnostnej skupiny. No ako môžeme vidieť v tejto tabuľke, transformácie týchto parametrov sú pomerne priamočiare. Rozdiel nájdeme v parametri mena skupiny, čo systém OpenStack nepotrebuje, pretože sa nachádza priamo v tele bezpečnostnej skupiny. Na druhej strane vidíme, že systém OpenStack má parameter smeru v ktorom sa budú pakety porovnávať s pravidlom. Ten ale nie je potrebný vo všeobecnom modeli a v systéme AWS, pretože tieto skupiny sa priamo nasadzujú v požadovanom smere toku.

Tabuľka 6: Transformačné pravidlá: Bezpečnostná skupina - pravidlá, zdroj autor

Všeobecný model	AWS	OpenStack	Azure
name	GroupName	-	name
description	-	-	description

-	-	direction	direction
protocol	IpProtocol	protocol	protocol
from_port	FromPort	port_range_min	sourcePortPrefix
to_port	ToPort	port_range_max	-
cidr	CidrIp	remote_ip_prefix	sourceAddressPrefix

4.4.5 Sieť

V nasledujúcej tabuľke sa nachádzajú transformácie jednotlivých atribútov siete do všeobecného modelu. Ak systém nepoužíva niektorý z parametrov, je parameter označený pomlčkou.

Tabuľka 7: Transformačné pravidlá: Sieť, zdroj autor

Všeobecný model	AWS	OpenStack	Azure
properties - name	properties - VpcId	properties - name	name
properties - subnets	-	-	subnets
properties - cidr	properties - CidrBlock	-	-
properties - tags	properties - Tags	properties - tags	tags

4.4.6 Podsieť

Z transformačných pravidiel podsiete v časti 4.2 je viditeľné, že v systémoch AWS a Azure je podsieť omnoho menej konfigurovateľná ako v systéme OpenStack. Čiastočne to ale vyplýva z povahy systémov AWS a Azure, kde poskytovateľ vopred určil niektoré hodnoty a používateľ ich nemôže meniť, ako napríklad predvolená brána má v oboch systémoch vždy prvú možnú adresu z podsiete. Taktiež nie je možné určiť verziu IP protokolu, pretože v súčasnosti AWS aj Azure ponúkajú len IPv4 konektivitu.

Tabuľka 8: Transformačné pravidlá: Podsieť, zdroj autor

Všeobecný model	AWS	OpenStack	Azure
properties - name	-	properties - name	name
properties - network	properties - VpcId	properties - network	-
properties - cidr	properties - CidrBlock	properties - cidr	addressPrefix
properties - dns	-	properties - dns_nameservers	-
properties - gateway	-	properties - gateway_ip	-
properties - ip_version	-	properties - ip_version	-
properties - tags	properties - Tags	properties - tags	tags

4.5 Implementácia a overenie

Pri implementácii prichádzalo do úvahy niekoľko možností implementácie. Z rôznych programovacích jazykov sme sa rozhodli pre jazyk Python, konkrétne pre verziu Python3. Je to z toho dôvodu, že tento jazyk je dostatočne vysokoúrovňový, interpretovaný, a tým pádom platformovo nezávislý.

Snažili sme sa dodržať niektoré konvencie, ktoré sa používajú pri písaní zdrojových kódov. Dôvodom bola snaha o sprehľadnenie kódu. Názvy tried sa začínajú veľkým, názvy metód malým začiatočným písmenom. Atribúty začínajú písmenom „a“, parametre začínajú písmenami „pa“. Taktiež názvy premenných sú s malým začiatočným písmenom a používajú tzv. „ťaviu notáciu“. Tá spočíva v tom, že ak je názov premennej viacslovný, každé slovo začína veľkým písmenom. Ako príklad uvedieme pramennú „pocet novych objektov“, ktorá by sa v danej notácii zapísala ako „pocetNovychObjektov“. Keďže veríme, že aplikácia bude aj naďalej vyvíjaná a vylepšovaná nielen na Slovensku, zdrojové kódy vrátane komentárov sú písané v anglickom jazyku. Kompletné zdrojové kódy sa nachádzajú v prílohe na CD. Taktiež sú voľne dostupné na webovom úložisku BitBucket, čo je verejne dostupný repozitár.

Aplikácia je navrhnutá modulárne. Znamená to, že je vytvorené jadro aplikácie, do ktorého sú následne jednotlivé moduly CC IaaS implementácií pridávané. Hlavnou triedou je „CloudMigration“, z ktorej sú volané funkcie ostatných tried. Predstavuje akési rozhranie, cez ktoré je možné ovládať aplikáciu.

Ďalšou triedou je „MainWindow“, ktorá obsahuje grafické rozhranie k aplikácii a pomocou ktorej používateľ interaguje s aplikáciou. Z tejto triedy sú následne volané funkcie triedy „CloudMigration“. Z tohto prístupu je zrejmé, že táto trieda môže byť nahradená akoukoľvek inou triedou, ktorá by ovládala aplikáciu. V budúcnosti by tak mohlo pribudnúť ovládania aplikácie cez napríklad príkazový riadok (CLI – Command Line Interface), alebo vytvorením API (Application Program Interface).

Trieda „Loader“ slúži na inicializovanie modulov a transformačných pravidiel, ktoré sa budú používať. Táto trieda taktiež inicializuje triedy, ktoré tvoria moduly a obsluhujú jednotlivé implementácie CC IaaS riešení (Generic, OpenStack, AWS a Azure). Každá implementácia CC IaaS riešenia má vlastnú triedu, ktorá obsluhuje transformácie z/do všeobecného modelu.

Priečinok „Mapper“ obsahuje textové súbory s definíciami priamych mapovaní zdrojov. Ku každému zdroju, ako napríklad podsieti, je vytvorený súbor, kde sú mapovania parametrov v rôznych implementáciách CC systémov. Ak niektorý systém neobsahuje parameter, je nahradený pomlčkou. Pre názornosť uvedieme príklad časť súboru pre podsieť.

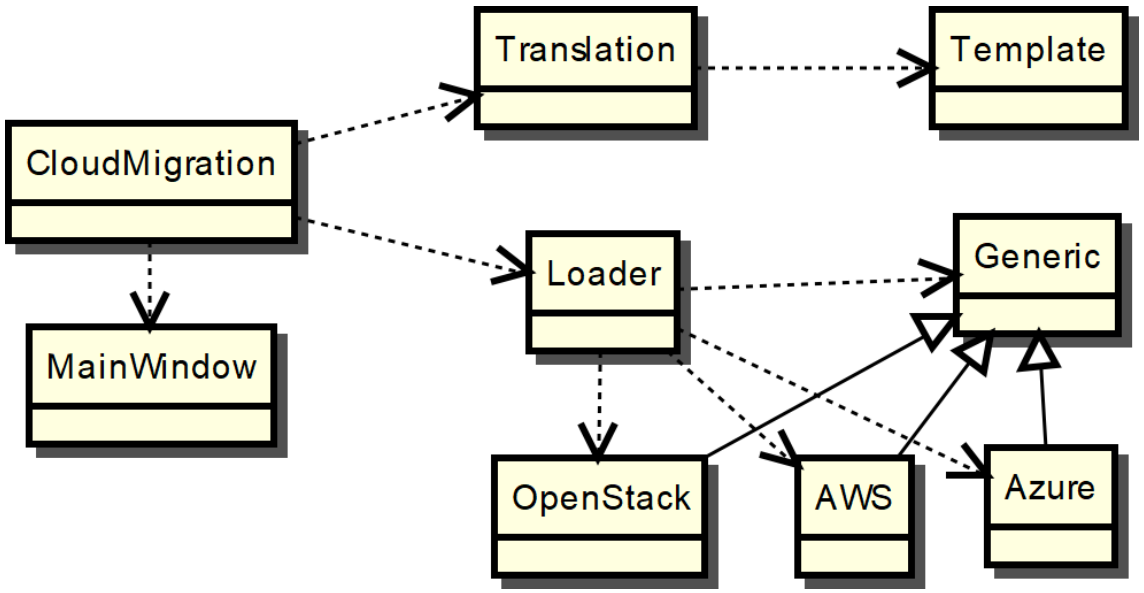
```
Generic      name   network      cidr
OpenStack    name   network      cidr
AWS          -      VpcId        CidrBlock
Azure        name   -            address-prefix
```

Ako môžeme vidieť, systém AWS nemá definované meno pre podsieť, a systém Azure nedefinuje sieť, do ktorej daná podsieť patrí. Keby sme potrebovali transformovať IP prefix zo všeobecného modelu (v ukážke „cidr“) do systému Azure, názov parametra by sme zmenili za „address-prefix“. Tieto mapovania zodpovedajú mapovacím tabuľkám, uvedeným v časti 4.4.

Trieda „Template“ je zodpovedná za načítanie schém, ktoré sa nachádzajú v priečinku „Schemas“. Každý modul má vlastnú schému parametrov, ktoré sa načítajú a následne podľa týchto schém prebieha preklad. Schémy sú písané v jazyku JSON. V schémach sú definované vlastnosti jednotlivých parametrov, ako ich stručný popis a ich typ. Typ môže momentálne nadobúdať 3 hodnoty – „value“, „list“ a „special“. Pokiaľ je hodnota „value“, transformácia prebehne priamo, to znamená hodnota sa jednoducho skopíruje. Podobne je to pri hodnote „list“, ktorá definuje, že hodnota je v skutočnosti zoznam hodnôt. Posledným typom je

„special“, ktorá hovorí, že nie je možné urobiť priame mapovanie, ale je nutné špeciálne ošetriť mapovanie v kóde. Ako príklad môžeme uviesť špeciálne hodnoty, napríklad verziu skriptu, ktorá je špecifická pre konkrétne CC platformy. V tomto prípade sa vo výslednom skripte zobrazí varovanie, že administrátor by mal manuálne upraviť a skontrolovať parameter.

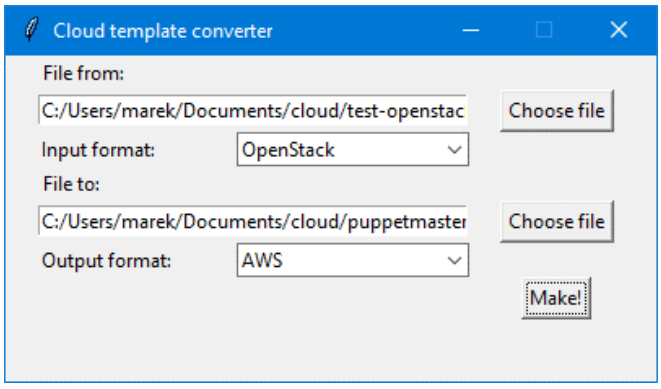
UML diagram tried sa nachádza na obrázku Obrázok 10.



Obrázok 10: UML diagram tried

Ak v budúcnosti pribudne modul implementujúci transformácie popisu iného CC prostredia do všeobecného modelu, pre úspešné pripojenie k aplikácii potrebuje spĺňať tri podmienky. Prvou je doplnenie transformácií parametrov do všetkých zdrojov v priečinku „Mapper“. Druhou je vytvorenie schémy konkrétnych parametrov v priečinku „Schemas“ a treťou podmienkou je vytvorenie triedy obsluhujúcej špeciálne prípady transformácií, ktoré nie je možné vykonať priamo.

Na obrázku Obrázok 11 sa nachádza grafické rozhranie aplikácie. Používateľ si môže definovať zdrojový a cieľový súbor, a taktiež CC prostredie, z/do ktorého sa udeje transformácia. Momentálne má na výber všeobecný model, CC systémy OpenStack, Amazon AWS a Microsoft Azure. Tieto systémy si vyberá z rozbaľovacieho menu, ktoré sa nachádza v okne aplikácie. Keďže dúfame, že naša aplikácia bude osožná administrátorom nielen na Slovensku, grafické rozhranie je vytvorené v anglickom jazyku.



Obrázok 11: Grafické rozhranie programu, zdroj autor

4.5.1 Overenie modelu

Po implementácii je potrebné vykonať verifikáciu modelu a implementácie. V našom prípade sme uskutočnili transformáciu popisov z/do všeobecného modelu. Pre syntaktickú kontrolu transformovaných popisov sme použili reálne implementácie CC systémov, ktoré nám garantujú správnosť transformácie. Ako referenčnú topológiu sme použili jednu virtuálnu inštanciu servera v sieti za logickým smerovačom, ktorá bola znázornená aj na obrázku Obrázok 3.

Vykonali sme transformácie skriptov zo všetkých troch modelov do všeobecného modelu, a následne zo všeobecného modelu spätne do konkrétneho modelu. Ako overenie prikladáme skripty popisujúce našu testovaciu topológiu. Všetky skripty, ktoré popisujeme v nasledujúcej časti sa nachádzajú v prílohe na CD v priečinku „Skripty“ a sú napísané v značkovacom jazyku YAML.

Prvým krokom bolo vytvorenie skriptu s testovacou topológiou. V našom prípade sme ako prvý systém pre transformáciu použili OpenStack, pretože s ním máme najviac skúseností. Skript pre systém OpenStack má názov „OpenStack.yml“. Po syntaktickom overení skriptu v systéme OpenStack nasledoval druhý krok, transformácia tohto skriptu do všeobecného modelu. Tento pretransformovaný skript sa nachádza v prílohe na CD s názvom „Generic-z-OpenStack.yml“. Tretím krokom bola transformácia skriptu vo všeobecnom formáte spätne do formátu pre systém OpenStack. Ten sa taktiež nachádza v prílohách s názvom „OpenStack-z-Generic.yml“. Tento súbor sme následne úspešne syntakticky validovali v systéme OpenStack, čím sme potvrdili správnosť a funkčnosť našej implementácie všeobecného modelu.

Rovnakou trojkrokovou metódou sme overili aj syntaktickú správnosť prekladu pre systémy Amazon AWS a Microsoft Azure. V prílohách na CD sa v priečinku „Skripty“ nachádzajú popisné skripty prostredí v systémoch AWS a Azure. Taktiež sa tam nachádzajú tieto skripty pretransformované do všeobecného modelu, a skripty, ktoré sú do týchto implementácií pretransformované spätne zo všeobecného modelu. V obidvoch prípadoch, pri AWS aj Azure, sme použili reálnu implementáciu CC prostredia pre overenie správnosti prekladu daného skriptu. Ako bolo spomenuté vyššie, v obidvoch prípadoch sme transformovali rovnakú testovaciu topológiu, ktorá pozostávala z jedného virtuálneho servera, ktorý sa nachádza vo virtuálnej sieti za logickým smerovačom.

Na základe testov syntaktickej správnosti na reálnych implementáciách CC IaaS systémov si dovolíme konštatovať, že naša implementácia všeobecného modelu, ako aj transformačných pravidiel je funkčná a pripravená na používanie. Pre reálne použitie je však potrebné budúce rozšírenie všeobecného modelu o ďalšie entity.

4.6 Diskusia výsledkov

V súčasnosti sú dostupné pre nasadenie CC s IaaS rovnocenné riešenia, kde ich konkrétny výber môže byť podmienený rôznymi faktormi (jednoduchosť používania, rozsiahlosť dokumentácie, jednoduchosť nasadenia v prípade privátneho CC, ...), a jeden z nich môže byť aktuálna cena. Tá sa však môže mnohokrát dynamicky vyvíjať a pri dostupnosti alternatívnych riešení s lepšou cenou pri rovnakých ponúkaných službách pripadajú do úvahy aj aspekty prechodu od jedného poskytovateľa CC k inému. Z tohto pohľadu vystupujú do popredia otázky prenositeľnosti svojej IaaS služby od poskytovateľa k poskytovateľovi a potencionálne s tým už uvádzaný problém *vendor lock-in*.

Navrhnutý spôsob transformácií umožní systémovo riešiť problematiku prenosu IaaS služby medzi rôznymi IaaS prostrediami, a je navrhnutý ako všeobecný a do budúcnosti otvorený pre rozširovanie. Vypracovanie špecifikácie uľahčí používateľom aspekt migrácie služby a prispeje k väčšej otvorenosti prostredia CC služieb. Vzhľadom na komplexnosť problematiky sa práca zameriava na iba na IaaS službu, vzniká tu však predpoklad, že po úspešnom overení je využiteľnosť riešenia aj pre iné CC služby. Práca je zároveň príspevkom k oblasti štandardizácie portability.

Veríme, že táto práca rieši otázku problému, ktorý je vo svete aktuálny. Toto tvrdenie podporuje aj skutočnosť, že v decembri roku 2017 spoločnosť ISO vydala štandard 19941 [55], ktorý definuje interoperabilitu a portabilitu z pohľadu poskytovateľov a používateľov CC prostredí.

Záver

Hlavným cieľom dizertačnej práce bolo navrhnutie a otestovanie všeobecného modelu pre skripty popisujúce CC prostredia. Prvým krokom bola analýza existujúcich CC prostredí. Z dôvodu utvorenia komplexnejšieho pohľadu sme analyzovali zástupcov privátnych, ale aj verejných CC IaaS prostredí. Analýza spočívala v pochopení daného riešenia v zmysle logickej stavby topológie. Následným krokom bolo zoznámenie sa s konkrétnymi logickými entitami a získanie zručností s ich praktickým používaním v popisných skriptoch.

Následne sme navrhli model, v ktorom sme sa snažili nájsť prienik medzi analyzovanými CC prostrediami, ktorý by bol čo najjednoduchší na implementáciu, a taktiež, aby čo najpriamejšie transformoval parametre daných CC prostredí. Zároveň bolo našou snahou, aby logická stavba a členenie navrhovaného modelu boli čo najviac podobné pôvodným modelom z toho dôvodu, aby bol administrátorom čo najfamiliárnejší.

Posledným krokom bola implementácia modelu v programovacom jazyku a následné praktické overenie. Neoddeliteľnou súčasťou bolo otestovanie pretransformovaných skriptov v reálnych implementáciách CC prostredí, čím bola zaručená syntaktická bezchybnosť transformácií.

Za prínos dizertačnej práce považujeme vytvorenie všeobecného modelu popisného skriptu CC IaaS prostredia, jeho implementáciu a praktické overenie. Taktiež je to prehľad aktuálneho stavu poznania CC IaaS služby doma a v zahraničí.

Veríme, že navrhnutý model, ako aj jeho implementácia budú používané pri reálnych transformáciách popisných skriptov a uľahčia prácu administrátorom. Model, ako aj zdrojové kódy implementácie v programovacom jazyku Python3 sú verejne dostupné a dúfame, že tento projekt bude aj naďalej rozširovaný a zveľaďovaný.

Možnosti ďalšieho smerovania výskumu a vývoja v CC oblasti vidíme v stanovení pravidiel transformácií, prípadne štandardizácii niektorého z popisných jazykov ako všeobecne platného a uznávaného, ktorý môže byť použitý v akomkoľvek CC prostredí. Podľa nášho názoru môže prísnejšia štandardizácia pomôcť zjednotiť trh v oblasti CC, čím donúti poskytovateľov k ponúkaniu kvalitnejších služieb, čo bude mať pozitívne dôsledky pre prevádzku používateľských aplikácií.

Zoznam použitej literatúry

[1] G. A. A. Santana, CCNA Cloud, Cisco Press, 2016, p. 609.

[2] European Commission, „Cloud Select Industry Group on Service Level Agreements," [Online]. Available: https://ec.europa.eu/digital-single-market/en/cloud-select-industry-group-service-level-agreements. [Cit. December 2017].

[3] European Commission, „Cloud Select Industry Group on Code of Conduct," [Online]. Available: https://ec.europa.eu/digital-single-market/en/cloud-select-industry-group-code-conduct. [Cit. December 2017].

[4] European Telecommunications Standards Institute, „Cloud Standards Coordination," [Online]. Available: http://csc.etsi.org/. [Cit. December 2017].

[5] European Commission, „Expert Group on Cloud Computing Contracts," [Online]. Available: http://ec.europa.eu/justice/contract/cloud-computing/expert-group/index_en.htm. [Cit. December 2017].

[6] ITU-T, „Y.3500 Overview and vocabulary," August 2014. [Online]. Available: https://www.itu.int/rec/T-REC-Y.3500-201408-I. [Cit. Január 2018].

[7] F. Liu a et.al., „NIST Cloud Computing Reference Architecture, SP-500-292," 8 September 2011. [Online]. Available: https://www.nist.gov/publications/nist-cloud-computing-reference-architecture. [Cit. Február 2017].

[8] Roadmap Working Group, „NIST Cloud Computing Standards Roadmap, SP 500-291 version 2," 2013. [Online]. Available: http://www.cloudwatchhub.eu/sites/default/files/NIST_Cloud-Standards-Roadmap_v2.pdf. [Cit. Január 2017].

[9] „What is a Private Cloud?," [Online]. Available: http://www.interoute.com/cloud-article/what-private-cloud. [Cit. Január 2017].

[10] TechTarget, „Cloud Bursting," Apríl 2017. [Online]. Available: http://searchcloudcomputing.techtarget.com/definition/cloud-bursting. [Cit. Január 2018].

[11] „OpenStack Project," [Online]. Available: https://www.openstack.org/. [Cit. marec 2017].

[12] Amazon AWS, „What Is AWS GovCloud (US)?," [Online]. Available: http://docs.aws.amazon.com/govcloud-us/latest/UserGuide/whatis.html. [Cit. December 2017].

[13] RightScale, „State of the cloud report," 2017. [Online]. Available: http://assets.rightscale.com/uploads/pdfs/RightScale-2017-State-of-the-Cloud-Report.pdf. [Cit. Marec 2017].

[14] „What is a Public Cloud?," [Online]. Available: http://www.interoute.com/cloud-article/what-public-cloud. [Cit. Január 2017].

[15] „What is a Hybrid Cloud?," [Online]. Available: http://www.interoute.com/cloud-article/what-hybrid-cloud. [Cit. Január 2017].

[16] M. Boniface a et.al, „Platform-as-a-Service Architecture for Real-time Quality of Service Management in Clouds," rev. *Fifth International Conference on Internet and Web Applications and Services*, 2010.

[17] L. Youseff, M. Butrico a D. Da Silva, „Toward a unified ontology of cloud computing,"

Grid Computing Environments (GCE) Workshop, pp. 1-10, 2008.

[18] A. Benlian a T. Hess, „Opportunities and risks of software-as-a-service: Findings from a survey of IT executives," *Decision Support Systems,* zv. 52, %1. vyd.1, pp. 232-246, 2011.

[19] S. Hopko, Problematika Cloud computing a jeho využitia v riešení pre potreby KIS, Žilina, 2015.

[20] R. L. Krutz a R. D. Vines, Cloud Security: A Comprehensive Guide to Secure Cloud Computing, Wiley Publishing, Inc., 2010.

[21] R. Dimpi a R. R. K., „A Comparative Study of SaaS, PaaS and IaaS in Cloud Computing," *International Journal of Advanced Research in Computer Science and Software Engineering,* zv. 4, %1. vyd.6, pp. 458-461, Jún 2014.

[22] Roadmap Working Group, „NIST Cloud Computing Standards Roadmap," 2013. [Online]. Available: http://www.cloudwatchhub.eu/sites/default/files/NIST_Cloud-Standards-Roadmap_v2.pdf. [Cit. Január 2017].

[23] ITU-T, „Y.3502 Reference architecture," August 2014. [Online]. Available: https://www.itu.int/rec/T-REC-Y.3502-201408-I/en. [Cit. Decemer 2017].

[24] Wikioedia, „Interoperability," Január 2018. [Online]. Available: https://en.wikipedia.org/wiki/Interoperability. [Cit. Január 2018].

[25] Techpedia, „Interoperability," Január 2018. [Online]. Available: https://www.techopedia.com/definition/631/interoperability. [Cit. Január 2018].

[26] D. Petcu, „Portability and interoperability between clouds: challenges and case study," rev. *ServiceWave'11 Proceedings of the 4th European conference on Towards a service-based internet*, Heidelberg, 2011.

[27] Cloud Standards Customer Council, Interoperability and Portability for Cloud Computing: A Guide, 2014, p. 31.

[28] A. V. Parameswaran a A. Chaddha, „Cloud Interoperability and Standardization," *SETLabs Briefings,* zv. 7, %1. vyd.7, pp. 19-26, 2009.

[29] M. Kostoska a et.al, „Cloud Computing Interoperability Approaches – Possibilities and Challenges," rev. *Proceedings of 5th Balkan Conf. in Informatics*, Novi Sad, Srbsko, 2012.

[30] IEEE, „Cloud Profiles Working Group," [Online]. Available: http://standards.ieee.org/develop/wg/CPWG-2301.html. [Cit. November 2017].

[31] IEEE, „Standard for Intercloud Interoperability and Federation," [Online]. Available: https://standards.ieee.org/develop/project/2302.html. [Cit. November 2017].

[32] OASIS, „TOSCA," [Online]. Available: https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=tosca. [Cit. November 2017].

[33] K. Bhavya, K. Yamini a V. Sreenivas, „Cloud Services Portability for secure migration," *International Journal of Computer Trends and Technology (IJCTT),* zv. 4, %1. vyd.4, pp. 546-549, 2013.

[34] I. Bojanova, „Cloud Interoperability and Portability II,“ 25 júl 2013. [Online]. Available: https://www.computer.org/web/irena-bojanova/content?g=5970564&type=blogpost&urlTitle=cloud-interoperability-and-portability-ii. [Cit. Október 2017].

[35] The Open Group, „Cloud Portability and Interoperability,“ [Online]. Available: http://www.opengroup.org/cloud/cloud_iop/p4.htm. [Cit. Október 2017].

[36] „PC Encyclopedia,“ [Online]. Available: https://www.pcmag.com/encyclopedia/term/62863/cloud-platform. [Cit. Január 2018].

[37] RightScale, „State of the cloud report,“ 2015. [Online]. Available: http://assets.rightscale.com/uploads/pdfs/RightScale-2015-State-of-the-Cloud-Report.pdf. [Cit. Marec 2017].

[38] RightScale, „State of the cloud report,“ 2016. [Online]. Available: http://assets.rightscale.com/uploads/pdfs/RightScale-2016-State-of-the-Cloud-Report.pdf. [Cit. Marec 2017].

[39] OMG, „Model Driven Architecture – A Technical Perspective,“ OMG, 2001.

[40] M. Kardoš, Transformácia CIM do PIM v modelom riadenej architektúre (MDA), Žilina, 2011.

[41] S. M. Drazen Brdjanin, „Model Driven Techniques for Data Model Synthesis,“ *Electronics,* zv. VOL. 17, %1. vyd.NO. 2, pp. 130-136, 2013.

[42] A. G. Kleppe, J. B. Warmer a W. Bast, MDA explained : the model driven architecture : practice and promise, Addison-Wesley, 2003.

[43] OMG, „CommonWarehouseMetamodel (CWM) Specificatio,“ OMG, 2001.

[44] S. Gyapay a D. Varró, „Automated Algorithm Generation for Visual Control Structures,“ December 2000. [Online]. Available: http://static.inf.mit.bme.hu/pub/TR-12-2000.pdf. [Cit. Október 2017].

[45] D. Varró, G. Varró a P. Apndrás, „Designing the automatic transformation of visual languages,“ *Science of Computer Programming,* zv. 44, %1. vyd.2, pp. 205-227, 2002.

[46] W3C, „XSL Transformations (XSLT) Version 3.0,“ W3C, 8 Jún 2017. [Online]. Available: https://www.w3.org/TR/xslt/. [Cit. November 2017].

[47] „Amazon Web Services,“ [Online]. Available: https://aws.amazon.com/about-aws/. [Cit. Marec 2017].

[48] „Browse All OpenStack Projects,“ [Online]. Available: https://www.openstack.org/software/project-navigator. [Cit. marec 2017].

[49] [Online]. Available: https://www.cloudave.com/wp-content/uploads/2015/02/slide1.jpg. [Cit. December 2017].

[50] „Microsoft Azure,“ [Online]. Available: https://azure.microsoft.com/en-us/overview/what-is-azure/. [Cit. marec 2017].

[51] „Azure Stack Development Kit,“ [Online]. Available: https://azure.microsoft.com/en-us/blog/microsoft-azure-stack-is-ready-to-order-now/. [Cit. 2017].

[52] M. Corporation, „Microsoft Azure Stack,“ [Online]. Available: https://azure.microsoft.com/en-us/blog/microsoft-azure-stack-is-ready-to-order-now/. [Cit. 2017].

[53] VMware, Inc., „vSphere and vSphere with Operations Management,“ [Online]. Available: http://www.vmware.com/products/vsphere.html. [Cit. marec 2017].

[54] VMware, Inc., „vRealize Automation,“ [Online]. Available: http://www.vmware.com/products/vrealize-automation.html. [Cit. marec 2017].

[55] ISO, „ISO/IEC 19941,“ December 2017. [Online]. Available: https://www.iso.org/standard/66639.html. [Cit. Január 2018].

[56] M. A. Bokhari, Q. M. Shallal a Y. K. Tamandani, „Cloud Computing Service Models: A Comparative Study,“ *3rd International Conference on Computing for Sustainable Global Development (INDIACom),* pp. 890-895, 16-18 Marec 2016.

[57] Z. Bizoňová, Model Driven E-learning Platform Integration, Žilina, 2008, p. 216.

[58] T. Mens a P. Van Gorp, „A Taxonomy of Model Transformation,“ *Electronic Notes in Theoretical Computer Science,* zv. 152, pp. 125-142, 2006.

[59] „Interoperability and Portability for Cloud Computing: A Guide,“ November 2014. [Online]. Available: http://www.cloud-council.org/deliverables/CSCC-Interoperability-and-Portability-for-Cloud-Computing-A-Guide.pdf. [Cit. Január 2017].

[60] L. Mohan a e. al., „A Comparative Study on SaaS, PaaS and IaaS Cloud Delivery Models in Cloud Computing,“ *International Journal on Emerging Technologies,* pp. 158-160, 2017.

Zoznam vlastných publikácií

1. Marek Moravčík et al.: Clouds in educational process, in ICETA 2015 : 13th IEEE international conference on Emerging eLearning technologies and applications, November 26-27, 2015, Starý Smokovec, Slovakia 2015, ISBN 978-1-4673-8534-3, s 269-275.
2. Jakub Hrabovsky et al.: Usability of the SIP protocol within smart home solutions, in Communications: scientific letters of the University of Žilina, ISSN 1335-4205, Vol. 18, no. 1A, 2016, s. 4-12.
3. Marek Moravčík et al.: Survey of real-time multimedia security mechanisms, in ICETA 2016 14th IEEE international conference on Emerging eLearning technologies and applications, November 24-25, 2016, Starý Smokovec, The High Tatras, Slovakia, 2016, ISBN 978-1-5090-4699-7, s 233-238.
4. Jozef Papán et al.: The IPFRR mechanism inspired by BIER algorithm, in ICETA 2016 14th IEEE international conference on Emerging eLearning technologies and applications, November 24-25,2016, Starý Smokovec, The High Tatras, Slovakia,2016, ISBN 978-1-5090-4699-7, s 257-262.
5. Jozef Papán et al.: The survey of current IPFRR mechanisms, in Proceedings of the 2015 federated conference on Software development and object technologies, ISBN 978-3-319-46534-0, 2017 s 244-255; Advances in intelligent systems and computing, vol. 511, ISSN 2194-5357.

6. Marek Moravčík et al.: Overview of cloud computing and portability problems, in ICETA 2017 15th IEEE international conference on Emerging eLearning technologies and applications, November 26-27, 2017, Starý Smokovec, The High Tatras, Slovakia, 2017, ISBN 978-1-5386-3296-3, s 313-318.
7. Marek Moravčík et al.: Teaching cloud computing in cloud computing, in ICETA 2017 15th IEEE international conference on Emerging eLearning technologies and applications, November 26-27, 2017, Starý Smokovec, The High Tatras, Slovakia, 2017, ISBN 978-1-5386-3296-3, s 319-324.
8. Jakub Hrabovský et al.: Systolic-based 2D convolver for CNN in FPGA, in ICETA 2017 15th IEEE international conference on Emerging eLearning technologies and applications, November 26-27, 2017, Starý Smokovec, The High Tatras, Slovakia, 2017, ISBN 978-1-5386-3296-3, s 147-154.
9. Jozef Papán et al.: Existing mechanisms of IP fast reroute, in ICETA 2017 15th IEEE international conference on Emerging eLearning technologies and applications, November 26-27, 2017, Starý Smokovec, The High Tatras, Slovakia, 2017, ISBN 978-1-5386-3296-3, s 343-350.
10. Jana Uramová et al.: Packet capture infrastructure based on Moloch, in ICETA 2017 15th IEEE international conference on Emerging eLearning technologies and applications, November 26-27, 2017, Starý Smokovec, The High Tatras, Slovakia, 2017, ISBN 978-1-5386-3296-3, s 487-494.
11. Matilda Drozdová et al.: Contribution to cloud computing security architecture, in ICETA 2017 15th IEEE international conference on Emerging eLearning technologies and applications, November 26-27, 2017, Starý Smokovec, The High Tatras, Slovakia, 2017, ISBN 978-1-5386-3296-3, s 117-122.
12. Pavel Segeč et al.: Securing SIP infrastructures with PKI — The analysis, in ICETA 2017 15th IEEE international conference on Emerging eLearning technologies and applications, November 26-27, 2017, Starý Smokovec, The High Tatras, Slovakia, 2017, ISBN 978-1-5386-3296-3, s 405-412.