Sem vložte zadání Vaší práce.

České vysoké učení technické v Praze

Fakulta informačních technologií

Katedra Softwarového inženýrství . . . (18102)

Diplomová práce

Podpůrné komponenty simulátoru počítačové sítě

Bc. Martin Lukáš

Vedoucí práce: Ing. Pavel Kubalík, Ph.D.

8. května 2012

Poděkování

Nejprve bych chtěl poděkovat své rodině za vytvořené zázemí, které mi bylo oporou. Dále bych chtěl poděkovat vedoucímu práce Ing. Pavlovi Kubalíkovi, Ph.D. za užitečné rady a připomínky při tvorbě této práce. Dále mé poděkování za dobrou spolupráci patří kolegům, kteří se na společném projektu podíleli.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.
Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, avšak pouze k nevýdělečným účelům. Toto oprávnění je časově, teritoriálně i množstevně neomezené.

V Praze dne 8. května 2012 .....

České vysoké učení technické v Praze
Fakulta informačních technologií

Odkaz na tuto práci

Martin Lukáš. Podpůrné komponenty simulátoru počítačové sítě: Diplomová práce. Praha: ČVUT v Praze, Fakulta informačních technologií, 2012.

Abstract

This master thesis describes the design and implementation of components of a computer network simulator PSImulator2. The first component allows remote text-oriented access (telnet) to active elements of a virtual network. The second component creates a connection to an simulator's graphical simulation interface. The third component implements a simple virtual file system as an extension of active elements. The fourth component is a text editor application implemented with an text user interface of virtual active elements. The fifth component is an interface for storing a virtual network configuration.

Keywords

psimulator2, telnet, simulator, linux, cisco

Abstrakt

Tato práce se zabývá návrhem a implementací komponent simulátoru počítačových sítí PSImulator 2. První komponenta umožňuje vzdálený textový přístup(telnet) k aktivním prvkům virtuální sítě. Druhá komponenta vytvoří napojení simulátoru na jeho grafické simulační rozhraní. Třetí komponenta realizuje rozšíření virtuálních aktivních prvků o jednoduchý virtuální souborový systém. Čtvrtou komponentou je aplikace textového editoru v textovém uživatelském rozhraní pro virtuální aktivní prvky. Pátou komponentou je rozhraní pro uložení konfigurace virtuální sítě.

Klíčová slova

psimulator2, telnet, simulátor, linux, cisco

Obsah

Seznam obrázků

Úvod

Pro účely výuky předmětu BI-PSI Počítačové sítě [1] byl vytvořen v roce 2010 v rámci bakalářských prací Tomáše Pitřince [4] a Stanislava Řeháka [12] síťový simulátor [6]. Na simulátoru je možné testovat konfiguraci rozhraní, překladu adres, statických směrovacích pravidel. Serverová konzole vypisuje informace o průběhu operací v simulátoru. Počátečním požadavkům simulátor vyhovoval, nicméně vyvstaly požadavky nové a proto bylo rozhodnuto o podstatném rozšíření funkcionality.

Tato rozšíření lze formulovat do 4 hlavních kategorií:

1. rozšíření síťových simulačních funkcí
2. rozšíření o grafickou editaci topologie sítě, vizualizace datových toků v režimu simulace
3. modifikace způsobu terminálového přístupu k aktivním prvkům virtuální sítě
4. rozšíření prvků simulujících prostředí GNU/Linux o jednoduchý souborový systém a aplikaci editoru

Vzhledem k objemu očekávané práce byl původní tým vývojářů rozšířen o další dva členy. Původní vývojáři Stanislav Řehák a Tomáš Pitřinec se ve svých diplomových pracích [13] [5] zajímají především o rozšíření síťových simulačních funkcí a to vzhledem k jejich zkušenosti s předchozí verzí simulátoru. Další člen týmu Martin Švihlík se ve své diplomové práci [14] věnuje grafickému editoru topologie sítě a vizualizaci datových toků uvnitř virtuální sítě. Terminálový přístup k aktivním prvkům a ostatní nedílné komponenty simulátoru pak řeší právě tato práce.

Širší popis motivace vývoje nového simulátoru, přehled nových požadavků na sítové simulační funkce simulátoru a jejich realizaci lze nalézt v pracích Stanislava Řeháka [13] a Tomáše Pitřince [5]. Motivace,požadavky a realizaci grafického rozhraní simulátoru lze pak nalézt v práci Martina Švihlíka [14].

V této práci jsou uvedeny motivace,požadavky a realizace:

- vzdálené textové správy aktivních prvků virtuální sítě
- textového editoru v terminálovém uživatelském rozhraní
- jednoduchého souborového systému aktivních prvků virtuální sítě
- komponenty persistentního ukládání konfigurace virtuální sítě
- napojení simulačního serveru(backendu) na grafické rozhraní

Hlavním programovacím jazykem celého projektu je JAVA ve verzi 1.7. Volba tohoto jazyka plyne především z jeho užití v předchozí verzi simulátoru, z něhož bude část kódu převzata. Odůvodnění použití tohoto jazyka v předchozí verzi simulátoru lze nalézt ve výše zmíněných bakalářských pracích [4] a [12]. V těchto pracích lze rovněž nalézt motivaci vývoje původní verze simulátoru a rovněž rešerši dostupných alternativ.

KAPITOLA 1

Úvod do projektu

1.1 Architektura simulátoru

V této kapitole bych se chtěl ve stručnosti věnovat architektuře simulátoru jako celku tak, aby z ní bylo možné pochopení návaznosti jednotlivých komponent.

Návrh architektury a obecně celá vize nové verze simulátoru je víceméně prací autorů původní verze simulátoru. Detailnější popis architektury a zdůvodnění konkrétního návrhu je tedy uveden v jejich pracích [13] [5].

Simulátor v původní verzi disponoval pouze textovým výstupem a konfigurace topologie virtuální sítě byla možná pouze editací XML popisného dokumentu v textovém editoru. Pro novou verzi simulátoru má vzniknout grafický editor umožňující editaci topologie mnohem komfortnější a přehlednější formou běžnou z obdobných simulátorů. Dále grafické rozhraní má v simulačním módu umožnit zobrazovat procházející data uvnitř virtuální sítě. Grafické rozhraní je tak jakousi vizualizační nadstavbou simulátoru a není jeho nezbytnou (i když zcela určitě užitečnou a důležitou) součástí. Existuje tedy minimum informací, které simulátor a grafické rozhraní sdílí. Z tohoto důvodu je napojení simulátoru a grafického rozhraní realizováno výhradně skrze zmíněný XML soubor a TCP/IP spojení pro přenos informací o tocích v síti. Projekt je tedy rozdělen na backend(část kde dochází k simulaci) a na frontend(část pro vizualizaci)

O vytváření zmíněného XML souboru pojednávám v kapitole 5 Komponenta pro persistentní ukládání konfigurace virtuální sítě

O přenosu informací o tocích v síti pojednávám v kapitole 6 Komponenta napojení simulátoru na grafické simulační rozhraní

1.2 Projektový tým, rozdělení práce

Vzhledem k očekávanému rozsahu projektu a zmíněné vrcholné architektuře simulátoru byl projekt rozdělen na dvě hlavní části sdílející jen několik málo datových struktur. Jsou to části:

- backend - simulační část
- frontend - editor topologie a vizualizační část

V následující sekci je uveden hrubý výčet prací a odpovědností strukturovaný dle vývojářů a částí simulátoru. Nástin tohoto výčtu vznikl již v době plánování vývoje simulátoru. Nicméně s průběhem implementace a vzniknutím nových požadavků byl postupně mírně obměňován, až byl ustanoven do této podoby.

1.2.1 Backend

1.2.1.1 Stanislav Řehák

- architektura jádra simulátoru
- návrh napojení na skutečnou síť
- síťová vrstva: ARP protokol
- síťová vrstva: IP protokol (bez routovací tabulky)
- síťová vrstva: dynamický i statický překlad adres (NAT)
- transportní vrstva
- architektura aplikací
- traceroute aplikace
- Cisco IOS: parser příkazů
- simulace bufferů na rozhraních, zpoždění

1.2.1.2 Tomáš Pitřinec

- architektura jádra simulátoru
- návrh a implementace napojení na skutečnou síť
- fyzická vrstva
- linková vrstva
- síťová vrstva: směrovací tabulka (s úpravami se použije z aktuální verze simulátoru)
- architektura aplikací
- ping aplikace
- Linux: parser příkazů
- napojení na reálnou síť (pro ICMP)

1.2.1.3 Martin Lukáš (já)

- vzdálená textová správa(terminál) virtuálních aktivních prvků
- souborový systém pro aktivní prvky, pro virtuální prvky GNU/Linux implementace manipulačních příkazů (rm, mv, touch ...)
- jednoduchý textový editor v terminálovém rozhraní pro virtuální prvky GNU/Linux
- ukládání, načítání topologie sítě do XML
- napojení simulátoru na grafické simulační rozhraní

1.2.2 Frontend

1.2.2.1 Martin Švihlík

- vytváření a úprava topologie sítě
- automatické rozmístění síťových prvků
- vizualizace průchodu paketů sítí
- vypisování paketů přijatých z backendu

1.2.2.2 Martin Lukáš (já)

- zabudovaný grafický telnet klient
- standalone grafický telnet klient

1.3 Použité nástroje

Při vývoji projektu bylo spontánně a shodně všemi vývojáři využito vývojové prostředí Netbeans 7.1. V případě problémů tak nebyl problém si vzájemně pomoci. Samotný kód je samozřejmě možné importovat a upravovat i v jiných prostředí.

Pro správu kódu byl využit repositář Subversion hostovaný u společnosti google na adrese http://psimulator.googlecode.com/svn/

Komunikace mezi vývojáři probíhala prostřednictvím neveřejného fóra elektronické pošty groups.google.com

Stránky projektu pro uživatele a případné pokračovatele jsou dostupné na adrese http://code.google.com/p/psimulator/ Na stránkách projektu je k nalezení:

- popis a funkce simulátoru
- spustitelná aplikace simulátoru
- snímky obrazovky grafického rozhraní
- webový přístup do repositáře, kód aplikace
- a další informace pro uživatele i vývojáře ...

KAPITOLA 2

Komponenta vzdáleného přístupu

V této kapitole bych se chtěl věnovat nedílné komponentě simulátoru řešící terminálový přístup k aktivním prvkům virtuální sítě. Tato část projektu je hlavním důvodem mé účasti na projektu a tvoří pomyslné jádro této práce.

V první části této kapitoly popisuji serverovou část terminálového přístupu a aplikaci příkazové řádky navrženou a implementovanou v původní verzi simulátoru. Druhá část se zabývá serverovou částí terminálového přístupu a aplikací příkazové řádky pro novou verzi simulátoru. Třetí část je pak věnována klientům terminálového přístupu.

2.1 Server terminálového přístupu v původní verzi simulátoru

2.1.1 Popis problému

Jedním z funkčních požadavků první verze simulátoru je možnost připojit se k jednotlivým virtuálním prvkům sítě pomocí protokolu telnet. Pro splnění tohoto požadavku tvůrci původního simulátoru využili nástroje rlwrap [1] [8]. Toto řešení poskytlo splnění tohoto funkčního požadavku, avšak přineslo řadu omezení. Dovolím si citovat z práce Tomáše Pitřince [4]:

> "Můj kolega nelezl program rlwrap, který poskytuje historii příkazů a jejich doplňování na straně klienta. Funguje v Linuxu, na Windows jen v prostředí Cygwin. Toto řešení bude o dost jednodušší a

[1] read line wrapper

> rozhodli jsme se ho realizovat, i když pro uživatele bude nevýhodou spouštění přes cygwin. I tak ovšem základní požadavek, že s aplikací bude možno komunikovat pomocí telnetu zůstane zachován. Uživatel ovšem přijde o komfort, který mu nabízí možnost doplňování, editace a historie příkazů. "

Celé řešení s využitím nástroje rlwrap pracuje následovně. Uživatel vytváří příkaz v prostředí rlwrap, které poskytuje funkce:

- editace příkazu (posun kurzoru, mazání a zápis znaků)
- doplňování příkazu za využití předpřipraveného konfiguračního souboru odkud je seznam možných příkazů načten
- správa historie příkazů, vyhledávání v historii při stisku kombinace kláves CTRL+R

Program rlwrap po načtení odřádkování předává vytvořený příkaz programu telnet. Program telnet slouží pro síťový přenos příkazu v textové podobě na server. Server je pak realizován jako čtení z TCP/IP Socketu. Načítá se textový řetězec ukončený sekvencí \r\n. Načtený příkaz je pak dále zpracován parserem příkazů a případně vyhodnocen vnitřními procesy virtuálního počítače.

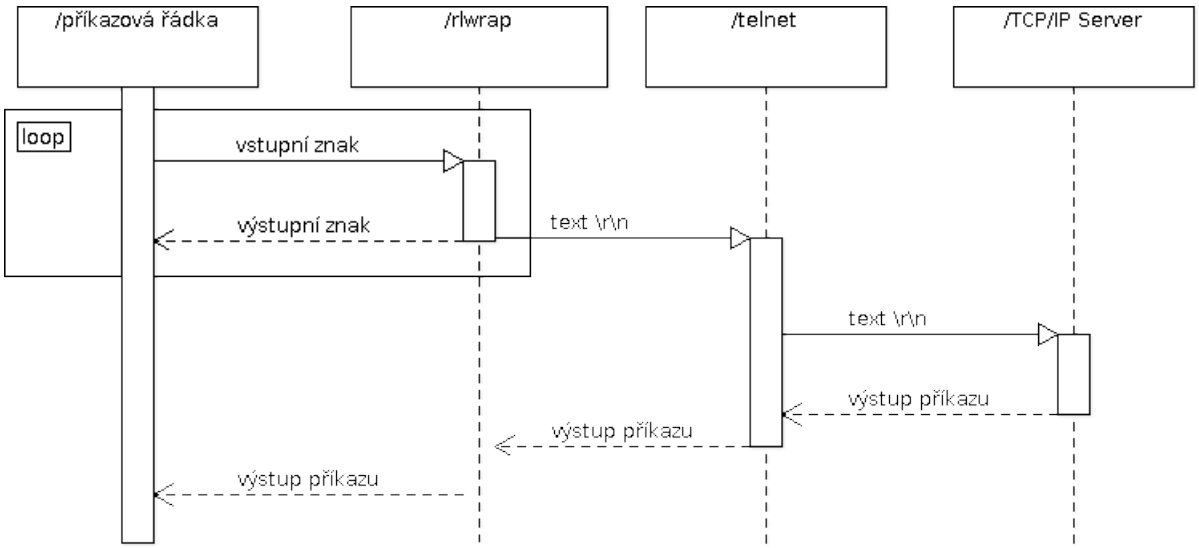
Pro názornost jsem zjednodušené schéma komunikace původního simulátoru s uživatelem zakreslil do obrázku: 2.1

Z výše uvedeného jsou patrná omezení takového řešení. Jde vlastně jen o jednoduchý způsob jak přenášet textový vstup a výstup, kdy načítání vstupu je volitelně doplněno o funkcionalitu nástroje rlwrap. Takovéto řešení vzdálené komunikace se označuje pod názvem "dumb terminal". Tímto způsobem například nelze realizovat aplikace typu textový editor a obecně používat pokročilé funkce vykreslování textového uživatelského rozhraní na straně serveru. Terminály toto umožňující se označují jako "video terminály" či "smart terminal"

Dalším omezením je samotná potřeba nástroje rlwrap, která je provozována na straně připojujícího se klienta a jehož instalace uživateli komplikuje provoz aplikace. Instalace rlwrap byla komplikací zejména na platformě Windows, kde bylo vyžadováno prostředí Cygwin.

2.1.2 Popis cíle

Cílem této části práce je vytvořit nedílnou komponentu původního simulátoru, která by alespoň z části nahradila funkcionalitu nástroje rlwrap. Projekt tak ztratil závislost na tomto nástroji a zároveň byla zachována použitelnost vzdálené



Obrázek 2.1: Zjednodušené schéma komunikace původního simulátoru s uživatelem

příkazové řádky. Odstranění závislosti na nástroji rlwrap rovněž zvýší komfort uživatele při instalaci i používání simulátoru.

Komponenta bude vyvíjena s vědomím, že bude později přenesena do nové verze simulátoru. Komponenta by měla být vyvinuta v původním simulátoru alespoň do takového stavu, že bude umožňovat:

- připojit se k aktivnímu prvku virtuální sítě
- zadat a editovat příkaz jako běžné textové pole
- doplnění příkazu na tvar ze staticky definované množiny řetězců. Vyvolání této akce při stisku klávesy TABULATOR
- zaznamenávání historie pro dané uživatelské sezení a listování v této historii pomocí kurzorových kláves UP a DOWN

Cílem je také mimo jiné prozkoumat a vyzkoušet možnosti práce s pokročilým terminálovým výstupem tak, aby v nové verzi simulátoru mohla být tato zkušenost využita pro vytvoření textového editoru či jiné terminálové interaktivní aplikace.

Není předpoklad, že by původní verze měla být po uvedení verze nové nadále používána a proto není nutné implementovat či zpětně portovat pokročilé funkce příkazové řádky z nové verze.

2.1.2.1 Nefunkční požadavky

Původní verze simulátoru je implementována výhradně v jazyku JAVA ve verzi specifikace 1.6, proto vytvářená komponenta musí být rovněž implementována v tomto jazyce.

Dalším nefunkčním požadavkem je absence nutnosti instalace jakéhokoliv nestandardního pomocného softwarového vybavení pro uskutečnění terminálového spojení. Za standardní softwarové vybavení je považován například telnet či ssh klient nikoliv však například nástroj rlwrap.

Při realizaci komponenty by neměly být využity uzavřené knihovny třetích stran. Případné licenční podmínky knihoven třetích stran by měly být takové, aby knihovny bylo možné dále svobodně upravovat a šířit s výsledným produktem.

2.1.3 Analýza problému

Problém je velmi dobře popsán,ohraničen a určen popisem cíle i nefunkčními požadavky. V první části analýzy se zaměřím na zkoumání dostupných standardních řešení použitelných pro realizaci výše zmíněných cílů. Byť v zadání je uveden přenosový protokol telnet, provedu alespoň krátký průzkum alternativních možností.

Stěžejní částí analýzy problému pak bude zejména průzkum knihoven a toolkitů pomocí nichž by měla realizace této komponenty být méně náročnou. Předložený problém v různých modifikací není nijak neobvyklým a proto je dobrý předpoklad, že vznikla řada projektů jež řešení problému usnadňují. Průzkum knihoven by se měl zaměřit především na knihovny implementující nalezená standardizovaná řešení z předchozí části.

2.1.3.1 Dostupná řešení

Cílem tohoto průzkumu dostupných řešení není detailně popisovat fungování protokolů a zkoumání specifikací, ale spíše nalézt přínosy a omezení pro řešení předloženého problému.

2.1.3.1.1 Telnet

Telnet je síťovým client-server protokolem postaveným nad spolehlivou transportní vrstvou, typicky nad protokolem TCP/IP.

Tento protokol byl využíván od 70.let převážně v lokálních sítích pro ovládání mainframe počítačů. Společně s rozšířením sítě internet na počátku 90.let se

začal využívat i pro vzdálenou správu serverů mimo lokální sítě. Brzy se však díky chybějícímu šifrování stal cílem útočníků a ještě než byl vytvořen standard rozšiřující ho o funkce šifrování, byl v praxi nahrazen protokolem Secure shell. V dnešní době se využívá pro terminálový přístup k zařízením při kterém nejsou vyžadovány funkce šifrování přenosu a autentifikace. Častým případem užití dnes je i kombinace SSH a telnet protokolu, kdy SSH spojení je použito pro průchod skrze nezabezpečenou síť a telnet protokol pro přímé spojení či spojení ve známé lokální síti.

Mezi hlavní výhody telnet protokolu je jeho relativní univerzálnost, jednoduchost použití a díky své dlouhé historii i velké rozšíření napříč systémy. Této vlastnosti využívají v praxi například různé vestavěné systémy s omezenou výpočetní a paměťovou kapacitou. Ve své jednoduché podobě(raw mode) byl využit i pro implementaci spojení uživatele a virtuálního aktivního prvku v předchozí verzi simulátoru.

2.1.3.1.2 NVT

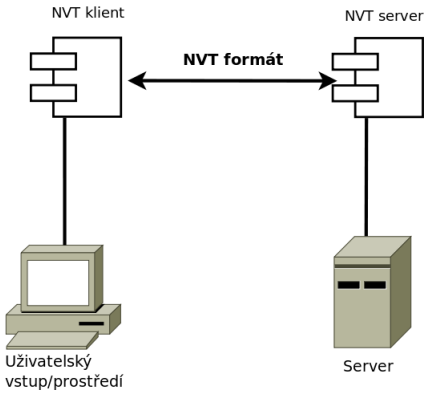
NVT(Network Virtual Terminal) je podmnožinou protokolu telnet. Tato podmnožina se se zabývá prezentací dat. NVT vzniklo za účelem ujednocení prezentace dat napříč různými druhy terminálů a systémů. Tento protokol se například využije pokud klientský terminál využívá pro označení odřádkování kód CR a server využívá CRLF. V tomto případě dojde k překladu jak ve směru od serveru, tak ve směru od klienta. Význam protokolu by měl být patrný z obrázku 2.2.

NVT jako překladová vrstva mezi různými typy terminálů omezuje schopnosti terminálu na takovou množinu funkcí jež je společná prakticky všem široce rozšířeným terminálům. Nicméně existuje zde mechanismus umožňující vyjednání nadstandardních funkcích, které nelze vynucovat. Vyjednání rozšiřujících funkcí lze spustit nejen bezprostředně při připojení, ale i během komunikace.

Dále během vyjednávání je předávána zejména geometrie terminálu(počet znakových řádků a sloupců), kódování a jiné.

2.1.3.1.3 Secure shell

SSH je protokolem, který vznikl jako reakce na chybějící zabezpečení proti odposlechu a podvržení identity v protokolu telnet. Pro sestavení zabezpečeného kanálu se využívá asymetrické kryptografie minimálně s privátním klíčem na straně serveru a veřejným na straně klienta. Lze však využít i konfigurace, kdy privátní klíč je uložen na straně klienta a veřejný je užíván na straně serveru, této konfigurace se využívá pro ověření identity uživatele. SSH kromě typického



Obrázek 2.2: NVT schéma použití

použití pro vzdálené připojení k serveru je využíváno i v dalších protokolech jako SFTP(SSH File Transfer) a SCP(Secure copy)

Mezi hlavní výhody SSH protokolu nad protokolem telnet patří zmíněná podpora zabezpečení. Tato podpora však s sebou přináší i složitější implementaci než právě telnet. Dále správa veřejných a privátních klíčů na serveru i klientu komplikuje nasazení v rámci simulátoru. Mnoho uživatelům by pravděpodobně i vadilo importovat si do systému klíče virtuálních prvků simulátoru.

Vzhledem k absenci předpokladu komunikace backendu simulátoru a uživatele na nezabezpečené síti a absenci rizika zneužití nezabezpečeného protokolu v našem nasazení, jsem se rozhodl pro realizaci vzdálené textové správy využít jednoduššího protokolu telnet.

2.1.3.1.4 Video terminál

Pojem "počítačový terminál" označuje fyzické zařízení schopné na základě vstupních příkazů, řídících kódů a jiných dat získaných z počítače zobrazovat znakový výstup a na základě akcí uživatele generovat vstup předávaný počítači. Nejčastěji jde o zařízení se znakovou zobrazovací jednotkou ve formě monitoru a vstupní znakovou jednotkou ve formě klávesnice 2.3.

V dnešní době se pojem terminál nesprávně využívá pro software poskytující znakové rozhraní počítače či přímo pro příkazovou řádku. Správný pojem označující takovýto software je "emulátor terminálu". Jde totiž o softwarovou emulaci znakově orientovaného fyzického zařízení na počítači využívající bodový (pixelový) výstup. Emulátor terminálu je navíc často schopný emulace několika různých terminálových specifikací současně.

První terminály umožňovaly jen kontinuální výstup na obrazovku, případně pohyb kurzoru a editaci na poslední řádce. Tyto terminály se označovali jako "dumb terminal". S ohledem na rostoucí požadavky výstupu počítačů byly vytvořeny tzv. video terminály. Tyto terminály umožnily pomocí speciálních kontrolních sekvencí, příkazů manipulovat s celým znakovým rastrem terminálu. Pro tyto kontrolní sekvence byly vytvořeny ANSI(American National Standards Institute) standardy tak, aby byla zajištěna kompatibilita terminálů a počítačů. Pro část kontrolních kódů a definice písmen byl použit standard ASCII kódů.

Zřejmě nejznámějším a nejpoužívanějším terminálem a specifikací terminálu je ANSI/VT100.

Pro ilustraci v následujícím odstavci uvádím jak vypadá navázání telnet spojení a vyjednání parametrů z pohledu klienta. Uvedeno je vyjednávání realizované oproti finální verzi telnet serveru síťového simulátoru.:

```
telnet> set options
Will show option processing.
telnet> open localhost 11000
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
RCVD WILL ECHO
SENT DO ECHO
RCVD DONT ECHO
RCVD DO NAWS
SENT WILL NAWS
SENT IAC SB NAWS 0 190 (190) 0 41 (41)
RCVD WILL SUPPRESS GO AHEAD
SENT DO SUPPRESS GO AHEAD
RCVD DO SUPPRESS GO AHEAD
SENT WILL SUPPRESS GO AHEAD
RCVD DO TERMINAL TYPE
SENT WILL TERMINAL TYPE
RCVD DO NEW-ENVIRON
SENT WILL NEW-ENVIRON
RCVD IAC SB TERMINAL-TYPE SEND
RCVD IAC SB ENVIRON SEND 000 003
SENT IAC SB ENVIRON IS
pc1:/#
```

Obrázek 2.3: VT100 terminál představený roku 1978

2.1.3.2 Knihovny

V této sekci bude provedena rešerše dostupných knihoven a toolkitů, které by měli pomoci při řešení předloženého problému. V případě, že samotná knihovna nebude dostatečně směřovat ke zjednodušení návrhu a implementace, je možné knihovny vhodně kombinovat. Knihovny budou zkoumány z pohledu následujících kritérií, která by měla být splněna:

- knihovny použitelné v jazyce JAVA, tj. implementovány přímo v jazyce JAVA nebo doplněny rozhraním pro tento jazyk s multiplatformní podporou.
- dostupný zdrojový kód a licenční podmínky umožňující úpravu kódu a jeho šíření s výsledným produktem.
- minimálně JavaDoc dokumentace, výhodou jsou ukázkové příklady nasazení a návody na použití
- knihovna s funkcemi směřujícími ke splnění funkčních a nefunkčních požadavků. Zejména podpora serveru vzdáleného připojení a dále funkce pro vytváření pokročilého textového uživatelského rozhraní.

Výběr knihovny do značné míry ovlivní návrh a implementaci řešení.

2.1.3.2.1 Knihovna CHARVA

CHARVA je knihovna umožňující programovat textové uživatelské rozhraní složené z elementů jako jsou okna, dialogy, tlačítka, vstupní textová pole, menu a jiná. Rozhraní knihovny je velmi podobné AWT či Swing rozhraní ze standardní knihovny Java.

Již existující aplikaci vytvořenou s AWT lze pouhou změnou "import" hlaviček přeměnit na aplikaci s výstupem v textovém terminálu viz. snímky 2.4 a 2.5. Knihovna je šířena v licenci GPL.

Knihovna se skládá ze dvou částí:

- Java tříd realizující rozličné "grafické" elementy
- dynamicky načítané sdílené knihovny zapsané v jazyce C.

Java třídy přistupují k funkcím knihovny v nativním kódu skrze JNI(Java native interface).

Výhodou této knihovny je zejména podobnost jejího rozhraní s rozhraními knihoven pro tvorbu grafického rozhraní v standardní edici javy. Za nevýhodu považuji její závislost na nativním kódu, jež by mohla přinést komplikace při portování simulátoru na různé platformy.

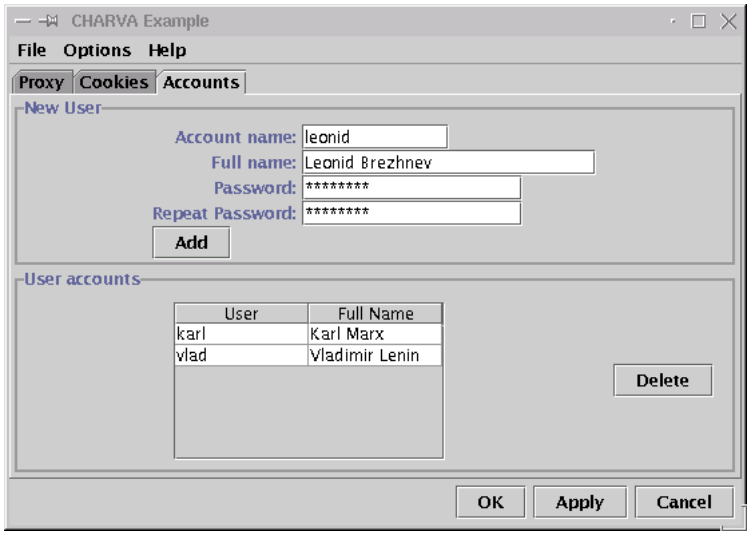
2.1.3.2.2 Knihovna telnetd2

Při hledání Java knihoven pracujících s protokolem telnet na straně serveru jsem nalezl prakticky pouze tento jediný projekt [9], který působil seriozním dojmem a netvářil se jako nedokončený studentský projekt s chybějící dokumentací.

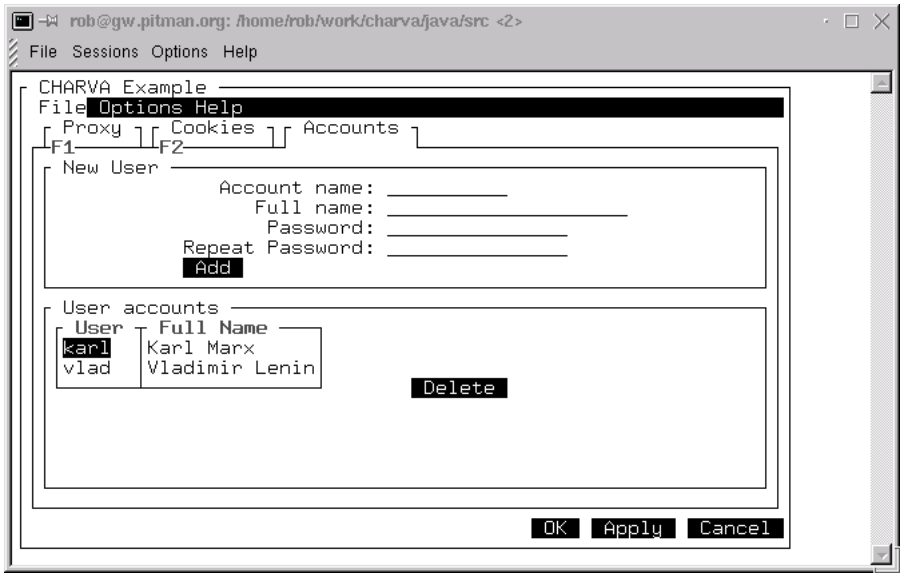
Seriozní dojem u mne vzbudila knihovna z důvodu:

- existence návodu na její použití
- dokumentace API knihovny pomocí JavaDoc
- její nasazení v komerčních projektech jako například Sun StoreEdge [10]
- přehlednosti jejího návrhu a jednoduchosti použití již po krátké době seznamování se s API knihovny.

Knihovna rovněž splňuje všechna výše vypsaná kritéria (2.1.3.2) a proto byla vybrána pro realizaci tohoto úkolu. Knihovna mě mimo jiné zaujala i z důvodu, že nejen pracuje s protokolem telnet, ale i poskytuje toolkit textového



Obrázek 2.4: Grafické rozhraní ukázkové aplikace využívající knihovny AWT. Zdroj: [2]



Obrázek 2.5: Textové rozhraní ukázkové aplikace využívající knihovny CHARVA. Zdroj: [2]

rozhraní. Tohoto toolkitu jsem později využil pro realizaci jednoduchého textového editoru viz. kapitola 3 . Jde tedy o ucelené řešení pro realizaci vzdálené textové správy. Využíti toolkitu komponent textového rozhraní je volitelné.

Pro realizaci TUI s touto knihovnou lze tedy použít:

- dodávaný toolkit komponent z balíčku telnetd.io.toolkit skládající se zejména z:
  - EditField - komponenta jednořádkového, krátkého textového vstupu
  - EditArea - komponenta víceřádkového textového vstupu
  - Form - zobrazovacího kontejneru
  - StatusBar - výstupní pole na konci(poslední řádce) terminálového okna
  - TitleBar - výstupní pole na začátku(první řádce) terminálového okna
  - Selection - komponenta výběru položky z menu
  - Label - komponenta jednoduchého výstupu
- vlastní realizaci textového rozhraní, čehož jsem využil pro realizaci příkazové řádky (třída CommandShell, potažmo ShellRenderer).
- knihovnu třetí strany s nastavitelným vstupem/výstupem tak, aby jej bylo možné přesměrovat patřičným metodám knihovny realizující telnet spojení. Pro tuto variantu se použitelnou jeví knihovna CHARVA, nicméně tato možnost nebyla zcela ověřena, ale pouze nastíněna autorem knihovny telnetd2.

Knihovna nebo nástroj telnetd2 [9] je open source software napsaným v jazyce java s cílem implementovat Java telnet server, který by byl snadno zabudovatelný do aplikací třetích stran. Mezi hlavní vlastnosti nebo části serveru patří:

- implementace protokolu telnet s podporou NVT, ECHO, TTYPE, NAWS, LINEMODE,NEWENV
- podpora terminálového výstupu pro různé typy terminálů (VT100, ANSI, xterm)
- jednoduchý toolkit textového uživatelského rozhraní implementovaný jako vrstva s objektově orientovaným přístupem postavená nad terminálovým vstupem a výstupem.

Knihovna vyžaduje pro svůj běh javu verze 1.5 a vyšší.

Logiku struktury knihovny a jejího použití lze dobře pochopit z části 2.1.4 návrh a implementace řešení, kde pojednávám o jejím zabudování a jejím využití ve vyvíjeném projektu.

2.1.4 Návrh a implementace řešení

Problém jako takový není svým zadáním nijak komplikovaný a nepotřebuje rozsáhlého návrhu a modelování. V této fázi jde především o správné pochopení fungování původního řešení simulátoru a o správné pochopení práce s vybranou knihovnou tak, aby bylo možné původní komunikační část simulátoru nahradit novou implementací.

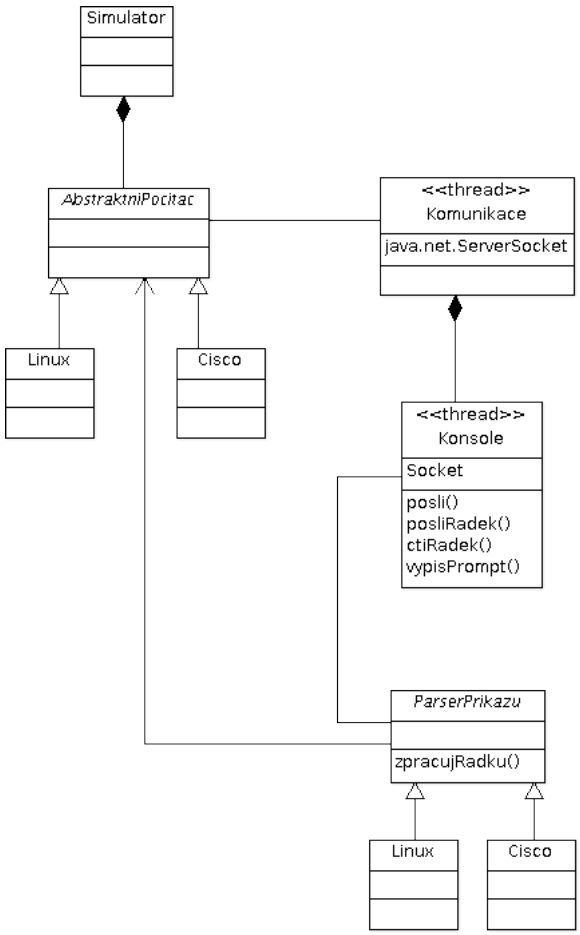
Pro představu o původním třídním modelu jsem vytvořil zjednodušený (vybrány jen relevantní třídy, metody, atributy) UML model. Tento model je na obrázku 2.6. Z tohoto obrázku je patrné, že nová komponenta by měla nahradit a navázat na třídy Komunikace a Konsole tak, aby nebylo nutné provádět výrazné změny v ostatních částech simulátoru.

Třída nazvaná Komunikace zajišťuje naslouchání na síťovém portu a vytváří nová vlákna začínající ve třídě Konsole. Tuto funkcionalitu naslouchání na síťovém portu bude zajišťovat dodaný server z knihovny telnetd2, proto tato třída bude odstraněna bez relevantní náhrady.

Třída Konsole, jejíž metody pro vstup/výstup jsou využívány v různých částech simulátoru, se stane třídou na rozhraní mezi telnet serverem z knihovny telnetd2 a ostatními strukturami simulátoru. Třída Konsole bude implementovat rozhraní z knihovny telnetd2 nazvané Shell.

Implementace rozhraní Shell znamená:

- implementaci metod vykonávající reakce na události spojení. Například odpojení, ztrátu spojení, vypršení spojení a jiné.
- implementaci metody run, které bude po vytvoření spojení s klientem předáno řízení v již vytvořeném vlákně. Jediným parametrem této metody je objekt třídy Connection, z níž je zejména dostupný:
  - objekt terminálového vstupu/výstupu. To je objekt implementující rozhraní BasicTerminalIO
  - různé parametry připojení jako například rozlišení rastru terminálu, Socket připojeného uživatele a jiné.

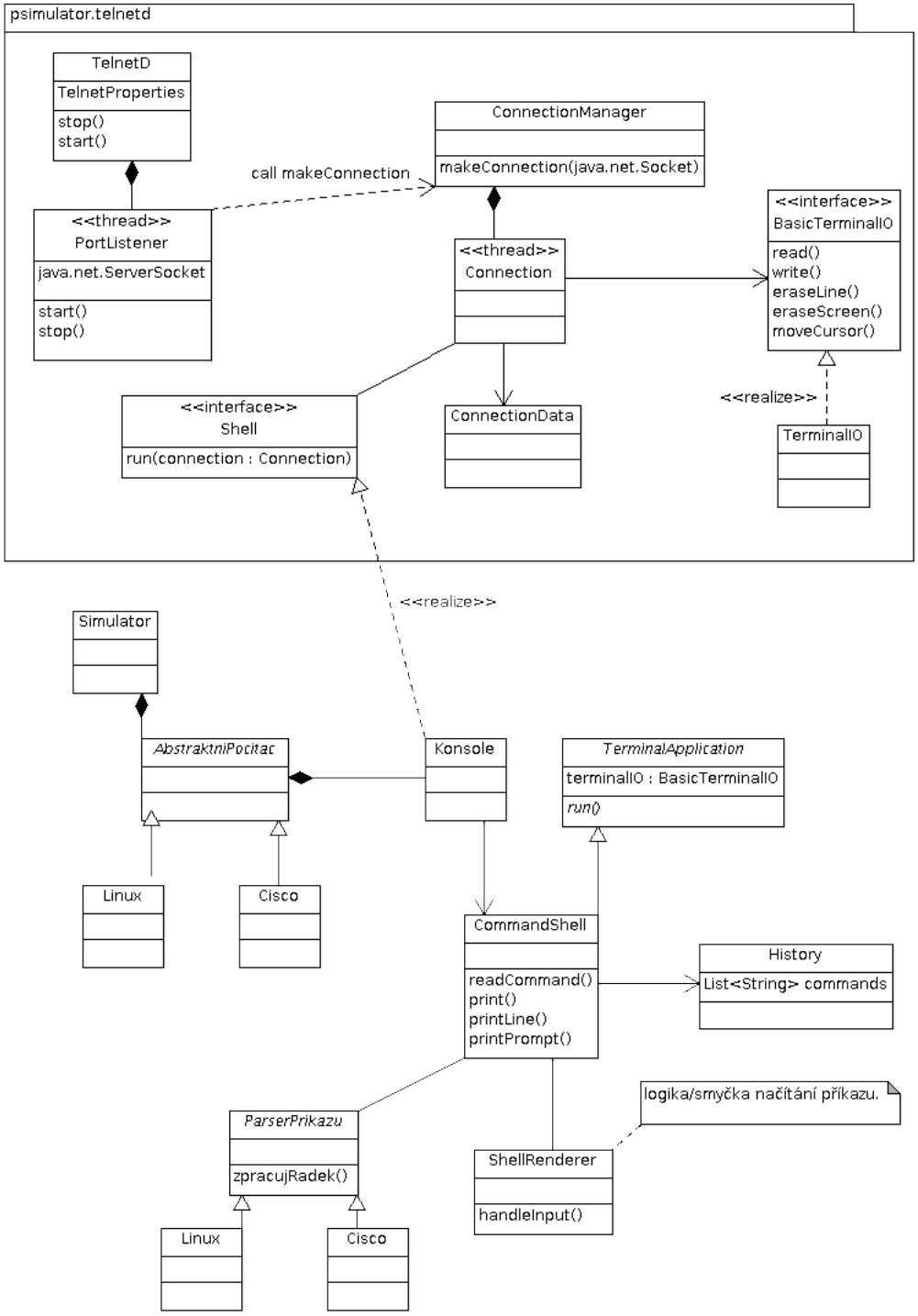


Obrázek 2.6: Zjednodušený třídní model původního komunikačního rozhraní původního simulátoru.

Návrh propojení jako zjednodušený(vybrány jen relevantní třídy, metody, atributy) třídní návrh je zakreslen na obrázku 2.7

Z třídy Konsole jsem v zájmu přehlednosti vyčlenil metody pro ovládání vstupu/výstupu do nově vzniklé třídy nazvané CommandShell. V třídě Konsole se tak nyní odehrává jen inicializace struktur simulátoru a dále je předáno řízení metodě CommandShell.run()

V metodě CommandShell.run() je ve smyčce, která může být ukončena změnou příznaku quit, volána metoda handleInput() třídy ShellRenderer. V metodě handleInput() je pak implementovaný jednoduchý automat, jež načítá uživatelský vstup znak po znaku. Výstupem tohoto automatu je pak načtený příkaz předaný struktuře ParserPrikazu, stejně jako tomu bylo v původní verzi simulátoru.



Obrázek 2.7: Zjednodušený třídní model nového komunikačního rozhraní původního simulátoru.

Automat rovněž obsluhuje řídící znaky jako UP/DOWN pro procházení historie a LEFT/RIGHT/BACKSPACE/ENTER pro editaci příkazu a TABULATOR pro vyvolání procedur vedoucí k doplnění příkazu.

Historie příkazů není mezi spojeními persistentně ukládána a při listování historií není ani rozlišováno mezi různými módy příkazové řádky pro CISCO. K procházení historie slouží klávesa UP, pro předchozí příkaz v historii, a klávesa DOWN pro následující příkaz v historii. Historie jako objekt třídy History je defacto jen seznam s objekty třídy String doplněný o manipulační metody.

Akce vyvolaná po stisku klávesy TABULATOR operuje se staticky definovaným seznamem příkazů(List<String>). Pokud k doplňovanému textu(příkazu) v příkazové řádce existuje pouze jedno možné doplnění, pak je provedeno. Pokud k doplňovanému textu existuje více než jedno doplnění, jsou tato doplnění uložena v bufferu a není proveden žádný výstup. V případě, že je další přečtený znak opět TABULATOR, je tento buffer vytisknut. V opačném případě je buffer vymazán.

2.1.5 Závěr, shrnutí

Terminálové rozhraní bylo implementováno do stavu použití srovnatelného s původní implementací využívající nástroje rlwrap. Jedinými citelnými výjimkami jsou persistentní uložení historie mezi sezeními a pokročilé vyhledávání v historii. Tato funkcionalita bude dopracována až v nové verzi simulátoru.

Úspěšnou realizací vzdáleného terminálového rozhraní v původní verzi simulátoru byla ověřena správnost směru vývoje textového rozhraní za využití knihovny telnetd2. Pokud nevyvstane v nové verzi simulátoru požadavek, který by znemožnil pokračování vývoje za pomocí zmíněné knihovny, bude využita knihovna telnetd2 i v nové verzi simulátoru.

2.2 Server terminálového přístupu v nové verzi simulátoru

2.2.1 Popis cíle

Ve stručnosti lze říci, že cílem vytvářené komponenty pro novou verzi simulátoru je se co možná nejvíce přiblížit způsobu vzdálené textové správy používaným v praxi alespoň tak, aby byly splněny všechny vypsané funkční a nefunkční požadavky. Soubor požadavků vznikal ve spolupráci s vývojáři původní verze simulátoru, kteří se danou problematikou již zabývali a současně mají určitou vizi nejen o funkcích navrhované komponenty, ale i o nové verzi síťového simulátoru jako celku.

2.2.2 Funkční požadavky

Funkční požadavky pro nové terminálové rozhraní původního simulátoru vznikaly víceméně ad-hoc z důvodu absence předpokladu dlouhodobého vývoje pro původní verzi simulátoru. Realizace terminálového rozhraní pro původní verzi simulátoru měla spíše charakter hlubšího zkoumání možných způsobů realizace tak, aby v nové verzi simulátoru mohly být využity zkušenosti z tohoto zkoumání.

Funkční požadavky pro novou verzi simulátoru byly vytvořeny z existující funkcionality nového terminálového rozhraní v původní verzi simulátoru a z rozšíření o další požadavky. Všechny tyto požadavky jsou sepsány v následujících několika odstavcích.

2.2.2.1 Požadavky na možnosti vzdáleného textového rozhraní

- při startu simulátoru bude možné určit síťové porty na kterých budou naslouchat servery terminálového spojení pro aktivní prvky virtuální počítačové sítě.
- Komponenta bude nějakým způsobem umožňovat tvorbu TUI nebo alespoň pokročilé funkce vykreslování textového rozhraní. Za pokročilé funkce vykreslování textového rozhraní je považován například pohyb kurzoru na obrazovce, vymazání řádky, vymazání obrazovky apod. Upřednostňovány jsou standardizovaná řešení jako například video terminál VT100 či xterm.
- V terminálu bude možné vykreslovat alespoň množinu znaků z ASCII tabulky

2.2.2.2 Požadavky na vytváření příkazů v terminálu

Jde o souhrn požadavků na způsob vytváření příkazu. Požadavky často kopírují funkce unixového shellu BASH [2]. Nicméně implementovat kompletní funkcionalitu tohoto shellu by bylo náročné a pro účely síťového simulátoru zbytečné. Proto jsou v následujících odstavcích vypsány funkční požadavky které je nutné splnit.

[2] Bourne again shell - zřejmě nejpoužívanější shell v Linuxu

2.2.2.2.1 Načítání příkazu a jeho předání k dalšímu zpracování

Načítání příkazu může být ukončeno:

- klávesou Enter - editovaný text bez ukončovací sekvence \r\n bude předán dále zpracující struktuře jako objekt datového typu String.
- kombinací kláves CTRL+C - editovaný text není předáván, dojde k odřádkování a načítání nového textu
- kombinací kláves CTRL+D - editovaný text není předáván, dojde k ukončení spojení

2.2.2.2.2 Pohyb kurzoru a editace příkazu

Oblast zadávání příkazu musí podporovat pohyb kurzoru a editaci pomocí:

- klávesy TABULATOR pro předání hodnoty editovaného příkazu dalším strukturám k automatickému doplnění příkazu
- kurzorových kláves. tj. pohyb kurzoru VLEVO a VPRAVO
- klávesy HOME pro pohyb kurzoru na začátek editované oblasti
- klávesy END pro pohyb kurzoru na konec editované oblasti
- kombinace kláves CTRL+W pro vymazání slova předcházejícího pozici kurzoru
- klávesy DELETE pro smazání znaku na pozici kurzoru
- klávesy BACKSPACE pro smazání znaku předcházejícího pozici kurzoru
- klávesy ENTER pro potvrzení a ukončení načítání příkazu

2.2.2.2.3 Módy příkazové řádky

Rozhraní příkazové řádky musí podporovat 3 základní módy:

1. COMMAND_READ - mód určený pro načítání příkazu
2. NORMAL_READ - mód určený pro obsluhu terminálového vstupu/výstupu v době běhu dlouhotrvajících příkazů(ping a podobné)
3. INPUT_FIELD - mód určený pro načtení libovolného textového řetězce

Během vykonávání dlouho trvajícího příkazu např. ping. bude použit mód NORMAL_READ. V něm bude zachytáván uživatelský vstup a v případě pokud :

- je načtena kombinace kláves CTRL+C a CTRL+D, informovat o tomto patřičné struktury simulátoru
- je načten ostatní vstup, je bez dalších akcí zapsán na výstup.

2.2.2.2.4 Podpora historie příkazů

Příkazová řádka musí umožňovat programově nastavitelnou změnu aktuální historie(seznam potvrzených příkazů) za jinou. Tento požadavek je vytvořen z důvodu, kdy je žádoucí např. při simulaci příkazového prostředí systému CISCO IOS oddělovat historii v různých konfiguračních módech.

Řádek potvrzený klávesou ENTER bude vložen do aktuální historie příkazů. Historie, jakožto seznam příkazů, musí být persistentně uložena při ukončení terminálového sezení tak, aby mohla být opětovně načtena a použita při následujícím terminálovém sezení. Pomocí procházení této historie bude možné příkaz načíst, zobrazit v příkazové řádce a buď jej okamžitě potvrdit a předat dále ke zpracování a vyhodnocení, nebo jej využít k další editaci.

Procházení historie bude vyvoláváno následujícími klávesami:

- kurzorovou klávesou “UP”, jež způsobí načtení předcházejícího příkazu. Opakováním této akce bude docházet k procházení historie zpět.
- kurzorovou klávesou “DOWN” jež slouží k pohybu v historii směrem vpřed.
- kombinací kláves CTRL+R pro vstoupení do módu vyhledávání. Vstup do toho módu musí být zjevně signalizován.
  - vyhledávání probíhá se změnou zadaného textu, je zobrazena vždy jedna odpovídající položka z historie příkazů
  - v módu vyhledávání lze opětovným zadáním CTRL+R listovat v záznamech odpovídajících zadanému textu
  - klávesou ENTER dojde k odeslání aktuálně zobrazené položky historie pro další zpracování
  - jednou z kurzorových kláves dojde k ukončení módu vyhledávání a k návratu do módu editace příkazu. Editační pole je nastaveno na poslední aktuálně zobrazenou položku historie.

2.2.3 Nefunkční požadavky

Nefunkční požadavky jsou oproti původnímu simulátoru změněny o verzi implementačního jazyka. V nové verzi simulátoru je využívána JAVA ve verzi specifikace 1.7.

2.2.4 Návrh řešení, implementace

2.2.4.1 Rozhraní komunikační vrstvy a simulačního jádra

Pro novou verzi simulátoru, v níž je podstatně rozšířena funkcionalita textového rozhraní, bylo potřeba část návrhu a kódu z původní verze simulátoru upravit(refaktoring) tak, aby vyhovovala rozhraní simulačního jádra, potažmo rozhraní třídy AbstractParser.

Návrh a vývoj třídy AbstractParser je v režii vývojářů Stanislava Řeháka a Tomáše Pitřince, proto zde uvedu pouze prototypy relevantních funkcí pro vrstvu zpracovávající komunikaci a jejich popis pro můj návrh řešení.

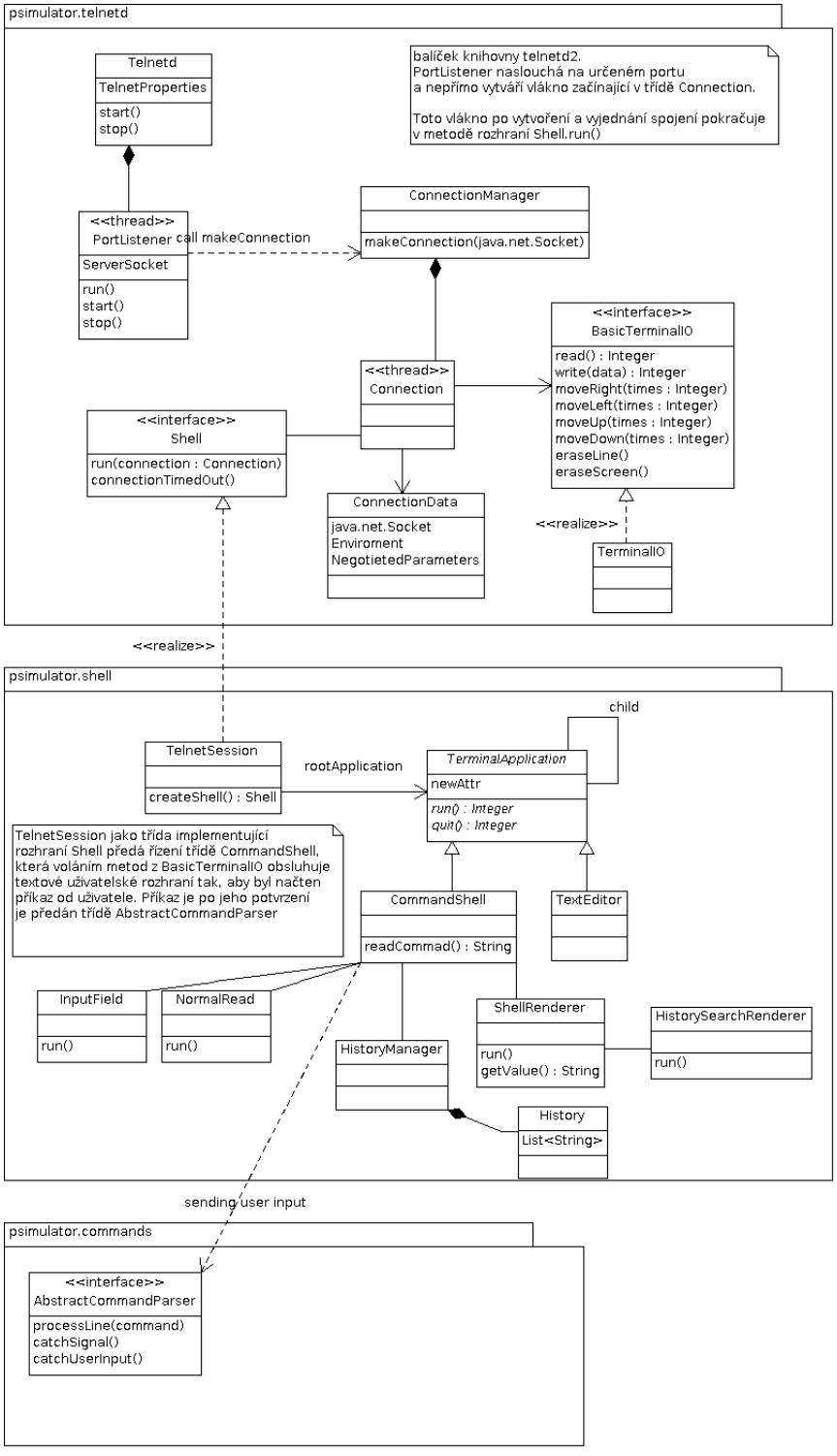
- processLine(String line) - metoda, které je předáván kompletní potvrzený uživatelský vstup(příkaz) v modu COMMAND_READ
- catchSignal(Signal signal) - metoda pro oznámení zachycení signálu(CTRL+C, CTRL+D) v modu NORMAL_READ
- catchUserInput(String input) - metoda pro předání textového vstupu přečteného v modu INPUT_FIELD

2.2.4.2 Diagram komunikační vrstvy

Následuje popis diagramu 2.8. Diagram je zjednodušený s cílem vyšší přehlednosti. Jsou vynechány některé asociace, většina atributů, metod a dokonce některé méně důležité třídy.

Třída TelnetSession je odrazem třídy Konsole z původního návrhu. Kromě změny jména třídy, které nyní více vystihuje její smysl, byl mírně upraven i její kód. Její smysl, tj. vstupní brána do komunikační vrstvy simulátoru po inicializaci knihovnou telnetd2, byl však víceméně zachován.

CommandShell jako třída reprezentující příkazovou řádku a rozhraní metod (readCommand, print, printLine) pro základní manipulaci s terminálovým výstupem byla mírně pozměněna. Do smyčky, která v původním návrhu periodicky volala metodu readCommand a výsledek předávala parseru, přibyl switch statement nad proměnou určující mód příkazové řádky.



Obrázek 2.8: Zjednodušený třídní diagram komunikační vrstvy simulátoru. Blíže popsaný v části 2.2.4.2

Módy příkazové řádky zmíněné v sekci 2.2.2.2.3 jsou implementovány v třídách ShellRenderer, NormalRead a InputRead.

Při implementaci reakcí na některé klávesové kombinace(CTRL+C, CTRL+D) a další klávesy(HOME, END) bylo potřeba do knihovny telnetd2 doplnit některé ANSI řídící kódy. Kódy jsem vyhledal na webu a doplnil do třídy TerminalIO, kde jsou uloženy i definice ostatních znaků.

2.3 Klienti terminálového přístupu

2.3.1 Podporovaní klienti

Již delší dobu v praxi není využívána realizace terminálu jako fyzického zařízení, ale jako prvku uvnitř softwarového emulátoru na běžném PC. Tyto emulátory nejen umožňují vícenásobné spuštění instance terminálu v rámci jednoho počítače, ale zejména podporují emulaci různých specifikací terminálu a znakových sad. Lze se tedy setkat s emulátorem terminálu využívající pro manipulaci s obrazovkou kontrolních kódů dle specifikace terminálu VT100 a zároveň s podporou pro rozšíření v podobě VT220, VT320, VT420 a současně podporou mnoha jiných nestandardních terminálů.

Většina knihoven pracujících s terminálovým výstupem považuje za pomyslný základ terminál VT100/ANSI. Pokud chtějí využít jiného terminálu s pokročilejšími funkcemi, proběhne mechanismus vyjednávání zmíněný v sekci 2.1.3.1.1. Stejně je tomu tak i v případě knihovny telnetd2.

Mezi vyzkoušené klienty telnet terminálového připojení patří:

- putty - http://www.putty.org/ - asi nejznámější a nejpoužívanější SSH/telnet klient používaný na platformě Windows. Dostupná je i verze pro GNU/Linux.
- BSD telnet klient + xterm
- grafický klient dodávaný se simulátorem, viz. další sekce 2.3.2

2.3.2 Zabudovaný grafický klient

Při implementaci síťového simulátoru jsme v kolektivu projektového týmu postupně dospěli k vytvoření nového požadavku a to vytvoření komponenty grafického telnet klienta, který by bylo možné spouštět přímo z grafického rozhraní simulátoru v simulačním režimu. Motivací vytvoření tohoto požadavku bylo zvýšení komfortu uživatele při používání simulátoru v simulačním režimu a to díky:

- zjednodušení připojování uživatele do vzdáleného textového terminálu aktivních prvků. Uživatel nebude již muset instalovat a spouštět program telnet klienta a zadávat údaje pro připojení.

- zvýšení přehledu o tom ke kterému prvku v topologii sítě se uživatel připojuje. Uživatel si nebude muset číst informační výstup ze spuštěného simulačního serveru(backend simulátoru) ve kterém by hledal dvojici (název prvku, telnet port), kterou by si musel zapamatovat.

Dále při splnění tohoto požadavku dojde k ucelení simulátoru z toho pohledu, že terminálový server již nebude muset dbát na kompatibilitu s různými klienty třetích stran, ale bude se možné při implementaci a testování aplikace omezit jen na podporu dodávaného zabudovaného klienta.

V rámci těchto myšlenek byly vytvořeny stručné funkční a nefunkční požadavky vypsané v následujících dvou seznamech:

Funkční požadavky

- terminálový klient musí by měl být spustitelný jako samostatná aplikace

- terminálový klient musí být snadno spustitelný a ovladatelný z programového kódu grafického rozhraní simulátoru.

- Grafické rozhraní komponenty musí disponovat vícejazyčnou podporou. Konkrétně alespoň podporu jazyků češtiny a angličtiny, podobně jako grafické rozhraní simulátoru.

- při spouštění komponenty musí být možné nastavit jazyk grafického rozhraní komponenty tak, aby se shodoval s jazykem grafického rozhraní simulátoru.

- při spouštění komponenty musí být možné nastavit cílového hosta a port k němuž se terminálový klient bude automaticky po spuštění připojovat.

Nefunkční požadavky

- Komponenta by měla být vytvořena s minimálním úsilím a v krátkém čase tak, aby práce vložená do její implementace negativně neovlivnila zbývající části simulátoru. Jde o doplňkovou funkcionalitu, která při jejím vypuštění neovlivní splnění požadavků na síťový simulátor jako celek.

- Komponenta by měla být šířena pod licenčním ujednáním takovým, že bude možný zásah do jejího kódu a komponentu bude možné šířit společně se simulátorem.

- Komponenta musí být multiplatformní, respektive spustitelná na platformách GNU/Linux, Windows, Mac OS.
- Preferovaným implementačním jazykem komponenty je jazyk Java z důvodu jeho využití v ostatních částech simulátoru a tedy možnosti využití schopností vývojářů z celého projektového týmu při případných úpravách.

2.3.2.1 Analýza požadavků, návrh řešení

Z výše sepsaných požadavků víceméně plyne, že by mělo být využito alespoň částečně již existujícího řešení tak, aby množství práce vložené do této komponenty neovlivnilo průběh vytváření ostatních komponent simulátoru. Jakožto komponenta pracující s terminálovým připojením byla její realizace přidělena mě. V provedeném průzkumu existujících řešení jsem nenalezl jediný grafický telnet klient, který by splňoval všechny výše zmíněné požadavky. Nicméně jsem nalezl projekt, jež splňoval převážnou většinu požadavků a u něhož existovala dobrá možnost projekt upravit tak, aby splnil i zbylé požadavky.

Projekt JTA [3] dodává grafického terminálového klienta:

- s funkcí telnet a ssh klienta
- s podporou VT100/ANSI video terminálu
- implementovaného výhradně v jazyce Java
- šířeného pod licencí GNU General Public License 2
- s grafickým rozhraním pouze v anglickém jazyce
- spustitelného jako samostatná aplikace
- spustitelného na různých platformách

2.3.2.2 Realizace

Pro splnění požadavků na vestavěného grafického terminálového klienta bylo třeba:

- dopracovat podporu vícejazyčnosti, tedy možnost nastavit grafické rozhraní s českými popisky
- upravit spouštění programu tak, aby byl terminál vytvořen a spuštěn voláním metody s vhodně nastavenými parametry.

A dále upravit výpis hlášení směrovaných do systémových výstupů(System.out a System.err) tak, aby jej bylo možné odklonit do jiného výstupu. Toto bylo provedeno zejména z důvodu ladění grafického rozhraní simulátoru proto, aby se oddělil ladící výstup těchto dvou komponent.

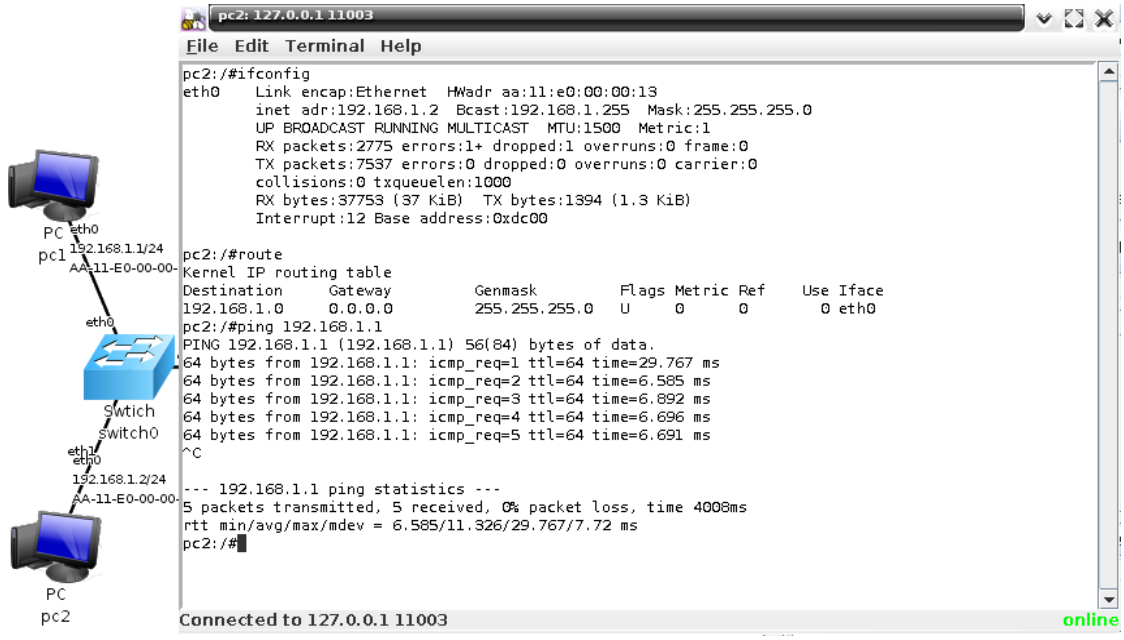
Podporu vícejazyčnosti jsem dopracoval dle doporučení nebo návodu pro lokalizaci aplikací v jazyce Java [7]. Všechny řetězce grafického rozhraní zakomponované přímo do programového kódu jsem extrahoval do souboru se speciálním pojmenováním, ve kterém jsou řetězce uloženy ve tvaru klíč=hodnota. Místa extrakce řetězců jsem pak zaplnil voláním metody objektu třídy ResourceBundle s vhodným argumentem klíče, která ze zmíněného souboru hodnotu načetla. Dále jsem vytvořil soubor se stejnými klíči a lokalizovanými řetězci. Objekt ResourceBundle pak vrací hodnoty dle jazykového nastavení. Výchozím jazykem je angličtina.

Spouštění grafického terminálového klienta jako samostatné aplikace a zároveň jako komponenty grafického rozhraní simulátoru jsem vyřešil následovně. Metoda Main.main, která je volána při spouštění z příkazové řádky případně i jinak, obsahovala fragmet kódu který vytvářel objekt JFrame, tedy okno aplikace. Tento kód jsem extrahoval do metody Main.run, jež vrací právě odkaz na objekt okna. Metodu Main.run pak využívá grafické rozhraní pro vytvoření, spuštění grafického terminálu a získání reference na objekt reprezentující jeho okno. Tomuto oknu nastavuje viditelnost, případně titulek okna a jiné parametry dle potřeby.

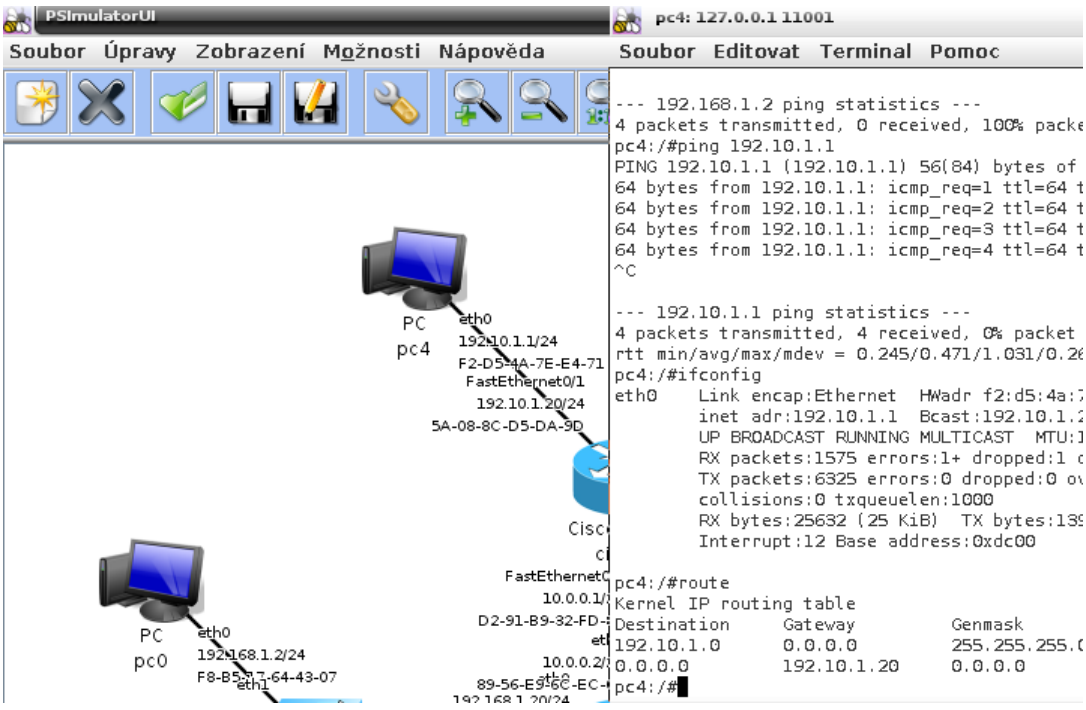
Metoda Main.main pak obsahuje volání metody Main.run s parametry ve formě řetězce předané z příkazové řádky. Tento návrh je zvolen z toho důvodu, že návratová hodnota metody main musí být vždy void. Tento návrh tedy umožňuje spuštění komponenty jak programovým voláním, tak i spuštění jako samostatné aplikace.

Parametry metody Main.run, potažmo i metody Main.main jsou následující:

- -lang cz/en/CZ/EN – volitelný parametr pro nastavení jazyka rozhraní. Výchozí hodnota je en, tedy angličtina
- host – volitelný parametr, pokud není nastaven pak se terminál nepřipojuje.
- port – číslo portu následující za parametrem host, pokud není nastaven použije se výchozí port 23



Obrázek 2.9: Grafický telnet klient. Výchozí jazyk angličtina. Na pozadí grafické rozhraní simulátoru



Obrázek 2.10: Grafický telnet klient s nastaveným jazykem čeština odpovídající jazyku grafického rozhraní simulátoru v pozadí.

2.4 Závěr, shrnutí

S úspěchem bylo realizováno vzdálené textové uživatelské rozhraní určené pro správu virtuálních prvků simulované sítě. Toto rozhraní bylo v omezené míře nejprve realizováno v původní verzi simulátoru a později v rozšířené podobě i v nové verzi simulátoru.

Díky využití knihovny telnetd2 byla implementace zvládnuta v dobrém čase a to i naproti tomu, že použití knihovny neprobíhalo zcela dle původních představ. Knihovnu, potažmo knihovnou dodávaný telnet server, bylo v některých případech nutné hlouběji prozkoumat a pochopit, což si vyžádalo nějaký čas.

Nad rámec zadání byl do projektu navíc začleněn grafický telnet klient viz obrázek 2.9 a 2.10. Grafický klient JTA vytvořený třetí stranou byl vhodně upraven tak, aby vyhovoval prostředí v němž je používán. Klient není nezbytnou součástí simulátoru ani grafického rozhraní, ale jeho přínos pro uživatele a jeho komfort při používání simulátoru je značný.

KAPITOLA 3

Komponenta textového editoru

Myšlenka vytvoření jednoduchého textového editoru v textovém vzdáleném rozhraní vznikla s ohledem na cíl implementovat do simulátoru v budoucnosti služby a jiné programy, které by bylo možné nastavovat skrze konfigurační soubory. Tento způsob konfigurace je běžný zejména v systémech vycházejících z UNIXu, proto by toto bylo možné zejména na virtuálních aktivních prvcích GNU/Linux.

3.1 Požadavky

Vzhledem k cílovému prostředí a účelu nasazení editoru výhradně uvnitř simulátoru, respektive uvnitř terminálového rozhraní virtuálních prvků GNU/Linux, nejsou uvedeny v požadavcích různé pokročilé funkce editace. Není tedy cílem realizovaného editoru být nejefektivnějším a nejkomfortnějším editorem vůbec, ale jen umožnit editaci textového souboru s minimálními požadavky na znalost uživatele a zároveň rozumnou náročností vývoje komponenty. Zjednodušeně lze říci, že je cílem implementovat vzdálenou terminálovou textovou editační plochu, tj. komponentu známou z mnoha toolkitů uživatelského rozhraní jako TextArea, s možností ukládání a načítání doplněnou o oznamovací oblast.

Funkčních požadavků není mnoho, proto je zde uvedu pouze ve formě seznamu:

- editor bude spouštěn z příkazové řádky s jediným argumentem a to se jménem editovaného souboru. Pokud soubor neexistuje nebo jej není možné vytvořit, editor není spuštěn a je vypsána varovná hláška do terminálu.

- editor musí umožňovat pohyb kurzoru a tedy pohyb v editovaném textu pomocí kurzorových kláves(UP/DOWN, LEFT/RIGHT) a pomocí speciálních kláves HOME/END
- editor bude reagovat na klávesy BACKSPACE/ENTER standardním způsobem.
- během editace bude možné upravený text uložit do souboru kombinací kláves CTRL+S a dále pokračovat v editaci.
- konec editace a ukončení editoru s návratem do příkazové řádky bude vyvoláno kombinací kláves CTRL+X
- editor bude disponovat oznamovací oblastí umístěné na konci obrazovky(posledním řádku terminálu) ve které:
  - bude uživatel informován o možnosti uložení textu a ukončení editoru pomocí klávesových kombinací
  - bude uživatel informován o výsledku provedených akcí(zatím pouze o uložení souboru)

Hlavním nefunkčním požadavkem je vývoj komponenty v existujícím terminálovém prostředí síťového simulátoru využívající knihovnu telnetd2. Nefunkční požadavky tak komponenta přebírá v celé své šíři z prostředí ve kterém bude vyvíjena. To je programovací jazyk a podobně. Pokud to nebude nezbytně nutné, neměla by komponenta využívat dalších knihoven třetích stran.

3.2 Analýza

Knihovna telnetd2 kromě implementace telnet serveru poskytuje i toolkit komponent uživatelského rozhraní viz. odstavec 2.1.3.2.2. Při návrhu a realizaci se tedy nabízí využití některých z těchto komponent, proto zde bude proveden jejich průzkum a popis. Konkrétně by měly být využity tyto komponenty:

- třída TitleBar pro zobrazení titulku textového editoru a případné další užitečné informace
- třída StatusBar pro vytvoření oznamovací oblasti
- třída EditArea pro vytvoření editační oblasti.

Třídy TitleBar a StatusBar jsou vcelku nezajímavé. Disponují metodami pro nastavení vzhledu a nastavení zobrazovaného textu. Dále zejména metodou

draw pro jejich překreslení. Tuto metodu je nutné zavolat vždy při změně nastaveného textu tak, aby vhodně nahradil text předchozí.

Zajímavou třídou z pohledu návrhu a implementace je třída EditArea. Tato třída pracuje na podobném principu jako nově vytvořená komponenta příkazové řádky. V smyčce ukončené při načtení specifického znaku postupně zpracovává uživatelský vstup a případně vykresluje výstup. Třída EditArea obsahuje seznam objektů třídy EditLine.

Třída EditLine realizuje pomyslnou vnitřní smyčku s návratovou hodnotou. Vnitřní smyčka obsluhuje vstup/výstup v rámci jedné řádky editoru. Tato smyčka obsluhuje vstup a výstup do chvíle dokud není vyvolána akce pro návrat způsobená:

- načtením klávesy Enter
- načtením kurzorové klávesy UP/DOWN
- načtením kurzorové klávesy LEFT v případě pozice kurzoru na začátku řádky
- načtením kurzorové klávesy RIGHT v případě pozice kurzoru na konci řádky
- načtením klávesy BACKSPACE v případě pozice kurzoru na začátku řádky

Při návratu ze smyčky je vrácena poslední přečtená hodnota. Vnější smyčka implementovaná ve třídě EditArea dle návratové hodnoty buď

- posune kurzor v odpovídajícím směru a případně překreslí zobrazované textové pole od kurzoru dále, pokud šlo například o klávesu Enter. Aktivuje odpovídající řádku textového pole, tj. nastaví referenci na objekt třídy EditLine a tomuto objektu předá řízení. Předání řízení znamená opětovný vstup do vnitřní smyčky, tentokrát však smyčky jiného objektu třídy EditLine.
- provede návrat i z vnější smyčky a řízení pokračuje v nadřazené komponentě, respektive v místě volání metody run na objektu třídy EditArea.

3.3 Návrh a implementace

Z výše zmíněné analýzy plyne, že textový editor bude vrcholnou komponentou skládající se z objektu třídy TitleBar, EditaArea a StatusBar. Úkolem vrcholné komponenty bude zejména inicializace a následné řízení editoru. Řízením editoru je myšlena reakce na akci uložení a ukončení editoru či v budoucnu implementované funkce. Samotná implementace této komponenty se bude odehrávat v balíčku shell.apps.TextEditor ve třídě nazvané TextEditor.

Implementace třídy TextEditor je poměrně jednoduchá. V následujícím seznamu je bodově popsán průběh metody run()

- otevření a načtení souboru
- inicializace objektu TitleBar
- inicializace objektu StatusBar
- inicializace objektu pro textové pole s nastavením načteného textu
- předání řízení objektu textového pole
- vyšetření návratové hodnoty textového pole:
  - uložení textového pole do souboru, vypsání informace do objektu StatusBar a pokračování v předchozím bodě.
  - ukončení textového editoru.

Takto realizovaný editor využívající komponenty TextArea z dodaného toolkitu však nefungoval dobře a také nesplňoval požadavky na pohyb po řádce pomocí kláves Home/End. Rovněž kombinaci kláves CTRL+S a CTRL+X nebylo možné odchytit a obsloužit.

Dále špatná funkčnost editoru se projevovala tím, že i přes volání metody setValue při inicializaci nebyla nastavována počáteční hodnota textového pole. V implementaci této metody jsem později zjistil, že není dokončena a zcela chybí plnění bufferu objekty třídy EditLine. Metoda byla zakončena komentářem: "think of a buffer filling strategy"

Pro dokončení metody však bylo nutné se s třídou EditArea a fungováním celého toolkitu dobře seznámit. Po nějakém čase stráveném studiem toolkitu jsem tedy plnění bufferu do knihovny dopracoval. Prvotní verze plnění bufferu způsobovala špatné umístění kurzoru a špatné vykreslení poslední řádky v textovém poli. Po relativně dlouhém zkoumání a ladění byla objevena chyba v podobě absence nastavení příznaku m_Firstrun na hodnotu false v dopracované

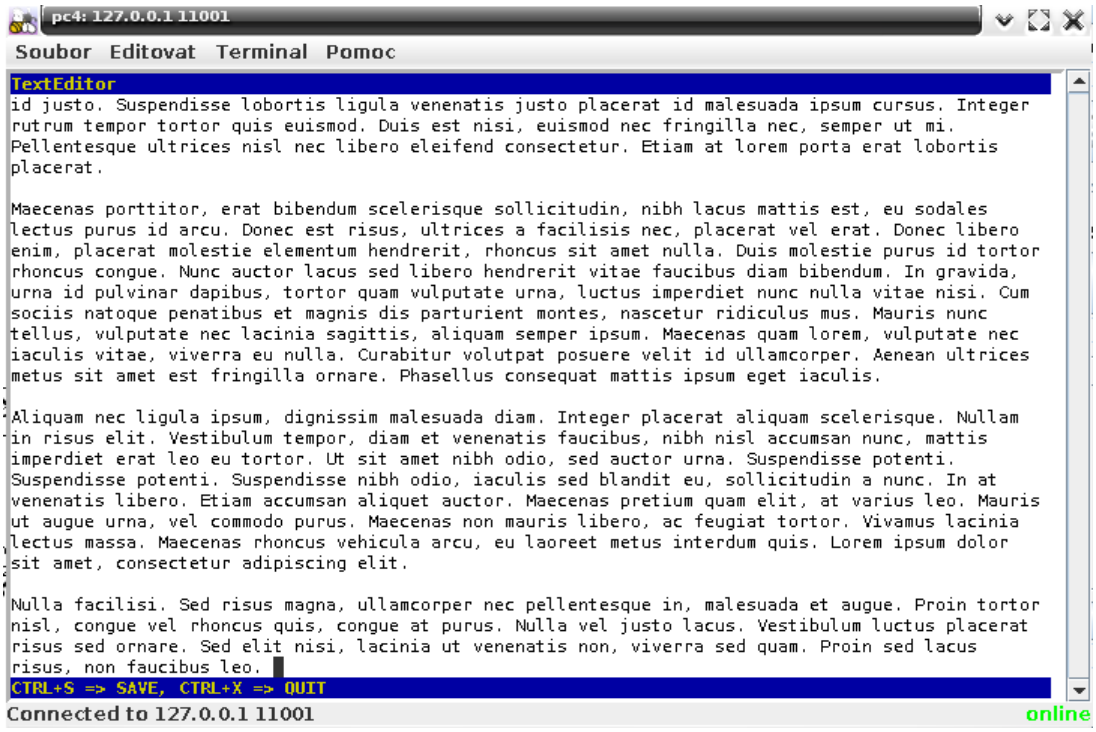
části metody. Tento příznak způsoboval v metodě run() přidání nové řádky do textového pole.

Pro obsluhu kláves Home a End by bylo potřeba zasáhnout do metody run třídy EditLine. Stejně tak pro propagaci načtení CTRL+S a CTRL+X by bylo nutné upravit metodu run ve třídě EditLine i EditArea. Aby nedošlo k přílišnému promíchání originálního kódu z knihovny a nového kódu, vytvořil jsem třídu MyEditArea dědící od třídy EditLine a třídu MyEditLine dědící od třídy EditLine. Obě třídy jsem uložil do balíčku myToolkit uvnitř knihovny telnetd2. Rozšíření tříd spočívalo hlavně v přetížení metod run() s doplněním správných reakcí na klávesy ve vnitřní a vnější smyčce. Dále ve vytvoření metody getExitCode pro získání návratové hodnoty z textového pole.

3.4 Závěr, shrnutí

Všechny funkční i nefunkční požadavky na aplikaci jednoduchého textového editoru realizovaného uvnitř terminálového rozhraní virtuálních prvků simulujících systém GNU/Linux byly úspěšně splněny. Výsledná aplikace zobrazená uvnitř grafického telnet klienta je s ukázkovým textem zobrazena na obrázku 3.1

Při realizaci zároveň došlo k ověření možnosti vytvářet v terminálovém rozhraní pokročilé interaktivní aplikace pomocí toolkitu knihovny telnetd2, nicméně při použití knihovny lze očekávat potřebu zásahu do jejího kódu. Výše popsané problémy při použití této knihovny způsobily větší časovou náročnost implementace aplikace editoru.



Obrázek 3.1: Snímek obrazovky. Grafický telnet klient se spuštěným editorem

KAPITOLA 4

Komponenta jednoduchého souborového systému

Jednoduchý souborový systém pro aktivní virtuální prvky simulující GNU/Linux by měl sloužit společně s aplikací textového editoru jako podpora celkových možností simulátoru v tom smyslu, že pomocí souborového systému bude možné strukturovaně ukládat textové soubory vytvořené nejen v textovém editoru.

Tyto soubory pak budou moci sloužit jako:

- konfigurační soubory pro různé programy(příkazy, služby)
- pomocné soubory - například pro persistentní uložení historie příkazové řádky
- soubory pro zápis skriptů

V GNU/Linux systému se navíc často uplatňuje způsob zápisu konfigurace jako skriptu(posloupnost příkazů). Takto například probíhá konfigurace síťových směrovacích pravidel pomocí iptables. Jednotlivá směrovací pravidla jsou defacto zapsána jako argumenty příkazu iptables. Proto by bylo vhodné tento způsob konfigurace uživateli simulátoru rovněž umožnit.

Umožněním průchodu souborového systému a editací konfiguračních souborů uvnitř simulátoru se virtuální prvky simulující GNU/Linux ještě více přiblíží reálnému použití v praxi, což vzhledem k edukačním záměrům simulátoru je jeden z hlavních cílů simulátoru.

4.1 Funkční požadavky

Vzhledem k zaměření simulátoru na síťové simulační funkce není cílem komponenty souborového systému podrobně implementovat všechny možnosti reálného souborového systému. Cílem vytvářeného souborového systému je vytvořit možnost ukládání textových souborů a poskytnutí základních manipulačních příkazů(rm, ls ...) známých ze systému GNU/Linux.

Simulovaný systém nedisponuje entitou uživatele jako takového, je možné pouze jakési přepínání mezi módy v terminálu. Z tohoto důvodu není ani možné při správě souborů uvažovat něco jako práva přístupu apod. Systém práv by navíc byl pro užití v kontextu síťového simulátoru zbytečný.

Přednost před pokročilými funkcemi souborového systému má jednoduchý přístup k souborům z programového kódu.

Požadována je implementace příkazů s vhodně omezenými parametry tak, aby s příkazy bylo možné provádět alespoň základní operace. Jde o příkazy:

- touch - příkaz pro vytvoření nového prázdného souboru pokud neexistuje
- cd - příkaz pro změnu aktuálního adresáře
- cp - příkaz pro kopii adresáře nebo souboru
- ls - příkaz pro výpis adresáře
- mkdir - příkaz pro vytvoření adresáře
- mv - příkaz pro přesun souboru nebo adresáře
- pwd - příkaz pro výpis aktuální cesty
- rm - příkaz pro smazání souboru nebo adresáře
- cat - příkaz pro výpis obsahu souboru

Implementace souborového systému by měla být platformě nezávislá. Za vhodnou se považuje realizace uzavírající celý souborový systém uvnitř jediného souboru.

4.2 Nefunkční požadavky

Nefunkční požadavky komponenty souborového systému jsou ovlivněny prostředím ve kterém bude komponenta nasazena. Proto všechny tyto požadavky jako například licence, implementační jazyk a tak dále, z tohoto prostředí přejímá. Tato omezení platí i pro případné knihovny třetích stran.

4.3 Analýza

Z funkčních požadavků plyne, že má dojít k realizaci:

- virtuálního souborového systému pro ukládání textových souborů
- příkazů pro manipulaci se soubory ve virtuálních prvcích simulujících GNU/Linux

V části 4.3.1, jež je hlavní části analýzy, se zabývám možností využití některé z dostupných knihoven pro realizaci souborového systému.

Příkazy pro manipulaci se souborovým systémem jsou určeny prozatím výhradně pro simulátor GNU/Linux prvků, proto při jejich realizaci budu spolupracovat s Tomášem Pitřincem, který je odpovědný právě za simulaci těchto prvků. Jeho část práce bude zakomponování manipulačních příkazů do jím implementovaného parseru a do subsystému doplňování příkazů. Moje část práce je implementace příkazů, tj. ovládání souborového systému.

4.3.1 Knihovna virtuálního souborového systému

První věcí vývojáře při řešení problému by mělo být zkoumání pomocných knihoven, protože znovu využívání existujícího kódu zvyšuje efektivnost vývoje. Proto jsem se i já rozhodl prozkoumat možnosti využití existující, odladěné, robustní, a dobře zdokumentované knihovny. Po pečlivém vyhledávání v obsahu webu s klíčovými slovy: "java virtual filesystem" jsem nalezl projekt trueZIP. Jak název knihovny napovídá, knihovna implementuje virtuální souborový systém nad archivními typy souborů, ale rovněž i například nad HTTP adresovatelným prostorem či běžnými typy souborů. Knihovna je modulární a jako archivní typ souborů lze použít formát ZIP, TAR, JAR a jiné. Pro bližší popis možností knihovny doporučuji web projektu [11], kde jsou k nalezení i návody na její použití.

Dovolím si citovat jak je projekt popsán na webu projektu:

> TrueZIP is a Java based virtual file system (VFS) which enables client applications to perform CRUD (Create, Read, Update, Delete) operations on archive files as if they were virtual directories, even with nested archive files in multithreaded environments. Key features are:
>
> - Easy To Use - transparent read/write access to archive files as if they were virtual directories.

- Fast Bulk I/O - provides convenient and powerful methods for fast bulk I/O operations like file or directory tree copying, moving, deleting, traversing
- Pluggable File System Drivers - ships with file system drivers for the file system schemes FILE, HTTP(S), ZIP, TAR and their relatives like JAR, TGZ, TBZ2 etc.
- Thread-Safe - applies fine grained locking, caching and accounting of resources where required so that multiple threads can safely and concurrently read and write entries
- Reliable - code assertions, unit tests, function tests, integration tests and static code analysis

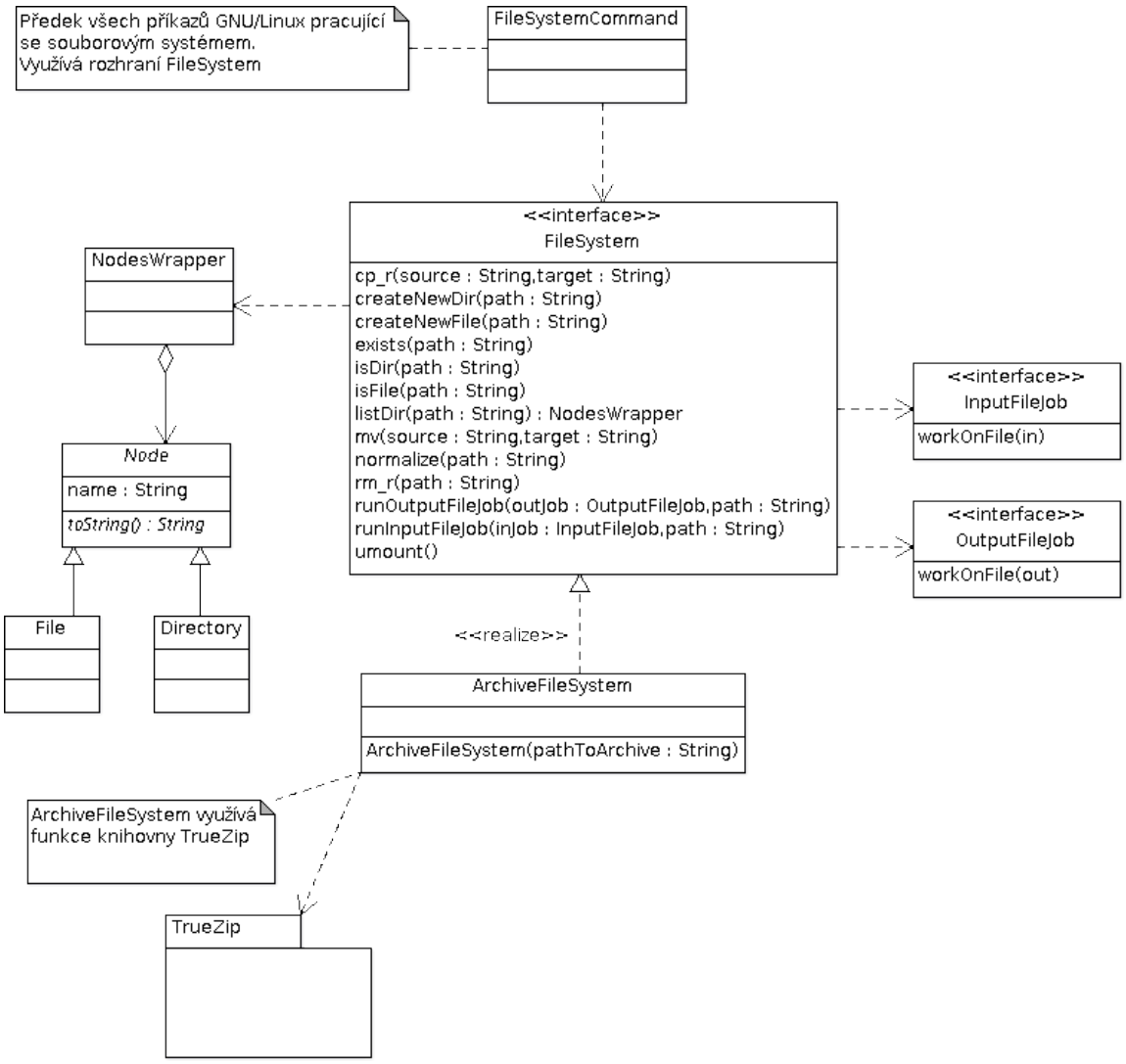
Z uvedených vlastností, rychlých návodů na použití knihovny a dobré dokumentace jsem nabyl dojmu, že by knihovna měla poskytnout funkcionalitu usnadňující nasazení virtuálního systému do projektu a bude tedy v projektu použita.

4.4 Návrh a implementace

Rozhodnutím využít knihovnu TrueZIP došlo k snížení náročnosti implementace virtuálního souborového systému. Knihovna TrueZIP nabízí řadu metod pro operaci s virtuálním systémem, nicméně ne všechny bude třeba v projektu využít. Proto bude vytvořeno rozhraní nazvané FileSystem a třída implementující toto rozhraní nazvané ArchiveFileSystem.

Vytvořením a používáním rozhraní FileSystem, namísto přímého volání funkcí knihovny, dojde k oddělení funkcí knihovny TrueZIP od zbytku simulátoru. Funkce používající knihovnu budou vyčleněny do třídy ArchiveFileSystem a bude tedy v případě potřeby možné knihovnu nahradit bez nutnosti úprav v celém projektu. Aby mohlo být oddělení knihovních funkcí od zbytku simulátoru kompletní, byly vytvořeny třídy reprezentující soubory(třída File) a adresáře(třída Directory) se společným abstraktním předkem(třída Node). Tyto struktury jsou využívány zatím pouze při výpisu obsahu adresáře. V budoucnu pokud to bude třeba, bude vhodné do těchto tříd zakomponovat rekurzivní procházení adresářové struktury. Vztahy zmíněných tříd jsou na obrázku 4.1

Třída ArchiveFileSystem pracující s knihovnou TrueZIP konfiguruje knihovnu tak, aby byl pro VFS(virtuální souborový systém) použit archiv formátu JAR. Je používán ovladač souborového systému JarDriver. Změna používání formátu pro uložení VFS je otázkou změny jediné řádky v kódu. JAR archiv byl zvolen z důvodu doporučení v návodech ke knihovně. Doporučován je proto, že by měl umožnit rychlejší přístup k souborům oproti např. ZIP archivu.



Obrázek 4.1: Zjednodušený třídní model komponenty souborového systému

Abych maximálně zjednodušil API pro čtení a zápis do souboru, vytvořil jsem rozhraní InputFileJob a OutputFileJob. Tato rozhraní s jedinou metodou, jež obsahuje práci se souborem jako výstupním nebo vstupním proudem, jsou implementována třídami, jejichž objekty jsou předávány patřičným metodám objektu třídy FileSystem. Tyto metody mají za úkol otevřít soubor pro čtení nebo zápis a předat řízení zmíněným objektům implementující InputFileJob nebo OutputFileJob. Po návratu řízení("ukončení práce na souboru") z těchto objektů je soubor v bloku finally automaticky uzavřen. Tento způsob řešení je díky automatickému uzavírání file descriptorů méně náchylný na chyby plynoucí z jejich neuzavření. Vztahy zmíněných tříd jsou na obrázku 4.1

Implementace samotných příkazů pro manipulaci se soubory znamenala vhodné volání funkcí z rozhraní FileSystem. V implementaci příkazů se například, v případě relativní cesty, provádí za pomoci cesty aktuálního adresáře sestavení absolutní cesty. Implementace příkazů je uložena v balíčku commands.linux.filesystem.

4.5 Závěr, shrnutí

Za využití vhodné knihovny byl realizován jednoduchý souborový systém s manipulačními příkazy implementovanými pro prvky simulující GNU/Linux. Souborový systém je prozatím úspěšně využíván pro ukládání historie zadaných příkazů, tak aby mohly být příkazy při dalším připojení uživatele k prvku opět načteny.

Souborový systém spolu manipulačními příkazy a jednoduchým textovým editorem považuji za řešení, které bude možné bez velkých změn využít v budoucnosti pro práci s konfiguračními soubory programů a služeb provozovaných na virtuálních prvcích simulujících GNU/Linux.

KAPITOLA 5

Komponenta pro persistentní ukládání konfigurace virtuální sítě

Jak bylo již zmíněno v části 1.1, simulátor se skládá ze dvou do značné míry nezávislých částí. Jednou ze sdílených částí mezi frontendem a backendem je struktura pro uložení topologie sítě a základní konfiguraci virtuálních prvků. Právě vytváření a používání této struktury se věnuje tato kapitola.

Frontend i backend jsou vytvářeny v jazyce JAVA, proto i zmíněná struktura je vytvořena a složena ze tříd implementovaných v tomto jazyce. Struktura je mezi frontendem a backendem předávána formou persistentně ukládaného souboru do kterého je serializována.

5.1 Model konfigurace sítě

Vývoj backendu(simulátor) a frontendu(vizualizační rozhraní) neprobíhal simultánně. Vizualizační část a její vnitřní struktury pro popis sítě a virtuálních prvků byly stvořeny dříve, než započal vývoj simulačního jádra. Nicméně tyto struktury částečně nevyhovovaly požadavku na jejich uložení a sdílení mezi projekty. Část dat byla závislá na datech používaných při vizualizaci sítě a jiných operací. Struktury nemohly tedy být použity pro persistentní ukládání konfigurace virtuální sítě tak, aby mohly být sdíleny. Proto bylo rozhodnuto o jejich přepracování a přemístění do sdíleného projektu nazvaného "Shared". Kompilací projektu Shared vznikne knihovna struktur (.jar soubor) a operací pro jejich serializaci do souboru. Tato knihovna je importována a používána v obou projektech, čímž je zajištěna alespoň částečně konzistence formátu dat.

Přepracovaný třídní model je dílem především Martina Švihlíka, jakožto autora vizualizační části a původních struktur. Já jsem se na tvorbě nového modelu podílel jen částečně a nejvíce pak s ohledem na serializaci struktur do souboru, proto zde model detailněji popisovat nebudu. Model je umístěn v balíčku shared.Components

5.2 Persistence modelu konfigurace sítě

5.2.1 Výběr formátu

Data o síti a jejích prvcích, která mají být ukládána, nejsou nikterak zvláště složitá ani rozsáhlá a tak formát uložených dat v souboru mohl být vybrán téměř libovolný. Preferovány byly při výběru rozšířené a uznávané formáty čitelné v textové podobě. Rozšířenost formátu měla zajistit dobrou podporu od existujících knihoven. Čitelná forma měla zajistit snadnou zpětnou kontrolu uložených dat vývojářem a otevřenost formátu pro jeho další případné využití.

Nakonec byl zvolen formát XML z těchto důvodů:

- rozšířený a uznávaný formát
- "human readable" - čitelný v textové podobě
- dobrá znalost formátu mezi vývojáři v projektovém týmu, možnost spolupráce
- použití v předchozí verzi simulátoru

Pokud by formát XML z nějakého důvodu v budoucnu nevyhovoval, není větší problém ho zaměnit za jiný, protože struktura dat je popsána formou tzv. POJO(plain old java object). Bylo by tedy pouze nutné vytvořit nový kód serializující strukturu do jiného formátu.

5.2.2 Nástroje serializace

Serializace java objektů do XML byl a je tak běžný problém, že pro něj vzniklo i několik desítek pomocných knihoven. Problémem tohoto množství pak je správný výběr. Z několika diskusí nad výběrem knihovny pro ukládání a načítání java objektů do/z XML dokumentu jsem nabyl dojmu, že většinu problémů souvisejících se serializací dokáže spolehlivě řešit knihovna JAXB, jež je distribuována s prostředím JSE 1.7, a není tedy třeba importovat do projektu knihovny třetích stran.

Strukturu dat našeho projektu a požadavek na jejich serializaci nepovažuji za nijak výjimečný a z tohoto důvodu jsem tedy zvolil knihovnu JAXB, která by měla vzhledem k předchozímu tvrzení poskytnout dobré řešení. Konkrétně jsem využil JAXB 2.0 přítomné v JDK 1.7.

Knihovna JAXB(Java Architecture for XML Binding) jak její název napovídá pracuje na základě spojování(binding) vazeb mezi XML daty a java objekty. Pro toto spojování však potřebuje nějaký předpis jakým způsobem vazbu provést. Tento předpis je zapsán formou anotací tříd zpracovávaných objektů. V případě, že nejsou entity popsány třídami, je možnost třídy generovat pomocí utility "xcj" z dodaného XML Schema. Opačný postup je rovněž možný, tj. generování XML Schema.

5.2.3 Serializace modelu konfigurace sítě

V tomto textu nebudou popsány všechny možnosti použití knihovny JAXB, toto ani není cílem této práce. V následujících odstavcích budou popsány použité anotace tříd a použití knihovny JAXB verze 2.0 pro načtení a uložení objektu z XML tak, aby případný pokračovatel projektu mohl tohoto popisu využít.

Kořen XML dokumentu, potažmo modelu sítě, tvoří třída NetworkModel. Toto značí použití anotace @XmlRootElement před názvem třídy.

Třída, jejíž objekty mají být serializované, musí obsahovat bezparametrický konstruktor. Jednotlivé atributy třídy není ve většině případů nutné anotovat. Pouze je nutné, aby ke každému serializovanému atributu existovala dvojice tzv. setter a getter metod. Pokud jsou atributy primitivním typem nebo jsou vlastněny objektem, pak jsou do výsledného XML dokumenty zapsány jako potomci s názvem elementu odpovídající jejich názvu uvnitř třídy. Pokud je požadavek na změnu tohoto jména, je možné využít anotace @XmlElement(name = "mojeJmeno").

Data nejsou vzhledem k svému charakteru ukládána čistě ve stromové struktuře a je tedy nutné rozhodnout o vlastnictví objektu(v XML vztah elementů rodič-potomek) a vyvarovat se cyklů. Objekt může vlastnit jen jeden jiný objekt. Pokud objekt není vlastněn třídou a zároveň je jejím atributem, pak jde o referenci na objekt(v XML Schema popsaný konstruktem keyref). Vlastnictví objektu jako typ určující způsob pro serializaci do XML je výchozí v knihovně JAXB. Pokud je požadována serializace vztahu reference, pak musí třída odkazovaného objektu disponovat identifikátorem a odkazovaný atribut musí být anotován anotací @XmlIDREF. Identifikátor se anotuje anotací @XmlID a musí být typu String.

Klíčové prvky v modelu sice disponují identifikátorem vytvářeným při editaci sítě v grafickém editoru, ale jsou typu Integer. Proto jsem do potřebných třídy dopracoval getter a setter metody pojmenované getIDAsString respektive setIDAsString. Ke getter metodě jsem pak doplnil anotaci @XmlAttribute signalizující nový atribut pro serializaci a anotaci @XmlID označující identifikátor objektu.

Omezením pro automatickou serializaci kolekcí je, že atribut značící kolekci musí být jedním ze základních typů(Map, List). Serializace kolekcí pod atributem typu HashMap, LinkedList a podobné nejsou knihovnou automaticky podporovány. Atribut typu Map však může nabývat hodnoty typu HashMap a serializace bude úspěšná.

Samotné provedení serializace/de-serializace objektů tříd s vhodnými anotacemi je z pohledu vývojáře velice jednoduché. Metody provádějící uložení a načtení modelu do/z XML jsou uloženy ve sdíleném projektu v balíčku shared.Serializer. Tím je opět určitým způsobem zvýšena úroveň udržování konzistence práce s daty mezi dvěma projekty.

5.3 Závěr, shrnutí

S použitím knihovny JAXB byla implementována persistence objektového modelu sítě do čitelného a otevřeného formátu XML. Vytvořený způsob ukládání a načítání topologie sítě je snadno upravitelný a dlouhodobě udržitelný. Do modelu lze téměř libovolně přidávat nové atributy a ty budou automaticky přidány i do procesu serializace a výsledného XML. Uvedený způsob persistence za využití knihovny JAXB a při její dobré znalosti je ve své podstatě relativně jednoduchý.

KAPITOLA 6

Komponenta napojení simulátoru na grafické simulační rozhraní

Jak bylo již zmíněno v části 1.1, simulátor se skládá ze dvou do značné míry nezávislých částí. Jednou ze sdílených částí mezi frontendem a backendem je i struktura pro zasílání informací o tocích dat ve virtuální síti. Tato struktura je uložena ve sdíleném projektu nazvaném Shared. Konkrétně jde o balíčky SimulatorEvents a telnetConfig.

6.1 Reprezentace a výměna dat

Balíček SimulatorEvents obsahuje třídy reprezentující jednotlivé události o tocích, paketech přenášených ve virtuální síti.

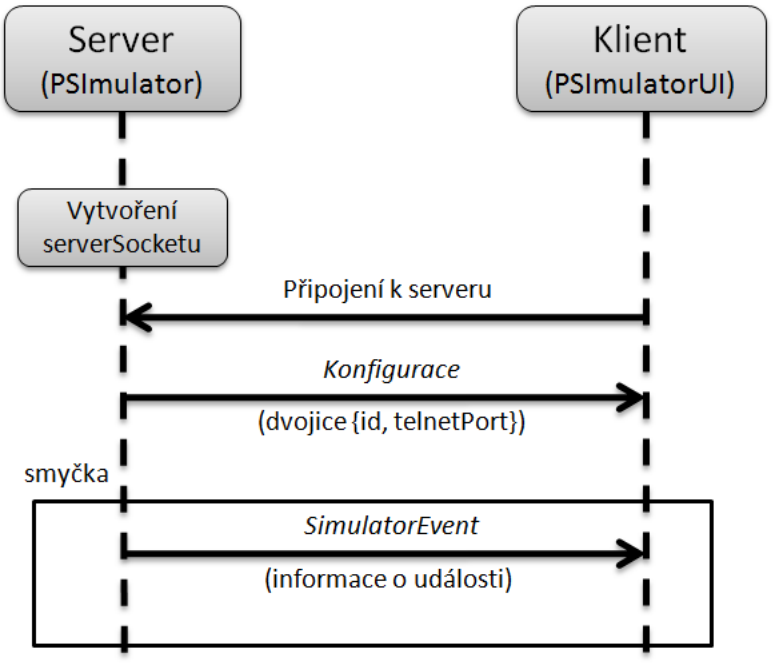
Nejdůležitější je v tomto balíčku třída SimulatorEvent, která obsahuje informace o:

- zdroji
- cíli
- časové značce
- obsahu
- typu paketu

Zdroj a cíl jsou identifikovány v rámci sítě jedinečným identifikátorem, nikoliv IP adresou či podobně. Identifikátor je celé číslo. Balíček rovněž obsahuje třídu SimulatorEventsSerializerXML s metodami pro uložení a načtení událostí do XML dokumentu.

Balíček telnetConfig obsahuje struktury pro reprezentaci dvojice "identifikátor virtuálního prvku" - "telnet port prvku".

Pro přenos struktur mezi simulátorem a grafickým vizualizačním rozhraním je použito TCP/IP spojení. Simulátor(backend) pracuje jako server (java.net.ServerSocket) a grafické rozhraní(frontend) jako klient(Socket). Komunikační schéma je poměrně jednoduché. Backend na začátku komunikace zašle strukturu s obsahem přiřazení telnet portů k jednotlivým prvkům. Tohoto přiřazení může později grafické rozhraní využít při spuštění zabudovaného grafického telnet klienta, který se díky znalosti IP adresy a portu může automaticky připojit. Další zasílané struktury jsou pak události s informacemi o tocích v síti. Schéma komunikace je zobrazeno na obrázku 6.1



Obrázek 6.1: Schéma komunikace simulátoru a vizualizačního rozhraní. Autor: Martin Švihlík

6.2 Server pro připojení vizualizačního rozhraní

V této sekci bych chtěl popsat návrh a implementaci serveru na něž se budou připojovat vizualizační rozhraní. Těmto rozhraním budou zasílány informace o tocích v síti.

Předpoklady, omezení dle kterých by měl server měl řídit jsou uvedeny v následujícím seznamu:

- měl by pracovat dle popsaného komunikačního schéma zobrazeného na obrázku 6.1.
- vstupem zasílaných událostí je metoda listen z rozhraní LoggerListener, které musí vstupní část serveru implementovat
- metoda pro předání vstupu(událostí) do serveru by měla být maximálně implementačně a výpočetně jednoduchá, např. vložení vstupu do fronty. Tento požadavek je vytvořen z důvodu minimálního ovlivnění simulačních funkcí při rozesílání událostí k vizualizaci.
- předávaný vstup není ve tvaru struktury pro odeslání po síti, před jeho odesláním musí dojít k jeho překladu.
- k serveru by mělo být možné vícenásobné připojení z různých vizualizačních rozhraní.

Z těchto předpokladů plyne, že server by měl využívat tři vlákna:

1. První vlákno pro naslouchání na určeném portu.
2. Druhé vlákno pro příjem a překlad událostí do tvaru určeného k přenosu.
3. Třetí vlákno pro odesílání událostí všem připojeným vizualizačním rozhraním.

První vlákno je implementováno ve třídě EventServer. Toto vlákno ve smyčce volá blokovací metodu accept() na objektu třídy ServerSocket. Po získání objektu třídy Socket vytvoří vlákno objekt třídy ClientSession. Na tomto objektu zavolá metodu initCommunication() pro zahájení komunikace obnášející i zaslání struktur s přiřazením telnet portů. Objekt třídy ClientSession je přidán do seznamu, který vlastní objekt reprezentující třetí vlákno. Smyčka se opakuje.

Druhé vlákno je implementováno ve třídě PacketTranslator. Tato třída implementuje rozhraní LoggerListener, potažmo implementuje metodu listen, v níž předává obdržený objekt události do své vnitřní blokující fronty. Pracující

druhé vlákno ve smyčce načítá objekty z fronty, provádí překlad objektů do formy pro odeslání a ukládá je do odesílací fronty třetího vlákna. Metodu listen pro předání objektu události nepřímo volá simulační jádro. Více o tom, kdy je událost emitována, lze nalézt v pracích kolegů Stanislava Řeháka [13] a Tomáše Pitřince [5].

Třetí vlákno je implementováno ve třídě EventsListener. Toto vlákno ve smyčce načítá a odebírá objekty z blokující fronty pro odeslání. Načtený objekt je odeslán skrze metody všech objektů třídy ClientSession patřičným připojeným vizualizačním rozhraním. Pro odesílání všech objektů se používá proud ObjectOutputStream získaný z objektu třídy Socket vytvořeného při připojení uživatele. Při odpojení vizualizačního rozhraní je objekt třídy ClientSession odebrán ze seznamu aktuálně připojených.

6.3 Závěr, shrnutí

Bylo úspěšně realizováno napojení simulátoru na jeho vizualizační rozhraní pomocí TCP/IP přenosu. Toto napojení umožňuje vícenásobné napojení vizualizačního rozhraní na jednu instanci simulátoru, čehož lze využít v řadě různých případů užití. Napojení je v ohledu na změnu struktur používaných při komunikaci snadno upravitelné díky sdíleným strukturám v projektu Shared.

KAPITOLA 7

Testování simulátoru

7.1 Uživatelské testování

Uživatelské testování se konalo v rámci třech laboratorních cvičeních předmětu BI-PSI na FIT ČVUT dne 11.4.2012. Testování probíhalo formou plnění předem vytvořených uživatelských scénářů. Získávání zpětné vazby od uživatelů pak probíhalo přímým rozhovorem nebo anonymním vyplněním dotazníku.

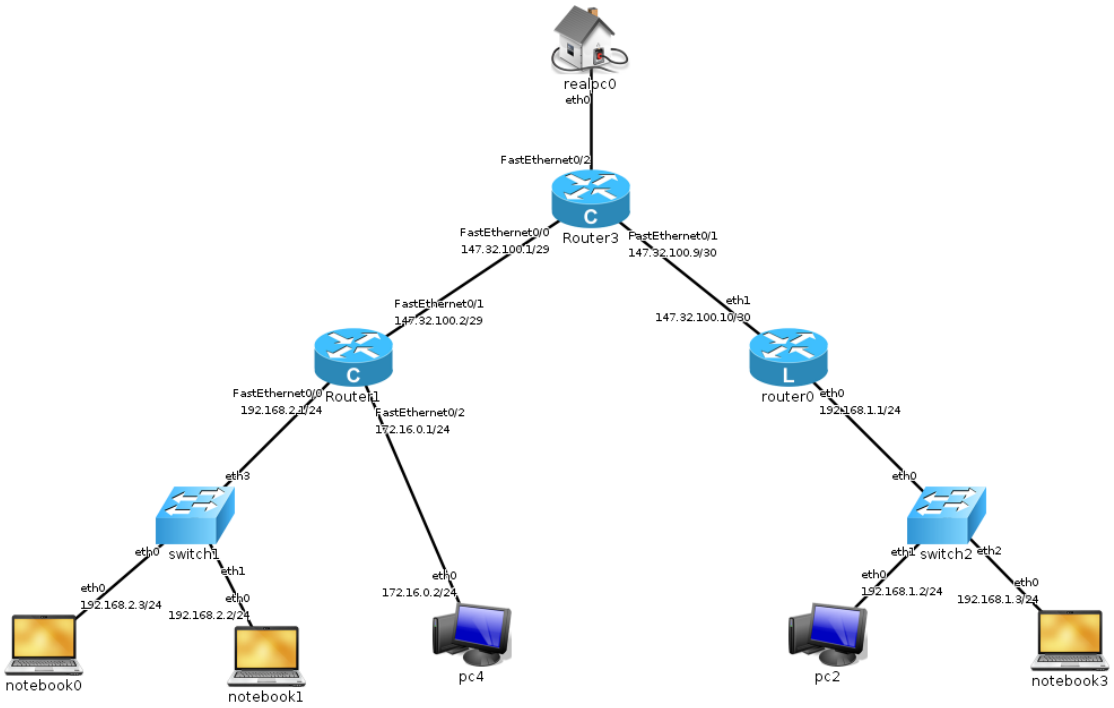
Scénář a dotazník pro použití grafického rozhraní byl vytvořen kolegou Martinem Švihlíkem. Scénář konfigurace vzorové sítě 7.1.1 byl vytvořen Stanislavem Řehákem a Tomášem Pitřincem. Z mé strany, s ohledem na charakter mé části práce na projektu, nebyl žádný scénář potřeba, neboť k otestování mé části práce na projektu by mělo dojít během plnění obou zmíněných scénářů. Každý z vývojářů se tedy víceméně zajímal o fungování své části programu.

7.1.1 Testovací síť

Studenti dostali ke konfiguraci kolegy účelně vytvořenou testovací síť. [3]. Grafické znázornění sítě je na obrázku 7.1

Testovací síť obsahuje jak prvky simulující CISCO a GNU/Linux, tak i prvky bez terminálového přístupu. Plněním výše zmíněných scénářů by tak mělo dojít k otestování převážné většiny funkcí terminálového rozhraní.

[3] Testovací síť je ke stažení na http://code.google.com/p/psimulator/source/browse/trunk/psimulator2/src/xml/na_testovani.xml?r=1079



Obrázek 7.1: Schéma testovací sítě tak, jak je graficky zobrazené v simulátoru. Autor: Stanislav Řehák

7.1.2 Uživatelé, prostředí

Testování se během třech cvičení zúčastnilo přesně 20 studentů druhého ročníku bakalářského programu. Jde tedy o hlavní cílovou skupinu vytvářeného simulátoru. Znalosti studentů o počítačových sítích a konfiguraci síťových prvků byly značně různé. Testování se účastnili studenti s certifikáty Cisco Networking Academy, ale rovněž i úplní začátečníci se zkušenostmi pouze z právě absolvovaného kurzu BI-PSI. Zejména studenti s certifikáty Cisco Networking Academy srovnávali námi vytvořený simulátor se známým simulátorem Cisco Packet Tracer.

Studenti pro spuštění simulátoru používali vlastních přenosných počítačů různých parametrů(rozlišení obrazovky, výkon) a s různými operačními systémy (GNU/Linux, Windows XP, Windows 7). Největší bariérou při spouštění simulátoru (backendu i frontendu) byla nutnost přítomnosti běhového prostředí JAVA 1.7. Tento požadavek je uveden v instalační příručce. Nicméně většina studentů požadavek nečetla nebo nerespektovala, což vyústilo v hlášení "psimulator2.Main was not found". Studentům jsme správnou verzi běhového prostředí pomohli nainstalovat. Podpora starší verze javy 1.6 bude společností Oracle časem ukončena a proběhne automatická aktualizace na verzi 1.7.

Jeden ze studentů na svém počítači využil JRE z projektu OpenJDK pro spuštění a bezchybný běh simulátoru. Toto je dobrá zpráva, protože to umožní běh programu i v licenčně svobodnějším prostředí.

7.1.3 Vyhodnocení

V následujících odstavcích je popsáno vyhodnocení uživatelského testování s ohledem na mnou realizované části simulátoru. Vyhodnocení uživatelského testování z pohledu ostatních částí simulátoru lze nalézt v pracích kolegů.

Během testování simulátoru jsem podobně jako mí kolegové asistoval a řešil problémy uživatelů. Žádný vážný problém z mého pohledu při testování nenastal. Nastalo však několik méně závažných.

Prvním menším ohlášeným nedostatkem bylo málo výstižné hlášení při startu simulátoru o portech, na kterých naslouchají telnet servery jednotlivých virtuálních prvků. Na tento nedostatek upozornili nezávisle na sobě dva studenti. Ostatní studenti dle mého pozorování využívali zabudovaného grafického telnet klienta, který se automaticky(stiskem tlačítka) připojuje a oni tak nejsou nuceni port zjišťovat. Hlášení bylo upraveno do více výstižné podoby a nemělo by tak již činit problém porozumět výpisu.

Dalším ohlášeným nedostatkem byla nefunkčnost příkazové řádky v telnet klientu dodávaného s operačním systémem Windows. Nefunkčnost se projevovala absencí reakce na klávesu Enter. Nedostatek nahlásil pouze jediný student. Nedostatek si vysvětluji tím, že tento klient odesílá signální znak Enteru v jiném tvaru než všichni ostatní klienti. Nedostatek jsem vzhledem ke správné funkčnosti terminálu v mnoha jiných různých klientech a zejména pak v zabudovaném klientu ohodnotil nízkou prioritou. K vyřešení tohoto problému jsem zatím nepřistoupil.

Dalším, spíše kosmetickým, nedostatkem bylo netradiční umístění posuvníku (scrollbaru) v zabudovaném telnet klientu na levé straně. Posuvník jsem přemístil na tradiční pravou stranu okna.

Další nápadem na vylepšení pocházející od uživatele je ovládání posuvníku a tedy výpisu terminálu pomocí kolečka myši. Tento způsob ovládání nebyl ve vybraném klientu podporován a byl tedy dodatečně dopracován.

Zřejmě nejvážnějším nedostatkem je chyba projevující se "zamrznutím" okna zabudovaného telnet klienta. Tato chyba vzniká výhradně na platformě Windows a výhradně při manipulaci(výběr textu) s myší v okně terminálu. Chybu se mi prozatím nepodařilo naleznout. Klient je napsán pro Javu verze nižší než používaná verze 1.7. Je tedy možné, že jde o chybu způsobenou nekompatibilitou běhového prostředí na platformě Windows.

Kromě zmíněných nedostatků se objevilo i několik pochvalných komentářů ohledně přehlednosti připojování se k virtuálním prvkům pomocí zabudovaného telnet klienta.

7.2 Automatizované testování

Vývoj simulátoru probíhal velice rapidním tempem. Proto, aby bylo snazší odhalení chyby vzniklé častými úpravami kódu, bylo potřeba vytvořit systém testování již existujících funkcí simulátoru.

V praxi se uplatňují dva způsoby nebo dvě úrovně testování. Prvním z nich je tzv. white-box testing. White-box testování má za úkol otestovat a sledovat vnitřní struktury a procesy software. Oproti tomu druhá úroveň testování tzv. black-box testing má za úkol otestovat software z pohledu plnění jeho funkcí jako celku, tj. defacto z pohledu uživatele.

V projektu jsme se rozhodli uplatnit pouze black-box testování zejména s ohledem na jeho nižší časové nároky na realizaci testů a jeho dostatečnou vypovídající sílu pro naše použití.

Se simulátorem lze z vnějšku interagovat pouze použitím vzdáleného textového rozhraní a proto se toto rozhraní stalo pomyslným vstupem "black boxu". Sledované výstupy z "black-boxu" jsou dva:

- přímý výstup simulátoru ve formě hlášení
- výstup do textového rozhraní uživatele

Prozatím bylo pro zjednodušení zvoleno sledování pouze přímého výstupu simulátoru. V budoucnu lze testování rozšířit i o sledování výstupu do textového rozhraní připojeného uživatele. Pro realizaci alespoň částečně automatizovaných testů bylo nutné:

- sestavit testovací scénáře
- nalézt řešení jak vykonat testovací scénáře s minimálním zásahem obsluhy a odchytit případné chyby

7.2.1 Testovací scénáře

Testovací scénáře by měli prověřit alespoň část funkcionality simulátoru přístupné skrze vzdálenou příkazovou řádku. Požádal jsem tedy kolegy o vytvoření popisných

scénářů, které se pokusím alespoň částečně automatizovat. Konkrétní znění scénářů je uloženo v příloze C. Tyto scénáře při jejich vykonávání současně testují i komponentu pro načítání konfigurace sítě ze souboru, protože kdyby došlo k chybě při načítání, tak se s velkou pravděpodobností objeví i chyba při spouštění scénářů. Tento způsob testování nepokrývá 100% napsaného kódu, ale pro účely vývoje simulátoru dostačuje.

7.2.2 Nástroj expect

Při hledání způsobu jak automatizovat proces připojení se telnet klientem k virtuálnímu prvku a postupné zadávání příkazů jsem objevil nástroj expect. Nástroj expect je, dle popisu v manuálových stránkách, program "mluvící" s ostatními interaktivními programy dle předem připraveného skriptu. Připravený skript obsahuje především návěští/příkazy:

- expect - testující výstup interaktivního programu. Skript nepokračuje do doby splnění regulárního výrazu či jiného způsobu zadání výstupu.
- send - zasílající předem daný vstup do interaktivního programu.

A mnoho dalších příkazů jako např. sleep apod.. Kompletní přehled lze nalézt v manuálových stránkách programu.

Program expect je zaručeně spustitelný na platformě GNU/Linux. Na ostatních platformách by měly být funkční některé alternativy, ty však vyzkoušené nemám. V projektovém týmu jsme využívali pouze distribuce GNU/Linux.

Pro snazší vytváření skriptů existuje program autoexpect. Tento program po spuštění s parametrem jména výstupního souboru(skriptu) a příkazem pro spuštění interaktivního programu, zaznamenává akce provedené uživatelem a výstup interaktivního programu. Na základě tohoto záznamu pak generuje patřičný skript. Způsob generování expect skriptu je závislý na mnoha parametrech, které lze nalézt v dodávaných manuálových stránkách programu. Zaznamenávání je ukončeno ve chvíli ukončení interaktivního programu. Vytvořený skript je čitelný a lze jej na základě znalosti jednotlivých příkazů dále upravovat.

7.2.3 Vytvoření skriptů

Skripty byly vytvořeny využitím nástroje autoexpect popsaného v předchozím odstavci. Testy nebyly nijak dále upravovány. Konkrétně byl příkaz spuštěn takto:
autoexpect -p -f pc5.exp telnet localhost 11000

Parametr -p značí očekávání "promptu" vypsaného v terminálu. Po jeho vypsání skript pokračuje v interpretaci. Tento mód je vhodný pro automatizaci příkazů, které v každém běhu vypisují různé hodnoty. Text označující prompt program autoexpect automaticky detekuje.

Parametr -f značí jméno výstupního souboru. Dále následuje příkaz pro spuštění interaktivního programu. V tomto případě BSD telnet klienta s parametry localhost a port 11000.

7.2.4 Použití testů, závěr

Za využití nástroje autoexpect byly vytvořeny expect skripty realizující výše popsané scénáře. Pro běh skriptu je nutné mít spuštěný simulátor s patřičnou sítí a aktivními prvky naslouchající na odpovídajících portech tak, aby bylo možné ustanovit spojení se zařízením odpovídajícím scénáři. Pro úplnou automatizaci testů by bylo potřeba vytvořit obalující skript, který by:

1. spustil simulátor s patřičnou sítí a odchytával hlášení ze simulátoru
2. odchytil hodnoty portů na kterých naslouchají telnet servery aktivních prvků
3. postupně spouštěl jednotlivé scénáře(expect skripty) s parametrem portu získaného v bodě 2
4. v odchycených hláškách ze simulátoru hledal chybová hlášení a klíčová slova jako "Exception" a podobně.

Prozatím nám při testování vyhovoval semi-automatický průběh testu a proto nebyla vytvořena jeho plně automatická verze. Implementace plně automatické verze testování je dobrým námětem na vylepšení pro případné pokračovatele v projektu. Rovněž pak rozšíření testovacích scénářů.

Závěr

Předložené zadání, vytvoření podpůrných komponent síťového simulátoru, se mi podařilo splnit. Navíc se mi podařilo vytvořit i funkce, které nebyly přímo vyžadovány, ale tvoří přidanou hodnotu simulátoru a zpříjemňují uživateli práci se simulátorem jako celkem.

Zadání kolegů byla rovněž úspěšně splněna, čímž vznikl simulátor sítě s virtuálními prvky simulující GNU/Linux a CISCO IOS. Dále vzniklo grafické vizualizační rozhraní umožňující editaci topologie sítě a vizualizaci přenosu paketů mezi prvky.

Jako u každého software, i v případě mnou vytvářených částí simulátoru je možné najít místa k vylepšení a navázaní vývoje například v rámci dalších diplomových prací. Inspirací pro drobná vylepšení mohou být údaje uvedené v části práce v níž popisuji uživatelské testování. Navázat na vývoj simulátoru jako celku lze například tvorbou nových síťových služeb a protokolů uvnitř virtuální sítě, které by využily potenciálu vytvořeného souborového systému a textového editoru.

Během vývoje jsem se seznámil s novými nástroji a knihovnami, které snad budu moci v budoucnosti uplatnit. Za dobrou zkušenost rovněž považuji spolupráci s kolegy v rámci projektu. Spolupráce s nimi byla na velice dobré úrovni. Dobrá spolupráce a dělení úkolů vedlo k efektivnímu vývoji a snadné integraci vytvářených komponent v rámci projektu.

Pokud bych měl rozdělit 100 bodů dle náročnosti realizace jednotlivých komponent, pak bych dělil takto:

- realizace vzdáleného textového rozhraní - 45 bodů
- komponenta textového editoru v textovém rozhraní - 15 bodů
- komponenta virtuálního souborového systému - 20 bodů
- persistentní ukládání topologie sítě - 10 bodů
- napojení simulátoru na vizualizační rozhraní - 10 bodů

Literatura

[1] Stránky předmětu BI-PSI Počítačové sítě. https://edux.fit.cvut.cz/courses/BI-PSI/, [Cited 2012-04-04].

[2] Stránky projektu CHARVA. http://www.pitman.co.za/projects/charva/Screenshots.html, [Cited 2012-04-04].

[3] Stránky projektu grafického telnet/VT100 klienta JTA. http://javatelnet.org, [Cited 2012-04-04].

[4] Pitřinec, T.: Bakalářská práce: Simulátor virtuální počítačové sítě Linux. https://dip.felk.cvut.cz/browse/details.php?f=F3&d=K13136&y=2010&a=pitritom&t=bach, 2010, [Cited 2012-04-04].

[5] Pitřinec, T.: Magisterská práce: Simulátor virtuální počítačové sítě Linux. 2012, [Cited 2012-04-28].

[6] Repozitář první verze síťového simulátoru PSImulator1. http://code.google.com/p/psimulator/source/browse/#svn%2Ftags%2Fbakalarka%2Fpsimulator-1.1, [Cited 2012-04-04].

[7] Lokalizace Java aplikací využitím Resource Bundle. http://java.sun.com/developer/technicalArticles/Intl/ResourceBundles/, [Cited 2012-04-04].

[8] Stránky projektu rlwrap. http://freecode.com/projects/rlwrap, [Cited 2012-04-04].

[9] Stránky projektu telnetd2. http://telnetd.sourceforge.net/, [Cited 2012-04-04].

[10] Projekty užívající telnetd2 knihovny. http://telnetd.sourceforge.net/project_users.html, [Cited 2012-04-04].

[11] Stránky projektu knihovny truezip. http://truezip.java.net/, [Cited 2012-04-04].

[12] Řehák, S.: Bakalářská práce: Simulátor virtuální počítačové sítě Cisco. https://dip.felk.cvut.cz/browse/details.php?f=F3&d=K13136&y=2010&a=rehaksta&t=bach, 2010, [Cited 2012-04-04].

[13] Řehák, S.: Magisterská práce: Síťový simulátor pro výukové účely na bázi směrovačů CISCO. 2012, [Cited 2012-04-28].

[14] Švihlík, M.: Magisterská práce: Vizualizace virtuální počítačové sítě. https://dip.felk.cvut.cz/browse/details.php?f=F8&d=K102&y=2012&a=svihlma1&t=dipl, 2012, [Cited 2012-04-28].

PŘÍLOHA A

Seznam použitých zkratek

GUI Graphical user interface

TUI Text user interface

XML Extensible markup language

JRE Java Runtime Enviroment

BI-PSI Počítačové sítě

DHCP Dynamic Host Configuration Protocol

DNS Domain Name System

FTP File Transfer Protocol

HTTP Hypertext Transfer Protocol

ICMP Internet Control Message Protocol

IDE Integrated Development Environment

IP Internet Protocol

SSH Secure Shell

TCP Transmission Control Protocol

TTL Time To Live

UDP User Datagram Protocol

UML Unified modeling language

PŘÍLOHA B

Instalační a uživatelská příručka

B.1 Systémové požadavky

B.1.1 Java Runtime Environment version 7+

Pro spuštění simulátoru je nutné stáhnout JRE verze 7 nebo vyšší:
http://www.oracle.com/technetwork/java/javase/downloads/

B.1.2 Telnet klient

Pro připojení k aktivním prvkům simulující GNU/Linux nebo CISCO je možné použít zabudovaného telnet klienta. Rovněž je možné v případě platformy Windows využít klienta putty http://www.putty.org/. Na platformách založených na UNIXu je možné použít BSD telnet klienta.

B.2 Instrukce pro instalaci

Stáhněte a rozbalte ZIP archiv psimulator2:
http://code.google.com/p/psimulator/downloads/list

B.3 Instrukce pro spuštění

B.3.1 Vytvoření sítě

Pro spuštění simulátoru je nutné mít k dispozici konfigurační soubor sítě, kterou chceme simulovat. Pro vytvoření sítě spusťte frontend příkazem v konzoli:
java -jar psimulator2_frontend.jar

B.3.2 Spuštění backendu

Po vytvoření sítě je možné spustit backend příkazem:
java -jar psimulator2_frontend.jar konfigurační_soubor

Pro vizualizaci paketů je možné propojit backend s frontendem. Ve spuštěném frontendu v simulačním režimu klikněte na Připojit k serveru.

B.3.3 Spuštění grafického telnet klienta

Volitelné a pravděpodobně málo potřebné spuštění grafického telnet klienta jako samostatné aplikace je možné příkazem:
java -jar swingTelnet.jar

B.3.4 Připojení na síťový prvek

Backend vypíše po spuštění seznam nastartovaných síťových prvků s čísly portů, na kterých poslouchají. Pro připojení k libovolnému prvku spusťte zvoleného telnet klienta na localhost(v případě, že provozujete simulátor na stejném stroji) s portem prvku, ke kterému se chcete připojit, např.:
telnet localhost 11001

Pokud je frontend napojen na backend, tak je možné provádět konfiguraci prvků z integrovaného telnet klienta ve frontendu: vpravo nahoře přepněte na záložku Simulátor a po kliknutí pravým tlačítkem na libovolný (konfigurovatelný) prvek je možné otevřít telnet spojení na vybraný prvek.

PŘÍLOHA C

Scénář automatického testování

pc1
- ping -c1 192.168.1.2 (očekává ICMP reply)
- ping -c1 192.168.1.20 (očekává ICMP reply)
- ping -c1 192.168.1.3 (očekává destination host unreachable od 192.168.1.1)
- ping -c1 8.8.8.8 (očekává destination net unreachable od 147.32.1.2)
- ping -c1 89.190.94.1 (očekává time to live exceeded)
- ping -c1 147.32.125.234 (neočekává nic, žádná odezva)
- ping -c1 147.32.1.6 (očekává ICMP reply)
- ping -c1 147.32.1.5 (očekává ICMP reply)

pc0
- ping -c1 8.8.8.8 (očekává „connect: Network is unreachable“)

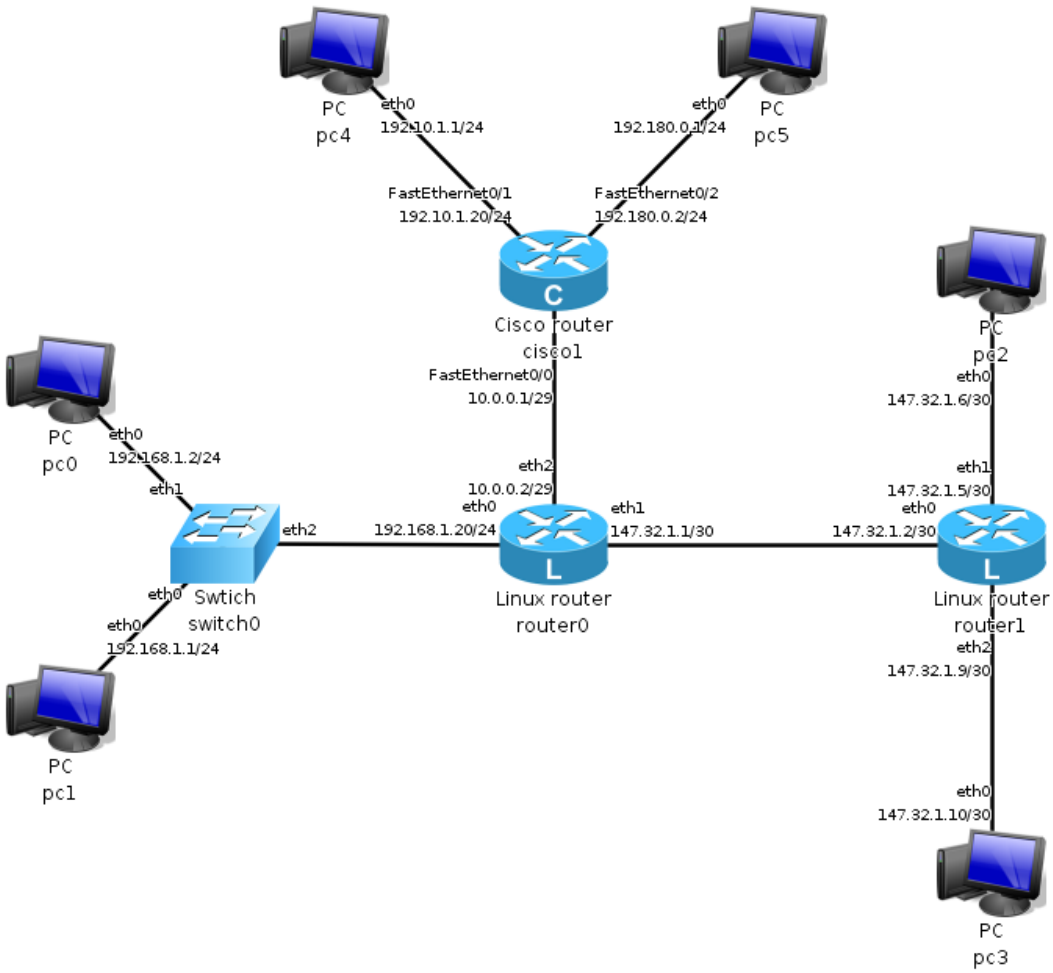
pc4
- ping -c1 192.168.1.1 (očekává „64 bytes from 192.168.1.1: icmp_req=1 ttl=62“)

pc5
- ping -c1 192.168.1.1 (očekává „64 bytes from 192.168.1.1: icmp_req=1 ttl=62“)

Testovací síť je na obrázku C.1.

C.1 Poznámky k jednotlivým prvům

pc1: má výchozí bránu na router0
pc0: nemá výchozí bránu
router0: nastavena maškaráda na odchozí rozhraní eth1, výchozí bránu na



Obrázek C.1: Síť pro automatické testování

router1
router1: není nastavena žádná maškaráda ani výchozí brána, brána na 89.190.94.0/24 na pc2, brána na 147.32.125.128/25 na 147.32.1.10
pc2: chybně nastavený počítač, má výchozí bránu na router1 (pro testování time to live exceeded)
pc3: chybně nastavený počítač, nemá žádnou výchozí bránu (pro testování, kdy se nemá vrátit nic)
pc4: má výchozí bránu na cisco1
pc5: má výchozí bránu na cisco1
cisco1: má výchozí bránu na router0, statický překlad adres pc4 na 10.0.0.6, dynamický překlad adres pc5 na 10.0.0.5

PŘÍLOHA D

Obsah přiloženého CD

- readme.txt ..... stručný popis obsahu CD
- bin ..... adresář se spustitelnými částmi projektu
  - backend ..... adresář se spustitelnou aplikací simulátoru
  - frontend ..... adresář se spustitelnou aplikací vizualizačního rozhraní
  - sampleNetworks ..... adresář s ukázkovými konfiguracemi sítí
  - swingTelnet .. adresář se spustitelnou aplikací grafického telnet klienta
- impl ..... zdrojové kódy implementace
  - backend ..... zdrojové kódy simulátoru
  - frontend ..... zdrojové kódy vizualizačního rozhraní
  - shared ..... zdrojové kódy sdílených struktur
  - swingTelnet ..... zdrojové kódy grafického telnet klienta
- text ..... text práce
  - sources ..... zdrojová forma práce ve formátu LaTeX, obrázky
  - DP-Lukas-Martin-2012.pdf ..... text práce ve formátu PDF