České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra softwarového inženýrství ... (18102)

Diplomová práce

Vizualizace virtuální počítačové sítě

Bc. Martin Švihlík

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

25. dubna 2012

Poděkování

Nejprve bych chtěl poděkovat rodičům a přítelkyni Míše za podporu a toleranci během studia. Dále bych chtěl poděkovat vedoucímu práce Ing. Pavlu Kubalíkovi, Ph.D., za užitečné rady a připomínky při tvorbě diplomové práce.

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 25. dubna 2012 .....

Odkaz na tuto práci

Martin Švihlík. *Vizualizace virtuální počítačové sítě: Diplomová práce*. Praha: ČVUT v Praze, Fakulta informačních technologií, 2012.

Abstract

This thesis describes the design, implementation and testing of a graphical user interface for the PSImulator project, which aims to create a network simulator for the needs of the subject BI-PSI at Faculty of Information Technology. The application developed in this thesis allows users to create and configure a virtual computer network using a GUI and subsequently offers a visual simulation of data packets passing through the network. The first part of this thesis is aimed at analyzing the existing solutions while the second part explains the development of the application itself. Great emphasis is placed at creating the GUI which was also tested by a sample group of target users. Results of the testing are to be found at the end of this thesis.

Keywords Thesis, PSImulator, network simulator, graphical user interface, editor, simulator, Java, Lo-Fi prototype, design patterns, genetic algorithm, JavaHelp, usability testing.

Abstrakt

Tato práce se zabývá návrhem, implementací a testováním grafického uživatelského rozhraní pro projekt PSImulator, jehož cílem je vytvoření síťového simulátoru pro potřeby předmětu BI-PSI na Fakultě informačních technologií. Aplikace vytvořená v této diplomové práci umožňuje uživateli tvorbu a nastavení virtuální počítačové sítě s využitím grafického editoru a následně vizualizaci průchodu paketů sítí v simulačním režimu. Nejdříve jsou v práci analyzována existující řešení a poté se práce věnuje vývoji aplikace samotné. Velký důraz je v práci kladen na tvorbu grafického uživatelského rozhraní, které je na závěr práce otestováno na vzorku cílové skupiny uživatelů.

Klíčová slova Závěrečná práce, PSImulator, síťový simulátor, grafické uživatelské rozhraní, editor, simulátor, Java, Lo-Fi prototyp, návrhové vzory, genetický algoritmus, JavaHelp, testování použitelnosti.

Obsah

Seznam obrázků

Seznam tabulek

Úvod

V dnešní době existuje na trhu množství síťových simulátorů. Některé jsou nabízeny komerčně a jiné jsou zdarma, ale žádný z nich nevyhovuje požadavkům předmětu Počítačové sítě na Fakultě informačních technologií.

Motivace ke vzniku práce

Motivací ke vzniku této práce byl dlouhodobý zájem autora o vývoj aplikací v programovacím jazyku Java, o problematiku návrhu a realizace grafických uživatelských rozhraní a touha po zdokonalení programátorských schopností a dovedností. Neméně významnou motivací byla možnost spolupráce s dalšími studenty na větším projektu, který je z velmi zajímavé oblasti počítačových sítí.

Vytyčení cílů práce

Hlavním cílem této práce je vytvoření multiplatformní aplikace, která umožní tvorbu a vizualizaci virtuální počítačové sítě a simulaci průchodu paketů v takové síti.

Aplikace se bude jmenovat PSImulatorUI a bude jednou z částí projektu nazvaném Project PSImulator, který si klade za cíl vytvořit uživatelsky přívětivý síťový simulátor prvků linux a cisco vhodný pro výukové účely na Fakultě informačních technologií.

Nejdůležitější částí práce je tedy návrh a realizace grafického uživatelského prostředí, které umožní vytvoření schématu virtuální počítačové sítě, její nastavení a vizualizaci průchodu paketů sítí. Výsledná aplikace by měla být uživatelsky přívětivá a snadno použitelná.

Struktura práce

Celá práce se skládá ze tří logických celků - teoretické, praktické a hodnotící.

Teoretickou část práce tvoří kapitoly jedna, dvě a část kapitoly čtyři. V první kapitole se nachází vymezení spolupráce mezi čtyřmi diplomovými

pracemi, které se zabývají projektem PSImulator. Druhou kapitolu tvoří rešerše existujících řešení na trhu síťových simulátorů a analýza jejich dobrých a špatných vlastností. Použité návrhové vzory jsou popsány v části 4.3 kapitoly Návrh a uzavírají teoretickou část práce.

Praktickou část práce tvoří kapitoly 3, 4, 5 a 6, ve kterých je rozebrán celý vývojový proces, který se skládá z analýzy, návrhu, realizace a testování aplikace. V kapitole Analýza jsou analyzovány požadavky na aplikaci a na uživatelské rozhraní a jsou zde vybrány technologie použité při vývoji. Kapitola Návrh se zabývá především návrhem uživatelského rozhraní a dále návrhem architektury aplikace, algoritmem pro automatické rozmístění komponent, datovým modelem a komunikací se simulátorem.

V kapitole Realizace je nejdříve popsána funkcionalita umístěná v jednotlivých částech aplikace a poté jsou popsány zajímavé části implementace. Na závěr páté kapitoly je umístěna ukázka výsledného grafického rozhraní aplikace.

Nedílnou součástí vývoje aplikací je testování, jehož popis se nachází v kapitole 6. Aplikace je nejdříve testována experty z oboru a poté předpokládanými skupinami uživatelů.

Poslední část práce tvoří ohlédnutí za celým vývojem aplikace a zhodnocení dosažených výsledků. Vytvořená aplikace je porovnána s existujícími řešeními a na konec jsou uvedena možná vylepšení a rozšíření aplikace.

KAPITOLA 1

Vymezení spolupráce

Tato diplomová práce je součástí sady čtyř diplomových prací, které se zabývají různými částmi projektu PSImulator. V této kapitole vysvětlím souvislosti mezi jednotlivými diplomovými pracemi a uvedu, co bylo jejich náplní.

1.1 Vymezení spolupráce

Cílem projektu PSImulator je vytvoření komplexního nástroje, který umožní vytvoření a simulaci virtuální počítačové sítě s prvky založenými na operačních systémech Linux a prvky založenými na Cisco směrovačích. Výsledná aplikace bude použita k výukovým účelům an Fakultě informačních technologií v předmětu BI-PSI.

Projekt PSImulator byl rozdělen do čtyř diplomových prací. Dále následuje jejich popis.

1.1.1 Vizualizace virtuální počítačové sítě

První částí projektu je tato práce, která se zabývá návrhem a realizací editoru pro tvorbu virtuálních počítačových sítí a vizualizací průchodu paketů sítí.

1.1.2 Síťový simulátor pro výukové účely na bázi prvků OS Linux

Diplomová práce Bc. Tomáše Pitřince má za úkol vytvořit softwarové prostředí pro simulaci virtuální počítačové sítě a je zaměřena především na konfiguraci síťových prvků založených na OS Linux.

1.1.3 Síťový simulátor pro výukové účely na bázi směrovačů CISCO

Bc. Stanislav Řehák ve své diplomové práci řeší také tvorbu softwarového prostředí pro simulaci virtuální počítačové sítě a zaměřuje se především na konfiguraci systému Cisco. Jádro aplikace je společné s prací Bc. Tomáše Pitřince a je jejich společnou prací. Výsledkem práce Tomáše a Stanislava je síťový simulátor, který má název PSImulator.

1.1.4 Podpůrné komponenty simulátoru počítačové sítě

Poslední diplomová práce je dílem Bc. Martina Lukáše, který se zaměřuje na tvorbu podpůrných komponent pro simulátor virtuální počítačové sítě. Jeho práce obsahuje tvorbu textového (telnet) přístupu k aktivním prvkům virtuální sítě, napojení simulátoru na grafické rozhraní a vytvoření virtuálního souborového systému pro virtuální aktivní prvky. Dále je součástí práce realizace textového editoru pro virtuální aktivní prvky a rozhraní pro uložení konfigurace virtuální sítě.

KAPITOLA 2

Existující řešení

Síťových simulátorů existuje mnoho, ale ne každý je vhodný k použití k výukovým účelům na fakultě informačních technologií v předmětu MI-PSI, který má studenty naučit základům počítačových sítí. Do této kapitoly jsem vybral nejdůležitější a nejzajímavější programy, které jsou na trhu k dispozici. Při analýze existujících řešení jsem využil zkušenosti Stanislava Řeháka z jeho bakalářské práce [5].

2.1 Cisco Packet Tracer

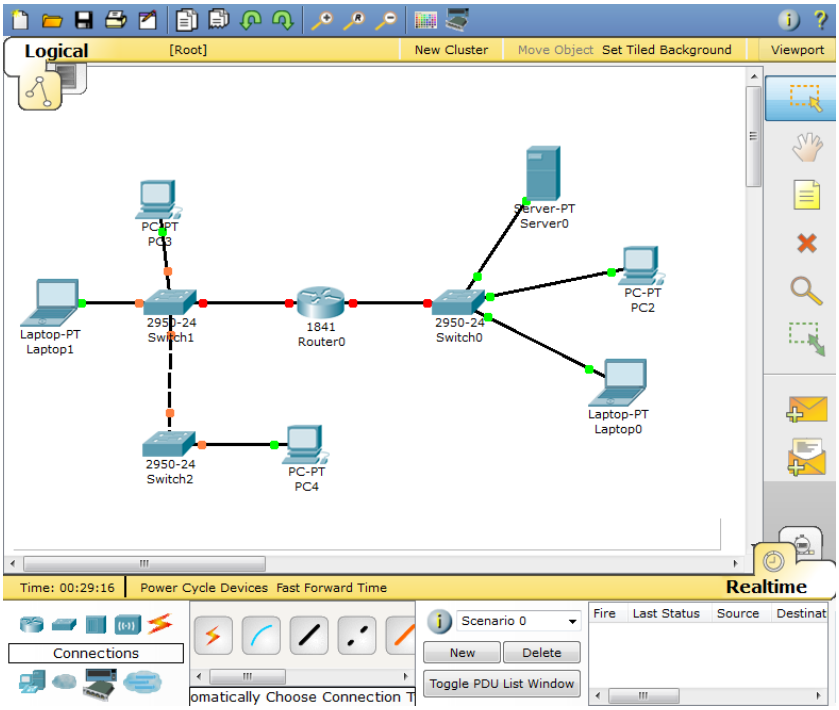
Cisco Packet Tracer [2] je síťový simulační program od firmy Cisco, vyvinutý pro potřeby Cisco Networking Academy.

2.1.1 Popis programu

Na trhu síťových simulátorů je Cisco Packet Tracer asi nejznámějším nástrojem. V programu je možné s pomocí grafického uživatelského rozhraní vytvořit počítačovou síť z nabízených komponent a propojit ji kabely. Poté uživatel může nastavit parametry zařízení buď s využitím grafického rozhraní, nebo s pomocí příkazové řádky. Po vytvoření a nastavení sítě je možné simulovat a vizualizovat provoz. Tento software je možné bez poplatků využít v Cisco Networking Academy a je k dispozici pro operační systémy Microsoft Windows a Linux.

2.1.2 Grafické uživatelské rozhraní

Jak je možné vidět na obrázku 2.1, grafické uživatelské rozhraní tohoto programu je na pohled přehledné a uživatelsky příjemné. Podle mého názoru je mezi síťovými simulátory jedno z nejpovedenějších, ale stále je v něm možné



Obrázek 2.1: Cisco Packet Tracer - vytvořená síť

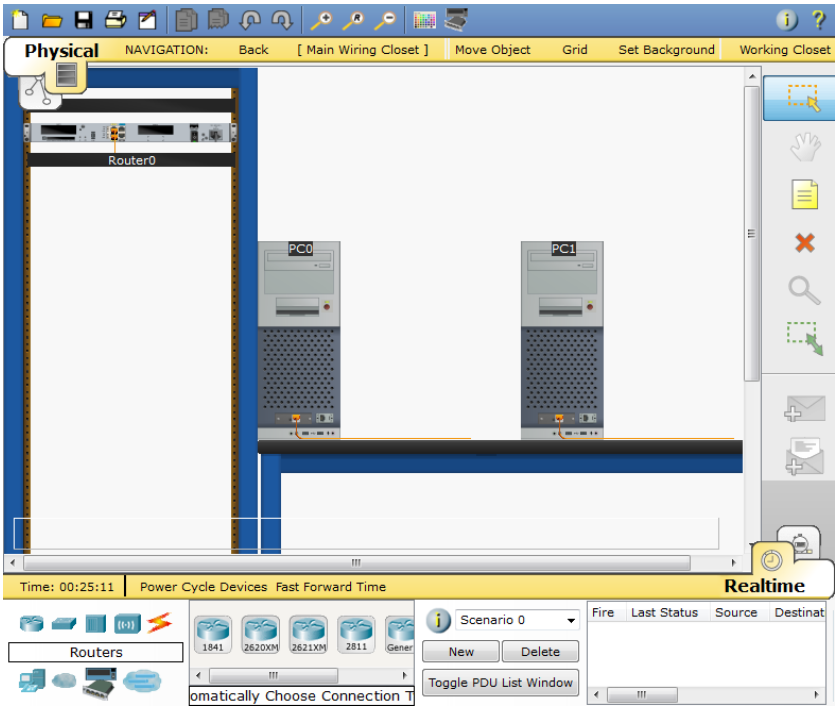
najít několik vad, které sice nebrání funkčnosti aplikace, ale znesnadňují její používání.

Myslím si, že vytváření počítačové sítě je uživatelsky poněkud kostrbaté, protože je každý kus hardwaru nutné vzít z palety zvlášť a poté ho umístit do plochy. Vhodnější by bylo vybrat nástroj a poté už jen umísťovat komponenty. Další vadou na kráse je graficky nepříliš přívětivé zobrazení komponent na ploše. Obrázky jsou v nízkém rozlišení a při přiblížení plochy se pouze interpolují na větší velikost a vypadají rozmazaně. Rovněž vyhlazování hran v tomto programu není aktivní. Další věcí, která by mohla být vizuálně vylepšena, jsou animace průchodu paketů sítí, které jsou velmi primitivní.

Zajímavou vlastností Cisco Packet Traceru je možnost komponenty rozmístit do okresů, měst, budov a místností. V místnosti jsou poté vidět reálné obrázky komponent (obrázek 2.2). Takovéto rozmístění přiblíží uživateli tvorbu sítě a pomůže mu představit si, jak se staví opravdová síť.

2.1.3 Výhody a nevýhody

Největší výhodou této aplikace je, že se autorům podařilo skloubit široké možnosti použití a věrné zachycení tvorby počítačové sítě spolu se zachováním jednoduchosti uživatelského rozhraní, které uživatele neodrazuje. Dalšími vý-



Obrázek 2.2: Cisco Packet Tracer - fyzický pohled

hodou je multiplatformnost aplikace a podpora velkého hráče na trhu síťových technologií (Cisco).

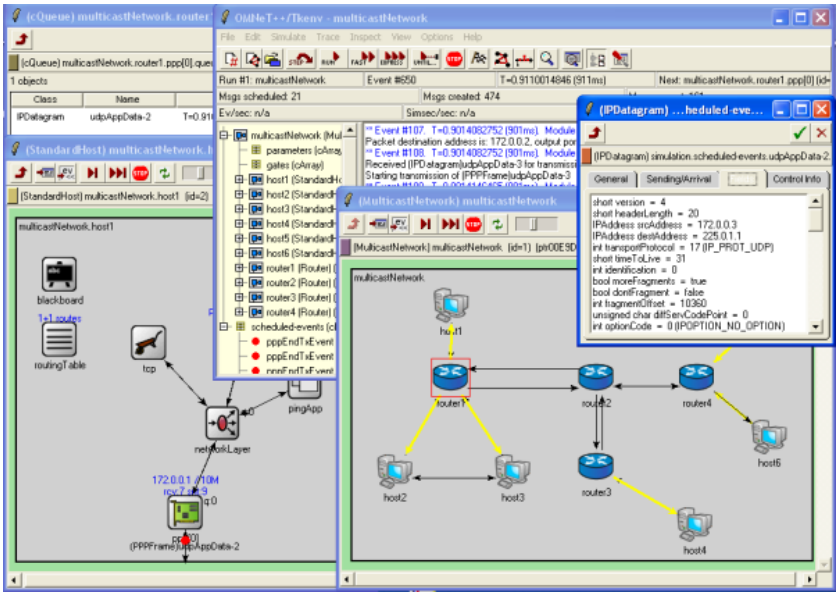
Obrovskou nevýhodou z pohledu použití v předmětu MI-PSI je licence aplikace. Program je možné zdarma použít pouze v rámci kurzu Cisco Networking Academy. Další nevýhodou jsou výše uvedené nedostatky v uživatelském rozhraní.

2.2 OMNeT++ simulátor

OMNeT++ [36] je simulační nástroj od Andráse Vargy primárně zaměřený na simulaci komunikačních sítí.

2.2.1 Popis programu

Tento software je založen na flexibilní modulární architektuře, která mu umožňuje při použití správných modulů modelovat drátové i bezdrátové sítě, protokoly, hardwarové architektury či počítačové sítě. Program OMNeT++ není zaměřen na konkrétní problém, hlavní důraz je kladen na podporu tvorby simulací. Aplikaci je možné používat na většině běžně dostupných operačních systémů.



Obrázek 2.3: OMNeT++ - uživatelské rozhraní

2.2.2 Grafické uživatelské rozhraní

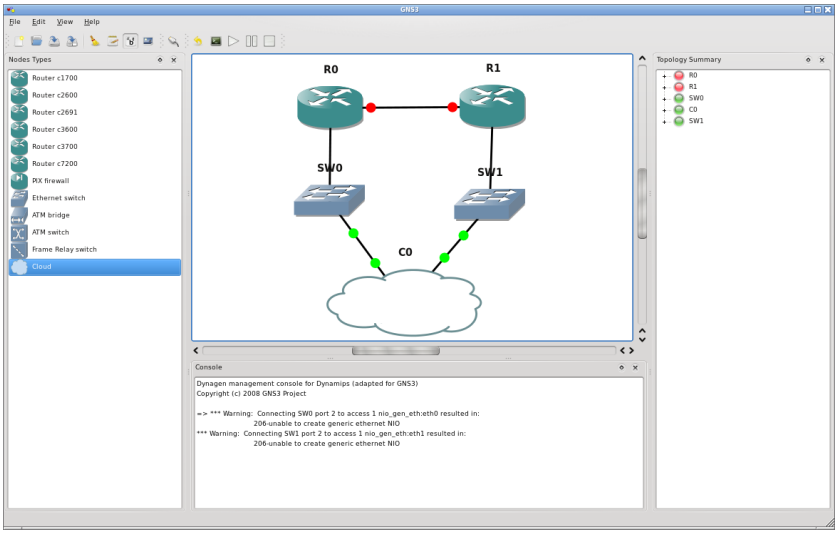
Jak je vidět na obrázku 2.3, uživatelské rozhraní tohoto nástroje je strohé a účelné. Důraz byl kladen spíše na funkce programu než na uživatelské rozhraní.

2.2.3 Výhody a nevýhody

Největší výhodou a zároveň nevýhodou je komplexnost celé aplikace. Pro studijní účely v předmětu MI-PSI je tento nástroj příliš rozsáhlý a složitý jak na používání, tak například na instalaci. Program OMNeT++ se spíše hodí pro vědeckou nebo inženýrskou práci.

2.3 Dynamips Cisco 7200 Simulator a grafické rozhraní GNS3

Dynamips Cisco 7200 Simulator [4] od Christophera Fillota je software, který emuluje Cisco IOS na tradičním PC. GNS3 [9] je grafické rozhraní od týmu programátorů vedených Jeremym Grossmannem právě pro simulátor Dynamips.



Obrázek 2.4: Grafické uživatelské rozhraní GNS3

2.3.1 Popis programu

Dynamips simulátor používá přímo originální obrazy Cisco IOS, díky čemuž podporuje všechny funkce emulovaného směrovače. Tento simulátor je distribuován pod licencí GNU GPL a je hojně využíván studenty Cisco Networking Academy.

Grafické rozhraní GNS3 je založeno na simulátoru Dynamips a umožňuje uživateli v přehledném grafickém prostředí vytvořit virtuální počítačovou síť. Tento program je také volně šiřitelný a to pod licencí GPLv2.

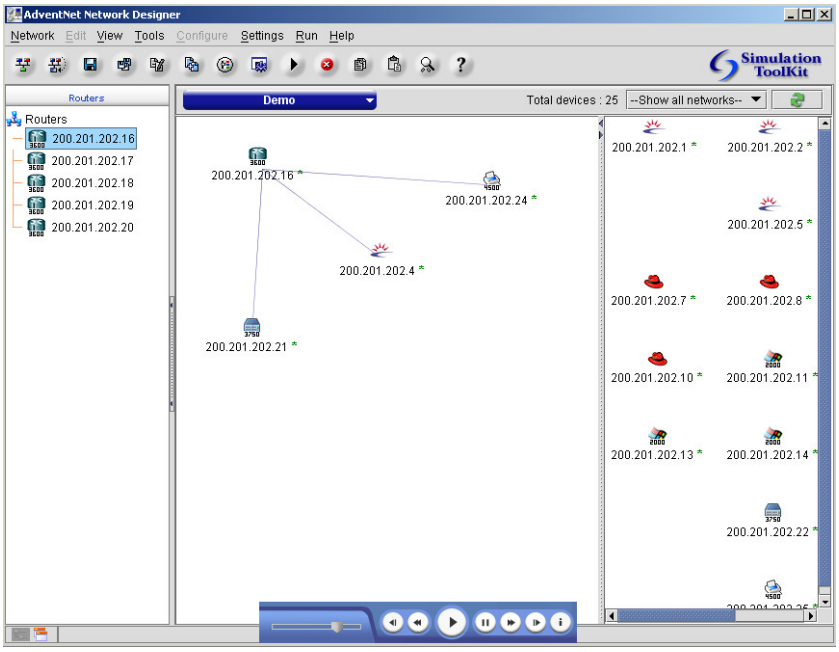
Oba programy fungují na platformách Mac OS, Linux a Windows.

2.3.2 Grafické uživatelské rozhraní

Na obrázku 2.4 je znázorněno uživatelské rozhraní programu GNS3. Rozhraní vyniká jednoduchostí a přehledností. Nedostatkem je, že software nepodporuje animace průchodu paketů sítí.

2.3.3 Výhody a nevýhody

Největší výhodou a zároveň slabinou je využití Cisco IOS obrazů síťových prvků Cisco. Program dokáže využít veškeré funkce Cisco prvků, ale nevýhodou je licenční omezení. Tyto obrazy nejsou volně šiřitelné a to činí GNS3 nepoužitelným pro potřeby předmětu BI-PSI.



Obrázek 2.5: Simulation Toolkit 7.0 - uživatelské rozhraní

2.4 Simulation Toolkit 7.0

2.4.1 Popis programu

Výrobcem softwaru Simulator Toolkit 7.0 [38] je společnost WebNMS, která simulátor vyvíjí jako čistě komerční produkt. Software je multiplatformní a umožňuje při tvorbě sítě vybírat z velkého množství zařízení včetně směrovačů Cisco. Licence je placená a její ceny začínají na částce $995[1].

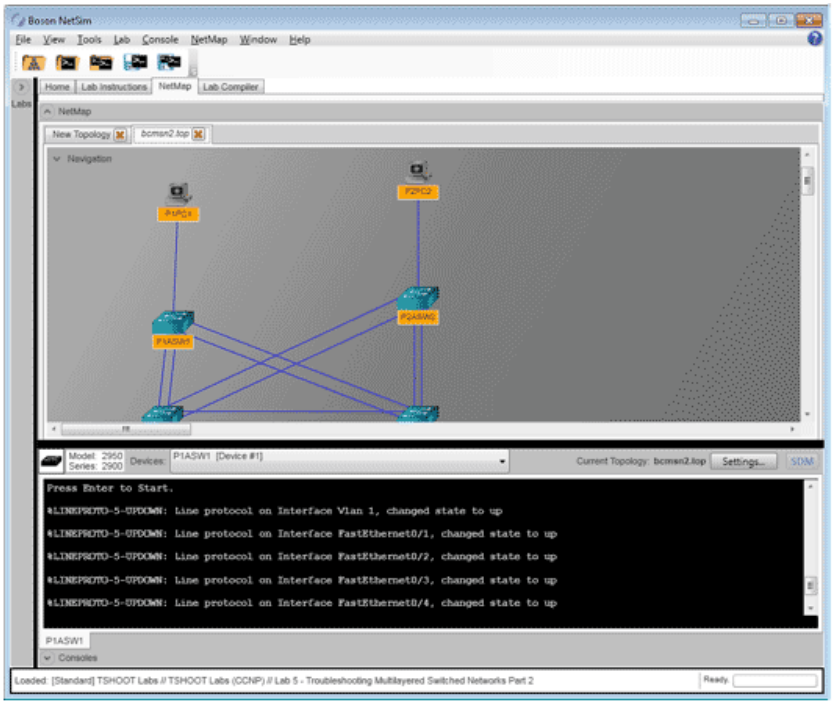
2.4.2 Grafické uživatelské rozhraní

Uživatelské rozhraní simulátoru (obrázek 2.5) je jednoduché a vcelku přehledné, ale ničím zajímavým nevyniká.

2.4.3 Výhody a nevýhody

Výhodou tohoto řešení je robustnost (výrobce uvádí simulaci až 5000 Cisco IOS zařízení najednou) a jednoduchost uživatelského rozhraní. Nevýhodou je placená a velmi drahá licence, kvůli níž není možné tento produkt využít pro výukové účely.

[1] Ke dni 6.3.2012



Obrázek 2.6: Boson NetSim 8.0 - uživatelské rozhraní

2.5 NetSim 8.0 Network Simulator

2.5.1 Popis programu

NetSim 8.0 Network Simulator [1] od společnosti Boson je dalším z řady simulátorů na trhu. Tento software je primárně určen pro studenty programů CCNP a CCNA a je placený. Nástroj umožňuje sestavit síť s až 200 zařízeními. K dispozici je 42 typů směrovacích zařízení a 6 typů přepínačů, z toho 48 zařízení Cisco. Výrobce nabízí tento produkt pouze pro operační systémy společnosti Microsoft.

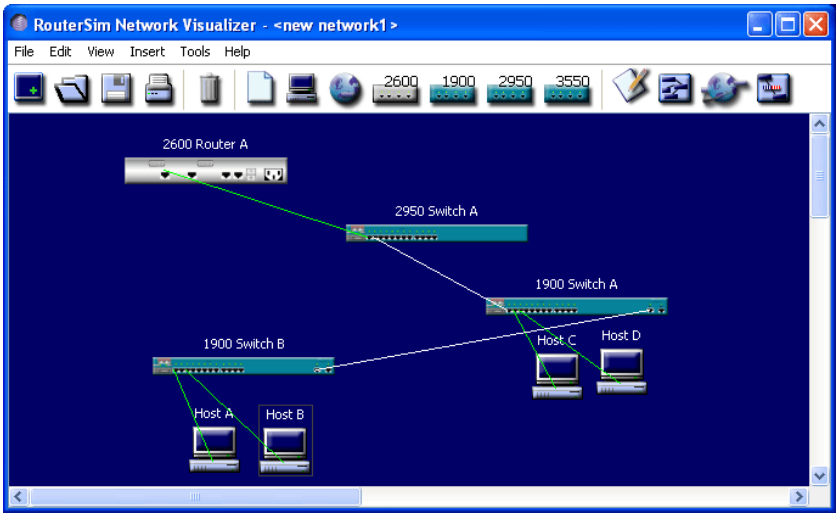
2.5.2 Grafické uživatelské rozhraní

Uživatelské rozhraní (obrázek 2.6) kombinuje konzolové a grafické rozhraní a je primárně zaměřené na plnění úkolů školících programů Cisco.

2.5.3 Výhody a nevýhody

Toto řešení má výhodu v podpoře široké škály zařízení Cisco. Nevýhodou je opět licence, kterou je nutné zakoupit ($99 - $349[2]) a také jednoúčelové

[2]Ke dni 6.3.2012



Obrázek 2.7: RouterSim - uživatelské rozhraní

zaměření na kurzy Cisco.

2.6 RouterSim

2.6.1 Popis programu

Router Sim [35] je další z řady placených programů, které jsou vytvořeny za účelem podpory výukových kurzů Cisco. Licence programu stojí od $79 do $179[3] a poté si uživatel kupuje jednotlivé kurzy.

2.6.2 Grafické uživatelské rozhraní

Tento nástroj jsem uvedl zejména kvůli jeho uživatelskému rozhraní, které je velmi zjednodušené a podle mého názoru nepříliš povedené (obrázek 2.7).

2.6.3 Výhody a nevýhody

Tento nástroj z mého pohledu žádnou výhodu nepřináší, snad až na jednoduchost uživatelského rozhraní. Nevýhodou je zaměření pouze na kurzy Cisco, placená licence a podpora pouze operačního systému Windows.

[3]Ke dni 6.3.2012

2.7 Další řešení

Na trhu se síťovými simulátory se vyskytuje množství dalších řešení. Většina výrobců se zaměřuje na podporu výukových programů Cisco a licence těchto produktů jsou placené.

2.8 Shrnutí analýzy existujících řešení

Analýza existujících řešení mi pomohla vytvořit si přehled parametrů, které by měl mít ideální program, jenž by vyhovoval požadavkům předmětu BI-PSI. Jednou z nejdůležitějších věcí je licence, která by měla umožňovat bezplatné použití ve školním prostředí. Další věcí je jednoduché, přehledné a srozumitelné uživatelské rozhraní, které nebude uživatele odrazovat od používání programu. V neposlední řadě je důležitá multiplatformnost aplikace, jež umožní studentům aplikaci využívat i doma na svých různorodých operačních systémech.

KAPITOLA 3

Analýza

Důležitou částí softwarového projektu je analýza, která slouží k sestavení požadavků na implementovaný systém. Po této fázi by mělo být jasné, jak má nový systém vypadat, co má umět a jakým požadavkům má vyhovovat.

3.1 Požadavky na implementovaný systém

V této podkapitole jsem se věnoval analýze zadání práce, ze kterého jsem vytvořil tzv. byznys požadavky[4]. Poté jsem z těchto požadavků a z konzultací se zákazníkem (vedoucím práce) sestavil funkční a nefunkční požadavky na implementovaný systém. Tento seznam požadavků bude dále sloužit jako podklad pro návrh a realizaci programu a také jako kontrolní seznam, podle kterého bude výsledný produkt otestován.

3.1.1 Přesné zadání práce

Pro přehlednost uvedu ještě jednou přesné zadání, které jinak najdete na první straně práce.

Navrhněte a realizujte aplikaci pro tvorbu a zobrazení virtuální počítačové sítě popsané s pomocí jazyka XML. Vyberte vhodný algoritmus pro optimální zobrazení jednotlivých síťových prvků tak, aby se spojení mezi prvky minimálně křížily. Aplikace bude umožňovat nakreslit schéma libovolné počítačové sítě, její nastavení a simulací průchodu paketů sítí. Množství zobrazených informací o stavu sítě bude možné změnit. Rychlost samotné simulace bude možné nastavit. Záznam simulace bude možné uložit a opětovně přehrát.

[4]Byznys požadavek - požadavek na systém nejvyšší úrovně (editor počítačové sítě.)

3.1.2 Požadavky byznysu

Požadavky byznysu, známé pod anglickým označením Business requirements, jsou formovány jako nejvyšší pohled na požadovaný systém. Říkají v několika stručných bodech, jaké věci jsou nejdůležitější pro zadavatele.

BP 1: Editor počítačové sítě.

BP 2: Vizualizace průchodu paketů sítí.

BP 3: Jednoduché použití.

BP 4: Kompatibilita s ostatními komponentami PSImulatoru[5].

3.1.3 Funkční požadavky

Funkční požadavky definují, co má aplikace umět, jak se má chovat a co podporovat. Jako podklad pro vytvoření funkčních požadavků slouží byznys požadavky a konzultace se zákazníkem (vedoucím práce). V další podkapitole jsou požadavky podrobně analyzovány (3.2).

FP 1: Aplikace bude mít dva operační módy: Editor a Simulátor.

FP 2: Aplikace bude umožňovat vytvořit virtuální počítačovou síť z nabízených komponent a kabelů.

FP 3: Aplikace bude umožňovat nastavit parametry komponent virtuální počítačové sítě.

FP 4: Aplikace bude umožňovat nastavit parametry kabelů virtuální počítačové sítě.

FP 5: Aplikace bude umožňovat uložení konfigurace sítě.

FP 6: Aplikace bude umožňovat načtení konfigurace sítě.

FP 7: Aplikace bude umožňovat automatické rozmístění komponent virtuální počítačové sítě na ploše.

FP 8: Aplikace bude umožňovat změnu množství zobrazovaných informací o virtuální počítačové síti.

FP 9: Aplikace bude podporovat operace zpět/vpřed (undo/redo).

FP 10: Aplikace bude umožňovat měnit úroveň přiblížení (zoom).

FP 11: Aplikace bude umožňovat spojení za pomoci počítačové sítě s komponentou PSImulator.

[5]Tím jsou myšleny komponenty, které tvoří zbylí tři studenti pracující na tomto projektu.

FP 12: Aplikace bude umožňovat přijímat události[6] z komponenty PSImulator.

FP 13: Aplikace bude umožňovat zaznamenávat události z komponenty PSImulator.

FP 14: Aplikace bude zobrazovat seznam zaznamenaných událostí.

FP 15: Aplikace bude umožňovat přehrávat události z komponenty PSImulator v reálném čase.

FP 16: Aplikace bude umožňovat zvolit rychlost přehrávání zaznamenaných událostí.

FP 17: Aplikace bude umožňovat sekvenční přehrávání zaznamenaných událostí.

FP 18: Aplikace bude umožňovat přehrávat zaznamenané události podle reálných časových značek.

FP 19: Aplikace bude umožňovat pohyb v seznamu zaznamenaných událostí.

FP 20: Aplikace bude umožňovat uložení seznamu zaznamenaných událostí.

FP 21: Aplikace bude umožňovat načtení seznamu zaznamenaných událostí.

FP 22: Aplikace bude umožňovat zobrazit detail zaznamenané události.

FP 23: Aplikace bude umožňovat připojení k virtuálním zařízením s využitím telnet protokolu.

FP 24: Aplikace bude vícejazyčná.

FP 25: Aplikace bude umožňovat změnu jazyka za běhu.

FP 26: Aplikace se bude přizpůsobovat aktuální velikosti okna.

FP 27: Aplikace bude obsahovat nápovědu.

3.1.4 Nefunkční požadavky

Nefunkční požadavky specifikují nároky na aplikaci, které přímo nesouvisí s její funkcionalitou, ale mají vliv na vývoj a podobu výsledného řešení. Typickým příkladem jsou výkonnostní požadavky, požadavky na provozní prostředí nebo požadavky na zvolený programovací jazyk.

NP 1: Aplikace bude multiplatformní.

[6] Událostí se myslí informace o zaslání paketu z komponenty A do komponenty B.

NP 2: Aplikace bude napsána v programovacím jazyku Java.

NP 3: Aplikace bude umět využít velkého rozlišení moderních monitorů.

NP 4: Aplikace bude výkonnostně stačit síťovému simulátoru.

NP 5: Aplikace bude využívat pro ukládání konfigurace sítě formát XML.

NP 6: Aplikace bude využívat pro ukládání seznamu zachycených událostí formát XML.

3.2 Analýza požadavků

Požadavky, které nejsou na první pohled jednoznačné a vyžadují hlubší analýzu, jsou rozebrány v následující podkapitole.

V požadavku FP 1 se hovoří o dvou operačních režimech celého programu, kterými jsou editační mód a simulační mód. Jak je z názvů patrné, v editačním režimu půjde o vytvoření a nastavení virtuální počítačové sítě a v režimu simulace o zachytávání a přehrávání (simulování) událostí nad vytvořenou virtuální počítačovou sítí. Tyto dva operační režimy dělí funkční požadavky do tří skupin: požadavky k editoru, simulátoru a společné.

3.2.1 Funkční požadavky týkající se editoru

V této části se budu zabývat funkčními požadavky, které souvisí s režimem editor (FP 2 - FP 10).

Požadavek FP 2 mluví o tvorbě virtuální sítě s pomocí nabídky komponent a kabelů. Tento požadavek v sobě ukrývá celý princip tvorby virtuální počítačové sítě v editoru a je rozebrán spolu s dalšími souvisejícími požadavky v části Analýza požadavků na uživatelské rozhraní 3.3.

V FP 3 a FP 4 je uvedeno nastavování parametrů jednotlivých komponent a kabelů. U komponent je to například IP adresa nebo MAC adresa. U kabelů je to zejména jeho zpoždění v nějaké časové jednotce.

Požadavky FP 5 a 6 uvádí potřebu ukládání a načítání konfigurace sítě. Programovací jazyk Java disponuje standardními prostředky pro čtení a zápis do souborů, tudíž pro splnění těchto požadavků nebude potřeba speciálních nástrojů.

Podle FP 7 je potřeba implementovat automatické rozmístění sítě v ploše. Toho lze dosáhnout návrhem a implementací nějakého pokročilého heuristického algoritmu. V úvahu připadá například Genetický algoritmus.

Požadavek FP 8 na možnost změny množství zobrazovaných informací o síti pochází přímo ze zadání práce. Pro implementaci to bude znamenat vypořádání se s umístěním jednotlivých popisků v zobrazené síti tak, aby byly dobře viditelné a zároveň aby mohly být kdykoliv zobrazeny nebo skryty.

Při implementaci jakéhokoliv uživatelsky přívětivého software je důležité myslet na chyby uživatele. Požadavek FP 9 uvádí nutnost přítomnosti operací zpět a vpřed v programu. Tyto operace umožní uživateli dělat potenciálně nebezpečné operace (např. smazání komponent) bez obav, že přijde o svou práci.

V požadavku FP 10 je zmíněna podpora funkce zoom. Změna úrovně přiblížení je v aplikaci potřeba kvůli možnosti tvořit větší virtuální sítě, které by se za normálních okolností nevešly na obrazovku. Díky možnosti změnit zoom si uživatel bude moci prohlédnout jak detaily sítě, tak zobrazit celkový přehled o síti.

3.2.2 Funkční požadavky týkající se simulátoru

V této části se budu zabývat požadavky, které souvisí s režimem simulátor (FP 11 - FP 22).

Požadavky FP 11 až FP 13 uvádí nutnost komunikačního spojení s komponentou PSImulator (simulátor sítě). Výsledné řešení by mělo být schopno přijímat informace o zaslání paketu ze síťového prvku A do prvku B a uchovávat je v seznamu událostí.

FP 14 hovoří o zobrazení seznamu událostí. Ideální formou takového zobrazení by mohla být tabulka, ve které budou události seřazené podle času přijetí.

Přehráváním událostí ve funkčním požadavku 15 je myšleno zobrazení průchodu paketu sítí. Nejvhodnější vizualizací je podle mého názoru animace.

Z požadavku FP 16 na změnu rychlosti přehrávání a z požadavku FP 15 vyplývá nutnost chytrého časování animací, tj. při tvorbě animace bude nutné brát v úvahu vzdálenost obou síťových prvků v ploše, nastavenou rychlost animace a také zpoždění nastavené na kabelu, na kterém se událost stala.

Sekvenční přehrávání (FP 17) je potřeba z důvodu přehledného zobrazení jednotlivých událostí. Pro uživatele je mnohem jednodušší sledovat cestu jednoho paketu po druhém, než sledovat pohyb více paketů najednou.

Přehrávání podle časových značek (FP 18) je vhodné zejména pokud chceme věrně reprodukovat zaznamenané události. Tento typ přehrávání nám umožní přehrát události ze záznamu přesně v takových časech, v jakých jsme je zaznamenali.

Požadavek FP 19 uvádí možnost pohybu v zaznamenaném listu. Ideálním řešením by byla úplný svoboda pohybu v seznamu událostí. V případě zobrazení seznamu jako tabulky by bylo vhodné využít pohyb šipkami i výběr s pomocí myši.

Pro potřeby předmětu BI-PSI je nutné implementovat ukládání a načítání seznamu událostí. Jen tak bude možné ověřit, že síť, kterou student navrhl, dělá opravdu to, co má.

Předposledním požadavkem na simulátor je zobrazení detailu zaznamenané události. Už nyní je zřejmé, že u události bude potřeba zobrazit zdro-

jový prvek, cílový prvek, označení jejich rozhraní, typ události, časovou značku a další. Zobrazit všechny tyto parametry v tabulce by sice bylo možné, ale tabulka by byla příliš velká. Proto by bylo vhodné zobrazit detail události v nějakém náhledu.

Posledním požadavkem (FP 23) je připojení k virtuálnímu zařízení s využitím telnet protokolu. Tato funkce umožní uživateli jednoduché spojení s virtuálními zařízeními přímo z grafického uživatelského prostředí.

3.2.3 Funkční požadavky týkající se obou režimů

V této části se budu zabývat požadavky, které souvisí s oběma operačními režimy (FP 24 - FP 27).

Požadavek FP 24 a 25 vyžaduje podporu více jazyků a možnost změnit jazyk za běhu aplikace. V dnešní době na trhu převládají řešení, která je možné jazykově přizpůsobit a je velmi vhodné, aby aplikace podporovala minimálně angličtinu a češtinu. Změna jazyka za běhu je užitečná věc, jelikož restartování programu kvůli aplikaci změn v nastavení není v žádném případě uživatelsky přívětivé.

Přizpůsobení aplikace aktuální velikosti okna (FP 26) souvisí s NP 4 a myslí se tím, že aplikace by měla efektivně využít místo na obrazovce. To znamená, že například při zvětšení okna se zvětší i plocha pro tvorbu sítě.

Posledním funkčním požadavkem je přítomnost nápovědy. Každá moderní uživatelsky přívětivá aplikace by měla nabízet možnost zobrazení nápovědy a je vcelku jedno, zda bude nápověda zobrazena v internetovém prohlížeči, či zda bude zabudována přímo v aplikaci.

3.2.4 Nefunkční požadavky

Požadavek NP 1 na multiplatformní aplikaci souvisí s požadavkem NP 2 na programovací jazyk Java a oba vycházejí z požadavku BP 5, který požaduje kompatibilitu s ostatními komponentami projektu PSImulator. V našem projektu, který má sdílené části a je tvořen z komponent, které do sebe navzájem zapadají, je nutné použít shodný programovací jazyk. Vybrán byl jazyk Java a to díky jeho podpoře velkého množství operačních systémů. Více o důvodech výběru jazyku Java naleznete v podkapitole Použité technologie3.4.

Požadavek NP 3 mluví o podpoře velkého rozlišení moderních monitorů. Vychází z toho, že v dnešní době má velké množství lidí velkou úhlopříčku, potažmo velké rozlišení svého monitoru a je tudíž vhodné tomu aplikaci uzpůsobit tak, aby velkou plochu na obrazovce dokázala využít.

NP 4 uvádí požadavek na výkon ve smyslu toho, že bude výkonnostně stačit na přehrávání (vizualizaci) paketů z komponenty PSImulator. Aplikace by měla zvládnout přijímat a zobrazovat tolik paketů, kolik dokáže síťový simulátor vyprodukovat.

Požadavky NP 5 a NP 6 hovoří o požadovaném datovém formátu. XML je velmi dobrou možností, jak zajistit čitelnost uloženého souboru i s pomocí obyčejného textového editoru. Tento formát umožní případné zásahy přímo do uloženého souboru bez nutnosti otevírání v grafickém prostředí.

3.3 Analýza požadavků na uživatelské rozhraní

Analýza požadavků na uživatelské rozhraní vychází z funkčních požadavků uvedených v předchozí kapitole (primárně z požadavku FP2). Cílem této analýzy je popsat systém z uživatelské perspektivy, tzn. vytvořit seznam interakcí, které bude uživatel se systémem vykonávat, a rozdělit jej do kategorií.

3.3.1 Use cases brainstorming

V části analýzy nazvané Use case brainstorming je důležité zachytit hlavní rysy uživatelského rozhraní aplikace.

Uživatelské rozhraní by mělo umožnit:

- Vybírat z nabídky nástrojů
- Přidávání komponenty na plochu
- Posuv komponent
- Označování komponent myší
- Mazání komponent
- Spojování komponent kabelem
- Operace zpět/vpřed
- Zoom
- Editaci vlastností komponenty
- Zachytávání událostí
- Přehrávání událostí
- Ukládání a načítání
- Přepínání jazyků

3.3.2 Task list

Vstupem části nazvané Task list je výše uvedený seznam hlavních rysů uživatelského rozhraní. Cílem tvorby task listu je rozdrobení jednotlivých bodů z fáze Use cases brainstorming do pokud možno atomických interakcí se systémem.

Následuje seznam atomických aktivit:

- Vytvoř nový projekt
- Otevři stávající projekt
- Ulož projekt
- Zavři projekt
- Zobraz nastavení programu
- Přepni z editačního režimu na simulační
- Vyber nástroj v liště nástrojů (toolbar)
- Změň nástroj ve skupině nástrojů
- Přidej nástrojem komponentu na plochu
- Označ komponentu
- Označ více komponent
- Označ všechny komponenty
- Posuň označené komponenty
- Smaž označené komponenty
- Spoj dvě komponenty kabelem
- Vyber rozhraní při spojování komponent
- Zobraz vlastnosti komponenty
- Zobraz vlastnosti kabelu
- Změň nastavení komponenty
- Změň nastavení kabelu
- Zarovnej komponenty do mřížky
- Automaticky rozmísti komponenty v ploše

- Změň úroveň přiblížení
- Zpět/vpřed
- Připoj se k síťovému simulátoru
- Zobraz vlastnosti připojení
- Odpoj se od síťového simulátoru
- Spusť zachytávání událostí
- Zastav zachytávání událostí
- Spusť realtime režim
- Zastav realtime režim
- Spusť přehrávání
- Zastav přehrávání
- Změň rychlost přehrávání
- Přejdi na předchozí událost
- Přejdi na následující událost
- Přejdi na první událost
- Přejdi na poslední událost
- Přejdi na událost výměrem s pomocí myši
- Ulož seznam událostí
- Načti seznam událostí
- Smaž seznam událostí
- Zobraz nápovědu

3.3.3 Task groups

Nyní je potřeba kvůli přehlednosti rozdělit uživatelské akce do skupin:

Společné uživatelské akce

- Vytvoř nový projekt
- Otevři stávající projekt
- Ulož projekt
- Zavři projekt
- Zobraz nastavení programu
- Přepni z editačního režimu na simulační
- Změň úroveň přiblížení
- Zobraz nápovědu

Uživatelské akce v editačním režimu

- Vyber nástroj v liště nástrojů (toolbar)
- Změň nástroj ve skupině nástrojů
- Přidej nástrojem komponentu na plochu
- Označ komponentu
- Označ více komponent
- Označ všechny komponenty
- Posuň označené komponenty
- Smaž označené komponenty
- Spoj dvě komponenty kabelem
- Vyber rozhraní při spojování komponent
- Zobraz vlastnosti komponenty
- Zobraz vlastnosti kabelu
- Změň nastavení komponenty
- Změň nastavení kabelu
- Zpět/vpřed
- Zarovnej komponenty do mřížky
- Automaticky rozmísti komponenty v ploše

Uživatelské akce v simulačním režimu

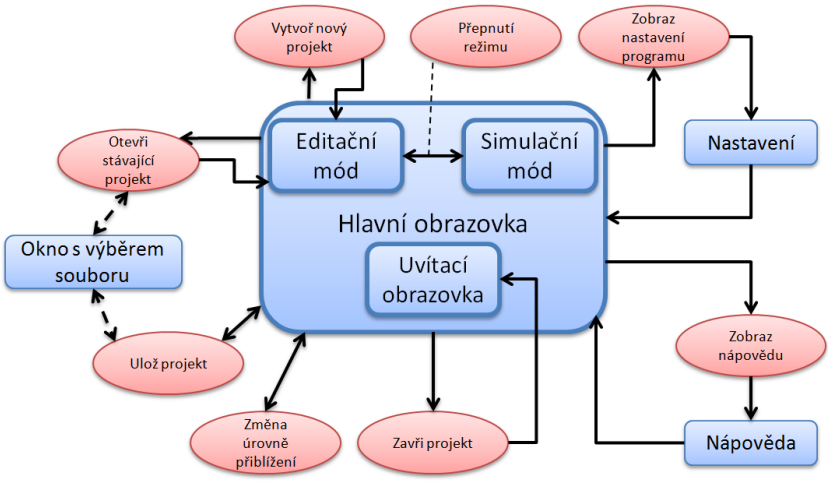
- Připoj se k síťovému simulátoru
- Zobraz vlastnosti připojení
- Odpoj se od síťového simulátoru
- Spusť zachytávání událostí
- Zastav zachytávání událostí
- Spusť realtime režim
- Zastav realtime režim
- Spusť přehrávání
- Zastav přehrávání
- Změň rychlost přehrávání
- Přejdi na předchozí událost
- Přejdi na následující událost
- Přejdi na první událost
- Přejdi na poslední událost
- Přejdi na událost výměrem s pomocí myši
- Ulož seznam událostí
- Načti seznam událostí
- Smaž seznam událostí

3.3.4 Task graph

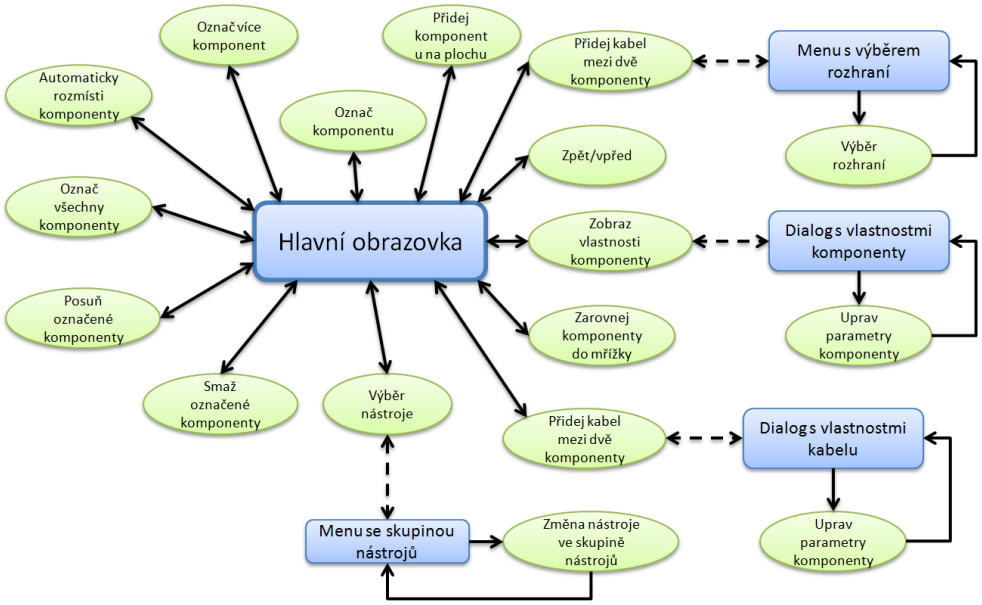
Na závěr této podkapitoly je z výše rozdělených uživatelských akcí vytvořen tzv. Task graph[7], který umožní lepší orientaci v uživatelských akcích. Graf je pro přehlednost rozdělen do tří částí podle výše uvedených kategorií a každá kategorie má v grafu svou barvu. Společné akce jsou červené, akce na editoru zelené a akce na simulátoru oranžové.

V první části grafu (obrázek 3.1) jsou uživatelské akce společné pro oba režimy. V druhé části (obrázek 3.2) se nachází akce, které lze provádět v editačním režimu a v třetí části (obrázek 3.3) jsou uživatelské akce pro simulační režim.

[7]Task graph - vizualizace uživatelských akci, která pomáhá strukturovat vztah jednotlivých uživatelských akcí.



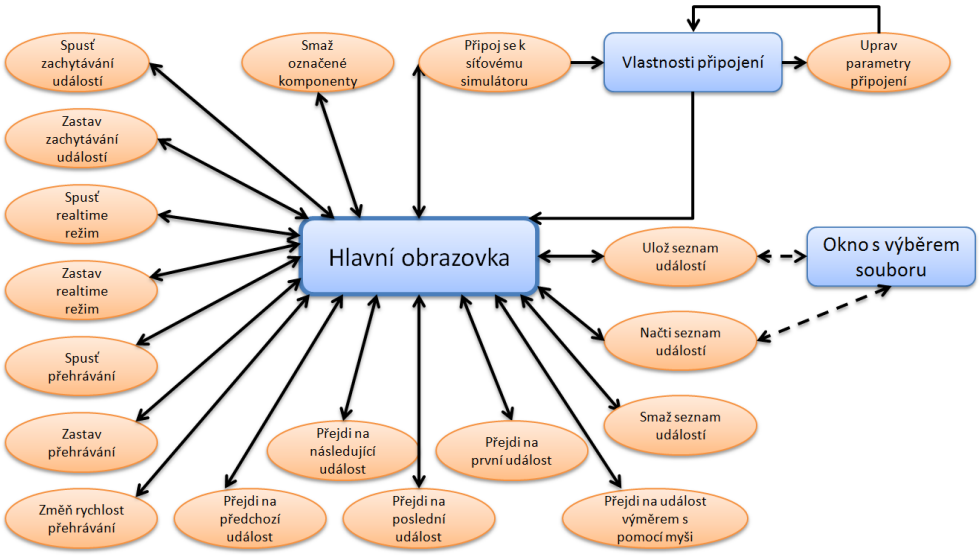
Obrázek 3.1: Task graph - společné uživatelské akce



Obrázek 3.2: Task graph - akce v editačním režimu

3.4 Použité technologie

Při vývoji programu je důležité nepodcenit správnou volbu použitých technologií. Vhodným výběrem programovacího jazyka, vývojového prostředí, knihoven a dalších technologií si můžeme během návrhu a implementace aplikace



Obrázek 3.3: Task graph - akce v simulačním režimu

usnadnit práci, proto je důležité se výběrem technologií zabývat už ve fázi návrhu aplikace.

3.4.1 Programovací jazyk a vývojové prostředí

Vzhledem k tomu, že projekt PSImulator se skládá ze čtyř diplomových prací, je vhodné jednotně zvolit programovací jazyk. Tato volba umožní sdílení společného kódu mezi jednotlivými komponentami a zabrání duplicitám a vnášení chyb během návrhu a implementace.

Základním požadavkem na jazyk byla znalost a zkušenosti s ním. Druhým požadavkem byla multiplatformnost výsledné aplikace. Takové požadavky splňuje programovací jazyk Java, který přináší programátorský komfort, stabilitu a možnost výslednou aplikaci používat na více operačních systémech. Zároveň máme s tímto jazykem velké zkušenosti ze školních projektů. Další výhodou jazyka Java jsou rozsáhlé knihovny pro práci s grafikou v balíčcích *java.awt* a *java.swing*, pro práci se sítí *java.net*, s různými jazykovými verzemi ve třídě *java.utils.ResourceBoundle*, s ukládáním nastavení aplikace *java.util.prefs.Preferences* a s datovým formátem xml v balíčku *javax.xml*. Využity budou zajisté i další balíčky z rozsáhlé knihovny jazyka Java.

Pro implementaci programu jsem si zvolil vývojové prostředí NetBeans ve verzi 7.1. Hlavním důvodem byla dlouhodobá praxe s tímto nástrojem.

3.4.2 Obrazový formát PNG

Jako obrazový formát pro obrázky v aplikaci byl zvolen bezeztrátový formát PNG. Rozměry obrázků jednotlivých komponent (PC, switch, router) v aplikaci se budou měnit jen do určité míry a proto není potřeba využít vektorový obrazový formát. Stačí použít větší rozměr obrázků (šířka okolo 140px) a vhodným způsobem je zmenšovat. Nespornou výhodou formátu PNG je velké množství volně dostupných ikon a obrázků, které je možné při respektování jejich licence snadno využít. Nebude tedy nutné navrhnout a vytvořit vlastní obrázky, což je proces, který je velmi náročný jak časově, tak na estetické cítění autora.

3.4.3 Grafický framework

Jedním z nejdůležitějších rozhodnutí je způsob realizace editační části programu. Jak plyne z analýzy požadavků, aplikace bude muset podporovat drag&drop[8] operace s pomocí myši, zoom, označování komponent, spojování komponent kabelem a mnoho dalších. Na trhu existují knihovny, které více či méně uvedené požadavky splňují a stojí za úvahu se rozmyslet, zda místo implementace vlastního řešení nevyužít nějaké existující.

Nejzajímavějším nástrojem, který jsem při analýze vhodných řešení nalezl, je NetBeans Visual Library[31]. Tato knihovna má podporu pro většinu z uvedených požadavků, ale přináší řadu nevýhod. Největší nevýhodou je, že s ní nemám vůbec žádné zkušenosti a musel bych se ji velmi dobře naučit, abych byl schopný ji efektivně využít. Další nevýhodou řešení s pomocí této knihovny je obtížnost přidávání nových funkcí. Pokud bych během vývoje narazil na funkci, kterou v knihovně postrádám, bylo by velmi náročné knihovnu doplnit. Už během zkoumání vlastností knihovny Visual Library jsem narazil například na absenci nástroje pro vizualizaci vícenásobného propojení komponent (dva paralelní spoje).

Po důkladné analýze existujících řešení a po konzultacích s komunitou vývojářů jsem došel k závěru, že nejlepším řešením bude vytvoření vlastního nástroje, který bude podporovat všechny požadované vlastnosti. Takové řešení přináší obrovskou výhodu, a to plné přizpůsobení všem uvedeným požadavkům. Řešení bude vytvořeno přímo na míru s ohledem na zaměření aplikace a její uživatele. Další výhodou tvorby vlastního řešení je cenné zdokonalení se v programování grafických záležitostí v jazyce Java.

3.4.4 Timing framework

Už během analýzy požadavků jsem přemýšlel o tom, jak vytvořím animace průchodu paketů sítí. Problémem u animace je její správné časování. Animace je v podstatě rychlé překreslování obrazu, u kterého lidské oko nezaznamená

[8] Drag&drop - uchopení, posun a puštění objektu použitím myši.

jednotlivé změny a bere obraz jako ucelený plynulý pohyb. Při zobrazení přesunu obrázku z bodu A do bodu B je důležité, jak dlouho má vizualizace trvat. To je dáno dobou zpoždění na kabelu a nastavenou rychlostí přehrávání. Rychlost pohybu obrázku po obrazovce závisí ještě na délce virtuálního kabelu. Na kabelu stejných vlastností, ale dvojnásobné délky a při stejné rychlosti přehrávání, musí být pohyb animovaného objektu dvakrát rychlejší, aby doba běhu byla totožná. Z toho vyplývá, že je potřeba v každém kroku animace (např. každých 25ms) vědět, kolik procent z doby trvání animace uplynulo. Z tohoto údaje a z informace o pozici komponent A a B už snadno dopočítáme aktuální polohu, na které se má animovaný objekt vykreslit.

Přesně k tomuto účelu slouží nástroj jménem Timing Framework[37], který vytvořil Chet Hasse a v roce 2011 předal vedení projektu Timu Halloranovi. Knihovna existuje v několika verzích, včetně verze pro Java Swing[9] a je dostupná pod licencí BSD. Poslední uveřejněnou verzí pro Swing je 4.01. Knihovna umožňuje vytvořit časovač (timer) a k tomuto časovači připojit tzv. Timing Target, což je objekt, který má být po uplynutí časového intervalu (např. 25ms) informován o změně. Využití této knihovny velmi usnadní tvorbu animací.

3.4.5 Síťový protokol

Vhodným protokolem pro síťovou komunikaci mezi grafickým uživatelským rozhraním PSImulatorUI a síťovým simulátorem PSImulator je protokol TCP. Důvodem pro tuto volbu je jistota doručení zasílaných paketů a také charakter přenášených dat (velmi krátké informace o událostech). Další výhodou této volby je podpora TCP spojení v balíčku *java.net* jazyku Java.

3.4.6 XML

Jako datový formát pro ukládání konfigurace sítě a zachyceného seznamu událostí byl na základě nefunkčních požadavků FP 5 a 6 vybrán formát XML. Tento formát přináší výhodu v možnosti úpravy uloženého souboru v téměř jakémkoliv textovém editoru. Další výhodou formátu XML společně s balíčkem *javax.xml* je částečné zachování zpětné kompatibility uložených souborů. Přidání nějakého parametru například ke kabelu nebrání v načtení dříve uloženého souboru, jelikož se místo chybějící hodnoty nového parametru dosadí výchozí hodnota.

3.4.7 JTA - Telnet/SSH for the JAVA platform

Vzhledem k tomu, že přístup k jednotlivým virtuálním zařízením v aplikaci PSImulator bude možný s pomocí telnet protokolu, bylo by vhodné umožnit vytvoření spojení přímo z grafického prostředí. K tomu jsme využili software

[9] Java Swing - sada nástrojů pro tvorbu grafických uživatelských rozhraní v jazyce Java.

jménem JTA[28], který pro naše potřeby upravil Martin Lukáš. Jedná se o aplikaci, která umožňuje v prostředí Swing vytvořit telnet klienta a začlenit ho do Java aplikace. V případě znalosti portu a IP adresy PSImulatoru je možné jednoduše se připojit přímo ke konkrétnímu virtuálnímu zařízení.

3.4.8 Nástroj pro vytvoření spustitelného souboru s ikonou

Využití programovacího jazyka Java a JVM přináší i své nevýhody, které se nejvíce projeví na operačním systému Windows, jehož uživatelé jsou zvyklí na určitý komfort. Například při spuštění java programu a absenci vhodného JRE se uživatel nedozví, co je špatně a místo toho na něj vyskočí varování, kterému běžný uživatel nerozumí. Další nevýhodou je nemožnost změnit ikonu spustitelného *.jar* souboru tak, jak jsme zvyklí.

Výše uvedené problémy řeší nástroj jménem Launch4j[29] (BSD licence). Tento nástroj umí v systému vyhledat vhodné JRE a případně využít přibalené (pokud existuje), odkázat na stažení vhodného JRE, změnit ikonu spustitelného souboru či nastavit jméno běžícímu procesu. Nástroj funguje tak, že vytvoří specifický spustitelný *.exe* soubor, který spouští náš *.jar*. Díky využití tohoto nástroje bude spouštění vytvořené aplikace pro uživatele pocitově stejné, jak u kterékoliv jiné aplikace.

3.4.9 Nástroj pro vytvoření nápovědy

Java SE jako taková neobsahuje nástroj pro vytvoření nápovědy. Bylo by jistě možné vytvořit okno s nápovědou ze standardních grafických komponent, naprogramovat pohyb v kapitolách a zobrazování jednotlivých témat (html souborů), ale to by byla zbytečná práce, protože existuje nástroj JavaHelp[27]. Tato knihovna v poslední verzi 2.0.05 z 3.10.2007 nabízí přesně tu funkcionalitu, kterou potřebujeme:

- Soubory s nápovědou ve formátu HTML,
- rozdělení nápovědy do témat a kapitol,
- využití hypertextových odkazů v rámci témat,
- vkládání obrázků,
- fulltextové vyhledávání v nápovědě,
- vícejazyčnost nápovědy.

Díky nástroji JavaHelp bude možné k aplikaci připojit vhodnou nápovědu, která uživatelům pomůže v používání programu.

KAPITOLA 4

Návrh

V této kapitole se budeme věnovat návrhu aplikace. Nejprve se podíváme na návrh uživatelského rozhraní a poté na návrh architektury aplikace a důležitých částí systému.

4.1 Návrh uživatelského rozhraní

V předchozí kapitole jsme se věnovali analýze požadavků na uživatelské rozhraní (3.3). Analyzovali jsme požadavky na systém, vytvořili jsme seznam úkonů, které bude uživatel se systémem provádět, a vytvořili tzv. Task graph. V této části návrhu se budeme věnovat vytvoření prototypu grafického uživatelského rozhraní a jeho ověření.

4.1.1 Lo-Fi prototyp

Pojem Lo-Fi prototyp, někdy zvaný wireframe, či mockup, by se dal do češtiny přeložit jako prototyp, neboli základní představa o rozložení komponent. Prototyp se vytváří buďto ve specializovaném programu, nebo jednoduše s pomocí tužky a papíru. Zásadní výhodou tohoto přístupu k návrhu uživatelského rozhraní je rychlost vytvoření prototypu a možnost ho kdykoliv snadno změnit. Je jasné, že během návrhu uživatelského rozhraní a jeho vyhodnocování vznikne mnoho požadavků na změny. Při použití technik prototypování je možné tyto změny v krátkém čase zapracovat a tím ušetřit čas v následujících částech vývoje.

Při tvorbě prototypu je možné postupovat buď odshora, tzn. nejdříve navrhnout obecné rozmístění prvků a až poté se soustředit na detaily, nebo odspodu, tzn. nejdříve navrhnout jednotlivé prvky a poté je skládat do větších celků. Nejlepším možným přístupem je kombinace obou zmíněných, tzn. nejdříve se zamyslet nad jednotlivými prvky a prostorem, který potřebují, poté

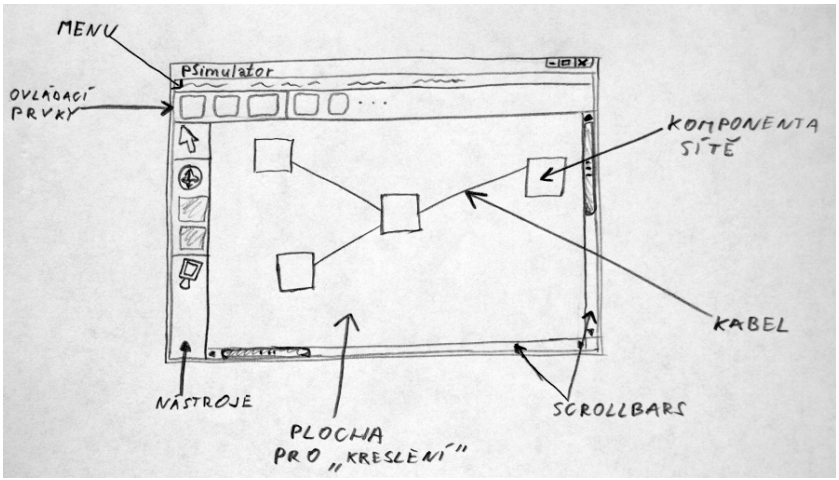
navrhnout jejich rozmístění a následně upravit detaily jednotlivých prvků. Nejlepších výsledků je možné dosáhnout iterací takového přístupu.

V případě tvorby programu specifického velkým množstvím možných rozestavení grafických komponent (např. editor) je možné s ohledem na budoucí testování prototypu vytvořit tzv. interaktivní prototyp. Takový prototyp se skládá z jednotlivých prvků uživatelského rozhraní, v našem případě např. hlavní okno, okno s nastavením programu či ikona komponenty.

Následuje detailnější popis hlavních součástí uživatelského rozhraní. Kompletní sada obrázků prototypu je v příloze B:

4.1.1.1 Hlavní okno aplikace

Při návrhu hlavního okna aplikace bylo důležité respektovat rozmístění, které běžně mívají aplikace v dnešních desktopových operačních systémech. Okno aplikace má menu bar (soubor, úpravy), toolbar (ovládací prvky aplikace a často používané funkce), panel nástrojů a plochu pro kreslení se scrollbary. Návrh okna i s náčrtem virtuální sítě se nachází na obrázku 4.1.

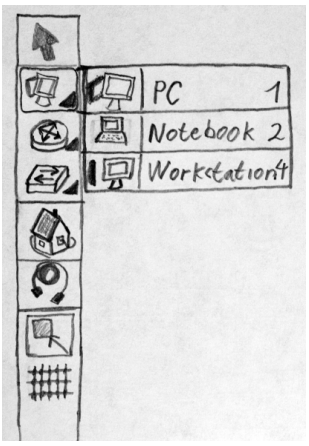


Obrázek 4.1: První návrh uživatelského rozhraní bez detailů

4.1.1.2 Menu s nástroji

Důležitou součástí editoru je menu s nástroji. Při jeho návrhu jsem se inspiroval nástrojovým menu programu Adobe Photoshop. Jak je vidět na obrázku 4.2, některá tlačítka mají v pravém dolním rohu černý trojúhelník, což znamená, že pod tímto tlačítkem je schované podmenu, dostupné s pomocí pravého tlačítka myši. V podmenu zvolíme vhodný nástroj a tento nástroj se stane výchozím v tomto menu. Pokud budeme chtít znovu vybrat ten samý ná-

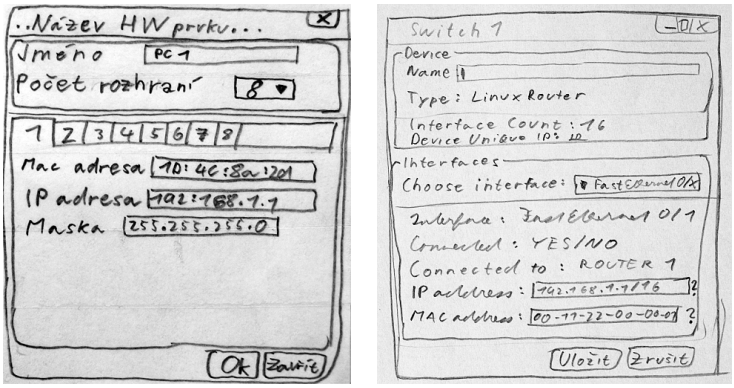
stroj, postačí kliknutí levým tlačítkem myši v nástrojovém menu (není nutné otevírat podmenu).



Obrázek 4.2: Režim editoru - menu s výběrem nástroje

4.1.1.3 Okno s nastavením komponenty

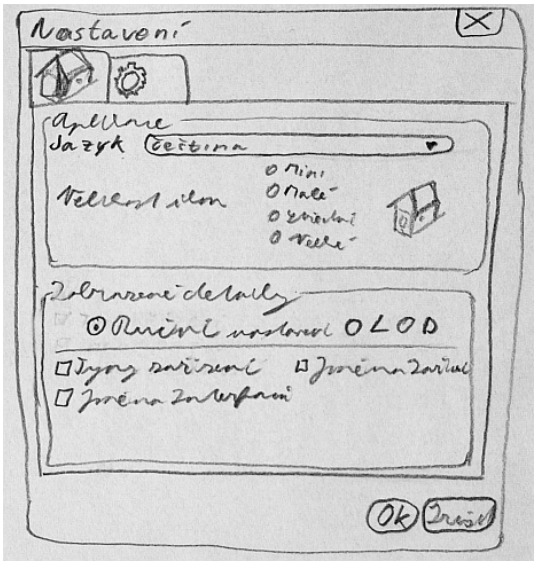
U okna s nastavením vlastností komponenty nebylo jasné, jak by mělo vypadat. Problémem byl způsob zobrazení vlastností jednotlivých rozhraní komponenty. Z tohoto důvodu jsem vytvořil dva návrhy. První z nich využívá záložky, kdy každé rozhraní má svou. Druhý návrh využívá tzv. drop-down menu, ve kterém uživatel vybere rozhraní, jehož detaily si přeje nastavit. Obě varianty jsou na obrázku 4.3.



Obrázek 4.3: Dvě varianty okna s nastavením komponenty. Vlevo využití záložek, vpravo drop-down menu.

4.1.1.4 Okno s nastavením programu

U okna s nastavením programu jsem se musel vypořádat s rozmístěním ovládacích prvků. V nastavení je potřeba měnit jazyk, detaily zobrazení i nastavovat parametry simulátoru. Okno jsem s využitím záložek rozdělil na dvě části - obecná nastavení a nastavení simulátoru. Výsledek návrhu tohoto okna je na obrázku 4.4.



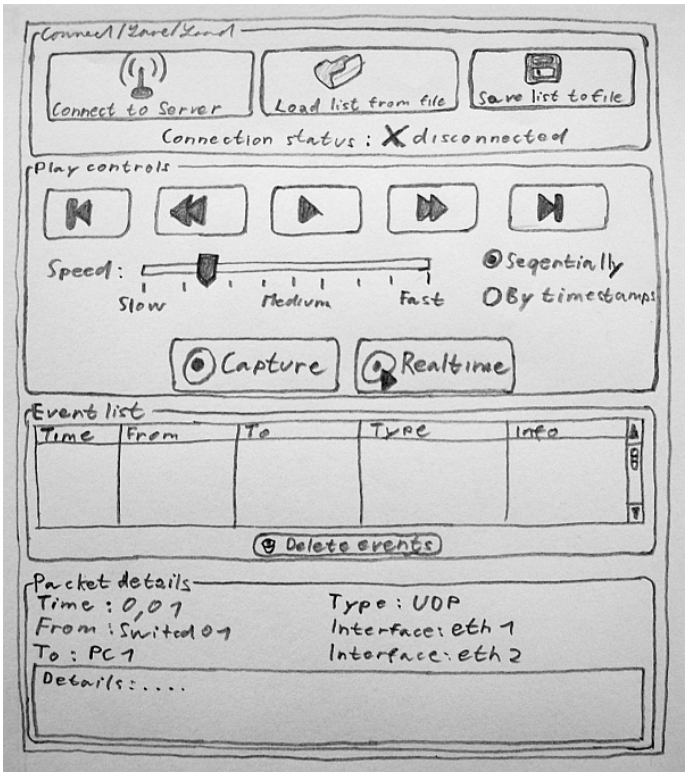
Obrázek 4.4: Okno s nastavením programu

4.1.1.5 Ovládací panel simulátoru

Nejtěžším prvkem grafického uživatelského rozhraní z hlediska návrhu je Ovládací panel simulátoru. Důvodem je velké množství funkcí, které musí integrovat, spolu s požadavkem na přehlednost a jednoduché použití. Ovládací panel musí sdružovat následující funkce: přehrávání, nastavení rychlosti, změna typu přehrávání, ovládání záznamu událostí, tabulku s přehledem událostí, detaily události, ukládání a načítání seznamu událostí, propojování se serverem a další. Návrh ovládacího panelu je na obrázku 4.5.

4.1.2 Vyhodnocení bez uživatelů

Při vyhodnocování kvality navrženého prototypu je potřeba nejprve vyhodnotit, zda návrh vyhovuje zažitým pravidlům a tzv. „Best practices“. K tomuto účelu slouží Nielsenova heuristická analýza od Jakoba Nielsona[32], která se skládá z následujících deseti pravidel[34]:



Obrázek 4.5: Režim simulátoru - ovládací panel

1. **Viditelnost stavu systému:** Systém nesmí zůstat „zamrzlý“, uživatel musí být informován o tom, co systém dělá.

2. **Shoda mezi systémem a realitou:** Ikony musí vypadat jako to, co mají reprezentovat. Zachování konvencí z reálného světa.

3. **Minimální zodpovědnost:** Možnost vrátit změny zpět (undo). Zrušení dlouhotrvajících akcí, potvrzování akcí.

4. **Shoda s použitou platformou a obecnými standardy:** Program pod Windows vypadá a chová se jako pod windows apod.

5. **Prevence chyb:** Uživatel by neměl mít možnost zadat špatnou hodnotu. Při složitějším vstupu vysvětlit, co systém požaduje.

6. **Kouknu a vidím:** Nezatěžovat uživatelovu paměť. Akce by měly být viditelné a snadno dosažitelné.

7. **Flexibilita a efektivita:** Použití klávesových zkratek, funkčních kláves. Pokročilý (advanced) mód.

8. Minimalita: Zobrazovat pouze informace, které jsou aktuálně potřeba. Grafika by neměla zastiňovat ovládání a účel

9. Smysluplné chybové hlášky: Chybové hlášky ve smysluplném jazyce. Hlášení by mělo popsat co se stalo špatně.

10. Nápověda a dokumentace: Systém by měl být použitelný bez nápovědy, ale nápověda musí být. Spíše než popisem by se měla zabývat příklady.

Uvedená pravidla slouží primárně jako pomůcka pro testování systému experty, ale měli bychom na ně myslet už při návrhu grafického rozhraní.

4.1.2.1 Testování experty

Expertem je člověk vzdělaný v oboru IT se zkušeností s návrhem uživatelských rozhraní. Experty, kteří testovali uživatelské rozhraní, byli Bc. Martin Vehovský a Ing. Zdeněk Klíma. Po fázi návrhu (vytvoření prototypu) ještě nemáme kompletní grafické rozhraní, ale i tak již můžeme provést první otestování. Nalezené problémy jsou následující:

- Chybí návrh klávesových zkratek.
- V podmenu s výběrem nástroje není jasné, co znamená uvedené číslo.
- V původním návrhu okna s vlastnostmi komponenty nevyhovuje zobrazení rozhraní s pomocí záložek, protože není vidět celkový přehled o rozhraních.
- Ve vylepšeném návrhu okna s vlastnostmi komponenty nevyhovuje zobrazení rozhraní s pomocí rozbalovacího menu, opět není vidět celkový přehled o rozhraních.

Nalezené nedostatky jsou zapracovány při vytvoření Hi-Fi prototypu v následující kapitole.

4.1.3 Ikony

Pro návrh grafického uživatelského rozhraní jsou důležité použité ikony, na které klademe několik požadavků. Prvním z nich je vyjádření funkce srozumitelným obrázkem. Ikona má bez nutnosti vysvětlování vyjadřovat funkci, kterou zastupuje. Druhým požadavkem je sladění ikon, jelikož vybrané ikony by měly ladit dohromady po estetické stránce. Posledním požadavkem je licence ikony, která musí umožňovat minimálně užití pro nekomerční použití.

4.1.3.1 Hlavní sada ikon

Jako hlavní sadu ikon pro program jsem zvolil Glaze icon set od Marco Martina[8]. Tato sada ikon obsahuje 149 ikon, mezi kterými jsem našel většinu potřebných pro tento program. Ikony jsou k použití pod licencí LGPL[30].

4.1.3.2 Další užité ikony

Ikony notebooku a pracovní stanice pochází ze sady ikon Icons Unleashed Vol. 1 - Computer Hardware[12] ze serveru PC Unleashed a jsou zdarma pro osobní i komerční užití, stejně jako ikona obálky[17] pocházející ze sady ikon Practika od DryIcons.

Ikona[11] pro nástroj ruka pochází ze serveru Gentleface.com a podléhá licenci Creative Commons (Attribution-Noncommercial 3.0)[3], stejně tak jako ikona balíčku[14], ikona výběru[22] a ikona nápovědy[16].

Ikona terminálu[21] je k použití pod licencí LGPL[30] a ikony pro automatické rozmístění[19] spolu s ikonou XML[24] podléhají licenci GPL[10].

Ikony ethernetového kabelu[6], počítače[18], ikona auta[13] a ikona reálného počítače[20] jsou zdarma pro nekomerční použití.

Jako ikony pro hardwarové prvky router a switch jsem použil standardní a všeobecně uznávané ikony[25] firmy Cisco, které jsou volně k použití.

4.1.3.3 Vytvořené ikony

K některým funkcím, které vyžadují přítomnost ikony, se mi nepodařilo nalézt odpovídající ikonu. Takovými funkcemi jsou: zarovnání do mřížky a zmenšení kreslící plochy. Ikony pro tyto funkce jsem musel vytvořit a tudíž jsem jejich autorem.

4.2 Architektura aplikace

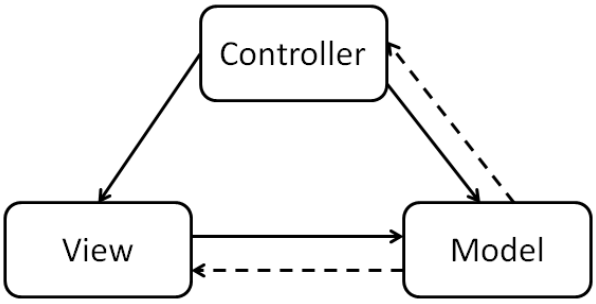
Návrh architektury aplikace je důležitý pro celý proces vývoje, jelikož správný návrh nám může usnadnit práci jak při návrhu jednotlivých částí aplikace, tak při implementaci. Vždy je lepší využít nějaký ověřený architektonický vzor, než budovat aplikaci jako monolit. Rozdělení aplikace do menších částí usnadní orientaci v kódu a umožní jednodušší úpravy.

Jako vhodný architektonický vzor byl vybrán vzor MVC (obrázek 4.6) (Model-View-Controller). Jak již název napovídá, aplikaci je možné podle vzoru rozdělit na tři části:

Model : Uchovává data a přistupuje k datům.

View : Zobrazuje data z modelu, přijímá informace o změně stavu modelu.

Controller : Přijímá požadavky od uživatele, obstarává aplikační logiku.



Obrázek 4.6: Architektonický vzor Model-View-Controller

V případě jazyku Java a využití komponent balíčku *javax.Swing* splývá zobrazování dat a interakce s uživatelem do jednoho celku a to z důvodu svázání ovládacích prvků aplikace s obslužnými metodami těchto prvků. Bylo by sice možné zobrazování a obslužné metody oddělit, ale bylo by to velmi nekomfortní pro programátora a velmi by to znepřehlednilo kód. Takový návrhový vzor se jmenuje Model-Delegate.

V případě této aplikace je výsledná architektura kombinací obou vzorů. V části zvané View se nachází zobrazování a interakce s uživatelem, v části Model se uchovávají data a přistupuje k souborům a v části Controller se nachází logika přehrávání paketů a komunikace se serverem.

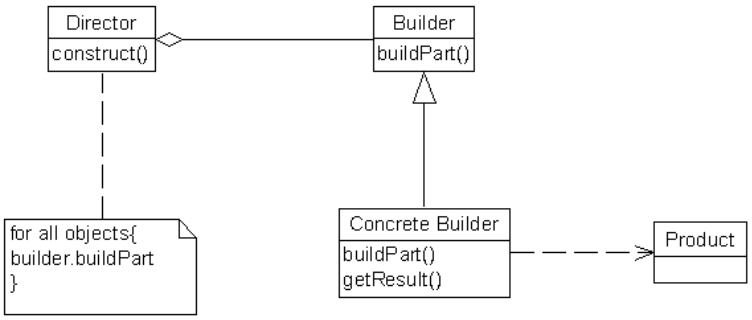
4.3 Využité návrhové vzory

Návrhový vzor je obecné, znovu použitelné řešení běžně se objevujících problémů v návrhu softwaru. Jedná se o „šablonu“, která nám dává návod, jak problém co nejlépe řešit. Návrhové vzory dělíme do tří kategorií: vzory tvorby (creational), vzory struktury (structural) a vzory chování (behavioral). Při návrhu a implementaci aplikace PSImulatorUI jsem využil níže uvedené návrhové vzory. Obrázky jednotlivých vzorů jsem čerpal z internetových stránek objekty.vse.cz[33].

4.3.1 Builder

Návrhový vzor Builder (obrázek 4.7) patří mezi vzory tvorby a odděluje proces konstrukce (jak) od konkrétní reprezentace (co). Vzor se skládá ze čtyř částí: Director, Builder, Concrete Builder a Product. Builder specifikuje abstraktní rozhraní pro tvorbu jednotlivých částí produktu. Concrete Builder implementuje rozhraní Builderu a je zodpovědný za tvorbu a skládání výsledného produktu. Director tvoří výsledný produkt s využitím Builderu a Product reprezentuje komplexní objekt, který tvoříme.

Vzor Builder je vhodné využít při vytváření komplexních objektových struktur. Při realizaci je tento návrhový vzor použit při tvorbě grafu, což je název pro grafickou reprezentaci modelu počítačové sítě. Model se předloží Directoru, který zajistí s pomocí Builderu vytvoření komplexně propojené struktury.



Obrázek 4.7: Návrhový vzor Builder

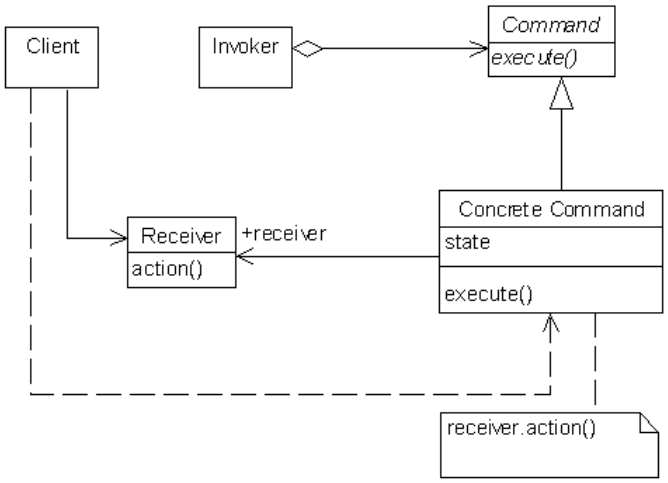
4.3.2 Command

Návrhový vzor Command (obrázek 4.8) patří mezi vzory chování. Zapouzdřuje nějakou činnost do objektu a odstiňuje klienta od procesu zpracování jeho požadavku. Vzor se skládá z pěti částí. První je třída Command, která definuje rozhraní pro vykonání operace. Druhým prvkem je ConcreteCommand, který definuje spojení mezi Recieverem a akcí a implementuje metodu *execute()*, která je zodpovědná za provedení požadavku. Client vytváří ConcreteCommand a určuje jeho Reciever. Invoker vyvolává metodu *execute()* na Commandech a poslední součást Reciever je příjemcem, na kterém ConcreteCommand provádí akci.

Tento návrhový vzor má výhodu v oddělení objektu, který ví, jak vykonat určitou akci, od objektu, který akci spouští. Další výhodou je jednoduché přidávání dalších potomků třídy Command. Příkladem použití tohoto vzoru jsou undo/redo operace. Například přidání komponenty do kreslící plochy může vyvolat přidání nové instance třídy *addComponentCommand* do seznamu provedených akcí. Při potřebě operace undo se na tomto commandu zavolá metoda *undo()* a objekt sám provede odstranění komponenty.

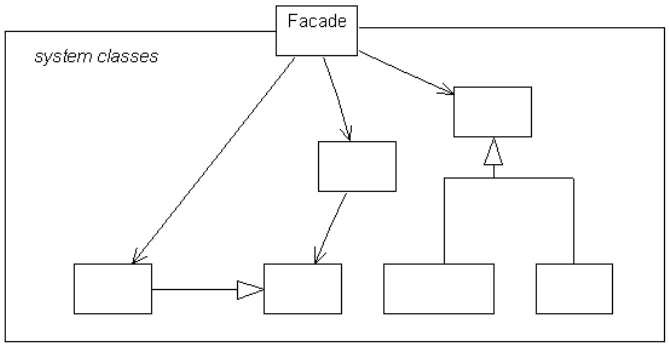
4.3.3 Facade

Facade (obrázek 4.9) je strukturální návrhový vzor, který má poskytnout jednoduché rozhraní ke složitému podsystému (skupině tříd). Vzor se skládá ze dvou částí. První je facade, což je třída, která má povědomí o třídách uvnitř



Obrázek 4.8: Návrhový vzor Command

systému a deleguje na ně požadavky. Druhou část tvoří třídy skrytého systému, které implementují funkcionalitu a zároveň neví o existenci facade. Výhodou tohoto vzoru je vytvoření slabé vazby[10] mezi podsystémem a jeho klienty (ti co přistupují k facade). Další výhodou je, že vzor facade nebrání v případě potřeby v přístupu k jednotlivým třídám podsystému.



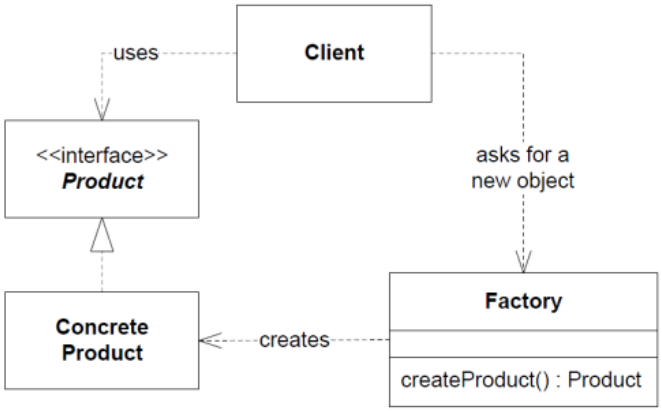
Obrázek 4.9: Návrhový vzor Facade

[10]Slabá vazba - třída je závislá na abstrakci, ne přímo na druhé třídě.

4.3.4 Factory

Factory (obrázek 4.10) je návrhový vzor z kategorie tvorby. Umožňuje vytvářet objekty bez vystavení principu vytváření klientovi, tzn. odstiňuje klienta od problematiky vytváření určitých objektů. To může být užitečné v případě, že vytvářené objekty nejsou úplně jednoduché, a nebo v případě, kdy chceme soustředit vytváření objektů na jedno místo. V tomto návrhovém vzoru vystupují: Product, Client a Factory. Product definuje rozhraní objektu, který Factory vytváří. Client žádá Factory o vytvoření Productu a poté Product používá. Factory je zodpovědná za tvorbu Productu.

Příkladem použití vzoru Factory může být vytváření jednotlivých komponent sítě v datovém modelu (kabel, komponenta, rozhraní...).



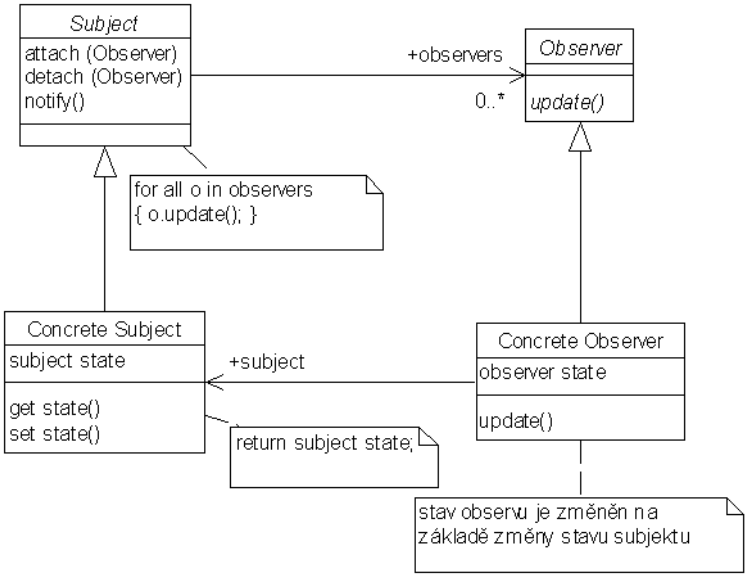
Obrázek 4.10: Návrhový vzor Factory

4.3.5 Observer

Observer (obrázek 4.11) patří do kategorie návrhových vzorů chování a umožňuje třídě, která ho implementuje, informovat závislé objekty o změně stavu. V tomto vzoru vystupují následující třídy: Subject, Observer, ConcreteSubject a ConcreteObserver. Subject ví o svých observerech (objekty, které chtějí být informovány o změnách) a poskytuje rozhraní k připojení, či odpojení observerů. Observer definuje rozhraní pro upozornění o změně stavu (většinou metoda *update*). ConcreteSubject posílá upozornění o změně stavu jednotlivým observerům. ConcreteObserver uchovává referenci na ConcreteSubject a implementuje rozhraní Observeru pro příjem upozornění o změně stavu.

Návrhový vzor observer se hodí všude tam, kde potřebujeme upozornit množinu objektů na změnu stavu a zároveň potřebujeme upozorňované objekty přidávat a odebírat. Příkladem použití může být například ZoomManager, který se má starat o změnu zoomu. Při jeho změně informuje všechny

zaregistrované objekty o tom, že se změnila úroveň přiblížení a dané objekty na tuto zprávu reagují. V programovacím jazyku Java je Subject realizován jako třída Observable a Observer je realizován jako rozhraní Observer. Nemusíme tedy tyto třídy implementovat a rovnou je můžeme použít.

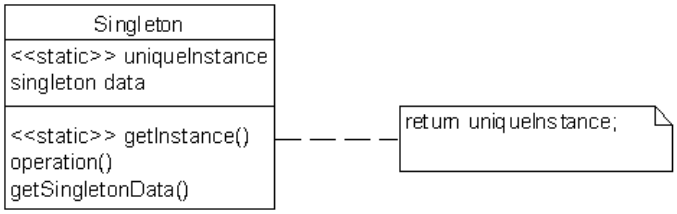


Obrázek 4.11: Návrhový vzor Observer

4.3.6 Singleton

Návrhový vzor Singleton (obrázek 4.12) patří mezi vzory tvorby a zajišťuje, že třída může mít pouze jednu instanci, která je dostupná odkudkoliv. Třída, která má být singletonem, musí definovat operaci, s pomocí které získáme její instanci. Zároveň musí zabránit vytvoření objektu z vnějšku, čehož docílí privátním konstruktorem. Při využití privátního konstruktoru je zajištěno, že instanci může vytvořit pouze tato třída. Pro přístup k instanci se využívá veřejná statická metoda *getInstance()*, která vrací instanci třídy. Pokud ještě instance nebyla vytvořena, je tato metoda zodpovědná za její vytvoření. Samotná instance je uložena v privátní statické proměnné.

Výhodou návrhového vzoru Singleton je možnost přístupu k instanci třídy odkudkoliv z celého systému. To se může hodit například u třídy, která je zodpovědná za načítání obrázků ze souboru a jeho změnu rozměrů. Z kterékoliv části systému můžeme požádat o obrázek, aniž bychom museli složitě předávat reference na instanci, či vytvářet novou.

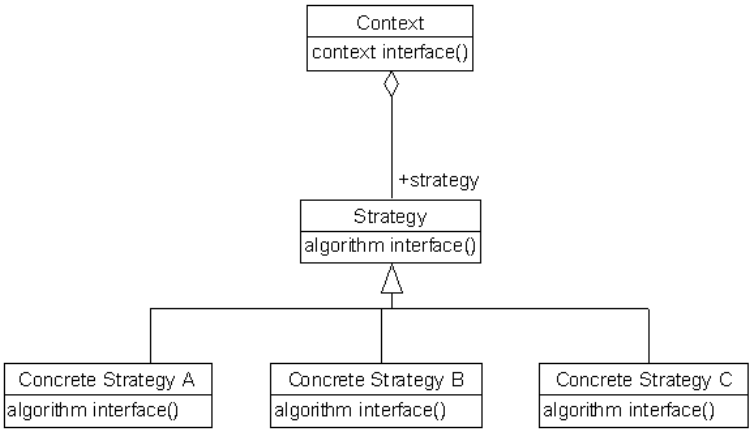


Obrázek 4.12: Návrhový vzor Singleton

4.3.7 Strategy

Strategy (obrázek 4.13) je vzor z kategorie chování, který umožňuje určit rodinu algoritmů, zapouzdřit je a navzájem je zaměňovat. Vzor se skládá z kontextu, strategie a jednotlivých strategií. Kontext uchovává referenci na určitou strategii a může být zodpovědný za výběr strategie. Strategie bývá abstraktní třída nebo rozhraní, které musí být implementováno jednotlivými strategiemi. Jednotlivé strategie implementují abstraktní strategii různým způsobem. Vzor Strategy je vhodné použít, když máme mnoho podobných tříd, které se liší pouze chováním, nebo např. když potřebujeme více variant stejného algoritmu.

Příkladem použití návrhového vzoru Strategy v našem programu jsou nástroje, které chceme používat na kreslící ploše: ruka, vkládání komponent či spojování komponent. Všechny tyto nástroje musí reagovat na tlačítka či kolečko myši a každý nástroj reaguje na jednotlivé události jiným způsobem.



Obrázek 4.13: Návrhový vzor Strategy

4.4 Návrh algoritmu pro automatické rozmístění

V této části kapitoly vybereme vhodný algoritmus pro automatické rozmístění počítačové sítě v ploše a navrhneme, jak by měl algoritmus pracovat. Počítačová síť je defacto graf[11] a dalo by se říci, že v určitých případech i multigraf[12]. Pokud bychom uvažovali mřížku, do které budeme umísťovat jednotlivé uzly (komponenty) o velikosti např. 1,5 násobku počtu uzlů $(1,5 * |V|,\ 1,5 * |V|)$, dostáváme při počtu deseti uzlů 225 políček, do kterých je možné umístit jednotlivé uzly. Z toho plyne, že je možné utvořit $7,4 * 10^{16}$ kombinací. Při snížení počtu políček problém pouze oddálíme, protože při potřebě rozmístění většího množství uzlů bude mřížka opět větší.

Je jasné, že pokud bychom se snažili vyzkoušet všechny možné kombinace hrubou silou, tak bychom se nedočkali výsledku. Řešením tohoto problému může být vhodně zvolený pokročilý heuristický algoritmus, například Genetický algoritmus.

4.4.1 Princip fungování Genetického algoritmu

Genetický algoritmus je heuristický přístup k řešení problémů a je inspirován procesy evoluční biologie: dědičností, mutací, selekcí a křížením. Algoritmus pracuje s generacemi jedinců, se kterými provádí uvedené operace. Každý jedinec představuje jedno řešení daného problému, které nemusí být a zpravidla ani není optimální. Kvalitě jedince se říká fitness a tato hodnota vyjadřuje, jak moc jedinec splňuje požadavky na řešení.

Na počátku algoritmu se vygeneruje úvodní populace. To je možné udělat buď zcela náhodně, nebo nějakou jinou heuristikou. Při přechodu do nové generace se provede ohodnocení jedinců fitness funkcí a do další generace se vyberou určitým způsobem vhodní jedinci právě podle fitness funkce (fáze selekce). Na nové generaci se provedou operace křížení a mutace a poté se opět spočítá fitness. Tento proces se opakuje, dokud není dosaženo ukončující podmínky.

Dále si rozebereme návrh genetického algoritmu na příkladu rozmístění grafu v ploše.

4.4.2 Návrh operací Genetického algoritmu

Pro genetický algoritmus je potřeba vymyslet, jak bude vypadat zakódování jedinců (tzv. genotyp), jak bude vypadat fitness funkce, jak budou prováděny operace selekce, křížení, mutace a jak bude vypadat ukončující podmínka. Neméně důležitou věcí je také velikost populace.

[11] Graf - dvojice dvou množin - vrcholů (V) a hran (E).
[12] Multigraf - graf s násobnými hranami.

Genotyp

Genotyp neboli kódování jedinců by mělo být co nejjednodušší, nejúspornější z hlediska prostorové náročnosti a také by mělo umožňovat provádění operací křížení a mutace. Genotyp jedince v našem případě tvoří pole uzlů se souřadnicemi $[x, y]$ a pole hran, ve kterém jsou dvojice $\{uzel, uzel\}$.

Fitness funkce

Fitness funkce je kritickým bodem genetického algoritmu, protože určuje, který jedinec je kvalitní, a tudíž ovlivňuje celý algoritmus. V případě našeho problému jsou nároky na kvalitu řešení následující:

- Počet překřížených hran co nejmenší,
- odchylka v délce hran co nejmenší,
- rovnoměrné rozložení uzlů v ploše.

Rovnoměrné rozložení uzlů v ploše se vypočítá jako suma minimálních vzdáleností pro každý uzel k nejbližšímu zbylému uzlu.

Problémem je, že operace, které bude muset fitness funkce provádět, jsou výpočetně náročné. Samotné zjištění počtu křížených hran je NP-úplný[7] problém a výpočet odchylek a rovnoměrného rozložení uzlů také není výpočetně nenáročný. Potvrdilo se tak, že výpočet fitness funkce bývá v genetickém algoritmu nejvíce náročnou částí na výpočetní výkon.

Fitness funkce je tedy vypočtena ze tří složek. Jedinec je penalizován za překřížené hrany a za odchylku v délce hran a odměňován za rovnoměrné rozložení uzlů v ploše. Výsledné číslo není nijak normalizované.

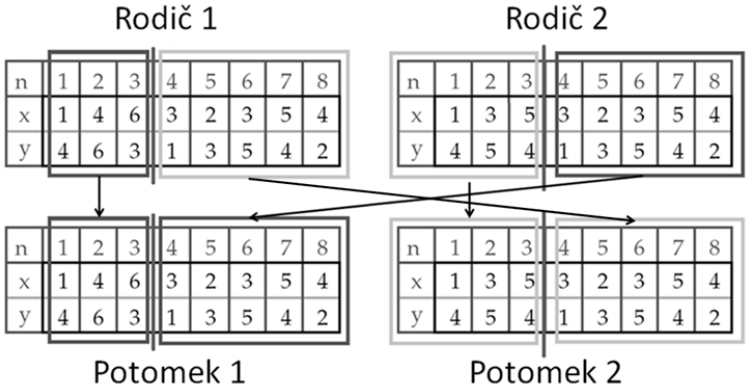
Selekce

Pro selekci je nejprve nutné přepočítat fitness hodnoty na pořadí jedinců, aby mezi nimi nebyly veliké rozdíly a aby měly šanci dostat se do další generace i slabší jedinci. Tomuto postupu přepočtu fitness funkce na pořadí se říká Ranking. Abychom nepřišli o nejlepšího jedince v populaci, využijeme Elitismus, což je přímé převedení nejlepšího jedince do další generace.

Pro samotný výběr použijeme Ruletový výběr. Lepší jedinec má vyšší šanci výběru do další generace než horší jedinec, tudíž na pomyslném ruletovém kole zabírá větší výseč. Vygenerujeme náhodné číslo (roztočíme ruletu) a jedince, na kterého výběr padne, převedeme do další generace. Tento postup opakujeme, dokud není nová generace naplněna. Může se tedy stát (a často se stává), že jedinec bude vybrán vícekrát.

Křížení

Po naplnění nové populace proběhne fáze křížení. Křížení probíhá tak, že se vyberou dva rodiče a vhodným křížením z nich vzniknou dva potomci. V našem případě pole uzlů se souřadnicemi $[x, y]$ u obou rodičů ve stejném náhodném bodě rozdělíme a křížem je spojíme. Tento postup se nazývá jednobodové křížení. Problémem je, že se může stát, že v poli uzlů budou po fázi křížení dva uzly na stejném místě (stejných souřadnicích). Z tohoto důvodu je potřeba oba potomky zkontrolovat a v případě potřeby překryv odstranit. To provedeme nalezením první volné pozice v okolí pro jeden ze dvou uzlů.



Obrázek 4.14: Genetický algoritmus - křížení

Mutace

Po fázi křížení probíhá fáze mutace, která má za úkol v populaci udělat evoluční změny. S určitou pravděpodobností (např. 10%) se na jedinci provede operace mutace. V úloze rozmístění grafu v ploše připadá v úvahu více způsobů, jak jedince mutovat. Pro tuto implementaci jsem vybral následující:

- Náhodná změna pozice uzlu,
- náhodná změna pozice obou uzlů hrany,
- náhodná změna pozice obou uzlů hrany se zachováním délky hrany,
- změna pozice uzlu s nejvíce sousedy.

Uvedené způsoby mutace jsou vybírány náhodně a se stejnou pravděpodobností, tzn. 25%. Po fázi mutace se kontroluje, zda není splněna ukončující podmínka.

Ukončující podmínka

Jak z názvu vyplývá, ukončující podmínka říká, kdy je vhodné algoritmus ukončit. U heuristického algoritmu nevíme, zda zatím nejlepší dosažené řešení je nejlepší možné, pouze víme, jaká je jeho kvalita podle našich kritérií. Vhodnou ukončující podmínkou může být počet generací, během kterých se řešení nezlepšilo.

Po kontrole ukončující podmínky algoritmus buďto končí, nebo pokračuje novou iterací.

Velikosti populace a další parametry

Pro genetický algoritmus je potřeba určit velikost populace, pravděpodobnost křížení a pravděpodobnost mutace. Většinou se tyto hodnoty určují až po implementaci algoritmu a vyzkoušení jeho kvality. Obvyklými hodnotami je velikost populace od 30 jedinců, pravděpodobnost křížení 85% a pravděpodobnost mutace 10%.

4.5 Návrh datového modelu a komunikace

V této části kapitoly se budeme zabývat návrhem dalších datového modelu virtuální sítě a komunikací mezi touto aplikací a síťovým simulátorem.

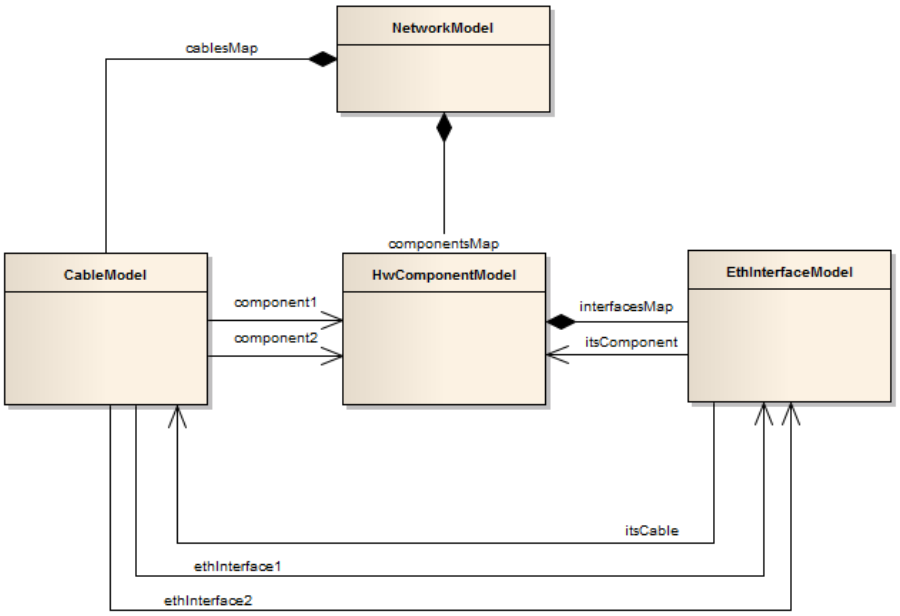
4.5.1 Návrh datového modelu

Důležitou součástí modelu aplikace je datový model virtuální počítačové sítě (diagram 4.15), který obsahuje třídy: *CableModel*, *HwComponentModel*, *EthInterfaceModel* a *NetworkModel*. Každý kabel má referenci na dvě komponenty a dvě rozhraní. Rozhraní má referenci na svůj kabel a svou komponentu. Komponenta má seznam rozhraní. Všechny uvedené třídy mají svůj *HwTypeEnum*, který vyjadřuje jejich typ. Třída *NetworkModel* si udržuje seznam kabelů a komponent.

4.5.2 Návrh komunikace client - server

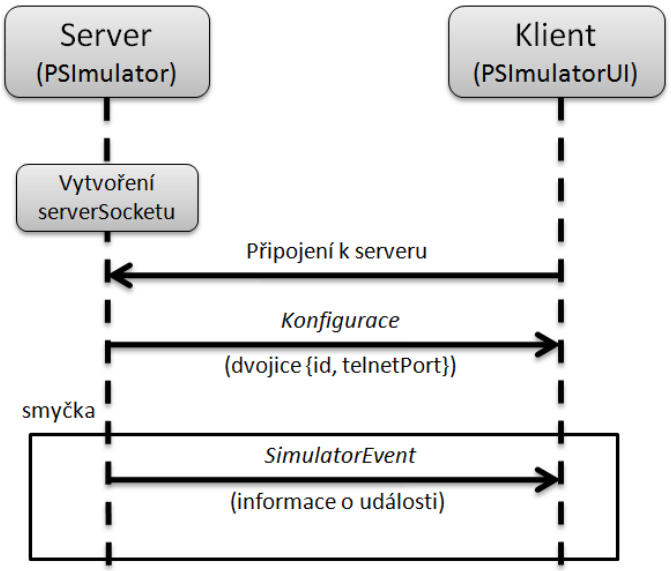
Aplikace PSImulatorUI bude v simulačním režimu při vizualizaci průchodu paketů sítí komunikovat se serverem na straně síťového simulátoru. V předchozí kapitole jsem odůvodnil, proč byl vybrán protokol TCP. Nyní se budeme věnovat návrhu komunikace a datům, která se budou přes komunikační kanál přenášet.

Nejprve je potřeba se k serveru připojit. Server po připojení klienta zašle informace o portech, na kterých poslouchají jednotlivé virtuální stroje. Díky těmto informacím bude možné připojení přímo ke konkrétnímu virtuálního zařízení s pomocí Telnet protokolu. Po této inicializaci bude klient čekat na



Obrázek 4.15: Datový model virtuální sítě

jednotlivé události, které budou nepravidelně přicházet ze serveru. Každá událost bude složena z unikátních identifikátorů dvou virtuálních zařízení, identifikátoru kabelu a informací o typu události. Vizualizace komunikace je na obrázku 4.16.



Obrázek 4.16: Návrh síťové komunikace

KAPITOLA 5

Realizace

V kapitole Návrh jsme zdůvodnili volbu programovacího jazyka Java, ve kterém je projekt PSImulator realizován. Pro jazyk Java je typické rozdělení kódu do balíčků. Balíčky mohou obsahovat další balíčky, či přímo třídy se zdrojovým kódem programu. Rozdělení zdrojového kódu respektuje zvolenou architekturu aplikace a dělí kód do částí *Model*, *View* a *Controller*. Část zdrojových kódů se nachází ve společné (Shared) knihovně, která se dá považovat za součást modelu.

V této kapitole se nejprve zaměříme na funkcionalitu soustředěnou v jednotlivých balíčcích a třídách. Poté se podíváme na zajímavé problémy, které bylo třeba řešit při implementaci a na závěr kapitoly se podíváme na výslednou podobu grafického uživatelského rozhraní.

5.1 PSImulator

Samostatnou část kódu tvoří kořenový balíček s názvem *PSImulator*. V tomto balíčku se nachází jediná třída, a to *Main*. Tato třída je zodpovědná za vytvoření instancí modelu, view a controleru a za spuštění aplikace. Na níže uvedeném kódu je znázorněno vytvoření instancí všech tří komponent s respektováním vzoru MVC. Model neví o ostatních částech programu, view ví pouze o modelu a controller má referenci jak na model, tak na view. V uvedeném kódu je také vidět vytvoření všech tří částí programu v EDT vlákně, což je typické pro programy s grafickým uživatelským rozhraním.

```
SwingUtilities.invokeLater(new Runnable() {
    @Override
    public void run() {
        DataLayerFacade model = new DataLayer();
        MainWindow view = new MainWindow(model);
```

```
        ControllerFacade controller = new Controller(model, view);
    }
});
```

5.2 Model

Do části Model patří třídy a balíčky související s datovým modelem, s ukládáním do souborů či s nastavením programu. Nejdříve jsou popsány využité části ze sdíleného projektu (Shared) a poté jsou rozebrány části přímo z projektu PSImulator, které řadíme do části DataLayer (z pohledu MVC se jedná o model).

5.2.1 Shared

Z důvodu sdílení některých částí kódu v projektech jednotlivých řešitelů bylo výhodné společný kód vyčlenit do samostatného projektu. Tento projekt se nazývá Shared a je tzv. „přilinkován“ k tomuto projektu.

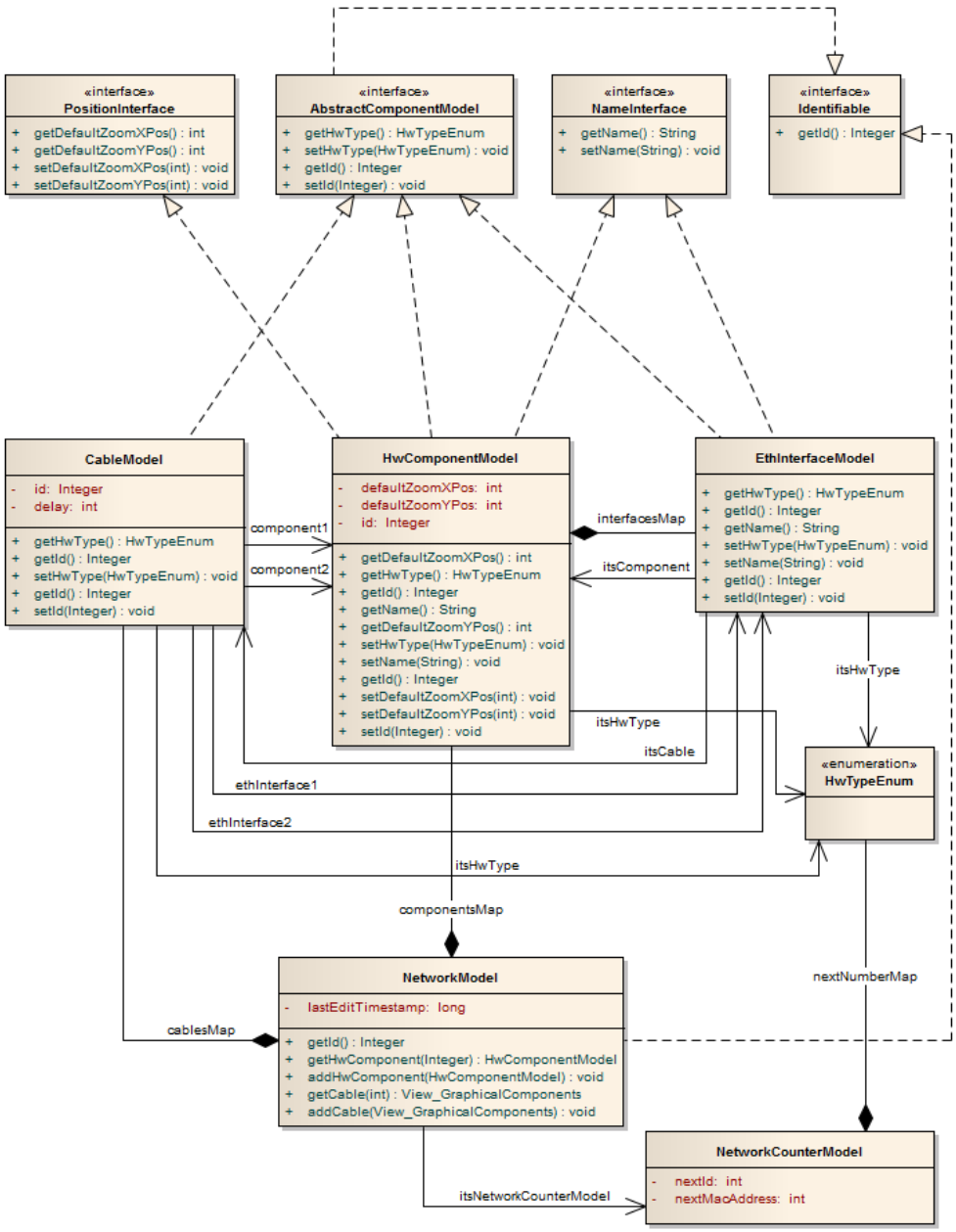
5.2.2 Shared.Components

V balíčku Components se nachází datový model virtuální počítačové sítě (diagram 5.1), který obsahuje třídy: *CableModel*, *HwComponentModel*, *EthInterfaceModel*, *NetworkModel* a *NetworkCounterModel*. Spolu s těmito třídami jsou zde čtyři rozhraní: *PositionInterface*, *AbstractComponentModel*, *NameInterface* a *Identifiable*. Uvedené prvky modelu doplňuje výčtový typ *HwTypeEnum*.

Nejprve se zaměříme na strukturu sítě (*NetworkModel*). Síť obsahuje kabely a komponenty a navíc má ještě referenci na *NetworkCounterModel*, který zajišťuje uložení číselných řad pro automatické generování unikátních jmen a identifikátorů pro jednotlivé typy komponent. Každý kabel (*CableModel*) má referenci na dvě komponenty (*HwComponentModel*) a dvě rozhraní (*EthInterfaceModel*). Rozhraní (*EthInterfaceModel*) má referenci na svůj kabel (*CableModel*) a svou komponentu (*HwComponentModel*). Komponenta (*HwComponentModel*) má list rozhraní (*EthInterfaceModel*). Všechny uvedené třídy mají svůj *HwTypeEnum*, který vyjadřuje jejich typ (router, switch, end device...).

Všechny třídy reprezentující hardwarový prvek (nebo jeho část) implementují rozhraní *AbstractComponentModel*, které umožňuje získat jedinečný identifikátor či typ komponenty (*HwTypeEnum*). Rozhraní *NameInterface*, které umožňuje pojmenování komponenty, implementují třídy *HwComponentModel* a *EthInterfaceModel*. Třída *HwComponentModel* implementuje rozhraní *PositionInterface* s metodami pro nastavení a získání $[x, y]$ souřadnic.

Nejlépe je celé schéma sítě vidět na obrázku 5.1.



Obrázek 5.1: Datový model virtuální sítě

5.2.3 Shared.Serializer

V předchozí kapitole jsme zdůvodnili výběr datového formátu XML. Balíček Serializer tedy obsahuje třídy pro uložení a načtení konfigurace celé virtuální počítačové sítě do souboru právě v tomto formátu. Při ukládání či načítání může vzniknout výjimka *SaveLoadException* a to z jednoho z následujících důvodů:

- Požadovaný soubor neexistuje (FILE_DOES_NOT_EXIST),
- nelze číst ze souboru (CANT_READ_FROM_FILE),
- nelze zapisovat do souboru (CANT_WRITE_TO_FILE),
- při čtení ze souboru nastala chyba (ERROR_WHILE_READING),
- při zápisu do souboru nastala chyba (ERROR_WHILE_WRITING),
- při vytváření souboru nastala chyba (ERROR_WHILE_CREATING),
- soubor není možné přepsat (CANNOT_OVERWRITE).

Jelikož je ukládání a načítání sítě voláno z uživatelského rozhraní, tak i výjimka se vrací do uživatelského rozhraní, kde je zpracována a uživatel je informován.

5.2.4 Shared.SimulatorEvents.SerializedComponents

V balíčku SerializedComponents se nachází třída *SimulatorEvent* reprezentující událost, kterou posílá síťový simulátor grafickému rozhraní. Událost má následující parametry: ID zdrojové a cílové komponenty, ID kabelu, časovou značku, typ paketu (výčtový typ *PacketType*) a informaci, zda byl paket úspěšně doručen. Třída *SimulatorEvent* obsahuje pouze nutné údaje, potřebné k rekonstrukci celé události. Podle uložených ID můžeme dohledat reference na komponenty či kabely, nebo jejich jména. Třída *SimulatorEventsWrapper* slouží k zapouzdření množiny událostí za účelem jejich uložení do souboru.

5.2.5 Shared.SimulatorEvents.Serializer

V tomto balíčku jsou třídy zabývající se uložením událostí (*SimulatorEventsWrapper*) do souboru ve formátu XML. Výjimky, které mohou při manipulaci se soubory nastat, jsou stejně jako při ukládání konfigurace sítě propagovány do uživatelského rozhraní.

5.2.6 DataLayer

Balík DataLayer spolu s projektem Shared tvoří část Model architektonického vzoru MVC. Přímo v balíku DataLayer se nachází abstraktní třída *DataLayerFacade*, od které dědí třída *DataLayer*. *DataLayer* využívá návrhového vzoru Facade (4.3.3) a je přístupovým bodem k celému modelu. Je přes ní možné nastavovat vlastnosti programu (jazyk, velikost ikon), přidávat observery k jednotlivým managerům (*LanguageManager*, *PreferencesManager*, *SimulatorManager*), či ukládat konfiguraci sítě nebo zachycené události do souboru. Další věcí, kterou *DataLayer* podporuje, je přístup k datovému modelu virtuální počítačové sítě s využitím *NetworkFacade* (5.2.8).

5.2.7 DataLayer.Interfaces

V balíčku Interfaces jsou dvě rozhraní, která implementuje abstraktní třída *DataLayerFacade*: *PreferencesInterface* a *LanguageInterface*. Rozdělení rozhraní podle funkcionality do menších celků odpovídá principu ISP[13]. Další třídou je *SaveableInterface*, kterou využívají třídy, které potřebují ukládat nastavení programu do systémových proměnných či registrů.

5.2.8 DataLayer.Network

Pro přístup k modelu virtuální počítačové sítě slouží třída *NetworkFacade*, jejíž název napovídá, že využívá návrhového vzoru Facade (4.3.3). Tato třída umožňuje přidávat či ubírat nové komponenty nebo kabely a nastavovat vlastnosti sítě. K vytváření nových součástí sítě slouží třída *NetworkComponentsFactory*, která využívá návrhového vzoru Factory (4.3.4) a jako jediná může vytvářet instance tříd *CableModel*, *HwComponentModel*, *EthInterfaceModel*, *NetworkModel* nebo *NetworkCounterModel*.

5.2.9 DataLayer.SimulatorEvents

V tomto balíčku se nachází pouze jedna třída, a to *SimulatorEventWithDetails*. Jak už její název napovídá, tato třída zapouzdřuje třídu *SimulatorEvent* a přidává k ní další informace (detaily). Tuto třídu vytvoříme po přijetí události od serveru a vyhledání potřebných údajů. Například podle ID virtuálních komponent dohledáme s pomocí *NetworkFacade* a metody *getHwComponentModelById(int id)* reference na komponenty nebo s pomocí dalších metod zjistíme jména komponent, případně přiřadíme události určitou barvu podle typu paketu. Barvy paketů využijeme při jejich vizualizaci pro lepší rozlišení jednotlivých typů a jsou přiřazeny následovně:

- TCP - zelená

[13]ISP (Interface Segregation Principle) - příliš velké rozhraní by mělo být rozděleno do menších a specifičtějších rozhraní podle funkcionality.

- UDP - modrá
- ICMP - šedá
- ARP - žlutá
- Generic - růžová
- Ethernet - černá
- IP - černá

5.2.10 DataLayer.Simulator

V balíčku Simulator se nachází třídy zodpovědné za uchovávání zaznamenaných událostí a za nastavení přehrávání, nahrávání nebo realtime režimu. Třída *EventTableModel* dědí od knihovní třídy Javy *AbstractTableModel* a slouží jako model pro tabulku událostí a zároveň tato třída udržuje aktuální pozici v tabulce a umožňuje pohyb v ní. Protože k této třídě přistupuje více vláken najednou, jsou metody synchronizované tak, aby nedošlo k anomáliím. Dále uvádím příklad synchronizované metody pro získání události (*SimulatorEventWithDetails*) z aktuální pozice:

```
public SimulatorEventWithDetails getSimulatorEvent(int i) {
    synchronized (lock) {
        if(eventList.size()>i){
            return eventList.get(i);
        }else{
            return null;
        }
    }
}
```

Další třídou je *SimulatorManager*, která implementuje rozhraní *SimulatorManagerInterface*. Tato třída uchovává informace o tom, zda je program právě připojen k serveru, jaká je nastavena rychlost přehrávání, zda právě přehrává, nahrává, či je v realtime režimu. Dále tato třída slouží jako ovládací prvek simulátoru a díky využití návrhového vzoru Observer (4.3.5) dokáže informovat registrované observery o změnách.

K třídě *SimulatorManager* přistupuje jak uživatel s pomocí ovládacího panelu (*ControllPanel*), tak vlákno přehrávače a nahrávací vlákno. Problémem je, že veškerá práce s grafickým rozhraním musí probíhat v EDT vlákně, tudíž například při potřebě informovat ovládací panel o úspěšném připojení k serveru musíme zajistit, aby toto informování proběhlo právě v EDT vlákně. Jinak řečeno, všechny metody ve třídě *SimulatorManager*, které je možné volat z jiného než EDT vlákna a které zároveň informují s pomocí vzoru Observer (4.3.5) jiné třídy, je nutné upravit tak, aby ono informování proběhlo

v EDT vlákně. K tomuto účelu slouží metoda *SwingUtilities.invokeLater*, která umožní naplánovat v EDT vlákně operaci, která bude provedena tak rychle, jak jen to bude možné. Následuje příklad pro metodu, která informuje o neúspěšném připojení k serveru.

```
@Override
public void connectingFailed() {
    isConnectedToServer = false;
    SwingUtilities.invokeLater(new Runnable() {

        @Override
        public void run() {
            // notify all observers
            setChanged();
            notifyObservers(ObserverUpdateEventType.
                                CONNECTION_CONNECTING_FAILED);
        }
    });
}
```

Poslední třídou obsaženou v balíčku Simulator je výjimka *ParseSimulatorEventException*, kterou vytvoříme, pokud se přijatou událost nepodaří „napasovat“ na právě otevřenou síť, což se stane například pokud se nepodaří nalézt v otevřené virtuální síti zařízení s danými identifikátory.

5.2.11 DataLayer.Singletons

Jak název balíčku napovídá, nachází se v něm třídy, které respektují návrhový vzor Singleton (4.3.6). První třídou je *GeneratorSingleton*, která je zodpovědná za generování názvů, identifikátorů a MAC adres jednotlivých komponent. Druhou třídou je *TimerKeeperSingleton*, která udržuje referenci na tzv. *TimingSource*, což je objekt, který umožňuje časování animací a který by měl být v systému vytvořen pouze jednou.

Poslední třídou je *ZoomManagerSingleton*. Tato třída udržuje aktuální hodnotu přiblížení, umožňuje přepočet mezi aktuálním zoomem a tzv. default zoomem[14]. Třída *ZoomManagerSingleton* využívá návrhového vzoru Observer (4.3.5) a při změně úrovně přiblížení informuje všechny zaregistrované objekty o změně.

5.2.12 DataLayer.Singletons.ImageFactory

Třída *ImageFactorySingleton* je příkladem další třídy, která využívá návrhového vzoru Singleton (4.3.6). Tato třída slouží k získání obrázků daných roz-

[14]Default zoom - referenční zoom, ve kterém jsou prováděny operace s rozmístěním sítě a ve kterém je uložena konfigurace sítě.

měrů pro nástrojová menu, animace nebo komponenty. Načítání ze souboru zajišťuje třída *BufferedImageLoader*, která využívá vyrovnávací paměti tak, aby stačilo konkrétní obrázek z pevného disku načíst pouze jednou. Samotná třída *ImageFactorySingleton* využívá také vyrovnávací paměti (*ImageBuffer*) a to pro všechny požadované rozměry daného obrázku. Využití vyrovnávacích pamětí zajistí možnost plynule měnit přiblížení a rychle vykreslovat jak virtuální síť, tak animace paketů.

5.2.13 DataLayer.Preferences

Třída *PreferencesManager* uchovává veškerá nastavení programu, jako zobrazované detaily, velikost ikon, naposledy otevřené soubory či adresu a port serveru. Nastavení se ukládá s využitím Java Preferences API[26], které umožňuje ukládat uživatelská nastavení bez nutnosti využití souborového systému. Například v operačním systému Windows Preferences API ukládá data přímo do registrů.

5.2.14 DataLayer.Language

V balíčku *Language* jsou umístěny třídy *LanguageManager* a *LanguageLoader*. Třída *LanguageLoader* slouží k načítání jazykových souborů ze souborového systému z adresáře Languages, do kterého je možné umístit libovolné množství překladů programu. Překlady využívají tzv. properties, což je způsob uložení dvojic typu klíč=hodnota. Přímo ve výsledném *.jar* souboru je přibalen český a anglický jazykový balíček. Další překlady je možné umístit právě do adresáře Languages. Následuje příklad části jazykového souboru:

```
BUNDLE_LANGUAGE_NAME=Čeština
WINDOW_TITLE=PSImulator
FILE=Soubor
EXIT=Konec
NEW_PROJECT=Nový Projekt...
CLOSE=Zavřít
OPEN=Otevřít...
RECENTLY_OPENED_FILES=Naposledy otevřené
```

Třída *LanguageLoader* přiložené jazyky zkontroluje vůči referenčnímu (angličtina) a pokud jazyk obsahuje všechny klíče, je přidán v programu do volby jazyka. Program kontroluje přítomnost jazykových souborů při každém spuštění.

Třída *LanguageManager* je zodpovědná za změnu jazyka a s využitím návrhového vzoru Observer (4.3.5) i za informování zaregistrovaných objektů o jeho změně. Dále třída uchovává informace o všech dostupných jazykových balíčcích.

5.3 Controller

Vzhledem k důvodům popsaným v návrhu architektury aplikace (4.2) je logika související s interakcí uživatele soustředěna spolu s uživatelským rozhraním v části UserInterface (z pohledu MVC se jedná o View). Z tohoto důvodu část aplikace jménem Controller (v kódu nazývaná LogicLayer) obsahuje v podstatě jen logiku zajišťující spojení se serverem a logiku zachytávání a přehrávání událostí.

5.3.1 LogicLayer

V této části aplikace je pouze jedna třída, a to *Controller*. Tato třída je odpovědná za vytvoření vláken *SimulatorClientEventRecieverThread* a *SimulatorPlayerThread*, která jsou popsána níže. Další úlohou třídy *Controller* je přidání obou vláken jako příjemců událostí z modelu, konkrétně ze třídy *SimulatorManager*.

5.3.2 LogicLayer.Simulator

V balíčku Simulator se nejdříve budeme zabývat třídou *SimulatorPlayerThread*. Instancí této třídy je vlákno, které je schopné přehrávat události a které ovládá *SimulatorManager* s pomocí vzoru Observer (4.3.5) a metody *update()*. Vlákno běží v nekonečné smyčce a podle nastavených příznaků vykonává funkce. Příznaky se nastavují právě v metodě *update()*, ve které se na vlákně v případě potřeby rychlé reakce volá metoda *interrupt()*, která vyvolá okamžité přerušení práce vlákna.

Pokud má vlákno nastavený příznak *isPlaying* nebo *isRealtime*, pak se stará o přehrávání událostí. Pokud žádný příznak nastaven není, pak vlákno spí. Přehrávání událostí spočívá v načtení události z aktuální pozice v seznamu, výpočtu délky animace a z vytvoření animace v *AnimationPanelu*. Poté se vlákno uspí a čeká na další probuzení.

Druhou třídou v balíčku Simulator je *SimulatorClientEventRecieverThread*. Tato třída má na starosti spojení se serverem a příjem událostí a funguje na podobném principu jako třída *SimulatorPlayerThread*. Vlákno běží v nekonečné smyčce a řídí se příznaky, které se nastavují v metodě *update()*, kterou volá s pomocí návrhového vzoru Observer (4.3.5) *SimulatorManager*. Na příkaz *doConnect* se vlákno pokouší spojit se serverem, na příkaz *doDisconnect* se vlákno odpojí a na příkaz *isConnected* přijímá události. Pokud není nastaven ani jeden příznak, vlákno spí.

Třetí třídou je *SimulatorClientEventRecieverMockupThread*, která je velmi podobná třídě *SimulatorClientEventRecieverThread*, až na to, že připojení k serveru i příjem paketů simuluje. Tato třída je vhodná pro ladění programu a generování velkého množství událostí.

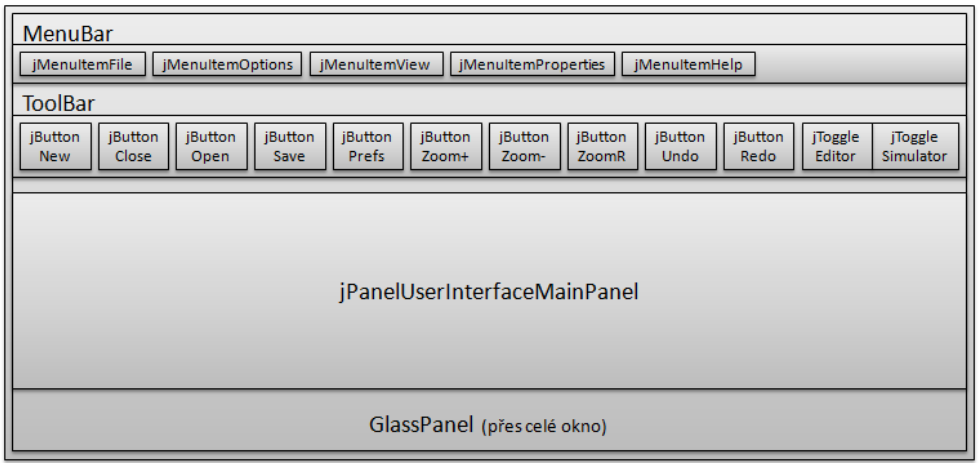
5.4 View

V části View se nachází balíčky a třídy související s grafickým uživatelským rozhraním a interakcí s uživatelem. Pro přehled a orientaci v propojení celého balíku *UserInterface* slouží diagram na obrázku (5.3). Tento diagram obsahuje pouze důležité třídy pro strukturu aplikace a pro přehlednost neobsahuje operace tříd ani jejich proměnné.

5.4.1 UserInterface

Základem uživatelského rozhraní je třída *MainWindow*, jejíž instance je prvním oknem vytvořeným po spuštění programu. Tato třída reprezentuje hlavní okno aplikace a obsahuje lištu s menu (*MenuBar*) a nástrojovou lištu (*ToolBar*). Třída *MainWindow* vytváří *UserInterfaceMainPanel*, který vyplňuje zbytek hlavního okna (obrázek 5.2) a je přístupný přes *UserInterfaceMainOuterInterface*. Třída *MainWindow* přijímá informace o změně jazyka s využitím vzoru Observer (4.3.5) a metody *update()*.

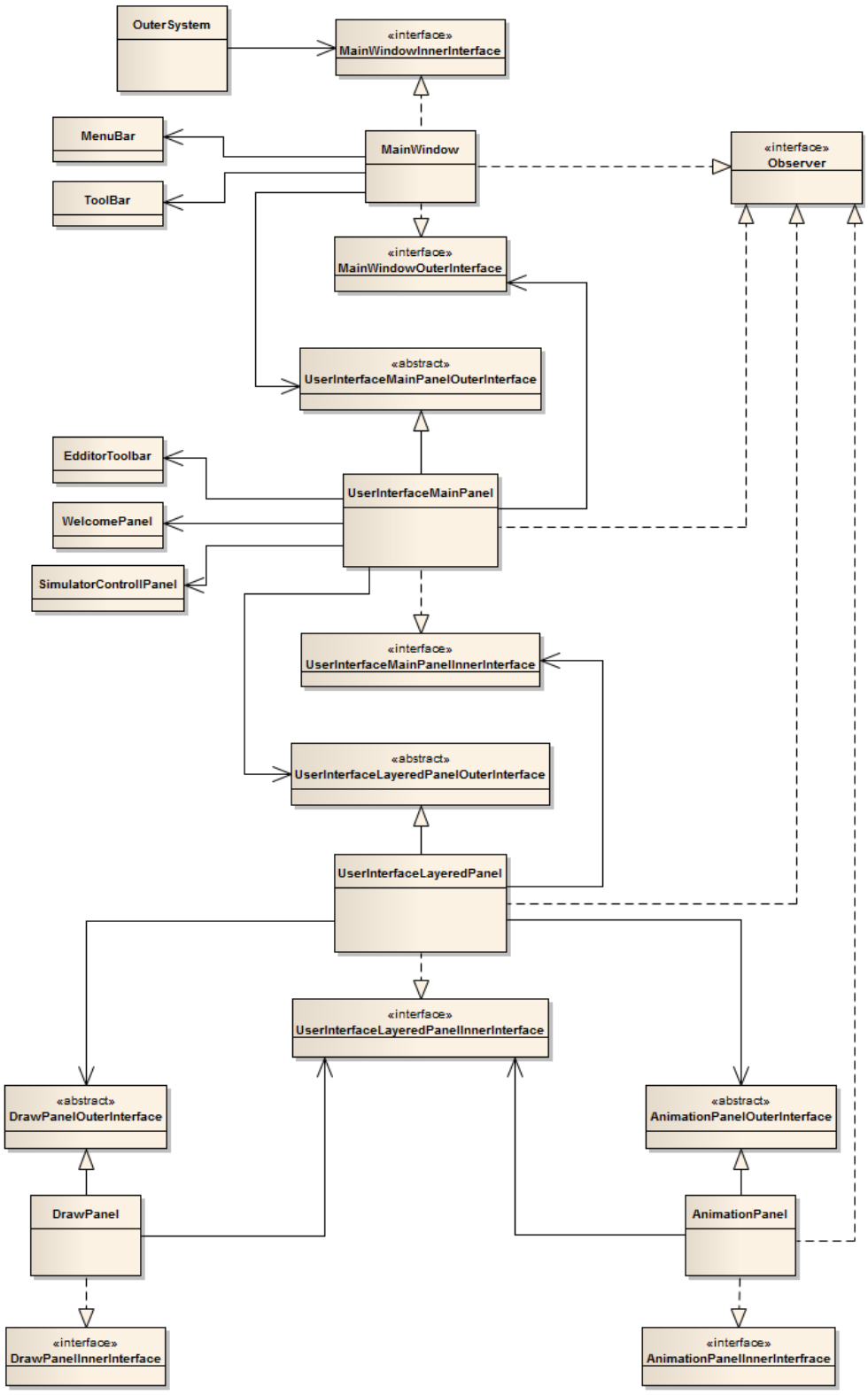
Na obrázku 5.2 jsou vidět součásti jak hlavního menu (*MenuBar*), tak nástrojové lišty (*ToolBar*).



Obrázek 5.2: Rozvržení hlavního okna aplikace

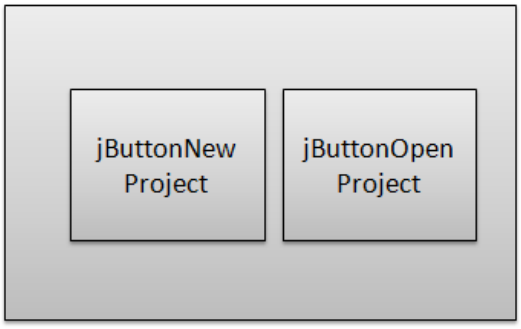
5.4.2 UserInterface.SimulatorEditor

Třída *UserInterfaceMainPanel* (dědí od *java.swing.JPanel*) zajišťuje přepínání mezi jednotlivými režimy: uvítací obrazovka, editor a simulátor. Při spuštění programu se automaticky nastaví v tomto panelu uvítací obrazovka (obrázek 5.4), tzv. *WelcomePanel*. Po vytvoření nového projektu nebo otevření existujícího se uvítací obrazovka nahradí instancí *UserInterfaceLayeredPanelu*

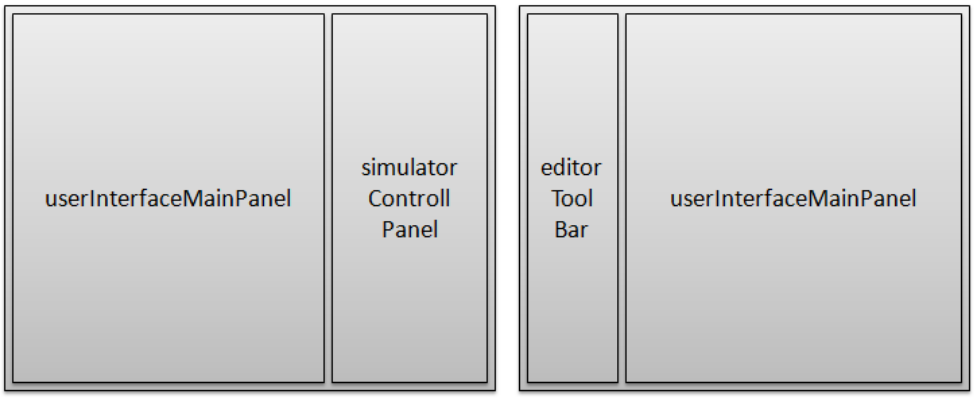


Obrázek 5.3: Diagram propojení uživatelského rozhraní

a vlevo se vytvoří panel s nástroji *EditorToolbar*. Po přepnutí do simulačního režimu se *EditorToolbar* schová a vpravo na obrazovce se vytvoří *SimulatorControllPanel*. Detailně je rozvržení obou komponent vidět na obrázku 5.5.



Obrázek 5.4: Rozvržení uvítací obrazovky



Obrázek 5.5: Rozvržení panelu *UserInterfaceMainPanel* v režimech editoru a simulátoru

Třída *UserInterfaceMainPanel* přijímá s využitím metody *update()* informaci o změně přiblížení a zajišťuje posun *UserInterfaceLayeredPanelu* v tzv. viewportu[15] tak, aby při přiblížení myší zajistil přiblížení podle aktuální polohy kurzoru. Při využití tlačítek zoomu v hlavním ovládacím panelu zajišťuje třída posun ve viewportu tak, aby provedl přiblížení podle středu (tzv. center zoom).

[15]Viewport - součást scroll panelu, která reprezentuje aktuálně viditelnou část celku.

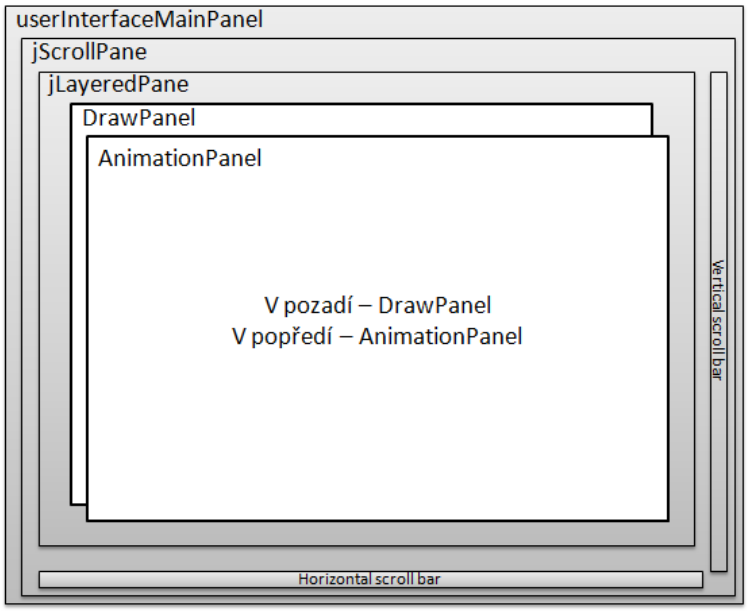
5.4.3 UserInterface.SimulatorEditor.SimulatorControlPanel

V balíčku SimulatorControlPanel je nejdůležitější stejnojmenná třída *SimulatorControlPanel*. Tato třída dědí od knihovní třídy *java.swing.JPanel* a tvoří ovládací prvek k celému simulačnímu režimu. Ovládací panel je složen z mnoha dílčích částí a jedná se o nejsložitější panel v programu. Vizualizaci složení panelu najdete na obrázku 5.7. Ovládací panel je napojen přímo na *SimulatorManagerInterface* v modelu a slouží k ovládání simulačního režimu uživatelem. Panel přijímá informace o změnách ze třídy *SimulatorManager* s pomocí vzoru Observer (4.3.5) a metody *update()* a promítá je do uživatelského rozhraní.

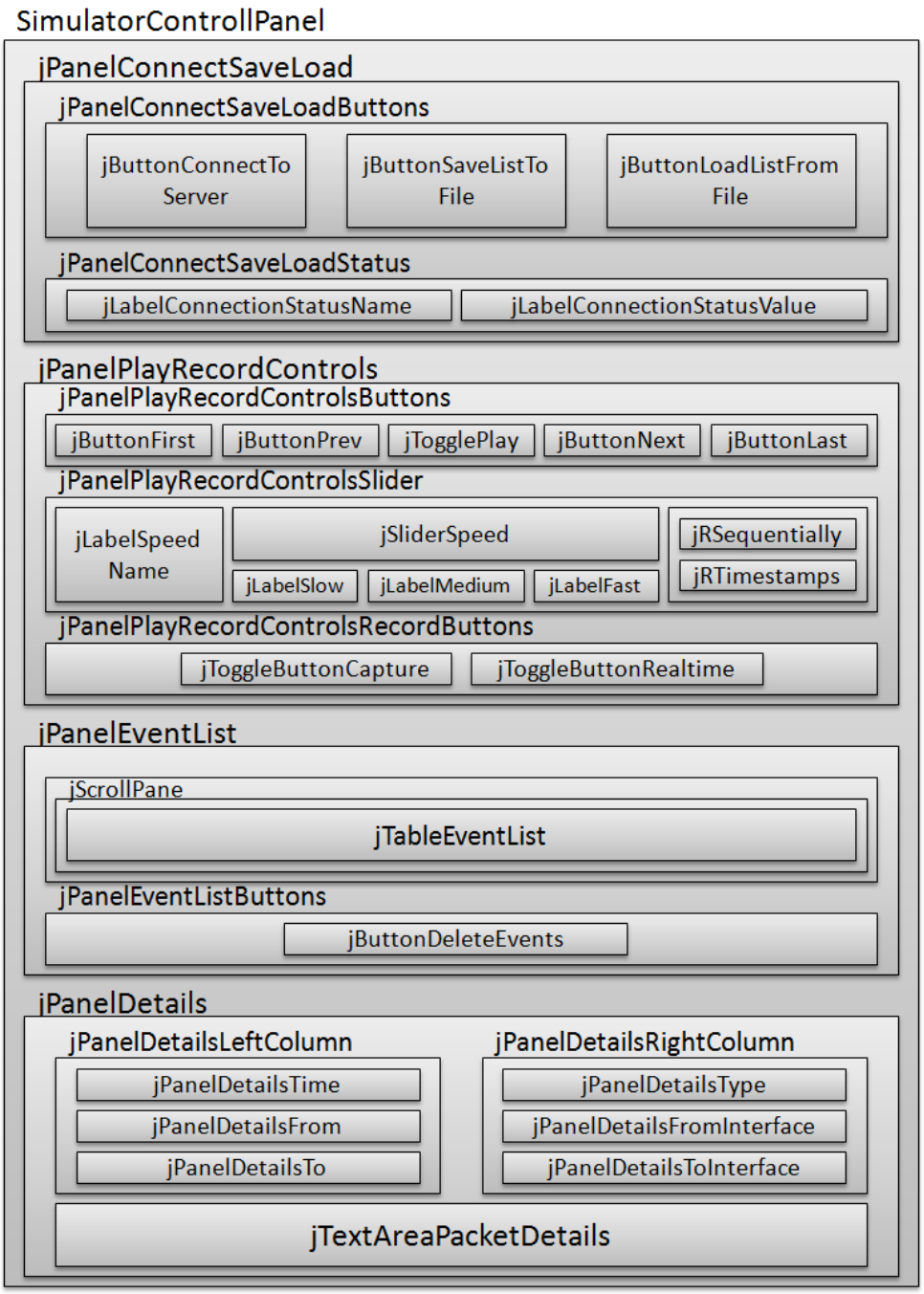
5.4.4 UI.SimulatorEditor.UserInterfaceLayeredPane

Jak již název napovídá, třída *UserInterfaceLayeredPane* ze stejnojmenného balíčku dědí od třídy *java.swing.LayeredPane* a tvoří ji více vrstev. Spodní vrstvou je *DrawPanel* neboli panel pro tvorbu virtuální sítě. Vrchní vrstvu tvoří *AnimationPanel*, ve kterém se odehrávají animace paketů v simulačním režimu.

Třída *UserInterfaceLayeredPane* je příjemcem informací o změně jazyka, přiblížení a nastavení programu. Tyto události deleguje primárně do *DrawPanelu*, ve kterém je virtuální síť vykreslována. Na obrázku se nachází schéma, jak do sebe zapadají *UserInterfaceMainPanel*, *UserInterfaceLayeredPane* a *AnimationPanel* s *DrawPanelem*.



Obrázek 5.6: Schéma panelu *UserInterfaceMainPanel*



Obrázek 5.7: Diagram složení Ovládacího panelu

5.4.5 UI.SimulatorEditor.AnimationPanel

Nejdůležitější třídou v tomto balíčku je třída *AnimationPanel*. Tato třída zajišťuje vykreslování animací s využitím TimingFrameworku. K časovači (*TimingSource*) je připojen listener, který volá metodu *paint()*, která je zodpovědná za vykreslování animací. Animace do tohoto panelu přidává vlákno *SimulatorClientEventRecieverThread*. Následuje ukázka napojení metody *paint()* jako listenera časovače.

```
postTickListener = new TimingSource.PostTickListener() {
    @Override
    public void timingSourcePostTick(TimingSource source,
                                     long nanoTime) {
        repaint();
    }
};
```

Třída *AnimationPanel* je příjemcem zpráv z modelu o zastavení či spuštění přehrávání, podle kterých se zapne či vypne překreslování panelu, které se jinak děje každých 15ms.

Další třídou v balíčku AnimationPanel je třída *Animation*. Ta reprezentuje jednu animaci a zajišťuje získání obrázku z *ImageFactory* a změnu souřadnic $[x, y]$ podle toho, kolik procent z času animace uběhlo. Tuto informaci získává po dobu trvání animace od třídy *Animator*. Po skončení animace se objekt sám odebere ze seznamu animací v *AnimationPanelu*. Následuje fragment kódu s vytvořením objektu Animator, který je zodpovědný za informování o uběhnuté době animace.

```
animator = new Animator.Builder().
     setDuration(durationInMilliseconds, TimeUnit.MILLISECONDS).
     setStartDirection(Animator.Direction.FORWARD).
     addTarget((TimingTarget)this).build();
```

5.4.6 UI.SimulatorEditor.DrawPanel

Třída *DrawPanel* má na starosti vykreslování virtuální počítačové sítě včetně vykreslování označovacího obdélníku pro označování komponent, či vykreslování aktuálně tvořeného spojení mezi komponentami. Dále má tato třída reference na tzv. Actions, které je možné na *DrawPanelu* provádět a také na *MosueListenery*, které je možné na *DrawPanelu* použít. Tyto komponenty budou popsány dále.

Při vykreslování se nejprve vykreslí aktuálně tvořený kabel (pokud nějaký je), přes něj se vykreslí celá virtuální síť (v kódu nazvaná Graph) a pokud je aktivní označovací obdélník, pak se navrch také vykreslí. *DrawPanel* drží referenci na aktuálně otevřenou počítačovou síť - instanci třídy *Graph*, která zajišťuje vykreslování virtuální počítačové sítě.

5.4.7 UI.SimulatorEditor.DrawPanel.Actions

Akce, které je možné provádět na *DrawPanelu* využívají návrhového vzoru Strategy (4.3.7). Abstraktní předek *AbstractAction* je šablonou, kterou všechny akce respektují a pouze jinak implementují metodu *actionPerformed()*. V balíčku Actions se nachází akce pro smazání komponent (*ActionOnDelete*), jejich označování (*ActionSelectAll*), zobrazení menu na komponentě (*ActionOpenProperties*), pro změnu aktuálního nástroje (*ActionSwitchTool*), pro zarovnání komponent do mřížky (*ActionAlignComponentsToGrid*) či pro automatické rozmístění sítě v ploše (*ActionAutomaticLayout*).

Uvedené akce je možné z *DrawPanelu* získat a posléze je na něm použít. Příkladem jejich využití jsou tlačítka v nástrojové liště.

5.4.8 UI.SimulatorEditor.DrawPanel.MouseActionListeners

Při přepnutí nástroje v panelu nástrojů se přepne v *DrawPanelu* aktuální *MouseListener*, což je třída, která reaguje na události vytvořené použitím myši. V balíčku MouseActionListeners se nachází třídy zodpovědné za přidávání komponent (*DrawPanelListenerStrategyAddHwComponent*), za přidávání kabelů (*DrawPanelListenerStrategyAddCable*), za manipulaci se sítí (*DrawPanelListenerStrategyHand*) a za posun celé plochy v tzv. viewportu (*DrawPanelListenerStrategyDragMove*). Nachází se zde také *MouseListener* použitý v režimu simulátoru (*DrawPanelListenerStrategySimulator*). Všechny uvedené třídy mají abstraktního předka *DrawPanelListenerStrategy* a využívají návrhového vzoru Strategy (4.3.7).

Následují dvě ukázky kódu z abstraktní třídy *DrawPanelListenerStrategy*.

V tomto fragmentu kódu je vidět reakce na události pocházející od kolečka myši. Podle směru rotace kolečka se mění úroveň přiblížení v *ZoomManagerSingletonu*.

```
@Override
public void mouseWheelMoved(MouseWheelEvent e) {
   if(e.getScrollType() == MouseWheelEvent.WHEEL_UNIT_SCROLL) {
     //System.out.println("Point:"+convertPoint(e.getPoint());
     // scroll down
     if (e.getWheelRotation() == -1) {
         ZoomManagerSingleton.getInstance().
           zoomIn(convertPoint(e.getPoint()), ZoomType.MOUSE);
         //scroll up
     } else if (e.getWheelRotation() == 1) {
         ZoomManagerSingleton.getInstance().
           zoomOut(convertPoint(e.getPoint()), ZoomType.MOUSE);
     }
   }
}
```

Zde je vidět reakce na klik myši, při kterém se nejprve zjistí, které tlačítko bylo stisknuto, a podle toho se volá metoda pro levé, či pravé tlačítko. Tyto metody jsou v potomcích třídy *DrawPanelListenerStrategy* přetížené a implementují požadovanou funkcionalitu.

```
@Override
public final void mouseClicked(MouseEvent e) {
    if (SwingUtilities.isLeftMouseButton(e)) {
        mouseClickedLeft(e);
    } else if (SwingUtilities.isRightMouseButton(e)) {
        mouseClickedRight(e);
    }
}

public abstract void mouseClickedLeft(MouseEvent e) {
}

public abstract void mouseClickedRight(MouseEvent e) {
}
```

5.4.9 UI.SimulatorEditor.DrawPanel.Components

V balíčku Components se nachází grafické prvky, které tvoří virtuální počítačovou síť. Schéma těchto prvků a jejich napojení na datový model v balíku Shared.Components je znázorněno na obrázku 5.8.

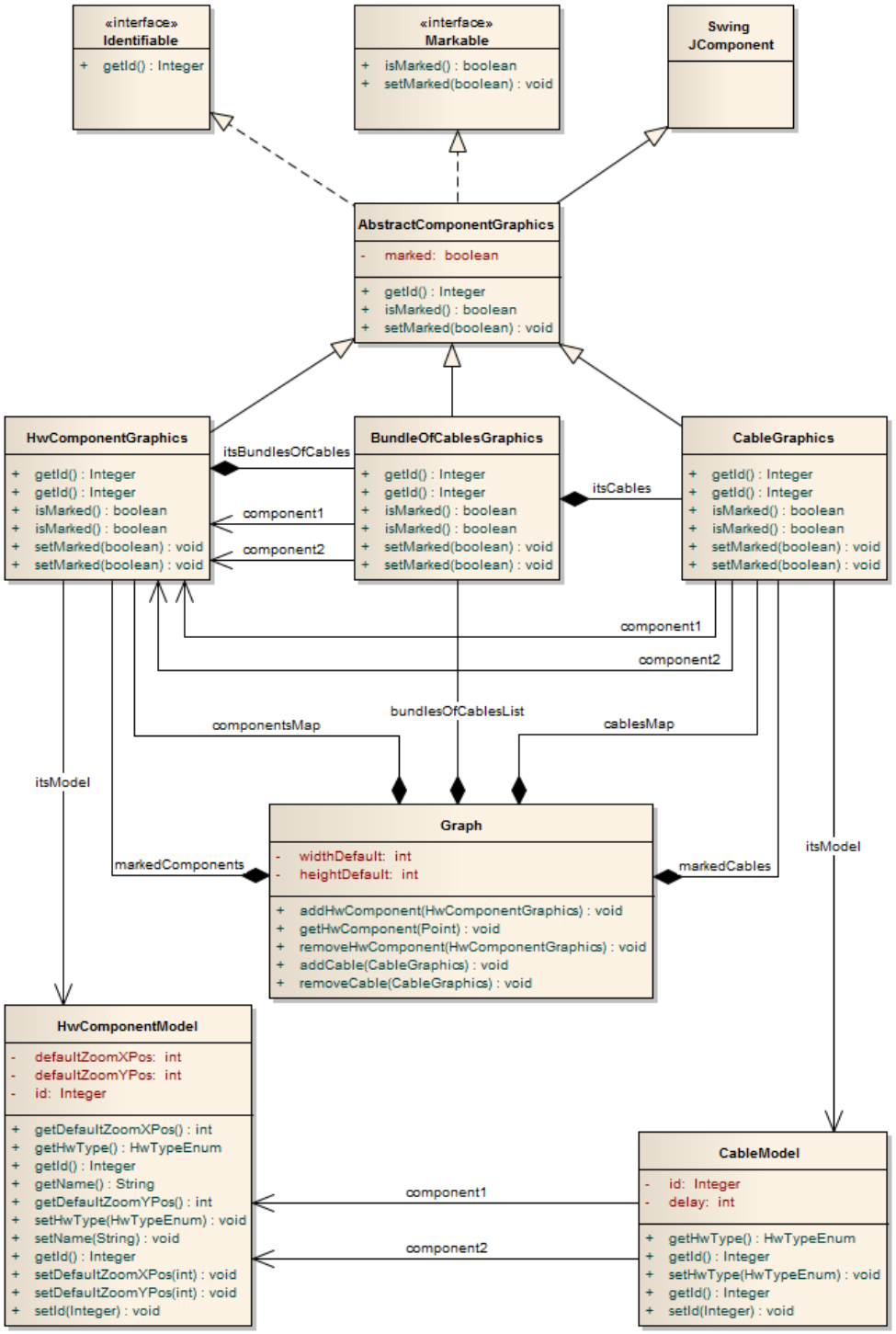
Abstraktní třída *AbstractComponentGraphics* dědí od knihovní grafické třídy *java.swing.JComponent* a implementuje rozhraní *Markable* a *Identifiable*. To znamená, že třída podporuje označování a jedinečnou identifikaci. Od této třídy dědí všechny zobrazované grafické elementy.

Prvním elementem je *HwComponentGraphics*, což je třída reprezentující virtuální zařízení. Druhým elementem je třída *CableGraphics*, která zastupuje spojení mezi dvěmi komponentami. Třetí třídou je *BundleOfCables*, což je třída reprezentující svazek kabelů mezi dvěmi komponentami. Tato třída je potřebná z důvodu vykreslování paralelních spojení tak, aby se kabely vzájemně nepřekrývaly.

Třída *HwComponentGraphics* je zodpovědná za uchovávání reference na aktuální ikonu a popisky, které jsou aktuálně zobrazovány. Podobně jako třída *CableGraphics*, která uchovává popisky pro obě rozhraní, ke kterým je kabel připojen, a také popisek o zpoždění samotného kabelu (delay). Popisky jsou umísťovány tak, aby nepřekrývaly ikonu komponenty, ani její jméno či typ.

5.4.10 UI.SimulatorEditor.DrawPanel.Graph

Třída *Graph* zastupuje grafickou část virtuální počítačové sítě. Jak je vidět na diagramu 5.8, graf obsahuje množinu komponent, označených komponent,



Obrázek 5.8: Grafický model a jeho napojení na datový model

kabelů, označených kabelů a svazků kabelů. Tyto množiny umožňují specifické vykreslení komponent celé sítě. Nejprve se vykreslí svazky kabelů. Každý svazek kabelů může obsahovat více paralelních spojů, z nichž mohou být některé označené, což je reprezentováno modrou barvou. Poté se vykreslí neoznačené komponenty a nakonec označené komponenty. Vykreslení označených prvků přes neoznačené umožní při přesunu komponent myší označené komponenty zobrazit v nejvyšší vrstvě.

Dalšími funkcemi třídy *Graph* je přidávání a odebírání komponent či kabelů, označování prvků nebo zarovnání komponent do mřížky. *Graph* také slouží jako informátor (*Observable*) pro třídu *UserInterfaceLayeredPane*, kterou informuje v případě změny rozměrů celé sítě např. v případě přidání nové komponenty, nebo posunu stávajících komponent.

5.4.11 UI.SimulatorEditor.DrawPanel.Graph.LayoutAlgorithm

Balíček LayoutAlgorithm obsahuje implementaci Genetického algoritmu popsaného v kapitole Návrh (4.4). Samotný běh algoritmu je zajištěn ve zvláštním vlákně tak, aby při jeho běhu nedošlo k „zamrznutí“ grafického uživatelského rozhraní. Uživatel je v průběhu algoritmu informován o aktuálním stavu s využitím dialogů, které jsou umístěné v balíčku DrawPanel.Dialogs.

5.4.12 UI.SimulatorEditor.DrawPanel.Graph.GraphBuilder

Při načtení modelu virtuální sítě ze souboru je potřeba vytvořit instanci výše uvedené třídy *Graph* spolu se všemi grafickými komponentami. K tomuto účelu slouží třídy v balíčku GraphBuilder, které, jak název napovídá, využívají návrhového vzoru Builder (4.3.1). K celému systému vytváření složité struktury virtuální sítě je možné přistupovat přes *GraphBuilderFacade*, která využívá vzoru Facade (4.3.3) a zjednodušuje vytváření uvedené struktury.

Návrhový vzor Builder zajišťuje stavbu složitého celku (*Graph*) s využitím malých stavebních bloků. Nejdříve je vytvořena instance třídy *Graph* a poté jsou postupně vytvářeny instance grafických objektů, které jsou přidávány do grafu. Na konci tohoto procesu je výsledný produkt vrácen klientovi.

5.4.13 UI.SimulatorEditor.DrawPanel.UndoCommands

Posledním balíčkem, kterým se budeme v podkapitole View podrobněji zabývat, je balíček UndoCommands. Třídy v tomto balíčku využívají návrhového vzoru Command (4.3.2) a umožňují pro provedené operace vytvořit tzv. „undoable objekt“. Na tomto objektu je možné podle potřeby volat operace *undo()* a *redo()*. V balíčku jsou třídy pro všechny operace, u kterých potřebujeme zachovat možnost vrátit změny zpět. Příkladem operace, kterou můžeme chtít vrátit zpět je smazání komponent. Při té se vytvoří instance třídy *UndoableDeleteComponents*, která se uloží do seznamu provedených operací.

V případě potřeby na instanci zavoláme operaci *undo()*, která zajistí opětovné přidání komponent do sítě. Pokud opět změníme názor a využijeme operace *redo()*, pak instance třídy *UndoableDeleteComponents* původně smazané komponenty znovu z grafu odstraní.

5.4.14 Další balíčky v UserInterface

V balíku UserInterface se nachází další balíčky a třídy, které jsou potřebné pro celkové fungování grafického rozhraní. Jedná se o různé dialogy s informacemi pro uživatele, výčtové typy použité ve výše uvedených třídách, či další použité grafické komponenty, jako například tabulka pro zobrazení zachycených událostí. Popis těchto balíčků a jejich obsahu není podle mého názoru pro čtenáře příliš zajímavý.

5.5 Podpůrné balíčky

V této podkapitole si rozebereme tzv. podpůrné balíčky, které neobsahují přímo kód aplikace. Jedná se o tzv. „resources“, tedy zdroje, které v programu využíváme. Příkladem jsou balíčky s obrázky (*resources.icons*), nápovědou (*resources.help*) či překlady (*resources.i18n*). Podrobněji se v této podkapitole podíváme na systém nápovědy.

5.5.1 Nápověda

Systém nápovědy je vytvořen s pomocí nástroje Java Help[27]. Tento systém staví nápovědu na konfiguračních (XML) a obsahových (HTML) souborech. Základním prvkem je definice okna nápovědy, ve které uvedeme umístění konkrétního balíku nápovědy a funkcionalitu, kterou požadujeme. Příkladem je následující fragment kódu, kde je nejdříve definováno umístění konfiguračního souboru s mapou témat nápovědy a pak jsou definovány tzv. *views*. První *view* jménem TOC definuje umístění obsahu (Table of Content) a druhé definuje umístění souborů s vyhledáváním. Pro obě *view* bude v okně nápovědy samostatná záložka.

```
<helpset version="1.0" xml:lang="en-US">
   <!-- title -->
   <title>Help</title>

   <!-- maps -->
   <maps>
     <homeID>item000</homeID>
     <mapref location="default/help_map.jhm" />
   </maps>
```

```
<!-- views -->
<view>
   <name>TOC</name>
   <label>Table Of Contents</label>
   <type>javax.help.TOCView</type>
   <data>default/help_toc.xml</data>
</view>

<view xml:lang="en-US">
  <name>Search</name>
  <label>Search</label>
  <type>javax.help.SearchView</type>
  <data engine="java.help.search.DefaultSearchEngine">
                                        JavaHelpSearch/en</data>
</view>
```

Konfigurační soubor *help_map.jhm* mapuje adresy jednotlivých témat na identifikátory, které budou použity v dalších souborech. Následuje příklad:

```
<mapID target="item100" url="topics/tutorial/1_editor/
                                          welcome_screen.html" />
<mapID target="item101" url="topics/tutorial/1_editor/
                                          project_create.html" />
```

V souboru *help_toc.xml* je vytvořen přímo obsah nápovědy, tak jak bude vidět v okně s nápovědou. V tomto souboru se využívají identifikátory zavedené v *help_map.jhm*.

```
<tocitem text="2. Simulation mode" image="chapter" >
    <tocitem text="Simulation" target="item201" image="topic" />
    <tocitem text="Playing" target="item204" image="topic" />
```

Další důležitou částí nápovědy je vyhledávání. Nástroj Java Help disponuje programem pro vytvoření vyhledávací databáze pro určené soubory. Výsledkem jsou soubory v proprietárním datovém formátu, které slouží pro fulltextové vyhledávání v nápovědě. Tyto soubory jsou vygenerované pro každý jazyk zvlášť.

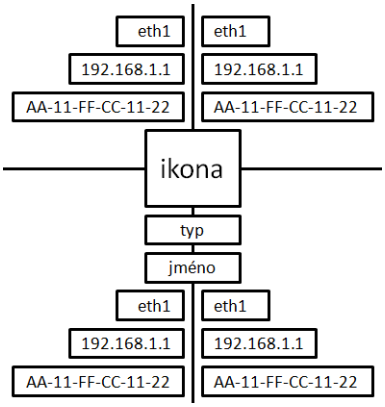
Poslední částí nápovědy jsou HTML soubory s obsahem nápovědy pro jednotlivá témata, která mohou být propojena s využitím hypertextových odkazů.

5.6 Zajímavé části implementace

V této části kapitoly se zaměříme na zajímavé aspekty vývoje a problémy, které bylo nutné řešit.

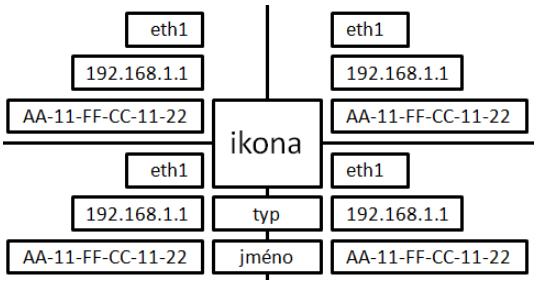
5.6.1 Umístění popisků komponent a kabelů

Nejprve se podíváme na umístění popisků komponent a kabelů. V případě zapnutí všech popisků je ve virtuální síti potřeba zobrazit: jméno prvku, typ prvku, jméno rozhraní, IP adresu, MAC adresu a zpoždění kabelu. Požadavkem bylo, aby popisky týkající se rozhraní byly u připojeného kabelu tak, aby nepřekrývaly popisky ani ikonu hardwarového prvku. Na obrázku 5.9 můžete vidět, jak jsou popisky umístěny v případě přechodu kabelu přes pomyslnou osu Y. V první a čtvrtém kvadrantu jsou popisky zarovnány vlevo a ve druhém a třetím jsou zarovnány na pravou stranu.



Obrázek 5.9: Umístění popisků v síti - přechod přes osu Y

Na obrázku 5.10 je vidět přechod kabelu přes pomyslnou osu X, kdy se popisek přesune nad kabel, resp. pod kabel.



Obrázek 5.10: Umístění popisků v síti - přechod přes osu X

Popisky kabelu v případě potřeby „obtékají“ popisky komponenty tak, aby se navzájem nepřekrývaly. Další věcí, která usnadňuje čitelnost popisků, je podbarvení textu bílou barvou, což je výhodné v případě, že popisek překrývá kabel či jinou komponentu.

5.6.2 Změna velikosti grafu

Zajímavým problémem, který jsem musel během vývoje řešit, byla velikost kreslící plochy, resp. velikost zobrazené virtuální počítačové sítě. Záměrem bylo neomezovat předem uživatele danou velikostí plochy, do které by se musel „vejít“ a ponechat mu volnost při tvorbě. Z těchto důvodů jsem vytvořil kreslící plochu, která se dynamicky mění podle velikosti virtuální sítě v ní.

Algoritmus, který zajišťuje změnu velikosti kreslící plochy, funguje tak, že při potřebě zvětšení (např. při přidání nové komponenty) se plocha zvětší, ale při potřebě zmenšení (odebrání komponenty) se nezmenší. Důvodem je nežádoucí efekt posunu celé kreslící plochy v tzv. viewportu a tím pádem zmatení uživatele. Tímto způsobem lze vytvářet téměř neomezeně veliké virtuální sítě.

5.6.3 Změna viewportu při zoomu

Dalším problémem, který bylo potřeba vyřešit je tzv. zoom podle polohy myši, který známe například z online mapových aplikací. Tento zoom funguje tak, že při přiblížení, či oddálení nechá bod pod kurzorem myši na tom samém místě.

Při změně úrovně přiblížení se změní i velikost kreslící plochy a tudíž se ve viewportu posune na nežádoucí pozici. Proto je potřeba tuto pozici korigovat. Korekce probíhá tak, že se spočítá vzdálenost kurzoru myši od souřadnic levého horního rohu viewportu. Těmto hodnotám budeme říkat *width* a *height*:

```
int width = mouseXOldZoom - oldPosition.x;
int height = mouseYOldZoom - oldPosition.y;
```

Poté vypočítáme polohu kurzoru myši v nové úrovni přiblížení:

```
int mouseXNewZoom = (int) ((mouseXOldZoom / zew.getOldScale())
                           * zoomEventWrapper.getNewScale());
int mouseYNewZoom = (int) ((mouseYOldZoom / zew.getOldScale())
                           * zoomEventWrapper.getNewScale());
```

A nakonec spočítáme polohu viewportu po přiblížení:

```
newPosition.x = mouseXNewZoom - width;
newPosition.y = mouseYNewZoom - height;
```

Výsledkem je, že poloha viewportu se změní tak, že to, co bylo na pozici kurzoru myši, tam také zůstalo.

5.6.4 Genetický algoritmus

Původním záměrem bylo umožnit automatické rozmístění počítačové sítě v ploše v případě načtení konfiguračního souboru bez [x,y] souřadnic. V průběhu vývoje se ukázalo, že takováto funkce nebude potřeba, jelikož jediný způsob, jak vytvořit konfigurační soubor, je s pomocí editoru. Funkci automatického rozmístění jsem nicméně v programu ponechal jako ukázku implementace Genetického algoritmu na reálné úloze.

Parametry genetického algoritmu jsem s pomocí experimentálního vyhodnocení nastavil následujícím způsobem: velikost populace 30, ukončující podmínka algoritmu 1500 iterací, během kterých nedojde ke zlepšení, pravděpodobnost křížení 85% a pravděpodobnost mutace 10%.

5.6.5 Informování uživatele - Glass Pane

Poslední věcí o které se v této části kapitoly zmíním, je způsob informování uživatele o úspěšných akcích. Pokud se například nepovede načtení ze konfigurace sítě souboru, je uživatel informován o této chybě tzv. „vyskakovacím“ dialogem. Ovšem pokud se otevírání povede, je zbytečné uživateli předkládat rovnou okno s informací o úspěchu, které bude muset zavřít. Podobný problém nastává při ukládání konfigurace, kdy při neúspěchu je uživatel informován, ale při úspěchu je zbytečné uživatele zatěžovat. Zároveň by bylo vhodné uživatele informovat o tom, že se ukládání povedlo, aby nežil v nejistotě, zda tomu tak opravdu je.

Řešením této situace je tzv. Glass Pane, což je panel, který překrývá celé okno aplikace, ale je průhledný, tudíž není vidět. Při informování uživatele se v tomto panelu objeví na krátkou chvíli (3s) informace o provedené akci, která po chvíli sama zmizí. Uživatel tedy není obtěžován oknem, které by musel zavírat, a zároveň je informován.

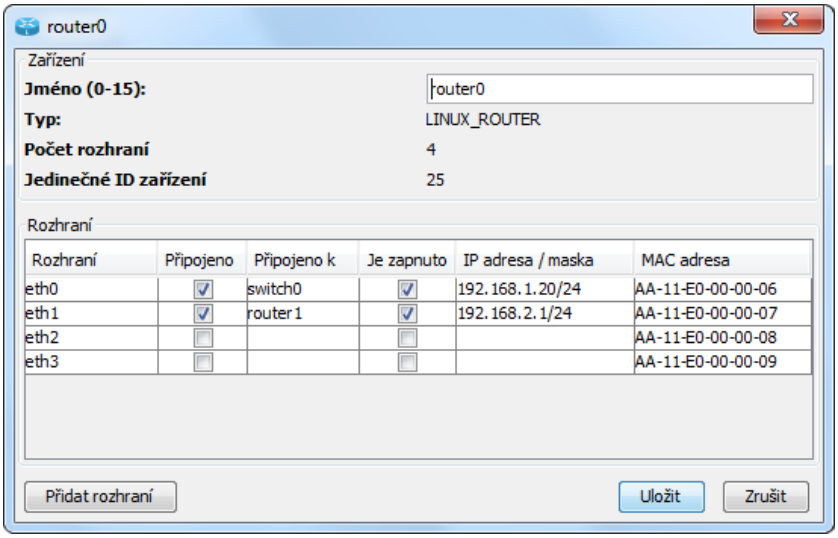
5.7 Výsledná podoba grafického uživatelského rozhraní

5.7.1 Změny oproti návrhu

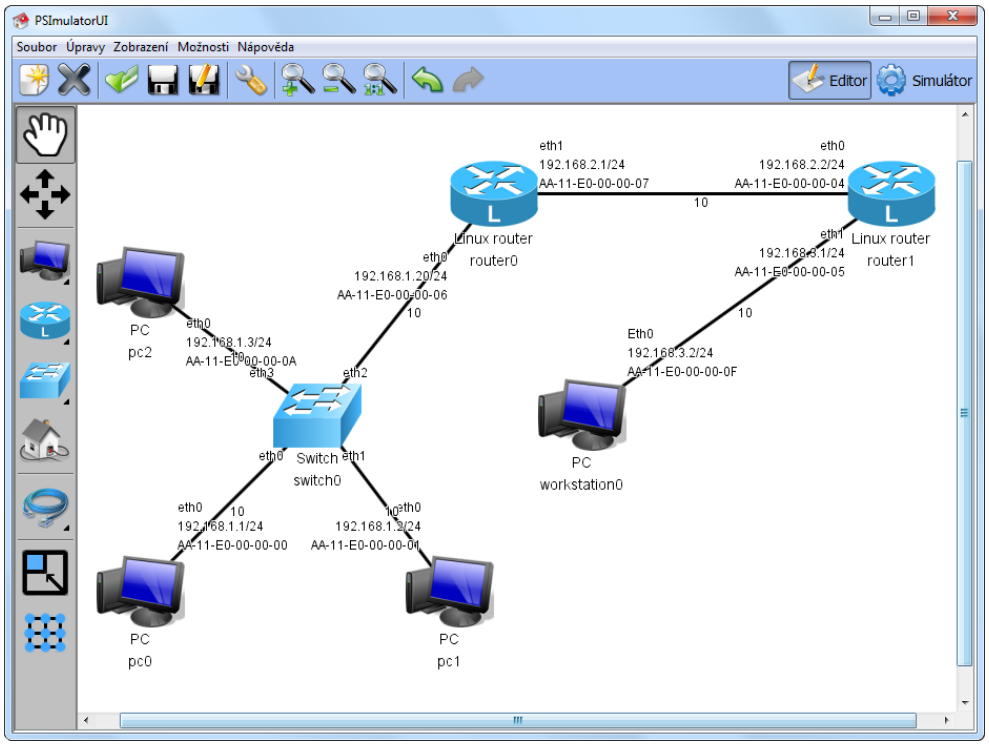
Při testování navrženého mockupu experty (viz minulá kapitola 4.1.2.1) bylo odhaleno několik nedostatků, které jsem napravil. Klávesové zkratky byly do aplikace doplněny, v podmenu s výběrem konkrétního nástroje byla vylepšena informace o počtu rozhraní a okno s vlastnostmi komponenty bylo přepracováno. Upravený dialog s detaily komponenty se nachází na obrázku 5.11.

5.7.2 Ukázka z aplikace

Na závěr kapitoly Realizace uvádím obrázek s výslednou podobou grafického uživatelského rozhraní aplikace (5.12). Více obrázků se nachází v příloze C.



Obrázek 5.11: Dialog s vlastnostmi komponenty



Obrázek 5.12: Okno aplikace v editačním režimu

KAPITOLA 6

Testování

Důležitou fází vývoje software je testování. Aplikaci je možné testovat z mnoha hledisek. Pro tuto aplikaci je nejdůležitější testování grafického uživatelského rozhraní aplikace, protože právě to určuje, jak snadno se bude s aplikací uživatelům pracovat.

6.1 Testování experty

Testování experty bylo popsáno v kapitole Návrh (4.1.2.1). Po vytvoření programu v předchozí kapitole (kterému můžeme říkat Hi-Fi prototyp[16]) je potřeba výsledné uživatelské rozhraní znovu otestovat. Nejdříve program otestovali odborníci, kteří vědí, jak má vypadat správné uživatelské rozhraní (Bc. Martin Vehovský a Ing. Zdeněk Klíma) a odborníci, kteří vědí, jak by měl vypadat výsledný produkt (vedoucí práce Ing. Pavel Kubalík PhD. a Bc. Stanislav Řehák). Následuje výčet problémů, které se při testování experty podařilo odhalit.

6.1.1 Nalezené problémy

Ex 1: Popisky v síti nejsou dobře čitelné v případě, kdy překrývají nějaký kabel.

Ex 2: MAC adresu je možné vkládat pouze ve formátu HH-HH-HH-HH-HH-HH. Bylo by vhodné umožnit také použití dvojteček jako oddělovače.

Ex 3: Obrázky pro směrovače Linux a Cisco jsou stejné a zařízení se špatně rozlišují.

Ex 4: Nelze změnit velikost okna s nastavením routeru.

[16] Hi-Fi prototyp - téměř hotový program, který je potřeba otestovat a doladit.

Ex 5: Zoom s pomocí kolečka myši reaguje obráceně.

Ex 6: Není možné otevřít více oken telnetu k jednomu prvku najednou.

Ex 7: Obrázky paketů v animacích jsou příliš velké vzhledem k obrázkům zařízení.

Ex 8: Smazání událostí zastaví zachytávání.

Ex 9: Při otevření projektu se změní úroveň přiblížení na výchozí. Program si nepamatuje nastavení přiblížení.

Ex 10: Jména směrovačů mají překlep: *swtich* místo *switch*.

Ex 11: Nejsou přednastaveny výchozí hodnoty pro připojení k serveru.

Ex 12: Při změně velikosti ikon se nemění ikony v panelu s nástroji a v ovládacím panelu.

Ex 13: Jméno zařízení umožňuje nulovou délku.

Ex 14: Nesmyslná chybová hláška při otevření seznamu událostí: „Všechny události budou smazány!“ s volbami „ANO“, „NE“.

Ex 15: Chybí funkce Naposledy otevřené (historie otevřených souborů).

Ex 16: Viditelné vizuální ohraničení sítě.

Ex 17: Chybí možnost přidat rozhraní u komponenty.

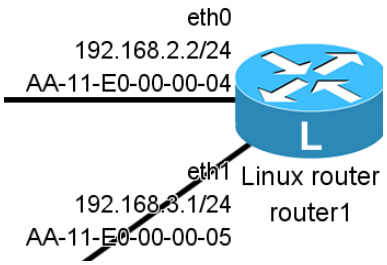
Ex 18: Automatické zarovnání do mřížky zarovnává až úplně k levému okraji a nejsou vidět celé popisky komponenty.

6.1.2 Řešení nalezených problémů

Nalezené problémy jsem analyzoval a pokusil se k nim nalézt odpovídající řešení. Následuje komentář k jednotlivým bodům.

Ex 1

Čitelnost popisků v případě, že překrývá kabel, jsem vyřešil podbarvením textu v popisku. Nápis má bílé okraje, které nejsou průhledné a i v případě přechodu přes kabel není čitelnost ovlivněna. Výsledek je vidět na obrázku 6.1.



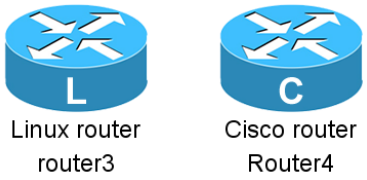
Obrázek 6.1: Podbarvení popisků při přechodu přes kabel

Ex 2

Požadavek na oba formáty MAC adres jsem zapracoval a nyní je možné vkládat adresy v obou požadovaných tvarech. Při použití oddělovače dvojtečka se během uložení adresy automaticky nahradí dvojtečky pomlčkami. Tím docílíme stejného formátu adres u všech rozhraní a zachování shodného vzhledu při vizualizaci popisků.

Ex 3

V Ex 3 je uveden požadavek na rozdílné ikony směrovačů Cisco a Linux. Různé ikony pro oba typy zařízení umožní snadné rozlišení ve vizualizaci. Upravené ikony jsou uvedeny na obrázku 6.2.



Obrázek 6.2: Rozdílné ikony pro směrovače Cisco a Linux

Ex 4

Při testování jsme objevili chybu, kvůli níž nebylo možné měnit velikost dialogu s nastavením vlastností komponenty. Tuto chybu jsem odstranil a nastavil pouze minimální velikost tohoto dialogu. Uživatel nyní může dialog libovolně zvětšovat.

Ex 5

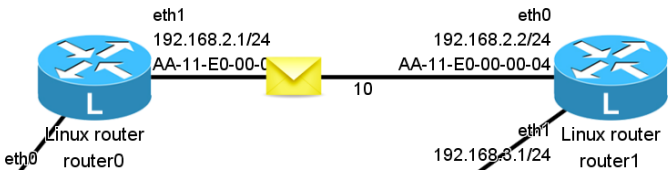
Další nedostatek se týká způsobu ovládání funkce zoom. Běžné online mapové aplikace reagují na pohyb kolečka nahoru přiblížením a dolů oddálením. V aplikaci byla tato funkce implementována přesně obráceně a tento nedostatek jsem napravil.

Ex 6

V aplikaci jsem původně implementoval funkci, kdy při otevření okna telnetu (pravým tlačítkem na komponentě) se okno zaregistrovalo do seznamu otevřených a při pokusu o opětovné otevření se toto okno přesunulo do popředí. Díky této funkci nemusel uživatel hledat okno na obrazovce, pokud jich měl otevřených víc. Problémem bylo, že uživatel mohl otevřít pouze jedno okno telnetu u jedné komponenty. Tento problém jsem vyřešil odstraněním podmínky existence pouze jednoho okna a nyní se při požadavku na připojení s využitím telnetu vytváří vždy nové okno. Uživatelé zvyklí na práci v konzolovém režimu mohou nyní využít více oken u jedné komponenty.

Ex 7

Ex 7 hovoří o příliš velkých obrázcích paketů v animacích. Jejich velikost byla původně nastavena stejně jako velikost komponent, což se ukázalo jako nevhodné. Velikost obrázků paketů jsem nastavil na 60% velikosti ikon jednotlivých zařízení. Výsledek úpravy je na obrázku 6.3.



Obrázek 6.3: Upravená velikost obrázků paketů

Ex 8

V případě spuštěného zachytávání událostí a smazání tabulky s nimi se zachytávání zastavilo. Tuto chybu jsem napravil a nyní je možné libovolně mazat obsah tabulky zachycených událostí, aniž by se zastavilo spuštěné zachytávání.

Ex 9

Během testování uživatelům vadilo, že při otevření konfiguračního souboru s uloženou sítí se vždy nastavila výchozí úroveň přiblížení. Tento problém

jsem vyřešil zachováním úrovně přiblížení z naposledy otevřené sítě. Uživatel tedy po otevření jiného souboru nemusí tzv. „odzoomovat“. Úroveň přiblížení se s využitím *preferences* (5.2.13) ukládá i pro příští spuštění programu.

Ex 10

Překlep v názvu prvku přepínače jsem opravil a již se v programu nevyskytuje.

Ex 11

Problém Ex 11 hovoří o chybějících výchozích hodnotách pro port a IP adresu serveru. Ve většině případů očekáváme, že uživatel bude používat serverovou i klientskou část na jednom počítači a při prvním spuštění aplikace bylo nutné nastavit adresu (127.0.0.1) a port (1200) ručně. Nyní jsou tyto hodnoty přednastaveny jako výchozí při prvním spuštění programu.

Ex 12

Dalším odhaleným problémem je změna velikosti ikon v programu. V nastavení je sice možné nastavit velikost ikon v hlavní nástrojové liště, ale v panelu s nástroji, či v ovládacím panelu tato možnost není. Tento problém jsem vyřešil přepracováním těchto dvou panelů, které nyní také reagují na nastavení velikosti ikon. Uživatel si díky tomu může nastavit ikony ovládacích prvků podle svých preferencí.

Ex 13

Dialog s nastavením vlastností virtuálního zařízení umožňoval nastavit prázdné jméno prvku. To způsobilo chybějící informaci v názvu telnet okna a uživatel se v otevřených oknech ztrácel, protože je nemohl v systémové liště identifikovat. Tento problém jsem vyřešil vynucením nenulové délky jména prvku.

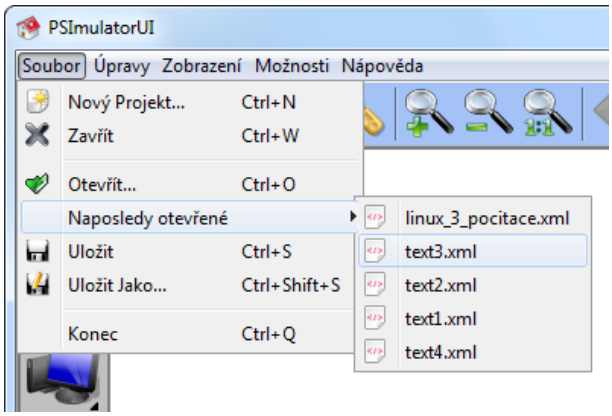
Ex 14

Při pokusu o načtení seznamu událostí objevili uživatelé nesmyslnou chybovou hlášku o nutnosti smazat zachycené události v tabulce. Hláška vypadala takto: „Všechny události budou smazány!“ s volbami „ANO“, „NE“. Tuto nesmyslnou hlášku jsem nahradil novou: „Všechny události v tabulce budou ztraceny! Přejete si pokračovat?“ s volbami „ANO“, „NE“.

Ex 15

Další věcí, na kterou uživatelé během testování upozornili, je chybějící funkce Naposledy otevřené, která by umožnila snadný přístup k naposledy otevřeným souborům. Je pravděpodobné, že uživatel bude v programu vícekrát otevírat stejné konfigurační soubory a tato funkce mu usnadní používání aplikace. Tuto

funkci jsem do programu implementoval a je dostupné v menu Soubor pod položkou Naposledy otevřené. Detail se nachází na obrázku 6.4.



Obrázek 6.4: Funkce Naposledy otevřené v menu Soubor

Ex 16

Další věcí, se kterou měli uživatelé při testování problém, je viditelné ohraničení sítě, které se zdálo zbytečné. Rozhodl jsem se nastavení zobrazení ohraničení umístit do nastavení programu a uživatel si ho může kdykoliv zapnout.

Ex 17

Předposledním problémem byla chybějící funkce Přidat rozhraní ke komponentě. Tato funkce nebyla v požadavcích na systém, ale byl vznesen požadavek na její přítomnost. Proto jsem lehce přepracoval dialog s nastavením komponenty a implementoval mechanismus přidání rozhraní. Nyní je možné přidávat rozhraní ke komponentě podle potřeby.

Ex 18

Posledním problémem bylo zarovnání komponent do mřížky. Program zarovnával komponenty v kreslící ploše až k jejímu levému okraji, přičemž se stávalo, že nebyly vidět rozměrnější popisky pod komponentou. Tento problém byl vyřešen zarovnáním na další pozici (ve vodorovné ose).

6.1.3 Shrnutí testování experty

Během testování aplikace experty se podařilo odhalit osmnáct problémů, u kterých jsem provedl nápravná opatření. Do další fáze testování (testy s uživateli) je aplikace připravena již bez odhalených chyb.

6.2 Testy použitelnosti

Testování použitelnosti, anglicky Usability testing, je testování s uživateli, které má ověřit kvality programu v reálném prostředí při typickém použití.

6.2.1 Typický uživatel

Typickým cílovým uživatelem této aplikace je student předmětu Počítačové sítě (BI-PSI) na ČVUT FIT. Předmět BI-PSI se vyučuje v druhém ročníku bakalářského studia, proto se dá očekávat věk typického uživatele okolo 20-22 let. U studentů se předpokládá také znalost oboru informačních technologií a základní vědomosti o počítačových sítích.

Druhým typem uživatele je člověk, který aplikaci vyhledá na internetu a spustí si ji v domácím prostředí. Takový uživatel se bude zřejmě zajímat o problematiku počítačových sítí a předpokládám, že bude i zkušeným uživatelem počítače.

6.2.2 Cíl testování

Cílem testů je ověření správnosti návrhu a realizace grafického uživatelského rozhraní implementované aplikace. Během testů použitelnosti je potřeba odhalit chyby a nedostatky, které je nutné odstranit před nasazením aplikace do provozu.

6.2.3 Nastavení testu

Prostředí

Testování proběhlo především v prostředí, kde se očekává nasazení programu, tedy přímo v hodině předmětu BI-PSI s cílovou skupinou uživatelů (studenti). Otestováni byli i lidé, kteří splňují definici druhého typu uživatele aplikace, tedy ti, kteří by si mohli program stáhnout z internetu a používat ho v domácích podmínkách.

Způsob testování

Testování každého jednotlivce má tři části. Nejdříve testovaná osoba vyplní předtestový dotazník, poté provede testové úlohy a nakonec vyplní potestový dotazník. Uživatel by si měl s úkoly poradit bez pomoci zadavatele testu a v případě problémů by měl využít nápovědu zabudovanou v aplikaci.

6.2.4 Předtestový dotazník

Předtestový dotazník (příloha E), anglicky zvaný screener, má zjistit, zda je daná osoba pro testování vhodná, či jaké má zkušenosti v daném oboru. V našem případě jsou studenti předmětu BI-PSI vhodnými kandidáty a potřebu-

jeme zjistit, jak jsou na tom se znalostmi v oboru počítačových sítí a zda se někdy setkali s nějakým síťovým simulátorem.

Výsledky předtestového dotazníku

Na testování se podařilo zajistit 20 studentů předmětu BI-PSI a 4 osoby, které předmět nestudují a reprezentují druhý typ uživatele aplikace.

	Student PSI	**Znalosti o počítačových sítích**	**Zkušenost se simulátorem**	**Pokud ano, s jakým**
P 1	ne	expert	ano	Cisco Packet Tracer
P 2	ne	začátečník	ne	-
P 3	ne	začátečník	ne	-
P 4	ano	pokročilý	ano	Cisco Packet Tracer
P 5	ano	pokročilý	ano	Cisco Packet Tracer
P 6	ano	pokročilý	ano	Cisco Packet Tracer
P 7	ano	pokročilý	ne	-
P 8	ano	pokročilý	ano	Cisco Packet Tracer
P 9	ano	pokročilý	ne	-
P 10	ano	pokročilý	ne	-
P 11	ano	pokročilý	ne	-
P 12	ano	pokročilý	ne	-
P 13	ano	mírně pokročilý	ne	-
P 14	ano	mírně pokročilý	ne	-
P 15	ano	mírně pokročilý	ne	-
P 16	ano	pokročilý	ano	Cisco Packet Tracer
P 17	ano	mírně pokročilý	ne	-
P 18	ano	mírně pokročilý	ne	-
P 19	ano	mírně pokročilý	ne	-
P 20	ano	mírně pokročilý	ne	-
P 21	ano	mírně pokročilý	ne	-
P 22	ano	mírně pokročilý	ano	Cisco Packet Tracer
P 23	ano	mírně pokročilý	ne	-
P 24	ne	pokročilý	ne	-

Tabulka 6.1: Výsledky předtestového dotazníku (příloha E)

Z předtestového dotazníku je patrné, že většina uživatelů se řadí mezi mírně pokročilé, či pokročilé v oboru počítačových sítí. Zajímavým faktem je, že přibližně třetina testovaných uživatelů má nějakou zkušenost se síťovým simulátorem. Ještě zajímavější je, že zkušenosti mají pouze s jediným simulátorem, a to programem Cisco Packet Tracer.

6.2.5 Testovací úkoly

Následuje seznam testovacích úkolů, které testované osoby prováděly.

T 1: Zprovozněte aplikaci na svém počítači.

T 2: Spusťte aplikaci.

T 3: Přepněte jazyk na češtinu.

T 4: Vyhledejte v nápovědě termín *vkládání komponent*.

T 5: Vytvořte nový projekt.

T 6: Vyberte v nástrojích switch se 16 porty.

T 7: Vložte switch se 16 porty na plochu.

T 8: Vložte na plochu dva počítače, jeden notebook a Linux router se čtyřmi porty.

T 9: Spojte komponenty kabelem. Zkuste vybrat rozhraní ručně.

T 10: Rozmístěte síť tak, jak se vám to líbí.

T 11: Zarovnejte komponenty do mřížky.

T 12: Nastavte IP adresy a masky u koncových zařízení a Linux routeru.

T 13: Nastavte u nějakého kabelu jiné zpoždění než výchozí (10).

T 14: Skryjte zobrazení MAC adres.

T 15: Uložte projekt.

T 16: Zavřete projekt.

T 17: Otevřete přiložený projekt s názvem linux_3_pocitace.xml.

T 18: Spusťte z příkazové řádky síťový simulátor (backend) příkazem java -jar psimulator2_backend.jar linux_3_pocitace.xml.

T 19: Přepněte se v grafickém editoru so simulačního režimu.

T 20: Spojte se se síťovým simulátorem (backend) na portu 12000.

T 21: Spusťte zachytávání paketů.

T 22: Otevřete telnet okno u počítače pc0.

T 23: Pošlete jeden ping na zařízení pc2 (ping 192.168.1.3 -c 1).

T 24: Vypněte zachytávání paketů.

T 25: Přehrajte zachycené pakety.

T 26: Uložte zachycené pakety.

T 27: Smažte zachycené pakety.

T 28: Zkuste si poslat další PINGy, hrajte si s přehráváním paketů.

T 29: Změňte velikost ikon v toolbaru na menší.

T 30: Přepněte se zpět do editačního režimu.

T 31: Spusťte automatické rozmístění komponent.

T 32: Ukončete aplikaci.

6.2.6 Potestový dotazník

V potestovém dotazníku (příloha F) byly dva druhy otázek: otázky s výběrem a vypisovací otázky. Možné odpovědi u otázek s výběrem odpovídají stupnici 1-5 (od nejlepší po nejhorší) a je z nich možné spočítat průměr. Pro snazší ztotožnění uživatele s možnostmi u odpovědí jsem v dotazníku použil slovní hodnocení. Vypisovací otázky slouží k získání různých informací o tom, zda se ta která věc v programu líbí, či nelíbí.

Výsledky potestového dotazníku - otázky s výběrem

V tabulce 6.2 jsou uvedeny výsledky potestového dotazníku. Grafické rozhraní uživatelé hodnotili průměrně známkou 1,75, práci v editačním režimu známkou 1,92, práci v simulačním režimu známkou 1,75 a práci s nápovědou známkou 1,77. Celkový dojem z aplikace ohodnotili průměrně známkou 1,63.

Výsledky potestového dotazníku - vypisovací otázky

Při analýze odpovědí na vypisovací otázky jsem zjistil, že jsou věci, které některým uživatelům vyhovují, zatímco jiným ne. Příkladem je menu s nástroji, které funguje na stejném principu, jako v grafickém editoru Adobe Photoshop. Někteří uživatelé si menu chválili a několika uživatelům se nelíbilo. Podobným problémem byly obrázky v nápovědě. Někteří uživatelé chválili dobře viditelné velké obrázky a některým připadaly příliš veliké.

Nejvíce uživatelé kritizovali nutnost instalace Javy ve verzi 1.7, absenci přístupu k vlastnostem komponenty přes dvojklik myší a výše uvedené nástrojové menu.

Pozitivní reakce jsem zaznamenal zejména na pěknou grafiku uživatelského rozhraní, jednoduché a intuitivní ovládání, dostatečnou velikost ikon a přehledné animace. Mnoho uživatelů uvedlo, že aplikace je tzv. user friendly, neboli že je uživatelsky přívětivá.

	Grafické rozhraní	Práce v editačním režimu	Práce v simulačním režimu	Práce s nápovědou	Celkový dojem
P 1	1	1	1	1	1
P 2	2	3	2	1	2
P 3	1	1	2	1	1
P 4	2	2	3	3,5	2,5
P 5	5	2	2	2	2
P 6	2	2	2	2	1,5
P 7	2	2	1	2	1
P 8	2	1	2	2	1
P 9	2	2	1	1	1
P 10	1	2	1	1	2
P 11	2	3	2	2	1
P 12	2	3	2	1	2
P 13	2	2	1	2	2
P 14	1	2	3	2	2
P 15	1	2	1	2	1
P 16	1	2	3	1	3
P 17	3	1	2	3	2
P 18	2	2	2	1	2
P 19	1	3	2	3	2
P 20	1	1	1	2	2
P 21	2	3	2	2	2
P 22	1	2	2	3	1
P 23	2	1	1	1	1
P 24	1	1	1	1	1
Avg.	1,75	1,92	1,75	1,77	1,63

Tabulka 6.2: Výsledky potestového dotazníku (příloha F)

6.2.7 Nalezené problémy

Následuje výčet problémů, které se při testování uživateli podařilo odhalit.

Ut 1: V menu Naposledy otevřené chybí informace o kompletní cestě k souboru.

Ut 2: Okno s výběrem souboru si nepamatuje poslední otevřený adresář při dalším spuštění programu.

Ut 3: Během ukládání a načítání sítě se nezobrazuje tzv. „přesýpací kurzor“.

Ut 4: Při uložení nového projektu se cesta k souboru nepřidá do Naposledy otevřených.

Ut 5: Nutnost instalace Javy 1.7.

Ut 6: Nefunguje dvojklik na komponentě.

Ut 7: Nutnost spojení s backendem.

Ut 8: Označené řádky v tabulce se ztracenými pakety jsou špatně čitelné.

Ut 9: Malé ikony v menu nápovědy.

Ut 10: Překlep v anglickém překladu v názvu směrovače.

6.2.8 Řešení nalezených problémů

Problémy nalezené během uživatelského testování jsem analyzoval a pokusil se k nim nalézt odpovídající řešení. Následuje komentář k jednotlivým bodům.

Ut 1

Problém absence informace o kompletní cestě k souboru v menu Naposledy otevřené jsem vyřešil přidáním tzv. tooltipu[17], ve kterém je zobrazena kompletní cesta k souboru.

Ut 2

Zapamatování posledního otevřeného adresáře v okně s výběrem souboru jsem vyřešil uložením cesty k poslednímu adresáři do nastavení programu.

Ut 3

Při ukládání či načítání větší sítě program na chvilku přestal reagovat a uživatel nevěděl, co se děje. Z tohoto důvodu jsem při ukládání či načítání ze souboru nastavil přepnutí na tzv. wait kurzor, který dává najevo, že program něco dělá.

Ut 4

Při testování se objevila chyba, že po vytvoření nového projektu a jeho uložení se cesta k souboru nepřidala do naposledy otevřených. Tuto chybu jsem odstranil.

[17] Tooltip - popisek v bublině zobrazený po najetí myši.

Ut 5

Uživatelé si stěžovali, že je ke spuštění aplikace potřeba instalovat Javu ve verzi 1.7. Tento problém se jako jeden z mála vyřeší sám, a to tím, že v brzké době bude verze 1.7 distribuována společností SUN v rámci automatických aktualizací. Nebude tedy nutné ji kvůli této aplikaci instalovat.

Ut 6

Tři uživatelé si stěžovali, že aplikace nepodporuje dvojklik myší na komponentě pro vstup do nastavení komponenty. Dialog s nastavením je přístupný přes pravé tlačítko myši. Dvojklik není podporován z důvodu velké komplexnosti nástroje ruka (hand) v editoru, který umožňuje manipulaci se sítí a označování komponent najednou. Přidání dvojkliku by zapříčinilo mnohonásobně složitější kód tohoto nástroje a také zpoždění reakcí programu, jelikož dvojklik je nutné detekovat jako dva samostatné kliky s nějakou pauzou mezi nimi. Aplikace by musela čekat na druhý klik a teprve kdyby se neobjevil, tak by provedla reakci na samostatný klik.

Ut 7

Další pro uživatele méně příjemnou věcí je nutnost spouštění dvou aplikací a jejich vzájemné spojování. Tato vlastnost vznikla z důvodu rozdělení projektu do z velké části samostatných projektů tak, aby v případě nejvyšší nouze bylo možné práci dodělat i bez některé z částí. Nevýhodou je nutnost spuštění dvou aplikací: grafického rozhraní a samotného simulátoru.

Ut 8

Během testování jsme objevili nejednotnost jednotlivých vzhledů grafického rozhraní (tzv. Look and Feel), kdy na operačním systému Windows je při nastavení tmavého pozadí v tabulce text automaticky bílý, ale na systému Linux zůstane text černý. To způsobuje špatnou čitelnost tmavě podbarvených řádků v tabulce. Problém byl vyřešen explicitním nastavením barvy textu na tmavém pozadí na bílou.

Ut 9

Systém Java Help disponuje ve výchozím nastavením malými ikonami v menu, které nejsou výrazné a uživatel je snadno přehlédne. Tyto ikony jsem nahradil novými. Ikona vyhledávání[23] pochází ze sady Oxygen Icons a je k dispozici pod licencí GPL[10]. Ikona knihy[15] pochází ze sady Copenhagen a je k použití pod licencí Creative Commons Attribution. Ostatní ikony pocházejí ze sady ikon Glaze icon set, kterou jsem popisoval v kapitole Návrh.

Ut 10

Překlep v názvu směrovače v anglickém překladu byl opraven.

6.3 Další testy

Kromě uživatelských testů byla aplikace otestována také zátěžovým testem. Na závěr jsem otestoval také Genetický algoritmus.

6.3.1 Zátěžový test

Cílem zátěžového testu bylo zjistit, zda grafické rozhraní bude výkonově dostačovat zachytávání velkého množství událostí ze simulátoru a jejich přehrávání. Test proběhl na následující konfiguraci:

```
Operační systém: Windows 7 32bit
Java: JRE 1.7.0_03
CPU: AMD Turion X2 2.0 GHz
RAM: 3GB
```

Test jsem provedl s využitím generátoru událostí, který je schopný simulovat připojení k serveru a přijímání událostí.

1. fáze

Nejdříve jsem vyzkoušel generovat, přijímat a vizualizovat 500 událostí za sekundu. Při tomto testu jsem nezpozoroval žádné zpomalení rychlosti přehrávání, ani zamrzání okna či zpomalení reakcí uživatelského rozhraní na podněty uživatele.

2. fáze

Ve druhé fázi testu jsem nastavil generování 1000 událostí za vteřinu. Ani při této konfiguraci nedošlo k žádným negativním vlivům. Aplikace zatížení zvládá velice dobře a nebude úzkým hrdlem při reálném použití.

6.3.2 Test genetického algoritmu

Implementovaný genetický algoritmus jsem testoval dvěma způsoby. Prvním bylo otestování doby běhu algoritmu a druhým bylo otestování kvality výstupu.

Doba běhu

Dobu běhu algoritmu jsem testoval na následujících sítích:

- Mřížka 3x3, 12 kabelů, 9 komponent,
- páry, 6 kabelů, 6 komponent,
- křížení, 18 kabelů, 9 komponent,
- úplný graf K5, 10 kabelů, 5 komponent,
- větší síť 1, 17 kabelů, 16 komponent,
- větší síť 2, 25 kabelů, 26 komponent.

Rozmístění jsem provedl desetkrát a z naměřených časů jsem spočítal průměr. Následuje tabulka s dobou běhu pro jednotlivé sítě.

Síť	**Průměrná doba běhu [s]**
Mřížka	9,3
Páry	8,1
Křížení	22,1
Úplný graf K5	3,4
Větší síť 1	22,3
Větší síť 2	44,2

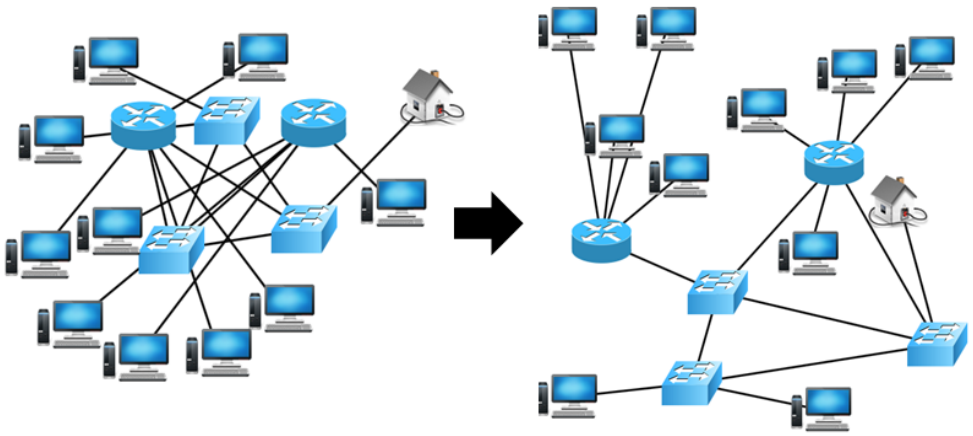
Tabulka 6.3: Doba běhu algoritmu pro jednotlivé sítě

Kvalita výstupu

Na obrázku 6.5 je dobře vidět výsledek automatického rozmístění sítě v ploše. Výsledek algoritmu je pokaždé jiný, protože se jedná o heuristický přístup. Více ukázek je možné nalézt v příloze D.

6.4 Závěr testování

Aplikace PSImualtorUI byla důkladně otestována nejdříve experty a poté uživateli. Při testování experty bylo odhaleno 18 nedostatků, které byly před uživatelskými testy odstraněny. Dvacet studentů předmětu BI-PSI otestovalo program ve výukovém prostředí a čtyři další osoby otestovali program v domácím prostředí. Během uživatelského testování bylo odhaleno dalších deset nedostatků, u kterých bylo buďto navrženo a provedeno opravné opatření,



Obrázek 6.5: Výsledek automatického rozmístění počítačové sítě

nebo bylo zdůvodněno, proč náprava není nutná nebo možná. Průměrně uživatelé hodnotili aplikaci známkou 1,63, což je podle mého názoru velmi dobrý výsledek.

V zátěžovém testu se ukázalo, že uživatelské rozhraní nebude úzkým hrdlem při zachytávání velkého množství paketů a test genetického algoritmu ukázal, že ačkoliv je funkce automatického rozmístění v programu spíše zajímavostí, lze s ní dosáhnou rozumného rozmístění sítě.

Aplikace je po testech připravena k nasazení do ostrého provozu v předmětu BI-PSI.

KAPITOLA 7

Zhodnocení

7.1 Zhodnocení

Cílem této práce bylo vytvořit grafické uživatelské rozhraní, které by umožňovalo vytvoření virtuální počítačové sítě, její nastavení a simulaci průchodu paketů v takové síti. Důraz byl kladen na dostatečně jednoduché a přívětivé uživatelské rozhraní.

Aplikace byla podle zadání vytvořena a splňuje všechny požadavky uvedené v kapitole Analýza. Důležité je, že aplikace je kompatibilní se síťovým simulátorem, na kterém pracovali další studenti v rámci projektu PSImulator.

Po vytvoření byla aplikace otestována nejdříve experty a poté uživateli. Uživatelské testy proběhly na vzorku dvaceti studentů předmětu BI-PSI a na čtyřech dalších osobách. Během testování se podařilo najít množství nedostatků, které byly odstraněny, a získat řadu podnětů ke zlepšení, které byly zapracovány. Na konci testování uživatelé hodnotili aplikaci velmi kladně a přisoudili jí průměrnou známku velmi dobře.

Aplikace je nyní připravena k plnohodnotnému nasazení k výukovým účelům v předmětu BI-PSI.

7.2 Porovnání s existujícími řešeními

V kapitole 3 jsem analyzoval existující řešení na trhu síťových simulátorů a stanovil základní parametry, které by měla ideální aplikace splňovat: multiplatformnost, uživatelskou přívětivost a licenci, která umožní použití v hodinách předmětu Počítačové sítě. Všechny tyto parametry výsledná aplikace splňuje a myslím si, že je zdatnou konkurencí k existujícím řešením. Aplikace vyniká především jednoduchým, přehledným a intuitivním grafickým rozhraním, které neodrazuje uživatele.

7.3 Možnosti vylepšení

Výslednou aplikaci by bylo zajisté možno vylepšit v různých směrech. Jedním z nich je přidání dalšího jazyka, na což je aplikace připravena a k čemuž není nutný zásah do zdrojových kódů. Dalším je přidání hardwarových prvků, ze kterých by bylo možné stavět virtuální sítě. Příkladem můžou být bezdrátové prvky. Pokud by byla měla být prakticky využívána funkce automatického rozmístění, pak by bylo vhodné vylepšit rychlost aktuálně implementovaného algoritmu nebo navrhnout a implementovat zcela jiný typ algoritmu, který by zkrátil dobu výpočtu.

Závěr

V této práci se podařilo navrhnout a implementovat aplikaci, která splňuje všechny zadané požadavky. Aplikace v editačním režimu umožňuje interaktivně vytvořit a nastavit virtuální počítačovou síť a v simulačním režimu umožňuje vizualizovat průchod paketů sítí. Množství zobrazovaných informací o síti je možné měnit a vytvořenou síť je možné uložit v požadovaném formátu XML. Aplikace je kompatibilní se síťovým simulátorem PSImulator a umožňuje přijímat jím vygenerované události, které je možné podle potřeby uložit či opětovně načíst taktéž ve formátu XML. Pro automatické rozmístění sítě v ploše byl využit genetický algoritmus, který v implementované variantě zajišťuje téměř vždy minimální křížení hran sítě.

Výsledná aplikace byla podrobena uživatelskému testování, ve kterém se ukázalo, že je na velmi dobré úrovni a že je připravena k plnohodnotnému nasazení v předmětu BI-PSI na Fakultě informačních technologií.

Literatura

[1] Boson - NetSim 8.0. http://www.boson.com/, stav z 5. 3. 2012.

[2] Cisco Systems, Inc. - Cisco Packet Tracer. http://www.cisco.com/web/learning/netacad/course_catalog/PacketTracer.html, stav z 5. 3. 2012.

[3] Creative Commons (Attribution-Noncommercial 3.0). http://creativecommons.org/licenses/by-nc/3.0/, stav z 19. 3. 2012.

[4] Dynampis Cisco 7200 simulator od Christophera Fillota. http://www.ipflow.utc.fr/index.php/Cisco_7200_Simulator, stav z 5. 3. 2012.

[5] Řehák, S.: *Simulátor virtuální počítačové sítě Cisco*. Bakalářská práce, ČVUT, FEL, 2010.

[6] Ethernet cable icon. http://www.iconarchive.com/show/vista-hardware-devices-icons-by-icons-land/Ethernet-Cable-icon.html, stav z 19. 3. 2012.

[7] Garey, M. R.; Johnson, D. S.: Crossing number is NP-complete. *SIAM Journal on Algebraic and Discrete Methods*, ročník 4, č. 3, 1983: s. 312–316.

[8] Glaze icon set - sada ikon od Marco Martina. http://www.iconarchive.com/show/glaze-icons-by-mart.html, stav z 19. 3. 2012.

[9] GNS3 Graphical Network Simulator. http://www.gns3.net/, stav z 5. 3. 2012.

[10] GPL - General Public License. http://www.gnu.org/copyleft/gpl.html, stav z 19. 3. 2012.

[11] Hand icon - ikona pro nástroj Ruka. http://www.iconfinder.com/icondetails/42209/48/cursor_drag_fingers_hand_pressed_icon, stav z 19. 3. 2012.

[12] Icons Unleashed Vol. 1 - sada ikon s ikonami notebooku a pracovní stanice. http://www.iconfinder.com/browse/iconset/icons-unleashed-vol1/, stav z 19. 3. 2012.

[13] Ikona auta. http://www.iconfinder.com/icondetails/40764/128/delivery_truck_icon, stav z 19. 3. 2012.

[14] Ikona balíčku. http://www.iconfinder.com/icondetails/38964/128/box_delivery_generic_inventory_package_product_icon, stav z 19. 3. 2012.

[15] Ikona knihy. http://www.iconfinder.com/icondetails/49335/64/book_icon, stav ze 13. 4. 2012.

[16] Ikona nápovědy. http://www.iconfinder.com/icondetails/27853/128/help_question_rectangular_support_icon, stav ze 3. 4. 2012.

[17] Ikona obálky. http://www.iconfinder.com/icondetails/56291/128/envelope_mail_icon, stav z 23. 3. 2012.

[18] Ikona počítače. http://icontexto.blogspot.com/2008/02/elite-icons.html, stav z 19. 3. 2012.

[19] Ikona pro automatické rozmístění. http://www.iconfinder.com/icondetails/21868/24/alignment_icon, stav z 19. 3. 2012.

[20] Ikona reálného počítače. http://www.iconfinder.com/icondetails/45416/128/connected_home_house_local_network_icon, stav z 19. 3. 2012.

[21] Ikona terminálu. http://www.iconfinder.com/icondetails/18728/16/console_dos_terminal_icon, stav z 19. 3. 2012.

[22] Ikona výběru. http://www.iconfinder.com/icondetails/64627/16/area_continuous_select_icon, stav z 19. 3. 2012.

[23] Ikona vyhledávání. http://www.iconfinder.com/icondetails/8826/32/, stav ze 13. 4. 2012.

[24] Ikona XML. http://www.iconfinder.com/icondetails/8992/16/application_code_html_script_xml_icon, stav z 19. 3. 2012.

[25] Ikony hardwarových prvků router a switch. http://www.cisco.com/web/about/ac50/ac47/2.html, stav z 19. 3. 2012.

[26] Java Preferences API - nástroj pro ukládání uživatelských nastavení. http://docs.oracle.com/javase/1.4.2/docs/guide/lang/preferences.html, stav z 25. 3. 2012.

[27] JavaHelp system - Knihovna pro tvorbu nápovědy. http://javahelp.java.net/, stav ze 17. 3. 2012.

[28] JTA - Telnet/SSH for the JAVA platform. http://javassh.org/space/start, stav z 25. 3. 2012.

[29] Launch4j - Cross-platform Java executable wrapper. http://launch4j.sourceforge.net/, stav ze 13. 3. 2012.

[30] LGPL - Lesser General Public License. http://www.gnu.org/licenses/lgpl.html, stav z 19. 3. 2012.

[31] NetBeans Visual Library - knihovna pro vizualizace a grafově orientované modelování. http://platform.netbeans.org/graph/, stav ze 13. 3. 2012.

[32] Nielsenova heuristická analýza - 10 pravidel návrhu UI. http://www.useit.com/papers/heuristic/heuristic_list.html, stav z 19. 3. 2012.

[33] Návrhové vzory. http://objekty.vse.cz/Objekty/Vzory, stav ze 20. 3. 2012.

[34] Žižkovský PhD., I. P.: Heuristická analýza - 10 pravidel návrhu UI. Přednáška prototypes_testing_without_users, stav z 19. 3. 2012.

[35] RouterSim - Router and network simulator. http://www.routersim.com/CCNA6_home.html, stav z 5. 3. 2012.

[36] Simulační prostředí OMNeT++ od Andráse Vargy. http://omnetpp.org/, stav z 5. 3. 2012.

[37] Timing Framework - knihovna pro časování animací. http://java.net/projects/timingframework/pages/Home, stav ze 13. 3. 2012.

[38] WebNMS - Simulation Toolkit 7.0. http://www.webnms.com/simulator/index.html, stav z 5. 3. 2012.

PŘÍLOHA A

Seznam použitých zkratek

AMD Advanced Micro Devices

BI-PSI Předmět počítačové sítě bakalářského oboru na FIT

BSD Berkeley Software Distribution. Svobodná licence, která vyžaduje uvedení autora a informace o licenci.

CCNP Cisco certified network professional

CCNA Cisco certified network associate

ČVUT České vysoké učení technické

EDT Event Dispatching Thread

EXE Executable

FIT Fakulta informačních technologií

FP Funkční požadavek

GHz Gigahertz

GUI Graphical user interface

HTML HyperText Markup Language

IOS Internetwork Operating System

IP Internet Protocol

JAR Java Archive

JVM Java Virtual Machine. Platformově nezávislý virtuální počítač pro spouštění tzv. Java bytecode.

JRE Java Runtime Environment. Soubor prostředků (obsahuje mimo jiné JVM), který umožňuje spouštět programy napsané v jazyce Java.

LGPL Lesser General Public License

MVC Model-View-Controller

PNG Portable Network Graphics

PX Pixel

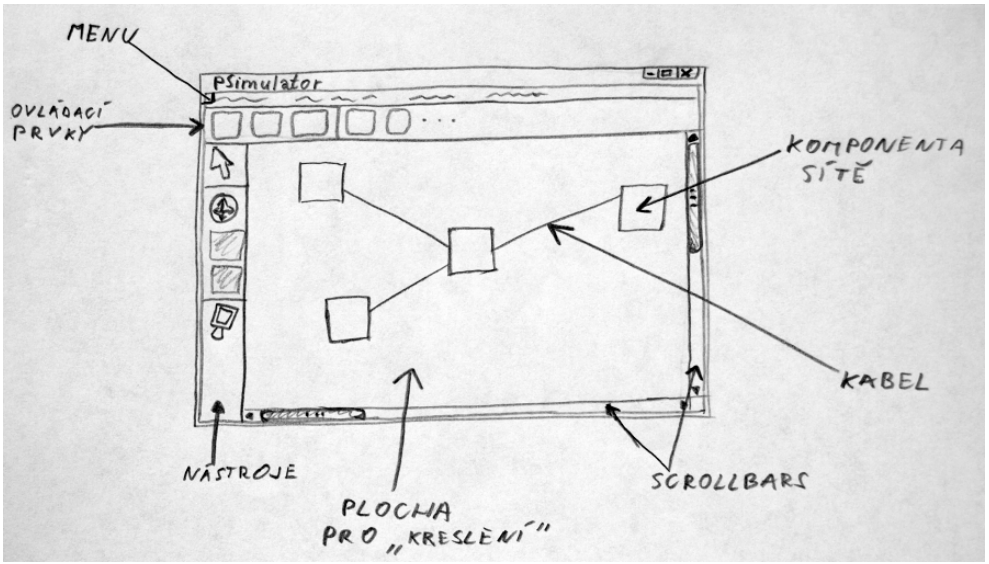
SE Standard edition

MAC Media Access Control

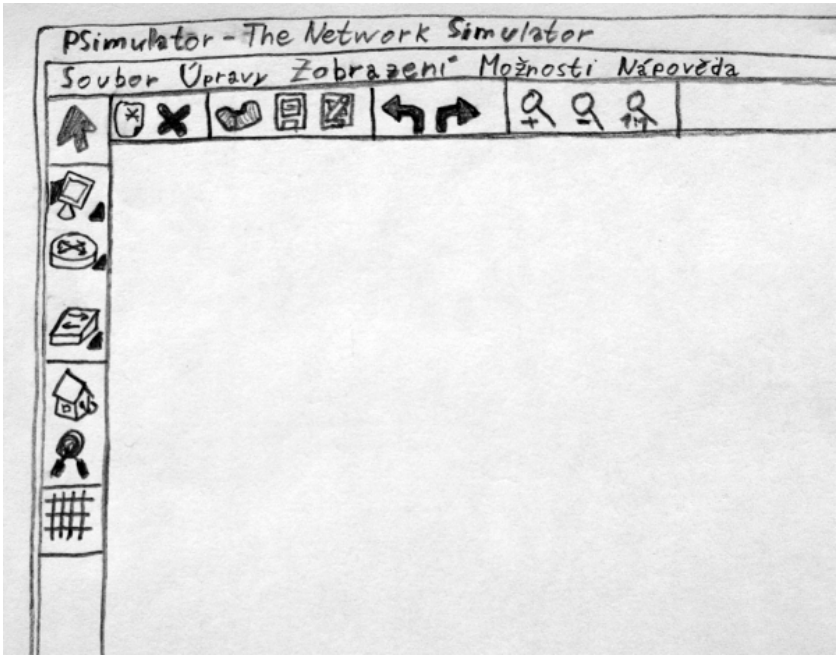
XML Extensible markup language

PŘÍLOHA B

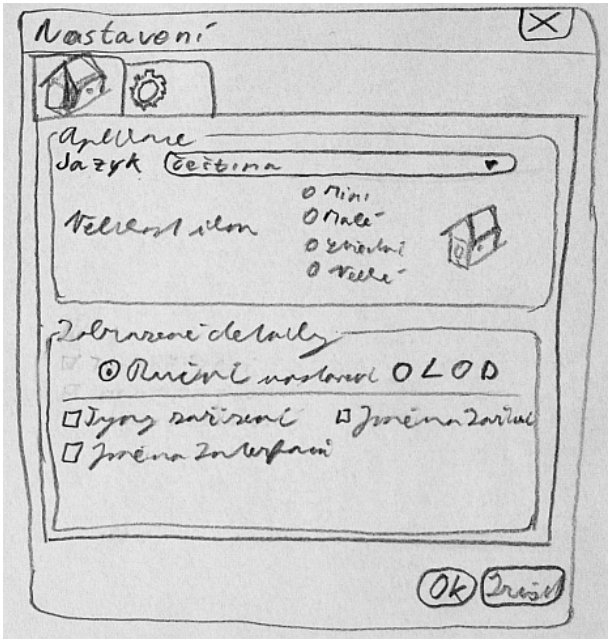
Prototyp



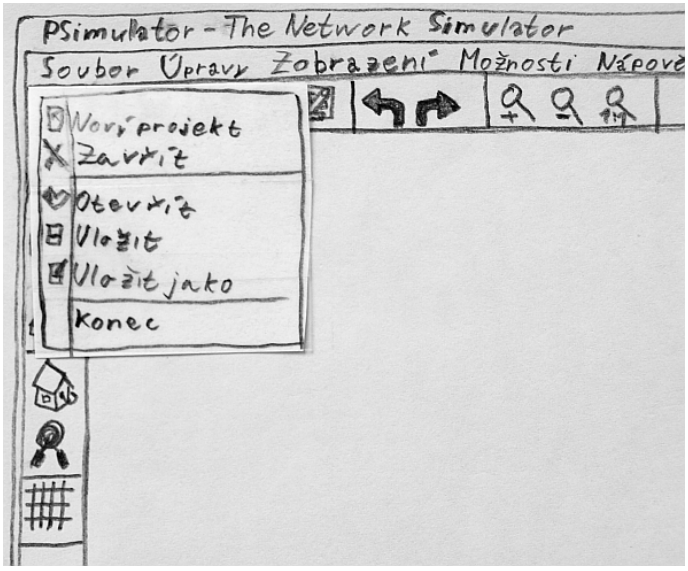
Obrázek B.1: První návrh uživatelského rozhraní bez detailů



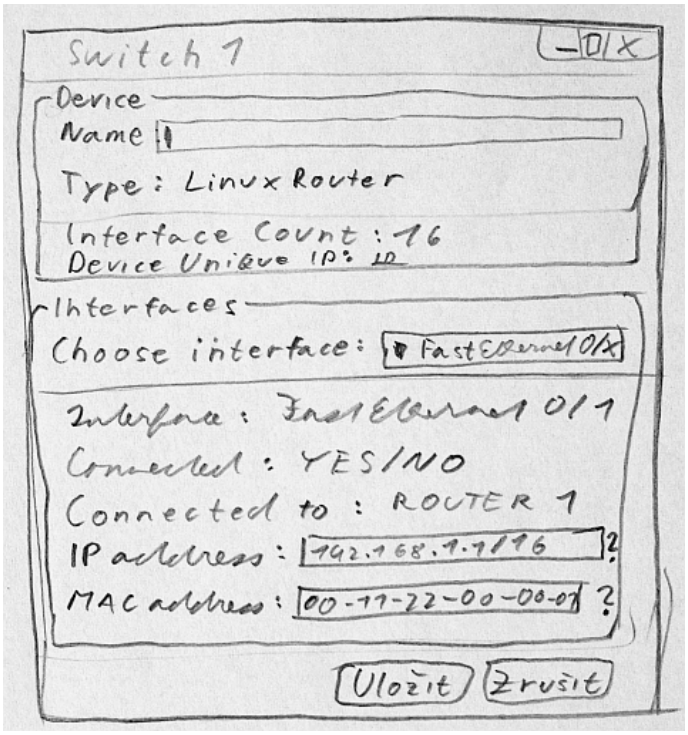
Obrázek B.2: Hlavní okno - rozvržení prvků s detaily



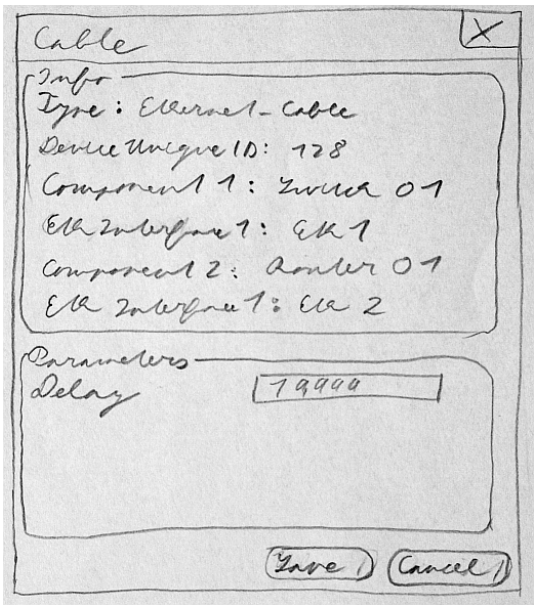
Obrázek B.3: Okno s nastavením programu



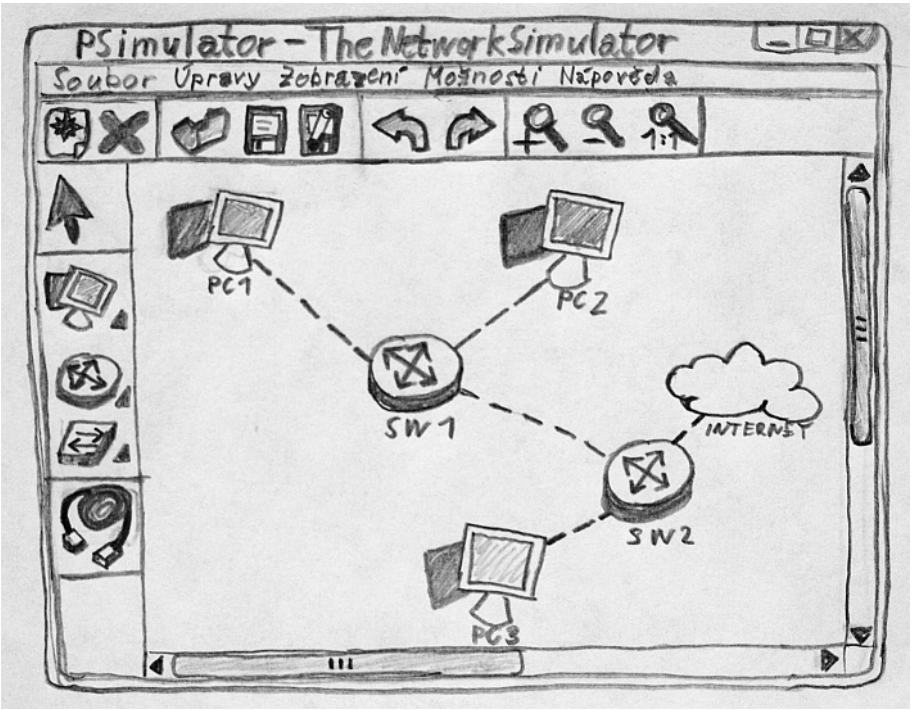
Obrázek B.4: Menu Soubor



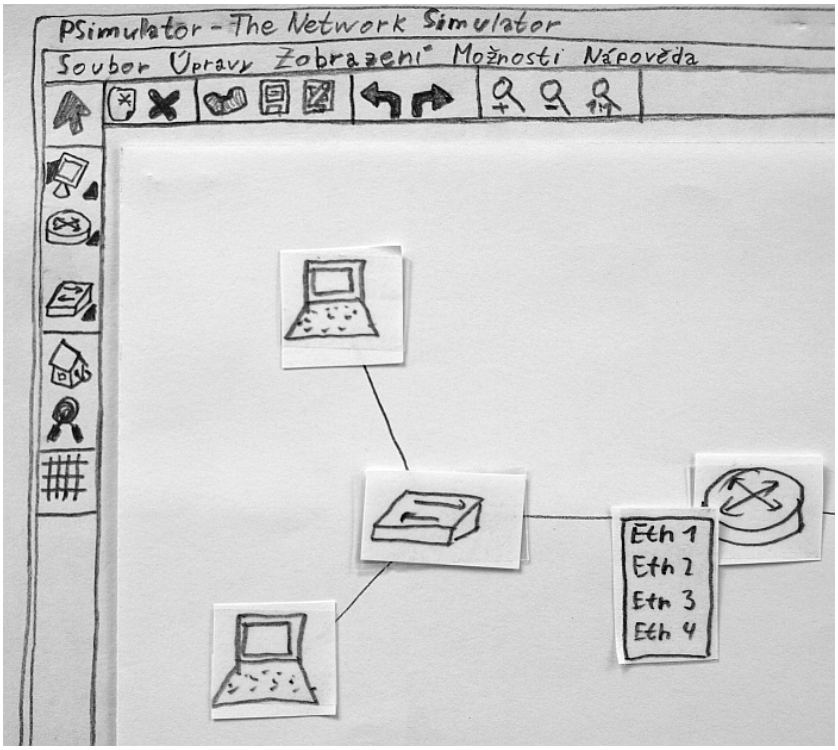
Obrázek B.5: Okno s nastavením komponenty (vylepšený návrh)



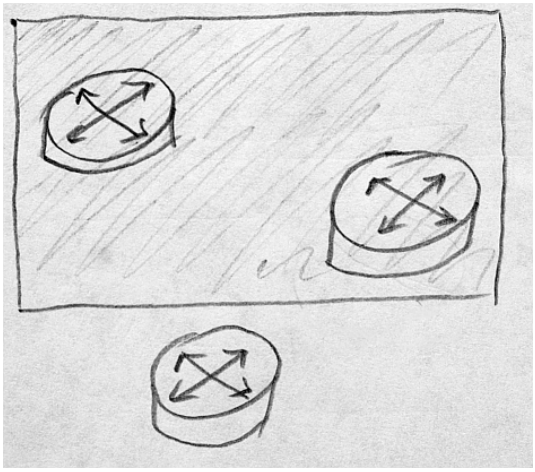
Obrázek B.6: Okno s nastavením kabelu



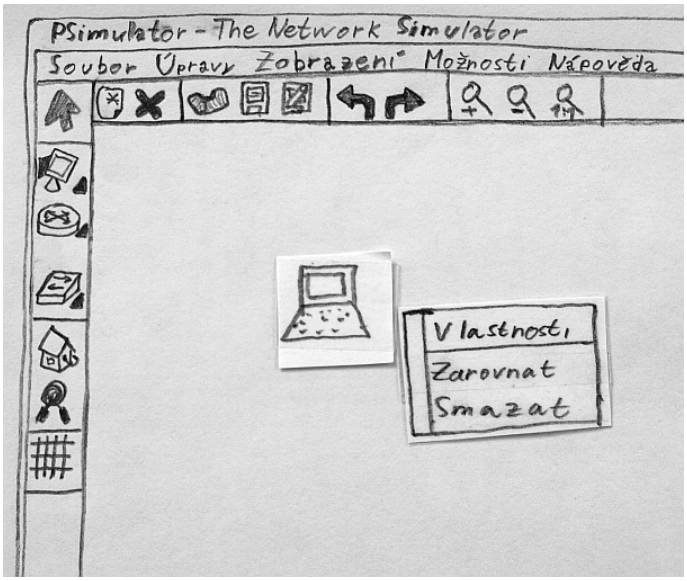
Obrázek B.7: Režim editoru - s vytvořenou sítí



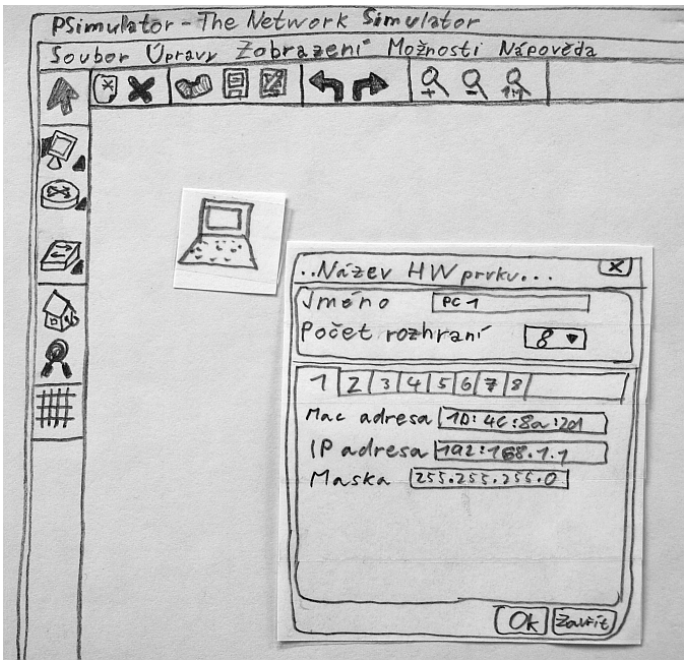
Obrázek B.8: Režim editoru - tvorba spojení mezi komponentami



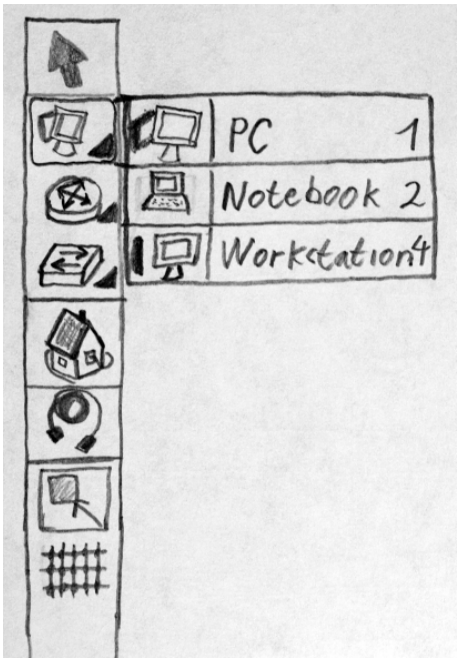
Obrázek B.9: Režim editoru - označování komponent



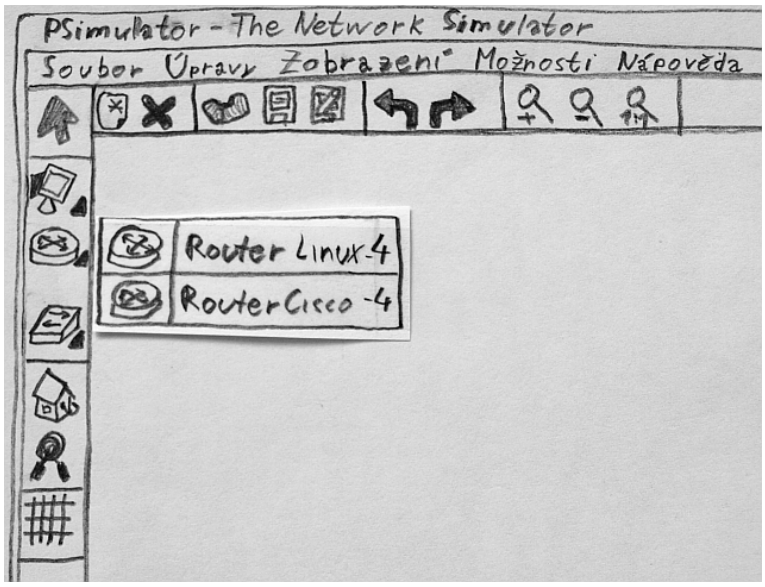
Obrázek B.10: Režim editoru - menu pod pravým tlačítkem myši na komponentě



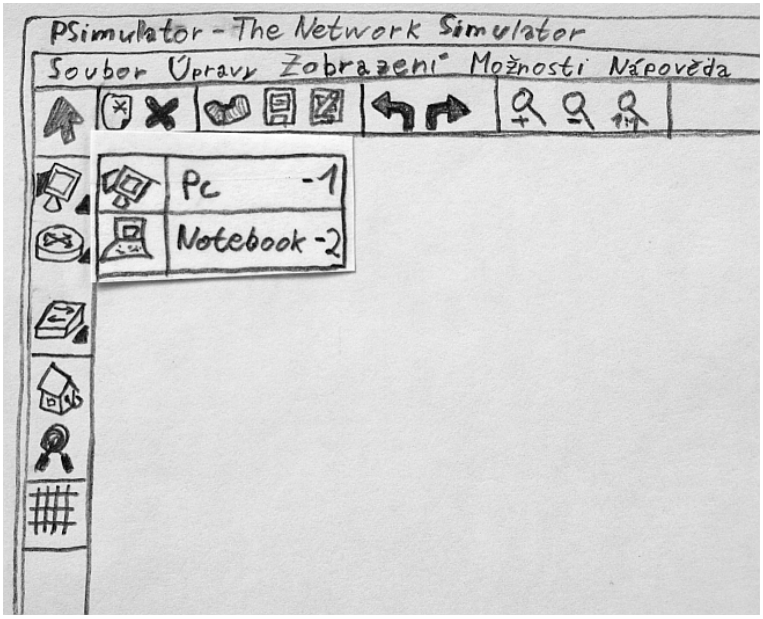
Obrázek B.11: Režim editoru - okno s vlastnostmi komponenty (původní návrh)



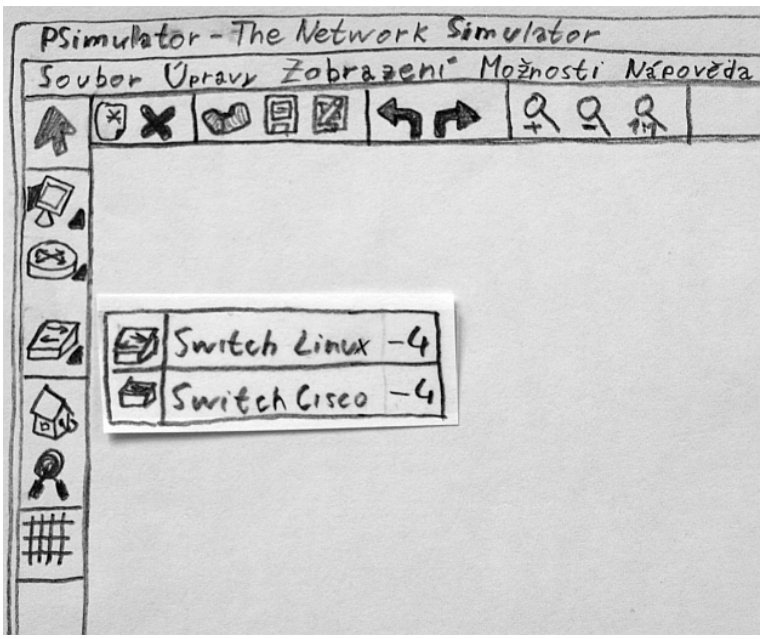
Obrázek B.12: Režim editoru - menu s výběrem nástroje



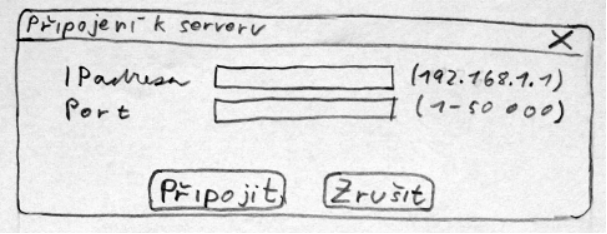
Obrázek B.13: Režim editoru - rozbalené menu s výběrem nástroje v kategorii Routery



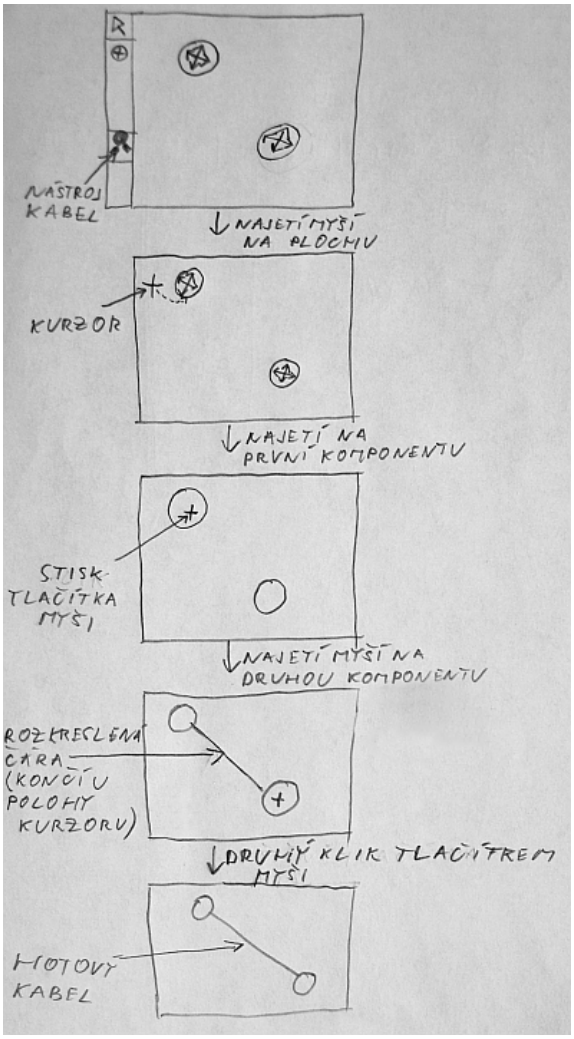
Obrázek B.14: Režim editoru - rozbalené menu s výběrem nástroje v kategorii Koncová zařízení



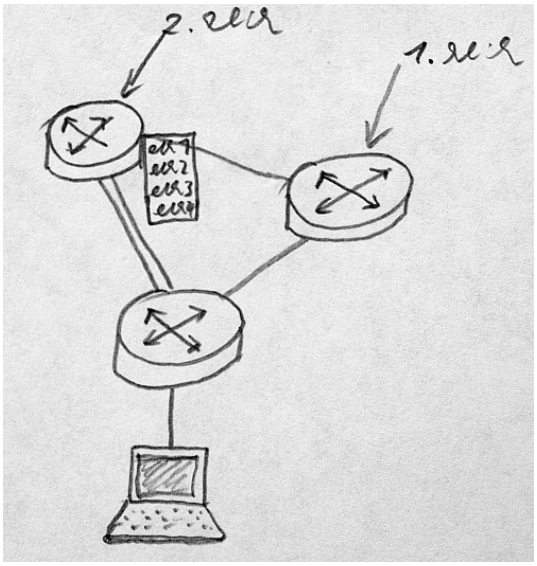
Obrázek B.15: Režim editoru - rozbalené menu s výběrem nástroje v kategorii Přepínače



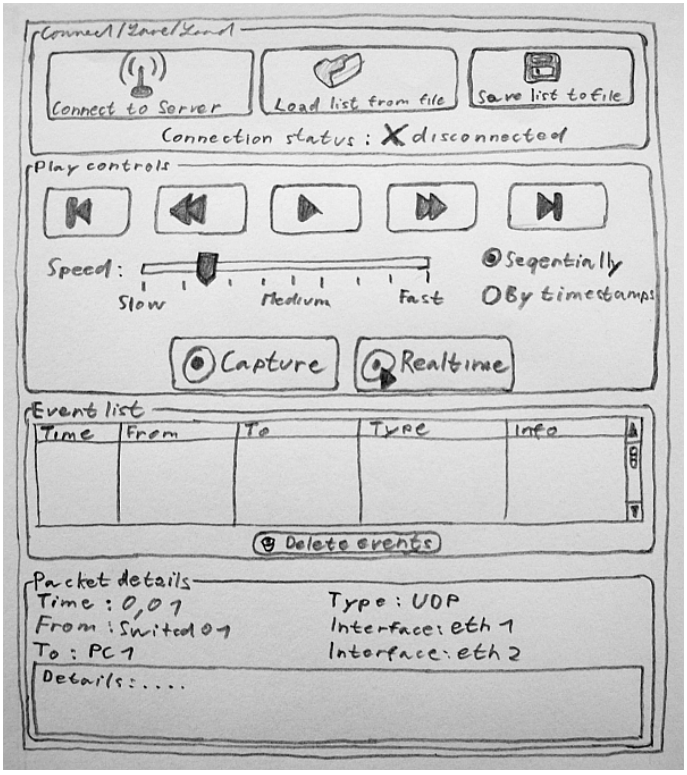
Obrázek B.16: Režim simulátoru - dialog připojení k serveru



Obrázek B.17: Režim editoru - spojování komponent



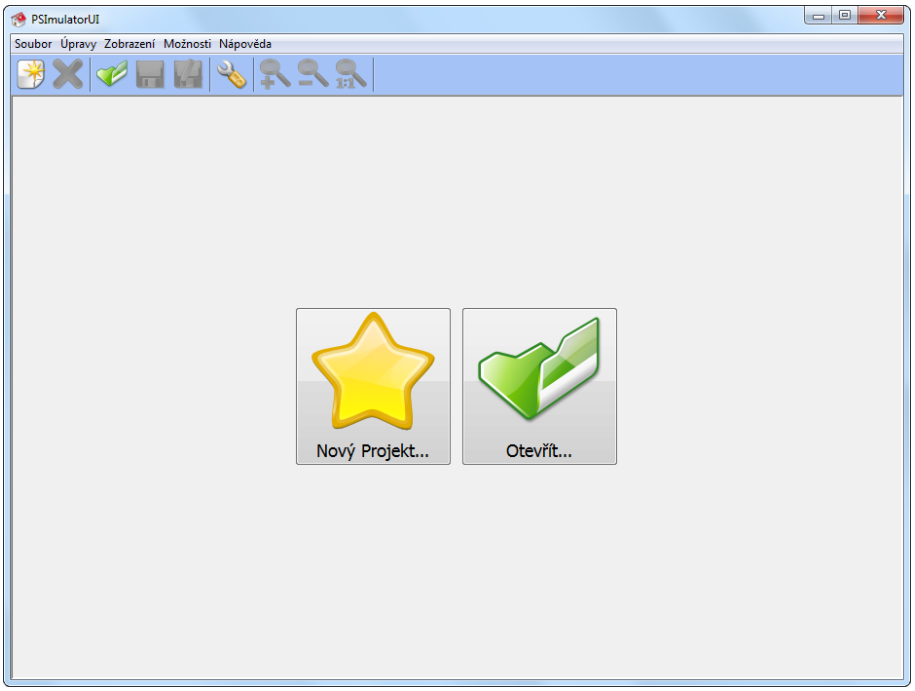
Obrázek B.18: Režim editoru - tvorba kabelu



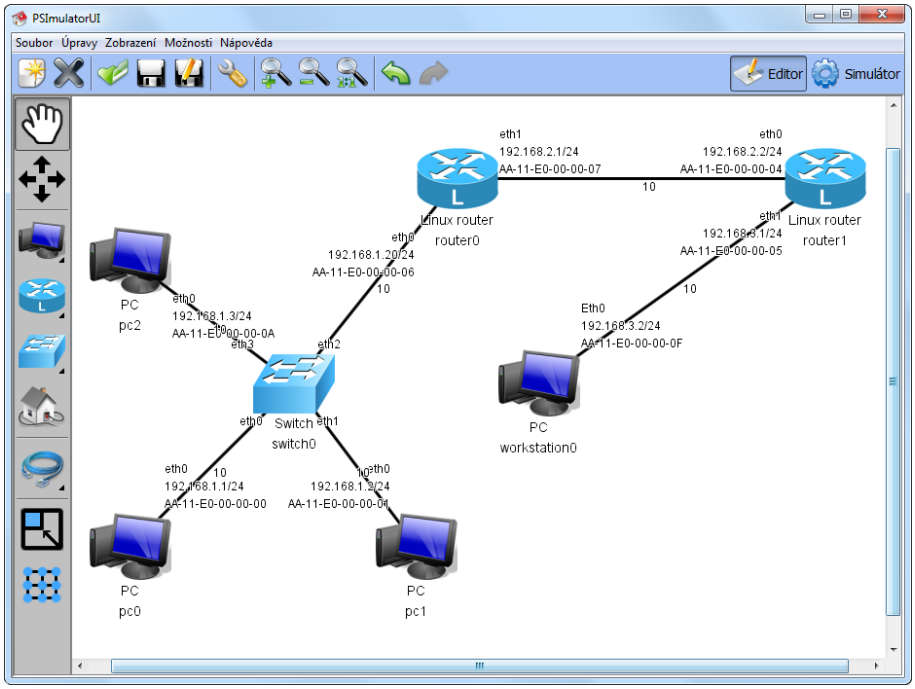
Obrázek B.19: Režim simulátoru - ovládací panel

PŘÍLOHA C

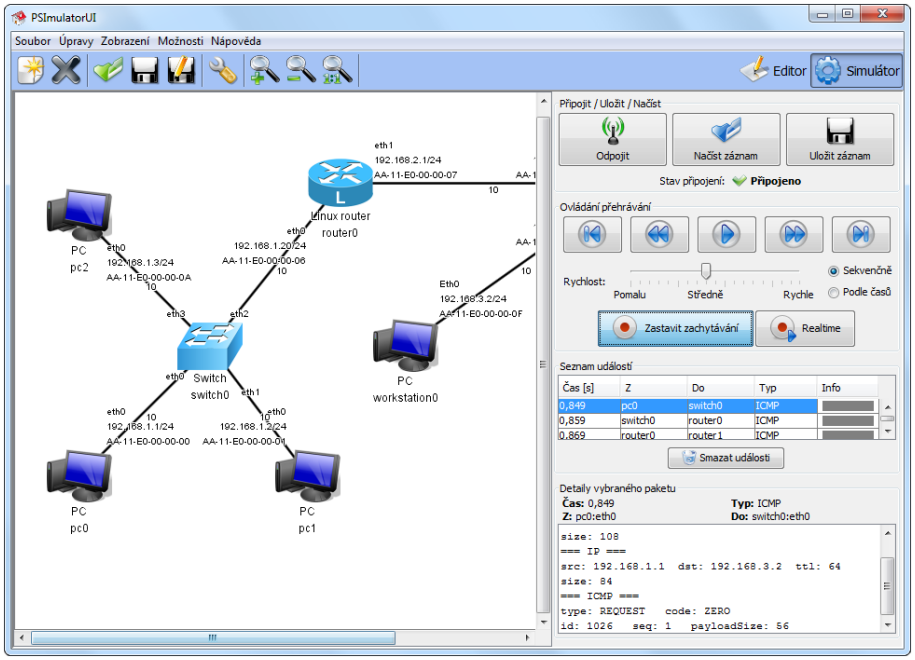
Výsledné grafické uživatelské rozhraní



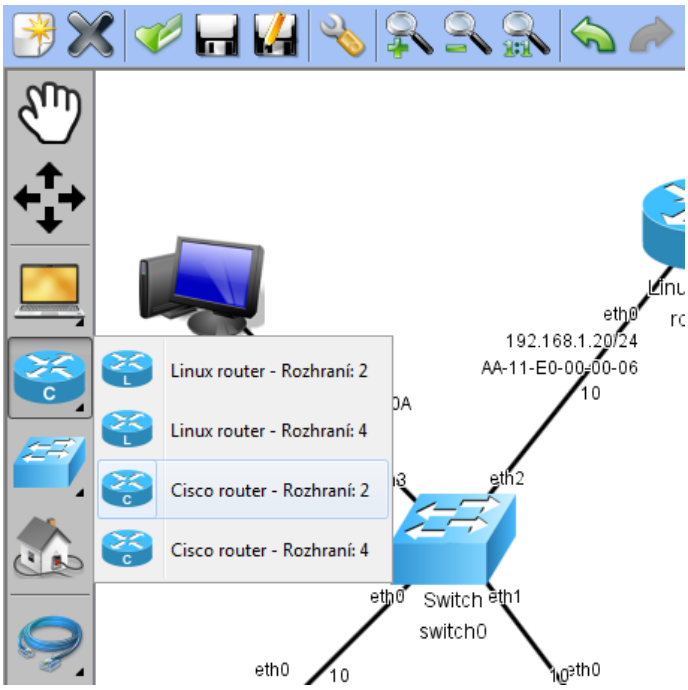
Obrázek C.1: Program po spuštění



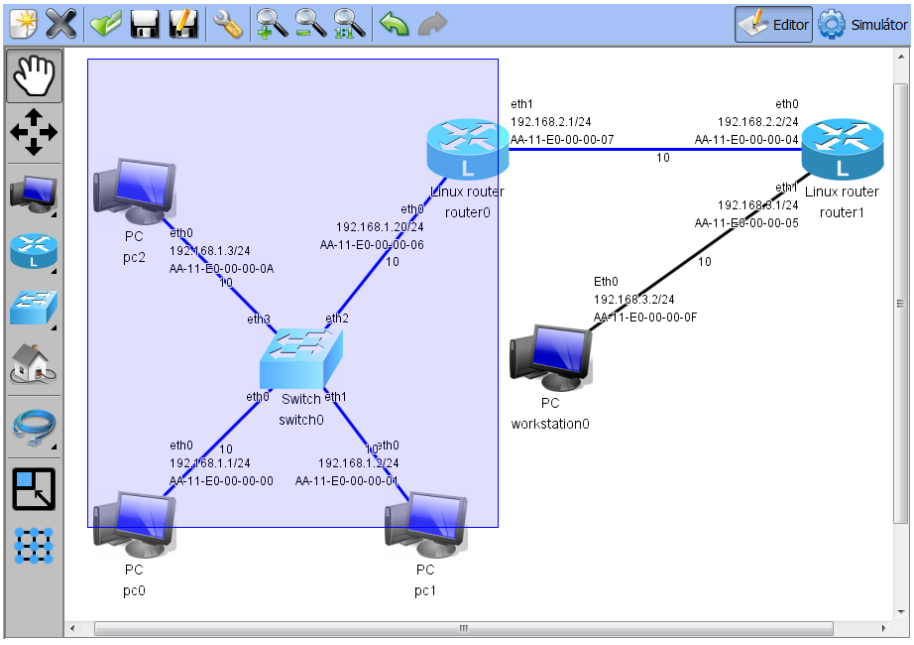
Obrázek C.2: Vytvořená síť v editačním režimu



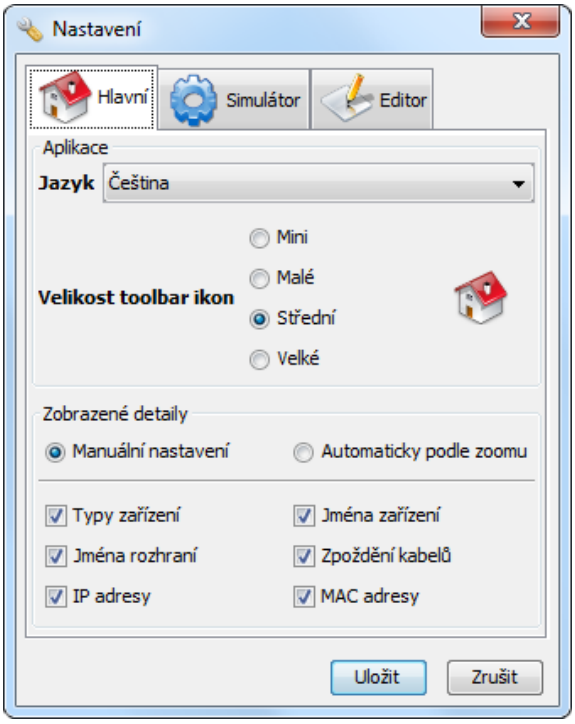
Obrázek C.3: Zachycené pakety v simulačním režimu



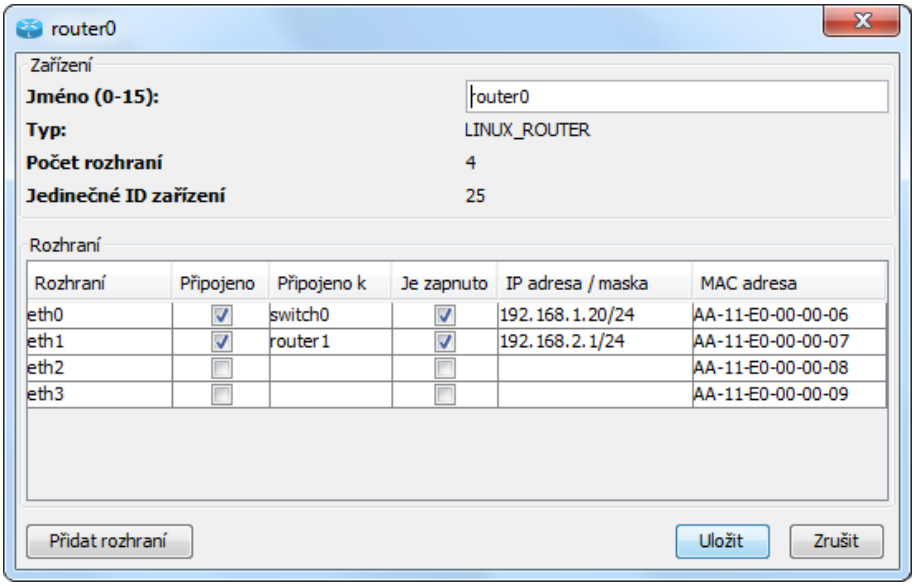
Obrázek C.4: Výběr komponenty v menu



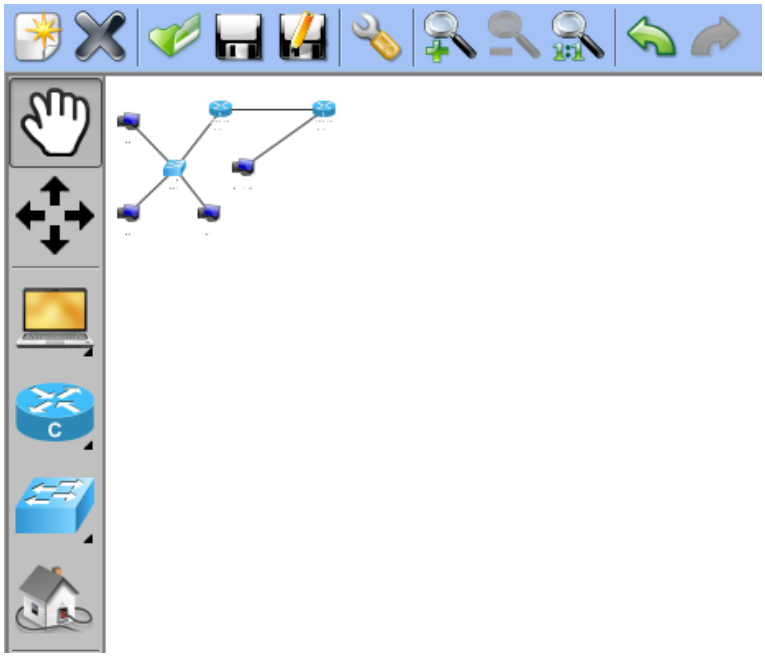
Obrázek C.5: Označování komponent



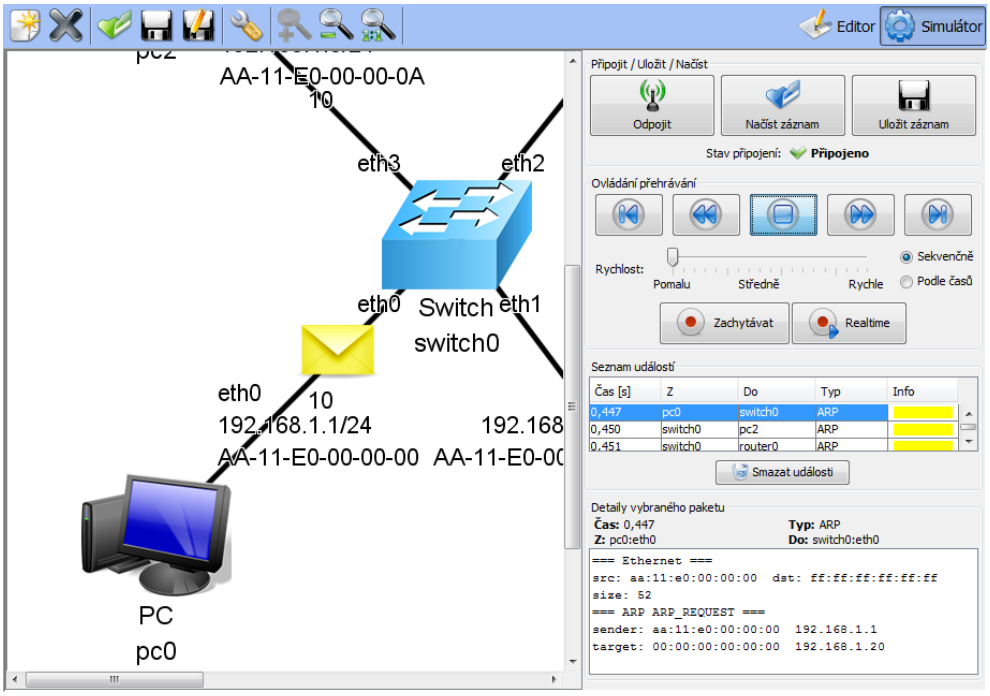
Obrázek C.6: Nastavení programu



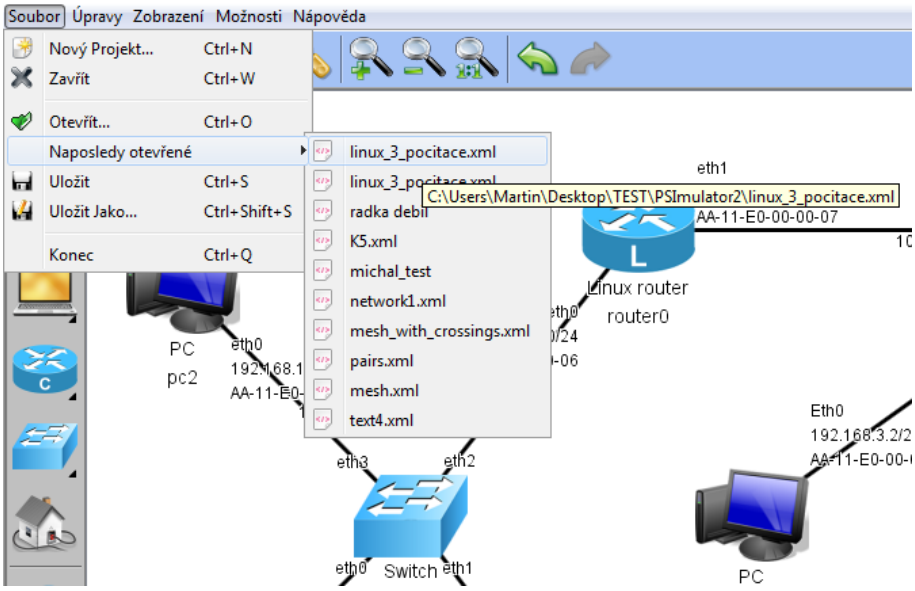
Obrázek C.7: Nastavení komponenty (routeru)



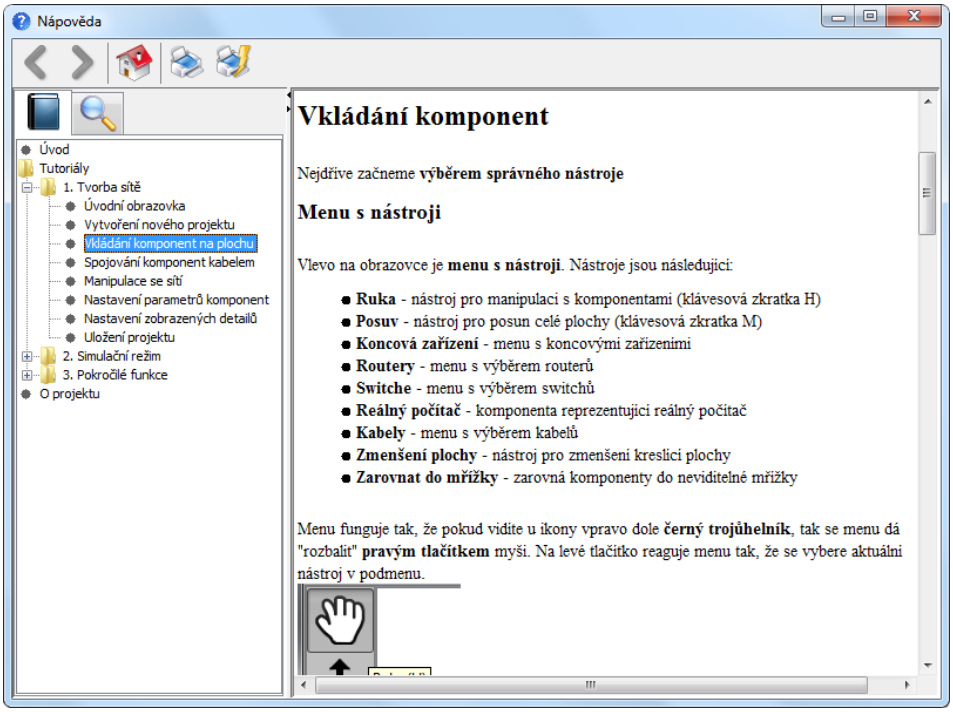
Obrázek C.8: Editor - minimální přiblížení



Obrázek C.9: Simulátor - animace paketů v maximálním přiblížení



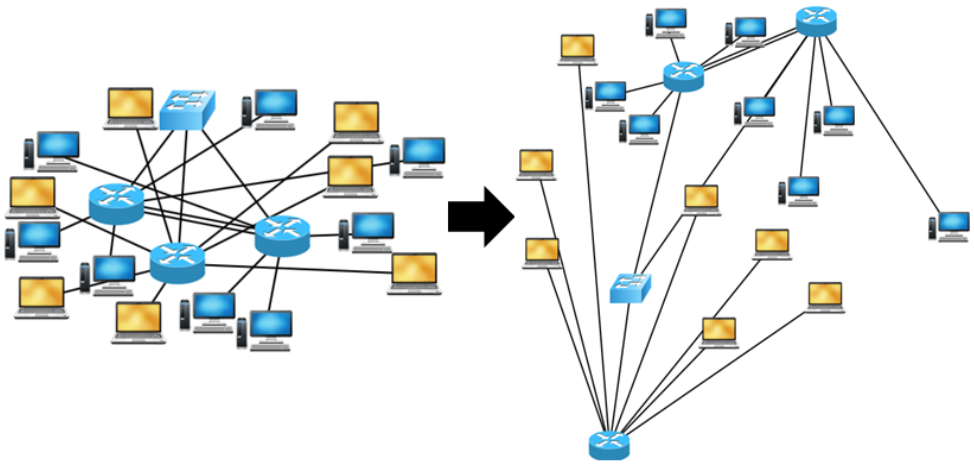
Obrázek C.10: Naposledy otevřené soubory s plnou cestou k souboru v tzv. „tooltipu“



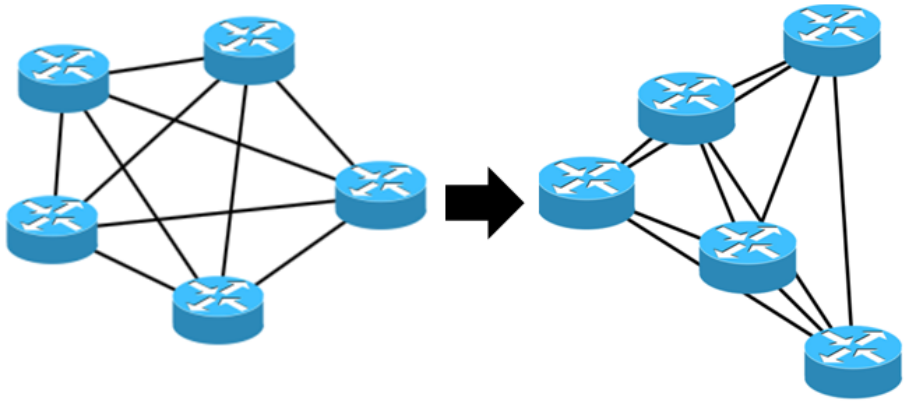
Obrázek C.11: Okno nápovědy

PŘÍLOHA D

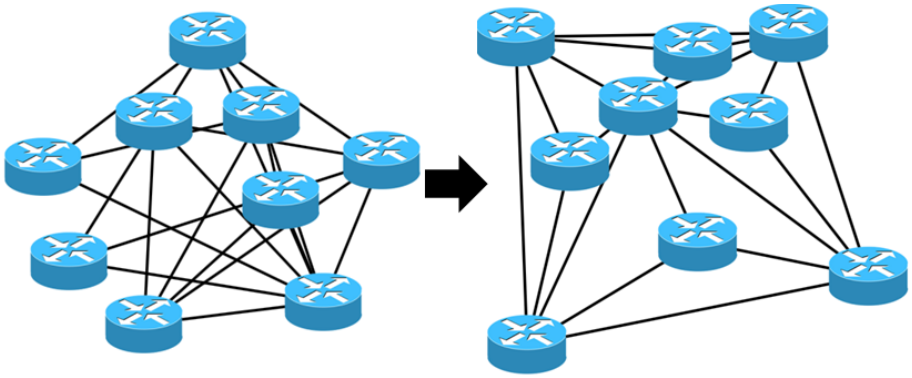
Ukázky automatického rozmístění



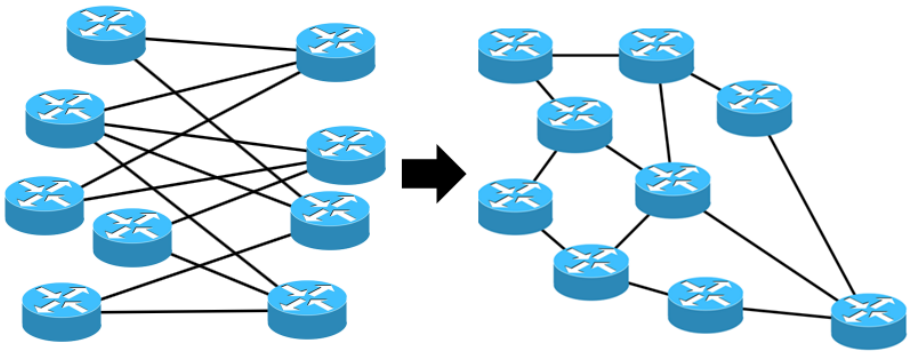
Obrázek D.1: Automatické rozmístění - počítačová síť



Obrázek D.2: Automatické rozmístění - úplný graf K5



Obrázek D.3: Automatické rozmístění - množství křížených hran



Obrázek D.4: Automatické rozmístění - mřížka 3x3

PŘÍLOHA E

Předtestový dotazník

1. **Jste student předmětu BI-PSI?**

2. **Ohodnoťte své znalosti o počítačových sítích:**
   - Téměř žádné, jsem začátečník
   - Něco málo vím, jsem mírně pokročilý
   - Vím toho dost, jsem pokročilý
   - Vím vše, jsem expert
3. **Máte zkušenost s nějakým síťovým simulátorem?**

4. **Pokud ano, s jakým a jak ho hodnotíte?**

PŘÍLOHA F

Potestový dotazník

1. **Jak se Vám líbí grafické rozhraní aplikace?**
   - velmi pěkné
   - pěkné
   - ujde to
   - nic moc
   - špatné
2. **Jak se Vám pracovalo s aplikací v editačním režimu?**
   - výborně
   - velmi dobře
   - dobře
   - špatně
   - velmi špatně
3. **Co Vám v editačním režimu vadilo?**

4. **Co na editačním režimu oceňujete?**

5. **Jak se Vám pracovalo s aplikací v simulačním režimu?**
   - výborně
   - velmi dobře
   - dobře
   - špatně
   - velmi špatně
6. **Co Vám v simulačním režimu vadilo?**
7. **Co na simulačním režimu oceňujete?**
8. **Jak se Vám pracovalo s nápovědou aplikace?**
   - výborně
   - velmi dobře
   - dobře
   - špatně
   - velmi špatně
9. **Co Vám na nápovědě vadilo?**
10. **Co na nápovědě oceňujete?**
11. **Jakým dojmem na Vás aplikace působí jako celek?**
    - **1** (výborně, bez chyby)
    - **2** (velmi dobře, občas mi něco nevyhovuje)
    - **3** (dobře, nedostatků je více)
    - **4** (špatně, aplikace mi vůbec nevyhovuje)
    - **5** (velmi špatně, nic horšího jsem nikdy nepoužíval(a))
12. **Jaké máte další dojmy a připomínky?**

PŘÍLOHA G

Obsah přiloženého CD

Přiložené CD obsahuje následující adresáře a soubory:

- `readme.html` ..... stručný popis obsahu CD
- `dist` ..... adresář se spustitelnou formou implementace
  - `windows` ..... spustitelné soubory pro OS Windows
  - `other` ..... spustitelné soubory pro ostatní systémy
- `html` ..... adresář s dokumentací a nápovědou
  - `Help` ..... adresář s nápovědou k programu
    - `Czech` ..... nápověda v českém jazyce
    - `English` ..... nápověda v anglickém jazyce
  - `Javadoc` ..... adresář s dokumentací ke zdrojovým kódům
    - `PSImulatorUI` ..... Dokumentace k projektu PSImulatorUI
    - `PSImulatorShared` .... Dokumentace k projektu PSImulatorShared
- `src` ..... adresář se zdrojovými soubory
  - `impl` ..... zdrojové kódy implementace
    - `PSImulatorUI.zip` ..... archiv s projektem pro NetBeans 7.1
    - `PSImulatorShared.zip` ..... archiv s projektem pro NetBeans 7.1
    - `SwingTelnet.zip` ..... archiv s projektem pro NetBeans 7.1
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
    - `DP_Švihlík_Martin_2012.zip` . archiv se zdrojovou forma práce ve formátu LaTeX
- `text` ..... text práce
  - `DP_Švihlík_Martin_2012.pdf` ..... text práce ve formátu PDF