Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Diplomová práce

Webový nástroj pro analýzu sítí front

Plzeň 2019 Pavel Kotva

Místo této strany bude zadání práce.

Prohlášení

Prohlašuji, že jsem diplomovou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 15. května 2019

Pavel Kotva

Poděkování

Rád bych poděkoval vedoucímu mé diplomové práce panu Ph.D. Ing. Richardu Lipkovi za odborné vedení, věnovaný čas a cenné rady, které mi pomohly při tvorbě této práce.

Abstract

This thesis deals with implementation of simulation the queueing theory. It contains a brief introduction to this problematic. There is also an analysis of existing applications, which are using for modeling these systems. The main part of thesis is desing and implementation own web application, which allows to create and then simulate queueing networks. The application should be used primary for the learning needs of the subject that deals with this problematic.

Abstrakt

Práce se zabývá implementací simulace systémů hromadné obsluhu. Obsahuje základní informace o těchto systémech. Dále se v ní nachází rozbor různých existujících aplikací, které slouží k modelování těchto systémů. Hlavní částí je pak návrh a implementace vlastní webové aplikace, která umožňuje vytvářet a následně simulovat sítě front. Aplikace by měla sloužit primárně pro učební potřeby předmětu, který se touto problematikou zabývá.

Obsah

1 Úvod

Již více než sto let se lidé zabývají problematikou sítí front. Co dříve začalo u telefonních rozvoden, dnes používáme v širokém spektru oblastí, kde tyto sítě pomáhají vytvářet nové nebo zlepšovat stávající systémy.

Není tedy takovým překvapením, že tuto problematiku nalezneme i v předmětu na naší univerzitě. Konkrétně jde o předmět výkonnost a spolehlivost programových systémů.

Právě pro učební potřeby tohoto předmětu vznikla tato diplomová práce. Jejím cílem je vytvořit webový nástroj sloužící pro modelování, včetně rozsáhlé konfigurace jednotlivých uzlů, a následnou simulaci systémů hromadné obsluhy. Výsledky této simulace by měli být v aplikaci zobrazeny.

Ač v současné době existují některé nástroje, které by se mohly pro podobné potřeby využít, každý z nich má jisté nevýhody, které jejich použití při výuce komplikují.

Výsledná webová aplikace by se těchto nevýhod měla vyvarovat. Její použití by pro studenty mělo být jednoduché a intuitivní. Musíme také myslet na fakt, že vyučující bude potřebovat nějaké výsledky jako výstup od studentů. Proto v aplikaci nesmí chybět možnost nejen exportu výsledků simulace, ale také importu a exportu samotného modelu systému.

2 Systémy hromadné obsluhy

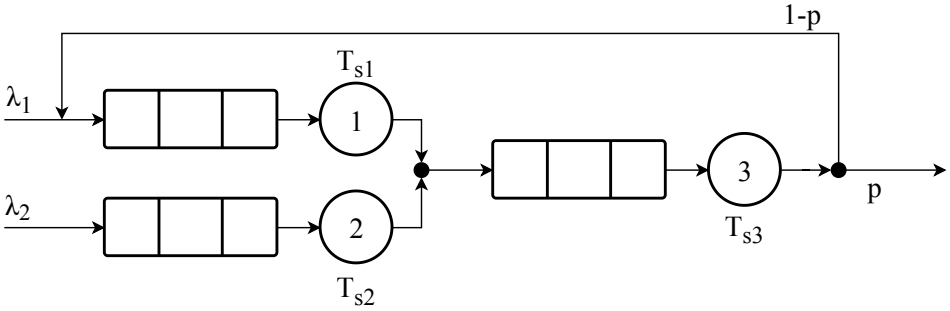
Systémy hromadné obsluhy, také známé pod pojmy teorie front nebo sítě front, jsou matematické modely popisující chovaní poměrně složitých obslužných systémů při různé zátěži. Díky tomu jsme pak schopni analyticky nebo simulačně zjistit určité výkonové vlastnosti tohoto systému ještě před jeho skutečným vytvořením.

Nejlepší představu si uděláme pomocí příkladu na reálném problému, který všichni známe. Proto využijeme menzu naší univerzity, která představuje celý systém hromadné obsluhy. Přicházející lidi do menzy popisujeme jako vstupní tok systému. Každý člověk je pak jeden požadavek procházející celým systémem. Jednotlivé úkony v menze jako je získání tácu s příborem, výběr jídla, konzumace jídla a odevzdání špinavého nádobí s tácem chápeme jako uzly obsluhy systému. Odchod s menzy pak popisujeme jako výstupní tok systému.

Modely by nám sloužit jako pomocník při rozhodování nebo určení ideálního řešení daného problému. Vycházejí ze statistiky, proto je pro jejich správnou funkčnost nezbytné větší množství požadavků. Ze začátku simulace nám model neposkytuje příliš přesná data, ale v jejím průběhu se tyto výsledky zpřesňují.

V dlouhodobém horizontu by se neměl model od reality příliš odlišovat. Většinou se v literatuře uvádí, že je potřeba okolo tisíce požadavků, které musejí projít systémem, abychom výsledky mohli považovat za relevantní.

Zjednodušeně můžeme systémy popsat jako přicházejících požadavky a uzly, které je zpracovávají a obsluhují. Požadavky pak mohou mezi jednotlivými uzly různě přecházet nebo u nich čekat. Tímto nám vznikne síť front, kterou můžeme vidět na obrázku 2.1.



Obrázek 2.1: Ukázka sítě front

2.1 Historie

První zmínky o teorii systémů hromadné obsluhy se objevují v dobách velkého rozmachu telefonních sítí v začátcích 20. století.

Jako prvního autora zajímající se o problematiku těchto modelů považujeme dánského inženýra A. K. Erlanga, který publikoval v roce 1909 článek ("The Theory of Probabilities and Telephone Conversations", Nyt Tidsskrift for Matematik B, vol 20, 1909.) zabývající se využití při modelování provozu telefonní ústředny. Na tomto základě pak pokračoval v práci a v roce 1917 publikoval svoji nejdůležitější práci ("Solution of some Problems in the Theory of Probabilities of Significance in Automatic Telephone Exchanges", Elektrotkeknikeren, vol 13, 1917.), kde definoval vzorce pro čekání a ztráty v teorii telefonního provozu.

V některých článcích[1] se můžeme dočíst o F. W. Johannsenovi, jakožto prvním člověkem zabívajícím se využití modelů hromadné obsluhy. Stejně jako A. K. Erlang, i on se věnoval teorii okolo telefonního provozu a publikoval o ní články v rocích 1907 a 1908, ale neměl na vývoj teorie systémů hromadné obsluhy tak zásadní vliv jako první zmíněný autor. Navíc je těžké posoudit, jestli jeho články můžeme zařadit do této oblasti.

2.2 Využití

Modely můžeme použít v různých odvětví a pro různé problémy, které se dají popsat jako požadavek a obsluha.

V dopravě například snadno popíšeme silniční síť města a auta, kde každá křižovatka značí obslužní bod a auta jsou našimi požadavky. Pro další dobrý příklad můžeme sáhnout do letecké dopravy, kde ranveje slouží jako obslužné místo, ať už pro přílet nebo vzlet, a letadla bereme jako požadavky. Podobné principy můžeme aplikovat i na úřady a další podobná místa, kde obslužný bod je úřednice a jednotliví lidé jsou požadavky.

Mohli bychom zde jmenovat ještě mnoho dalších příkladů využití, ale jejich princip je vždy velmi podobný ať už modely využíváme v IT, dopravě, službách nebo jiných odvětví.

2.3 Složení sítě

Systémy hromadné obsluhy se skládají z jednotlivých elementárních uzlů obsluhy, generátorů požadavků a případně uzlů konzumující zpracované po-

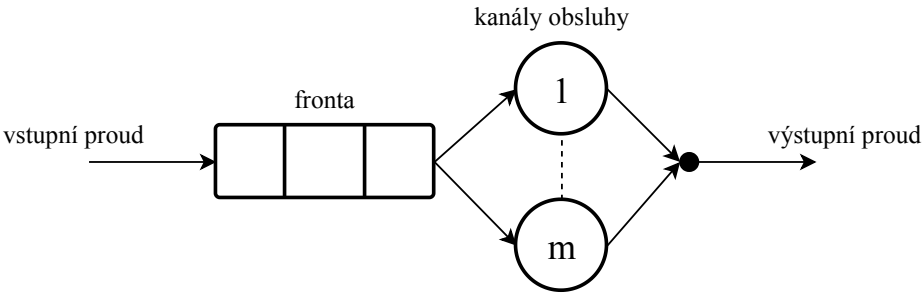
[1] `https://link.springer.com/article/10.1134/S0005117909120042`

žadavky. V některé literatuře se můžeme dočíst ještě o křižovatkách, které v našem případně zařadíme přímo do elementárního uzlu.

2.3.1 Elementární uzel sítě

Jedná se o nejdůležitější prvek sítě front, bez kterého by neměla smysl. Skládá se ze vstupního proudu, fronty, kanálu nebo kanálů obsluhy a výstupního proudu. Tento uzel můžeme vidět na obrázku 2.2.

Pro zjednodušení modelování jsme do něj zařadili i křižovatky, kterou připojíme hned za výstupní proud elementárního uzlu. Dále si jednotlivé části blíže rozebereme.



Obrázek 2.2: Elementární uzel sítě front

Vstupní proud

Udává časovou posloupnost vstupujících požadavků do uzlu. Tyto požadavky jsou zařazeny do fronty nebo zahozeny pokud je fronta již plná.

Mezi typické vstupní proudy se řadí Poissonovský, Gausovský a rovnoměrný. Pokud je vstupní proud stacionární, tedy nemění v čase, a nezávislý, tedy příchod následujícího požadavku není ovlivněn předchozím, je poměrně snadné nalézt analytické řešení.

Fronta

Další nedílná součást elementárního uzlu. Rozlišujeme dva základní druhy front a to o pevné délce nebo frontu nekonečnou. Pokud zvolíme frontu o pevné délce je třeba definovat co budeme dělat s prvky, které se do fronty již nevejdou. Tyto prvky obvykle zahazujeme a zaznamenáváme do statistiky systému.

Dále je možné určit jakým způsobem vybíráme prvky z fronty, zde se nám nabízí například FIFO (první přijde, první odejde), LIFO (poslední přijde, první odejde), náhodný výběr nebo jiný systém výběru. Pro snadné analytické řešení jsou ideální fronty FIFO, které jsou nekonečné a bez priorit.

Kanál obsluhy

Obsluhuje příchozí požadavky, které získává z fronty. Kanálů obsluhy může být i více, například máme jednu frontu čekajících na úřadě, ale obsluhuje je více okýnek. Nejčastěji narazíme na situace s jedním kanálem obsluhy. Stejně jako u vstupního proudu je pro výpočet jednoduché analytické řešení vhodné, aby pravděpodobnostní rozdělení bylo stejné pro všechny požadavky.

Výstupní proud

Výstup z kanálů obsluhy požadavků může mít stacionární nebo nestacionární režim.

Je-li celý systém hromadné obsluhy ve stacionárním režimu, tedy koeficient využití obsluhy je ostře menší jedné ($\rho < 1$), poté platí, že výstupní proud má stejnou střední periodu a frekvenci jako proud vstupní. Takový systém je schopen obsluhovat všechny příchozí požadavky v konečném čase.

Naopak čím více náhody v příchodech a době obsluhy je, tím více se budou tvořit fronty. Pokud je využití kanálu rovno nebo větší než jedna ($\rho \geq 1$) nazýváme tento system nestacionárním. Pro nestacionární režim systému hromadné obsluhy platí, že všechny kanály obsluhy stále pracují, systém je přetížený a jeho fronta nebo fronty narůstají.

2.3.2 Křižovatka

Určuje jakou cestou se dále vydá požadavek po opuštění elementárního uzlu v síti front. Toto rozhodnutí se dělá na základě pravděpodobnosti jednotlivých přechodů. V praxi to vypadá tak, že vygenerujeme náhodné číslo od nuly do jedné a podle toho do jakého z intervalů pro jednotlivé přechody toto číslo spadá, se vybere následující uzel. Je očekávané, že součet těchto pravděpodobností musí být rovný jedné.

2.4 Charakteristiky SHO

Uzel nebo celý systém můžeme popsat několika charakteristikami [12]. Díky tomu můžeme jednoduše získat jisté podvědomí o tom, co se v systému děje. Ty si krátce popíšeme níže:

$$\rho = \frac{1}{m} \cdot \frac{\lambda}{\mu} = \frac{1}{m} \cdot \frac{T_s}{T_a} \tag{2.1}$$

- ρ – využití kanálu (pro exponenciální doby příchodů)

  - $\rho = 0$ – nic nedělá
  - $\rho = 1$ – plně vytížen
  - $\rho > 1$ – fronta narůstá
- λ – střední frekvence příchodů požadavků
- μ – střední frekvence obsluh požadavků
- m – počet kanálů obsluhy
- T_s – střední doba obsluhy
- T_a – střední perioda příchodů

$$L_q = E(q) \tag{2.2}$$

- L_q – střední počet požadavků
- q – počet požadavků v systému (náhodná veličina)

$$T_q = E(t_q) \tag{2.3}$$

- T_q – střední doba odezvy
- t_q – odezva systému – doba průchodu jednoho požadavku systémem

2.5 Littleovy vzorce

V případě, že je systém stacionární, všechny fronty jsou FIFO a systém neobsahuje preempci, pak pro něj platí Littleovy vzorce [9]. Pomocí nich si můžeme odvodit některé jeho zajímavé vlastnosti. Ty platí pro jakékoliv pravděpodobnostní rozdělení vstupního proudu a můžeme je využít i pro složené systémy. Vzorce vypadají následovně:

$$L_q = \Lambda \cdot T_q \tag{2.4}$$

$$L_w = \Lambda \cdot T_w \tag{2.5}$$

$$T_w = L_w \cdot T_a \tag{2.6}$$

2.6 Kendallova klasifikace

Jde o jedno z možných klasifikací systémů hromadné obsluhy. Byla vytvořena v 50. letech minulého století D. G. Kendallem [8]. Systém klasifikujeme podle důležitých vlastností a to následující pěticí:

$$X/Y/n/l/f$$

- X – pravděpodobnostní rozdělení vstupního proudu
- Y – pravděpodobnostní rozdělení dob obsluhy
- n – počet kanálů obsluhy
- l – maximální délka fronty
- f – typ fronty (FIFO, LIFO..)

3 Existující řešení

Nejdříve, než začneme psát vlastní program, by bylo dobré se podívat na možnosti již vytvořených aplikacích. Právě proto se lehce podíváme na různé knihovny a programy, které umožňují aspoň částečně modelovat a simulovat sítě front.

Softwaru poskytujícího práci v oblasti problematiky systému hromadné obsluhy existuje celá řada. Jako dobrý přehled pro začátek nám dobře poslouží seznam[1] různých aplikací a knihoven z této oblasti od Dr. Myrona Hlynky z kanadské univerzity ve Windsoru. Ten se touto problematikou zabývá [10], a proto jeho seznam určitě není špatným místem pro začátek našeho průzkumu, ale jistě neskončíme jen u něho.

Dále si pár z našeho pohledu nejzajímavějších rozebere. Podíváme se na jejich silné a hlavně slabé stránky, abychom se při tvorbě našeho programu vyhnuli stejným chybám, které by vedly k horší výsledné aplikaci.

3.1 Webové aplikace

Pro nás jistě nejzajímavější budou aplikace webové, protože i náš program by měl být takovou aplikací. Zaměříme se tedy hlavně na tento druh softwaru.

Bohužel takových programů není příliš mnoho a většinou poskytují jen omezené možnosti. I tak se pokusíme zmínit aspoň některé z nich a nastíníme jejich funkcionalitu.

3.1.1 Webová aplikace postavená na SIM.JS

Velmi známá a rozšířená JavaScriptová knihovna SIM.JS pro simulování pomocí diskrétních událostí. Právě na téhle knihovně postavili její autoři i webovou aplikaci, která umožňuje tvorbu a následnou simulací sítí front.

Bohužel v současné době poskytuje pouze simulaci M/M/1 sítí, což je dosti limitující. Možnosti nastavení této aplikace nejsou příliš velké a rovněž vytváření modelů je velmi nepohodlné. Na uživatelském rozhraní je vidět, že nepatří k těm nejnovějším.

I když tvůrci slibují další rozšiřovaní, tak vzhledem k faktu, že na celé simulační knihovně se již několik let nepracuje, není moc důvod doufat, že tato webová aplikace bude v budoucnu rozšířena.

[1] `https://web2.uwindsor.ca/math/hlynka/qsoft.html`

V naší aplikaci bychom se určitě měli snažit vytvořit pohodlnější ovládání, protože budování větších modelů by bylo v tomto programu vysoce otravné a zdlouhavé. Dále bychom určitě chtěli nabídnou uživateli větší škálu nastavení a v neposlední řadě možnost exportovat výsledky modelu pro další potřeby.

3.1.2 Ostatní

Najít další podobné webové nástroje je poměrně těžké. Je sice možné najít různé kalkulátory jednotlivých elementárních prvků, jako je kalkulátor[2] pro M/M/K prvky, který nabízí hezké zobrazení výsledků i za pomoci grafů. Dalším takovým kalkulátorem je Rcalclite[3], který nabízí větší množství druhů elementárních prvků jakou jsou M/M/C, M/M/Inf, M/M/C/K a M/M/C/*/M. Dává nám i větší množství nastavení a výsledky prezentuje velmi dobře.

Avšak ani jeden z nástrojů se nedá použít k výpočtu celé sítě, takže pro náš problém jsou nepoužitelné.

3.2 Desktopové aplikace

Webových nástrojů není příliš mnoho, proto se podíváme i na desktopové programy umožňující simulace sítí front. Těch je o poznání více, ale dost často je simulaci systémů hromadné obsluhy pouze malou částí aplikace. Ve většině případů se totiž programy zaměřují na specifickou oblast, kde slouží například k optimalizaci a plánování.

Nástrojů zabývající se čistě teorií sítí není mnoho. Jde hlavně o programy sloužící k výuce na školách. Jeden z takových je QNAT[4] [7]. Jde o nástroj z roku 2000, což je vidět hlavně na jeho uživatelské prostředí. Umožňuje tvorbu sítí a jejích řešení.

Ostatní aplikace zde již moc rozebírat nebudeme, protože nemá smysl se zabývat aplikacemi, které mají primárně jiný účel než modelování a simulování sítí front.

[2] `https://ace-ebert.shinyapps.io/queue_simulator_mmk/`

[3] `https://www.supositorio.com/rcalc/rcalclite.htm`

[4] `https://pdfs.semanticscholar.org/90a7/7e88d14f2fa10429f276cc347a892d5731d4.pdf`

3.3 Knihovny

Poslední pomyslnou kategorií jsou knihovny, které můžeme využít k tvorbě vlastních aplikací. Zde je o poznání více možností. Od knihoven pro MATLAB nebo GNU Octave, které se využívají k rychlému vytvoření modelu a jeho simulování, po knihovny napsané v Javě, C++, C# nebo Pythonu. Podobné knihovny najdeme takřka pro každý trochu rozšířený programovací jazyk a samozřejmě zde najdeme knihovny napsané v JavaScriptu, které nás budou zajímat nejvíce.

3.3.1 JavaScript

Ač se to může zdát překvapivé, tak JavaScript nám nenabízí příliš mnoho možností při výběru knihovny, kterou bychom potřebovali. Asi nejlépe z našeho pohledu vypadá již zmíněná open source knihovna SIM.JS[5].

Poskytuje nástroje pro vytváření simulací založených na diskrétních časových událostech. Můžeme za její pomoci vytvářet objekty, které jsou v systému aktivní a zapouzdřují stav a logiku. Dále objekty konzumující požadavky. Tyto objekty mohou mít další přidanou logiku v podobě front, mezipaměti a podobně. Obsahuje také podporu pro vytváření komunikačních objektů a samozřejmě nesmí chybět nástroje pro získávání statistik z proběhlé simulace. Zároveň má v sobě zakomponovanou knihovnu pro generování náhodných čísel z různých rozděleních pravděpodobnosti. Mezi ně patří například rovnoměrné, exponenciální, normální a další.

Na první pohled se může zdát, že její využití by bylo zcela logické, avšak její využití má několik problémů. První a to nejdůležitější problém je ten, že knihovna je napsaná v čistém JavaScriptu. Její zakomponování do naší TypeScriptové aplikace by bylo sice možné, ale museli bychom buďto porušit zásady, které nám TypeSciprt dává, nebo si napsat vlastní definice pro celou knihovnu, což by nám zabralo nemálo času. Další problém je v tom, že bychom z daleka nevyužili většinu funkcionality knihovny. Takže práce strávená psaním definic by se nám zas tolik nevrátila.

Knihovna již není od roku 2012 vyvíjena a to by mohlo v budoucnu přinést různé komplikace. Pokud bychom se i přesto pro tuto knihovnu rozhodli, její licence LGPL[6] by nám nijak nebránila v jejím libovolném použití.

Díky zmíněným problémům a faktu, že vytvoření vlastní simulace není zas tak náročné, jsme se rozhodli tuto knihovnu raději nevyužít.

[5] `http://simjs.com/`

[6] `https://en.wikipedia.org/wiki/GNU_Lesser_General_Public_License`

3.3.2 Ostatní

Pokud bychom chtěli rychle vytvořit model a zjistit tak o něm základní informace, určitě si vybereme knihovnu pro MATLAB, GNU Octave nebo Python. Pro MATLAB se nabízí například KPC Toolbox[7] a pro Octave existuje podobná varianta v podobě balíčku Qnetworks[8] pro analýzu sítí front. Pro Python se nabízí například Ciw[9], který nám určitě urychlí práci při vytváření takového modelu.

V případě vytváření komplexnější aplikace pravděpodobně použijeme jiný jazyk jako je Java, kde existuje moc hezká knihovna Java Modelling Tools[10]. Pokud bychom chtěli použít nástroj, který vznikl na naší univerzitě, můžeme využít J-SIM[11]. Pro C++ pak můžeme využít například SyDEVS[12]. I pro další jazyky najdeme podobné knihovny a nástroje. Je jen na nás pro jaký jazyk se rozhodneme.

[7] `http://www.cs.wm.edu/MAPQN/kpctoolbox.html`
[8] `http://www.moreno.marzolla.name/software/qnetworks/`
[9] `https://ciw.readthedocs.io`
[10] `http://jmt.sourceforge.net/`
[11] `http://www.kiv.zcu.cz/j-sim/`
[12] `https://autodesk.github.io/sydevs/`

4 Výběr jazyka a frameworku

Úkolem této práce je vytvořit program vhodný pro práci se sítěmi front. Nástroj by měl umožňovat jejich modelování, simulaci a popřípadě i najít analytické řešení.

Jako dobré řešení se nabízí webový nástroj, který umožní všechny požadované funkce. V tomto případě máme dvě základní možnosti, jak může aplikace vypadat. Můžeme vytvořit webovou aplikaci nebo pouze knihovnu, která půjde jednoduše implementovat do libovolné webové stránky. Protože se plánuje využít aplikaci i při výuce předmětu Výkonnost a spolehlivost programových systémů, je dobré předpokládat, že v budoucnu bude snaha o integraci dalších aplikací pokrývající zbývající témata předmětu do jedné ucelené stránky.

Z tohoto důvodu bylo vybráno řešení pomocí JavaScriptové knihovny. To nám zaručí potřebnou flexibilitu a poskytne možnost dalšího rozšíření potažmo integrace s ostatními aplikacemi. JavaScript nám nabízí celou řadu nadstaveb a frameworků pro lepší vývoj a výsledný produkt. Níže si pak rozebereme několik z nich a podíváme se na jejich silné a slabé stránky.

4.1 TypeScript

Jedná se o programovací jazyk, který je nadstavbou pro JavaScript. Byl vytvořen Microsoftem, který se u něj inspiroval Javou a C#. Cílem bylo odstranit nedostatky, které se v JavaScriptu, zejména u větší projektů, objevují[1].

Syntaxe TypeScriptu je mu velmi podobná, ale navíc přidává spoustu užitečných věcí, které usnadňují programátorovi práci. Mezi ně například patří statické datové typy, třídy a dědičnost, rozhraní a generické datové typy.

Takto napsaný kód se pak kompiluje do klasického JavaScriptu. Díky tomu máme možnost odchytit některé chyby už při překladu, popřípadě psaní kódu, pokud využíváme sofistikovanější vývojové prostředí. Na rozdíl od toho, v JavaScriptu se chyba většinou projeví až při běhu programu.

[1] `https://arstechnica.com/?post_type=post&p=151894`

4.2 CoffeeScript

Podobně jako TypeScript, i CoffeeScript je nadstavbou pro JavaScript a umožňuje psát kód, který je následně překládán do klasického JavaScriptu.

Zde, ale podobnost končí, protože jeho záměr je úplně odlišný. Jeho cílem je odstranit z kódu nadbytečné věci a výsledný kód co nejvíce zjednodušit a zkrátit. Odstranil tak například nutnost definovat proměnné klíčovým slovem, středníky a mnoho dalšího. Je velmi inspirován jazyky Python a Ruby.

4.3 NodeJS

Velmi rozšířený framework pro JavaScript, který je určení primárně pro aplikace bežící na serveru.

Klade důraz na minimalizaci režie správy procesů za pomoci řízení událostmi a asynchronními IO operacemi. Dále poskytuje asi nejrozšířenější ekosystém s open source knihovnami.

Jádrem NodeJS je V8 JavaScript engine, který poskytuje vysokou rychlost, protože je napsaný v moderním C++. Díky implementaci částí ze specifikace CommonJS je možné kód spustit nejen v prohlížeči, ale může být i interpretován pomocí REPL.

4.4 React

Jde o JavaScriptovou knihovnu zaměřenou primárně na tvorbu uživatelských rozhraní. O jeho vývoj se stará Facebook, komunika inviduálních vývojářů a několik dalších společností.

Mezi hlavní výhody patří jednosměrný tok dat mezi jednotlivými komponentami, které obdržují data v podobě neměnných hodnot (properties). Dále si komponenty uchovávají svůj stav a jsou schopny svoje data předat dále svým potomkům.

V neposlední řadě je dobré zmínit virtuální DOM, který se pak efektivně aktualizuje do DOMu uživatele. Obsahuje ještě celou řadu užitečných věcí, které usnadní programátorovi práci a zlepší chod výsledné aplikace.

4.5 Zhodnocení

Zmínili jsme jen zlomek existujících nadstaveb a frameworků pro JavaScript, ale i tak je jasné, že výběr těch pravých nebude rozhodně jednoduchý úkol.

Jako dobrý základ je vybrat jednu z nadstaveb rozšiřující klasický JavaScript pro příjemnější psaní kódu a jeho následnou údržbu. Z těchto možností se jako nejlepší varianta jeví TypeScript, který se velmi podobá Javě, což je hlavní jazyk používaný ve většině předmětů univerzity. Navíc poskytnutá typová kontrola umožní velmi dobrou kontrolu chyb a také v budoucnu zjednoduší správu výsledné knihovny.

Další frameworky by sice mohly nabídnou užitečné funkce a vylepšení, ale nezkušenému JavaScript vývojáři by pravděpodobně způsobily více škod než užitku. Z tohoto důvodu je raději nevyužijeme. Dále využijeme NodeJS, jehož součástí je modul NPM. Ten slouží pro stahování a správu JavaScritpových knihoven a modulů.

5 Výběr knihoven

Abychom si nemuseli vše psát od začátku, hodilo by se v práci využít několik knihoven, které nám poskytnout potřebnou funkcionalitu.

Například pro testování aplikace se využití již existující knihovny přímo nabízí, ne-li je skoro nezbytné.

Dále by se nám určitě hodilo zjednodušení vytvoření a manipulace s grafem.

Určitě budeme potřebovat i některé matematické operace. Zde je možnost využít JavaScriptový objekt nebo sáhnout po sofistikovanější knihovně, která umožňuje několik věcí navíc.

5.1 Vykreslení grafu

Pro práci s grafy existuje v JavaScriptu velké množství knihoven. Některé nabízí specifické možnosti určené pro konkrétní potřeby, jiné naopak poskytují komplexní možnosti od různých druhů grafů, map a dalších druhů vizualizace dat. Poslední možností je využití knihovny, která pouze obsahuje základní operace na vykreslování a fyziku vykreslených objektů a samotný graf si vytvářet ručně.

Níže si rozebereme některé zástupce zmíněných kategorií a zvolíme ty, které využijeme.

5.1.1 Cytoscape

Cytoscape[1] je vysoce optimalizovaná JavaScriptová open source knihovna, která obsahuje vše potřebné pro vizualizaci a práci se síťovými grafy. Pro ně poskytuje veškeré funkce a další možnosti pro případnou úpravu podle požadavků uživatele. Zároveň podporuje všechny moderní prohlížeče, a protože je napsaná v čistém JavaScriptu, nebude problém ji integrovat do libovolného frameworku.

Dále jsou vnitřní objekty jednoduše serializovatelné do JSON souborů, a tak zajistit komunikace s případnými dalšími aplikacemi.

Okolo knihovny existuje poměrně silný uživatelská základna a je její vývoj i nadále pokračuje, takže se nemusíme bát, že by knihovna v nejbližší době mohla zastarat a vývojáři ji přestali dodávat potřebnou podporu.

[1] `http://js.cytoscape.org/`

Také je dobré zmínit přehlednou a jasnou dokumentaci, která jistě usnadní použití a úprava knihovny pro vývojáře, kteří se jí rozhodnout použít.

5.1.2 Vis.js

Vis.js[2] je dynamickou vizualizační knihovnou poskytující základní práci s datovými množinami, časovými osami, sítěmi a grafy. Umožňuje jejich vizualizaci, manipulaci a interakci.

Jejím cílem je jednoduché ovládání a použití. Zároveň klade velký důraz na rychlost, aby bylo možno zpracovat velké množství dynamických dat. Jde o open source knihovnu, která se ale bohužel už rok dále nevyvíjí.

5.1.3 D3.js

D3.js[3] je jedna z nejoblíbenějších open source JavaScriptových knihoven pro vizualizaci, manipulaci a další práci s daty, ke které využívá HTML, SVG a CSS. Nabízí nepřeberné množství už vytvořených komponent pro dynamickou vizualizaci, interakci a animaci. Vše je postavené na velmi výkonném jádru knihovny, které je schopno pracovat s rozsáhlými datovými množinami.

Okolo knihovny se rozvinu velká komunita lidí, která vytváří další možné moduly, které je možné spolu s knihovnou využít. I vývoj samotné knihovny nadále pokračuje a nemusíme se tak bát, že by v blízké době mohl vývoj ustat.

5.1.4 Pts.js

Pts.js[4] oproti předchozím knihovnám, které slouží přímo pro grafy a podobné vizualizace, nabízí pouze nástroje pro jejich vytvoření. Pts.js nabízí prostředky pro vytváření grafických prvků a sofistikovanou fyziky vytvořených předmětů. Proto i tato knihovna by se dala využít pro vytvoření sítě front, avšak bylo by nutné vytvořit poměrně pracně jednotlivé komponenty pro vizualizaci.

Tímto bychom získali přímou správu grafických částí naší knihovny a zároveň získali ovládání fyziky jednotlivých komponent a mohly si ji libovolně upravit.

[2] `http://visjs.org/`
[3] `https://d3js.org/`
[4] `https://ptsjs.org/`

I v tomto případě jde o open source knihovnu, která je vysoce optimalizovaná a její vývoj nadále pokračuje. Tedy i zde se nemusíme bát blízkého ukončení podpory ze strany vývojářů.

5.1.5 Zhodnocení

Námi prezentované čtyři knihovny nabízejí různorodé možnosti použití a každá si určitě najde svoje využití.

Jako první můžeme z výběru vyřadit poslední jmenovanou knihovnu Pts.js, protože její použití by bylo zbytečně komplikované a získané výhody bychom v naší práci příliš nevyužili.

Rovněž knihovnu Vis.js by bylo dobré vynechat, protože její vývoj již ustal a jen těžko by se opravovali případné nalezené chyby a v budoucnu by mohl nastat problém s nekompatibilitou staré knihovny s novými prohlížeči a dalšími technologiemi.

Zbývá nám porovnat Cytoscape s D3.js, zde nejsou rozdíly příliš velké, avšak první zmíněná knihovna je přímo určená přesně pro vizualizaci, kterou potřebujeme, a její případná úprava by byla rychlá. Navíc obsahuje velmi přehlednou dokumentaci, která nám práci jen usnadní. Proto se použití Cytoscape zdá jako logická volba.

5.2 Matematika

V naší práci budeme jistě potřebovat různé matematické operace. K tomu můžeme využít standardní JavaScriptový objekt Math, který poskytuje základní matematické operace. Další možností je použít speciální matematickou knihovnu, která může navíc obsahovat užitečné funkce a konstanty. Níže si je trochu rozebereme.

5.2.1 Math objekt

JavaScript obsahuje objekt pro základní matematické operace. Ten obsahuje statické funkce pro řadu matematických úkonů a typické matematické konstanty. Tyto funkce přijímají potřebná data přes parametry a výsledky vrací v podobě návratových hodnot. Bohužel chybí například funkce pro derivování, integrování a pro další složitější matematické operace.

5.2.2 Math.js

Velmi rozšířená open source matematická knihovna pro JavaScript aplikace. Obsahuje rozsáhlou množinu funkcí a konstant pro matematické operace.

Na rozdíl od výše zmíněného matematického objektu umožňuje parsování výrazů, převod jednotek a spojování jednotlivých matematických operací za sebou, což při vhodném použití může usnadnit práci a zpřehlednit výsledný kód.

Dále obsahuje některé operace, které v základním objektu nenajdeme, jako například derivování, integrování a mnoho dalších. Jedná se o knihovnu, která je i nadále rozšiřována a v budoucnu v ní určitě přibudou další možnosti.

5.2.3 Maths.ts

Jedna z mála větších TypeScriptových knihoven pro matematické operace, která je adaptací populární Math.js knihovny. Open source projekt, který bohužel již nevypadá jako aktivní, ale i tak zahrnuje základní operace a konstanty.

V porovnání se zmíněným Math.js se jedná o menší projekt a obsahuje jen zlomek funkcí, které najdeme v předchozí knihovně. I tak je zde dobré zmínit aspoň nějakého zástupce čistě TypeScriptové knihovny pro porovnání.

5.2.4 Zhodnocení

Zde je rozhodování poměrně snadné. TypeScriptová knihovna Maths.ts neobsahuje všechnu potřebnou funkcionalitu a zápis matematických operací pomocí standardního JavaScriptového objektu není příliš pohodlný. Navíc i rozsah možností je velmi omezený, proto bude velmi vhodné využít populární knihovnu Math.js, která nám poskytne veškerou potřebnou podporu v matematických výpočtech.

5.3 Testování

JavaScript nabízí celou řadu knihoven pro testování naší aplikace. Mezi nimi nejsou příliš velké rozdíly, ale i tak si je trochu rozebereme a pokusíme se vybrat tu nejlepší pro nás. Rozhodujícím faktory budou přehledná dokumentace a jednoduché použití, abychom neztratili testováním zbytečně moc času.

5.3.1 Mocha

Velmi populární framework pro testování JavaScriptových aplikací, který běží na NodeJS. Jeho hlavním úkolem je spouštět sériově jednotlivé testy a vytvářet poměrně přehledné reporty s jejich výsledky. Okolo frameworku Mocha je rozsáhlá komunita, která tuto open source knihovnu podporuje a dále vyvíjí.

Její zápis testů je jednoduchý a krátký, ale bohužel je většinou vhodné využít další knihovnu pro lepší zápis složitějších testů. Nejčastěji se kombinuje s knihovnou Chai. Naštěstí existuje nespočet přehledných tutoriálů, které toto spojení popisují a umožňují hladký začátek s testováním.

5.3.2 Chai

Tato knihovna se velmi často používá v kombinaci s knihovnou Mocha. Jejich spojením získáme nástroje pro snadné a rychlé testování naší aplikaci.

Chai poskytuje tři základní příkazy pro zápis testovacích výrazů: assert, expect a should. Jejich použití je velmi podobné. Dokonce je možnost vytvořit si vlastní plugin, ale to v našem případě asi nebude nutné.

Stejně jako předchozí knihovna, i tato je obklopena velkou komunitou a její vývoj nadále pokračuje. I tato knihovna je open source projekt, jako asi většina JavaScriptových knihoven.

5.3.3 Jasmine

Další velmi oblíbená open source knihovna pro testování JavaScript aplikací. Nabízí vše potřebné pro kvalitní testování. Na rozdíl od knihovny Mocha obsahuje vlastní výrazy pro testování, takže není potřeba využívat další knihovnu jakou je například Chai.

Nabízí širokou podporu prohlížečů, ale i technologií. Můžeme ji využít k testování nejen JavaScriptových aplikací, ale také k webovým aplikacím založených na Pythonu nebo Ruby. Zároveň, díky poměrně velké komunitě se nemusíme obávat, že by se přestala v nejbližší době vyvíjet a její podpora ustala.

5.3.4 Tape

Minimalistická knihovna poskytující jen nejdůležitější funkcionalitu k testování JavaScriptových aplikací. Stejně jako předchozí knihovny se jedná open source projekt. Komunita okolo knihovny je sice v porovnání ostatními zmí-

něnými knihovnami o trochu menší, ale i tak se nemusíme obávat podpory v budoucnu.

Obsahuje jen minimum věcí, které jsou nezbytné k psaní jednotkových testů, ale i tak poskytuje dostatečné možnosti ke kvalitnímu otestování naší aplikace. Zároveň její podpora frameworků je velmi dobrá.

5.3.5 Zhodnocení

Jak už bylo zmíněno, všechny možnosti jak testovat JavaScriptové aplikace pomocí jednotkových testů jsou velmi podobné. Výběr tedy spíše záleží na osobních preferencí.

Dále se jistě musí zohlednit přehledná dokumentace a snadná implementace, mezi což můžeme zařadit i kvalitní návody na použití.

Když se podíváme na většinu návodů na jednotkové testování TypeScript aplikaci, zjistíme, že většina těchto návodů využívá kombinaci knihoven Mocha a Chai nebo knihovnu Jasmine. Více přehledné návody se jeví ty pro knihovnu Jasmine, a proto i my využijeme tuto knihovnu pro testování naší výsledné aplikace.

6 Návrh podoby aplikace

Jak již bylo zmíněno jádrem celé aplikace bude knihovna Cytoscape. Knihovna nám poskytuje základní nástroje pro vytvoření grafu, ke kterým budeme muset přidat vlastní implementaci, proto by bylo dobré si nejdříve trochu rozebrat možnosti této knihovny.

Dále se pak podíváme na návrh samotné aplikace, které si rozdělíme na jednotlivé logické části a ty popíšeme. Začneme s propojením Cytoscape knihovny s naší aplikací a modelem sítě front. Potom se podíváme na menu, které umožňuje měnit stavy aplikace a další užitečné akce. Následně na vytváření dialogů, které jsou nezbytné pro nastavení elementů a zobrazení jejich případných statistik. Dále se podíváme na část stojící za generování náhodných čísel podle různých rozděleních pravděpodobnosti. Nesmíme zapomenout na rozebrání samotné simulace, což je klíčová část naší aplikace, a na závěr si rozebereme ostatní možnosti, které ve výsledném programu budeme moci najít.

6.1 Cytoscape

Jak už bylo dříve zmíněno, knihovna nabízí velkou škálu možností a nástrojů [4]. Bylo by zbytečné zde všechny vyjmenovávat a rozebírat. Přece jen detaily si může každý najít v přehledné dokumentaci[1] knihovny. Z tohoto důvodů si zde rozebereme jen části, které nás zajímají a mají potenciál k využití ve výsledné aplikaci.

Jde hlavně o funkce k vytváření elementů, možnosti operací s elementy jako je vyhledávání a podobně, možnosti CSS v knihovně, události, které knihovna podporuje, a funkce k přizpůsobení rozmístění elementů.

6.1.1 Vytváření grafu

K vytvoření grafu se dají použít základní funkce `cy.add()` a `cy.remove()`. Díky těmto funkcím můžeme přidávat a odebírat jednotlivé uzle a hrany.

Parametrem funkce pro přidání je definice uzlu nebo hrady. Ta je zapsaná pomocí JSONu. V definici je možné uvést skupinu, data a pozici elementu. Skupiny slouží pro určení typu elementu, tedy jestli jde o uzel nebo hranu. V datech můžeme najít velké množství informacích o elementu. Pro nás nejdůležitější je nastavení id elementu, dále je zde možné například definovat

[1] `http://js.cytoscape.org`

jméno elementu, jeho velikost a podobně. Elementům je rovněž možno přiřadit různé třídy, stejně jako klasickým HTML elementům.

Funkce pro odstranění elementu požaduje přímo element nebo jeho selektor. Element můžeme získat právě pomocí selektoru v parametru voláním hned několika funkcí `cy.$()`, `cy.elements()`, `cy.nodes()`, `cy.edges()` a `cy.filters()`.

6.1.2 Selektory

Selektory fungují podobně jako v JavaScriptu nad HTML stránkou. Zde jen vyhledáváme nad skupinou elementů knihovny a to podle skupiny neboli typu, id a třídy elementů. Skupiny můžou být různé. Od všech uzlů, hran nebo elementů, až po skupiny získané z předešlých vyhledávání nebo jiných akcí. K získání zmiňovaných skupin použijeme funkce, které byly zmíněny v předchozí kapitole.

Můžeme zde vyhledávat i podle stavů elementů. Například to může být styl elementů, jejích vlastnosti a podobně. Užitečné můžou být vlastnosti typu rodič (`:parent`) nebo potomek (`:child`). Možností je zde opravdu hodně. Další informace nalezneme v dokumentaci.

6.1.3 CSS elementů

Stejně jako jsme zvyklí upravovat vzhled HTML elementů za pomoci CSS, tak i elementy v Cytoscape knihovně můžeme podobným způsobem měnit do požadované podoby. Možnosti CSS v Cytoscape jsou trochu omezené a jejich přesné změní můžeme zjistit v již zmíněné dokumentaci v kapitole 6.1.

CSS má v knihovně dvě podoby. První z nich je nastavení definované elementům a třídám. Toho nastavení je globální pro všechny elementy daného typu nebo třídy. Na rozdíl od klasického CSS zde nefunguje klíčové slovo `!important`, takže toto nastavení nepřekrývá nastavení u jednotlivých elementů. K nastavení CSS elementů se využívá funkci `style()`, která se volá na příslušným elementem nebo skupinou elementů. Parametrem této funkce je styl, který chceme aplikovat, zapsaný pomocí JSONu.

6.1.4 Události

Na Cytoscape elementy můžeme vázat posluchače událostí stejně jako to děláme u HTML elementů. Funguje to velmi podobně. K nastavení použijeme funkci `on()`. První parametr určitě druh události. Zde je na výběr z velkého množství událostí od klasických `click` po komplexnější typu `cxttap`.

Všechny možnosti pak nalezneme v dokumentaci knihovny. Druhým parametrem je selektor, podle kterého můžeme filtrovat elementy, nad kterými je funkce volaná. Posluchač se pak aplikuje pouze na vyfiltrované elementy. Tento parametr je nepovinný. Poslední parametrem je samotná funkce, která se při události má vykonat.

Další možnosti jsou velmi podobné tomu co známe z klasického JavaScriptu, proto nemá smysl je tu dlouze rozepisovat. Pro bližší detaily se opět můžeme podívat do dokumentace knihovny.

6.1.5 Rozložení elementů

Další užitečnou funkcionalitou knihovny je možnost stanovit si rozložení elementů. Samotná knihovna poskytuje bezpočet již vytvořených rozložení jako je například mřížka, kruh nebo pružina. Každé rozložení má pak velké množství vlastního nastavení.

Umístění elementů do požadované struktury zajistíme pomocí zavolání funkce `layout()` nad příslušnými elementy. V parametru funkce definujeme příslušné rozložení a jeho nastavení opět pomocí JSONu. Následně zavoláním funkce `run()`, nad objektem získaným z předešlé funkce, způsobíme spuštění změny rozložení elementů podle definovaného rozložení.

Cytospace podporuje vytváření vlastních rozloženích, které aplikujeme stejným způsobem jako již ty předem definovaná.

Knihovna také umožňuje aplikovat rozložení jen na část elementů a zajistit tak některým elementům neměnnou pozici.

6.1.6 Ostatní

Cytoscape obsahuje ohromné množství funkcí, které by aplikaci mohli v příštích verzí ještě více vylepšit. Prozatím je nebudeme používat, avšak určitě by bylo dobré, aby se při dalším rozšiřovaní výsledné aplikace důkladně prošly existující možnosti této povedené knihovny.

6.2 Model systému hromadné obsluhy

Nastínili jsme si možnosti knihovny Cytoscape, proto je teď čas na rozebrání toho, jak knihovnu využijeme a vytvoříme pomocí ní model sítě front. Vytvoření modelu si rozdělíme na jednotlivé logické části a ty níže popíšeme.

Nejdříve se podíváme na vytvoření samotného modelu systému hromadné obsluhy a následně si rozebereme možnosti konfigurace jednotlivých prvků systému.

6.2.1 Vytvoření modelu

Základem aplikace bude samotný model vytvořený pomocí knihovny Cytoscape. To jak se s knihovnou pracuje jsme již rozebrali dříve, proto tuto část v této kapitole opomeneme.

Systém hromadné obsluhy se v podstatě skládá ze tří různých uzlů a jejich projení. Tyto uzle musíme v aplikaci nějak odlišit. K tomu se nabízí využít možnosti každému elementu v Cytoscape přiřadit třídu. Tato třída pak může mít vlastní CSS a tím docílíme toho, že každý zmíněný uzel bude vypadat odlišně. Zároveň tím získáme jednoduchý výběr uzlů jednoho druhu, když využijeme selektory knihovny pro hledání podle třídy elementu.

Každý element typu uzel musí mít svůj unikátní identifikátor (ID), které budeme samy generovat pomocí kombinace písmena 'n', které vzniklo z anglického překladu slova uzel, tedy `node` , a pořadového čísla. Tento identifikátor se zobrazí u každého takového elementu.

Vstupní uzel

První z nich je uzel generující požadavky. Pro zjednodušení vizualizace bude součástí uzlu také křižovatka určující kudy poputují požadavky, které uzel generuje. Pravděpodobnost jednotlivých cest je uváděna v procentech a jejich součet musí být roven stu. Logicky z těchto uzlů mohou vést hrany pouze ven, tedy žádná hrana nesmí vést do vstupního uzlu. Výstupních hran může být libovolná počet.

Tomuto uzlu přiřadíme třídu `input-node`. Uzel bude mít podobu pětiúhelníku se světle šedou barvou.

Uzel obsluhy

Nejdůležitějším uzel modelu je jistě uzel, který obstarává obsluhu požadavků. Stejně jako uzel vstupní, tak i tento uzel bude obsahovat křižovatku. Pravidla pro ní jsou rovněž stejná. Z uzlu i do něj může vést libovolný počet hran.

Uzel nebude mít specifickou třídu, ale pouze to bude element typu uzel. Bude mít šedou barvu a tvarem bude kruh.

Výstupní uzel

Posledním z jmenovaných tří uzlů modelu je ten, který obstarává výstup požadavků ze systému. Půjde o nejjednodušší uzel, protože hrany do něj budou moci pouze vstupovat.

Tento uzel získá třídu `output-node`. Díky tomu bude tmavě šedý a bude mít podobu čtverce.

Hrany

Propojení mezi jednotlivými uzly zajišťují elementy typu hrana. Tyto elementy můžeme jednoduše přidávat pomocí již zmíněné funkce z knihovny Cytoscape.

6.2.2 Konfigurace elementů

Teď máme vytvořenou síť, ale ta by nám byla k ničemu, pokud bychom neumožnili nastavení jednotlivých elementů sítě. Tato konfigurace nebude sice moc rozsáhlá, ale funkčnost celého systému je zcela nezbytná.

Níže si pouze rozebereme jaké možnosti budou u jednotlivých elementů, ale samotný proces nastavení si probereme až později.

Vstupní uzel

Ve vstupních bodech sítě musí jít nastavit rozdělení pravděpodobnosti příchozích požadavků a to včetně všech jeho parametrů. Tyto parametry pak závisí na vybraném rozdělením pravděpodobnosti. To by tedy mělo jít nastavit pomocí výběru z listu možných rozdělení a následně by se vypsaly jeho parametrů k nastavení.

Dále je třeba určit jestli při následné simulaci celého modelu budeme ukládat detailní statistická data pro konfigurovaný uzel. Zde postačí klasické zaškrtávací tlačítko.

Poslední důležitou částí je nastavení křižovatky. Pro každou výstupní hranu je umožněno nastavit pravděpodobnost přechodu z konfigurovatelného uzlu po této hraně. V aplikaci tuto pravděpodobnost budeme určovat v procentech.

K nastavení pravděpodobnosti by se hodil posuvník s hodnotami nula až sto. Navíc by bylo vhodné dovolit uživateli zadat přímo pravděpodobnost, proto k tomuto posuvníku přidáme políčko, kam půjde zadat toto číslo. Tyto dva elementy bychom zároveň měli propojit, aby se hodnota při nastavení jednoho měnila i v druhém a naopak.

Zde nastává trochu komplikace, protože musíme navíc zajistit, aby součet všech pravděpodobností byl jedna, v našem případě, protože používáme procenta, by to mělo být sto. Aby si uživatel nemusel složitě počítat součet hran, bylo by dobré přidat tlačítko, které samo poměrově upraví hodnoty

pravděpodobností jednotlivých hran, aby jejich součet odpovídal požadované celkové hodnotě.

V neposlední řadě bychom měli umožnit tento vstupní uzel smazat.

Uzel obsluhy

Nejsložitější prvek celého systému hromadné obsluhy. Jedná se o uzly zajišťující obsluhu požadavků. Zpracování požadavků odpovídá některému z rozděleních pravděpodobnosti. Tedy tato část konfigurace by tu měla být stejná jako u vstupního uzlu. Jen v pozadí nastavujeme obsluhu požadavků namísto jejich příchodu.

Jak u napovídá název sítě front, měli bychom mít v modelu fronty. Jejich místo je právě u obslužných uzlů. Fronty mohou být konečné nebo nekonečné. Pro určení maximální délky fronty využijeme vstupní pole, z kterého získáme celé číslo.

Pokud je číslo záporné, předpokládáme, že je fronta nekonečná, zatímco nula nebo kladné číslo nám určuje maximální délku fronty. Pokud fronta dosáhne této délky, požadavky musí být zahazovány.

Následně, stejně jako u předchozího uzlu, nastavíme jestli chceme ukládat detailní statistické informace.

Stejně jako vstupní uzel, i uzel obsluhy obsahuje výstup v podobě křižovatky výstupních hran. Tato křižovatka bude stejná jako u vstupního uzlu, proto zde nebudeme vypisovat podrobnosti.

Také tento uzel může uživatel smazat.

Výstupní uzel

Výstupní uzel sám o sobě nebudeme mít žádní nastavení. Jediné co bude možné určit, stejně jako u předchozích uzlů, je jestli budeme ukládat detailní statistické data při simulaci.

Hrany

Hrany půjdou pouze smazat, to bude jediná možná operace, kterou konfigurační okno dovolí.

6.3 Menu

Nedílnou součástí aplikace by měl být panel pro její ovládání. V našem případě půjde o menu umístěné v horní části uživatelského okna programu.

Abychom zajistili požadovanou funkcionalitu, která již byla zmíněna, je nutné vytvořit systém ovládání. Toto ovládání by mělo být hlavně pohodlné pro uživatele.

Jako vhodný způsob, jak dosáhnout tohoto cíle, se jeví využití přepínatelných módů, v kterých se aplikace může nacházet. V každém módu by kliknutí v aplikačním okně mělo různou funkcionalitu. Byl by zde mód pro prohlížení, kde by kliknutí nemělo žádnou speciální funkcionalitu, dále pro přidání vstupního, obslužného a výstupního uzlu a nakonec pro přidání hrany.

Následuje dvojice tlačítek pro import a export modelu. Ty slouží ke stažení nebo nahrání JSON souboru s koncovkou `.qtjs`. V tomto souboru nalezneme veškeré informace sloužící k vytvoření uloženého modelu včetně veškerého nastavení.

Dalším prvkem menu by mělo být tlačítko pro aplikaci definovaného rozložení. Pro naši aplikaci jsme vybrali pružinové rozložení. Poskytuje nejpřehlednější rozmístění elementů v porovnání s ostatními dostupnými rozděleními, o kterých jsme se již zmínili v textu o Cytoscape knihovně 6.1.5. Je založený na využívání simulace fyziky pro rozložení grafu. Detailní informace o jeho algoritmu můžeme nalézt v článku[2] z roku 2009.

Nesmíme zapomenou na podstatu celé aplikace. Tím je simulace systému hromadné obsluhy. Nastavení simulace a její ovládací prvky by se pohodlně měly vměstnat do menu. Jedná se o nastavení simulačního času, semínko pro generátory, spuštění simulace, export výsledků simulace a odstranění simulačních výsledků.

6.4 Dialogy

V aplikaci budeme jistě potřebovat dialogová okna. Ty by měly sloužit k nastavení požadovaných vlastností elementů, jako potvrzovací okno a k zobrazení užitečných dat jako jsou například výsledky ze simulace. Z toho plyne, že budeme potřebovat tři druhy dialogů. Ty by se měly skládat z nadpisu, obsahu dialogu a jednotlivých tlačítek s požadovanou funkcionalitou. Také potřebujeme zajistit správné umístění otevřeného dialogu v aplikaci. Vše co jsme zde jmenovali si trochu více rozebereme níže.

[2] `https://dl.acm.org/citation.cfm?id=1498047`

6.4.1 Typy dialogů

Jak už bylo zmíněno, v našem programu budeme potřebovat tři druhy dialogů. Ty budou využívány pro různé potřeby, ale jejich základ bude stejný pro všechny druhy oken. Níže si všechny druhy dialogů podrobně popíšeme.

Konfigurační dialog

První z nich je konfigurační okno elementů. To by mělo zároveň po jeho otevření zamezit jakékoliv jiné manipulaci s aplikací než tou v zobrazeném okně. To můžeme zajistit zobrazením lehce průhledného panelu přes celou aplikaci, nad který umístíme dialogové okno.

Potvrzovací dialog

Dalším oknem je potvrzovací dialog. Ten bude nést v nadpisu otázku a na panelu s možnostmi nalezneme pouze potvrzovací a zamítací tlačítka. Stejně jako předchozí, i tento dialog by měl znemožnit uživateli jakékoliv jiné akce s aplikací než potvrzení nebo odmítnutí dotazované akce.

Informační dialog

Posledním typem dialogů je okno pro zobrazení informacích. V něm budeme moc zobrazit výsledné data ze simulace.

6.4.2 Umístění dialogů

Nesmíme opomenout důležitou otázku kde budeme okno zobrazovat. Zde máme dva možné scénáře. První z nich je zobrazit dialog vždy na stejném místě. Pro tuto možnost se nabízí střed aplikace. Druhou možností je vytvořit dialog vždy v místě, kde uživatel provedl akci, která vedla k otevření konkrétního okna.

Ač je jasné, že druhá varianta bude složitější na implementaci a zároveň přinese komplikace s udržením dialogu v aplikaci, která ne vždy musí zabírat celou stránku, rozhodli jsme se vybrat tuto uživatelsky příjemnější variantu.

Okno se tedy otevře v místě, kde uživatel spustil akci, která otevřela dialog. Pozice tohoto okna se musí zároveň korigovat tak, aby okno bylo stále v aplikačním panelu.

6.5 Generování náhodných čísel

Jak už jsme zmínili, v aplikaci budeme potřebovat generovat náhodná čísla z různých rozděleních pravděpodobnosti. Ty budou sloužit hlavně při simulaci u vstupních a obslužných uzlů.

V první verzi aplikace se bude jednat pouze o tři generátory. Přesněji půjde o rovnoměrné, exponenciální a Gaussovo rozdělení. Bylo by vhodné si jednotlivá rozdělení aspoň částečně rozebrat, abychom věděli, jak při implementaci postupovat.

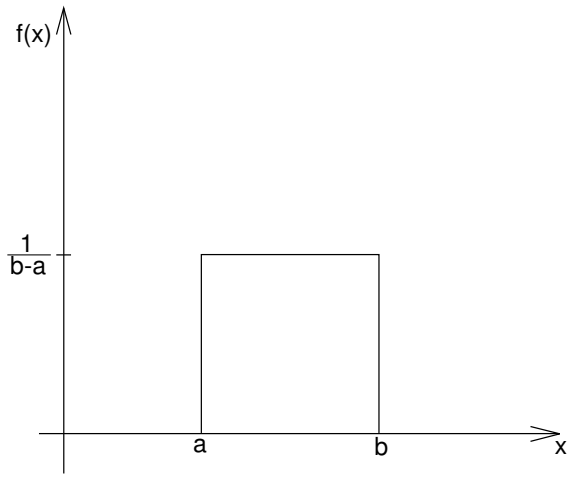
6.5.1 Rovnoměrné rozdělení

Jde o základní rozdělení pravděpodobnosti. V našem případě nás zajímá pouze spojitá varianta tohoto rozdělení.

Stanovujeme u něj minimální a maximální hodnotu rozdělení, pro náhodná čísla. Vzniká nám tak interval, ve kterém je hustota pravděpodobnosti konstantní. To můžeme vidět na obrázku 6.1, kde zmiňované minimum a maximum je označeno písmeny a a b. Dále by nás mohla zajímat střední hodnota $E(X)$ a rozptyl $D(X)$ rozdělení:

$$E(X) = \frac{a+b}{2} \tag{6.1}$$

$$D(X) = \frac{(b-a)^2}{12} \tag{6.2}$$



Obrázek 6.1: Hustota rovnoměrného rozdělení pravděpodobnosti
od autora Pajs, CC BY-SA 3.0

6.5.2 Exponenciální rozdělení

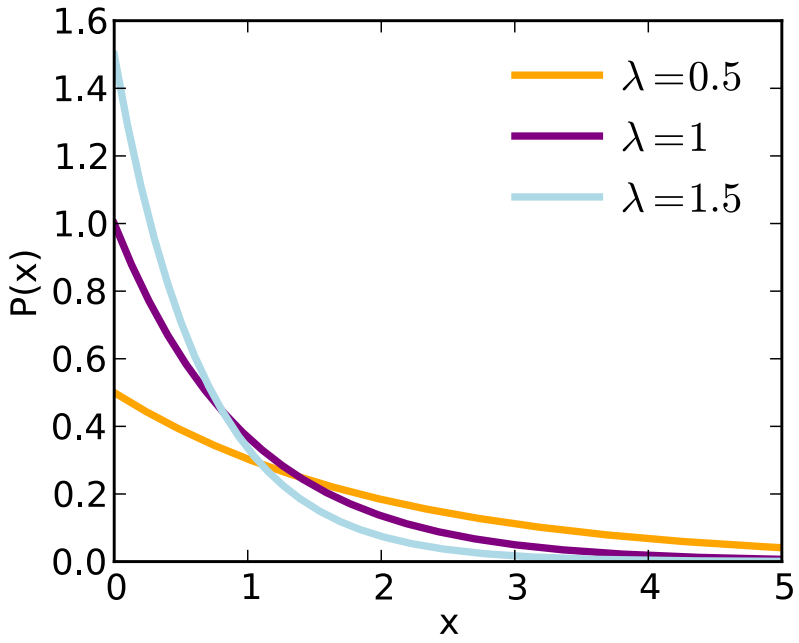
Další na řadě je exponenciální rozdělení. U něho existuje na rozdíl od předešlého pouze spojitá varianta. Využívá se hojně v různých simulací, protože dobře reprodukuje délky intervalů mezi náhodně se vyskytujícími událostmi, které se dají popsat Poissonovo rozdělením.

Exponenciální rozdělení disponuje pouze jedním parametrem λ, který můžeme brát jako průměrnou dobu mezi událostmi, tedy například průměrnou dobu obsluhy nebo průměrnou dobu mezi příchozími požadavky. Tento parametr stanovuje podobu celého rozdělení a musí být větší než nula. Tedy má platit tento vztah $\lambda > 0$.

Na obrázku 6.2 můžeme vidět jeho hustotu pravděpodobnosti pro různé hodnoty parametru lambda. Stejně jako u rovnoměrného rozdělení i u exponenciálního můžeme stanovit jednoduchými vzorci střední hodnotu $E(X)$ a rozptyl $D(X)$ rozdělení:

$$E(X) = \frac{1}{\lambda} \tag{6.3}$$

$$D(X) = \frac{1}{\lambda^2} \tag{6.4}$$



Obrázek 6.2: Hustota exponenciálního rozdělení pravděpodobnosti
od autora Skbkekas, CC BY-SA 3.0

6.5.3 Gaussovo rozdělení

Poslední je Gaussovo rozdělení, které je také známé pod označením normální rozdělení. Existuje pouze ve spojité variantě. Využití má stejně jako předchozí rozdělení velmi široké, například hezky popisuje rozdělení výšky lidí v populaci a podobně.

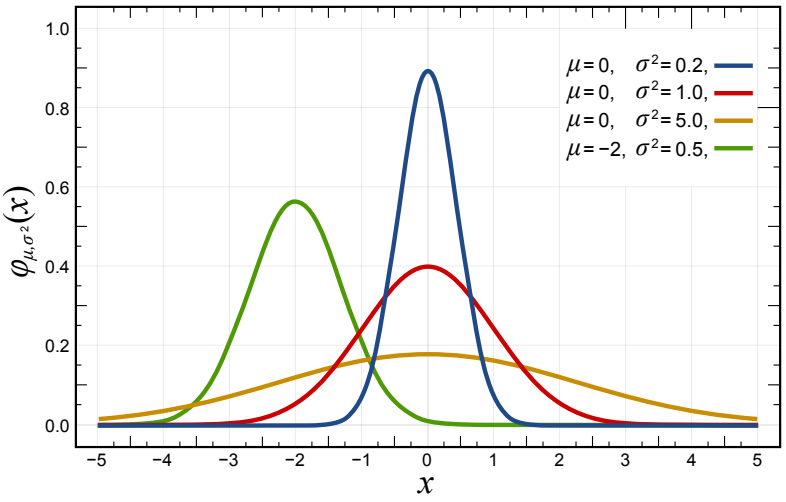
Podobu normálního rozdělení určují dva parametry. Prvním z nich je μ, které odpovídá střední hodnotě. Druhým parametrem je σ^2, která se rovná rozptylu rozdělení. Pro posledně jmenovaný parametr musí navíc platit vztah $\sigma^2 > 0$

Podobu hustoty pravděpodobnosti tohoto rozdělení můžeme vidět na obrázku 6.3. Tato funkce tvoří Gaussovo křivku, proto se někdy používá toto označení i pro celé rozdělení.

I zde nás bude zajímat jeho střední hodnota $E(X)$ a rozptyl $D(X)$. Určení těchto hodnot bude velmi jednoduché, protože přímo odpovídají parametrů rozdělení:

$$E(X) = \mu \tag{6.5}$$

$$D(X) = \sigma^2 \tag{6.6}$$



Obrázek 6.3: Hustota normálního rozdělení pravděpodobnosti od autora Inductiveload, CC BY-SA 3.0

6.6 Simulace

Zásadní částí aplikace bude simulace systému hromadné obsluhy. Ta je nezbytnou součástí pro analýzu modelu, protože možnosti analytického řešení sítí front jsou jen omezené a pro složitější modely jsou tyto způsoby velmi obtížné a v některých případech dost možná nemožné.

Samotnému způsobu simulace se budeme více věnovat v části o implementaci. Zde si spíše rozebereme čeho bychom chtěli dosáhnout, jak výsledky simulace prezentovat a jak dál s nimi pracovat.

Konkrétně si popíšeme způsoby zobrazení výsledků simulace systému. Dále se podíváme jak můžeme dosažené výsledky exportovat z aplikace, abychom je mohli využít například v jiných aplikacích.

6.6.1 Zobrazení výsledků

Nezbytnou součástí simulace je zobrazení výsledků. Může se jednat o vizualizaci přímo v modelu nebo mimo něj. Zde máme v zásadě tři způsoby jak k tomuto problému přistoupit.

Můžeme model vizuálně změnit, aby nám reprezentoval výsledky simulace. Například může jít o obarvení a přidání různých popisků reprezentující dosažené výpočty simulace. Druhou možností, která přece jen nabízí větší škálu využití, je zobrazení informací v okně u konkrétního elementu simulace. Jako poslední způsob můžeme označit vypsání výsledných dat mimo model.

Pro naši aplikaci budou nejdůležitější první dvě zmiňované možnosti, protože nejlépe poslouží k analýze částí modelu a ulehčí případné změny, aby systém splňoval požadované vlastnosti.

Obarvení a popisky grafu

Asi nejlepší způsob, jak rychle a efektně zobrazit výsledky simulace a přiblížit tak model uživateli, je obarvit model a přidat k jednotlivým elementům grafu popisky.

Obarvení můžeme mít dvojího druhu. První zaměřené na jednotlivé obslužné uzly systému a druhé pro jednotlivé hrany grafu. Celé obarvení bude pomocí různých odstínů červené barvy. Pro malé hodnoty použijeme světlé odstíny a pro větší pak naopak tmavší.

Jako první se podíváme na obarvení uzlů modelu. Zde máme několik možností, které bychom mohli barvou reprezentovat. Jako dobrý faktor ukazující zatížení uzlu bychom mohli využít maximální délku fronty. Podle této

délky snadno poznáme, který uzel byl aspoň po krátkou dobu silně zatížen požadavky a tedy se u něj tvořili dlouhé fronty.

Díky tomuto rozhodnutí budeme obarvovat pouze obslužné uzly, protože pouze u nich máme fronty. Vstupy a výstupy z modelu zůstanou v původní barvě a v grafu systému tak o to více vyniknou.

Další zmíněnou věcí je obarvení hran. Zde je opět více možností, ale jako nejočekávanější by z pohledu uživatele bylo obarvení podle zatížení. Tedy o tom kolik požadavků po hraně přešlo.

Pro větší zdůraznění navíc hranám nastavíme různou šířku odpovídající zmíněnému zatížení. Pro větší zatížení bude hrana silnější a tmavší, naopak u slabšího bude tenčí a světlejší.

Díky těmto krokům bude z modelu na první pohled poznat jeho chování a možnosti pro jeho optimalizaci. I vizuálně bude odsimulovaný model působit velmi dobře a přehledně, díky použití pouze odstínů jedné barvy.

Zobrazení výsledků

Obarvení modelu můžeme ještě vylepšit přidáním popisků. Tím dosáhneme ještě větší přehlednosti odsimulovaného systému. Díky tomu bude velmi snadné a rychlé určit slabá místa modelu a ty popřípadě upravit a vylepšit, aby zátěž zvládala lépe.

Stejně jako u obarvení budeme mít popisky dvojího typu, ač budou mít tentokrát stejný význam.

Začneme popiskami uzlů. Ty budou zobrazovat kolik požadavků uzel zpracoval. Tento popisek bude u každého uzlu. Popisek bude i u vstupů a výstupů z modelu, protože i zde mají svůj význam.

U obslužných uzlů navíc přidáme informaci o počtu zahozených požadavků. Toto číslo bude umístěno za číslem určující počet obsloužených požadavků a navíc bude označeno pomocí hranatých závorek. Tuto přidanou informaci, z důvodu lepší přehlednosti, přidáme pouze pokud byly nějaké požadavky zahozeny.

Zbývá nám zmínit popisky jednotlivých hran. Ty budou velmi jednoduché, protože budou určovat pouze počet prošlých požadavků hranou. Z důvodu zvýšení přehlednosti budou tyto popisky hran lehce průhledné u hran s nejmenšími hodnotami.

6.6.2 Export výsledků

Možností co z výsledků exportovat a hlavně jakým způsobem data zapsat je hned několik. Je jen na nás, pro který se rozhodneme. Těžko můžeme určit

jak s exportovanými daty bude uživatel chtít naložit, proto se budeme snažit zajistit co nejsnazší a zároveň nejširší využití těchto dat v dalších aplikacích.

Důležitou otázkou je výstupní formát. V současné době jsou asi nejtradičnějšími formáty `CSV` a `JSON`. Pro přímou práci s výslednými daty se lépe jeví formát `CSV`, protože tento soubor snadno otevřeme například v aplikaci Microsoft Excel. Můžeme pak velmi rychle vytvořit grafy, které nám poskytnou další zajímavé informace o systému, nebo data dále zpracovat jinými způsoby. V budoucnu můžeme aplikaci rozšířit i o další formáty.

Exportovat budeme veškerá data, která v simulaci získáme. Tyto data budeme poskytovat v syrové podobě bez jakéhokoliv zpracování, abychom z nich neodstranili případné informace, které by uživatele mohli zajímat.

7 Implementace aplikace

Další velmi důležitou částí této práce je popsání samotné implementace aplikace. Je důležité si implementaci rozdělit do jednotlivých logicky ucelených částí. Ty si pak samostatně popíšeme a detailně rozebereme.

Jak již bylo zmíněno k implementaci jsme si vybrali JavaScript, konkrétně framework TypeScript. Ten přidává možnost datových typů a základní podporu pro objektové programování. To se projeví i v popisu naší implementace, kde možností TypeScriptu hojně využijeme.

Nejdříve se podíváme na základ celé aplikace. Dále na jednotlivé části sloužící k vytvoření menu a dialogů. Důležitou součástí programu bude jistě nastavení jednotlivých uzlů, které si také rozebereme. Následně se podíváme na implementaci pro generátory z různých rozděleních pravděpodobnosti a zakončíme to velmi důležitým rozborem simulace.

7.1 Základ aplikace

Základem celé aplikace je třída `App`. Ta spojuje knihovnu Cytoscape, která slouží k vytváření grafu, a ostatní části aplikace. Mezi ně patří menu, které obsahuje všechny ovládací prvky programu, plátno, kde je prostor pro vytváření modelu systému, dialog manager, který slouží pro vyvolání různých druhů oken, a samotné nastavení systému hromadné obsluhy.

Obsahuje funkci pro vytvoření celé aplikace. V ní voláme další funkce, díky kterým připravíme vše potřebné od menu, plátna a dalších potřebných částí, po nastavení všech důležitých posluchačů a akcí pro menu a elementy z knihovny Cytoscape.

V budoucnu by zde mohly přibýt funkce k ovládání aplikace 'z venku'. Jinak řečeno, aby aplikace šla lépe integrovat s dalšími části výsledné stránky. V současné době se do určeného okna na stránce vytvoří námi popisovaná aplikace.

7.2 Menu

Většina implementace menu se nachází ve stejnojmenné třídě `Menu`. Ta poskytuje funkce k vytvoření celého menu aplikace. Výsledkem je pak HTML element, který se v hlavní části aplikace vloží do příslušného místa.

Samotná třída `Menu` zajišťuje přepínání jednotlivých stavů aplikace. Jde

o stavy prohlížení, vložení vstupního uzlu, vložení uzlu obsluhy, vložení výstupního uzlu a vložení hrany. Na tento stav jsou pak navázány různé akce v hlavní třídě `App`.

Tento stav je uložený pomocí výčtu neboli anglicky `enumu`. To je jedna s přidaných možností Typescriptu, kterou známe například s Javy, C# nebo podobných programovacích jazyků.

Dále nastavuje zpětné volání funkcí poskytnutých opět z hlavní třídy. Pomocí těchto zpětných volání zajistíme například rozmístění grafu a ovládání simulace spolu s prací s jejími výsledky.

7.3 Dialogy

O trochu složitější bude implementace dialogů. I když se to nemusí na první pohled zdát, máme zde hned několik problémů, které bude potřeba vyřešit. První z nich je znemožněných akcí mimo dialog, pokud je tento dialog otevřen. Druhým problémem je samotné umístění dialogového okna.

7.3.1 Dialog manager

Nejdůležitější částí je jistě samotný dialogový manager, který umožňuje manipulaci s jednotlivými typy oken. Jeho implementaci nalezneme ve stejnojmenné třídě `DialogManager`. Zde nalezneme pro každý typ dialogu funkce, které slouží pro otevření a případně uzavření okna.

7.3.2 Zamezení akcí

Důležitou součástí dialogů je i zamezení akcí mimo otevřené okno. Toho jde jistě dosáhnout různými způsoby, ale my využijeme asi nejvíce použitelný přístup, kdy překryjeme zbytek aplikace HTML elementem s určitou průhledností. Díky tomu dosáhneme hezkého vizuálního efektu, který jednak zamezí akcím mimo dialog a zároveň nám zvýrazní otevřené okno.

7.3.3 Umístění okna

Dalším problémem k řešení je umístění dialogů při jejich otevření. Jistě bychom mohli zvolit nejsnazší řešení a dialog vytvořili vždy uprostřed celé aplikace, popřípadě na jiném stálém místě. Ač by bylo toto řešení akceptovatelné, určitě by nebylo uživatelsky příliš příjemné.

My proto zvolíme trochu komplikovanější řešení, které spočívá v tom, že okno vytvoříme v místě, kde se stala událost, která vedla k jeho otevření. V naší aplikaci budeme mít dva druhy umístění.

První z nich bude pro dialogy sloužící k nastavení uzlů a potvrzovací okna. Ty se otevřou přímo v místě vyvolání události. Zde jen zkontrolujeme jestli se výsledné okno vejde do prostoru aplikace, pokud ne, dialog zobrazíme posunutý o jeho velikost buďto v ose horizontální nebo vertikální. To záleží v jaké ose nám okno vylézá z aplikace.

Druhou možností je zobrazení statistik ze simulace. Zde výsledné okno umístíme k uzlu, kterého se výsledky týkají. I zde musíme řešit, aby se vytvořené okno vešlo do aplikace a přístup bude velmi podobný jako v předchozím případě, jen okno posuneme navíc o velikost uzle, aby okno nepřekrývalo v žádném místě uzel.

7.3.4 Typy dialogů

V aplikaci, jak už jsme zmínili, budeme potřebovat tři druhy dialogů. Ty máme rozděleny do jednotlivých tříd, kde máme jedno hlavní, od které další typy dědí některé vlastnosti. Níže si všechny třídy důkladně popíšeme.

Hlavní dialog

Implementace hlavního dialogu se nachází ve třídě `Dialog`. Jedná se o základní typ okna. Tento dialog se používá pro nastavení jednotlivých elementů modelu.

Poskytuje funkce pro vytvoření samotného dialogu v podobě HTML elementu, který obsahuje všechny potřebné prvky, pro nastavení panelu, který zakrývá zbytek aplikace, a funkce pro otevření samotného okna, v které řešíme umístění dialogu a nastavení všech potřebných akci pro jednotlivá tlačítka.

Potvrzovací dialog

Další typ dialogu najdeme ve třídě `ConfirmDialog`. Jak z názvu napovídá jedná se o potvrzovací okno. To nabídne otázku, na kterou bude možné odpovědět pouze ano nebo ne.

Tato třída dědí od hlavní třídy `Dialog` a pouze překrývá funkci pro vytvoření okna, protože HTML struktura dialogu musí být odlišná.

Okno pro statistiky

Poslední typ okna nalezneme ve třídě `StatisticsDialog`. Tento dialog slouží pouze pro zobrazování informací a to přesněji statistických dat ze simulace.

Třída, stejně jako předchozím případě, dědí od hlavní třídy `Dialog`. Překrývá funkci pro otevření dialogu, protože pozice tohoto okna je trochu odlišná oproti klasickému dialogu. Dále překrývá funkci pro vytvoření dialogu, protože i toto okno má rozdílnou HTML strukturu.

Navíc přidává funkce pro schování celého okno, abychom ho mohli kdykoliv schovat. Samotný dialog se bude zobrazovat v případě, že máme výsledky simulace a uživatel najede myší na patřičný uzel. Abychom mohli okno následně po posunutí myši mimo uzel schovat, potřebujeme výše zmíněnou funkci.

7.4 Nastavení uzlů

Pochopitelně nejsložitější částí bude nastavení jednotlivých uzlů. Zde jsou požadované vlastnosti velmi rozsáhle a stále musíme myslet na další možnou rozšiřitelnost.

Implementaci můžeme rozdělit na třídy pro jednotlivé typy uzlů a pomocné třídy, které v těchto třídách využíváme. Níže se podíváme na všechny zmíněné třídy a popíšeme si jejich spojení a jejich případné možnosti.

7.4.1 Typy uzlů

Protože v aplikaci máme tři typy uzlů, logicky i pro jejich nastavení budeme potřebovat tři různé třídy. Všechny tyto možné nastavení si popíšeme níže.

Každá třída poskytuje rozličné možnosti. Aby byla práce z těmito nastaveními uzlů jednodušší, tak všechny tyto třídy implementují společné rozhraní `INodeConfig`, které jasně definuje požadované funkce pro získání jména, typu, konfiguračního formuláře v podobě HTML elementu, akci k vykonání při odeslání formuláře, k resetování nastavení a získání statistických dat.

Nastavení vstupních uzlů

První probíranou třídou bude ta, která poskytuje nastavení vstupních uzlů. Její implementaci nalezneme ve třídě `InputNodeConfig`. Implementuje zmíněné společné rozhraní a navíc také rozhraní `IConfig`, které slouží k rozšíření nastavení uzlů.

Zároveň dění od třídy `EdgeConfig`, díky které získává nastavení pro výstupní hrany. Toto nastavení se týká křižovatky, kterou tento uzel společně s uzlem obsluhy obsahuje. Více o této třídě si povíme o trochu níže v samostatné části.

Dále v této třídě nalezneme objekty pro nastavení generátorů z různých rozděleních pravděpodobnosti a pro nastavení a ukládání statistik z průběhu simulace. Detaily informace o těchto třídách si povíme rovněž níže. Nic dalšího v této třídě nenalezneme.

Nastavení uzlů obsluhy

Nejsložitější nastavení bude to pro uzly obsluhy. Její implementaci se nachází ve třídě `NodeConfig`. Stejně jako nastavení vstupního uzle, i toto nastavení implementuje dvě zmíněné rozhraní.

Také dědí od třídy `EdgeConfig` a obsahuje objekty pro nastavení generátorů a pro nastavení a ukládání statistik z průběhu simulace.

Zde již podobnost s předchozím nastavením končí. Obsahuje navíc informaci o maximální délce fronty, objekt reprezentující frontu a také informaci o tom, jestli uzel pracuje, tedy zda probíhá obsluha nějakého požadavku. Tyto data navíc slouží pro potřeby simulace.

Nastavení výstupních uzlů

Nejjednodušší bude nastavení výstupních uzlů, které nalezneme ve třídě `OutputNodeConfig`. Stejně jako předchozí třídy i tato implementuje společné rozhraní, a proto obsahuje zmíněné funkce. Samotný uzel neobsahuje žádné nastavení a jediné věc, která se uzlu dá nastavit, je jestli chceme sbírat detailní statistiky při průběhu simulace. Obsahuje také objekt sloužící k uložení těchto statistik.

7.4.2 Pomocné třídy

Pro lepší oddělení jednotlivé funkcionality a omezení kopírování stejného kódu, bylo vytvořeno hned několik pomocných tříd poskytujících zmíněnou funkcionalitu. Tyto třídy si zde postupně rozebereme. Začneme tou sloužící pro nastavení generátorů, dále se podíváme na křižovatky a nakonec probereme statistiky.

Nastavení generátorů

Důležitou součástí nastavení uzlů je samotné nastavení generátorů pro různá rozdělení pravděpodobnosti. Tuto implementaci nalezneme ve třídě `Distri-`

butionConfig. Obsahuje funkce potřebné k vytvoření seznamu všech dostupných generátorů a po vybrání konkrétního typu i políčka k nastavení jeho parametrů.

Nastavení křižovatek

Další zajímavou částí je nastavení křižovatek, které určují kudy bude požadavek dále pokračovat na základě pravděpodobnosti jednotlivých přechodů. Zde je implementace rozdělená do dvou tříd.

První z nich je třída `EdgeConfig`. Ta zapouzdřuje následující třídu a přidává pár užitečných věcí navíc. Jako jsou pomocné funkce pro odstranění nebo pro přidání hrany a podobně. Dále poskytuje místo pro napojení případné další funkcionality.

Druhou je pak třída `OutputCrossroad`. Zde nalezneme implementaci spojenou se samotnou křižovatkou. Nalezneme v ní funkce pro vytvoření posuvníků a políček pro nastavení jednotlivých hran a jejich pravděpodobností. Samozřejmě zde najdeme i tyto informace a funkce pro jejich změnu a případně jejich získání. Také obsahuje funkce pro přepočítání zadaných pravděpodobností podle poměru, tak aby jejich součet byl požadovaných 100 procent.

Nastavení statistik

Poslední zmíněnou třídou bude třída pro statistiky ze simulace jejíž implementace můžeme naleznou ve třídě `StatisticsConfig`. Ta slouží nejen pro nastavení toho, jestli chceme sbírat detailní informace o průběhu simulace, ale také poskytuje přístup k strukturám pro uložení všech dostupných statistik z této simulace.

Nalezneme v ní funkce pro nastavení těchto statistik, pro vytvoření HTML elementu obsahující nastavení statistik a funkce pro získání HTML elementu s výsledky simulace.

Tuto třídu obsahuje každý uzel našeho modelu. Díky tomu můžeme jednoduše zaznamenávat všechny potřebné údaje přímo při simulaci a zároveň i nastavení je pro každý uzel modelu vlastní.

Uložení statistik

K uložení statistik ze simulace slouží hned několik druhů tříd, které všechny implementující rozhraní `IStatisticsData`. Toto rozhraní definuje funkce pro zaznamenání události do statistik, resetování statistik, získání HTML elementu obsahující statistiky, ověření existence statistik, získání celkového

počtu odbavených událostí, získání mapu s počty zpracovaných požadavků pro každou výstupní hranu a získání statistik v CSV formátu pro jejich uložení.

Protože statistiky pro jednotlivé typy uzlů budou mít jistě spoustu společných částí, využijeme zde nejdříve abstraktní třídy `StatisticsData`, kde implementujeme společné věci pro každý uzel. Nalezneme zde tak zaznamenání statistik pro obsloužené události a to jak jejich variantu pro pouhé součty, tak i tu s přesnými časy, pokud je uzel pro jejich ukládání nastaven. Tuhle abstraktní třídu dědí jednotlivé třídy pro každý typ uzlu. Ty si detailně popíšeme níže.

Vstupní uzel První z nich jsou statistiky pro vstupní uzel. Ty najdeme ve třídě `InputNodeStatisticsData`. Jak již bylo řečeno, třída dědí od abstraktní třídy `StatisticsData` a díky tomu získává velkou část své funkcionality.

V samotné třídě najdeme pouze přidání ukládání součtu intervalů mezi příchody dalšího požadavku. Tento součet se následně využívá k výpočtu průměrné doby příchodu požadavků, který zobrazíme ve výsledných statistikách pro konkrétní uzel.

Uzel obsluhy Dalším uzlem, jehož statistiky potřebujeme ukládat, je uzel obsluhy. Jeho statistiky zaznamenáváme pomocí třídy `NodeStatisticsData`, která také dědí od abstraktní třídy `StatisticsData`.

Uvnitř třídy máme hned několik přidaných statistik. První z nich je statistika počtu zahozených požadavků a to včetně varianty, která navíc zaznamenává i přesné časy, kdy k této události došlo, pro případ uložení detailních statistik. K jejímu zaznamenání slouží přidaná funkce s odpovídajícím názvem.

Dále jsou zde umístěny součty pro čas, který požadavky stráví ve frontě při čekání, a pro čas samotné obsluhy požadavku. Z nich se pak vypočítávají průměrné doby čekání a obsluhy. I pro ně existují příslušné funkce pro jejich záznam.

Poslední statistikou je maximální délka fronty v průběhu simulace. Ta se nastavuje rovněž přidanou funkcí, která otestuje aktuální stav fronty oproti maximu. Všechny tyto zmíněné statistiky jsou také přidány do výsledných statistik v příslušné funkci.

Výstupní uzel Nakonec se podíváme na statistiky pro výstupní uzel. Jejich ukládání najdeme ve třídě `OutputNodeStatisticsData`, která jako obě předchozí dědí od abstraktní třídy `StatisticsData`.

Sám o sobě nám uzel nenabízí příliš velké množství věcí, které bychom mohli měřit, ale i tak je zde přidáno ukládání součtu času, který strávili požadavky v celém systému, než dorazili do tohoto výstupní uzlu. Tento součet pak jednoduše využijeme k výpočtu průměrné doby požadavku v systému a tuto statistiku přidáme do výsledných statistik uzlu.

7.5 Generátory

Velmi důležitou částí je implementace jednotlivých generátorů náhodných čísel pro různá rozdělení pravděpodobnosti. Tu bychom si mohli rozdělit na několik částí. První z nich je vytváření instancí generátorů. Dále se musíme podívat na implementaci jednotlivých generátorů. Protože každý generátor bude vyžadovat různé parametry, je potřeba vytvořit jednoduchý způsob, který nám snadno umožní nastavovat parametry různého druhu. To bude poslední věc, kterou si rozebereme.

7.5.1 Vytvoření generátorů

K získání dostupných typů generátorů a k vytváření jejich instancí slouží třída `DistributionFactory`. Ta poskytuje dvě funkce. Jednu k získání konstruktorů pro každý dostupný generátor a funkci vytvoření samotné instance generátoru podle zadaného jména.

Pole jednotlivých konstruktorů získáme díky takzvaným 'dekorátorům'. Ty umožňují jednoduchým příkazem před deklaraci třídy, podobně jako se používají například anotace v Javě, zavolat určitou funkcionalitu pro každou třídu, kde je použit. V našem případě bude tento příkaz vypadat následovně: `@IDistribution.register`.

Díky tomu se nám pro každou takto označenou třídu zavolá funkce, která se nachází v rozhraní pro distribuce a slouží k registraci implementací. Takto velmi elegantně získáme výsledné pole konstruktorů, která používá již zmíněná tovární třída.

7.5.2 Jednotlivé generátory

Implementace jednotlivých generátorů je založená na již lehce zmíněném rozhraní `IDistribution`. To nám určuje, že výsledný generátor musí mít funkci pro vrácení jeho jména, získání jeho parametrů, nastavení jeho parametrů, vygenerování náhodného čísla a pro zresetování generátoru. Všechny tyto funkce jsou nezbytné pro každý generátor.

Z důvodu toho, že některá funkcionalita bude pro každý generátor stejný, existuje zde ještě abstraktní třída `Distribution`, která implementuje zmíněné rozhraní. Ta slouží pro nastavení semínka generátoru.

Dále bychom si měli rozebrat jednotlivé generátory. Ty máme v první verzi aplikace celkem tři. Jedná se o generátory s rovnoměrným, normálním a exponenciálním rozdělením. Jejich implementace nalezneme ve třídách `Uniform`, `Gaussian` a `Exponential`. Všechny dědí o třídy `Distribution`, díky tomu nemusíme řešit manipulaci se semínkem pro generátor.

Jednotlivé generátory vytváříme za pomoci knihovny `d3-random`[1], která poskytuje několik základních typů generátorů, které jsou dobře otestovány. Zároveň pro ni existuje plná podpora v TypeScriptu, díky vytvořenému definičnímu souboru.

7.5.3 Parametry generátorů

Poslední důležitou součástí generátorů je zpracování parametrů. K tomu slouží třída `ParameterParser`. Ta poskytuje funkce k zpracování jednoho nebo více parametrů. Zpracování spočívá v přeměně datového typu, protože od uživatele dostaneme pouze řetězec, ale většina parametrů generátorů vyžadují čísla.

K práci s parametry slouží typy definované v `Parameter` a `ParsedParameter`. V současné době aplikace využívá pouze parametry typu řetězec a číslo, ale v budoucnu je počítáno s přidáním dalších typů, pokud je budou nové generátory potřebovat.

7.6 Simulace

Bez simulace by naše aplikace postrádala smysl. Hlavní část implementace nalezneme ve třídě `Simulation`. Simulace běží pomocí interpretace jednotlivých událostí a to do doby, než dosáhneme maximálního času simulace.

Na začátku simulace musíme připravit všechny součásti simulace. K tomu slouží příslušná funkce, v které resetujeme všechny potřebné části. Dále je třeba inicializovat první události. Ty získáme z každého vstupního uzle tím, že vygenerujeme následující událost. Události vkládáme do kalendáře, který si popíšeme níže.

Zbytek simulace probíhá vybráním následující události a její interpretace. V systému máme tři typy uzlů, proto i zde se nabízejí tři možnosti co s událostmi dělat.

[1] `https://github.com/d3/d3-random`

Jako poslední je potřeba zajistit ukládání statistik, ať už základních, které jsou kumulativní, nebo detailních, kde zaznamenáváme pro uzel každou jeho událost.

7.6.1 Typy reakcí na událost

Jak již bylo zmíněno, události se vážou vždy k určitému uzlu. Díky tomu máme podstatě tři typy reakcí událost. Pro každý uzel budeme na událost reagovat jinak a vykonávat tak jiné činnosti, které si níže popíšeme.

Vstupní uzel

Začneme reakcemi na události pro vstupní uzly. Na začátku této události musíme vygenerovat následující událost pro konkrétní vstupní uzel a tu zařadit do kalendáře. Dále vykonáme samotnou obsluhu uzlu. Ta je velmi jednoduchá, protože pouze určíme následující uzel a pro něj vygenerujeme další událost.

Zde mohou nastat tři případy. První z nich je, že vstupní uzel nemá žádného následovníka. V tomto případě vypíšeme upozornění na do konzole. Další možností je, že uzel má jednoho následovníka. Potom je následující uzel jasný. Poslední variantou je pak křižovatka. Pro tu musíme vypočítat následující uzel podle pravděpodobnosti pro jednotlivé přechody.

Uzel obsluhy

Další v pořadí je uzel obsluhy. Jeho reakce na události jsou o trochu komplikovanější než pro předchozí vstupní uzel, protože má navíc frontu, v které se požadavek může nacházet a čekat na případnou obsluhu.

Aby bylo možné zjistit v jaké stavu se požadavek neboli událost nachází, máme zde proměnnou určující, jestli je již obsloužen. Díky tomu víme, jestli událost pro obslužný uzel právě přišla z fronty nebo byla již obsloužena.

U první možnosti musíme požadavek obsloužit, tedy vygenerovat čas potřebný k jeho zpracování, nastavení zmíněné proměnné na obsloužen a zařazení události zpět do kalendáře s novým časem. Tento čas je pak součtem aktuálního času a doby vygenerované obsluhy.

Pokud je událost již obsloužena, musíme rozhodnout kam z uzlu dál poputuje. Toho dosáhneme stejným způsobem jako u předchozího vstupního uzle pomocí výpočtu pro křižovatku. Dále je potřeba se podívat do fronty obslužného uzlu a v případě, že se v ní nachází nějaký požadavek, tak ho vybrat a umístit do kalendáře. Naopak prázdná fronta znamená, že uzel od této chvíle nepracuje a mi mu musíme tuto skutečnost nastavit, abychom věděli

jestli příchozí požadavek umístit do fronty pro čekání nebo ho posunout přímo k obsluze.

Výstupní uzel

Nejvíce jednoduchá bude zpracování požadavků pro výstupní uzel. Zde totiž nemáme podstatě žádnou práci, protože požadavek opouští síť a mi nad ním ztrácíme kontrolu. Je to tedy místo, kde můžeme pouze zaznamenat tuto skutečnost do případných statistik, ale další akce zde nebudou již třeba.

7.6.2 Kalendář událostí

Důležitou součástí simulace je kalendář událostí, jehož implementaci nalezneme ve třídě `Calendar`. Zajišťuje řazení událostí dle jejich času. Poskytuje funkce pro jednoduché vkládání a vybírání těchto požadavků, aby se programátor při jejich použití nemusel starat o zbytečné detaily.

Celá implementace kalendáře spočívá ve spojovém seznamu jednotlivých událostí jejichž implementaci nalezneme ve třídě `SimulationEvent`. Kalendář obsahuje pouze první událost, která v sobě obsahuje odkaz na další. Události jsou do spojového seznamu zařazovány podle času, který každá událost obsahuje.

Kalendář tak obsahuje funkce k přidání a odebrání událostí. Dále u něho můžeme najít funkci pro resetování kalendáře, která slouží k prvotní přípravě kalendáře. Smaže všechny události v kalendáři a připraví jej tak k použití v simulaci. Poslední je pak jednoduchá funkce k otestování, jestli kalendář obsahuje některou událost, tedy jinými slovy jestli je kalendář prázdný.

7.6.3 Jednotlivé události

Jak již bylo zmíněno, tak implementaci jednotlivých událostí nalezneme ve třídě `SimulationEvent`. Samotná třída pro ukládání události obsahuje identifikační číslo určeného uzlu, ke kterému se váže, její čas, následující událost a informaci jestli byla událost již obsloužena. Poskytuje potřebné funkce k nastavení a získání těchto dat. Správnost časového průběhu událostí, které jsou na sobě vázané pomocí odkazů zajišťuje přímo kalendář, který tedy nese veškerou odpovědnost za chronologické řazení těchto událostí.

7.7 Ostatní

V naší aplikaci jistě využijeme i další věci, které nespadají do zmíněných kategorií. Proto si některé z nich, které nám přijdou nejzajímavější, rozebere v následujících kapitolách.

7.7.1 Obarvení grafu

Pro obarvování modelu systému hromadné obsluhy slouží pomocná třída jejíž implementaci nalezneme ve třídě `Coloring`. Ta pomocí možností, které nabízí knihovna Cytoscape, a výsledků simulace obarví celý model již popsaným způsobem.

Ve třídě není žádná složitá implementace, kterou by zde bylo nutné speciálně rozebírat. Pouze projde všechny uzly a jejich hrany, kterým nastaví příslušné obarvení a u hran navíc tloušťku, na základě minimálních a maximálních hodnot.

7.7.2 Import

K importu souborů do aplikace slouží třída `Import`. Zde nalezneme v současné době pouze jednu metodu, která má na starosti načtení souboru pro import modelu systému hromadné obsluhy do aplikace. Využívá k tomu JavaScriptový objekt `FileReader`, který umožňuje načtení souboru od uživatele, a JSON objekt, který umí z řetěze vytvořit JSON objekt. Výsledný objekt pak můžeme dále využít v naší aplikaci pro vytvoření požadované struktury. Předání načteného souboru s případným chybovým textem je zajištěno pomocí zpětného volání.

7.7.3 Export

Pro export souborů z aplikace slouží třída `Export`. Uvnitř ní najdeme dvě funkce. První z nich umožňuje uživateli stáhnout soubor obsahující model systému včetně veškerého nastavení jednotlivých uzlů. Tento soubor má JSON strukturu a je označen příponou `.qtjs`. Druhá funkce poskytuje uživateli možnost získat soubor s výsledky simulace v podobě CSV souboru. V současné době je to jediný způsob jak tyto souhrnné výsledky simulace získat, protože aplikace zatím umožňuje pouze zobrazení výsledků pro jednotlivé uzly. Obě funkce využívají knihovnu FileSaver[2], která zajišťuje veškerou funkcionalitu pro umožnění stažení souboru uživatelem.

[2] `https://github.com/eligrey/FileSaver.js`

7.7.4 Logování

K možnosti logování v aplikaci slouží třída `Logger`. Ta poskytuje statické funkce k vytváření záznamů v průběhu různých událostí v aplikaci. Nalezneme zde funkce pro oznámení závažné chyby, varování, informační zprávy a zprávy sloužící pro ladění aplikace. Použití je velmi jednoduché a zároveň umožňuje nastavení jednoduchých šablon pro tyto zprávy.

8 Testování

O tom jak správně otestovat aplikaci si můžeme přečíst nespočet knih a článků. Způsobů jak aplikaci dobře otestovat je hned několik. My bychom se vždy měli snažit využít ten způsob, který je pro nás nejvýhodnější z pohledu stráveného času a získaného užitku.

K ověření správnosti simulace celého systému můžeme použít velmi jednoduchou metodu. Ta spočívá v porovnání výsledků simulace systému s jeho analytickým řešením. Musíme jen vybrat takový systém hromadné obsluhy, pro který toto analytické řešení umíme pokud možno jednoduše nalézt. Díky tomu získáme ověření o správnosti naší simulace a zároveň tímto otestujeme i některé další části aplikace. Z uvedeného textu vyplývá, že hlavní částí ověření správnosti aplikace bude funkční testovávání.

Dále je nutné zmínit, že všechny knihovny, které v aplikaci využíváme, jsou důkladně otestovány jejich tvůrci. Proto u nich předpokládáme správnost a jejich testováním se zaobírat nebudeme.

8.1 Ověření příkladem

Jak již bylo zmíněno, nejdůležitější částí testování bude ověření výsledků simulace oproti analytickému řešení systému. K tomu si nejdříve musíme vysvětlit několik vzorečků, které při analytickém řešení budeme potřebovat.

Abychom umožnili jednodušší výpočet analytického řešení, budeme předpokládat splnění několika podmínek, které nenaruší ověření funkčnosti simulace naší aplikace. Tyto podmínky jsou nazývány Jacksonovým teorémem [11].

První z nich je podmínka, že všechny elementární uzly systému hromadné obsluhy budou ve stacionárním režimu ($\rho < 1$). Vzorec pro výpočet zatížení nalezneme v kapitole 2.1 a bude také uveden v následujícím příkladu.

Protože generátory všech typů pravděpodobnostních rozdělení vytváříme pomocí knihovny, která je již otestována, nemusíme při našem testování zkoušet všechny tyto typy generátorů, ale můžeme si vybrat ty, které se nám budou nejvíce hodit.

To využijeme ke splnění další podmínky, která nám definuje, že všechny vstupní toky do systému musí mít poissonovský charakter. To splníme jednoduše použitím exponenciálního rozdělení pro generátor intervalů mezi příchody požadavků do sítě, protože pak výsledné příchody mají Poissonovo rozdělení. Dále potřebujeme, aby jednotlivé elementární uzly měli exponen-

ciální rozdělení dob obsluh. I toto nebude problém v aplikaci zařídit, protože stačí každému uzlu nastavit generátor s tímto rozdělením pravděpodobnosti.

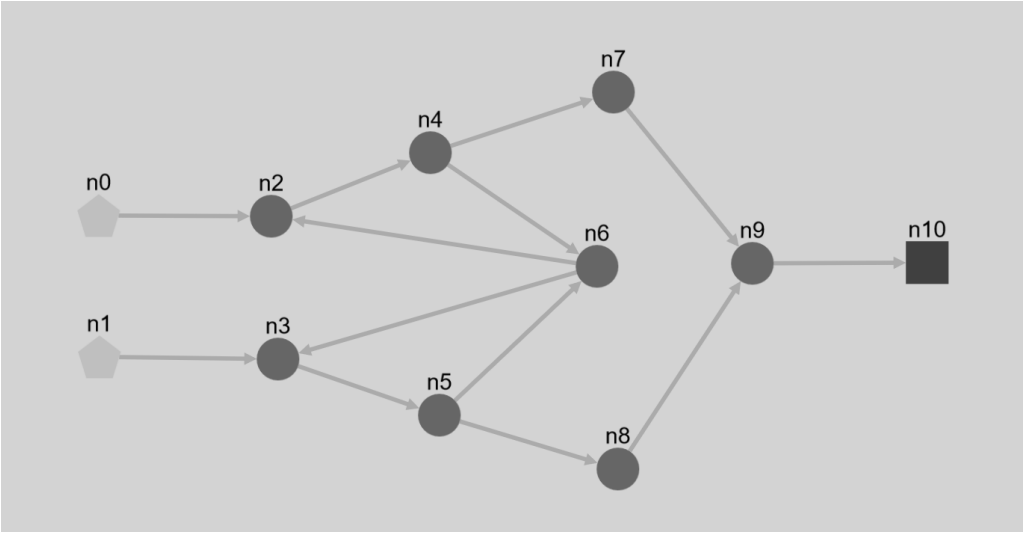
Poslední podmínka nám říká, že po ukončení obsluhy, požadavek okamžitě přechází do dalšího uzlu bez jakékoliv zpoždění. To vše náhodně s určitou pravděpodobností pro konkrétní přechod.

Díky těmto podmínkám můžeme řešit elementární systémy hromadné obsluhy jednotlivě jako samostatné M/M/m. Stačí nám pouze zjistit jaký bude vstup do každého uzlu sítě. Ty můžeme určit jako součet jednotlivých vstupních toků, protože víme, že součet poissonovských toků nám ve výsledku dá opět poissonovský tok [2] a také průchod uzlem s exponenciální dobou obsluh nám neporuší poissonovský charakter toku.

8.1.1 Příklad

Celou síť front můžeme vidět na obrázku 8.1, který je pořízen přímo z námi vytvořené aplikace. Jako první by bylo vhodné stanovat vstupní proudy do celého systému. Ty představují vstupní uzly $n0$ a $n1$ a jak jíž bylo zmíněno, budeme je modelovat pomocí generátoru s exponenciálním rozdělením pravděpodobnosti. V aplikaci nemáme určení časových jednotek, které používáme, ale je důležité dodržovat stejné jednotky pro všechny zadávané hodnoty, proto je v příkladu budeme uvádět pro lepší přehlednost. Parametry lambda pro vstupní toky vypadají následovně:

$$\lambda_0 = 6\ hod^{-1}$$
$$\lambda_1 = 4\ hod^{-1}$$



Obrázek 8.1: Model pro první příklad

Určitě nás budou také zajímat pravděpodobnosti přechodů mezi uzly. Nebudeme zde uvádět přechody pro uzly, kde je možný pouze jeden přechod, protože to by nemělo velký význam. Pravděpodobnost si označíme symbolem $p_{i,j}$, která nám určuje pravděpodobnost přechodu z uzlu i do uzlu j. Hodnoty pravděpodobností jsou následující:

$$p_{4,6} = 0,3$$
$$p_{4,7} = 0,7$$
$$p_{5,6} = 0,3$$
$$p_{5,8} = 0,7$$
$$p_{6,2} = 0,5$$
$$p_{6,3} = 0,5$$

Dále je potřeba stanovit parametry pro uzly obsluhy. Ty budeme nastavovat pomocí parametru μ, který představuje intenzitu obsluhy. Z této hodnoty snadno vypočteme střední dobu obsluhy T_s pomocí vzorce 8.1 [12].

$$T_s = \frac{1}{\mu} \tag{8.1}$$

Díky tomu po dosazení získáme následující hodnoty:

$$T_{s2} = \frac{1}{10} = 0,1\ hod$$
$$T_{s3} = \frac{1}{10} = 0,1\ hod$$
$$T_{s4} = \frac{1}{14} \doteq 0,0714\ hod$$
$$T_{s5} = \frac{1}{14} \doteq 0,0714\ hod$$
$$T_{s6} = \frac{1}{8} = 0,125\ hod$$
$$T_{s7} = \frac{1}{6} \doteq 0,1667\ hod$$
$$T_{s8} = \frac{1}{6} \doteq 0,1667\ hod$$
$$T_{s9} = \frac{1}{15} \doteq 0,0667\ hod$$

Následně jsme schopni určit rovnice pro vnitřní toky sítě. Jak již bylo zmíněno, tak jednotlivé vstupní toky pro každý uzel můžeme sečíst a pracovat pouze se součtem jako by se jednalo o jeden tok. Pro součet využijeme vzoreček 8.2 [12]. Kde Λ_i značí součet toků pro uzel i a $p_{k,i}$ je pak pravdě-

podobnost přechodu z uzlu k do uzlu i.

$$\Lambda_i = \sum_k \Lambda_k p_{k,i} \tag{8.2}$$

Na základě těchto poznatků umíme vytvořit následující soustavu rovnic:

$$\begin{aligned}
\Lambda_2 &= \lambda_0 + (\Lambda_6 p_{6,2}) \\
\Lambda_3 &= \lambda_1 + (\Lambda_6 p_{6,3}) \\
\Lambda_4 &= \Lambda_2 p_{2,4} \\
\Lambda_5 &= \Lambda_3 p_{3,5} \\
\Lambda_6 &= (\Lambda_4 p_{4,6}) + (\Lambda_5 p_{5,6}) \\
\Lambda_7 &= \Lambda_4 p_{4,7} \\
\Lambda_8 &= \Lambda_5 p_{5,8} \\
\Lambda_9 &= (\Lambda_7 p_{7,9}) + (\Lambda_8 p_{8,9})
\end{aligned}$$

Po dosazení získáme následující soustavu:

$$\begin{aligned}
\Lambda_2 &= 6 + (\Lambda_6 \cdot 0,5) \\
\Lambda_3 &= 4 + (\Lambda_6 \cdot 0,5) \\
\Lambda_4 &= \Lambda_2 \cdot 1 \\
\Lambda_5 &= \Lambda_3 \cdot 1 \\
\Lambda_6 &= (\Lambda_4 \cdot 0,3) + (\Lambda_5 \cdot 0,3) \\
\Lambda_7 &= \Lambda_4 \cdot 0,7 \\
\Lambda_8 &= \Lambda_5 \cdot 0,7 \\
\Lambda_9 &= (\Lambda_7 \cdot 1) + (\Lambda_8 \cdot 1)
\end{aligned}$$

Jednoduchými matematickými operacemi získáme rovnice pro Λ_2 a Λ_3:

$$\begin{aligned}\Lambda_2 &= 6 + ((\Lambda_2 \cdot 0,3) + (\Lambda_3 \cdot 0,3)) \cdot 0,5\\ \Lambda_3 &= 4 + ((\Lambda_2 \cdot 0,3) + (\Lambda_3 \cdot 0,3)) \cdot 0,5\end{aligned}$$

$$\begin{aligned}\Lambda_2 &= 6 + (\Lambda_2 \cdot 0,15) + (\Lambda_3 \cdot 0,15)\\ \Lambda_3 &= 4 + (\Lambda_2 \cdot 0,15) + (\Lambda_3 \cdot 0,15)\end{aligned}$$

$$\begin{aligned}\Lambda_2 &= 6 + (\Lambda_2 \cdot 0,15) + (\Lambda_3 \cdot 0,15)\\ -\Lambda_3 &= 4 + (\Lambda_2 \cdot 0,15) + (\Lambda_3 \cdot 0,15)\end{aligned}$$

$$\begin{aligned}\Lambda_2 - \Lambda_3 &= 2\\ \Lambda_2 &= 2 + \Lambda_3\end{aligned}$$

$$\begin{aligned}\Lambda_3 &= 4 + ((2 + \Lambda_3) \cdot 0,15) + (\Lambda_3 \cdot 0,15)\\ \Lambda_3 &= 4 + (0,3 + (\Lambda_3 \cdot 0,15)) + (\Lambda_3 \cdot 0,3)\\ \Lambda_3 &= 4 + 0,3 + (\Lambda_3 \cdot 0,3)\end{aligned}$$

Z těchto rovnic pak získáme hodnoty pro již zmiňované Λ_2 a Λ_3:

$$\begin{aligned}\Lambda_3 &= 4 + 0,3 + (\Lambda_3 \cdot 0,3)\\ \Lambda_3 \cdot 0,7 &= 4,3\\ \Lambda_3 &= \frac{4,3}{0,7}\\ \Lambda_3 &\doteq 6,143\end{aligned}$$

$$\begin{aligned}\Lambda_2 &= 2 + \Lambda_3\\ \Lambda_2 &= 2 + \frac{4,3}{0,7}\\ \Lambda_2 &\doteq 8,143\end{aligned}$$

Díky získaným Λ_2 a Λ_3 můžeme získat i ostatní hodnoty:

$$\Lambda_4 \doteq 8,143$$
$$\Lambda_5 \doteq 6,143$$
$$\Lambda_6 \doteq 4,286$$
$$\Lambda_7 \doteq 5,7$$
$$\Lambda_8 \doteq 4,3$$
$$\Lambda_9 \doteq 10$$

Teď již máme všechny potřebné údaje k ověření stacionarity jednotlivých uzlů systému. To je jedna z našich stanovených podmínek, abychom mohli využít jednodušší vzorce pro výpočet analytického řešení. Pro ověření využijeme následující vzorec 8.3 [12], kde m_i bude v našem případě vždy rovné jedné, protože každý uzel v aplikaci má právě jeden uzel obsluhy.

$$\rho_i = \frac{1}{m_i}\Lambda_i T_{si} \tag{8.3}$$

Pro každý uzel sítě jsme tak schopni ověřit stacionaritu následovně:

$$\rho_2 = (2 + \frac{4,3}{0,7}) \cdot \frac{1}{10} \doteq 0,8143$$
$$\rho_3 = \frac{4,3}{0,7} \cdot \frac{1}{10} \doteq 0,6143$$
$$\rho_4 = (2 + \frac{4,3}{0,7}) \cdot \frac{1}{14} \doteq 0,5816$$
$$\rho_5 = \frac{4,3}{0,7} \cdot \frac{1}{14} \doteq 0,4388$$
$$\rho_6 = ((2 + \frac{4,3}{0,7}) \cdot 0,3 + (\frac{4,3}{0,7} \cdot 0,3)) \cdot \frac{1}{8} \doteq 0,5357$$
$$\rho_7 = ((2 + \frac{4,3}{0,7}) \cdot 0,7) \cdot \frac{1}{6} \doteq 0,9500$$
$$\rho_8 = (\frac{4,3}{0,7} \cdot 0,7) \cdot \frac{1}{6} \doteq 0,7167$$
$$\rho_9 = 10 \cdot \frac{1}{15} = 0,6667$$

Jak můžeme vidět, všechny uzly splňují podmínku $\rho_i < 1$, takže můžeme prohlásit, že všechny uzly sítě jsou stacionární.

Teď již můžeme spočítat střední doby průchodu požadavku pro jednotlivé uzly. Na to budeme potřebovat vzoreček 8.4 [12], díky kterému jsme schopni odhadnout přibližnou střední dobu průchodu požadavku uzlem. Tento odhad

je velmi přesný pro poissonovské sítě, kde $m \in \{1, 2\}$.

$$T_{qi} \cong \frac{T_{si}}{1 - \rho_i^m} \tag{8.4}$$

Na základě tohoto vzorečku vypočteme jednotlivé střední doby průchodu požadavku pro každý uzel:

$$T_{q2} \cong \frac{\frac{1}{10}}{1 - (2 + \frac{4,3}{0,7}) \cdot \frac{1}{10}} \doteq 0,5385$$

$$T_{q3} \cong \frac{\frac{1}{10}}{1 - \frac{4,3}{0,7} \cdot \frac{1}{10}} \doteq 0,2593$$

$$T_{q4} \cong \frac{\frac{1}{14}}{1 - (2 + \frac{4,3}{0,7}) \cdot \frac{1}{14}} \doteq 0,1707$$

$$T_{q5} \cong \frac{\frac{1}{14}}{1 - \frac{4,3}{0,7} \cdot \frac{1}{14}} \doteq 0,1273$$

$$T_{q6} \cong \frac{\frac{1}{8}}{1 - ((2 + \frac{4,3}{0,7}) \cdot 0,3 + (\frac{4,3}{0,7} \cdot 0,3)) \cdot \frac{1}{8}} \doteq 0,2692$$

$$T_{q7} \cong \frac{\frac{1}{6}}{1 - ((2 + \frac{4,3}{0,7}) \cdot 0,7) \cdot \frac{1}{6}} \doteq 3,3333$$

$$T_{q8} \cong \frac{\frac{1}{6}}{1 - (\frac{4,3}{0,7} \cdot 0,7) \cdot \frac{1}{6}} \doteq 0,5882$$

$$T_{q9} \cong \frac{\frac{1}{15}}{1 - 10 \cdot \frac{1}{15}} \doteq 0,2000$$

Získali jsme tak střední dobu průchodů jednotlivými uzly, ale určitě by nás zajímala i střední doba průchodu celým systém T_q, někdy označována také jako odezva systému. Jeho výpočet zajistíme vzorcem 8.5 [12], kde L_q znamená počet požadavků v systému a *Lambda* označuje součet vstupních toků do systému.

$$T_q = \frac{L_q}{\Lambda} \tag{8.5}$$

Nejdříve musíme zjistit hodnotu pro L_q, protože bez ní jen těžko vypočteme odezvu celého systému. Celkový počet požadavků v systému spočteme jako součet požadavků v jednotlivých uzlech, proto nám stačí vzorec 8.6, díky kterému zjistíme požadovanou hodnotu pro každý uzel systému. Stejně jako u 8.4, se i zde jedná pouze o odhad se stejnými předpoklady.

$$L_{qi} \cong \frac{m\rho_i}{1 - \rho_i^m} \tag{8.6}$$

Po dosazení do zmíněného vzorce pro každý uzel dostaneme následující hodnoty:

$$
\begin{aligned}
L_{q2} &\cong \frac{(2+\frac{4,3}{0,7})\cdot\frac{1}{10}}{1-(2+\frac{4,3}{0,7})\cdot\frac{1}{10}} && \doteq 4,3846\\
L_{q3} &\cong \frac{\frac{4,3}{0,7}\cdot\frac{1}{10}}{1-\frac{4,3}{0,7}\cdot\frac{1}{10}} && \doteq 1,5926\\
L_{q4} &\cong \frac{(2+\frac{4,3}{0,7})\cdot\frac{1}{14}}{1-(2+\frac{4,3}{0,7})\cdot\frac{1}{14}} && \doteq 1,3902\\
L_{q5} &\cong \frac{\frac{4,3}{0,7}\cdot\frac{1}{14}}{1-\frac{4,3}{0,7}\cdot\frac{1}{14}} && \doteq 0,7818\\
L_{q6} &\cong \frac{((2+\frac{4,3}{0,7})\cdot 0,3+(\frac{4,3}{0,7}\cdot 0,3))\cdot\frac{1}{8}}{1-((2+\frac{4,3}{0,7})\cdot 0,3+(\frac{4,3}{0,7}\cdot 0,3))\cdot\frac{1}{8}} && \doteq 1,1538\\
L_{q7} &\cong \frac{((2+\frac{4,3}{0,7})\cdot 0,7)\cdot\frac{1}{6}}{1-((2+\frac{4,3}{0,7})\cdot 0,7)\cdot\frac{1}{6}} && \doteq 19,0000\\
L_{q8} &\cong \frac{(\frac{4,3}{0,7}\cdot 0,7)\cdot\frac{1}{6}}{1-(\frac{4,3}{0,7}\cdot 0,7)\cdot\frac{1}{6}} && \doteq 2,5294\\
L_{q9} &\cong \frac{10\cdot\frac{1}{15}}{1-10\cdot\frac{1}{15}} && \doteq 2,0000
\end{aligned}
$$

Díky těmto hodnotám pak spočteme jednoduše počet požadavků v celém systému pomocí součtu jednotlivého počtu požadavků v každém uzlu systému. Výsledný výpočet vypadá pro náš systém bude vypadat následovně:

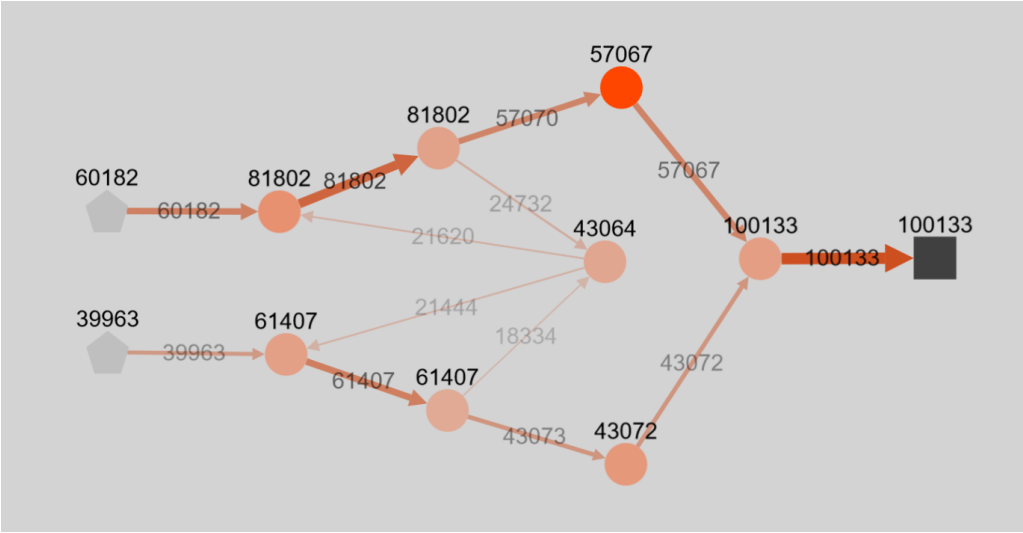
$$
\begin{aligned}
L_q &= \sum_{i=2}^{9} L_{qi}\\
L_q &\doteq 32,8324
\end{aligned}
$$

Následně nám stačí použít vzorec 8.5 a dosazením do něj získáme střední dobu průchodu požadavku celým systém:

$$
\begin{aligned}
T_q &= \frac{L_q}{\Lambda}\\
T_q &= \frac{32,8324}{10}\\
T_q &= 3,28324
\end{aligned}
$$

Srovnání

Teď již máme všechny potřebná data pro srovnání s výsledky simulace. Pro ilustraci zde přidáme i odsimulovanou síť, kterou můžeme vidět na obrázku 8.2. Při této simulaci modelem prošlo přibližně 100 000 a během necelé vteřiny jsme dostali výsledky, které použijeme pro srovnání níže.



Obrázek 8.2: Simulace modelu pro první příklad

Pro srovnání využijeme jednotlivé tabulky, které nám hezky ukáží rozdíly mezi výsledky z analytického řešení oproti simulačnímu. Nejprve se podívejme se pojďme podívat spíše na zajímavé srovnání nastavení dob obsluh jednotlivých uzlů. To můžeme vidět v tabulce 8.1. Když se podíváme na jen drobné rozdíly ve výsledcích, můžeme konstatovat, že systém máme nastavený správně a můžeme pokračovat ve srovnávání.

Tabulka 8.1: Srovnání středních dob obsluh uzlů systému

Uzel	Analytický model	Simulace	**Rozdíl**
T_{s2}	0,1	0,0999	0,0001
T_{s3}	0,1	0,0999	0,0001
T_{s4}	0,0714	0,0714	0,0000
T_{s5}	0,0714	0,0719	-0,0005
T_{s6}	0,125	0,1254	-0,0004
T_{s7}	0,1667	0,1660	0,0007
T_{s8}	0,1667	0,1666	0,0001
T_{s9}	0,0667	0,0668	-0,0001

Další porovnání najdeme v tabulce 8.2. V něm vedle sebe dáváme hodnoty středních dob průchodů požadavků uzlem systému. Jak můžeme vidět, ani zde nejsou příliš velké rozdíly.

Tabulka 8.2: Srovnání středních dob průchodů požadavků uzlem systému

Uzel	Analytický model	Simulace	**Rozdíl**
T_{q2}	0,5385	0,5237	0,0148
T_{q3}	0,2593	0,2598	-0,0005
T_{q4}	0,1707	0,1713	-0,0006
T_{q5}	0,1273	0,1282	-0,0009
T_{q6}	0,2692	0,2696	-0,0004
T_{q7}	3,3333	3,3072	0,0263
T_{q8}	0,5882	0,5912	-0,0030
T_{q9}	0,2000	0,2017	-0,0017

Posledním je porovnání střední doby průchodu požadavku celým systémem. Samotné srovnání můžeme najít v tabulce 8.3. I zde jsou rozdíly minimální, proto můžeme prohlásit, že simulace odpovídá výsledkům analytického řešení.

Tabulka 8.3: Srovnání středních dob průchodů požadavků celým systému

Vlastnost	Analytický model	Simulace	**Rozdíl**
T_q	3,2832	3,2625	0,0207

9 Závěr

Hlavním výsledkem této práce je webová aplikace, která slouží k modelování a simulování systémů hromadné obsluhy. Ta by měla sloužit primárně pro potřeby výuky předmětu výkonnost a spolehlivost programových systémů na Fakultě aplikovaných věd, ale určitě můžeme nalézt i jiné využití mimo tento předmět.

Vytvořili jsme nástroj s intuitivním ovládáním, které se velmi rychle a bez větších problémů naučí každý student zmíněného předmětu. Umožňuje jednoduché nahrání již vytvořených modelů nebo naopak export nově zkonstruovaných systémů. Zároveň můžeme získat výsledky simulace ve formátu CSV, který je vhodný pro další zpracování a jiné úkony.

Distribuce programu je jednoduchá v podobě jednoho HTML souboru a k jejímu spuštění postačí jakýkoliv prohlížeč. Protože veškeré knihovny obsahuje přímo aplikace, nepotřebujeme k jejímu běhu ani internetové připojení. I případné nasazení na jakoukoliv již existující stránku nebude moc komplikované.

Celá aplikace byla vyvíjena s předpokladem dalšího vývoje, proto by případné rozšiřovaní aplikace v budoucnu nemělo klást velké překážky při přidávání nové funkcionality.

Přehled zkratek

- IT (`Information technology`) – informační technologie
- FIFO (`First In, First Out`) – označení fronty první přijde, první bude obsloužen
- LIFO (`Last In, First Out`) – označení fronty poslední přijde, první bude obsloužen
- HTML (`Hypertext Markup Language`) – značkovací jazyk pro tvorbu webových stránek
- SVG (`Scalable Vector Graphics`) – formát pro vektorovou grafiku
- CSS (`Cascading Style Sheets`) – jazyk pro popis způsobu zobrazení elementů
- JSON (`JavaScript Object Notation`) – formát pro zápis dat
- CSV (`Comma Separated Values`) – jednoduchý formát pro zápis dat

Literatura

[1] Adan, I. – Resing, J. Queueing Theory. 2001. Dostupné z: http://wwwhome.math.utwente.nl/~scheinhardtwrw/queueingdictaat.pdf.

[2] Adelson, R. M. Compound Poisson Distributions. *Journal of the Operational Research Society*. 1966, 17, 1. doi: 10.1057/jors.1966.8. Dostupné z: https://doi.org/10.1057/jors.1966.8.

[3] Meer Kishor S. Trivedi, G. B. S. G. H. Queueing Networks and Markov Chains: Modeling and Performance Evaluation With Computer Science Applications. 2006. doi: 10.1002/0471791571.

[4] Franz, M. et al. Cytoscape.js: a graph theory library for visualisation and analysis. *Bioinformatics*. 09 2015, 32, 2. ISSN 1367-4803. doi: 10.1093/bioinformatics/btv557. Dostupné z: https://doi.org/10.1093/bioinformatics/btv557.

[5] Heyman, D. P. Queueing Systems, Volume 2: Computer applications. by Leonard Kleinrock John Wiley & Sons, Inc., New York 1976. *Networks*. 1977. doi: 10.1002/net.3230070308. Dostupné z: https://onlinelibrary.wiley.com/doi/abs/10.1002/net.3230070308.

[6] Hlynka, M. *The M/M/1 Queue*. American Cancer Society, 2011. doi: 10.1002/9780470400531.eorms0891. Dostupné z: https://onlinelibrary.wiley.com/doi/abs/10.1002/9780470400531.eorms0891. ISBN 9780470400531.

[7] Kaur, H. – Manjunath, D. – Bose, S. The queueing network analysis tool (QNAT). 02 2000. doi: 10.1109/MASCOT.2000.876557.

[8] Kendall, D. G. Some Problems in the Theory of Queues. *Journal of the Royal Statistical Society: Series B (Methodological)*. 1951, 13, 2. doi: 10.1111/j.2517-6161.1951.tb00080.x. Dostupné z: https://rss.onlinelibrary.wiley.com/doi/abs/10.1111/j.2517-6161.1951.tb00080.x.

[9] Little, J. D. C. – Graves, S. C. *Little's Law*. Springer US, Boston, MA, 2008. doi: 10.1007/978-0-387-73699-0_5. Dostupné z: https://doi.org/10.1007/978-0-387-73699-0_5. ISBN 978-0-387-73699-0.

[10] Mandelbaum, M. – Hlynka, M. Examples of Applications of Queueing Theory in Canada. *INFOR: Information Systems and Operational*

Research. 2008, 46, 4. doi: 10.3138/infor.46.4.247. Dostupné z: https://doi.org/10.3138/infor.46.4.247.

[11] R. Jackson, J. Network of Waiting Lines. *Operations Research*. 08 1957, 5, s. 518–521. doi: 10.1287/opre.5.4.518.

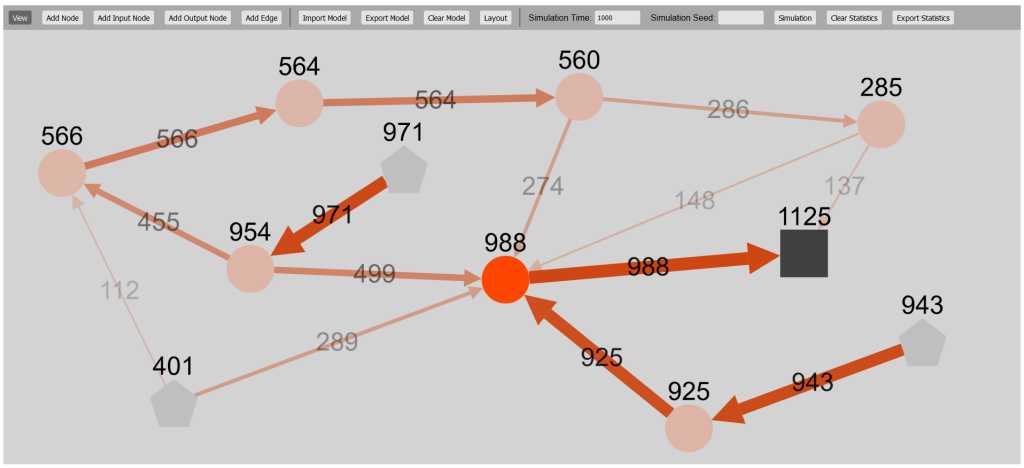
[12] Racek, S. Pravděpodobnostní modely počítačů. *ZČU, Plzeň*. 2002, s. 35–49.

[13] Rastogi, A. et al. Safe & Efficient Gradual Typing for TypeScript. *SIGPLAN Not.* January 2015, 50, 1. ISSN 0362-1340. doi: 10.1145/2775051.2676971. Dostupné z: http://doi.acm.org/10.1145/2775051.2676971.

A Přílohy

A.1 Uživatelská příručka

Ovládání celé aplikace je jednoduché a velmi intuitivní. Všechny funkce aplikace jsou jasně viditelné a jednoduše pochopitelné, ale i tak jsme vytvořili krátkou uživatelskou příručku, kde si popíšeme jednotlivé možnosti aplikace. Ukázku aplikace můžeme vidět níže na obrázku A.1.



Obrázek A.1: Ukázka aplikace

A.1.1 Základní ovládání

Aplikaci si můžeme rozdělit na dvě základní části. První z nich je horní pruh, který reprezentuje menu. V menu nalezneme veškeré tlačítka sloužící k ovládání aplikace. Druhou částí aplikace je pak samotné plátno, kde pracujeme s grafem reprezentující systém hromadné obsluhy.

Menu

I samotné menu si můžeme rozdělit do jednotlivých částí, které jsou i vizuálně odděleny. Jedná se o nastavení aktivního módu aplikace, akce s modelem a ovládání simulace s jejími výsledky.

Aktivní mód První z nich je sekce zabývající se nastavením aktivního módu aplikace. Máme na výběr z módu pro prohlížení, který najdeme pod tlačítkem `View`. Další je pak tlačítko `Add Node`, v kterém při kliknutí do plátna

přidáme uzel obsluhy. Podobný princip mají i další dvě tlačítka `Add Input Node` a `Add Output Node`, které využijeme pro přidání vstupních a výstupních uzlů. Posledním je mód pro přidávání hran. Ten nalezneme pod tlačítkem `Add Edge`. Pro přidání hrany stačí kliknout na uzel, z kterého chceme hranu vést, a na cílový uzel. Opakovaným kliknutím na vybraný uzel tento výběr zrušíme.

Akce s modelem Další skupina tlačítek slouží k různým akcím s modelem systému. První z nich je tlačítko pro nahraní modelu ze souboru. To nalezneme pod tlačítkem `Import Model`. Následuje tlačítko `Export Model`, které má opačnou funkcionalitu, tedy uložení aktuálního modelu do souboru, který si můžeme stáhnout. Dalším tlačítkem je `Clear Model`, který vymaže celý obsah plátna. Posledním je pak tlačítko `Layout`, které uskupí model podle pružinového rozložení.

Simulace Poslední částí jsou pak ovládací prvky pro simulaci. První z nich je vstupní pole pro čas simulace označené popiskem `Simulation Time`. Následuje nepovinné políčko pro nastavení semínka všem generátorům modelu označené textem `Simulation Seed`. Následuje tlačítko `Simulation`, které spustí simulaci. Poslední dvě tlačítka jsou zapnutá pouze pokud máme výsledky simulace. První z nich je tlačítko pro `Clear Statistics`, které vymaže výsledky simulace a odstraní zobrazené výsledky v modelu. Druhým je pak tlačítko `Export Statistics`, které umožňuje stáhnout výsledky simulace.

Plátno

Ovládání samotného plátna je velmi jednoduchá. Posun v plátně zajistíme držením stisknutého levého tlačítka myši a jejím následným posunem. K přiblížení a oddálení slouží kolečko myši. Akce pro krátké stisknutí levého tlačítka myši jsou určeny aktivním módem aplikace. Další možností je pak kliknutí pravým tlačítkem na uzel nebo hranu. Ten způsobí otevření dialogového okno, které je pro každý druh elementu trochu odlišné a poskytuje různé možnosti. Poslední akcí je pak najetí myši na uzel modelu, který způsobí otevřené okna s výsledky simulace v případě, že tyto data máme.

A.1.2 Možnosti elementů

Jak bylo již zmíněno, různé elementy nám nabízí rozličné možnosti při otevření dialogu. Ten otevřeme pomocí kliknutí pravým tlačítkem myši na konkrétní element.

Uzly

Rozmanité možnosti nabízí dialogy pro uzly modelu. Zde nalezneme různá nastavení, která jsou rozdílná pro vstupní, obslužný a výstupní uzel.

Uvnitř nich můžeme nalézt nastavení distribučního rozdělení pro náhodný generátor včetně jeho parametrů. Dále můžeme nastavit jestli chceme ukládat detailní statistiky pro uzel. Nesmíme ani opomenout nastavení křižovatek, kde určujeme pravděpodobnost jednotlivých přechodů a pro správný součet můžeme využít tlačítko `Recalculate` pro přepočítání těchto pravděpodobností na správný součet. Dále je zde možnost určit maximální délku frontu pro obslužný uzel. Nekonečnou frontu nastavíme zápornou hodnotou.

Samotný dialog má tři možná tlačítka. První z nich je tlačítko `Submit` sloužící k potvrzení změn. Dále tlačítko `Cancel`, které zruší dialog bez uložení změn, a jako poslední zde nalezneme tlačítko `Remove`. To slouží k vymazání vybraného elementu.

Hrany

Dialog pro hrany je velmi prostý a nabízí pouze potvrzení nebo zamítnutí akce pro vymazání vybrané hrany. Potvrdit odstranění můžeme pomocí tlačítka `Yes` a naopak odmítnutí tlačítkem `No`.