FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

ZADÁNÍ DIPLOMOVÉ PRÁCE

Název:	Výuková aplikace pro metodu GERT
Student:	Bc. Přemysl Černý
Vedoucí:	Ing. Petra Pavlíčková, Ph.D.
Studijní program:	Informatika
Studijní obor:	Webové a softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce letního semestru 2018/19

Pokyny pro vypracování

Výuková aplikace pro matematické modelování na základě metody GERT. Cílem této aplikace je vylepšení výuky matematického modelování a to přímo metody GERT, se kterou studenti mají problémy. Dalším cílem je nástroj, který umožní zlepšení časových odhadů v rámci projektů a úspěšném dokončení projektů.

1. Určete procesy, které bude aplikace podporovat.
2. Namapujte jednotlivé procesy na funkční požadavky aplikace.
3. Navrhněte grafické rozhraní aplikace.
4. Určete technologie, nad kterými bude vytvořena aplikace.
5. Připravte scénáře pro základní průchody aplikací.
6. Proveďte implementaci aplikace.
7. Otestujte aplikaci dle testovacích scénářů a opravte případné chyby.
8. Vytvořte uživatelskou a instalační příručku.
9. Proveďte ekonomické zhodnocení navržené aplikace z pohledu nákladů a benefitů.

Seznam odborné literatury

Dodá vedoucí práce.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
děkan

V Praze dne 3. ledna 2018

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Diplomová práce

Výuková aplikace pro metodu GERT

Bc. Přemysl Černý

Katedra softwarového inženýrství
Vedoucí práce: Ing. Petra Pavlíčková, Ph.D.

29. dubna 2018

Poděkování

Děkuji vedoucí této diplomové práce Ing. Petře Pavlíčkové, Ph.D. za vysvětlení a konzultace k metodám CPM a GERT.

Prohlášení

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

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

V Praze dne 29. dubna 2018 .....

České vysoké učení technické v Praze
Fakulta informačních technologií
© 2018 Přemysl Černý. Všechna práva vyhrazena.
Tato práce vznikla jako školní dílo na Českém vysokém učení technickém v Praze, Fakultě informačních technologií. Práce je chráněna právními předpisy a mezinárodními úmluvami o právu autorském a právech souvisejících s právem autorským. K jejímu užití, s výjimkou bezúplatných zákonných licencí a nad rámec oprávnění uvedených v Prohlášení na předchozí straně, je nezbytný souhlas autora.

Odkaz na tuto práci

Černý, Přemysl. *Výuková aplikace pro metodu GERT*. Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2018.

Abstrakt

Tato diplomová práce se zaměřuje na tvorbu výukové aplikace pro metodu GERT, která se používá pro odhad doby trvání projektů. Po teoretickém úvodu k projektovému řízení, k teorii grafů a k metodám CPM, PERT a GERT následuje analýza vlastní aplikace založená na sběru požadavků a případech užití, návrh architektury, implementace aplikace pro webové prohlížeče a tvorba základních výukových tutoriálů, které jsou dále rozšiřitelné, modifikovatelné a replikovatelné. Důraz je kladen především na interaktivitu aplikace, tvorbu vlastních činností projektu a jejich přidávání do grafové reprezentace a snadnou navigaci mezi tutoriály a jejich vnitřními stránkami. Hlavním cílem této práce je s pomocí zmíněné výukové aplikace zefektivnit výuku metody GERT a zvýšit tak pravděpodobnost jejího využívání při návrhu reálných projektů.

Klíčová slova interaktivní tutoriály, metoda CPM, metoda GERT, projektové řízení, tvorba výukové aplikace

Abstract

This thesis is focused on creating an educational application for the GERT method, which is commonly used to estimate the duration of projects. After a theoretical introduction to project management, graph theory and CPM, PERT and GERT methods, there follows an analysis of the previously mentioned application based on gathering requirements and on use cases, a design of architecture, an implementation of the application to be used in website browsers and creating basic educational tutorials, which are further expandable, modifiable and replicable. The emphasis is on interactivity of the application, creation of custom activities of projects and their addition to the graph representation and easy navigation among tutorials and their inner pages. The main objective of this thesis is, using the previously mentioned application, to make teaching the GERT method more effective and in this way increase the probability of its use during a design of real projects.

Keywords CPM method, creating an application for education, GERT method, interactive tutorials, project management

Obsah

Seznam obrázků

Seznam úryvků kódu

Seznam tabulek

Úvod

Metoda GERT je stochastická metoda síťové analýzy, která podobně jako metody CPM a PERT slouží především pro co nejpřesnější odhad doby trvání projektu. Na rozdíl od těchto uvedených metod však metoda GERT poskytuje přesnější výsledky díky využití hodnot pravděpodobností realizace jednotlivých činností projektu a lepšího pochopení závislostí mezi těmito činnostmi díky využití různých typů uzlů v grafové reprezentaci. Tato práce se zabývá vytvořením jednoduché, interaktivní a rozšiřitelné aplikace pro výuku metody GERT, která je především určená pro studenty bakalářského studia vysokých škol.

Aplikace bude obsahovat výukové tutoriály, které pomocí vysvětlujících popisných textů a modifikovatelných příkladů uživatele uvedou do problematiky síťového grafu, metody CPM a z ní vycházející metody GERT, kdy daný tutoriál uživatele podnítí ke tvorbě vlastního reálného projektu. V teoretické části této práce jsou tyto metody, způsob jejich použití a výpočtu vysvětleny spolu s nutným úvodem do problematik projektového řízení a teorie grafů. Diskutovány jsou rovněž ekonomické výhody metody GERT v porovnání s metodami ostatními. Za tímto teoretickým úvodem pak následuje sběr požadavků, analýza, návrh architektury, zvolení technologií a implementace vlastní aplikace, následována tvorbou samotných tutoriálů, testováním a zamyšlením nad možnými dalšími rozšířeními aplikace. Předpokládá se, že učitelé či studenti budou následně tvořit tutoriály vlastní, případně rozšiřovat zdrojový kód aplikace, čemuž je samotná aplikace spolu s jejím návrhem již od počátku přizpůsobena. Součástí analýzy je i návrh nejběžnějších případů užití, které popisují interakci mezi uživatelem a aplikací v základních případech.

Celá výuková aplikace je určena pro zobrazení ve webovém prohlížeči bez přístupu k internetovému připojení, aby byla snadno přenositelná na různé platformy. Pevně doufám, že tak bude snadno přístupná širšímu okruhu studentů a učitelů, kteří následně mohou zvážit praktické využití metody GERT pro odhad doby trvání projektů ve svých vlastních oborech, kterým se věnují. Nejsilnější stránkou metody GERT je právě její stochastická podstata, která

díky pravděpodobnosti umožňuje zlepšit odhady vypočítávaných veličin, přesněji dob nejdříve možných začátků a dob nejpozději přípustných začátků jednotlivých činností, a tedy i celkové doby trvání daného projektu. Tyto výpočty následně umožňují tvorbu přesnějších odhadů dob trvání činností budoucích projektů a tedy zvýšení efektivity a snížení nákladů, které by jinak vznikly například zpožděním projektu nebo nutností krizového řízení.

Další motivací pro tvorbu této výukové aplikace je fakt, že metoda GERT je obtížná na pochopení, především co se týče jejího využití kritické cesty v grafické reprezentaci spolu s pravděpodobnostmi realizace a s typy jednotlivých uzlů. Vzhledem k tomu, že tutoriály v této aplikaci tvoří návaznou posloupnost a podněcují k interaktivitě, lze tak dosáhnout lepšího pochopení ze stran studentů a ušetření času při výuce základní podstaty a smyslu této metody.

Kapitola 1

Cíle práce

Hlavním cílem této práce je navrhnout a vytvořit výukovou aplikaci, na které by se především studenti bakalářského studia vysokých škol naučili základy metody GERT, která se spolu s metodami PERT a CPM používá k odhadům doby trvání projektu, a to v závislosti na pravděpodobnostech realizace daných činností projektu. Studenti si na názorných příkladech s interaktivním grafem, jednoduchým popisem a navigací osvojí postupy, na kterých je metoda GERT založená, načež si sami budou moci vytvářet příklady vlastní. Jednotlivými cíli této práce tedy jsou:

- poskytnout teoretický úvod k projektům a k projektovému řízení
- poskytnout teoretický základ nutný k pochopení metod GERT, PERT a CPM, a to včetně matematických vzorců
- provést analýzu aplikace pro výuku metody GERT včetně analýzy požadavků a tvorby případů užití
- navrhnout architekturu této aplikace
- implementovat tuto aplikaci v rámci zvolených technologií
- vytvořit jednotlivé tutoriály s příklady pro metody GERT a CPM (na které je metoda GERT založená)
- vložit tyto tutoriály do aplikace
- otestovat funkčnost této aplikace
- zhodnotit stav výsledného řešení a možná další rozšíření

Kapitola 2

Teoretická část

2.1 Projektové řízení

Metody CPM, PERT a GERT, které jsou v této kapitole definovány a popsány, se uplatňují především v oblasti projektového řízení [1] při odhadu dob trvání jednotlivých činností projektu i projektu jako celku. Samotné projektové řízení se definuje jako plánování organizování a řízení činností včetně jejich zdrojů v rámci uceleného projektu, zatímco se respektují časová, zdrojová a nákladová omezení. Projekt je v této souvislosti souhrnem vzájemně provázaných činností, které je třeba provést k dosažení předem stanoveného cíle. Činnosti představují základní stavební kámen projektu, který má předem definovanou dobu trvání. Tato doba trvání pak tvoří základ výše uvedených metod a pro její určení je nutné vytvořit správný odhad. Při odhadu dob trvání jednotlivých činností je nutné uvažovat množství zdrojů, které jsou k vykonání dané činnosti potřebné, spolu s jejich dostupností. Odhady samotné se tvoří na základě odborného úsudku, simulace, podobností apod. Nejčastější postupy tvorby odhadů jsou následující [2]:

- odhad na základě osobní zkušenosti (*best guess*)
- odhad prováděný definovaným počtem prokazatelných expertů (*expert guess*)
- odhad na základě norem
- odhad na základě dokumentace předchozích projektů
- dále například parametrické odhadování apod.

Další oblastí, kterou je potřeba brát pří řízení projektů v potaz, je samotný kontext projektu neboli vazby s jeho okolním prostředím. Takové vazby jsou ovlivněny tím, kdo je původcem projektu, co je jeho výstupem a cílem, jaká je

jeho velikost a pochopitelně i obecné legislativní uspořádání [2]. Projektové řízení využívá celou řadu metodik a nástrojů, jak projekt a jeho kontext správně definovat, zanalyzovat a provést. Mezi takové nástroje se počítá například takzvaný logický rámec projektu, který slouží jako pomůcka při stanovování a dosahování jednotlivých cílů projektu, SWOT analýza, která slouží především k analýze strategie, kterou daný projekt naplňuje, řízení rizik v projektu apod. Fáze řízení samotného projektu lze rozdělit na:

- předprojektovou neboli definiční fázi
- projekt (jeho zahájení, přípravu, realizaci a ukončení)
- poprojektovou (vyhodnocovací) fázi

Účelem předprojektové fáze je posoudit proveditelnost vlastního projektu. V této fázi se zpracovávají potřebné analýzy a studie. Poté následuje vlastní zahájení, příprava neboli plánování projektu, jeho realizace a konečně jeho ukončení. V poprojektové fázi se pak vyhodnocují poznatky a zkušenosti, které daný projekt přinesl. Kromě zjištění chyb pro jejich následné neopakování se tyto poznatky využívají právě i ke tvorbě odhadů dob trvání činností příštích projektů. Projektové řízení samo o sobě obsahuje mnoho metodik a prolíná se i s jinými oblastmi, například s krizovým řízením a se řízením personálním. Pro zasazení metod CPM, PERT a GERT do potřebného kontextu předchozí vysvětlení však již stačí.

2.2 Výuková aplikace

Základním smyslem aplikačního software určeného pro výuku je bezpochyby usnadnit samotný výukový proces, který se týká daného tématu podporovaného aplikací [3]. Studenti si interaktivně dle notoricky známé zásady Jana Ámose Komenského „škola hrou“ projdou otázkami a teorií, nad kterou jsou nuceni kvůli vyžadované zpětné vazbě hlouběji přemýšlet a nacházet souvislosti. Následně prezentací vizuálně přívětivých příkladů, které si mohou studenti interaktivně rovněž upravovat, si danou problematiku snáze zapamatují. Realizace takové výukové aplikace pak závisí na tom, pro jaké téma, typ studentů a technologickou platformu má být určená.

Specifičnost výukové aplikace pro studenty bakalářského studia tkví především v její otevřenosti nejrůznějším úpravám. To se týká hlavně rozšíření (pluginů) aplikace a samozřejmě i tvorby obsahu (příklady, texty apod.). Pokud je navíc aplikace a její kód snadno pochopitelný a upravitelný, studenti informatických oborů si sami mohou přizpůsobovat vzhled a funkčnost aplikace, a tím opět uplatnit svou tvořivost a kreativitu ke snazšímu pochopení dané problematiky.

Vzhledem k tomu, že cílem této práce je implementace výukové aplikace pro metodu GERT, následuje teoretický úvod k této metodě a matematickým pojmům, které jsou pro její pochopení důležité.

2.3 Teorie grafů

Před vlastním popisem metod, kterými se tato práce zabývá, je důležitý krátký úvod do teorie grafů, na které jsou tyto metody založené. Graf G lze definovat jako uspořádanou dvojici $G = (V, E)$, kde V je neprázdná množina vrcholů a kde E je množina hran, tj. množina vybraných dvouprvkových podmnožin množiny V. Nechť jsou tedy vrcholy v a u prvky množiny V, pak řekneme, že jsou tyto vrcholy spojené hranou uv právě tehdy, když hrana $uv \in E(G)$ [4]. V teorii grafů existuje spousta pojmů, které je užitečné zadefinovat pro různé případy a příklady. Co se týče metod CPM, PERT a GERT, které budou popsány níže, zde je důležité především definovat následující [5]:

- *konečný graf* je takový graf, který má konečnou a neprázdnou množinu vrcholů
- *orientovaný graf* je takový graf $G = (V, E)$, kde V je množina vrcholů a E je množina orientovaných hran; orientovaná hrana uv pak není dvouprvková množina $\{u, v\}$, ale uspořádaná dvojice (u, v) dvou prvků $u, v \in V(G)$, kde vrchol u je počáteční vrchol a vrchol v je koncový vrchol hrany uv
- *acyklický graf* je takový graf, jehož žádný podgraf není cyklus; cyklem na n vrcholech se označuje graf takový, u kterého platí, že na n vrcholech (kde $n \geq 3$), které jsou po řadě spojeny n hranami tak, že každý vrchol je spojen s následujícím vrcholem, je navíc poslední vrchol spojený s prvním vrcholem

Kromě těchto pojmů je důležité rovněž pochopit tzv. souvislost grafu [5]. Jelikož cílem této práce je v této souvislosti poskytnout pouze krátký úvod, nebudu se zde pouštět do složitých definicí. K zjištění toho, co to znamená souvislý graf, je nutné zavést pojem sledu. Zjednodušeně lze říci, že sled mezi dvěma vrcholy u a v popisuje putování v daném grafu z vrcholu u do vrcholu v po hranách a vrcholech grafu. Potom řekneme, že vrchol v je dosažitelný z vrcholu u, pokud v grafu existuje sled z vrcholu u do vrcholu v. Navíc můžeme definovat speciální případ sledu, kterým je cesta. *Cesta* je pak orientovaný sled, ve kterém se neopakují žádné vrcholy (a tedy ani hrany). Hamiltonovská cesta je pak cesta, která prochází všemi vrcholy v grafu. Pochopení těchto pojmů následně usnadní pochopení metod, kterými se tato práce zabývá.

Nyní je možné přejít k samotné definici *souvislého grafu*. Graf nazveme souvislý, pokud pro každé dva vrcholy u a v je vrchol v dosažitelný z vrcholu

u (bez ohledu na orientaci hran). Všechny výše uvedené definice jsou pro další pochopení metod CPM, PERT a GERT zásadní. K vlastnímu provedení těchto metod, k získání vstupů a zapisování výstupů je nutné graf samotný doplnit o potřebné údaje. Proto navíc definujeme:

- *hranově ohodnocený graf* je takový graf, kdy je každé hraně přiřazeno alespoň jedno číslo
- *uzlově ohodnocený graf* je takový graf, kdy je každému uzlu přiřazeno alespoň jedno číslo

Teorie grafů je mnohem rozsáhlejší, ale tyto základní pojmy pro další pochopení postačí. Důležitý je však ještě jeden pojem, a to pojem síťového grafu, který si zaslouží vlastní sekci. V následujícím textu této práce bude jako synonymum k pojmu *vrchol grafu* uváděno rovněž označení *uzel grafu*.

2.4 Síťový graf

Pro definici následujících metod je důležitý pojem síťového grafu, který se používá jako matematický model projektu [6]. Síťový graf je konečný, souvislý, orientovaný (s hranami, které mají přiřazený určitý směr), acyklický graf s ohodnocenými hranami nebo uzly [7]. Při sestavování síťového grafu je především potřeba brát ohled na následující pravidla:

- síťový graf má právě jeden počáteční uzel
- síťový graf má právě jeden koncový uzel
- do každého uzlu s výjimkou počátečního musí vést alespoň jedna hrana
- z každého uzlu s výjimkou koncového musí vést alespoň jedna hrana
- libovolné dva uzly v grafu mohou být spojeny pouze právě jednou hranou

Pro reprezentaci projektů lze využít jak hranově, tak i uzlově definované síťové grafy. V prvním případě jsou jednotlivé činnosti, které tvoří součást projektu, představovány hranami, v druhém uzly síťového grafu. Nadále budu uvažovat pouze první případ, kdy jsou činnosti projektu představovány hranami. Uzly potom naopak představují jednotlivé události, kterými jsou začátky a konce jednotlivých činností. Do uzlů se zpravidla vepisují hodnoty veličin, které souvisejí s nadcházející činností (hranou).

2.5 Metoda kritické cesty

Metoda kritické cesty, anglicky Critical Path Method (dále jen „metoda CPM“), je deterministickou metodou síťové analýzy, která využívá tzv. kritickou cestu především k odhadu délky trvání projektů podle předem stanovených dob trvání jeho jednotlivých činností [8]. Budeme-li se na celou problematiku dívat z pohledu síťového grafu, potom jedna činnost odpovídá jedné hraně grafu (viz Síťový graf výše), která je ohraničena jejími počátečními a koncovými uzly. Posloupnost hran síťového grafu taková, kdy se koncový uzel každé hrany shoduje s počátečním uzlem hrany následující, se pak nazývá cesta. Součet dob trvání všech činností, které se na dané cestě nacházejí, označujeme jako dobu trvání cesty. Obecně pak lze říci, že kritická cesta je taková cesta od počátečního bodu grafu do koncového bodu grafu, která má nejdelší dobu trvání. Pro snazší pochopení nyní metodu CPM představím z praktického hlediska. V grafickém znázornění jednotlivé uzly v síťovém grafu zpravidla obsahují dvě následující hodnoty:

$T_i^{(0)}$ – nejdříve možný začátek činnosti (v síťovém grafu hrany), která vychází z uzlu i

$T_i^{(1)}$ – nejpozději přípustný začátek činnosti (v síťovém grafu hrany), která vychází z uzlu i

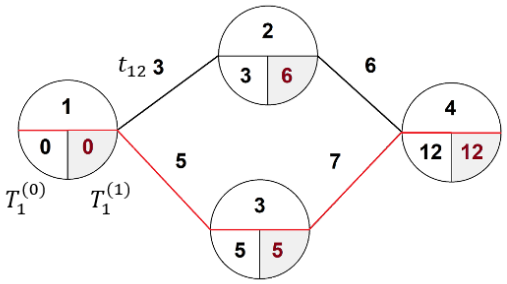
Nejdříve možný začátek činnosti $T_i^{(0)}$ se spočítá při jednoduchém průchodu síťovým grafem od počátečního bodu ke koncovému. Pro každou vstupující hranu do daného uzlu se spočítá doba tak, že se k z jejího počátečního uzlu zjištěné hodnotě $T_j^{(0)}$ přičte doba trvání dané činnosti t_{ji} mezi uzly j a i. Z těchto hodnot se pak zvolí ta největší z nich, která odpovídá nejdříve možnému začátku činnosti $T_i^{(0)}$. Matematicky to lze vyjádřit následovně:

$$T_i^{(0)} = \max_j (T_j^{(0)} + t_{ji}),\ T_1^{(0)} = 0 \tag{2.1}$$

Při zpětném průchodu od koncového bodu síťového grafu k počátečnímu lze ke každé činnosti zjistit její nejpozději přípustný začátek $T_i^{(1)}$. Pro každou vystupující hranu se doba spočítá jako rozdíl následujícího nejpozději přípustného začátku $T_k^{(1)}$ a doby trvání následující činnosti t_{ik} mezi uzly i a k. Z těchto hodnot se pak vybere ta nejmenší, která odpovídá zjištěnému nejpozději přípustnému začátku. Opět zde uvedu matematicky zápis, kde uzel s číslem n představuje koncový uzel grafu.

$$T_i^{(1)} = \min_k (T_k^{(1)} - t_{ik}),\ T_n^{(1)} = T_n^{(0)} \tag{2.2}$$

Rozdíl mezi nejpozději přípustným začátkem činnosti a jejím nejdříve možným začátkem nazýváme rezervou dané činnosti. Vrátíme-li se zpět k definici



Obrázek 2.1: Příklad grafické reprezentace metody CPM v síťovém grafu. Kritická cesta je znázorněna červeně. Šedě podbarvené hodnoty jsou spočítány při zpětném průchodu.

metody CPM, kritická cesta je taková cesta od počátečního bodu grafu do koncového bodu grafu, jejíž každá činnost má nulovou rezervu.

2.6 Metoda PERT

Zobecněním metody CPM je takzvaná Program Evaluation and Review Technique (dále jen „metoda PERT“). Doba trvání jednotlivých činností zaznamenaných v síťovém grafu není v metodě PERT na rozdíl od metody CPM pevně dána, nýbrž se chápe stochasticky jako náhodná proměnná [9]. Z toho vyplývá, že každá doba trvání činnosti má své rozložení pravděpodobnosti. V praxi se nejčastěji využívá takzvané beta rozdělení (anglicky beta distribution) z teorie pravděpodobnosti [10]. Jedná se o spojité rozdělení, jehož momenty se v metodě PERT vypočítávají na základě odhadů, které tvoří odborníci v daném poli působnosti, kde se metoda PERT uplatňuje. Tyto odhady jsou zpravidla tři [11], a to následující:

- **optimistický odhad** a, který uvažuje nejkratší dobu trvání činnosti s předpokladem, že nenastanou žádné okolnosti prodlužující dobu trvání dané činnosti
- **nejpravděpodobnější odhad (modus)** m, kde se jedná o nejpravděpodobnější hodnotu doby trvání dané činnosti
- **pesimistický odhad** b, který naopak vychází z předpokladu, že nastanou zvažované nepříznivé okolnosti a dojde tak k prodloužení doby trvání dané činnosti

Očekávaná doba trvání dané činnosti se pak spočítá podle následujícího jednoduchého vzorce [11]:

$$t_{ji} = \frac{a + 4m + b}{6} \tag{2.3}$$

Důležité pro výpočet následných pravděpodobností je znát rozptyl (variaci) dob trvání jednotlivých činností $\sigma^2_{t_{ji}}$. Ten lze s výše uvedenými proměnnými vypočítat následovně:

$$\sigma^2_{t_{ji}} = \left(\frac{(b-a)}{6}\right)^2 \tag{2.4}$$

Následně lze pak například spočítat pravděpodobnost dodržení plánované doby ukončení projektu T_P s využitím předpokládané doby ukončení projektu T_E, jejíž směrodatná odchylka σ_{T_E} se spočítá jako odmocnina součtu rozptylů dob trvání jednotlivých činností ($\sigma^2_{t_{ji}}$) spočítaných podle vzorce uvedeného výše. Známe-li tyto tři hodnoty, výslednou pravděpodobnost lze spočítat následovně:

$$P(T_E \leq T_P) = F\left(\frac{T_P - T_E}{\sigma_{T_E}}\right), \tag{2.5}$$

kde $F(x)$ je distribuční funkce normálního rozdělení

Detailnější vysvětlení a rozebrání této metody je mimo zaměření této práce. Proto doporučuji nahlédnout do zdrojů uvedených k této práci [9].

2.7 Metoda GERT

Graphical Evaluation and Review Technique (dále jen „metoda GERT“), kterou se tato práce zabývá, je stochastickým rozšířením metody CPM, nicméně se od ní a od metody PERT v mnohém odlišuje. Podobně jako metoda PERT, která místo deterministických dob trvání činností pracuje s dobami stochastickými, metoda GERT kromě deterministických uzlů síťového grafu pracuje i s uzly stochastickými [12]. V takovém případě se vystupující hrany z daného stochastického uzlu realizují v závislosti na jim přiřazené pravděpodobnosti. S pravděpodobností se však pracuje jak při výstupu, tak i při vstupu do jednotlivých uzlů. Zde rozlišujeme tři základní typy:

- **konjunktivní uzel**, který se realizuje v případě, že se realizují všechny do něj vstupující hrany
- **inkluzivní uzel**, který se realizuje v případě, že se realizuje alespoň jedna vstupující hrana
- **disjunktivní uzel**, který se realizuje v případě, kdy se realizuje právě jedna vstupující hrana

Rozdílem oproti prosté metodě CPM je tedy zjišťování hodnoty pravděpodobnosti realizace daného uzlu $P(i)$, a to pro každý uzel v síťovém grafu. Dalším rozdílem je i samotný výpočet hodnot nejdříve možného začátku činnosti $T_i^{(0)}$ a nejpozději přípustného začátku činnosti $T_i^{(1)}$ (viz Metoda kritické cesty výše), kde se nyní zohledňují pravděpodobnosti. Pro každý uzel se tedy spočítají hodnoty tří veličin, a to $T_i^{(0)}$, $T_i^{(1)}$ a $P(i)$, následovně:

Pro konjunktivní uzly:

$$T_i^{(0)} = \max_j (T_j^{(0)} + t_{ji} \times P(j) \times p(j,i)),\ T_1^{(0)} = 0 \tag{2.6}$$

$$T_i^{(1)} = \min_k (T_k^{(1)} - t_{ik} \times P(i) \times p(i,k)),\ T_n^{(1)} = T_n^{(0)} \tag{2.7}$$

$$P(i) = \prod P(j) \times p(j,i), \tag{2.8}$$

kde $p(j,i)$ je zadaná pravděpodobnost realizace hrany (j,i)

Pro inkluzivní uzly:

$$T_i^{(0)} = \max_j (T_j^{(0)} + t_{ji} \times P(j) \times p(j,i)),\ T_1^{(0)} = 0 \tag{2.9}$$

$$T_i^{(1)} = \min_k (T_k^{(1)} - t_{ik} \times P(i) \times p(i,k)),\ T_n^{(1)} = T_n^{(0)} \tag{2.10}$$

$$P(i) = 1 - \prod \left(1 - P(j) \times p(j,i)\right), \tag{2.11}$$

kde $p(j,i)$ je zadaná pravděpodobnost realizace hrany (j,i)

Pro disjunktivní uzly:

$$T_i^{(0)} = \max_j (T_j^{(0)} + t_{ji} \times P(j) \times p(j,i)),\ T_1^{(0)} = 0 \tag{2.12}$$

$$T_i^{(1)} = \min_k (T_k^{(1)} - t_{ik} \times P(i) \times p(i,k)),\ T_n^{(1)} = T_n^{(0)} \tag{2.13}$$

$$P(i) = \sum P(j) \times p(j,i) \times \prod \left(1 - P(k) \times p(k,i)\right), \text{ kde } k \neq j \tag{2.14}$$

a $p(j,i)$ je zadaná pravděpodobnost realizace hrany (j,i)
a $p(k,i)$ je zadaná pravděpodobnost realizace hrany (k,i)

V koncovém uzlu síťového grafu se pak nachází hodnota pravděpodobnosti realizace uzlu $P(n)$ odpovídající pravděpodobnosti realizace celého projektu podle nastavených (či odhadnutých) pravděpodobností realizací jednotlivých hran/činností.

Jak je možné vidět z výše uvedených vzorců, pravděpodobností jsou ovlivněny i začátky jednotlivých činností $T_i^{(0)}$, $T_i^{(1)}$, což vede k lepšímu odhadu na dobu trvání celého projektu, než pokud by se pracovalo s čistě deterministickými hodnotami.

2.8 Ekonomické zhodnocení metody GERT

Metoda CPM vychází pouze ze základních dob trvání jednotlivých činností projektu a jejich vzájemných návazností, což jí ve srovnání s ostatními metodami poskytuje nízkou analytickou přesnost. Jistého pokroku dosahuje metoda PERT, která díky využitým optimistickým, nejpravděpodobnějším a pesimistickým odhadům na doby trvání jednotlivých činností umožňuje lépe vystihnout stochastickou podstatu doby trvání celého projektu. Metoda GERT může metodu PERT ve své nejjednodušší podobě simulovat, pokud se využijí pouze deterministické konjunktivní uzly. Ovšem na rozdíl od metody PERT může metoda GERT modelovat složité stochastické projekty s různými podobami závislostí jednotlivých činností na jejich předchůdcích, a to s ohledem na pravděpodobnost jejich realizace. Metoda GERT tedy v porovnání s ostatními zmíněnými metodami představuje nejobecnější a nejpřesnější způsob jak odhadnout dobu trvání celého projektu.

Simulace efektivnosti metody GERT provedená institutem Project Management Institute [12] ukázala, že metoda GERT je především citlivá na změnu pravděpodobnosti realizace jednotlivých činností zvoleného projektu spíše než na změnu doby trvání daných činností. Z toho vyplývá, že je nutné hledat nejvhodnější rozdělení pravděpodobnosti při tvorbě odhadů, než pouze jednoduše přijmout beta rozdělení, jako je tomu v případě metody PERT. Management je tak nucen provádět podrobnější výzkum jednotlivých činností, což vede automaticky k lepším odhadům zjišťovaných veličin a obecně k lepší analýze celého projektu.

Kapitola 3

Analýza a návrh

3.1 Procesy a požadavky

Smyslem práce je navrhnout a implementovat výukovou aplikaci, která by studenty interaktivně provedla základy metody GERT na příkladu projektů představovaných síťovými grafy. Následující procesy, které bude aplikace podporovat, vznikly ve spolupráci s vedoucí práce doktorkou Petrou Pavlíčkovou. Tyto základní procesy jsou následující:

- graficky a po krocích ukázat postupný výpočet pomocí metody CPM, a to směrem ke koncovému uzlu (počítání nejdříve možných začátků činností $T_i^{(0)}$), i směrem k uzlu počátečnímu (počítání nejpozději přípustných začátků činností $T_i^{(1)}$), a to s využitím jednoduchých konjunktivních deterministických uzlů
- graficky a po krocích ukázat postupný výpočet pomocí metody GERT oběma výše zmíněnými směry spolu s výpočtem pravděpodobností realizace jednotlivých uzlů, a to s využitím deterministických i stochastických konjunktivních, inkluzivních a disjunktivních uzlů
- umožnit uživateli vkládat jednotlivé uzly a hrany do síťového grafu
- umožnit uživateli měnit ohodnocení hran (doby trvání činností, pravděpodobnosti realizace činností)
- umožnit uživateli exportovat a importovat zadané síťové grafy

3.1.1 Funkční požadavky

Po detailnějším rozpracování jednotlivých procesů, které by měla vyvíjená aplikace podporovat, vznikly následující **funkční** požadavky na výukovou aplikaci.

3.1.1.1 Požadavky na zobrazení tutoriálů

- aplikace bude ze souborů načítat vytvořené tutoriály pro metodu GERT a metodu CPM
- aplikace bude uživateli umožňovat výběr jednotlivých tutoriálů, a to z hlavního menu a na úvodní obrazovce, kde ke každému bude uveden i krátký popis
- aplikace bude umožňovat návrat na úvodní obrazovku pomocí položky v hlavním menu
- aplikace bude zobrazovat jednotlivé vybrané tutoriály, kde jeden tutoriál bude rozdělen na několik stránek, mezi nimiž uživatel bude schopen přepínat pomocí šipek
- aplikace bude umožňovat nejen přepnutí na další stránku tutoriálů, ale i na stránku předchozí
- aplikace bude zobrazovat každou stránku tutoriálů s popisem a instrukcemi k dané problematice
- aplikace bude uživateli umožňovat tyto popisy interaktivně měnit
- aplikace bude na každé stránce tutoriálu zobrazovat interaktivní graf

3.1.1.2 Požadavky na grafy

- aplikace načte příslušný graf z daného tutoriálu
- aplikace v grafu zobrazí jednotlivé předdefinované uzly, hrany a jejich ohodnocení
- aplikace u každého uzlu vypočítá a zobrazí jeho identifikační číslo, nejdříve možný začátek následujících činností $T_i^{(0)}$, nejpozději přípustný začátek následujících činností $T_i^{(1)}$ a v případě metody GERT i pravděpodobnost realizace daného uzlu
- aplikace u každé hrany zobrazí dobu trvání dané činnosti, kterou daná hrana představuje, a pravděpodobnost realizace
- aplikace bude uživateli umožňovat s jednotlivými uzly pohybovat
- aplikace bude uživateli umožňovat přidávat jednotlivé uzly
- aplikace bude uživateli umožňovat přidávat jednotlivé hrany spolu s jejich ohodnocením (doba trvání dané činnosti a v případě metody GERT i pravděpodobnost realizace dané činnosti)

- aplikace bude uživateli umožňovat měnit jednotlivé uzly, tj. editovat jejich identifikační číslo a v případě metody GERT i jejich typ (konjunktivní, inkluzivní a disjunktivní)
- aplikace bude uživateli umožňovat měnit jednotlivé hrany, tj. odkud a kam směřují a jejich ohodnocení

3.1.1.3 Požadavky na uložení tutoriálů

- aplikace bude uživateli umožňovat daný tutoriál se všemi jeho změnami uložit do souboru, a to pomocí položky z hlavního menu a klávesové zkratky
- aplikace uživatele vyzve k uložení souboru, pokud provedl nějaké změny
- aplikace bude poskytovat srozumitelný formát souborů, který lze snadno měnit pomocí textového editoru
- aplikace bude schopná tento soubor načíst jako vlastní tutoriál

3.1.1.4 Požadavky na obsah tutoriálů

- aplikace bude ve výchozím nastavení obsahovat čtyři tutoriály, kde první z nich uživateli vysvětlí způsob práce s aplikací, druhý objasní metodu CPM, třetí objasní metodu GERT, a čtvrtý bude obsahovat složitější příklad pro metodu GERT
- aplikace bude v prvních třech základních tutoriálech obsahovat alespoň tři stránky
- aplikace bude v každém z těchto základních tutoriálů přecházet od velmi snadno pochopitelného příkladu ke složitějšímu, a to v posloupnosti jednotlivých stránek
- aplikace bude v tutoriálu se složitějším příkladem pro metodu GERT dbát na interaktivitu a podněcovat uživatele k přidávání vlastních uzlů a hran

3.1.2 Nefunkční požadavky

Po zpracování jednotlivých procesů a pochopení způsobu fungování aplikace vyvstaly následující **základní nefunkční** požadavky, které zohledňují především snadnou dostupnost aplikace na různých platformách a její případnou rozšiřitelnost zájemci z řad studentů. Tyto požadavky jsou následující:

- aplikace bude napsána pro webový prohlížeč
- aplikace bude nezávislá na externím serveru a internetovém připojení

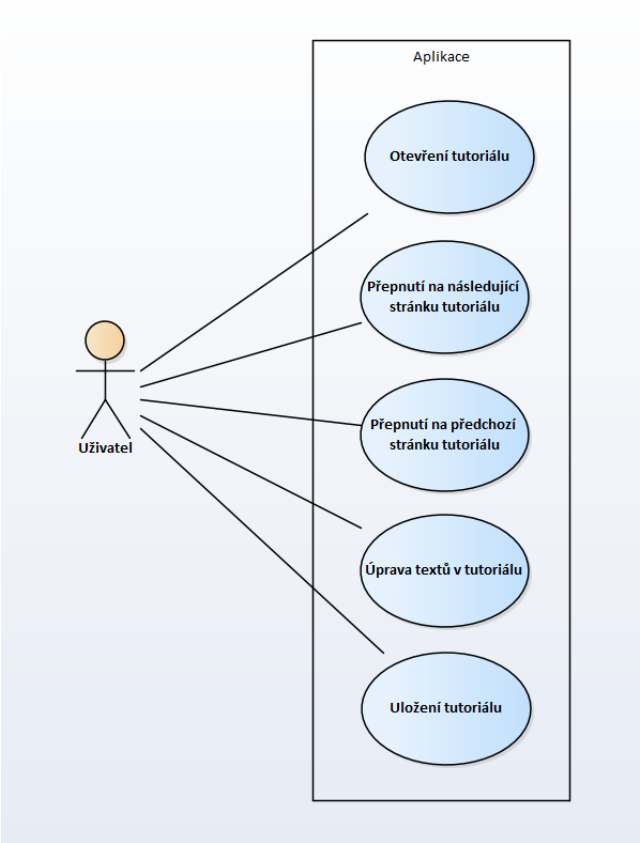
- aplikace bude jednotlivé tutoriály načítat ze složky ve svých zdrojích, pouze s omezeními danými zvolenou technologií
- aplikace zobrazí jednotlivé tutoriály i s jejich grafy maximálně do 2 sekund
- aplikace zobrazí úvodní obrazovku pro výběr tutoriálu maximálně do 3 sekund
- aplikace bude srozumitelně napsaná pro případnou rozšiřitelnost
- aplikace bude snadno instalovatelná

3.2 Případy užití

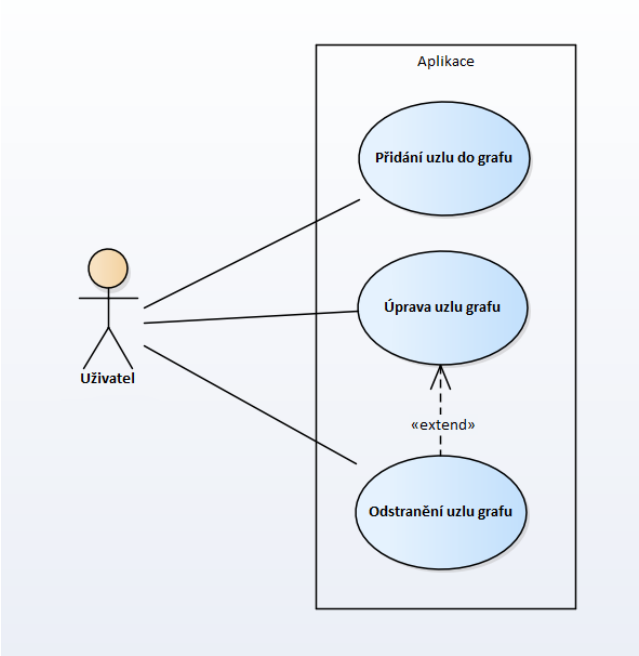
Jazyk UML („Unified Modeling Language“), který slouží pro navrhování programových systémů, definuje pojem případ užití [13]. Případ užití charakterizuje určité pojmenované použití daného systému uživatelem. Případ užití může být znázorněn graficky v diagramu či detailněji specifikován v rámci takzvaných scénářů, kde je přímo popsána interakce mezi uživatelem a systémem. Na odchylky od hlavního scénáře případu užití pak mohou reagovat jeho alternativní scénáře [14], které řeší jeho nestandardní verze. Případy užití jsou vytvořeny rozpracováním a mapováním jednotlivých požadavků na systém.

K detailnímu provedení celé analýzy jsem pak z jednotlivých požadavků na aplikaci detailně zpracoval jednotlivé případy užití, definované jejich scénáři. Případy užití tedy přímo navazují na funkční požadavky aplikace a podrobněji zpracovávají interakci mezi uživatelem a aplikací. Následující jednoduché diagramy, vytvořené v programu Enterprise Architect, zobrazují nejdůležitější případy užití rozdělené do tří částí:

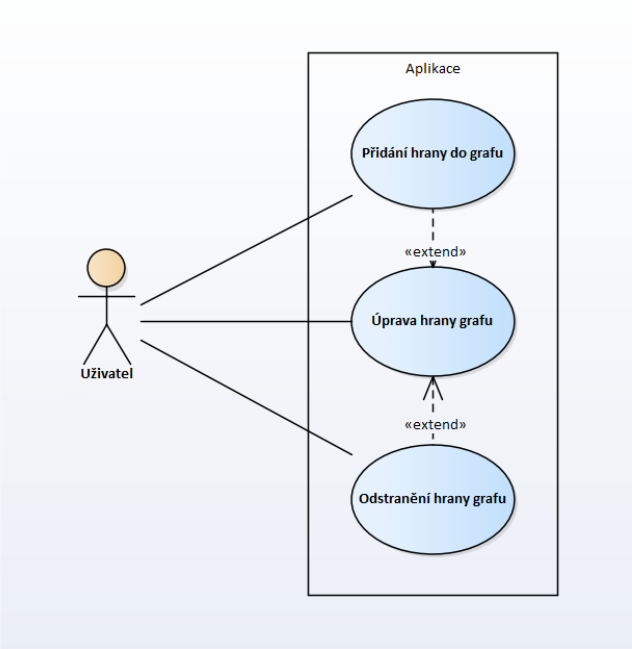
- případy užití týkající se použití tutoriálů
- případy užití týkající se uzlů v grafu
- případy užití týkající se hran v grafu



Obrázek 3.1: Diagram případů užití tykajících se použití tutoriálů



Obrázek 3.2: Diagram případů užití tykajících se uzlů v grafu



Obrázek 3.3: Diagram případů užití tykajících se hran v grafu

3.2.1 Otevření tutoriálu

1	Aplikace uživateli zobrazí úvodní obrazovku s detaily jednotlivých tutoriálů, kde se zobrazí název daného tutoriálu spolu s jeho krátkým popisem a tlačítkem pro otevření. Jednotlivé detaily jsou od sebe vizuálně odděleny.
2	Uživatel zvolí daný tutoriál kliknutím na tlačítko pro otevření.
3	Aplikace zobrazí první (předem definovanou) stránku daného tutoriálu spolu s popisy a s grafem, ve kterém již bude graficky znázorněn proběhlý výpočet pomocí metod GERT/CPM.

Tabulka 3.1: **Hlavní scénář 1:** Otevření tutoriálu z úvodní obrazovky

1	Uživatel otevře hlavní menu pomocí tlačítka pro jeho zobrazení.
2	Aplikace pod tlačítky pro návrat na úvodní obrazovku a pro uložení souboru zobrazí názvy jednotlivých tutoriálů.
3	Uživatel klikne na název zvoleného tutoriálu.
4	Aplikace zavře hlavní menu a zobrazí první (předem definovanou) stránku nově zvoleného tutoriálu spolu s popisy a s grafem, ve kterém již bude graficky znázorněn proběhlý výpočet pomocí metod GERT/CPM.

Tabulka 3.2: **Hlavní scénář 2:** Otevření jiného tutoriálu ze současného přímo

1	Uživatel otevře hlavní menu pomocí tlačítka pro jeho zobrazení.
2	Aplikace zobrazí hlavní menu s tlačítkem pro návrat na úvodní obrazovku, s tlačítkem pro uložení souboru a s názvy jednotlivých tutoriálů.
3	Uživatel zvolí tlačítko pro návrat na úvodní obrazovku.
	Následující postup je stejný jako v případě hlavního scénáře „Otevření tutoriálu z úvodní obrazovky“.

Tabulka 3.3: **Hlavní scénář 3:** Otevření jiného tutoriálu ze současného přes úvodní obrazovku

3.2.2 Přepnutí na následující stránku tutoriálu

1	Uživatel v otevřeném tutoriálu klikne na symbol šipky, který se nachází po pravé straně grafu. **Pokud** symbol šipky není aktivní, následující body se neprovedou.
2	Aplikace načte soubor s následující stránkou tutoriálu.
3	Aplikace zobrazí následující stránku tutoriálu spolu se všemi texty a s grafem podle dat ze souboru.

Tabulka 3.4: **Hlavní scénář:** Přepnutí na následující stránku tutoriálu

3.2.3 Přepnutí na předchozí stránku tutoriálu

1	Uživatel v otevřeném tutoriálu klikne na symbol šipky, který se nachází po levé straně grafu. **Pokud** symbol šipky není aktivní, následující body se neprovedou.
2	Aplikace načte soubor s předchozí stránkou tutoriálu.
3	Aplikace zobrazí předchozí stránku tutoriálu spolu se všemi texty a s grafem podle dat ze souboru.

Tabulka 3.5: **Hlavní scénář:** Přepnutí na předchozí stránku tutoriálu

3.2.4 Úprava textů v tutoriálu

1	Uživatel v otevřeném tutoriálu dvakrát klikne na vybraný popisný text. Tyto texty se nachází: • pod hlavním menu, kde se jedná o název stránky tutoriálu • nad grafem, který slouží pro uvedení uživatele do dané problematiky • na levé straně grafu, který slouží k hlubšímu vysvětlení tématu, popřípadě k popisu předchozí stránky tutoriálu • na pravé straně grafu, který slouží k hlubšímu vysvětlení tématu, popřípadě k popisu následující stránky tutoriálu • pod grafem, který slouží k popisu příkladu v grafu a k uvedení závěru
2	Aplikace zobrazí dialog pro úpravu zvoleného textu.
3	Uživatel klikne na tlačítko „Uložit“. **Alternativní:** 3a Uživatel provede příkaz k zavření dialogu úpravy textu **Návrat:** End
4	Aplikace zavře dialog a na příslušném místě v tutoriálu zobrazí upravený text.

Tabulka 3.6: **Hlavní scénář:** Úprava textů v tutoriálu

1	Pokud uživatel provedl příkaz k zavření dialogu pro úpravu textu kliknutím na symbol křížku nebo na tlačítko „Zavřít“, aplikace dialog zavře a neprovede žádnou změnu v tutoriálu.

Tabulka 3.7: **Alternativní scénář:** Uživatel provede příkaz k zavření dialogu úpravy textu

3.2.5 Uložení tutoriálu

1	Uživatel otevře hlavní menu pomocí tlačítka pro jeho zobrazení.
2	Aplikace zobrazí hlavní menu s tlačítkem pro uložení souboru podobně jako s tlačítkem pro návrat na úvodní obrazovku a s názvy jednotlivých tutoriálů.
3	Uživatel zvolí tlačítko pro uložení souboru.
3	Aplikace zavře hlavní menu a nechá uživatele stáhnout soubor s definicí stránky tutoriálu, kde se zároveň nachází všechny popisné texty a údaje potřebné k zobrazení grafu.

Tabulka 3.8: **Hlavní scénář 1:** Uložení tutoriálu pomocí položky v hlavním menu

1	Uživatel na příslušné stránce tutoriálu klikne na klávesu *S*.
2	Aplikace nechá uživatele stáhnout soubor s definicí stránky tutoriálu, kde se zároveň nachází všechny popisné texty a údaje potřebné k zobrazení grafu.

Tabulka 3.9: **Hlavní scénář 2:** Uložení tutoriálu pomocí klávesové zkratky

3.2.6 Přidání uzlu do grafu

1	Uživatel dvakrát klikne na volné místo v grafu, kam chce přidat nový uzel.
2	Aplikace na daném místě v grafu vytvoří a zobrazí výchozí případ uzlu takový, že: • identifikační číslo uzlu je nastaveno na o jedno větší, než je nejvyšší identifikační číslo z již existujících uzlů • typ uzlu je nastaven na konjunktivní • k uzlu nejsou připojeny žádné hrany
3	Aplikace přepočítá a zobrazí hodnoty v grafu závislé na výpočtu pomocí metody GERT/CPM.

Tabulka 3.10: **Hlavní scénář:** Přidání uzlu do grafu

3.2.7 Úprava uzlu grafu

1	Uživatel dvakrát klikne na příslušný uzel v grafu, který chce upravit.
2	Aplikace otevře dialog pro úpravu uzlu s možností zadání nového identifikačního čísla uzlu a výběru typu uzlu (konjunktivní, inkluzivní a disjunktivní).
3	Uživatel změní identifikační číslo uzlu nebo jeho typ.
4	Uživatel klikne na tlačítko „Uložit“. **Alternativní:** 4a Uživatel provede příkaz k zavření dialogu úpravy uzlu **Alternativní:** 4b Uživatel zadá již existující identifikační číslo jiného uzlu **Návrat:** End
5	Aplikace zavře dialog a zobrazí změněné hodnoty uzlu v grafu. Následně přepočítá a zobrazí hodnoty v grafu závislé na výpočtu pomocí metody GERT/CPM.

Tabulka 3.11: **Hlavní scénář:** Úprava uzlu grafu

1	Pokud uživatel provedl příkaz k zavření dialogu pro úpravu uzlu kliknutím na symbol křížku nebo na tlačítko „Zavřít“, aplikace dialog zavře a neprovede žádnou změnu v grafu.

Tabulka 3.12: **Alternativní scénář:** Uživatel provede příkaz k zavření dialogu úpravy uzlu

1	Pokud uživatel vyplnil již existující identifikační číslo jiného uzlu, aplikace uživatele upozorní pomocí chybové hlášky: „Uzel s tímto číslem již existuje!“, načež neprovede žádnou změnu a dialog ponechá otevřený.

Tabulka 3.13: **Alternativní scénář:** Uživatel zadá již existující identifikační číslo jiného uzlu

3.2.8 Odstranění uzlu grafu

1	Uživatel dvakrát klikne na příslušný uzel v grafu, který chce odstranit.
2	Aplikace otevře dialog pro úpravu uzlu s přidanou možností kliknutí na tlačítko „Odstranit“.
3	Uživatel klikne na tlačítko „Odstranit“.
4	Aplikace zavře dialog a zobrazí graf bez odstraněného uzlu a bez jeho vstupních i výstupních hran. Následně přepočítá a zobrazí hodnoty v grafu závislé na výpočtu pomocí metody GERT/CPM.

Tabulka 3.14: **Hlavní scénář:** Odstranění uzlu grafu

3.2.9 Přidání hrany do grafu

1	Uživatel jednou klikne na příslušný uzel v grafu, ze kterého má vést nová hrana.
2	Aplikace zobrazí křížek místo dosavadního kurzoru myši.
3	Uživatel jednou klikne na příslušný uzel v grafu, do kterého má nová hrana vést. **Alternativní:** 3a Uživatel klikne do prázdného místa v grafu **Návrat:** End
4	Aplikace místo křížku zobrazí výchozí kurzor myši.
5	Aplikace následně otevře dialog pro vytvoření nové hrany s možností zadání doby trvání dané činnosti, kterou nová hrana reprezentuje, a v případě metody GERT i s možností zadání pravděpodobnosti realizace dané činnosti. **Alternativní:** 5a Hrana již existuje **Návrat:** End
6	Uživatel vyplní dobu trvání dané činnosti (popř. i pravděpodobnost realizace).
7	Uživatel klikne na tlačítko „Uložit“. **Alternativní:** 7a Uživatel provede příkaz k zavření dialogu přidání/úpravy hrany **Návrat:** End
8	Aplikace zavře dialog a zobrazí novou hranu v grafu, která vede mezi příslušnými zvolenými uzly, spolu s dobou trvání a pravděpodobností realizace. Následně přepočítá a zobrazí hodnoty v grafu závislé na výpočtu pomocí metody GERT/CPM.

Tabulka 3.15: **Hlavní scénář:** Přidání hrany do grafu

1	Pokud uživatel klikl do prázdné oblasti v grafu mimo uzly, aplikace zobrazí místo křížku výchozí kurzor myši a neprovede žádnou úpravu.

Tabulka 3.16: **Alternativní scénář:** Uživatel klikne do prázdného místa v grafu

1	Pokud aplikace zjistila, že daná hrana již mezi zvolenými uzly existuje, a to bez ohledu na její orientaci, zobrazí dialog pro úpravu existující hrany.
2	Uživatel vyplní novou dobu trvání dané činnosti nebo její pravděpodobnost realizace.
3	Uživatel klikne na tlačítko „Uložit“. **Alternativní:** 3a Uživatel provede příkaz k zavření dialogu přidání/úpravy hrany **Návrat:** End
4	Aplikace zavře dialog a zobrazí v grafu nové hodnoty doby trvání činnosti a pravděpodobnosti realizace. Následně přepočítá a zobrazí hodnoty v grafu závislé na výpočtu pomocí metody GERT/CPM.

Tabulka 3.17: **Alternativní scénář:** Hrana již existuje

1	Pokud uživatel provedl příkaz k zavření dialogu pro přidání/úpravu hrany kliknutím na symbol křížku nebo na tlačítko „Zavřít“, aplikace dialog zavře a neprovede žádnou změnu v grafu.

Tabulka 3.18: **Alternativní scénář:** Uživatel provede příkaz k zavření dialogu přidání/úpravy hrany

3.2.10 Úprava hrany grafu

1	Uživatel jednou klikne na příslušný uzel v grafu, ze kterého vede hrana, která má být upravena.
2	Aplikace zobrazí křížek místo dosavadního kurzoru myši.
3	Uživatel jednou klikne na příslušný uzel v grafu, do kterého upravovaná hrana vede. **Alternativní:** 3a Uživatel klikne do prázdného místa v grafu **Návrat:** End
4	Aplikace místo křížku zobrazí výchozí kurzor myši.
5	Aplikace následně otevře dialog pro úpravu existující hrany s možností úpravy doby trvání dané činnosti, kterou hrana reprezentuje, a v případě metody GERT i s možností úpravy pravděpodobnosti realizace dané činnosti.
6	Uživatel vyplní dobu trvání dané činnosti (popř. i pravděpodobnost realizace).
7	Uživatel klikne na tlačítko „Uložit“. **Alternativní:** 7a Uživatel provede příkaz k zavření dialogu úpravy hrany **Návrat:** End
8	Aplikace zavře dialog a zobrazí v grafu nové hodnoty doby trvání činnosti a pravděpodobnosti realizace. Následně přepočítá a zobrazí hodnoty v grafu závislé na výpočtu pomocí metody GERT/CPM.

Tabulka 3.19: **Hlavní scénář:** Přidání hrany do grafu

1	Pokud uživatel klikl do prázdné oblasti v grafu mimo uzly, aplikace zobrazí místo křížku výchozí kurzor myši a neprovede žádnou úpravu.

Tabulka 3.20: **Alternativní scénář:** Uživatel klikne do prázdného místa v grafu

1	Pokud uživatel provedl příkaz k zavření dialogu pro přidání/úpravu hrany kliknutím na symbol křížku nebo na tlačítko „Zavřít“, aplikace dialog zavře a neprovede žádnou změnu v grafu.

Tabulka 3.21: **Alternativní scénář:** Uživatel provede příkaz k zavření dialogu úpravy hrany

3.2.11 Odstranění hrany grafu

1	Uživatel jednou klikne na příslušný uzel v grafu, ze/do kterého vede hrana, kterou chce odstranit.
2	Aplikace zobrazí křížek místo dosavadního kurzoru myši.
3	Uživatel jednou klikne na příslušný uzel v grafu, do/ze kterého vede hrana, kterou chce odstranit. **Alternativní:** 3a Uživatel klikne do prázdného místa v grafu **Návrat:** End
4	Aplikace místo křížku zobrazí výchozí kurzor myši.
5	Aplikace následně otevře dialog pro úpravu hrany s přidanou možností kliknutí na tlačítko „Odstranit“
6	Uživatel klikne na tlačítko „Odstranit“.
7	Aplikace zavře dialog a zobrazí graf bez odstraněné hrany. Následně přepočítá a zobrazí hodnoty v grafu závislé na výpočtu pomocí metody GERT/CPM.

Tabulka 3.22: **Hlavní scénář:** Odstranění uzlu grafu

1	Pokud uživatel klikl do prázdné oblasti v grafu mimo uzly, aplikace zobrazí místo křížku výchozí kurzor myši a neprovede žádné odstranění.

Tabulka 3.23: **Alternativní scénář:** Uživatel klikne do prázdného místa v grafu

3.3 Shrnutí

Z procesů, které má aplikace podporovat, přímo vyplývají funkční požadavky. Ty se týkají jak rozvržení vlastní aplikace, tak i základní funkčnosti a podoby tutoriálů, které jsou dále děleny na jednotlivé stránky. Funkční požadavky dále specifikují, že bude mezi jednotlivými stránkami tutoriálu možno přepínat navigačními šipkami a že každá stránka bez výjimky bude obsahovat své vlastní textové popisy a svůj vlastní graf s předem definovanými uzly a hranami. Každý uživatel pak do grafu bude schopný přidat vlastní uzly a hrany, upravovat textové popisy, typy a identifikační čísla uzlů a doby trvání a pravděpodobnosti realizace činností představovaných jednotlivými hranami. Uzly a hrany v grafu bude možné odstraňovat, s uzly pak i interaktivně pohybovat. Co se týče samotné aplikace, ta bude umožňovat uložení zvolené stránky tutoriálu včetně všech úprav, aby o ně uživatel nepřišel. Kromě toho bude obsahovat úvodní stránku s výběrem tutoriálů a hlavní menu s toutéž funkčností. Kritický pak z hlediska nefunkčních požadavků je přístup k aplikaci z webového prohlížeče, snadná instalace a rychlá odezva na interaktivní změny.

Rozpracované nejdůležitější případy užití spolu se svými hlavními a popřípadě i alternativními scénáři pak dále tyto požadavky specifikují na úroveň detailu takovou, aby bylo možné popsat interakci mezi uživatelem a aplikací. Vlastní diagram případů užití jsem rozdělil na tři části: na část týkající se použití tutoriálů, na část týkající se uzlů v grafu a na část týkající se hran v grafu. Samotné případy užití na sobě závisí jen minimálně, avšak především ty, které se týkají otevření a uložení tutoriálů, nabízejí různé scénáře, jak dosáhnout stejného výsledku (viz výše). Z analýzy pomocí případů užití pak především v případě hran v grafu vyplývá vzájemné provázání mezi dialogem pro tvorbu hran a pro jejich úpravu, stejně tak jako přidání funkce pro odstranění. Po každé změně v grafu se pak znovu provede výpočet pomocí metody GERT či metody CPM a graf samotný se překreslí. Alternativní scénáře případů užití zpravidla nastávají, pokud uživatel zavřel dialog (např. pro úpravu hrany) či zadal nesprávný/nepřijatelný údaj.

Dosud nebyly specifikovány konkrétní technologie a formáty souborů. Ty budou určeny až po návrhu architektury.

Kapitola 4

Implementace

4.1 Architektura

4.1.1 Návrh architektury a základní abstrakce

Z analýzy, kterou jsem prezentoval v předchozí kapitole, vyplynula následující základní fakta týkající se toho, co aplikace bude z pohledu uživatele provádět. Nutné je při nich vzít v potaz vzájemné provázání jednotlivých činností, které představují, a jejich návaznost, čemuž odpovídá následující pořadí:

1. aplikace reaguje na uživatelské vstupy (kliknutí na odkaz, reakce na události vyvolané myší v grafu, použití klávesové zkratky apod.)

2. aplikace vypočítá hodnoty určených veličin uzlů a hran v grafu, případně načte či uloží soubor s tutoriálem

3. aplikace překreslí a zobrazí graf v tutoriálu, případně celý tutoriál včetně všech popisných textů, či dialog

Tato základní fakta v tomto pořadí z obecného hlediska přímo reprezentují abstrakci všech ostatních funkcí aplikace, které pak tato fakta dále konkrétně specifikují. Po důkladném zvážení různých úskalí jsem dospěl k rozhodnutí použít na teoretické úrovni softwarovou architekturu MVC [15], která logiku aplikace rozděluje do tří nezávislých komponent:

- komponenta **controller**, která spravuje řídící logiku aplikace (reaguje na uživatelské vstupy, komunikuje s dalšími komponentami apod.)

- komponenta **model**, která spravuje datovou a výpočetní logiku aplikace

- komponenta **view**, která spravuje vykreslovací logiku aplikace a přímo vytváří uživatelské rozhraní

Jak si můžete všimnout, jednotlivé komponenty softwarové architektury MVC přímo odpovídají výše uvedeným základním faktům a poskytují tak možnou abstrakci a základní rozdělení pro tvorbu aplikace. Předem však podotýkám, že vzhledem k očekávání spuštění aplikace pomocí webového prohlížeče, co nejuniverzálnějšímu přístupu, snadné rozšiřitelnosti a rychlosti (viz uvedené nefunkční požadavky v kapitole Analýza) nebude možné teoretickou nezávislost jednotlivých komponent vždy dodržet. Nicméně pro základní návrh a abstrakci je softwarová architektura MVC vhodná, přihlédneme-li rovněž k faktu, že se tento typ architektury v současnosti používá právě při tvorbě webových aplikací. Nyní zbývá k jednotlivým komponentám přiřadit funkce, které vyplynuly z analýzy aplikace.

4.1.2 Rozdělení základních funkcí do komponent

Po podrobném rozboru výstupů analýzy, především funkčních požadavků a případů užití, je možné jednotlivým komponentám softwarové architektury MVC přiřadit funkce, které implementovaná aplikace bude poskytovat. S využitím podrobnější úrovně abstrakce jsem definoval základní funkce aplikace, kde jsem se zatím nezabýval jejich využitím pro jednotlivé případy (např. reakce na vstup z myši v grafu, reakce na vstup z myši v hlavním menu apod.):

- reakce na vstup z klávesnice
- reakce na vstup z myši
- načtení tutoriálu ze souboru
- uložení tutoriálu do souboru
- přepnutí na jinou stránku tutoriálu
- překreslení tutoriálu (popisné texty a graf)
- vytvoření dialogu pro úpravu (uzlu, hrany, textu apod.)
- zobrazení dialogu pro úpravu (uzlu, hrany, textu apod.)
- výpočet hodnot veličin v grafu
- překreslení grafu

Tyto základní funkce aplikace jsem následně přiřadil jednotlivým komponentám softwarové architektury MVC, které jsou zodpovědné za jejich správu. Přiřazení je přímočaré a nevyžadovalo hlubší zamyšlení.

Model

- načtení tutoriálu ze souboru

- uložení tutoriálu do souboru
- výpočet hodnot veličin v grafu

View

- překreslení tutoriálu (popisné texty a graf)
- zobrazení dialogu pro úpravu (uzlu, hrany, textu apod.)
- překreslení grafu

Controller

- reakce na vstup z klávesnice
- reakce na vstup z myši
- přepnutí na jinou stránku tutoriálu
- vytvoření dialogu pro úpravu (uzlu, hrany, textu apod.)

Toto základní rozdělení slouží k tomu, aby bylo možné provést rozdělení detailnějších funkcí aplikace tak, aby respektovaly výstupy analýzy, tj. především funkční požadavky a případy užití. Díky tomu pak lze přistoupit k vlastní implementaci.

4.1.3 Detailnější rozdělení funkcí pomocí diagramů tříd

Jazyk UML poskytuje takzvaný diagram tříd, který lze použít k detailnímu rozdělení funkcí do objektů, které jsou zodpovědné za jejich správný průběh. Tyto objekty lze pak přímo implementovat ve zdrojovém kódu. Nebudu se zde zabývat vysvětlením, co to jsou třídy, objekty a jejich atributy a metody, neboť to je mimo zaměření této práce. Tuto znalost však budu v následujících sekcích předpokládat, proto zájemce odkazuji na zdrojovou literaturu, která se zabývá jazykem UML a potřebným teoretickým úvodem.

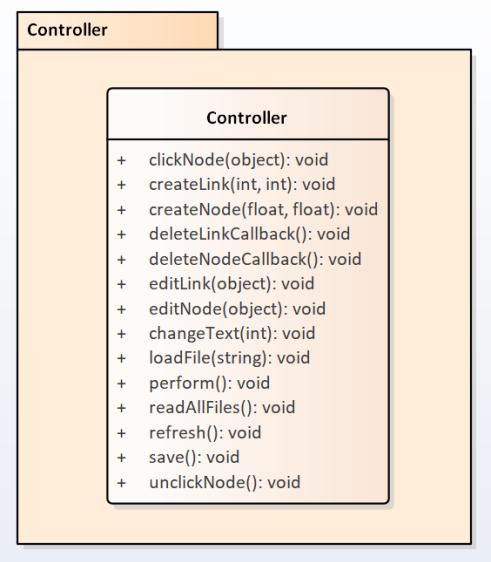
Pro vlastní tvorbu diagramu tříd je nutné definovat funkci daných tříd, název a vzájemnou vazbu s ostatními třídami v rámci i napříč jednotlivými komponentami. V této fázi za důležitou považuji rovněž základní definici nejdůležitějších metod, které přímo splňují jednotlivé funkční požadavky a které lze postupně v různém pořadí zavolat ke splnění všech případů užití. K tomu je potřeba nicméně znát datové typy jednotlivých parametrů, které metody dostávají. Kromě standardních typů je také nutné definovat datový typ, který bude reprezentovat definici tutoriálu, a případně ve kterém se budou data ukládat do souboru. Vzhledem k tomu, že má aplikace nativně běžet ve webovém prohlížeči, je možné zvolit z těch datových typů či formátů souborů, které se v případě webového prostředí nejčastěji využívají a jsou tak kompatibilní

napříč různými knihovnami. Z tohoto důvodu jsem se rozhodl pro formát a datový typ JSON („JavaScript Object Notation“), který lze použít jak pro uložení souboru tutoriálu, tak pro uložení a práci s daty tutoriálu přímo v aplikaci. Následující diagramy tedy s tímto typem počítají.

Nyní uvádím samostatně diagramy jednotlivých komponent, které obsahují definované třídy, načež závěrem uvedu diagram, ve kterém budou znázorněny vazby mezi třídami. Předem je důležité zmínit, že aplikace je z hlediska množství objektů značně jednoduchá a její funkčnost přímočará, proto si jednotlivé komponenty zpravidla vystačí s jedním objektem, který spravují a za jehož udržování jsou zodpovědné. Všechny diagramy byly vytvořeny v programu Enterprise Architect.

4.1.3.1 Komponenta controller

Třída *Controller*, starající se o vlastní běh aplikace a zpracovávající uživatelský vstup, obsahuje ze všech tříd v aplikaci nejvíce metod, které se týkají spuštění aplikace, načtení tutoriálu ze souboru, přidání uzlu a hrany do grafu atd. Pod diagramem jsou jednou větou popsány funkčnosti jejích nejdůležitějších metod.



Obrázek 4.1: Diagram komponenty controller

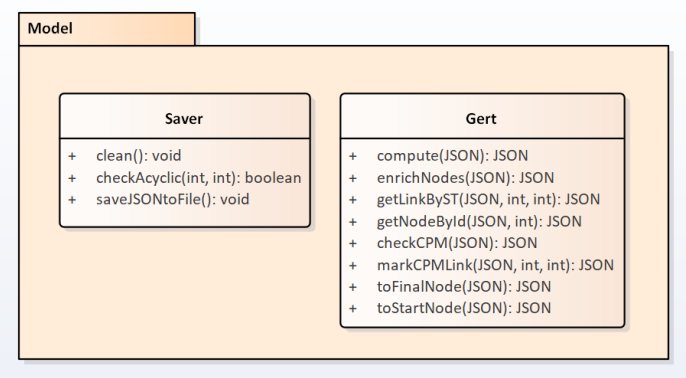
- *perform(): void* – načte úvodní obrazovku s jejím obsahem a funkcemi
- *readAllFiles(): void* – přečte všechny předem definované soubory s tutoriály a načte jejich název a umístění
- *loadFile(string): void* – načte konkrétní soubor s tutoriálem, jehož obsah nechá vykreslit
- *refresh(): void* – nechá překreslit celý graf
- *createNode(float, float): void* – vytvoří nový uzel na zadané pozici v grafu
- *createLink(int, int): void* – vytvoří novou hranu mezi vybraným počátečním a koncovým uzlem pomocí dialogu pro vytvoření nové hrany
- *editNode(object): void* – upraví zadaný uzel pomocí dialogu pro úpravu uzlu

- *editLink(object): void* – upraví zadanou hranu pomocí dialogu pro úpravu hrany
- *deleteNodeCallback(): void* – odstraní zadaný uzel, který si aplikace drží v paměti, včetně všech jeho hran
- *deleteLinkCallback(): void* – odstraní zadanou hranu, kterou si aplikace drží v paměti
- *changeText(int): void* – upraví zadaný text pomocí dialogu pro úpravu textu
- *clickNode(object): void* – uloží do paměti uzel, na který uživatel klikl
- *unclickNode(): void* – odstraní z paměti ukazatel na uzel, na který předtím uživatel klikl
- *save(): void* – nechá uložit tutoriál do souboru

Samozřejmě jsou popisy metod značně jednoduché a záleží na konkrétní implementaci, jak se k realizaci jejich funkcí postaví. S tím souvisí i to, že si každá implementace určí vlastní atributy objektů, které potřebuje k uchovávání dat v paměti a k reakci na uživatelské vstupy. Stejně tak si jednotlivé realizace mohou přidat vlastní pomocné metody, nicméně musí zároveň respektovat tento architektonický návrh. To se týká i tříd v ostatních komponentách.

4.1.3.2 Komponenta model

Komponenta model obsahuje dvě třídy, a to třídu *Saver* starající se o správu a uložení dat tutoriálu do souboru, a třídu *Gert*, která počítá hodnoty veličin v grafu podle metod GERT a CPM. Pod diagramem následuje krátký popis jejich metod.



Obrázek 4.2: Diagram komponenty model

Třída *Saver*

- *clean(): void* – odstraní z paměti ta data z grafu, která jsou závislá na výpočtu a nesouvisí se zadáním
- *saveJSONtoFile(): void* – uloží data tutoriálu do souboru, který metoda nechá uživatele stáhnout k sobě do počítače
- *checkAcyclic(int, int): void* – provádí poslední kontrolu, jestli je uložený graf acyklický (aplikace nebude umět pracovat s, ani ukládat cyklické grafy)

Třída *Gert*

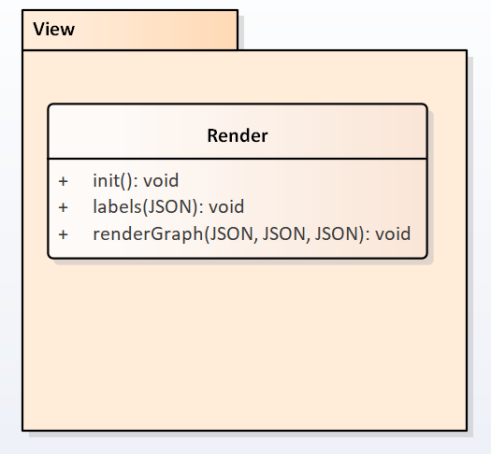
- *compute(JSON): JSON* – provede výpočet hodnot veličin grafu z jeho zadání, a to buď metodou GERT nebo CPM
- *toFinalNode(JSON): JSON* – provede výpočet pomocí metody GERT/CPM směrem ke koncovému uzlu grafu
- *toStartNode(JSON): JSON* – provede výpočet pomocí metody GERT/CPM směrem k počátečnímu uzlu grafu
- *markCPMLink(JSON, int, int): JSON* – pomocná metoda, která označí hrany na kritické cestě

- *checkCPM(JSON): JSON* – pomocná metoda, která zkontroluje označení kritické cesty
- *enrichNodes(JSON): JSON* – pomocná metoda, která k uzlům přiloží seznam počátečních a koncových uzlů hran, které z/do nich vedou
- *getNodeById(JSON, int): JSON* – pomocná metoda, která vrátí uzel zadaný jeho identifikačním číslem
- *getLinkByST(JSON, int, int): JSON* – pomocná metoda, která vrátí hranu grafu definovanou počátečním a koncovým uzlem

Obě uvedené třídy se starají vysloveně o práci s uloženými daty, načež nad nimi pracují a berou je jako svůj vstup i výstup. Tyto třídy tedy přenechávají uživatelské prostředí a obecně interakci s uživatelem jiným komponentám.

4.1.3.3 Komponenta view

Komponenta view obsahuje pouze jedinou třídu *Render*, která však zároveň předpokládá přístup ke zdrojům (obrázky, styly, fonty apod.) a případně přístup k vykreslovacím knihovnám. Popis jejích základních metod je nicméně přímočarý.



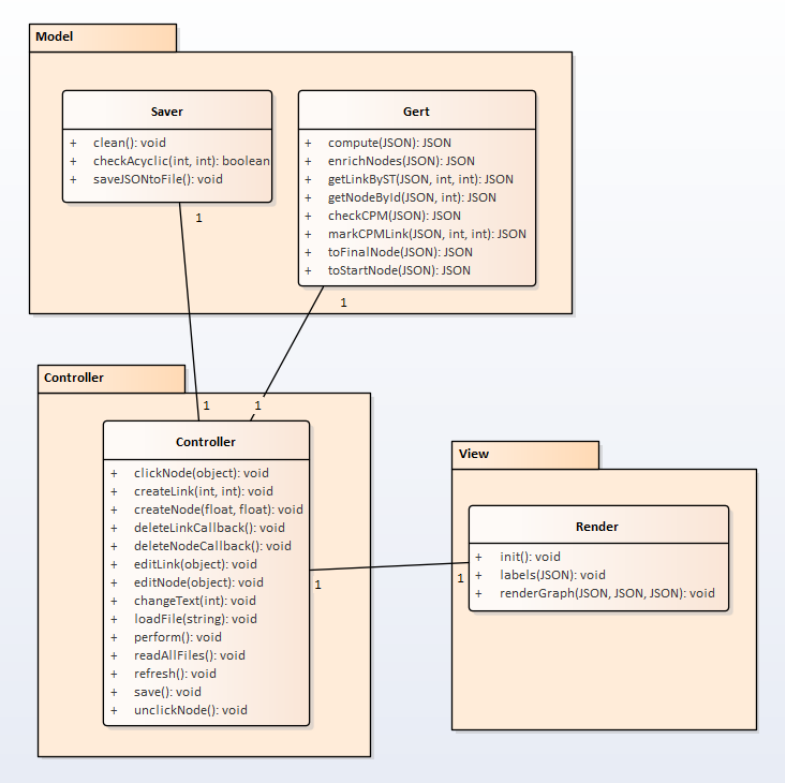
Obrázek 4.3: Diagram komponenty view

- *init(): void* – inicializuje vykreslovací jádro, popřípadě knihovnu, a oblast, do které se bude zobrazovat graf
- *renderGraph(JSON, JSON, JSON): void* – vykreslí do určené oblasti graf zadaný předem definovaným nastavením, uzly a hranami
- *labels(JSON): void* – pomocná metoda, která zobrazí všechny popisné texty

Vlastní vykreslení je pak v realizaci dané implementace aplikace a knihoven, které bude využívat.

4.1.3.4 Vzájemná interakce mezi objekty tříd

Následující diagram poskytuje širší pohled na spolupráci mezi třídami a tedy i mezi jednotlivými komponentami softwarové architektury MVC. Z diagramu je patrné, že každá třída vytváří právě jeden objekt, který se stará o výše uvedené funkcionality aplikace. Takováto jednoduchost je spojena s nefunkčním požadavkem na srozumitelnost a snadnou rozšiřitelnost aplikace, která se sama o sobě stará především o zobrazení dat a grafu z předem napsaných tutoriálů. Vzhledem k tomu, jak bylo uvedeno výše, že jsou data uložena ve formátu JSON, který sám poskytuje flexibilní strukturu a v zásadě libovolnou velikost, je takováto jednoduchost dostačující.



Obrázek 4.4: Diagram tříd aplikace

4.2 Zvolené technologie

Zvolené technologie pro implementaci je nutné přizpůsobit funkčním a nefunkčním požadavkům. Jelikož se jedná o aplikaci, která má nativně běžet ve webovém prohlížeči a zároveň která má být snadno rozšiřitelná, je vhodné zvolit takovou webovou technologii, kterou jednotlivé prohlížeče přímo podporují. Výše jsem již uvedl, že jsem se přesně z tohoto důvodu rozhodl jako formát souboru k ukládání a načítání tutoriálů zvolit formát JSON, který zároveň poskytuje základní strukturu dat a rovněž může sloužit jako vlastní datový typ.

K napsání aplikace pro webový prohlížeč jsem samozřejmě upřednostnil, a tedy i zvolil, nativní technologii vykreslování založenou na HTML („HyperText Markup Language“) a CSS („Cascading Style Sheets“). Pro tvorbu kódu vlastní aplikace je možné si zvolit z celé palety programovacích jazyků v závislosti na tom, jestli bude aplikace běžet na serveru, nebo přímo u uživatele na počítači. Nefunkční požadavek „aplikace bude nezávislá na externím serveru a internetovém připojení“ (viz kapitola Analýza) však serverovou možnost vyvrací. Proto jsem se rozhodl jako programovací jazyk zvolit takový, jehož kód bude běžet u uživatele na počítači a zároveň který bude nativně podporovaný webovými prohlížeči. Takovým jazykem je JavaScript [16], který má ještě několik dalších výhod [17]:

- JavaScript je rychlý, a to především z toho důvodu, že běží přímo u uživatele na počítači
- JavaScript je jednoduchý na pochopení
- JavaScript využívá DOM model a jeho funkcionality

Abych ještě funkčnost JavaScriptu rozšířil a nemusel tedy znovu psát v různých projektech se opakující kód, rozhodl jsem se zvolit a použít softwarovou strukturu jQuery [18], která je vzhledem ke své malé velikosti bohatá na různé funkcionality, například i pro ovlivňování CSS stylů.

Pro podporu a vykreslování jak grafu, tak i celé webové aplikace, jsem hledal vhodné knihovny, které by byly jak jednoduché a snadné na pochopení, tak i vhodné ke splnění požadavků na aplikaci (přidávání nových uzlů do grafu, pohyb s uzly na obrazovce apod.). Jelikož jazyk HTML nativně podporuje vykreslování SVG („Scalable Vector Graphics“) grafiky, rozhodl jsem se využít knihovnu, která je napsaná v JavaScriptu a která zároveň umí elementem SVG manipulovat a přidávat do něj vlastní prvky. Nakonec jsem se rozhodl zvolit open-source knihovnu s názvem d3js [19], jejíž kód si případní další tvůrci rozšíření pro aplikaci mohou prohlédnout a tedy snadno pochopit.

Zbývalo zvolit knihovnu pro podporu zobrazení celé aplikace, především například hlavního menu, stylu textů a podobně. Po hledání a přemýšlení jsem nakonec zvolil známou knihovnu Bootstrap [20], která umožňuje snadno

tvořit grafiku a strukturu elementů aplikace pomocí CSS tříd. K zobrazení ikonek přepínání na další/předchozí stránku tutoriálu jsem zvolil knihovnu Font Awesome [21], která zároveň poskytuje vlastní fonty.

Jako podpůrnou knihovnu pro uložení souborů jsem zvolil malinkou kompaktní knihovnu s názvem FileSaver [22], která umožňuje případně uložit i velké soubory, kterými mohou soubory ve formátu JSON s údaji o grafu a celém tutoriálu včetně popisných textů být.

4.3 Příklady z implementace

Návrh architektury jsem musel přizpůsobit vlastní implementaci v jazyce JavaScript, a to především z toho důvodu, že jazyk JavaScript nativně neobsahuje možnost instancování tříd. Vzhledem k tomu, že každá třída v návrhu architektury je instancována právě jednou, rozhodl jsem se každou třídu z architektury promítnout přímo do jednoho objektu v implementaci, který již sám obsahuje definice potřebných metod a atributů. Každý takový objekt je pak uložený do proměnné, jejíž název odpovídá navrženému názvu třídy. Kromě tohoto omezení je nicméně samotná implementace značně přímočará a odpovídá specifikům skriptovacího jazyka. Pro názornější představu nyní uvedu příklad implementace některých zajímavých metod spolu s popisem jejich fungování a návrhu. Veškeré komentáře, jakož i názvy objektů, metod a proměnných, jsou psány v angličtině. Ke každé metodě je v kódu uveden popis její struktury, aby byla snadno pochopitelná pro případná další rozšíření (viz požadavky v kapitole Analýza). Ze stejného důvodu jsem se snažil jednotlivé metody psát co nejkratší, aby bylo co nejvíce jasné, jaký kód vykonávají a jaký výsledek vracejí.

4.3.1 Model

4.3.1.1 Příklady z objektu Gert

Metoda *toFinalNode* objektu *Gert* provádí první část výpočtu pomocí metod GERT a CPM, který prochází grafem od jeho počátečního uzlu ke koncovému. Tato metoda využívá pomocnou metodu s názvem *checkIfComputedByNodes* k zastavení cyklu v případě, že již do výpočtu byly zahrnuty všechny uzly grafu, a tedy není dále co v tomto směru výpočtu zjišťovat.

```
/**
 * One-way of the GERT algorithm to the final node.
 * It uses different computations based on
 * the type of nodes.
 * @param {JSON} enrichedJSON
 * @returns {JSON} computedJSON
**/
Gert.toFinalNode = function (enrichedJSON) {
```

```
    var computedJSON = enrichedJSON;
    while (!Gert.checkIfComputedByNodes(computedJSON.nodes,
                        true)) {
        for (var i in computedJSON.nodes) {
            var node = computedJSON.nodes[i];
            if (node.targetTo === undefined) {
                // The first node time is zero
                computedJSON.nodes[i].time0 = 0;
                computedJSON.nodes[i].probability = 1;
                computedJSON.nodes[i].computed = true;
            } else if (Gert.checkIfComputedByIds(computedJSON,
                                node.targetTo, true)) {
                if (node.type !== undefined
                        && node.type == "inclusive")
                    computedJSON = Gert.inclusiveNodeToFinal(
                                        computedJSON, i);
                else if (node.type !== undefined
                        && node.type == "disjunctive")
                    computedJSON = Gert.disjunctiveNodeToFinal(
                                        computedJSON, i);
                else
                    computedJSON = Gert.conjunctiveNodeToFinal(
                                        computedJSON, i);
            }
        }
    }
    return computedJSON;
};
```

Zdrojový kód 4.1: Příklad metody toFinalNode objektu Gert

V případě, že se jedná o počáteční uzel grafu, metoda *toFinalNode* nastaví nejdříve možný začátek následujících činností na nulu a pravděpodobnost jeho realizace na hodnotu *1*. Pokud se naopak jedná o uzel, jehož předcházející uzly již prošly výpočtem (o to se stará pomocná metoda *checkIfComputedByIds*), metoda podle typu uzlu rozhodne způsob výpočtu, který přenechá jedné z následujících pomocných metod. Ty pak provedou výpočet podle zadaných vzorců (viz kapitola Teoretická část).

- *conjunctiveNodeToFinal*, která se stará o výpočet hodnot veličin u konjunktivního uzlu ve směru od počátečního uzlu ke koncovému
- *inclusiveNodeToFinal*, která se stará o výpočet hodnot veličin u inkluzivního uzlu ve směru od počátečního uzlu ke koncovému

- *disjunctiveNodeToFinal*, která se stará o výpočet hodnot veličin u disjunktivního uzlu ve směru od počátečního uzlu ke koncovému

Metoda nakonec vrátí JSON s vypočítanými hodnotami, které je možné dále zpracovávat. Nyní následuje ukázka metody *conjunctiveNodeToFinal*, na které metoda *toFinalNode* závisí. Metoda *conjunctiveNodeToFinal* se věnuje výpočtu nejdříve možného začátku, nejpozději přípustného začátku a pravděpodobnosti realizace u konjunktivního uzlu ve směru od počátečního uzlu grafu ke koncovému.

```
/**
 * Computes the first way of the GERT algorithm
 * for conjunctive nodes.
 * @param {JSON} computedJSON
 * @param {int} currentNodeIndex
 * @returns {JSON} computedJSON
**/
Gert.conjunctiveNodeToFinal = function (computedJSON,
        currentNodeIndex) {
    // Preparing the variables
    var maximumTime = 0;
    var probability = 1;
    // Preparing current values
    var currentNode = computedJSON.nodes[currentNodeIndex];
    // Getting the maximum time value and the probability
    for (var j in currentNode.targetTo) {
        var sourceNode = Gert.getNodeById(computedJSON,
                        currentNode.targetTo[j]);
        var link = Gert.getLinkByST(computedJSON,
                        currentNode.targetTo[j], currentNode.id);
        var currentProbability = sourceNode.probability
                * link.probability;
        var time = sourceNode.time0 + link.duration
                * currentProbability;
        if (time > maximumTime) {
            maximumTime = time;
        }
        probability *= sourceNode.probability * link.probability;
    }
    // Saving and returning
    computedJSON.nodes[currentNodeIndex].time0 = maximumTime;
    computedJSON.nodes[currentNodeIndex].probability = probability;
    computedJSON.nodes[currentNodeIndex].computed = true;
    return computedJSON;
```

```
};
```

Zdrojový kód 4.2: Příklad metody conjunctiveNodeToFinal objektu Gert

Metoda *conjunctiveNodeToFinal* postupně prochází všemi uzly, ze kterých vedou hrany k určenému uzlu, zadanému parametrem *currentNodeIndex*. Co se týče hodnoty nejdříve možného začátku následujících činností, metoda vybere a zapíše maximální hodnotu doby takové, která se rovná součtu doby nejdříve možného začátku předcházející činnosti a doby trvání předcházející činnosti, a to v souladu se specifikací metod GERT a CPM.

Většina následujících metod objektu *Gert* obsahuje podobně založené výpočetní metody závislé na zvoleném vzorci, nicméně z hlediska implementace se jedná o podobný způsob, který představují i výše uvedené ukázky zdrojového kódu.

4.3.1.2 Příklady z objektu Saver

Metody objektu *Saver* jsou jednoduché a krátké, jelikož se v zásadě starají pouze o odstranění spočítaných hodnot z grafu pro uložení a o uložení dat s využitím malé knihovny FileSaver. Zajímavá je nicméně metoda *checkAcyclic* pro zjištění acykličnosti grafu, jejíž ukázku zde uvádím.

```
/**
 * Checks if a graph is acyclic using a new edge.
 * @param {integer} source
 * @param {integer} target
 * @returns {boolean} acyclic
*/
Saver.checkAcyclic = function (source, target) {
    var queue = new Array();
    queue.push(target);
    while (queue.length >= 1) {
        var element = queue.shift();
        if (element == source)
            return true;
        for (var i in Saver.JSON.links) {
            if (Saver.JSON.links[i].source.id == element) {
                queue.push(Saver.JSON.links[i].target.id)
            }
        }
    }
    return false;
};
```

Zdrojový kód 4.3: Příklad metody checkAcyclic objektu Saver

Tato krátká metoda se stará o to, aby zjistila, že ukládaný graf je skutečně acyklický, jelikož aplikace z definice metod GERT a CPM nepodporuje cyklické grafy, u nichž by výpočet pomocí těchto metod nebyl možný. Metoda *checkAcyclic* využívá frontu k uložení uzlů, které následují (tj. vedou do nich hrany) po již vybraných uzlech z fronty. Pokud se dostane zpět k zadanému zdrojovému uzlu, graf je zcela určitě cyklický. Tato kontrola musí logicky proběhnout u každé hrany, kde metoda *checkAcyclic* očekává její počáteční a koncový uzel jako vstupní parametry.

4.3.2 View

4.3.2.1 Příklady z objektu Render

Objekt *Render* ve svých metodách přímo využívá knihovnu d3js pro vykreslování a práci s grafem. Stejně tak pro manipulaci s HTML elementy (strukturou) a jejich CSS styly (vzhledem) využívá knihovnu jQuery, která JavaScriptový kód značně zjednodušuje, což ostatně můžete sami posoudit. Objekt *Render* rovněž pracuje se zdroji aplikace pro vykreslování, ať už se jedná o soubor s CSS styly či o obrázky pozadí jednotlivých uzlů. Těchto obrázků je přesně šest v závislosti na tom, zda se jedná o:

- konjunktivní (podle vstupu) deterministický (podle výstupu) uzel
- konjunktivní (podle vstupu) stochastický (podle výstupu) uzel
- inkluzivní (podle vstupu) deterministický (podle výstupu) uzel
- inkluzivní (podle vstupu) stochastický (podle výstupu) uzel
- disjunktivní (podle vstupu) deterministický (podle výstupu) uzel
- disjunktivní (podle vstupu) stochastický (podle výstupu) uzel

Tyto obrázky jsou uložené v adresáři *img* u souborů určených komponentě view. Nyní následuje ukázka **části** metody *renderGraph*, která se stará o vykreslení grafu se všemi jeho součástmi. V následující ukázce tato metoda napřed odstraní z elementu SVG, kde provádí samotné vykreslování, předchozí objekty, aby nevznikla situace, kdy by se vykreslilo několik grafů na sobě. Následně zobrazí veškeré popisné texty, šipky pro přechod na předchozí/další stránku tutoriálu a vytvoří hrany mezi jednotlivými uzly včetně symbolů šipek.

```
/**
 * Draws the graph from specified JSON objects.
 * @param {JSONObject} settings
 * @param {JSONArray} links
 * @param {JSONArray} nodes
```

```
**/
Render.renderGraph = function (settings, links, nodes) {
    // Clean
    Render.cleanSVG(Render.svg);

    // Show labels
    Render.labels(settings);

    // Setting next and previous listeners
    // Next
    if (settings.next !== undefined
        && settings.next !== "") {
        Render.arrowLinkStyle("arrow-next", true);
        $("#arrow-next").unbind("click");
        $("#arrow-next").click(function () {
                if (Saver.saved
                || confirm(text)) {
                        Saver.saved = true;
                        Controller.loadFile(settings.next);
                } });
    } else {
        Render.arrowLinkStyle("arrow-next", false);
        $("#arrow-next").unbind("click");
    }
    // Previous
    if (settings.previous !== undefined
        && settings.previous !== "") {
        Render.arrowLinkStyle("arrow-previous", true);
        $("#arrow-previous").unbind("click");
        $("#arrow-previous").click(function () {
                if (Saver.saved
                || confirm(text)) {
                        Saver.saved = true;
                        Controller.loadFile(settings.previous);
                } });
    } else {
        Render.arrowLinkStyle("arrow-previous", false);
        $("#arrow-previous").unbind("click");
    }

    Render.link = Render.svg.selectAll(".link")
        .data(links)
        .enter()
        .append("line")
```

```
        .on("dblclick", function (d) {
            Controller.editLink(d); // registering an event
            d3.event.stopPropagation();
        })
        .attr("class", function (d) { return "link"
                + (d.critical_confirmed !== undefined
                ? d.critical_confirmed.toString() : "0"); })
        .attr('marker-end', 'url(#arrowhead)')

    Render.link.append("title")
        .text(function (d) { return d.duration.toString(); });

//
// nasledujici kod vynechan
//
```

Zdrojový kód 4.4: Příklad části metody renderGraph objektu Render

Po vykreslení základních popisných textů se metoda *renderGraph* rozhodne, zda zobrazí šipky pro přechod na předchozí či následující stránku tutoriálu. Toto rozhodnutí provádí pomocí toho, zda tutoriál předchozí/další stránku opravdu obsahuje. Pokud ne, metoda šipku zabarví tak, aby byla hůře viditelná, a znemožní kliknutí. Přidání hran do grafu je díky knihovně d3js jednoduché, kdy stačí vykreslovacímu objektu předat předformátovaný JSON s údaji o hraně a nastavit příslušné atributy, především symboly šipek a třídu pro práci s CSS styly. Nakonec metoda v této ukázce ke hraně připojí popisek s dobou trvání dané činnosti, kterou daná hrana reprezentuje.

Ostatní (pomocné) metody objektu *Render* jsou značně jednodušší a v zásadě se starají o vykreslování pouze jim svěřené části grafu či o zpracování nejnutnějších událostí. Objekt *Render* rovněž obsahuje mnoho konstant pro nastavení správné velikosti a pozic objektů v grafu.

4.3.3 Controller

4.3.3.1 Příklady z objektu Controller

Objekt *Controller* obsahuje důležité metody pro spuštění vlastní aplikace, především pro načítání souborů s tutoriály. Rovněž se stará o otevření dialogů pro přidání/úpravu hran či uzlů v grafu a změnu popisných textů. Tento objekt přímo využívá uživatelských vstupů k provedení vlastních akcí a k upozornění uživatele na požadované události (například upozornění po změně v grafu, že nedošlo k uložení, či pro upozornění, že zadané nové identifikační číslo uzlu již v grafu existuje).

Metoda *readAllFiles* nejen že načítá jednotlivé soubory s tutoriály, ale zároveň nechává zobrazit úvodní obrazovku, kde si uživatel může z jednotlivých

tutoriálů vybrat, a odesílá údaje o tutoriálech do hlavního menu. Pro kliknutí na název daného tutoriálu (v případě hlavního menu) nebo pro kliknutí na spouštěcí tlačítko (v případě úvodní obrazovky), metoda *readAllFiles* registruje metodu *loadFile*, která daný tutoriál načte a zobrazí. Následuje její ukázka. Všimněte si, že metoda *loadFile* využívá metod všech výše zmíněných objektů, tj. objektů *Gert*, *Saver* a *Render*.

```
/**
 * Loads a graph from a specified file.
 * @param {String} fileName
**/
Controller.loadFile = function (fileName) {
    // Load a file
    d3.json("content/" + fileName, function (error, graph) {
        if (error)
            throw error;
        // Checking CPM limits
        if (graph.settings.type == "CPM") {
            // Nodes
            for (var i in graph.nodes) {
                graph.nodes[i].type = undefined;
            }
            // Links
            for (var i in graph.links) {
                graph.links[i].probability = 1;
            }
        }
        // Save the graph info to model
        Saver.JSON = graph;
        // Compute the graph for GERT and CPM methods
        var gertJSON = Gert.compute(Saver.JSON);
        // Draw the graph
        Render.renderGraph(gertJSON.settings,
                gertJSON.links, gertJSON.nodes)
    });
};
```

Zdrojový kód 4.5: Příklad metody loadFile objektu Controller

Metoda *loadFile* stejně jako metody objektu *Render* využívá výhod knihovny d3js, která sama obsahuje mechanismus pro správné načítání JSON souborů, kde jsou uložené specifikace jednotlivých tutoriálů. Pokud je u daného tutoriálu zvolena čistě metoda CPM, aplikace nedovolí použití pravděpodobností a jiného než konjunktivního uzlu (viz kapitola Teoretická část). Po načtení souboru s tutoriálem ho metoda *loadFile* předá objektu *Saver*, který data o něm spravuje. Následně pak proběhne výpočet pomocí metod GERT a CPM, načež

dojde k vykreslení grafu a popisných textů na obrazovku do specifikovaného HTML elementu, o což se již stará objekt *Render*.

Jak již bylo uvedeno výše, objekt *Controller* se rovněž stará o práci s dialogy pro přidání/úpravu hrany, uzlu a popisných textů. V závislosti na předem daném uživatelském vstupu metoda nechá příslušný dialog zobrazit, načež registruje odezvu na danou událost, přesněji na kliknutí na tlačítko pro uložení či odstranění. Poté provede danou akci, informuje objekt *Saver*, který se stará o práci s daty, a nechá překreslit graf tutoriálu. Jak metody pro zobrazení dialogu, tak pro zpracování události využívají knihovnu jQuery.

Následuje ukázka metody *createLink*, která nechá zobrazit dialog pro vytvoření hrany na pozici mezi předem zadanými uzly v závislosti na uživatelském vstupu (viz Případy užití v kapitole Analýza). Poté registruje reakci na událost kliknutí na ukládací tlačítko, kterou zpracovává pomocná metoda *editLinkCallback*.

```
/**
 * Shows a dialog to create a link.
 * @param {integer} source
 * @param {integer} target
**/
Controller.createLink = function (source, target) {
    Saver.tempData = null;
    // Values to be changed
    $('#edit-modal').find('.modal-title').text(text);
    $('#edit-save-button').unbind('click');
    $('#edit-save-button').click(function () {
                Controller.editLinkCallback();
        });
    $('#delete-link').unbind('click');
    // Default values in inputs
    $("#input2a").val(source.toString());
    $("#input2b").val(target.toString());
    $("#input2c").val(1);
    $("#input2d").val(1);
    // Showing used parts
    $("#input1-part").hide();
    $("#input2a-part").show();
    $("#input2b-part").show();
    $("#input2c-part").show();
    if (Saver.JSON.settings.type == "CPM") {
        $("#input2d-part").hide();
    } else {
        $("#input2d-part").show();
    }
```

```
    $("#select-part").hide();
    $("#delete-part").hide();
    // Showing the modal
    Controller.allowInput = false;
    $('#edit-modal').modal('show');
};
```

Zdrojový kód 4.6: Příklad metody createLink objektu Controller

Metoda *createLink* napřed předá dialogu jeho nadpis a od tlačítka pro uložení odebere předchozí reakce na události, načež přidá vlastní. Následně jednotlivým formulářovým políčkům předá výchozí hodnoty, které uživatel bude moci změnit, a zobrazí pouze ty, které jsou pro vytvoření nové hrany relevantní, kde ostatní nepotřebná políčka schová. Potřebnost políček je určena i typem zvolené metody, ať už se jedná o metodu GERT či CPM. Metoda *createLink* na svém konci zavolá příkaz ke zobrazení dialogu.

Ostatní metody objektu *Controller* pak využívají podobných příkazů či jsou pouze podpůrné (například rychlé překreslení grafu, „odkliknutí“ předem zvoleného uzlu apod.). Tyto metody jsou krátké a tedy snadné na okamžité pochopení.

4.4 Instalační a uživatelská příručka

Součástí aplikace je instalační příručka, která popisuje způsob spuštění aplikace pomocí webových prohlížečů Mozilla Firefox, Microsoft Edge a Google Chrome. Tato instalační příručka s názvem *instalace.md* se nachází na přiloženém DVD, které je součástí této práce, a to ve složce *impl*.

Uživatelská příručka pak popisuje ovládání aplikace, přidávání uzlů a hran do grafu, změnu parametrů apod. Tato příručka je přímo součástí implementované aplikace v podobě tutoriálu nazvaného Práce s aplikací (viz Tvorba tutoriálů níže), ke kterému se přistupuje po otevření aplikace kliknutím na tlačítko „Spustit“ pod názvem a popisem příslušného tutoriálu.

4.5 Shrnutí

Zvolená softwarová architektura MVC, která zdrojový kód aplikace dělí do tří komponent, kde jedna slouží pro výpočty nad daty a pro správu dat, druhá pro vykreslování objektů na obrazovku a třetí pro komunikaci s uživatelem a pro správu řídících procesů, slouží zároveň i jako abstrakce nad základními procesy, které aplikace podporuje. Díky tomuto rozlišení bylo možné jednotlivé základní funkčnosti aplikace rozdělit mezi tyto komponenty, kdy například překreslení tutoriálu patří ke komponentě view, reakce na vstup z klávesnice ke komponentně controller, výpočet hodnot veličin v grafu ke komponentě model atp. Pro bližší specifikaci pak slouží jednotlivé třídy se svými metodami

a atributy, definované diagramy tříd, kde jsou rovněž uvedeny jejich vzájemné vazby. Každá třída se svou funkčností náleží právě k jedné z výše uvedených komponent a obsahuje metody, které přímo pomáhají splnit jednotlivé funkční požadavky na aplikaci a jejichž voláním v určeném pořadí lze nasimulovat zjištěné případy užití.

Po vytvoření architektury a definici základních metod jsem zvolil technologie, na nichž bude aplikace založená. Jelikož hlavním požadavkem je, aby aplikace běžela ve webovém prohlížeči, zvolil jsem skriptovací jazyk JavaScript a podpůrné knihovny d3js (pro vykreslování grafu), jQuery (pro práci s HTML elementy a CSS styly) a FileSaver (pro ukládání souborů). Pro vlastní zobrazení posloužily technologie HTML (struktura) a styly CSS (vzhled) s podpůrnými knihovnami Bootstrap a Font Awesome, které obsahují definice základní podoby a vzhledu elementů zobrazovaných na obrazovce. Při vlastní implementaci nastala potřeba vytvořit pomocné metody a atributy, aby nedošlo k opakování stejného kódu na různých místech a k nepřehlednosti, které je potřeba se vzhledem k možnosti dalšího rozšiřování aplikace vyhnout. Každá třída v implementaci instancuje právě jeden objekt, proto ani absence třídní logiky v jazyce JavaScript není na závadu, jelikož lze přímo vytvořit objekty s danými metodami a atributy. Těmito objekty jsou objekt *Saver* pro ukládání souborů a *Gert* pro výpočet dle metod GERT a CPM (komponenta model), *Render* pro vykreslování tutoriálu včetně grafu a popisných textů (komponenta view) a *Controller* pro reakce na uživatelský vstup a volání ostatních objektů (komponenta controller). Komponenta view kromě objektu *Render* s definovanými metodami a konstantami obsahuje i potřebné zdroje aplikace pro vykreslování, konkrétně se jedná o obrázky, které se zobrazují na pozadí jednotlivých uzlů v grafu dle jejich typů, a o soubor s CSS styly.

Jako formát souboru pro uložení tutoriálu a jeho následné načítání byl zvolen typ JSON, který zároveň slouží pro dočasné uložení a práci s daty ve vlastní webové aplikaci. Zdrojový kód aplikace je přehledně rozdělen do adresářů dle jednotlivých komponent uvedených výše.

Kapitola 5

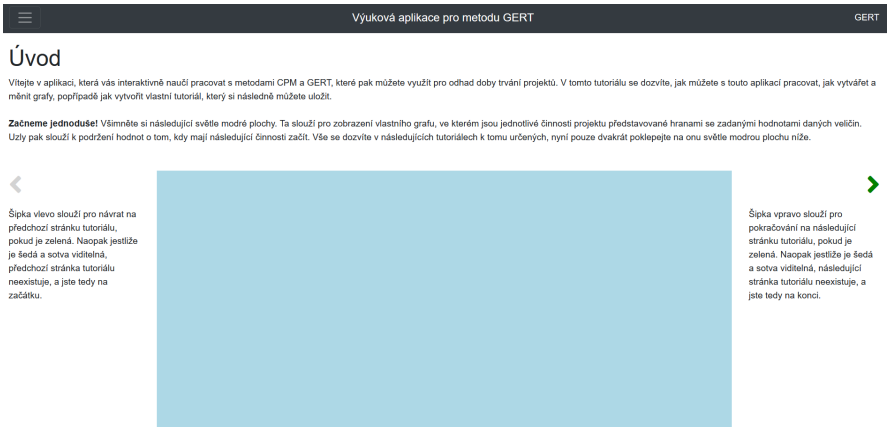
Tvorba tutoriálů

Po implementaci aplikace je možné přistoupit k tvorbě základních tutoriálů, které uživatele uvedou do vlastní aplikace a které mu přiblíží metodu CPM a metodu GERT přesně podle specifikovaných funkčních požadavků. Vlastní implementace aplikace poskytuje jistý rámec, v rámci kterého je možné tutoriály tvořit a využívat například šipek pro navigaci mezi stránkami tutoriálu, předdefinované umístění popisných textů a systém načítání tutoriálů z adresáře *content* ve zdrojích aplikace. Nicméně vzhledem k tomu, že aplikace spolu s popisnými texty kromě vlastního textového obsahu načítá i případné HTML značky, je možné chování tutoriálu a vlastně celé aplikace upravit vložením vlastního JavaScriptového kódu s knihovnou jQuery, čehož jsem na mnoha místech využil, aby každý z tutoriálů byl co nejvíce interaktivní (toto je v souladu s funkčními požadavky v kapitole Analýza). Tutoriály je možné dále modifikovat buď přímo v adresáři *content* v rámci příslušných JSON souborů s definicemi, nebo pomocí vlastní aplikace a jejího zabudovaného systému ukládání. Jednotlivé tutoriály jsou tedy stejně jako celá aplikace modifikovatelné a rozšiřitelné.

5.1 Práce s aplikací

První tutoriál se týká samotné práce s aplikací, kde se uživatel naučí pracovat s tutoriály, navigací, grafy (včetně přidávání a úpravy uzlů a hran), popisnými texty a ukládáním. Tento tutoriál stručně a především interaktivně vysvětluje všechny tyto základní funkčnosti a to, co představují (například hrany v grafu představují jednotlivé činnosti projektu). Rozdělen je do následujících čtyř stránek:

- **Úvod** – tato stránka uživatele naučí základní navigaci v rámci tutoriálu a naučí ho využívat oblast grafu
- **Uzly** – zde se uživatel dozví smysl uzlů v grafu, jaké hodnoty obsahují, načež si sám upraví identifikační číslo uzlu



Obrázek 5.1: Úvodní stránka tutoriálu pro práci s aplikací s vyznačeným pozadím grafu

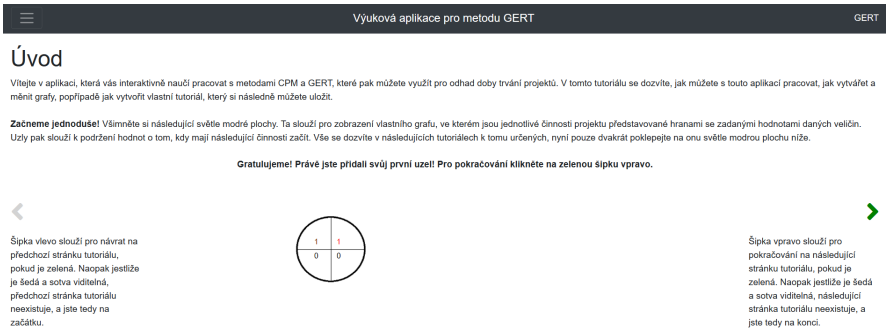
- **Hrany** – tato stránka uživateli vysvětlí problematiku hran (a činností projektu) grafu a provede ho dialogem pro vytvoření nové hrany
- **Tutoriály a ukládání** – na této stránce se uživatel naučí pracovat s hlavním menu, s ukládáním vlastních tutoriálů či příkladů a se změnou popisných textů

Minimálním požadavkem bylo vytvoření tří stránek, ovšem rozdělení do čtyř se ukázalo být mnohem lepší z toho důvodu, aby uživatel nebyl na jedné stránce zahlcen mnoha informacemi. První tři stránky tohoto tutoriálu nezobrazují při přepnutí na další stránku grafu hlášku, že došlo ke změnám a jestli je nechce uživatel uložit. Je to z toho důvodu, že dosud nebyl uživateli vysvětlen smysl ukládání a tato hláška by ho v tomto základním tutoriálu mohla mást.

5.1.1 Stránka Úvod

Stránka úvod se liší od všech ostatních stránek v základních tutoriálech aplikace tím, že obsahuje kód pro změnu barvy pozadí grafu v souvislosti s uživatelským vstupem. Díky tomu je možné zvýšenou mírou interaktivity uživatele navést k poklepání na zvýrazněnou plochu a tedy zároveň k přidání nového uzlu. Následná pochvalná hláška pouze doplní celý efekt a podnítí uživatele k přepnutí na další stránku pomocí navigační šipky.

Stránka obsahuje, když nepočítáme krátké texty k vysvětlení smyslu navigačních šipek, následující úvodní text. Všimněte si, že jsou texty v tutoriálu psány méně formálně tak, aby byla co nejvíce podnícena zvědavost uživatele a aby měl z aplikace příjemný pocit (koneckonců se jedná o výukovou aplikaci).



Obrázek 5.2: Úvodní stránka tutoriálu pro práci s aplikací po přidání uzlu

Vítejte v aplikaci, která vás interaktivně naučí pracovat s metodami CPM a GERT, které pak můžete využít pro odhad doby trvání projektů. V tomto tutoriálu se dozvíte, jak můžete s touto aplikací pracovat, jak vytvářet a měnit grafy, popřípadě jak vytvořit vlastní tutoriál, který si následně můžete uložit.

***Začneme jednoduše!** Všimněte si následující světle modré plochy. Ta slouží pro zobrazení vlastního grafu, ve kterém jsou jednotlivé činnosti projektu představované hranami se zadanými hodnotami daných veličin. Uzly pak slouží k podržení hodnot o tom, kdy mají následující činnosti začít. Vše se dozvíte v následujících tutoriálech k tomu určených, nyní pouze dvakrát poklepejte na onu světle modrou plochu níže.*

Důležité oblasti v textu u jednotlivých tutoriálů jsou samozřejmě zvýrazněny, ať už tučně nebo kurzívou. Jakmile uživatel úspěšně přidá nový uzel, pozadí grafu se vrátí do normálního stavu a zobrazí se následující pochvalná hláška:

Gratulujeme! Právě jste přidali svůj první uzel! Pro pokračování klikněte na zelenou šipku vpravo.

5.1.2 Stránka Uzly

Tato stránka uživatele podnítí k úpravě již v grafu přidaného uzlu, respektive jeho identifikačního čísla. Pokud je uživatel již obeznámen s metodou GERT, může se pokusit změnit i typ uzlu, ovšem tomu se věnuje jiný tutoriál. Po stranách grafu jsou pro názornost na této stránce zobrazeny obrázky dialogu pro úpravu uzlu a výsledku po změně identifikačního čísla uzlu. Další text se nachází pod samotným grafem. Ten uživatel upozorňuje na možnost odstranění uzlu, ke které lze přistoupit pomocí dialogu pro úpravu. Úvodní texty

jsou následující:

Uzly v grafu slouží jako „startovní“ bod, ze kterého vycházejí zvolené činnosti projektu. Každý uzel obsahuje své identifikační číslo (zbarveno lehce hnědou barvou), pravděpodobnost realizace (zbarvená červeně), nejdříve možný začátek následujících činností (pod identifikačním číslem) a nejpozději přípustný začátek následujících činností (pod pravděpodobností realizace). Počáteční uzel grafu má samozřejmě obě tyto posledně zmíněné hodnoty nulové, neboť všechny následující činnosti jsou rovněž počáteční a „startují“ v čase 0. Pozdní vstávání či ranní přestávka na kávu se při počátku činností neberou v potaz.

***Nyní zkuste změnit identifikační číslo uzlu** tak, že dvakrát kliknete na příslušný uzel. Poté se vám zobrazí dialog, kam můžete nové číslo zadat, načež klikněte na tlačítko „Uložit“. Můžete rovněž změnit typ uzlu – co to znamená se dozvíte v tutoriálu k metodě GERT.*

***S uzlem můžete v grafu libovolně pohybovat!** Nachází-li se nějaký uzel mimo oblast, oddálením nebo přiblížením stránky oblast zvětšíte.*

Opět si v souvislosti s těmito popisnými texty můžete všimnout neformálního jazyka se zvýrazněním emocí (například pomocí vykřičníků).

5.1.3 Stránka Hrany

Podobně jako stránka Uzly i tato stránka obsahuje pro názornost texty s obrázky dialogu, tentokrát pro přidání hrany, a výsledné podoby grafu s přidanou hranou. Pod grafem se opět nachází text s upozorněním na možnost odstranění přidané hrany s využitím dialogu pro úpravu. Kromě toho je uživateli popsán i vlastní smysl hran jakožto představitelů jednotlivých činností projektu a jejich zadání (doba trvání dané činnosti, pravděpodobnost realizace). Úvodní text je následující. Texty jsou samozřejmě uživatelům zobrazeny s patřičným zvýrazněním a rozložením.

Hrany v grafu přímo představují jednotlivé činnosti projektu, které jsou definované dobou trvání a v případě metody GERT i pravděpodobností, jestli se uskuteční/realizují. Každá hrana vede mezi dvěma uzly, z nichž jeden je počáteční a druhý koncový. Činnost začne v závislosti na dobách uvedených u počátečního uzlu, načež určuje, kdy mohou začít následující činnosti, které navazují na její koncový uzel.

***Pojďme si společně hranu vytvořit!** Jak se to dělá?*

Napřed si zvolte uzly, mezi kterými chcete danou hranu vést. Jakmile budete mít vybrán její počáteční uzel, klikněte na něj (jednou). Po kliknutí by se měl kurzor myši změnit na křížek. To značí, že byl vybrán uzel k další akci.

Když tedy máte takto kurzor myši označený křížkem, zvolte druhý koncový uzel tak, že na něj opět jednou kliknete. V tu chvíli se kurzor myši vrátí zpět do původního stavu, načež se vám okamžitě zobrazí dialog pro vytvoření hrany.

Tento dialog obsahuje informace o počátečním a koncovém uzlu (identifikační čísla) a o době trvání a v případě metody GERT pravděpodobnosti realizace dané činnosti projektu, kterou hrana představuje. Zkuste upravit dobu trvání na vámi zvolené číslo, například 3, 10, 12 apod. Jakmile budete hotovi, klikněte na tlačítko „Uložit“.

Po vytvoření nové hrany se přepočítají hodnoty nejdříve možného začátku a nejpozději přípustného začátku u jejího koncového uzlu, a to v závislosti na tom, jakou dobu trvání dané činnosti jste zvolili. Detaily vysvětlují tutoriály k metodám CPM a GERT.

Naprosto stejný postup můžete zopakovat i pro úpravu hrany!

5.1.4 Stránka Tutoriály a ukládání

Tato stránka uživatele podněcuje k vlastní úpravě grafu a ke změně popisných textů. Na rozdíl od předchozích stránek tato stránka již zobrazuje ve všech výchozích případech hlášku, jestli si uživatel po změnách nepřeje tutoriál uložit. Úvodní text obsahuje obrázek spolu s informacemi o hlavním menu, na což navazuje následující text, který vysvětluje jednotlivé položky v menu, způsob ukládání a práci s popisnými texty:

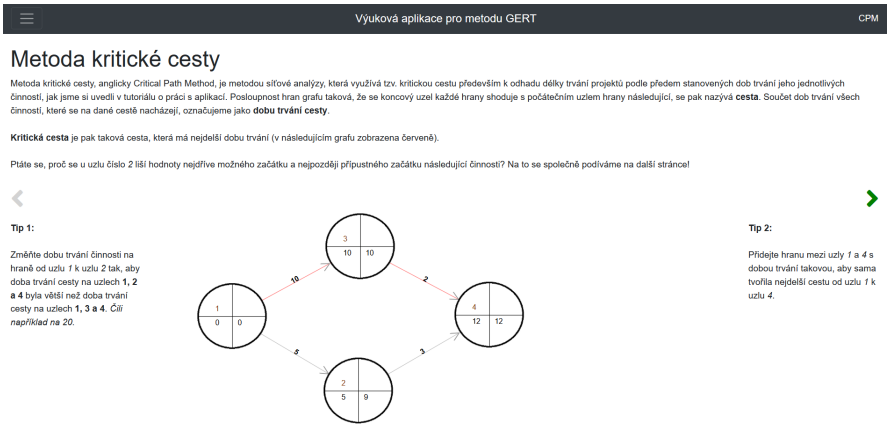
Nad seznamem všech dostupných tutoriálů, mezi kterými můžete libovolně přepínat, se nachází ještě dvě tlačítka, a to tlačítko „Zpět“ a tlačítko „Uložit do souboru“. Tlačítko „Zpět“ má stejný efekt jako kliknutí na nadpis aplikace, tj. na „Výuková aplikace pro metodu GERT“. Obě vás vrátí zpět na úvodní obrazovku s tutoriály a jejich krátkým popisem. Provedli-li jste nějaké změny v grafu (přidali jste uzel či hranu), budete dotázáni, jestli opravdu chcete tuto stránku opustit bez uložení.

Tlačítko „Uložit do souboru“ má stejný efekt jako stisknutí klávesy S. Obě možnosti vám dovolí si stáhnout soubor s definicí tutoriálu spolu se všemi změnami, které jste provedli, a to ve formátu JSON. Tento tutoriál pak můžete nalinkovat do aplikace po vzoru těch ostatních (adresář content a soubor include.json ve zdrojích aplikace).

Abyste si mohli tutoriály sami přizpůsobit, aplikace vám dovolí měnit všechny popisné texty v tutoriálu, a to tak, že jednoduše dvakrát kliknete na příslušný text, načež se vám zobrazí dialog pro změnu textu. Sem můžete vkládat text spolu s HTML značkami, kam můžete případně vložit i JavaScriptový kód a příkazy z knihovny jQuery.

5.2 Metoda kritické cesty

Tento tutoriál uživateli objasňuje smysl metody CPM a jak probíhají jednotlivé výpočty. Jedná se, v souladu se zadáním, o třístránkový interaktivní tutoriál s ukázkou grafů bez hodnot a parametrů, které jsou specifické pouze pro metodu GERT a které se netýkají přímo základní metody CPM. Stránky



Obrázek 5.3: Úvodní stránka tutoriálu pro metodu CPM s jednoduchým příkladem

tohoto tutoriálu obsahují i tipy na úpravy či tvorbu příkladů, aby uživatelé tuto metodu snáze pochopili spolu s jejím praktickým uplatněním. Tutoriál tedy obsahuje tyto tři stránky:

- **Metoda kritické cesty** – tato stránka uživatele uvede do smyslu metody CPM spolu s vysvětlením, co to znamená kritická cesta a jak se projevuje (v grafu)
- **Kritická cesta a začátky činností** – zde se uživatel dozví, jaké hodnoty se vlastně při metodě CPM počítají a jak se k nim může sám dobrat, a to spolu s vysvětlením základních pojmů
- **Složitější příklad** – na této stránce si uživatel může prakticky ověřit získané znalosti z tohoto tutoriálu a z tutoriálu pro práci s aplikací, zatímco může postupně sledovat díky přidávání/odebírání hran a uzlů v grafu samotný výpočet

Správné pochopení vysvětlení a příkladů, které jsou v tomto tutoriálu prezentované, je poté důležité k pochopení metody GERT, která je na metodě CPM založená.

5.2.1 Stránka Metoda kritické cesty

Na této stránce je po krátkém vysvětlení smyslu metody CPM a termínů cesta a kritická cesta uvedeno po stranách grafu několik tipů k jeho úpravě (například ke změně hodnot na hranách či k přidání nové hrany tak, aby se změnila kritická cesta). Úvodní text je krátký a zní následovně:

Metoda kritické cesty, anglicky Critical Path Method, je metodou síťové analýzy, která využívá tzv. kritickou cestu především k odhadu délky trvání projektů podle předem stanovených dob trvání jeho jednotlivých činností, jak jsme si uvedli v tutoriálu o práci s aplikací. Posloupnost hran grafu taková, že se koncový uzel každé hrany shoduje s počátečním uzlem hrany následující, se pak nazývá ***cesta****. Součet dob trvání všech činností, které se na dané cestě nacházejí, označujeme jako* ***dobu trvání cesty****.*

Kritická cesta *je pak taková cesta, která má nejdelší dobu trvání (v následujícím grafu zobrazena červeně).*

Ptáte se, proč se u uzlu číslo 2 liší hodnoty nejdříve možného začátku a nejpozději přípustného začátku následující činnosti? Na to se společně podíváme na další stránce!

5.2.2 Stránka Kritická cesta a začátky činností

Na této stránce je uživateli vysvětlen vlastní způsob výpočtu jednotlivých veličin, přesněji dob nejdříve možných a nejpozději přípustných začátků následujících činností u každého uzlu, a to spolu s pojmem rezerva (viz kapitola Teoretická část). Kromě vysvětlení jsou uživateli zobrazeny tipy ke změnění hodnot na hranách a pozorování změny vypočítaných hodnot těchto veličin, které přímo odrážejí vysvětlený způsob výpočtu. Uživatel je rovněž podnícen vytvořit si vlastní příklad, který by napřed vypočítal v hlavě nebo na papíře, a potom pro ověření si nechal zobrazit hodnoty získané z aplikace. Text s vysvětlením vypadá takto:

Všimněte si, že se na kritické cestě (zvýrazněná červeně) doby nejdříve možných začátků a nejpozději přípustných začátků činností v rámci daného uzlu rovnají. *Je to proto, že se doba trvání kritické cesty považuje za dobu trvání celého projektu, a logicky se tedy na kritické cestě veškerý čas spotřebuje na provedení činností (například celkový čas 10 dní se rozdělí mezi 2 činnosti, kdy každá trvá 5 dní) a nezbude žádný čas na rezervu. Pokud se ovšem dostaneme mimo kritickou cestu, tento celkový čas se na této jiné cestě může rozdělit jinak (například celkový čas 10 dní se rozdělí mezi 2 činnosti, kdy každá trvá 4 dny + zbydou 2 dny jako rezerva).*

Jak se ale tyto rezervy počítají? *Abychom to zjistili, musíme si napřed uvědomit, jak se počítají jednotlivé doby začátků.*

Nejdříve možný začátek činnosti $T_i^{(0)}$ se spočítá při jednoduchém průchodu grafem od počátečního bodu ke koncovému. Pro každou vstupující hranu do daného uzlu se spočítá doba tak, že se k z jejího počátečního uzlu zjištěné hodnotě $T_j^{(0)}$ přičte doba trvání dané činnosti t_{ji} mezi uzly j a i. Z těchto hodnot se pak zvolí ta největší z nich, která odpovídá nejdříve možnému začátku činnosti $T_i^{(0)}$.

Při zpětném průchodu od koncového bodu síťového grafu k počátečnímu lze

ke každé činnosti zjistit její nejpozději přípustný začátek $T_i^{(1)}$ (u koncového uzlu grafu se rovná nejdříve možnému začátku). Pro každou vystupující hranu se doba spočítá jako rozdíl následujícího nejpozději přípustného začátku $T_k^{(1)}$ a doby trvání následující činnosti t_{ik} mezi uzly i a k. Z těchto hodnot se pak vybere ta nejmenší, která odpovídá zjištěnému nejpozději přípustnému začátku.

Rozdílem nejpozději přípustného začátku a nejdříve možného začátku vznikne u daného uzlu rezerva.

5.2.3 Stránka Složitější příklad

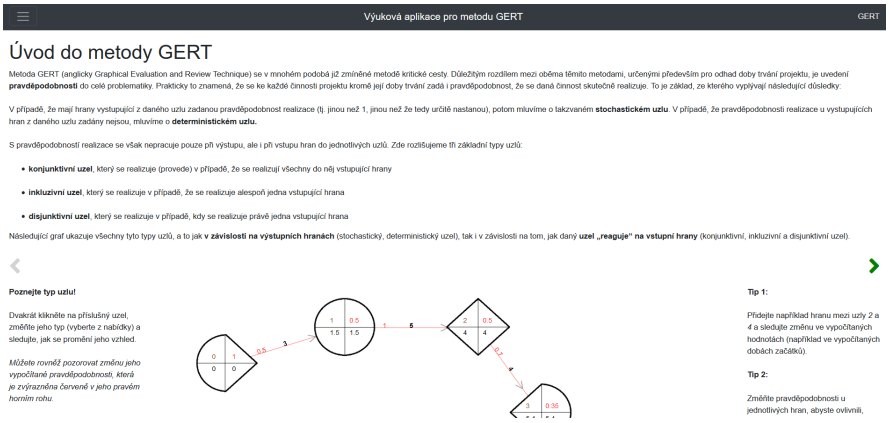
Na této stránce se v grafu nachází složitější příklad na metodu CPM, přesněji dva příklady v jednom grafu. Uživatel je požádán, aby sám graf upravil způsoby, které se naučil v tomto tutoriálu i v tutoriálu pro práci s aplikací (přidání uzlu a hrany, odstranění uzlu apod.). Tato stránka předpokládá, že na ni uživatel stráví nějaký čas tvorbou vlastních příkladů, které si následně může uložit do souboru. Úvodní text zde vzhledem k jeho stručnosti a prostému imperativnímu charakteru neuvádím.

5.3 Úvod do metody GERT

Tento tutoriál tvoří základ celé výukové aplikace. Vzhledem k tomu, že předchozí dva tutoriály objasnily základní principy práce s aplikací a fungování metody CPM, může se tento tutoriál soustředit na jádro metody GERT a její stručné vysvětlení. K tomu jsou na rozdíl od tutoriálu, který se věnoval metodě CPM, zpřístupněny možnosti přidání pravděpodobnosti realizace daných činností a změny typu uzlu, neboť toto jsou základní možnosti, které metoda GERT oproti metodě CPM navíc nabízí (viz kapitola Teoretická část). Tento třístránkový interaktivní tutoriál si tedy klade za cíl uživatelům ukázat základní principy, výpočty a smysl metody GERT pro odhad doby trvání projektů. Strukturován je podobně jako tutoriál pro úvod do metody CPM, což ilustruje i následující rozložení do stránek:

- **Úvod do metody GERT** – tato stránka představuje základní rozdíly oproti prosté metodě CPM a vysvětluje typy jednotlivých uzlů, ať už v souvislosti s výstupními (deterministický, stochastický uzel) nebo se vstupními (konjunktivní, inkluzivní, disjunktivní uzel) hranami; tato stránka rovněž obsahuje tipy a příklad k úpravě grafu

- **Výpočty metody GERT** – na této stránce se uživatel seznámí s jednotlivými výpočty, které se v metodě GERT provádějí (vypočítání dob začátků a pravděpodobnosti realizace uzlu), kde si získané znalosti může ověřit na příkladu



Obrázek 5.4: Část úvodní stránky tutoriálu pro metodu GERT

- **Příklad** – uživatel si na této stránce bude moci na složitějším příkladu pomocí úprav grafu vyzkoušet, jak metoda GERT pracuje, jak každá změna ovlivňuje vypočítané hodnoty apod.

Tyto tři stránky by měly uživatele dostatečně vybavit znalostmi, aby mohl tvořit vlastní zadání příkladu v grafu a provádět výpočty.

5.3.1 Stránka Úvod do metody GERT

Tato stránka představuje těžiště celé aplikace, neboť uživateli objasňuje základy a smysl metody GERT. Po vysvětlení významu této metody a objasnění typů uzlů je uživateli ukázán v grafu příklad, ve kterém se nachází od každého z těchto typů uzlů právě jeden. Uživatel je v popisném textu požádán, aby měnil jednotlivé typy u daných uzlů a následně sledoval proměnu jejich vzhledu, případně vypočítaných hodnot (v této fázi především pravděpodobnosti realizace u daného uzlu). Následně jsou uživateli nabídnuty dva tipy, jak může tento velmi jednoduchý příklad dále rozšířit, a to tak, že do grafu přidá novou hranu k pozorování vypočítaných hodnot, nebo naopak že změní pravděpodobnosti u jednotlivých hran tak, aby jejich koncové uzly změnily svůj typ z deterministického na stochastický či obráceně. Úvodní popisný text nad tímto příkladem zní následovně (samozřejmě je v aplikaci lépe vzhledově rozložen):

Metoda GERT (anglicky Graphical Evaluation and Review Technique) se v mnohém podobá již zmíněné metodě kritické cesty. Důležitým rozdílem mezi oběma těmito metodami, určenými především pro odhad doby trvání projektu, je uvedení ***pravděpodobnosti*** *do celé problematiky. Prakticky to znamená, že se ke každé činnosti projektu kromě její doby trvání zadá i pravděpodobnost, že*

se daná činnost skutečně realizuje. To je základ, ze kterého vyplývají následující důsledky:

V případě, že mají hrany vystupující z daného uzlu zadanou pravděpodobnost realizace (tj. jinou než 1, jinou než že tedy určitě nastanou), potom mluvíme o takzvaném ***stochastickém uzlu****. V případě, že pravděpodobnosti realizace u vystupujících hran z daného uzlu zadány nejsou, mluvíme o* ***deterministickém uzlu****.*

S pravděpodobností realizace se však nepracuje pouze při výstupu, ale i při vstupu hran do jednotlivých uzlů. Zde rozlišujeme tři základní typy uzlů:

konjunktivní uzel*, který se realizuje (provede) v případě, že se realizují všechny do něj vstupující hrany*

inkluzivní uzel*, který se realizuje v případě, že se realizuje alespoň jedna vstupující hrana*

disjunktivní uzel*, který se realizuje v případě, kdy se realizuje právě jedna vstupující hrana*

Následující graf ukazuje všechny tyto typy uzlů, a to jak ***v závislosti na výstupních hranách*** *(stochastický, deterministický uzel), tak i v závislosti na tom, jak daný* ***uzel „reaguje“ na vstupní hrany*** *(konjunktivní, inkluzivní a disjunktivní uzel).*

5.3.2 Stránka Výpočty metody GERT

Stránka Výpočty metody GERT, jak už její název napovídá, uživateli prezentuje jednotlivé vzorce, podle kterých se vypočítávají doby začátků a pravděpodobnost realizace u daného uzlu. Po vysvětlení je uživateli ukázán jednoduchý příklad se čtyřmi uzly, u kterého má po stranách uvedeny dva tipy. Příklad je jednoduchý proto, aby si uživatel mohl sám v hlavě nebo na papíře provést výpočet zjišťovaných veličin u daných uzlů, když by se změnila například hodnota pravděpodobnosti realizace u nějaké hrany. K tomuto přesně nabádá první z tipů. Druhý uživatele žádá, aby přidal nový uzel s libovolným typem a aby z/do něj vedl hranu. Díky tomu pak může sledovat změněné hodnoty daných veličin, které si může ověřit aplikováním výše uvedených vzorců. Protože jsou vzorce poněkud rozsáhlé, popisný úvodní text je nyní kratší. Zde je jeho první část:

Rozdílem ve výpočtech oproti jednodušší metodě kritické cesty (CPM) je zde ***zjišťování hodnoty pravděpodobnosti realizace daného uzlu****, a to pro každý uzel v síťovém grafu. Tato hodnota je zvýrazněna červeně a umístěna do pravého horního rohu uzlu. Dalším rozdílem je samotný výpočet hodnot nejdříve možného začátku činnosti a nejpozději přípustného začátku činnosti (viz metoda kritické cesty) v případě inkluzivních a disjunktivních uzlů. Pro každý uzel se tedy spočítají hodnoty tří veličin,* $T_i^{(0)}$, $T_i^{(1)}$ *a* $P(i)$*, podle následujících vzorců:*

Nyní následuje seznam vzorců stejný, jaký je uveden u metody GERT v kapitole Teoretická část v této práci. Za výčtem těchto vzorců se pak nachází druhá část úvodního textu:

V koncovém uzlu grafu se pak nachází hodnota pravděpodobnosti realizace celého projektu podle zadaných pravděpodobností realizací jednotlivých hran/-činností. Jak je možné vidět z výše uvedených vzorců, pravděpodobností jsou ovlivněny i začátky jednotlivých činností, což vede k lepšímu odhadu na dobu trvání celého projektu, než pokud by se pracovalo s čistě deterministickými hodnotami, jako je tomu v případě metody kritické cesty.

5.3.3 Stránka Příklad

Tato stránka je podobná stránce s názvem Složitější příklad z tutoriálu Metoda kritické cesty. Uživateli je prezentován složitější příklad spolu s popisnými texty, které ho nabádají graf dále upravovat (měnit uzly a hrany), především však nyní měnit hodnoty dob trvání a pravděpodobností realizace u jednotlivých činností projektu, které představují hrany v grafu. Některé hrany jsou v příkladu vedeny paralelně a jiné se kříží, takže má uživatel možnost sledovat různé cesty v grafu od počátečního uzlu grafu ke koncovému a měnit hodnoty na hranách tak, aby se hodnoty veličin vypočítané na dané cestě měnily. Popisný text je velmi krátký a proto ho zde neuvádím.

5.4 Příklad metody GERT

V tutoriálu s názvem Příklad metody GERT si uživatel prakticky vyzkouší všechny získané znalosti z předchozích tří tutoriálů, a to na složitějším příkladu. Tento tutoriál je jednostránkový a staví na tom, že si uživatel sám vytvoří reálný projekt (nejlépe podle svého oboru studia) spolu se zadáním jednotlivých činností a pravděpodobností, že se tyto činnosti skutečně realizují. Příklad v grafu pak může být upraven přesně tak, aby odpovídal danému vytvořenému projektu a jeho činnostem. Z tohoto důvodu tento příklad předpokládá, že na něm uživatel stráví určitý čas a prakticky si tak ověří, jestli metodu GERT, a díky ní i metodu CPM, chápe. Samotný příklad si může následně uložit a případně ho odeslat k porovnání či ohodnocení učiteli podle jejich vzájemné dohody. Zvládnutím tohoto tutoriálu by si měl být uživatel vědom základů metody GERT a měl by být schopen vypočítat a popsat jednodušší příklady, které jsou s ní spojené.

Úvodní popisný text, nacházející se nad samotným grafem, vypadá následovně (v aplikaci je samozřejmě lépe rozložen). Uživatel by se měl díky němu soustředit na zjištění doby trvání celého projektu.

Tento těžší příklad představuje ukázku výpočtů pomocí metody GERT, která byla popsána v tutoriálu s názvem Úvod do metody GERT. ***Zkuste si vymys-***

let reálný projekt *(z IT, zemědělství, stavebnictví apod.) s tolika jednotlivými činnostmi, jaký je počet hran v následujícím grafu. Každá hrana obsahuje zadání, jak dlouho má daná činnost, kterou představuje, trvat a jaká je pravděpodobnost její realizace. Sami pak zkuste ověřit, jestli výpočet sedí. Můžete si ke snazšímu pochopení postupu výpočtu vypomáhat nastavováním různých hodnot včetně typů uzlů nebo i odstraňováním hran a uzlů.*

Následně pak do svého vytvořeného projektu přidejte činnosti nové s vlastní dobou trvání a pravděpodobností realizace a uvědomte si, jak na sobě jednotlivé činnosti závisí, například jestli:

mají končit stejně, tj. vedou do stejného uzlu

mají začít stejně, tj. vedou ze stejného uzlu

jedna vyvrací druhou, tj. vedou do disjunktivního uzlu

stačí, když se z vybraných činností stane pouze jedna, tj. vedou do inkluzivního uzlu

Pro jednoduchost můžete odstranit některé ze stávajících činností a uzlů. Zkoušejte sami, jak vám výpočet vychází a jaká ***celková doba trvání*** *(doba uvedená u koncového uzlu grafu) vašeho projektu z něj vyplývá.*

5.5 Shrnutí

Základní tutoriály pro aplikaci jsou vytvořeny přesně podle funkčních požadavků takovým způsobem, aby jejich posloupnost a zároveň posloupnost jejich vnitřních stránek na sebe navazovaly vzhledem k postupně získávaným znalostem o projektech, grafech, metodě CPM a metodě GERT. Všechny tutoriály jsou tvořeny interaktivně a pomocí tipů a neformálních popisných textů uživatele podněcují k úpravě již existujících příkladů v grafu a ke tvorbě vlastních. V aplikaci se v základním provedení nachází čtyři tutoriály, a to čtyřstránkový tutoriál s názvem Práce s aplikací, který mimo jiné vysvětluje i význam uzlů a hran v grafu v rámci reprezentace projektu, třístránkový tutoriál nazvaný Metoda kritické cesty, třístránkový tutoriál Úvod do metody GERT a jednostránkový příklad s názvem Příklad metody GERT, na kterém si uživatel prakticky vyzkouší všechny dříve získané znalosti formou tvorby vlastního reálného příkladu projektu, který následně nechá analyzovat metodou GERT. První stránka tutoriálu Práce s aplikací je navíc jedinečná zvýšenou mírou interaktivity, kdy využívá zabarvení plochy grafu a pochvalnou hlášku k podnícení uživatele k práci s grafem a tedy i s celou aplikací. Uživatel se pomocí tohoto tutoriálu naučí využívat všechny funkce aplikace včetně přidávání/-úpravy hran a uzlů a ukládání vlastních tutoriálů do souboru, které může následně vložit do zdrojů aplikace. Tutoriály Metoda kritické cesty a Úvod do metody GERT se v mnohém podobají, neboť každý na první stránce objasňuje danou metodu a její smysl, na druhé stránce způsob výpočtů zjišťovaných veličin a na třetí stránce obsahuje příklad k procvičení. Tutoriál Metoda kritické cesty pro jednoduchost neobsahuje možnost práce s pravděpodobnostmi a typy

uzlů, což jsou specifika metody GERT. Poslední tutoriál s názvem Příklad metody GERT obsahuje složitější modifikovatelný příklad a návod k vytvoření vlastního zadání/projektu. Společně všechny tyto tutoriály tvoří navazující a kompaktní celek, který nicméně může být tvorbou učitelů či studentů dále rozšířen.

Kapitola 6

Ověření řešení

Ověření řešení spočívá především v testování aplikace a případných úpravách řešení tak, aby bylo v souladu s funkčními a nefunkčními požadavky na aplikaci a zároveň aby bylo dostatečně optimalizováno tak, aby umožňovalo plynulé přechody mezi jednotlivými případy užití bez zbytečných chyb či zdržení. To se konkrétně týká i přepínání mezi jednotlivými tutoriály, ať už pomocí úvodní obrazovky nebo hlavního menu, či samozřejmě přepínání mezi jednotlivými stránkami vybraného tutoriálu, kdy má překreslování grafů probíhat plynule bez záseků.

6.1 Testování aplikace

Po implementaci aplikace a vložení vytvořených základních tutoriálů jsem napřed provedl ověření požadavků na aplikaci. Toto ověření proběhlo na následujících strojích:

- **Lenovo IdeaPad Y700-17ISK** s operačním systémem Windows 10 a s procesorem Intel Core i7 6700HQ Skylake
- **Apple MacBook Pro 13,3"** s operačním systémem Mac OS a s procesorem Intel Core i5
- **Lenovo IdeaPad 510-15IKB Silver** s operačním systémem Windows 10 a s procesorem Intel Core i7 7500U Kaby Lake

Postupně jsem na těchto strojích ve webovém prohlížeči Mozilla Firefox prošel aplikací, abych ověřil splnění všech funkčních požadavků, rozdělených do následujících kategorií. Testování proběhlo v tom pořadí, v jakém jsou tu jednotlivé kategorie uvedené.

1. ověření požadavků na zobrazení tutoriálů
2. ověření požadavků na grafy

3. ověření požadavků na uložení tutoriálů
4. ověření požadavků na obsah tutoriálů (základní tutoriály)

Následovalo ověření jednotlivých zanalyzovaných případů užití, které z těchto funkčních požadavků vyplývaly. Zde došlo k drobným úpravám, které uvádím níže v sekci Úpravy řešení. Aplikace tedy základní znění těchto požadavků a případů užití po ověření skutečně splňuje, čemuž odpovídá i funkční možnosti jednotlivých vytvořených tutoriálů, především tutoriálu s názvem Práce s aplikací, který splnění funkčních požadavků svým návrhem přímo emuluje, tj. bez splnění požadavků by nemohlo dojít k jeho zavedení do aplikace.

Na všech výše uvedených strojích jsem rovněž ověřil jednotlivé nefunkční požadavky. Vzhledem k tomu, že je aplikace psána přímo pro webový prohlížeč, je tento stejně nazvaný požadavek rovněž implicitně splněn. Při testování aplikace jsem několikrát odpojil internetové připojení, abych ověřil, že aplikace skutečně funguje i bez něj. Další nefunkční požadavky se pak týkají načtení a zobrazení tutoriálů a odezvy aplikace. Načítání tutoriálů probíhá ze zdrojů aplikace přímo ze souborů, což je ověřeno samotným načtením a zobrazením jednotlivých tutoriálů v aplikaci. Ta navíc nevyžaduje žádnou instalaci, pouze funkční moderní webový prohlížeč s povoleným přístupem k lokálnímu souborovému systému, čili lze objektivně ve srovnání s ostatními aplikacemi (webovými či desktopovými) prohlásit její instalaci za snadnou. Na všech výše uvedených strojích se jak jednotlivé tutoriály, tak i úvodní obrazovka načetly maximálně do cca 350 ms, čili jsou splněny i dosti volné nefunkční požadavky na odezvu aplikace.

Na ověření architektury a implementace přímo závisí požadavek na srozumitelnou rozšiřitelnost aplikace. Implementace kromě respektování navržené architektury obsahuje zpravidla krátké funkce, kde je každá zdokumentovaná komentářem nad svou deklarací, kde daný komentář obsahuje stručný popis metody a obsah jejích parametrů. Srozumitelnosti rovněž přispívá rozdělení celé aplikace do několika souborů, jejichž obsah skriptovací jazyk JavaScript načítá dle potřeby.

6.2 Úpravy řešení

Úpravy řešení se týkaly především opravy několika chyb, které však neměly zásadní dopad na funkční požadavky aplikace. Další úpravy se pak týkaly doplnění komentářů do zdrojového kódu do míst, které nemusí být snadno pochopitelné (průchod grafem při metodě GERT apod.). Opravy chyb byly následující:

- **nespuštění simulace grafu** – umístění uzlů v grafu na příslušná místa je závislé na spuštění simulace, která je za správné umístění uzlů zodpovědná; při přepínání mezi jednotlivými tutoriály po spuštění aplikace

docházelo k tomu, že se tato simulace již znovu nespouštěla a uzly v grafu tak byly umístěny v levém horním rohu jeden na druhém, což se dalo znovu opravit tím, že se na uzel kliklo myší, simulace se spustila a tedy došlo ke správnému umístění; oprava spočívala ve spuštění této simulace přímo po načtení nového grafu (a tedy tutoriálu)

- **tlačítko „Zpět“ v hlavním menu se chovalo jako přepínač** – po kliknutí na tlačítko „Zpět“ zůstalo toto tlačítko zabarvené, a tedy se chovalo jako nadpis tutoriálu, kdy právě zvýrazněný název tutoriálu odpovídá skutečně vybranému tutoriálu; oprava spočívala v odebrání CSS třídy, která způsobovala zvýraznění tlačítka, a to po úspěšném kliknutí na tlačítko

- **neposunutí okna prohlížeče při přepnutí na další stránku tutoriálu** – pokud byl úvodní text tutoriálu delší a navigační šipky pro přepnutí na další stránku tutoriálu se tak objevily až níže po scrollování okna prohlížeče, při přepnutí na další či předchozí stránku docházelo k tomu, že se okno prohlížeče neposunulo zpět na horní pozici, tedy další či předchozí stránka tutoriálu se zobrazila posunutá (scrollovaná) přesně na pozici předchozí stránky; oprava spočívala v umístění JavaScriptového kódu s využitím knihovny jQuery, kdy se po každém kliknutí na zpřístupněnou navigační šipku spustí kód, který vrátí okno prohlížeče (oblast) zpět do výchozí pozice

Neuvádím zde drobné kosmetické úpravy v názvech proměnných, ke kterým docházelo jak během implementace, tak i během ověřování aplikace, a to z toho důvodu, aby byl její kód snadno pochopitelný a přístupnější k dalším rozšířením ze strany uživatelů.

6.3 Možnosti rozšíření aplikace

Hlavní oblastí rozšíření aplikace zůstávají samozřejmě samotné tutoriály, které lze jednoduše vytvořit úpravou popisných textů a grafu přímo v aplikaci s využitím následného uložení do souboru. Tento soubor pak lze nalinkovat do aplikace po vzoru již existujících tutoriálů přímo do jejích zdrojů, či ho lze odeslat k ohodnocení učiteli, pokud se na tom student a učitel vzájemně domluví. Popisné texty mohou být libovolné a obsahovat kromě zvýraznění textu i různé barvy, obrázky, tabulky či interaktivní příklady přesně dle toho, co umožňuje značkovací jazyk HTML (s využitím CSS stylů) a skriptovací jazyk JavaScript. Graf samotný dovoluje vytvořit libovolný příklad na odhad doby trvání projektu s využitím metod CPM a GERT. Úprava tutoriálu ve vlastní aplikaci poskytuje větší srozumitelnost a rychlost, než pokud by uživatel musel veškeré údaje do předformátovaného souboru zadávat ručně, především co se týče různých pozic uzlů v grafu a definice výchozích a koncových uzlů hran.

Druhou možností je rozšířit samotnou aplikaci, což je díky jednoduché definici metod a pomocných komentářů relativně snadné. Lze například snadno do komponenty *model* aplikace přidat výpočty pro další metody, například pro metodu PERT, a to po vzoru již existujících výpočtů pro metodu GERT. Lze rovněž rozšiřovat možnosti samotného grafu a jeho vykreslování (komponenta *view*). Existence pouze jediné HTML šablony umožňuje snadno definovat strukturu aplikace a místa pro zobrazení jednotlivých komponent (grafu, popisných textů, navigace apod.). To samé se týká i souboru s definicemi CSS stylů, kde lze velice rychle a jednoduše měnit vzhled celé aplikace a libovolných jejích prvků.

Načítání souboru je případně možné řešit kromě načítání z lokálních zdrojů i ze serveru, kdy stačí v komponentě *controller* vložit do místa pro načítání tutoriálů vhodnou funkci. Další možností je zobrazení více grafů v jednom tutoriálu, kdy stačí různé metody pro vykreslování volat na různé SVG elementy. To vše je však již mimo rozsah této práce.

6.4 Shrnutí

Po ověření aplikace lze konstatovat, že splňuje všechny definované funkční a nefunkční požadavky. Zjištěné chyby a nepřesnosti byly při testování opraveny, což se týkalo především drobností souvisejících s opětovným zobrazením grafu mezi přepínáním jednotlivých tutoriálů a s neposunutím zobrazované oblasti v prohlížeči při přepínání mezi jednotlivými stránkami tutoriálu. Aplikace byla testována na třech strojích, a to všude s lepší než požadovanou odezvou týkající se vlastní aplikace i načítání a zobrazení tutoriálů. Pomocné komentáře a tvorba tutoriálů přímo v rámci aplikace umožňují její snadnou rozšiřitelnost, tvorbu vlastních tutoriálů s příklady a vysvětlením pro metody CPM a GERT, případně tvorbu vlastního zdrojového kódu pro zobrazování více grafů v rámci jedné stránky tutoriálu, přidání dalších metod pro odhad doby trvání projektu (například přidání metody PERT), načítání tutoriálů ze vzdáleného serveru nebo úpravu struktury a vzhledu aplikace, čehož může být rychle dosaženo editací hlavní HTML šablony a souvisejících souborů. Samotná aplikace je okomentována tak, aby i začínající studenti v oblasti informatiky mohli sami spolu se svými učiteli přidávat další funkce a tutoriály s využitím již existujících souborů.

Závěr

Výstupem této práce je výuková aplikace, která by především studentům bakalářského studia vysokých škol objasnila smysl a způsob výpočtů metod CPM a GERT při odhadu doby trvání stanoveného projektu, a to podle dob trvání a pravděpodobností realizace jeho jednotlivých činností. Ze základních procesů, které má aplikace podporovat, vyplynuly funkční a nefunkční požadavky na aplikaci, které především specifikují podobu vlastní aplikace, výukových tutoriálů a zobrazovaného grafu, kde jsou jednotlivé činnosti projektu reprezentovány hranami mezi uzly s vypočítanými hodnotami dob nejdříve možných a nejpozději přípustných začátků následujících činností, v případě metody GERT i pravděpodobností realizace. Na těchto požadavcích jsou pak založené zanalyzované případy užití, které specifikují průběh hlavních interakcí mezi aplikací a uživatelem. Tyto případy užití například popisují výběr tutoriálu z úvodní obrazovky a hlavního menu, způsob přidání nové hrany do grafu a možnost úpravy uzlu. Nefunkční požadavky pak specifikují nutnost použití aplikace přes webový prohlížeč, snadnou rozšiřitelnost aplikace a minimální dobu odezvy aplikace.

Při návrhu architektury byla po zvážení a rozpracování požadavků na aplikaci zvolena softwarová architektura MVC, která dělí zdrojový kód aplikace do tří komponent podle toho, zda se jedná o výpočetní či datovou logiku (model), vykreslování grafu a zobrazení popisných textů (view) nebo o komunikaci s uživatelem a reakci na uživatelské vstupy (controller). V rámci tohoto rozvržení lze pokračovat od této úrovně abstrakce k větší konkretizaci toho, co bude součástí jednotlivých komponent, přesněji jaké třídy a objekty. Aplikace obsahuje čtyři základní objekty, a to *Gert*, *Saver*, *Render* a *Controller*, rozdělené do několika souborů s okomentovanými a krátkými funkcemi, které podporují následnou snadnou rozšiřitelnost aplikace. Zvolené technologie vzhledem k požadavkům na aplikaci a návrhu architektury zahrnují formát dat JSON, který lze použít pro ukládání do souborů stejně jako pro práci s daty ve vlastní aplikaci, skriptovací jazyk JavaScript a podpůrnou knihovnu jQuery. Pro vzhled a styly je využita knihovna Bootstrap. V rámci těchto technologií pak došlo

k vlastní implementaci aplikace a jejích funkcí. Jednotlivé třídy jsou v jazyce JavaScript implementovány přímo jako objekty, tedy aplikace vždy využívá pouze jednu jejich instanci. Objekt *Gert* například obsahuje veškerou výpočetní logiku potřebnou pro metody CPM a GERT, a to s ohledem na průchod grafem ke koncovému uzlu grafu i na zpětný průchod směrem k uzlu počátečnímu.

Po implementaci aplikace bylo možné vytvořit čtyři základní tutoriály v souladu s požadavky na aplikaci. Tyto tutoriály se nazývají Práce s aplikací, Metoda kritické cesty, Úvod do metody GERT a Příklad metody GERT. První jmenovaný tutoriál na čtyřech stránkách interaktivně provede uživatele samotnou aplikací, tvorbou grafů a tvorbou, ukládáním a načítáním tutoriálů spolu se změnou popisných textů, kterou může uživatel spolu se změnou grafu provádět přímo v samotné aplikaci. Jednotlivé tutoriály na sebe systematicky navazují a předpokládají vstupy ze strany uživatele, především v rámci příkladů, které si může uživatel dle příslušného návodu sám upravovat.

Při testování a ověřování aplikace a jednotlivých tutoriálu nedošlo k odhalení zásadních chyb, které by znemožňovaly aplikaci naplnit její funkční i nefunkční požadavky. Drobné úpravy se týkaly především nevhodného posunutí zobrazované oblasti v okně prohlížeče při přepínání pomocí navigačních šipek na další/předchozí stránku tutoriálu, dále pak například špatného zobrazení uzlů v grafu při přepínání jednotlivých tutoriálů. Aplikace je nyní vhodná k dalšímu rozšiřování, především co se týče tvorby tutoriálů, kterou uživatel může provádět přímo ve vlastní aplikaci. Zájemci ze stran studentů či učitelů pak mají možnost aplikaci rozšířit, například o metodu PERT či o získávání tutoriálů ze vzdáleného serveru. Všechny funkce v aplikaci jsou stručně okomentovány a rozděleny do souborů tak, aby byl zdrojový kód snadno pochopitelný a modifikovatelný. Následná práce na aplikaci pak může podpořit zájem o metodu GERT u samotných studentů při tvorbě vlastních projektů a její využití místo metod CPM a PERT, které mají horší statistické výsledky (jak bylo uvedeno v Teoretické části), a to především z toho důvodu, že v omezené míře berou v potaz pravděpodobnosti, které se týkají jednotlivých činností projektu, a jednotlivé závislosti mezi činnostmi, které metoda GERT díky využití různých typů uzlů dokáže lépe vystihnout.

Literatura

[1] ROSENAU, M. D.: *Řízení projektů*. Brno: Computer Press, první vydání, 2003, ISBN 80-7226-218-1.

[2] DOLEŽAL, J.; MÁCHAL, P.: *Projektový management podle IPMA*. Praha: Grada, druhé vydání, 2012, ISBN 978-80-247-4275-5.

[3] LEPIL, O.: *Teorie a praxe tvorby výukových materiálů*. Olomouc: Univerzita Palackého v Olomouci, první vydání, 2010, ISBN 978-80-244-2489-7, [vid. 02-04-2018]. Dostupné z: `http://zvyp.upol.cz/publikace/lepil.pdf`

[4] HLINĚNÝ, P.: *Teorie Grafů [online]*. Fakulta informatiky Masarykovy univerzity, 14. 8. 2008 [vid. 02-04-2018]. Dostupné z: `https://www.fi.muni.cz/~hlineny/Vyuka/GT/Grafy-text07.pdf`

[5] KOVÁŘ, P.: *Úvod do Teorie grafů [online]*. Technická univerzita Ostrava and Západočeská univerzita v Plzni, [vid. 09-04-2018]. Dostupné z: `http://homel.vsb.cz/~kov16/files/uvod_do_teorie_grafu.pdf`

[6] FRIEBELOVÁ, J.: *Síťová analýza [online]*. Ekonomická fakulta Jihočeské univerzity v Českých Budějovicích, 28. 12. 2006 [vid. 04-04-2018]. Dostupné z: `http://www2.ef.jcu.cz/~jfrieb/rmp/data/teorie_oa/SITOVA%20ANALYZA.pdf`

[7] EKONOMICKO-SPRÁVNÍ FAKULTA MASARYKOVY UNIVERZITY: *Metody plánování projektů: Síťové grafy [online]*. [vid. 05-04-2018]. Dostupné z: `http://promis.econ.muni.cz/lecture/2/2/3/`

[8] VÍTEČKOVÁ, M.; PŘIDAL, P.: *Metoda kritické cesty [online]*. Fakulta strojní Vysoké školy báňské – Technická univerzita Ostrava, [vid. 07-04-2018]. Dostupné z: `http://books.fs.vsb.cz/SystAnal/texty/25.htm`

[9] VÍTEČKOVÁ, M.; PŘIDAL, P.: *Metoda PERT [online]*. Fakulta strojní Vysoké školy báňské – Technická univerzita Ostrava, [vid. 07-04-2018]. Dostupné z: http://books.fs.vsb.cz/SystAnal/texty/26.htm

[10] PRŮCHA, L.: *Některá důležitá rozdělení [online]*. Katedra matematiky Fakulty strojní ČVUT, [vid. 03-04-2018]. Dostupné z: https://math.feld.cvut.cz/prucha/mstp/2pu.pdf

[11] EKONOMICKO-SPRÁVNÍ FAKULTA MASARYKOVY UNIVERZITY: *Metody plánování projektů: Metoda PERT [online]*. [vid. 05-04-2018]. Dostupné z: http://promis.econ.muni.cz/lecture/2/2/6/

[12] TAYLOR, B. W.: *Project management using GERT analysis [online]*. Project Management Institute, [vid. 02-04-2018]. Dostupné z: https://www.pmi.org/learning/library/gert-graphical-evaluation-review-technique-5716

[13] KANISOVÁ, H.; MÜLLER, M.: *UML srozumitelně*. Brno: Computer Press, druhé vydání, 2007, ISBN 80-251-1083-4.

[14] COCKBURN, A.: *Writing effective use cases*. Boston: Addison-Wesley, první vydání, 2001, ISBN 02-017-0225-8.

[15] KUBÍČEK, L.: *Využití přístupů založených na modelu MVC pro tvorbu aplikací [online]*. Mendelova univerzita v Brně, 26. 5. 2011 [vid. 10-04-2018]. Dostupné z: https://is.mendelu.cz/lide/clovek.pl?zalozka=13;id=18583;studium=41885;zp=29014;download_prace=1

[16] ZAKAS, N. Z.: *JavaScript pro webové vývojáře*. Brno: Computer Press, první vydání, 2009, ISBN 978-80-251-2509-0.

[17] JSCRIPTERS.COM: *JavaScript: advantages and disadvantages [online]*. [vid. 08-04-2018]. Dostupné z: http://www.jscripters.com/javascript-advantages-and-disadvantages/

[18] THE JQUERY FOUNDATION: *jQuery [online]*. [vid. 08-04-2018]. Dostupné z: https://jquery.com/

[19] BOSTOCK, M.: *Data-Driven Documents [online]*. [vid. 08-04-2018]. Dostupné z: https://d3js.org/

[20] OTTO, M.: *Bootstrap [online]*. [vid. 09-04-2018]. Dostupné z: https://getbootstrap.com/

[21] FONTICONS, INC.: *Font Awesome [online]*. [vid. 09-04-2018]. Dostupné z: https://fontawesome.com/

[22] GREY, E.: *FileSaver.js [online]*. [vid. 09-04-2018]. Dostupné z: https://github.com/eligrey/FileSaver.js/

Příloha **A**

Seznam použitých zkratek

CPM Critical Path Method

CSS Cascading Style Sheets

DOM Document Object Model

HTML HyperText Markup Language

JSON JavaScript Object Notation

GERT Graphical Evaluation and Review Technique

PERT Program Evaluation and Review Technique

SVG Scalable Vector Graphics

UML Unified Modeling Language

Příloha **B**

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `impl` ..... implementace aplikace
  - `app` ..... implementace aplikace včetně zdrojových kódů
  - `instalace.md` ..... instalační příručka aplikace ve formátu MD
- `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `thesis.pdf` ..... text práce ve formátu PDF