Západočeská univerzita v Plzni

Fakulta aplikovaných věd

Katedra informatiky a výpočetní techniky

DIPLOMOVÁ PRÁCE

Plzeň, 2015

Jan Moulis

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

Diplomová práce

Vizualizace časové osy

Plzeň, 2015 Jan Moulis

ZÁPADOČESKÁ UNIVERZITA V PLZNI
Fakulta aplikovaných věd
Akademický rok: **2014/2015**

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Bc. Jan MOULIS**
Osobní číslo:	**A13N0118P**
Studijní program:	**N3902 Inženýrská informatika**
Studijní obor:	**Softwarové inženýrství**
Název tématu:	**Vizualizace časové osy**
Zadávající katedra:	**Katedra informatiky a výpočetní techniky**

Zásady pro vypracování:

1. Seznamte se s problematikou vizualizace grafů, s důrazem na interaktivitu.
2. Seznamte se s existujícími nástroji a technikami pro tvorbu časových os.
3. Navrhněte nástroj pro vizualizaci časové osy založené na grafu precedence událostí tak, aby uživatel mohl s grafem v reálném čase pracovat.
4. Navržené řešení implementujte.
5. Otestujte vytvořenou implementaci a její využitelnost.

Rozsah grafických prací: **dle potřeby**

Rozsah pracovní zprávy: **doporuč. 50 s. původního textu**

Forma zpracování diplomové práce: **tištěná**

Seznam odborné literatury:

dodá vedoucí diplomové práce

Vedoucí diplomové práce: **Ing. Richard Lipka, Ph.D.**
Katedra informatiky a výpočetní techniky

Datum zadání diplomové práce: **1. září 2014**

Termín odevzdání diplomové práce: **14. května 2015**

Doc. RNDr. Miroslav Lávička, Ph.D.
děkan

L.S.

Prof. Ing. Jiří Šafařík, CSc.
vedoucí katedry

V Plzni dne 15. září 2014

Prohlášení

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

V Plzni dne 11. 5. 2015

Jan Moulis

Poděkování

Touto cestou bych rád poděkoval všem, kteří mi pomáhali a podporovali mě během vytváření této práce. Zejména pak chci poděkovat vedoucímu mé práce Ing. Richardovi Lipkovi, Ph.D. za cenné rady.

Abstract

Timeline visualization

This diploma thesis deals with visualization of timeline and historical events, represented by the graph of events. The main emphasis is focused on the relationships and connections between those events and interactivity with the visualization. This thesis summarizes the theoretical knowledge, possibilities of visual representation, types of interaction and existing tools for time, events and graph visualization. The main goal and outcome of this work is the application for timeline and graph visualization. The application also includes an interactive elements that enhance and complement the visual representation. This thesis also includes performance testing of the application, verification of visual representation on real and artificial data and summarizes positives and negatives both the visualization and the application.

Abstrakt

Vizualizace časové osy

Tato diplomová práce se zabývá problémem vizualizace časových os a historických událostí, reprezentovaných grafem událostí. Hlavní důraz je zde kladen na vztahy a vazby mezi těmito událostmi a interaktivitu vizualizace. V práci jsou shrnuty teoretické poznatky o času a grafech, možnosti jejich reprezentace, druhy interakce a existující nástroje pro zobrazení času, událostí a grafů. Hlavním cílem a výstupem práce je aplikace, která dokáže časovou osu a graf událostí zobrazit a zároveň je rozšířena o interaktivní prvky, které vylepšují a doplňují samotnou vizuální reprezentaci. Součástí práce je také výkonnostní testování výsledné aplikace, ověření vizuální reprezentace na reálných a umělých datech a shrnutí pozitiv a negativ vizualizace a aplikace jako celku.

Obsah

1 Úvod

Vizualizace časových os se využívá v nejrůznějších oblastech, ale její nejčastější použití je při ukázkách a vysvětlení vývoje historických, respektive budoucích událostí, ať ve školách při výuce nebo v médiích. Existuje více možností zobrazení historických a časových událostí, ale nejběžnější je patrně časová osa. Problémem ovšem je, že u jednoduché časové osy, kde jsou události zobrazeny jako body v čase, není možné zjistit, jak na sebe události navazují, co bylo příčinou nebo následkem té dané události a jaké jsou mezi nimi vazby. Lze tedy události reprezentovat jako graf, kde vrcholy grafu tvoří jednotlivé události a hrany grafu reprezentují závislosti mezi událostmi. Dalším aspektem je interaktivita. Statické časové osy jsou dobré k některým účelům (např. tisk, obrázek na webových stránkách), ale možnost manipulovat s osou a zobrazovat dodatečné informace přináší značné výhody.

Právě tímto problémem se práce zabývá. Shrnuje teoretické poznatky, techniky, možnosti reprezentace a existující nástroje pro zobrazení času, časových událostí a grafů. Práce se snaží využít všech získaných informací a vytvořit interaktivní vizuální reprezentaci časové osy a grafu, která by mohla být použitá například jako výukový nástroj ve školách.

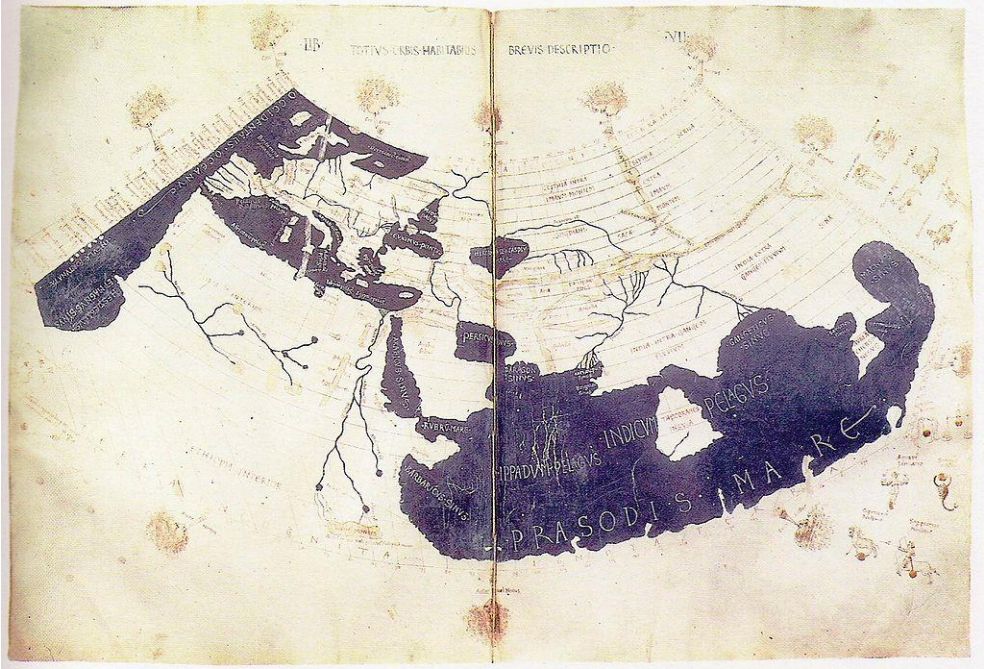
Historické události, jak již bylo řečeno, jsou uloženy ve formě grafu. Datovou vrstvu vytvořil David Merunko v rámci diplomové práce [1][1] a mezivrstvu poskytující API a algoritmy nad databází Davit Hrbáček [2][1]. Na datovou vrstvu a API (Application Programming Interface) navazuje tato práce a získaná data zobrazuje v časové ose a přidává další ovládací prvky, díky kterým je zajištěna interaktivita s osou a grafem.

Součástí práce je samozřejmě také otestování výsledné reprezentace na skutečných a umělých datech tak, aby se ověřilo, zda má interaktivita nějaký přínos při zobrazení. Díky tomu lze objevit nedokonalosti, pozitiva vizualizace a vyvodit závěry o jejím přínosu.

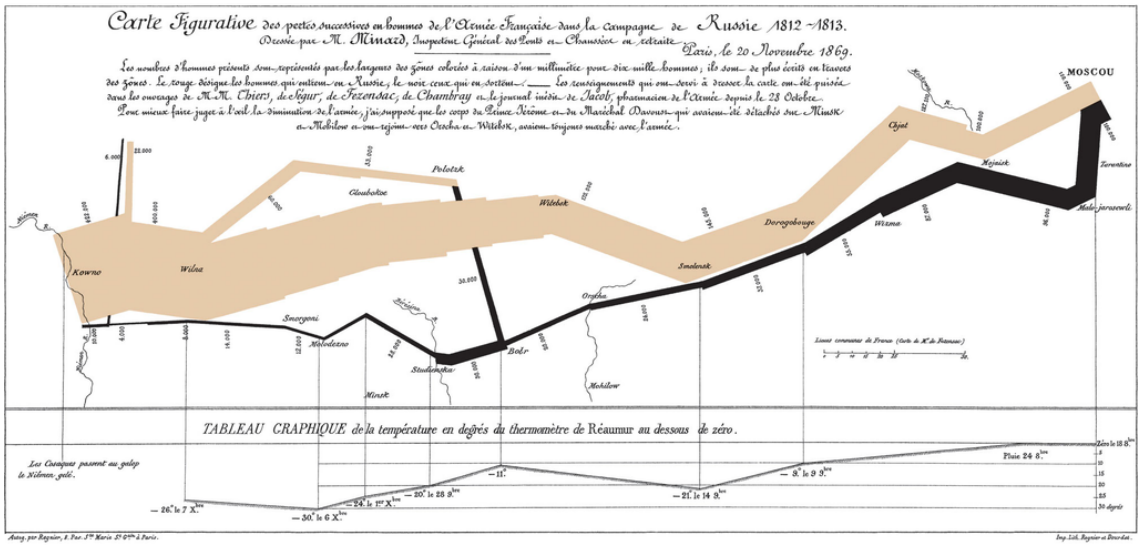
[1] práce zatím není obhájená

2 Vizualizace

Vizualizaci nejrůznějších dat využívá lidstvo již po několik tisíc let. Zejména se používala pro zobrazení map, vědeckých výkresů a diagramů. Jako příklad lze uvést Ptolemaiovu mapu (viz obrázek 2-1), mapu Číny z roku 1137 n. l. nebo Minardův diagram invaze Napoleonovy armády do Ruska (viz obrázek 2-2). Zkušenosti a koncepty získané při vytváření těchto obrázků se přenesly i do oblasti počítačové vizualizace. [3]



Obrázek 2-1: Ptolemaiova mapa světa (asi 150 n. l.); zdroj obrázku [4]



Obrázek 2-2: Minardova mapa Napoleonova tažení do Ruska; zdroj obrázku [5]

Vizualizace dat je definovaná jako zobrazení informací za pomoci grafické reprezentace. Jedná se vlastně o proces nebo prostředek pro zkoumání dat a informací z grafické reprezentace daného problému. Nejrůznější vyobrazení se používala jako komunikační prostředek ještě před vznikem psaného jazyka.

Jeden obrázek v sobě může nést velké množství informací a je snáze pochopitelný než text se srovnatelným významem. Důvod je ten, že obraz vnímáme paralelně, zatímco čtení psaného textu je sekvenční úloha. Rychlost čtení u každého jedince je také různá. Velkou výhodou grafické prezentace dat je také to, že je částečně nezávislá na jazyku pozorovatele (ale stejně tak jako čtení jsou i pozorovací a rozeznávací schopnosti závislé na dané kultuře a získaných a naučených dovednostech). Například skupina lidí, kde každý jedinec mluví jiným jazykem, dokáže pochopit graf, nebo se orientovat v mapě. [6]

2.1 Proč je vizualizace důležitá

Důvodů, proč je vizualizace dat důležitá, je mnoho, ale tím nejzásadnějším bude patrně fakt, že jako hlavní smysl pro orientaci, učení a získávání nových informací používáme zrak. [6]

Dalším důvodem může být fakt, že člověk si není schopen představit a pochopit velké množství dat, pokud tato data nejsou přehledně zobrazena. Jako příklad lze uvést množinu souřadnic bodů v 2D. Z tabulky, kde jsou vypsány X a Y souřadnice bodů, nebude člověku na první pohled jasné, jak jsou body na ploše rozmístěné. Když je ale uvidí vykresleny například ve formě grafu, ihned pozná, co body znázorňují.

Uvedu ještě další příklad, který úzce souvisí s tématem této diplomové práce. Představme si, že se chceme dozvědět, jaké byly příčiny nějaké historické události, co vše se stalo v daném období a jaký byl dopad na budoucí vývoj událostí. Kdybychom měli k dispozici pouze soupis událostí s časem a datem začátku a jejich trváním, nebude naše představa o souvislostech nikterak přesná. Pokud ale uvidíme historické události zobrazené například jako graf, kde jednotlivé události budou tvořit vrcholy grafu a hrany grafu budou znázorňovat závislosti mezi událostmi (např. příčina, důsledek, dědic, potomek, část, atd.), velmi rychle se zorientujeme a pochopíme daný úsek historie jako celek.

2.2 CG[2] a oblasti využití

Již od počátku vzniku počítačů a prvních GUI [3] se využívala počítačová grafika k znázornění nejrůznějších informací. Použití bylo ze začátku omezeno nedostatkem grafické výkonu tehdejšího HW. Dnes, se stále se zvyšujícím výkonem, již tento problém mizí. 2D grafika, která byla základem v počátcích počítačů, se v 80. a 90. letech rozšířila o 3D grafiku díky počítačovým hrám a filmovému průmyslu.

[2] CG – „computer graphics“, „počítačová grafika“; využití počítače k tvorbě grafických objektů
[3] GUI – „graphical user interface“, „grafické uživatelské rozhraní“

S různými typy CG a vizualizace se setkáváme v každodenním životě v různých oblastech [6]:

- tabulky v novinách, doplňující informace k článku,
- plány železniční a autobusové dopravy s časy příjezdů a odjezdů,
- mapy regionů a zemí,
- předpověď počasí se zvýrazněním srážkových a bouřkových mraků,
- analýzy výsledků finančních a akciových trhů,
- design mechanických a strojírenských konstrukcí,
- MRI[4] a 3D rekonstrukce zranění ze skenu CT[5],
- simulace rozsáhlých projektů a
- analýza simulací reálných fyzikálních procesů.

2.3 Důležité otázky vizualizace

Počítačová vizualizace si klade za cíl propojit lidské vnímání (zrak) a výpočetní sílu počítačů tak, aby mohl uživatel jednoduše analyzovat data a rozuměl jim. Aby mohlo být takového cíle dosaženo, musí být splněny tři základní kritéria [7]:

- expresivita,
- efektivita a
- vhodnost.

Expresivita se vztahuje k podmínce zobrazovat pouze a jen informace, které jsou skutečně obsaženy v datech, nic víc ani míň nesmí být zobrazeno. *Efektivitou* je myšleno najít takovou vizuální reprezentaci daného úkolu, která bude samo vypovídající a uživatel v ní najde potřebné informace. A nakonec *vhodnost*, která určuje poměr mezi cenou a výsledkem nebo přínosem vizualizace. Zatímco přínos vizualizace není lehké dopředu odhadnout, cenu, která například značí časovou náročnost výpočtu a přípravy dat, dopředu známe.

Každá vizualizace by měla brát tyto kritéria v potaz a hlavně se zaměřit na dva aspekty: data a úkol. Respektive na otázky: *„Co za data chceme zobrazit?“* a *„Proč chceme data zobrazit?“*. K těmto dvěma základním otázkám můžeme přidat ještě jednu: *„Jakým způsobem budeme data zobrazovat?“*.

[4] MRI – „magnetic resonance imaging“, „magnetická rezonance“; zobrazovací technika používaná ve zdravotnictví k zobrazení vnitřních orgánů lidského těla

[5] CT – „computed tomography“, „počítačová tomografie“; radiologická vyšetřovací metoda k zobrazení vnitřních orgánů pomocí rentgenového záření

3 Čas a jeho zobrazení

V následující kapitole se seznámíme s pojmem čas a s technikami, kterými je možné zobrazovat čas. V této části bylo čerpáno zejména ze zdrojů [7] a [8].

3.1 Čas

Fyzikální veličina soustavy SI označována písmenem ***t***, která určuje:

- čas – okamžik, kdy nastala nějaká událost a nebo
- dobu, po kterou událost trvala, tedy rozdíl mezi dvěma časy.

Základní jednotkou je sekunda (*s*), ke které existují dekadické násobky, respektive díly se standardními předponami: milisekunda (*ms*), mikrosekunda (μs), nanosekunda (*ns*), pikosekunda (*ps*) a dále jednotky:

- minuta (*min*), *1 min = 60 s*,
- hodina (*h*), *1 h = 60 min = 3600 s* a
- den (*d*), *1 d = 24 h = 86400 s*.

Existují také další jednotky označující delší časové období. Využívají se především v kalendářích, ale jejich délka trvání není přesně daná vlivem přestupných roků a rozdílům v letním a zimním čase:

- kalendářní den (vlivem letního času je jeho délka ± 1 hodina a kvůli korekci UTC[6] ± 1 přestupná sekunda),
- týden = sedm kalendářních dní,
- měsíc = 28 až 31 kalendářních dní v závislosti na daném měsíci a
- kalendářní rok = 365 nebo 366 (přestupný rok) kalendářních dní.

Díky tomu, že předešlé časové jednotky nemají přesně danou délku, není lehké je zobrazit na ose s pevně daným krokem. O tomto problému bude zmínka dále v kapitole 3.2.2.

3.2 Zobrazení času

Čas lze zobrazovat několika různými způsoby a nelze říci, který je ten jediný správný. Při zobrazení hlavně záleží na daném problému a na tom, co očekáváme od výsledné reprezentace. V následujících kapitolách popíši, jaké jsou způsoby zobrazení, k čemu je důležitá granularita[7] času a jaká existují časová primitiva, se kterými můžeme ve vizualizaci pracovat.

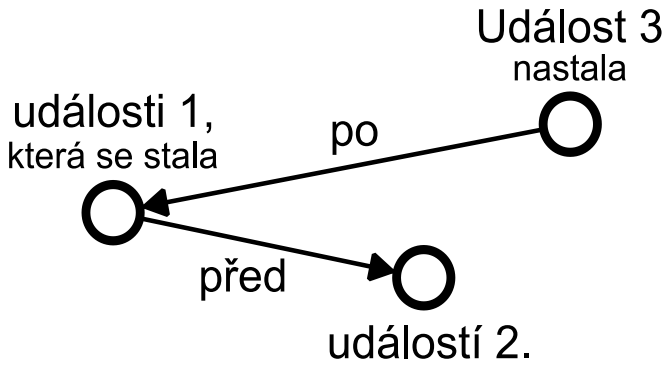
[6] UTC – „coordinated universal time“, „koordinovaný světový čas“
[7] granularita – míra detailu; čím vyšší granularita, tím větší úroveň detailu

3.2.1 Aspekty zobrazení

V následujících odstavcích bych se chtěl krátce věnovat problematice zobrazení času, jaké existují jeho druhy a jak mohou být reprezentovány.

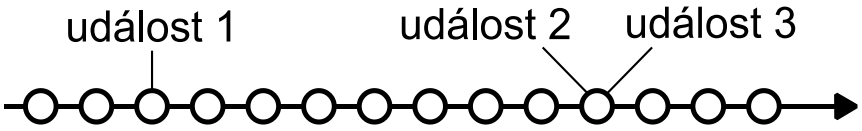
3.2.1.1 Měřítko: ordinální, diskrétní a spojité

Při použití *ordinálního* měřítka jsou události [8] v přesně daném pořadí, jak se staly, a existují mezi nimi pouze dvě závislosti: před a po, viz obrázek 3-1.



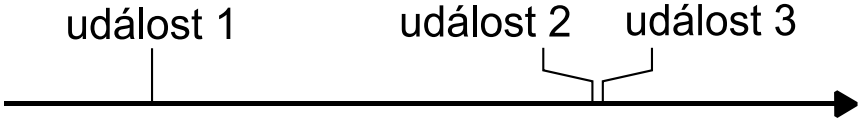
Obrázek 3-1: Ordinální měřítko

V *diskrétním* zobrazení se mapují časy na celá čísla. Jako základní jednotka času se volí pokud možno co nejmenší hodnota (sekundy nebo milisekundy), aby byla zachována přesnost. Díky tomu jsou zde opět závislosti jako v případě ordinálního měřítka, ale navíc můžeme měřit i vzdálenost, respektive dobu mezi dvěma body, viz obrázek 3-2. Z obrázku ale můžeme vidět problém tohoto zobrazení. Pokud je v tomto případě nejmenší jednotkou času minuta a události *2* a *3* nastaly v té samé minutě, nelze pak přesně určit, která ze dvou událostí se stala jako první.



Obrázek 3-2: Diskrétní měřítko

Spojité měřítko je velice podobné *diskrétnímu*, jen s tím rozdílem, že je zde čas mapován na reálná čísla. Výsledkem toho je, že vždy existuje časový bod mezi každou dvojicí bodů, viz obrázek 3-3. Nyní, na rozdíl od příkladu u *diskrétního* měřítka, jsme schopni přesně rozlišit, že *událost 2* nastala těsně před *událostí 3*.

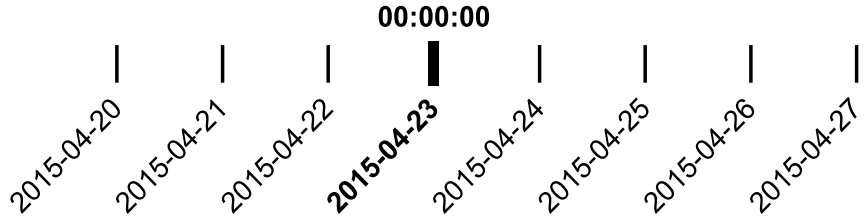


Obrázek 3-3: Spojité měřítko

[8] ve smyslu časových okamžiků, respektive bodů

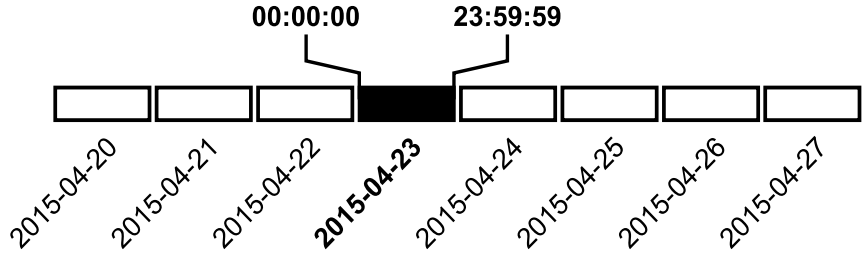
3.2.1.2 Časový rámec: bodový a intervalový

Zobrazení času ve formě samostatného *bodu* lze přirovnat k zobrazení bodu v Eukleidovském prostoru. Velikost takového bodu je nulová a neobsahuje informace například o délce trvání. Nelze tedy například určit, zda dvě události se v nějakém čase překrývaly a tak podobně, viz obrázek 3-4.



Obrázek 3-4: Zobrazení času jako samostatného bodu

Naopak *interval* má vždy velikost větší než nula a znázorňuje časový úsek od-do, viz obrázek 3-5. Interval úzce souvisí s granularitou, která bude popsána dále v kapitole 3.2.2.



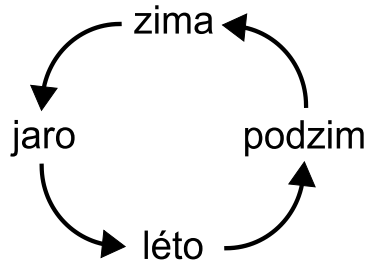
Obrázek 3-5: Zobrazení času jako intervalu

3.2.1.3 Uspořádání: lineární a cyklické

Tento aspekt je silně závislý na úkolu, který se snažíme vizualizovat. Naše chápání času je především *lineární*. Představujeme si ho jako nepřetržitý proud, který plyne od minulosti do budoucnosti, viz obrázek 3-6.

Obrázek 3-6: Lineární uspořádání času

Občas je ale dobré zobrazit čas *cyklicky*, kdy se opakují jednotlivé události v přesně daném pořadí. Nejlepším příkladem jsou roční období nebo dny v týdnu, viz obrázek 3-7. Aby bylo možné mít u cyklického upořádání nějaké smysluplné vztahy mezi událostmi, je dobré používat vztahy: *přímý předchůdce* a *přímý následovník*.



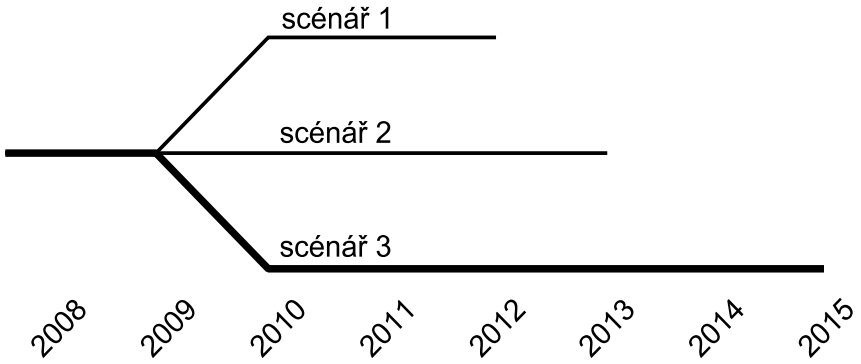
Obrázek 3-7: Cyklické uspořádání času

V některých případech lze obě uspořádání zkombinovat, pokud to úkol vyžaduje a dosáhnout tak zobrazení s cyklicky se opakujícími událostmi s lineární závislostí měřených údajů.

3.2.1.4 Pohled: řazený, větvený a vícenásobná perspektiva

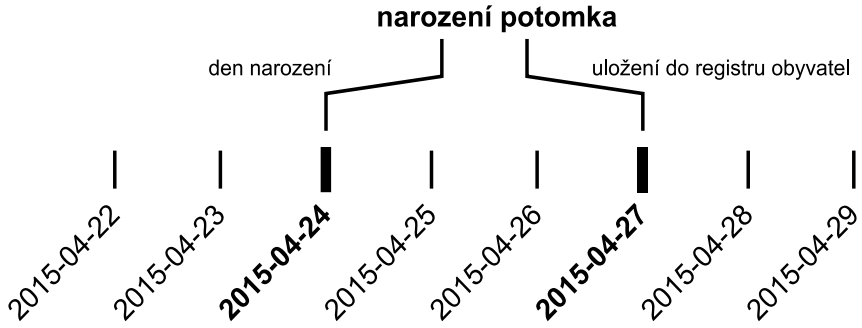
Pohledem rozumíme zobrazení časových dat a jak se na ně díváme. *Řazené* události jsou takové, které se staly po sobě jedna za druhou. Rozlišují se zde *absolutně řazené* a *částečně řazené* časové domény. *Absolutně řazené* domény mají specifikum v tom, že zde může být v jeden okamžik pouze jedna událost (nemohou se překrývat dvě a více události). Naproti tomu *částečně řazené* domény překrývání umožňují.

Větvení patří mezi komplexnější formy časových domén a spadá pod *částečné řazení*. V tomto případě běží několik paralelních větví, které mohou popisovat například porovnání alternativních scénářů. Vždy ale nastane pouze jedna z větví, viz obrázek 3-8. Tohoto se využívá při plánování, nebo při zjišťování jaký dopad mělo určité rozhodnutí.



Obrázek 3-8: Větvení času

Vícenásobná perspektiva dovoluje nahlížet na události, které se mohly stát v různých časech. Abych toto upřesnil, uvedu zde příklad. Po narození dítěte musí být tato skutečnost uvedena například do registru osob. Vytvoření záznamu ale neprobíhá okamžitě, jak ukazuje obrázek 3-9. Proto bude událost obsahovat dvoje data: skutečné datum narození a datum uložení.

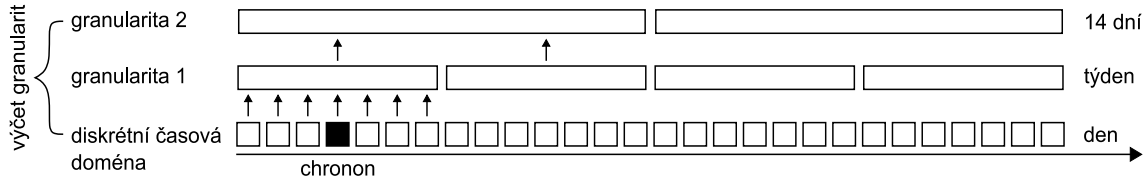


Obrázek 3-9: Vícenásobná perspektiva

3.2.2 Granularita

Jak již bylo napsáno v úvodu nadřazené kapitoly, granularita znamená míru detailu. V pojetí času se tedy jedná o přesnost (respektive úroveň), s jakou čas rozlišujeme. My jako lidé rozlišujeme čas na základní úrovně, viz kapitola 3.1. Obecně granularitou můžeme rozumět popis mapování času na větší nebo menší abstraktní úrovně.

Pokud časový model obsahuje granularitu společně s jejím výčtem a řazením, jedná se o *vícenásobnou granularitu*, viz obrázek 3-10. Dále pak existuje také *jednoduchá granularita*, kdy je čas udán pouze jako časová značka (například v milisekundách) anebo *žádná*.



Obrázek 3-10: Granularita čas (základní jednotkou den)

Většina systémů, které pracují s časem, jsou založeny na diskrétním modelu. Mají tedy zvolenou nejmenší velikost granularity času[9] a od ní se odvíjí další. Příklad z praxe je definice času v programovacím jazyce Java, respektive v její třídě `Date` a knihovně Joda-Time [9].

Java používá jako spodní granularitu času milisekundy reprezentované celým číslem long. Díky tomu může být jakákoliv časová sekvence popsána jako sled dále nerozložitelných časových intervalů, kterým se říká chronony. Také to umožňuje zapsat každý časový okamžik jako počet chrononů od určitého referenčního data (např. pro Javu: počet milisekund = chrononů uplynulých od půlnoci 1. ledna 1970, vyjádřeno přesněji: 00:00:00 1. 1. 1970).

Dalo by se tedy říci, že granularity tvoří jakési vrstvy, které se vzájemně na stejné úrovni nikdy nemohou překrývat.

[9] tzv. spodní granularitu

Vícenásobná granularita se používá při potřebě zobrazovat kalendáře[10]. Přesněji řečeno, je zde snaha najít propojení mezi zobrazením času a reálným časem tak, aby bylo pro lidi srozumitelné. Jak již bylo napsáno v kapitole 3.1, u kalendářních jednotek (dnů, měsíců a roků) není přesně daná stejná doba jejich trvání, kvůli přestupným dnům, vteřinám a rozdílnému počtu dní v jednotlivých měsících.

V této práci je zobrazení kalendářních dat stěžejní úlohou. Jako základní jednotka chronon je použita 1 milisekunda a následně granularity: hodina, den, měsíc, rok, desetiletí, století, tisíciletí a desetitisíciletí. Díky tomu je zaručena interaktivita a možnost zobrazovat různé detaily času na časové ose (více v 6.3 a 7.3.3).

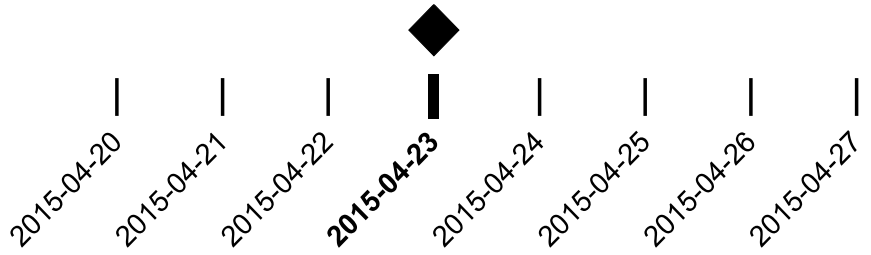
3.2.3 Časová primitiva

Když jsme si již prošli, jaké jsou aspekty zobrazení času a k čemu se využívá granularita, je důležité také uvést, jaké existují druhy časových primitiv. Jedná se o trojici základních primitiv:

- okamžik[11],
- interval a
- trvání[12].

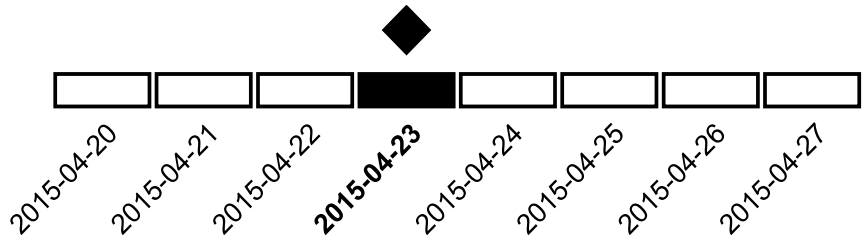
Primitiva lze rozdělit do dvou skupin: *absolutní*[13] a *relativní*[14]. Mezi *absolutní* primitiva spadá *okamžik* a *interval*, jelikož jsou přímo spjaté s časem a je u nich přesně určený začátek a délka trvání. Do druhé skupiny *relativních* primitiv patří pouze *trvání*. To popisuje pouze délku časového úseku, ale nemá informaci o tom, kdy se stal, nebo kdy skončil.

U *okamžiku* záleží na zvoleném časovém rámci (viz 3.2.1.2), kterým budeme *okamžik* popisovat. V případě volby *bodového rámce* bude *okamžik* pouhým bodem v čase bez jakékoli informace o délce trvání, viz obrázek 3-11. Naopak při volbě *intervalového rámce* bude mít *okamžik* i informaci o délce trvání, která bude záležet na granularitě, viz obrázek 3-12.



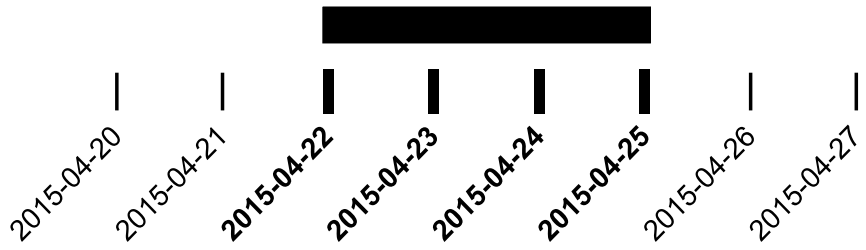
Obrázek 3-11: Okamžik jako bodový časový rámec (bez délky trvání)

[10] například Gregoriánský kalendář
[11] v angličtině „instant“
[12] v angličtině „span“
[13] označované také jako „ukotvený“, v angličtině „anchored“
[14] označované také jako „nekotvený“, v angličtině „unanchored“



Obrázek 3-12: Okamžik jako intervalový časový rámec (trvání okamžiku 1 den)

Interval značí úsek nějakého času a může být popsán dvěma okamžiky: začátkem intervalu a koncem intervalu. Další možností, jak popsat interval, jsou opět okamžiky začátku a konce a navíc délka trvání tohoto intervalu. Délka trvání může být jak kladná (do budoucnosti) tak i záporná (do minulosti). Navíc interval může anebo nemusí obsahovat krajní okamžiky. Ukázka intervalu viz obrázek 3-13.



Obrázek 3-13: Interval (začátek 22. 4. 2015, konec 25. 4. 2015)

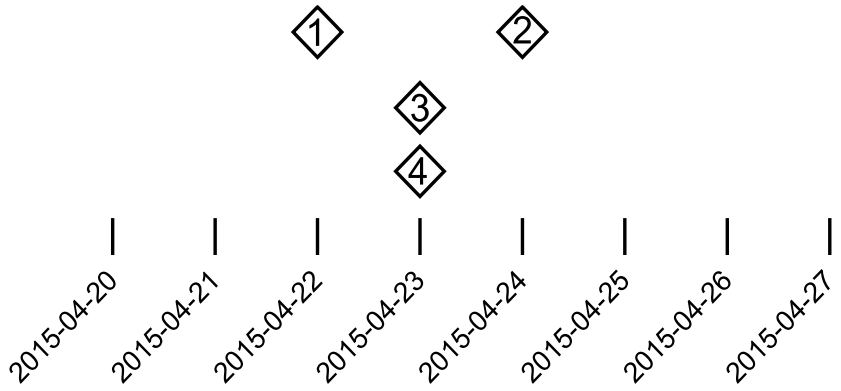
Jak již bylo řečeno, *trvání* je jediným zástupcem *relativních* primitiv a označuje pouze dobu trvání bez vazeb na začátek nebo konec. Příkladem může být: délka letních prázdnin „2 měsíce“, délka vyučovací hodiny „45 minut“ anebo příklad s délkou trvání 4 dny, viz obrázek 3-14.

Obrázek 3-14: Trvání s délkou 4 dnů (granularita dny)

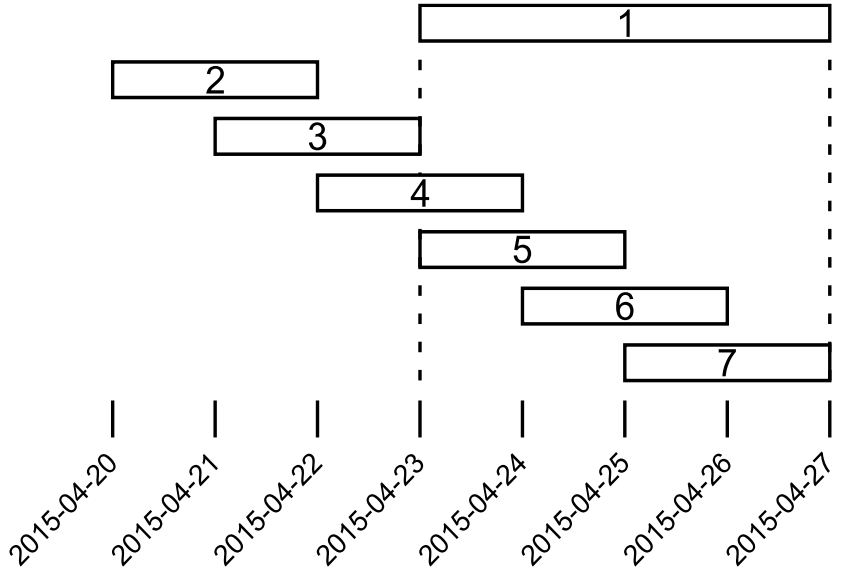
3.2.3.1 Vztahy mezi časovými primitivy

Jelikož primitiva označují určité časové období, platí mezi nimi vztahy z pohledu času, kdy se staly, zda se mohly překrývat nebo na sebe těsně navazovaly. Tyto vztahy jsou důležité při zobrazování časových os s událostmi, jelikož si pozorovatel dokáže představit sled těchto událostí a vyvodit z nich určité výsledky.

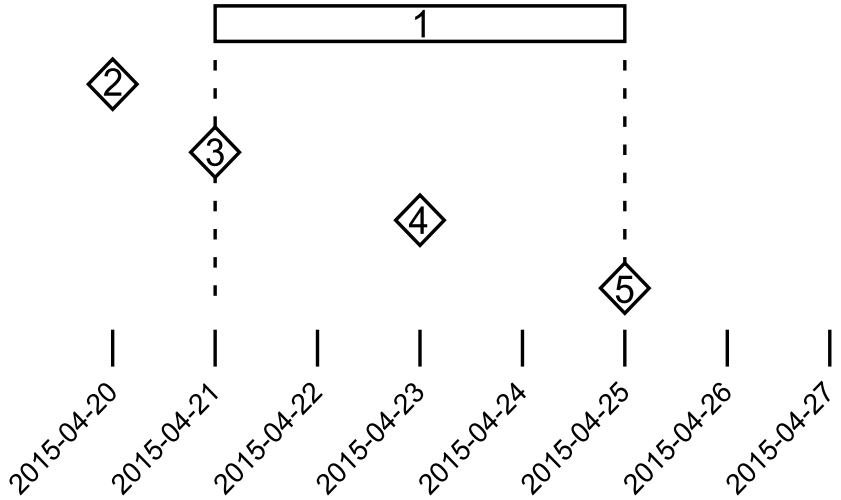
Vztahy jsou naznačeny na následujících obrázcích pro jednotlivé *okamžiky* (obrázek 3-15), pro dvojici *intervalů* (obrázek 3-16) a pro kombinaci *okamžiku* s *intervalem* (obrázek 3-17).



Obrázek 3-15: Vztahy mezi okamžiky. Tří různé vztahy: 1 ***před*** *2, 2* ***po*** *1 a 3* ***rovno*** *4.*



Obrázek 3-16: Vztahy mezi intervaly. Jedenáct různých vztahů: 2 ***před*** *1, 1* ***po*** *2, 3* ***těsně před*** *1, 1* ***těsně po*** *3, 4* ***překrývá*** *1, 1* ***překrytý*** *4, 5* ***začíná stejně s*** *1 a naopak, 6* ***během*** *1, 1* ***obsahuje*** *6, 7* ***končí stejně jako*** *1 a naopak.*



Obrázek 3-17: Vztahy mezi okamžiky a intervalem. Osm různých vztahů: 2 ***před*** *1, 1* ***po*** *2, 3* ***na začátku*** *1, 1* ***začíná*** *3, 4* ***během*** *1, 1* ***obsahuje*** *4, 5* ***na konci*** *1 a 1* ***končí*** *5.*

3.3 Nástroje a ukázky časových os

Zde bych chtěl krátce zmínit, jaké existují nástroje pro tvorbu a vizualizaci časových os se zaměřením na prezentaci událostí. Většina nástrojů je ve skriptovacím jazyce JavaScript, ale jsou zde i knihovny pro jazyk Java. Bohužel tyto knihovny jsou primárně zaměřeny na Ganttovy diagramy nebo kalendáře. Proto v této práci vznikla vlastní implementace časové osy, viz 7.3.1.5.

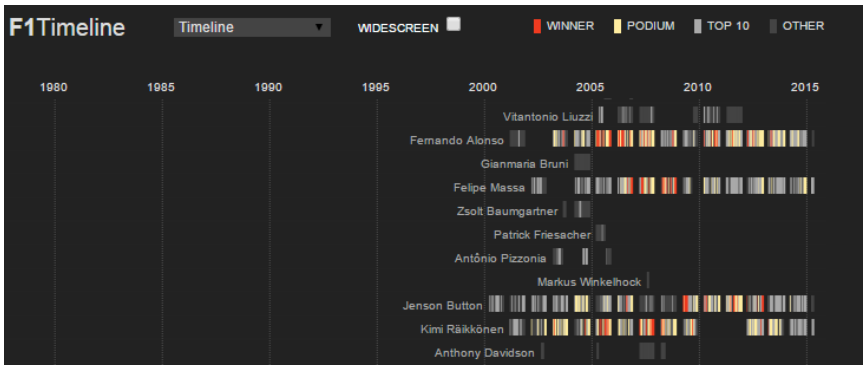
3.3.1 JavaScript knihovny

D3.js Data-Driven Documents [10]

D3.js je open-source (BSD licence) universální knihovna pro zobrazení různorodých dat se zaměřením na interaktivitu. Její velkou výhodou je univerzálnost, framework se nesnaží obsáhnout veškerou problematiku a funkcionalitu, ale poskytnout stěžejní řešení pro vizualizaci dat: efektivní manipulaci s dokumenty (výsledné reprezentace) založenými na datech. Díky tomu je nástroj velice flexibilní a může plně využít webových standardů HTML (HyperText Markup Language), SVG (Scalable Vector Graphics) a CSS (Cascading Style Sheets). Je velice rychlý a podporuje velkou řadu dynamických chování a animací pro interakci s daty a je ho možné rozšiřovat pomocí dalších komponent a pluginů.

Autor knihovny Mike Bostock ji stále udržuje aktivní (aktuální verze 3.5.5) a na webových stránkách nástroje je nespočet ukázek s vizualizací různorodých dat společně se zdrojovými kódy.

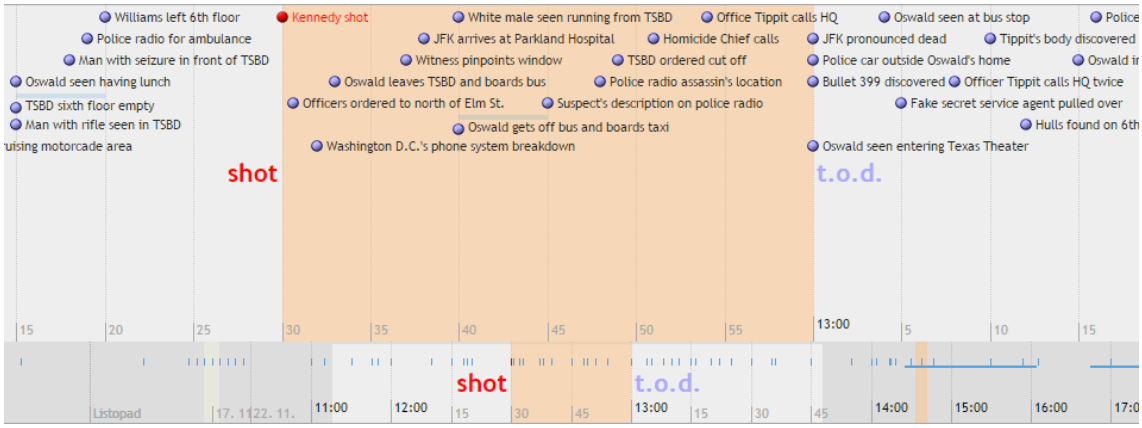
Pro práci bohužel nebylo využití této knihovny dostačující, nicméně tuto knihovnu uvádím jako víceúčelový nástroj. Obrázek 3-18 ukazuje vizualizace časové osy se zobrazením výsledků závodníků soutěže F1.



Obrázek 3-18: Vizualizace časové osy nástrojem D3.js; zdroj obrázku [11]

Timeline – Web Widget for Visulizing Temporal Data [12]

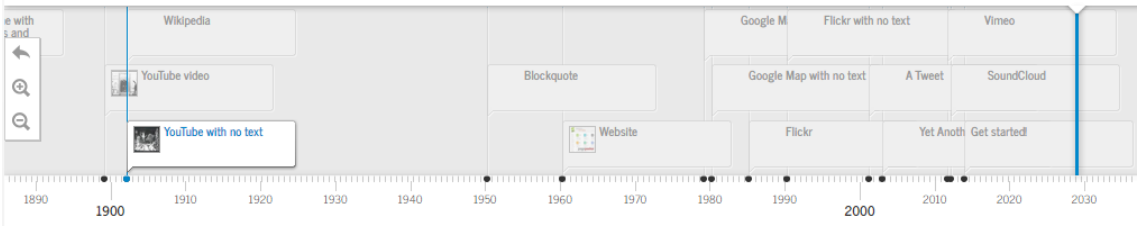
Knihovna Timeline je stejně jako D3.js zaměřena na webové prostředí a jedná se také o open-source nástroj (BSD licence). Knihovna se snaží maximalizovat interaktivitu s časovou osou a umožňuje zobrazovat detailní informace o událostech v ní zobrazených. Bohužel nepodporuje zobrazení závislostí mezi jednotlivými událostmi. Ukázka viz obrázek 3-19.



Obrázek 3-19: Ukázka zobrazení časové osy nástrojem Timeline (historická událost zastřelení Kennedyho); zdroj obrázku [12]

Timeline^{JS} [13]

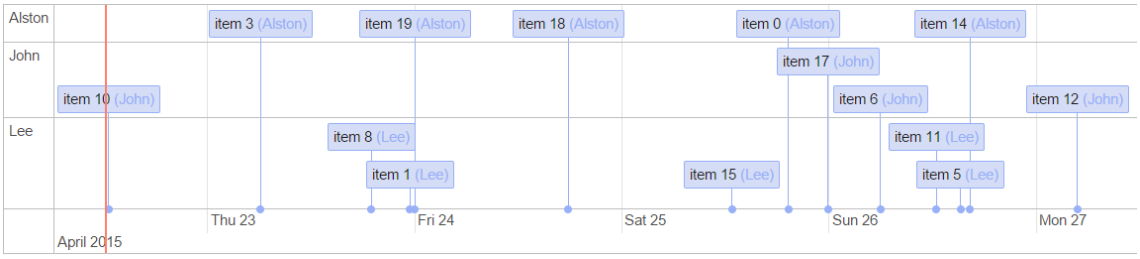
Jednoduchý open-source nástroj (Mozilla Public License, v. 2.0.), který dokáže vytvořit interaktivní časovou osu přímo z nástroje Tabulky Google nebo z JavaScriptového objektového zápisu JSON (JavaScript Object Notation). Může obsahovat odkazy na velké množství médií z různých zdrojů, nativně podporuje Twitter, Flicker, Google Maps, YouTube, Vimeo, Vine, Dailymotion, Wikipedii, SoundCloud a další. Na stránkách nástroje je popsaný jednoduchý návod, díky kterému dokáže uživatel během pár minut vytvořit a vložit na stránky časovou osu s vlastními daty. Ukázku časové osy ze stránek nástroje zobrazuje obrázek 3-20.



Obrázek 3-20: Časová osa vytvořená nástrojem Timeline^{JS}; zdroj obrázku [13]

vis.js [14]

Vis.js je dynamická open-source (Apache 2.0 a MIT licence) vizualizační knihovna zaměřená na webové prohlížeče. Jejím cílem je jednoduché používání, vysoká výkonnost při zobrazení velkého množství dynamických dat, manipulace a interakce se zobrazenými daty. Knihovna obsahuje čtyři základní moduly pro zobrazení sítí (grafů), časových os, 2D grafů a 3D grafů. Jednoduchou časovou osu ukazuje obrázek 3-21.



Obrázek 3-21: Časová osa v podání nástroje vis.js; zdroj obrázku [14]

3.3.2 Java knihovny

jaret timebars [15]

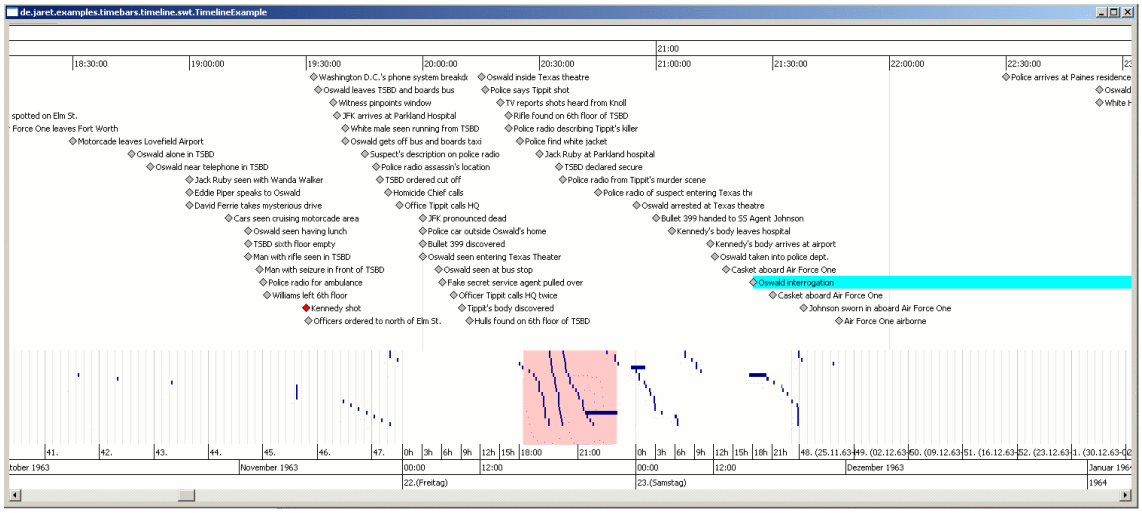
Jaret timebars je open-source (GNU Public licence) knihovna pro zobrazení datových modelů obsahujících časové intervaly, zejména tedy pro Ganttovy diagramy. Implementace zahrnuje SWING a AWT a pro většinu práce jsou použití delegáti. Aktuálně se knihovna nachází ve verzi 1.49, která byla vydána v roce 2013.

Knihovna může zobrazovat události jako separátní řádky nebo stromovou hierarchii obsahující řádky událostí. Celá se zakládá na MVC (Model View Controller) modelu, renderování může být kompletně upraveno a defaultní renderery slouží pouze k jednoduchému a rychlému použití. Podporuje interakci jak s osou samotnou, tak i s časovými intervaly (posun, zvětšování), vztahy mezi jednotlivými intervaly, zobrazení aktuální pozice nebo výběru například při zoomování. Dále nabízí internacionalizaci, ovládání osy klávesovými zkratkami, filtrování a řazení řádků s událostmi, posluchače událostí (změna události, výběr oblasti anebo změna časové osy) pro další uživatelsky definovanou interakci.

Z pohledu využití je knihovna kompatibilní s verzí Java 5 a novější, vývoj probíhal s použitím SWT 3.4, ale neměl by být problém i s verzí 3.3 a platformě se jedná o nezávislou knihovnu.

Knihovna má bohužel omezenou schopnost pracovat s granularitou (viz 3.2.2), která je zde pouze od sekund do týdnů. Pro uložení časové značky událostí je použit celočíselný datový typ závislý na platformě a tedy při přesnosti sekund je maximální rozsah zobrazení přibližně 68 let. V případě přesnosti na milisekundy (pouze jako časová značka nikoliv zobrazení) je rozsah omezen jen na 24 dní.

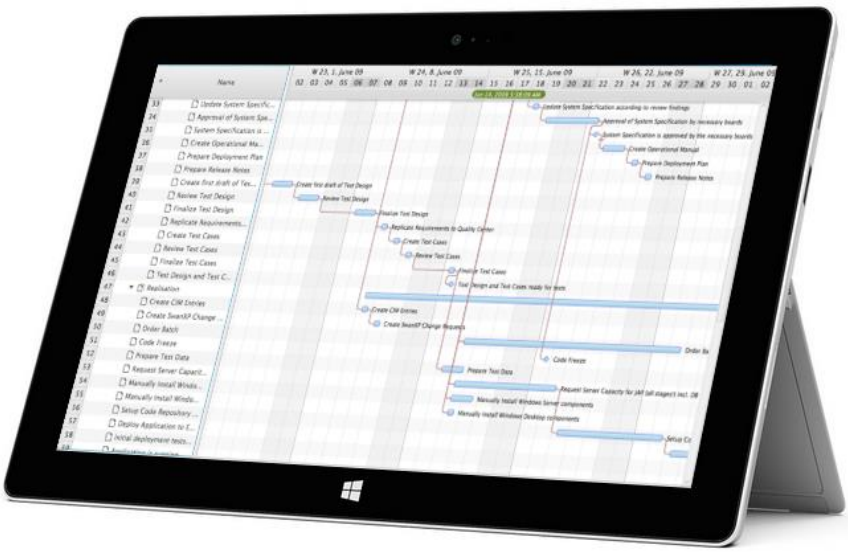
Ukázkový příklad knihovny (viz obrázek 3-22) zobrazuje stejnou historickou událost jako obrázek 3-19.



Obrázek 3-22: Časová osa a události vykreslené nástrojem jaret; zdroj obrázku [15]

DLSC [16]

Je moderní knihovna určená k tvorbě Ganttových diagramů (Java 8 a JavaFX 8) a kalendářů (pouze JavaFX 8). Bohužel není veřejně dostupná, ale její reference jsou velmi dobré. Je založena na MVC, je plně interaktivní a lze upravit layout zobrazení, časové osy a další parametry. Ukázka knihovny viz obrázek 3-23.



Obrázek 3-23: DLSC FlexGantt; zdroj obrázku [16]

4 Vizualizace grafů

Další důležitou částí je zobrazení grafů a seznámení se s již existujícími knihovnami speciálně pro jazyk Java. Při hledání dostupných nástrojů jsem vycházel z bakalářské práce Jakuba Žáčka [17].

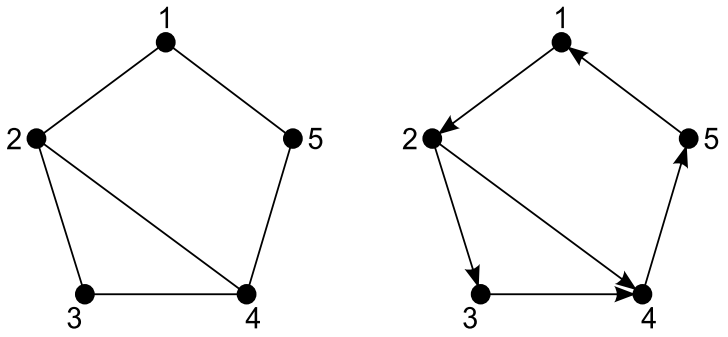
4.1 Teoretický úvod

Skripta předmětu diskrétní matematika [18] popisují graf G jako dvojici[15] $G = (V, E)$, kde V je konečná množina vrcholů (uzlů) grafu a E množina hran grafu. Grafy rozdělujeme na dva základní typy: neorientované a orientované. Pro neorientované grafy je definovaná množina hran jako:
$E \subset \binom{V}{2} = \{\{x, y\}: x, y \in V \; a \; x \neq y\}$ a pro orientované grafy $E \subset V \times V$. Hrany grafu mohou být navíc ohodnoceny. Číslo $w(e)$ udává ohodnocení hrany e a nazývá se váhou.

Vrcholy grafu x a y jsou sousední, pokud mezi nimi existuje hrana. Hrana neorientovaného grafu $\{x, y\}$ nemá směr a nelze o ní říci, kde začíná a kde končí, respektive který z dvojice vrcholů je počáteční a který je koncový. Hrany $\{x, y\}$ a $\{y, x\}$ jsou tedy totožné. Naproti tomu u orientovaného grafu na pořadí vrcholů záleží. Dvojice (x, y) značí orientovanou hranu vycházející z vrcholu x a končící ve vrcholu y. Hrany (x, y) a (y, x) označují dvě různé hrany (protichůdné).

Neorientované grafy podle definice nemohou obsahovat násobné hrany mezi stejnou dvojicí vrcholů a také nemohou obsahovat smyčky. Orientované grafy také neumožňují z definice násobné hrany, ale mohou obsahovat smyčky (počáteční i koncový vrchol je stejný).

Vrcholy grafu jsou obvykle znázorněny jako body v rovině a každá hrana je vykreslena jako čára mezi dvojicí vrcholů. U orientovaných grafů je směr hrany znázorněny šipkami. Obrázek 4-1 ukazuje reprezentaci neorientovaného a orientovaného grafu.



Obrázek 4-1: Neorientovaný graf (vlevo) orientovaný graf (vpravo)

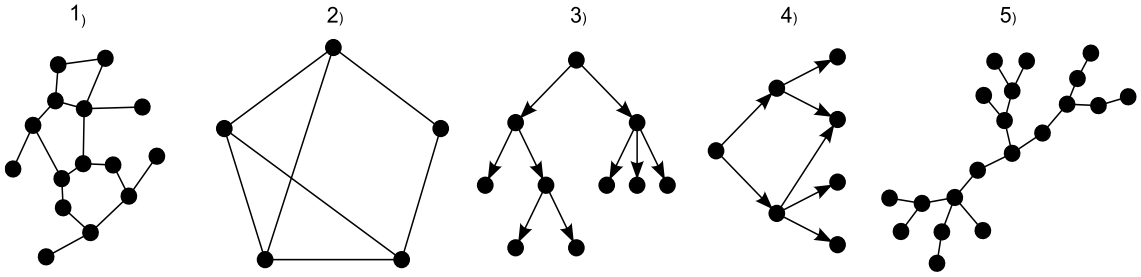
[15] označení dvojice prvků grafu V a E pochází z anglické terminologie, kde je vrchol V pojmenován *vertex* a hrana *E edge*

4.2 Zobrazení grafů

Jak již bylo napsáno a ukázáno v předešlé kapitole, grafy, respektive uzly a hrany, se nejčastěji zobrazují jako body a přímky (šipky) v rovině. Pro některé účely to ale není dostačující a je proto nutné reprezentovat uzly a hrany jiným způsobem. Příkladem může být rozdílná velikost vrcholů, která znázorňuje například jeho stupeň (počet hran, které zasahují do vrcholu), nebo zobrazení vrcholu jako tabulky s informacemi. Podobně může být u hrany zobrazena tloušťkou nebo stylem čáry dodatečná informace například o ohodnocení hrany.

Druhým aspektem při reprezentaci grafu je jeho rozložení neboli layout. Jelikož graf sám o sobě neobsahuje žádné informace o tom, kde se mají vrcholy nacházet, musí se nějakým způsobem vypočítat před samotným zobrazením. Existují různé druhy layoutů a jejich použití závisí především na daném typu úlohy a také na datech samotných. Mezi základní typy rozložení patří, viz obrázek 4-2:

- náhodné – rozmístění vrcholů nemá žádný řád,
- kruhové – vrcholy jsou rozmístěny po obvodu pomyslné kružnice,
- stromové (vertikální, horizontální) – vrcholy jsou zobrazeny od kořene k listům (každý uzel má pouze jednoho předka),
- hierarchické (vertikální, horizontální) – stejné jako stromové, ale vrchol může mít více předků,
- organické[16].



Obrázek 4-2: Rozložení grafů; 1) náhodné, 2) kruhové, 3) vertikální stromové, 4) horizontální hierarchické a 5) organické

Speciálně bych se krátce zastavil u posledního rozložení. Organický layout se vyznačuje rozložením vrcholů na základě fyzikálního modelu. Tento model funguje na principu přitažlivých F_a[17] a odpudivých F_r[18] sil. Jeho kvalita se měří pomocí takzvané energetické funkce. Ta vychází z přitažlivých a odpudivých sil a je cílem tuto energii minimalizovat a tím zlepšit výsledné rozložení. Minimalizace energetické funkce probíhá posunem vrcholů ve směrech přitažlivých a odpudivých sil, dokud energetická funkce neklesne pod definovanou maximální přípustnou hodnotu. Výsledek by tedy měl být v ideálním případě takový, že uzly spojené v grafu budou co nejblíže u sebe, zatímco žádné dva uzly, které nejsou spojeny hranou, nebudou blízko jeden druhému [19].

[16] také označovaný jako „force-based“
[17] z anglického „attraction force“
[18] z anglického „repulsion force“

Každé zobrazení grafu (nezáleží na použitém layoutu) by mělo splňovat základní podmínky „správné vizualizace“. Nelze je samozřejmě striktně dodržet a proto se berou spíše jako doporučení [19]:

- uzly a hrany by se neměly překrývat,
- hrany by neměly být zbytečně dlouhé,
- rozměry grafu by neměly být zbytečně velké či malé,
- hierarchie uzlů (např. kořen → list) by měla být snadno rozeznatelná,
- struktura, velikost a hloubka jednotlivých podstromů by měla být snadno rozeznatelná a porovnatelná s ostatními a
- rozložení by mělo být snadno měnitelné.

4.3 Grafové knihovny

Knihoven pro zobrazení grafu existuje velké množství, ale já jsem se zaměřil speciálně na knihovny pro jazyk Java a vycházel jsem z již existující práce o zobrazení grafů [17]. V následujících kapitolách bude krátké seznámení s nástroji a v kapitole 5 bude detailní popis jedné z knihoven JGraphX [20], kterou jsem původně využil v implementaci, ale poté ji nahradil vlastním vykreslováním.

4.3.1 GraphStream

Je aktivně vyvíjená knihovna [21] v jazyce Java Univerzitou Le Harve. Knihovna poskytuje nástroje a algoritmy pro reprezentaci dynamických grafů v paměti, na obrazovce a souborech. Pojem „dynamický graf“ je v knihovně definován jako tok grafových událostí. Nástroj tedy není omezen pouze na statickou reprezentaci grafu, ale může zobrazit i vývoj, respektive historii grafu a jeho komponent.

Knihovna se aktuálně nachází ve verzi 1.3, ale v brzké době by měla vyjít verze 2.0. Na webových stránkách nástroje se nachází API celé knihovny, návody a další materiály potřebné k práci s knihovnou.

Z pohledu struktury a poskytnuté implementace knihovna obsahuje 4 různé definice grafů:

- `DefaultGraph` – základní implementace, která by měla vystačit všem běžným účelům,
- `MultiGraph` – velmi podobná jako `DefaultGraph`, ale může obsahovat více hran mezi stejnými uzly,
- `AdjacencyListGraph` – málo paměťově náročná implementace grafu, ale v některých případech může být pomalejší a nakonec
- `ConcurrentGraph` – stejná implementace jako `AdjacencyListGraph` určená pro použití ve více vláknových aplikacích.

Dále obsahuje algoritmy (např. Dijkstrův, spanning-tree, A* a další), generátory grafů (náhodný, mřížkový, kruhový, úplný, atd.), metody pro import a export grafů a definici vizuální reprezentace grafu na způsob kaskádových stylů CSS (knihovna

nepodporuje veškeré možnosti klasických CSS, ale pouze se inspiruje jejím použitím a zápisem).

Stylování, jak již bylo napsáno, je založeno na kaskádových stylech CSS, které se používají v HTML. Jádro knihovny poskytuje pouze změnu zobrazení na úrovni barev a velikostí elementů. Při použití UI knihovny se rozšíří možnosti o plné využití schopností CSS. Styly mohou být definované dvěma způsoby: přímo zakódované v programu nebo jako URL (Uniform Resource Locator) adresa na CSS textový soubor (lokální nebo uložený na webových stránkách). Základní vlastnosti, které může CSS obsahovat a definovat, jsou: barva a styl vyplnění elementů (vrcholů a hran), barva a styl vykreslení elementů, pořadí elementů, stínování, rozložení, barva, styl a zarovnání textu, velikost a tvar elementů, typ tvaru hran (přímé, zaoblené, pravoúhlé, …) a další. Bohužel styly zatím nenabízí možnost nastavit u vrcholu text o více řádcích či tabulku bez implementace vlastního rendereru.

Grafický výstup knihovny ukazuje obrázek 4-3. V rámci seznámení s knihovnou a vyzkoušení jejích funkcí vznikl obrázek s organickým layoutem (vlevo). Pravá část obrázku zobrazuje síť ulic ve městě Le Harve ve Francii. Obrázek pochází z ukázkového příkladu, který je poskytnut na stránkách nástroje.

Obrázek 4-3: Ukázky knihovny GraphStream; organický layout (vlevo, vlastní obrázek) a síť ulic ve městě Le Harve ve Francii (vpravo, zdroj obrázku [21])

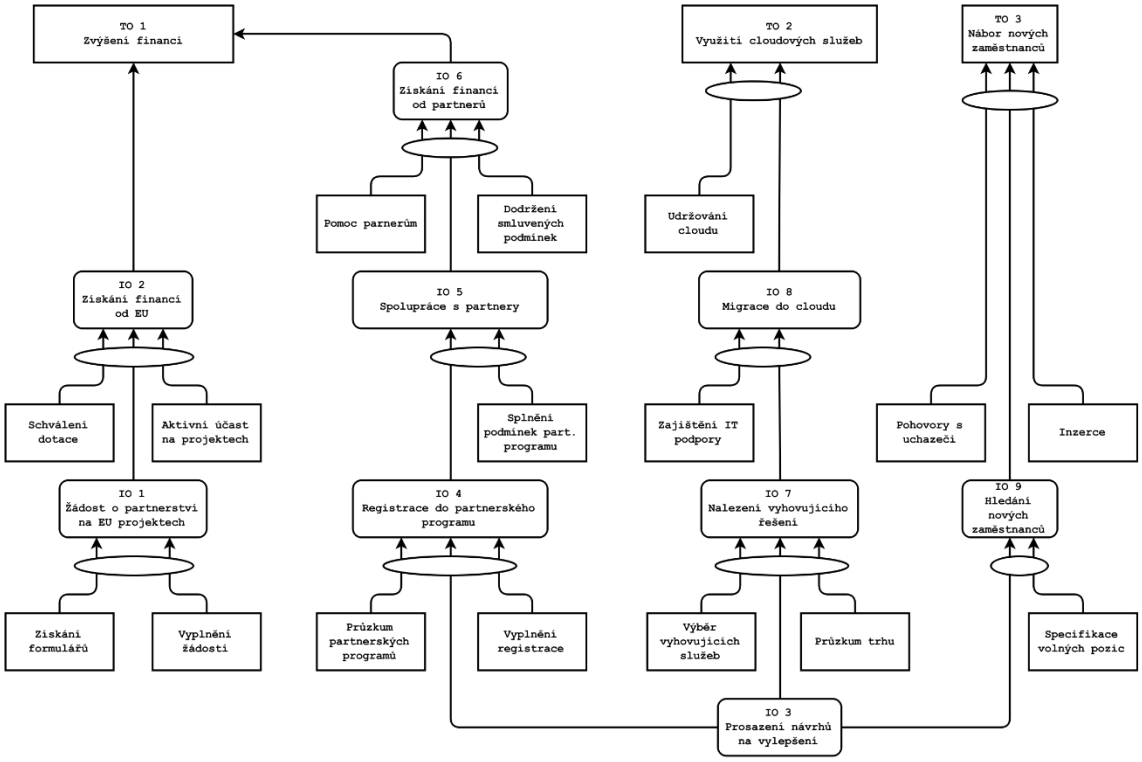
4.3.2 JGraphX

JGraphX je Java Swing knihovna pro vizualizaci grafů licencovaná pod BSD licencí. Původní jméno knihovny bylo JGraph (verze 1 až 5), JGraphX je šestým pokračováním a jméno bylo změněno, jelikož byla kompletně přepsaná implementace celého API. Knihovna je stále aktivně vyvíjená a aktuálně se nachází ve verzi 3.3.0.0. Tvůrci knihovny ji poskytují jako open-source a zároveň vyvíjí komerční JavaScript verzi mxGraph [22]. Oba dva nástroje mají stejné API, které je dobře dokumentované a ke kterému je velké množství návodů.

Struktura knihovny bude popsaná dále v kapitole 5.1, ale podobně jako GraphStream ale obsahuje implementaci pro zobrazení grafu (neměnného v čase), metody pro import a export, předpisy základních layoutů, které mohou být dále rozšířené o vlastní a definování stylu grafu a komponent v XML (Extensible Markup Language) souboru.

Knihovna se mi ze začátku velice líbila jak její strukturou, tak i funkcionalitou, kterou poskytuje a také možnostmi nastavení různých parametrů a přizpůsobením výsledné reprezentace grafu. Bohužel se ale při integraci s časovou osou objevily problémy, které nešly odstranit a které velmi omezovaly interaktivitu a plynulost programu. Vše bude popsané dále v 5.4.

Jako ukázku nástroje jsem zvolil jeho webovou podobu draw.io [23] pro tvorbu diagramů, grafů a nejrůznějších vizualizací, viz obrázek 4-4.



Obrázek 4-4: Ukázka knihovny JGraphX (resp. mxGraph); transition tree[19] v TOC[20]; vlastní obrázek

[19] také TT popisuje v detailech, jaké kroky a akce je nutné podniknout k dosažení cílů podniku

[20] TOC - „Theory of Constraints“ neboli „teorie omezení“; metoda využívaná při řízení podniků ke zlepšení chodu procesů a odstranění nežádoucích omezení

5 Knihovna JGraphX

Zde bych chtěl detailněji popsat strukturu knihovny a její použití, jelikož jsem z počátku knihovnu využíval pro vizualizaci grafu a později jsem se částečně inspiroval její strukturou při vytváření vlastní implementace.

5.1 Struktura knihovny

Základní komponenty (respektive třídy a rozhraní) knihovny jsou:

- `mxCell` – element grafu (vrchol nebo hrana),
- `mxGeometry` – umístění a velikost elementů grafu,
- `mxStylesheet` – předpis grafických stylů elementů grafu,
- `mxIGraphModel` – reprezentace grafu (wrapper grafových elementů),
- `mxGraph` – definice grafu pro vizuální reprezentaci (podle modelu a stylů),
- `mxGraphLayout` – rozložení grafu,
- `mxGraphComponent` – komponenta pro vykreslení grafu do GUI,
- `mxGraphOutline` – mini mapa/náhled grafu,
- `mxConstants` – globální konstanty knihovny a
- `mxEvents` – názvy všech globálních událostí knihovny.

Všechny třídy začínají prefixem „mx“ kvůli dodržení konvence s JavaScript verzí nástroje mxGraph [22].

5.1.1 mxCell

Předpis vrcholů a hran grafu, elementy grafu nemají separátní třídy a jsou rozlišeny pouze příznaky `vertex` a `edge` uvnitř třídy. Při vytvoření elementu je nutné specifikovat tři základní parametry: hodnotu, geometrii elementu a jeho styl. Hodnotou se rozumí jakýkoli objekt, který má element reprezentovat nebo popisovat. Geometrie popisuje umístění na vykreslované ploše a styl určuje vizuální reprezentaci elementu.

Elementy se nevytvářejí jako samostatné objekty přes konstrukci `new`, ale používají se metody, které poskytuje `mxGraph` pro vložení vrcholu a hrany do grafu: `insertVertex` a `insertEdge`.

Základní parametry při vložení vrcholu jsou: rodič, id, hodnota, geometrie a styl. Pokud nechceme vytvářet hierarchii vrcholů, kvůli jejich seskupování, lze jako rodiče uvádět `defaultParent` element, který lze získat z `mxGraph`. Identifikátor elementů je v knihovně řešen jako řetězec, geometrií u vrcholů je myšlena jeho pozice (souřadnice levého horního rohu) a velikost (šířka a výška). Styl vrcholu lze přímo definovat buď při jeho vkládání, nebo použít předefinované styly například z XML souborů, viz 5.1.3.

Parametry při vkládání hrany do grafu jsou: rodič, id, hodnota, počáteční vrchol, koncový vrchol a styl. Rodič, id, hodnota a styl jsou stejné parametry jako v případě vkládání vrcholu. Parametry počáteční a koncový vrchol jsou identifikátory vrcholů, mezi kterými se má vytvořit nová hrana.

5.1.2 mxGeometry

Geometrie popisuje u vrcholu jeho pozici (souřadnice levého horního rohu) a velikost (šířku a výšku). U hran definuje počáteční a koncový bod.

5.1.3 mxStylesheet

Styly elementů grafu musí být definovány při jejich vkládání do grafu jako poslední parametr v metodách `insertVertex` a `insertEdge`. Styly lze vytvořit třemi různými způsoby:

1. přímým zápisem jako řetězec při vytváření elementů,
2. vytvořením `mxStylesheet` třídy s definicí stylů nebo
3. definicí v XML souboru/souborech.

Přímý zápis je nejjednodušší způsob, ale nedá se nijak konfigurovat mimo zdrojový kód. Ukázka kódu 5-1 znázorňuje použití takového zápisu pro změnu barvy pozadí a rámečku elementu.

```
String style = "strokeColor=red;fillColor=green";
Object vertex = graph.insertVertex(defaultParent, "id", "Value", 0, 0, 100, 100,
style);
```

Ukázka kódu 5-1: Zápis stylu vrcholu jako řetězce

Pokud chceme definovat styly přes třídu `mxStylesheet`, je nutné nejprve získat styly, které graf obsahuje a následně do nich přidat styl vlastní a pojmenovat ho. Následně při vkládání vrcholu uvedeme jako styl jméno našeho nového stylu, viz ukázka kódu 5-2.

```
// ziskani stylu grafu
mxStylesheet stylesheet = graph.getStylesheet();
// vytvoreni hash tabulky s definici vlastniho stylu
Hashtable<String, Object> myStyle = new Hashtable<String, Object>();
// definice barvy ramecku
style.put(mxConstants.STYLE_STROKECOLOR, mxUtils.getHexColorString(Color.RED));
// definice barvy pozadi
style.put(mxConstants.STYLE_FILLCOLOR, mxUtils.getHexColorString(Color.GREEN));
// pridani vlastniho stylu mezi styly grafu
stylesheet.putCellStyle("myStyle", myStyle);
// pouziti stylu pri vkladani vrcholu
Object vertex = graph.insertVertex(defaultParent, "id", "Value", 0, 0, 100, 100,
"myStyle");
```

Ukázka kódu 5-2: Definice stylu přes mxStylesheet

Poslední možností je využití XML souborů s definicemi stylů. Soubory mohou obsahovat více stylů a tyto styly od sebe mohou dědit vlastnosti a dále je rozšiřovat. Je tedy možné vytvořit jeden základní styl a následně z něj děděním vytvořit další styly, které jej budou podle potřeby upravovat. Ukázka kódu 5-3 obsahuje XML definici stylu a ukázka kódu 5-4 načtení XML souboru a použití.

```
<mxStylesheet>
	<add as="myStyle">
		<add as="strokeColor" value="red"/>
		<add as="fillColor" value="green"/>
	</add>
</mxStylesheet>
```

Ukázka kódu 5-3: XML soubor s definicí stylů

```
// nacteni XML souboru s definicemi stylu
mxCodec codec = new mxCodec();
Document doc = mxUtils.loadDocument("URI_k_souboru");
codec.decode(doc.getDocumentElement(), graph.getStylesheet());

// pouziti stylu pri vkladani vrcholu
Object vertex = graph.insertVertex(defaultParent, "id", "Value", 0, 0, 100, 100,
"myStyle");
```

Ukázka kódu 5-4: Načtení XML s definicemi a použit

5.1.4 mxIGraphModel

Obalový objekt všech elementů grafu. Stará se o úpravu a notifikace při změnách v grafu (například vkládání a mazání uzlů). Pokud chceme provádět nějaké změny v grafu, je nutné nejprve zavolat metodu `beginUpdate` a po skončení úprav `endUpdate`. Tím bude zaručeno, že se veškeré změny úspěšně zpropagují.

5.1.5 mxGraph

Je definice grafu, ze které se s využitím modelu a stylů vytvoří vizuální reprezentace grafu. Vyvolává velké množství událostí, na které reagují další komponenty nebo uživatelské akce. Mezi tyto události například patří seskupení (`GROUP_CELLS`), rozbalení (`UNGROUP_CELLS`), přidání (`ADD_CELLS`), smazání (`REMOVE_CELLS`), posun (`MOVE_CELLS`) uzlů a další.

Zároveň se dají u grafu nastavit příznaky, zda může být například změněna velikost vrcholů, zda jsou editovatelné, jaký typ dat vrcholy obsahují, respektive jaký typ dat bude vykreslován (prostý text nebo HTML), zda se budou hrany vykreslovat v popředí a tak dále.

5.1.6 mxGraphLayout

Rozložení grafu je jednou z nejdůležitějších částí pro vizualizaci. Knihovna obsahuje předpisy pro šest druhů layoutů, které mohou být dále rozšířeny o vlastní. Základní implementace layoutů v knihovně jsou: `mxCircleLayout`, `mxCompactTreeLayout` (vertikální a horizontální), `mxEdgeLabelLayout` (rozložení bere v potaz textové popisky hran a jejich pozice), `mxFastOrganicLayout` (vylepšená implementace organického rozložení z pohledu rychlosti výpočtu), `mxHierarchicalLayout` a `mxOrganicLayout`.

V případě vlastního rozložení je nutné oddědit třídu od základního rozložení `mxGraphLayout` a přepsat tělo metody `execute`, která se spouští při vypočtení rozložení grafu. V této práci, kde je nutné mít umístěné vrcholy nad časovou osou vždy v místě, kde se staly (začaly nebo skončily), jsem vytvořil vlastní implementaci rozložení `GraphTimeBasedLayout`, který podle data začátku (respektive konce) události rozmisťuje vrcholy v horizontálním směru a zároveň počítá souřadnice vrcholů ve vertikálním směru tak, aby se vrcholy vzájemně nepřekrývaly. Stejný princip rozložení grafu jsem použil i ve finální implementaci, viz 6.2 a 7.3.4.5.

5.1.7 mxGraphComponent

Je SWING grafická komponenta grafu. Skládá se z `JScrollPane` posuvného panelu, který v sobě obsahuje samotný vykreslený graf v `JPanel` komponentě. Díky tomu je možné mít vykreslený celý graf ve vnitřní komponentě a posuvem si zobrazovat pouze část grafu nebo v grafu zoomovat. Komponenta vyvolává události `BEFORE_PAINT` a `AFTER_PAINT`, na které lze reagovat a vykreslovat vlastní objekty pod graf (událost `BEFORE_PAINT`) nebo nad graf (`AFTER_PAINT`).

Dále lze v komponentě nastavit, zda se má pod graf vykreslovat mřížka (například body), ke kterým se budou při posunu přichytávat vrcholy, zda bude povolen posun grafu nebo bude automatický, zda budou zobrazovány tooltipy při přejetí nad vrcholy a další.

5.1.8 mxGraphOutline

Je SWING grafická komponenta minimapy grafu. Při vytvoření potřebuje referenci na `mxGraphComponent`, podle které následně vykresluje náhled grafu. V náhledu lze posouvat viditelný výběr nebo ho zvětšovat a zmenšovat (zoom). Zároveň lze tuto interakci zakázat a ponechat komponentu jen jako zobrazení.

5.1.9 mxConstants

Obsahuje definice globálních konstant knihovny. Hlavními konstantami jsou zde názvy a možné hodnoty stylů elementů grafu. Při definování stylů (viz 5.1.3) je nutné vycházet z těchto konstant.

5.1.10 mxEvents

Definice všech názvů událostí, které jsou v knihovně vyvolávány a na které reaguje jak knihovna sama, tak i uživatelské posluchače. Události lze rozdělit do pěti základních skupin: aktualizace modelu (přidání a odebrání elementů), změna hodnot elementů, posun elementů, změna vykreslování elementů (sbalení, rozbalení a změna stylu) a vykreslování grafu.

5.2 Ovládání a interaktivita

Již základní implementace knihovny nabízí velké množství interakcí s grafem, které může být dále uživatelsky rozšířeno implementací vlastních posluchačů reagujících na události knihovny samotné nebo na uživatelské vstupy z klávesnice a myši. Základní funkce jsou:

- posun v grafu a posun zobrazeného výřezu v minimapě,
- zoom (zvětšení/zmenšení),
- posun vrcholů,
- vytváření hran,
- připojování hran k vrcholům,
- změna zobrazovaných hodnot u vrcholů a hran,
- přichytávání vrcholů k pomyslné mřížce a
- zobrazení tooltipů při přejetí kurzoru myši nad vrcholy.

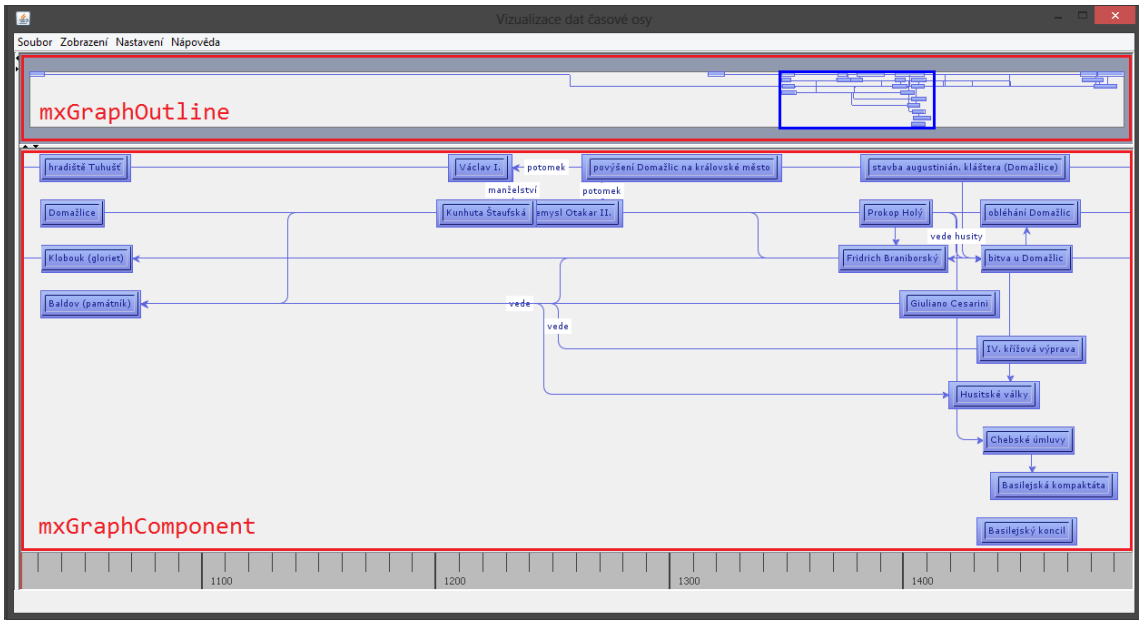
Nebudu zde rozepisovat všechny události, na které lze v knihovně reagovat (viz 5.1.10), ale pouze ukáži na jednoduchém příkladu, jak reagovat na knihovní událost. Ukázka kódu 5-5 popisuje, jak v programu reagovat na událost `BEFORE_PAINT`, která se vyvolá před vykreslením grafu. Lze ji tedy použít například pro vykreslování vlastních objektů pod graf.

```
// vytvoreni grafu spolecne s modelem
mxGraph graph = new mxGraph(new mxGraphModel());
// vytvoreni grafove komponenty
mxGraphComponent graphComponent = new mxGraphComponent(graph);

// pridani posluchace ke grafove kompnente
graphComponent.addListener(mxEvent.BEFORE_PAINT, new mxIEventListener() {
	/**
	* Osetreni udalosti.
	*/
	@Override
	public void invoke(Object sender, mxEventObject evt) {
		// ziskani platna z parametru udalosti
		Graphics g = (Graphics) evt.getProperty("g");

		// pokud mezi parametry platno neni, navrat z metody
		if (g == null)
			return;

		// vykresleni vlastnich objektu pod graf
	}
});
```

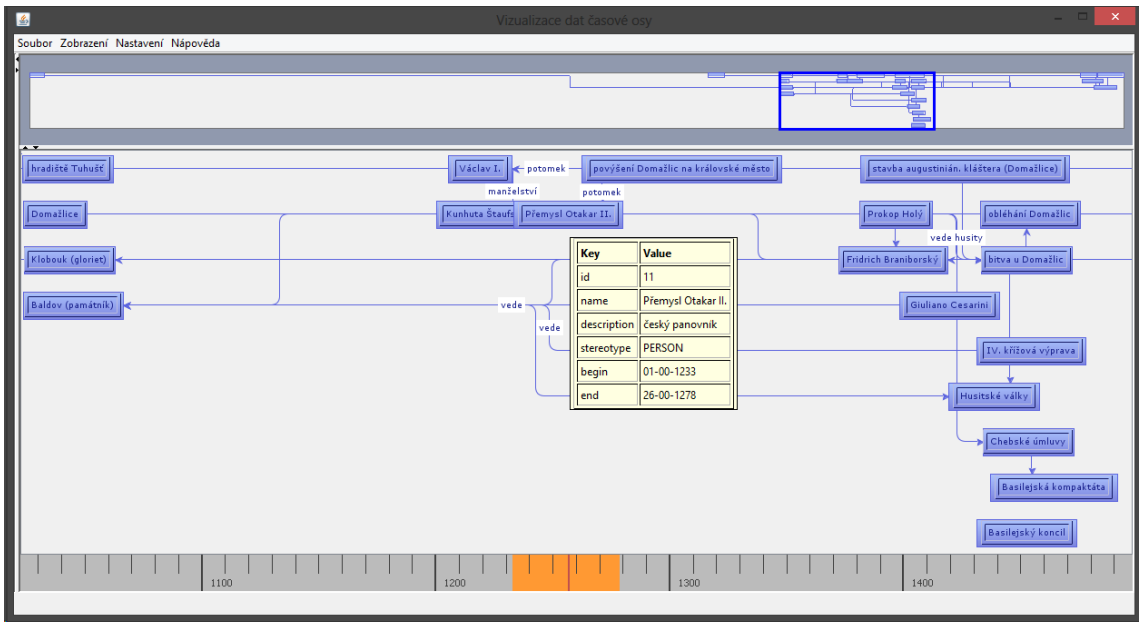
Ukázka kódu 5-5: Ošetření knihovní události BEFORE_PAINT

5.3 Ukázky zobrazení

Než jsem narazil na nedostatky a omezení knihovny, používal jsem ji ve vizualizaci k zobrazení grafu událostí. Obrázek 5-1 a obrázek 5-2 ukazují výslednou reprezentaci grafu v programu. Na prvním z nich jsou zvýrazněny použité komponenty JGraphX. Druhý obrázek zobrazuje interakci s grafem, přesněji řečeno, zobrazení tooltipu při přejetí myši nad vrcholem a zobrazení délky trvání události na časové ose (vlastní komponenta).



Obrázek 5-1: Použití knihovny JGraphX; zvýrazněné komponenty grafu a minimapy



Obrázek 5-2: Ukázka interakce knihovny; zobrazení tooltipu vrcholu „Přemysl Otakar II.“ a zvýraznění jeho života na časové ose ve vlastní komponentě

5.4 Zhodnocení

JGraphX je velice zdařeným nástrojem pro zobrazení, interakci a práci s grafy. Svědčí o tom i dobré hodnocení na stránkách knihovny a u nástroje draw.io (mxGraph knihovna – JavaScript obdoba nástroje). Knihovna je dobře komentovaná, má vlastní fórum [24] pro uživatelské dotazy a další podporu lze nalézt na stránkách stackowerflow.com [25].

Knihovna se perfektně hodí pro vytváření a vizualizaci různých typů diagramů, ale bohužel ji nelze využít pro účely zobrazení grafu jako časové osy. Důvody a problémy, které mě vedly k rozhodnutí výměny JGraphX knihovny za vlastní implementaci, byly následující:

1. Komponenta grafu `mxGraphComponent` (viz 5.1.7) je tvořena, jak již bylo psané, `JSrollPane` SWING komponentou. První problém tedy nastal při synchronizaci horizontálního posuvníku s časovou osou (samostatná komponenta) tak, aby souhlasila data na časové ose s daty začátků událostí v grafu (levá hrana vrcholu znamená začátek události). Problém jsem částečně odstranil, ale stále nastávaly případy, kdy se synchronizace nezdařila a časová osa byla vůči grafu posunuta (například pokud byl horizontální posuvník zcela vpravo a provedla se akce zoom).

2. Druhý problém také souvisí s komponentou `mxGraphComponent`. Tím, že se komponenta skládá z `JScrollPane`, ve které je uvnitř `JPanel` s celým vykresleným grafem, umožňuje knihovna velmi rychlý posun v grafu. Pokud je ovšem graf rozsáhlý (ve smyslu šířky a výšky), vnitřní panel s vykresleným grafem se musí této velikosti přizpůsobit a tím roste jeho velikost (šířka, výška) a paměťové nároky. Při přibližování v časové ose, kdy se zobrazený interval horizontálně rozšiřuje, tedy docházelo k zvětšování šířky grafu (souřadnice vrcholů v grafu musí odpovídat souřadnicím na časové ose). Se zvětšující se šířkou se zvyšovaly i paměťové nároky a čas potřebný k vykreslení tak širokého panelu. Tím docházelo k výraznému zpomalení programu.

3. Rychlost vykreslování při přibližování a oddalování. Tento problém souvisí také s bodem 2. Při každém přiblížení nebo oddálení bylo nutné přepočítat souřadnice všech vrcholů podle časové osy a následně graf překreslit. Šířka grafu se ale při přiblížení zvětšovala a vykreslování se tím zpomalovalo.

6 Úkoly a interakce s časovou osou a grafem

Zde bych chtěl uvést, jaké existují způsoby interakce s časovou osou, grafem událostí a jejich kombinací. Ne všechny z uvedených druhů interakcí byly implementovány do finálního programu a tedy některé z nich slouží jako možné rozšíření.

6.1 Úkoly a aspekty interakce

Jelikož se nezabýváme statickým zobrazením informací, ale chceme s časovou osou a grafem pracovat a získávat z nich informace, je nutné definovat úkoly, které nám k dosažení takového cíle pomohou. Úkoly k získání relevantních výsledků z vizualizace dat jsou [7]:

- zobrazení dat jako celku,
- přiblížení zajímavého úseku nebo naopak oddálení,
- odfiltrování nesouvisejících nebo nezajímavých dat,
- zvýraznění zajímavých dat a detailů a
- zobrazení závislostí mezi jednotlivými prvky dat.

Z pohledu interakce se dělí úkoly do dvou kategorií na úkoly „ukaž mi“ a doplňkové [7]. První z nich se zaměřují na okamžitou změnu vizualizace podle přání uživatele a patří mezi ně:

- ukaž mi něco jiného,
- ukaž mi jiné uspořádání,
- ukaž mi jinou reprezentaci,
- ukaž mi více či méně detailů,
- ukaž mi vyfiltrovaná data a
- ukaž mi související data.

Druhá kategorie doplňuje úkoly „ukaž mi“ o:

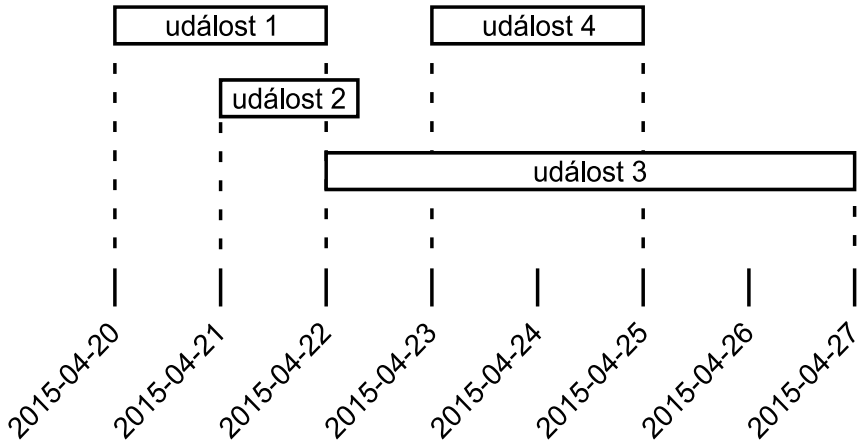
- označ data jako vybrané,
- dovol mi návrat zpět a
- dovol mi konfigurovat grafické zobrazení.

6.2 Layout

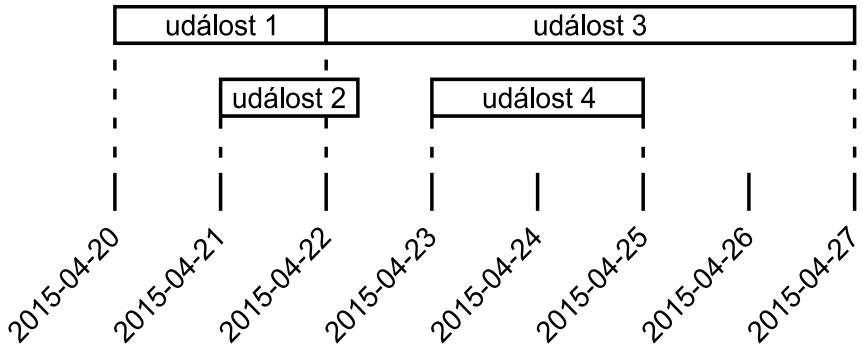
O druzích rozložení grafů již byla zmínka v kapitolách 4.2 a 5.1.6. Zde se zaměřím pouze na horizontální časově orientované rozložení. Časově orientované rozložení umisťuje vrcholy grafu tak, aby jejich pozice odpovídaly datům na časové ose. Přesněji řečeno, každému vrcholu, který reprezentuje historickou událost s datem začátku (případně i konce), se vypočte jeho horizontální souřadnice podle pozice začátku události na časové ose. Dále je nutné rozmístit vrcholy vertikálně tak, aby se vzájemně

nepřekrývaly. Pokud mají události jak datum začátku, tak i datum konce, lze na tuto skutečnost v rozložení reagovat a nastavit šířku vrcholů podle těchto dat.

Obrázek 6-1 ukazuje, jak by mohl vypadat výsledný časový layout pro čtyři události. O událostech *1*, *3* i *4* víme, kdy začaly a kdy skončily, o události *2* víme pouze datum začátku (může se jednat pouze o časový bod). Pozice a šířky událostí tedy odpovídají datům na časové ose a zároveň jsou události rozložené ve vertikálním směru tak, že se vzájemně nepřekrývají a vždy se umisťují odshora. Pouze zde záleží na rozhodnutí, zda události, které na sebe těsně navazují, mohou být umístěny vedle sebe (obrázek 6-2) nebo ne (obrázek 6-1). Rozmisťování vrcholů v obou případech probíhá v pořadí: *událost 1*, *událost 2*, *událost 3* a *událost 4*.



Obrázek 6-1: Výsledek časového rozložení; události 1, 3 a 4 mají známý datum začátku i konce; událost 2 má pouze datum začátku

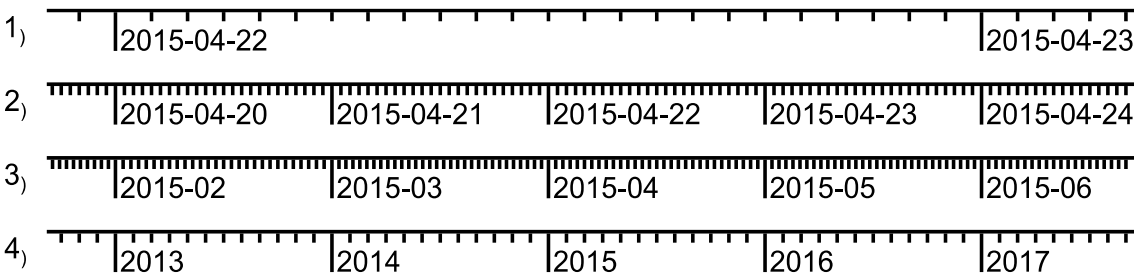


Obrázek 6-2: Časové rozložení (těsně navazující události vedle sebe)

6.3 Zoom

Přiblížení nebo naopak oddálení zobrazovaného časového úseku ve smyslu změny šířky spodní granularity, respektive chrononu a změnu granularity (viz 3.2.2). Pro představu mějme chronon definovaný jako jeden den. Při přiblížení budeme jeho šířku zvětšovat

a získáme detailnější náhled na jednotlivé hodiny daného dne a naopak při oddálení se délka dne bude zmenšovat až do změny granularity na měsíce, roky, desetiletí a tak dále, viz obrázek 6-3.



Obrázek 6-3: Zoom a změna granularity; 1) detail dne s hodinami, 2) zobrazení dnů, 3) zobrazení měsíců, 4) zobrazení roků

S tím jak se mění šířka chrononu, se samozřejmě mění i pozice určitého data na časové ose a je tedy nutné upravovat souřadnice vrcholů grafu, aby si vzájemně odpovídaly.

Při zoomování lze provádět také selekci událostí, které budou při dané úrovni přiblížení vidět. Podobné chování lze pozorovat například u webových map, kde se při přiblížení objevují názvy měst a vesnic a naopak při oddálení tyto informace mizí. Aby bylo možné takového efektu docílit, je nutné mít u každého vrcholu informaci o jeho důležitosti nebo přesně dané rozmezí, kdy se má zobrazovat a kdy skrývat. Díky práci Davida Hrbáčka [2], jejíž součástí je i výpočet pageranku[21] jednotlivých událostí, mohou mít události informaci o jejich důležitosti, díky které by mohla být selekce prováděna.

6.4 Posun

Posun může znamenat jak posun zobrazovaného časového úseku, tak ale i posun jednotlivých vrcholů grafu.

Posun v časové ose je společně se zoomem základní interakcí, bez které by byla časová osa pouze statickým zobrazením. Posun dovoluje jednoduše nahlížet do minulosti a budoucnosti a vracet se zpět.

Posun vrcholů grafu znamená změnu jeho pozice. Posun vrcholů grafu s událostmi skýtá ale menší problém. Pokud události umisťujeme tak, aby jejich datum začátku a levá hrana vrcholu odpovídaly datu na časové ose, nesmíme dovolit uživatelský posun v horizontálním směru. Jinak bychom přišli o provázání informací mezi časovou osou a grafem. Řešení tohoto problému jsou dvě. Prvním z nich je povolení pouze vertikálního posunu vrcholů. Musíme podle toho ale upravit i výpočet rozložení grafu, aby nepočítal vertikální pozici pro vrcholy, které uživatel posunul ručně. Druhým řešením je dočasné povolení horizontálního posunu vrcholů. Uživateli

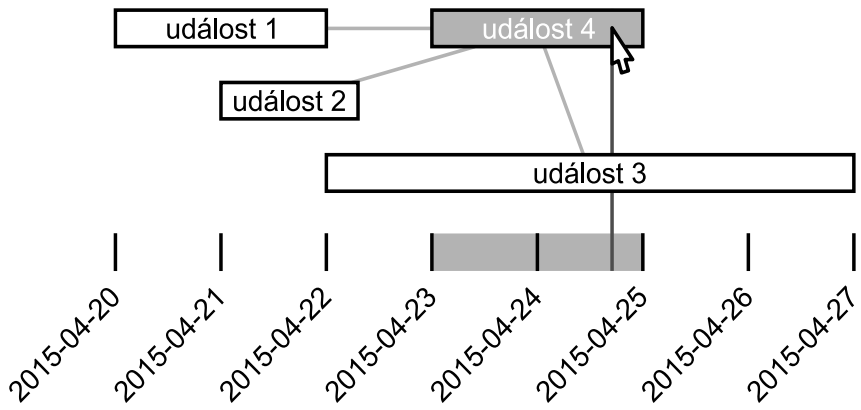
[21] pagerank – algoritmus pro ohodnocení důležitosti na základě provázání

dovolíme libovolně hýbat s vrcholy do doby, než se nově přepočte rozložení grafu (například při přiblížení nebo oddálení) a vrcholy se vrátí na své původní pozice.

6.5 Zvýraznění

Při interakci je důležité zobrazovat uživateli aktuální stav a zároveň indikovat prvky, se kterými může interagovat. Mezi zobrazení aktuálního stavu v časové ose a grafu událostí patří zvýraznění pozice kurzoru myši a výpis data a času, ve kterém se kurzor myši nachází. Díky tomu má uživatel vždy přehled o tom, v jakém časovém okamžiku se pohybuje.

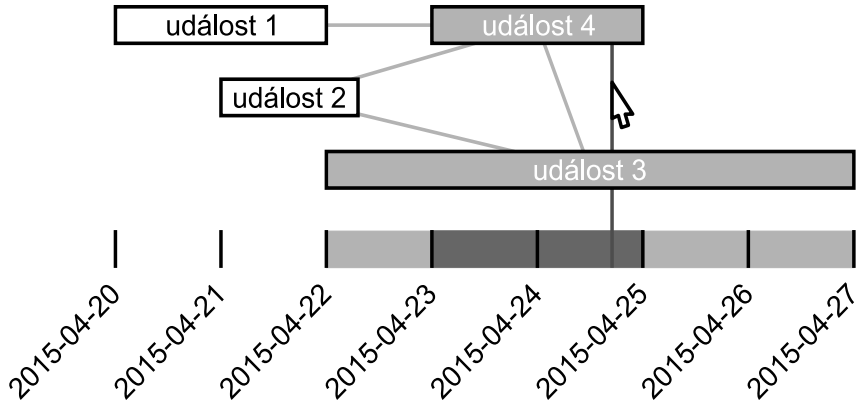
Dále do zvýraznění může patřit obarvení nebo jiné indikování objektů, se kterými může uživatel pracovat a získat o nich další informace. V případě grafu událostí se tedy jedná o zvýraznění vrcholů, pokud se kurzor myši dostane nad daný vrchol, a dále zvýraznění hran, které vedou z daného vrcholu. Jelikož vizualizace kombinuje graf s časovou osou a události v grafu obsahují časové informace o svém začátku a konci (některé události mohou obsahovat pouze jednu informace anebo žádnou), můžeme na časové ose zvýraznit časový úsek, po který daná událost trvala. Jak by mohlo zvýraznění všech popsaných objektů vypadat, ukazuje obrázek 6-4.



Obrázek 6-4: Zvýraznění pozice kurzoru, události 4, hran a časového úseku

6.6 Výběr vrcholů

Pokud funguje zvýraznění vrcholů při přejetí kurzorem myši a zobrazení dodatečných informací (hrany, časový úsek na ose, atd.), je dobré z uživatelského hlediska dovolit i výběr událostí, o které má uživatel zájem. Při výběru se opět zvýrazní vybrané vrcholy, hrany s nimi spojené a časové úseky na ose, kde mohou být navíc protínající se časové úseky odlišeny od ostatních. Toto zvýraznění je trvalé, dokud uživatel neodznačí dané události. Díky tomu lze vybrat ze všech událostí pouze ty, které souvisí se zkoumanou historickou událostí.



Obrázek 6-5: Výběr událostí 3 a 4, zvýraznění vrcholů, hran a časových úseků (barevné odlišení průniku)

6.7 Zobrazení dodatečných informací

Pouze grafické zobrazení není vždy plně vyhovující a proto je dobré vypisovat i textové informace o zvýrazněných nebo vybraných událostech. Události mohou uchovávat různé informace od základních, jako jsou například název, popis, začátek a konec události, typ události až po doplňující uživatelské parametry a značky (tagy). U textového výpisu do informačního panelu lze rovněž provádět různé výpočty, jako je například délka trvání události v rocích, měsících a dnech.

V textovém zobrazení mohou být také informace o časové ose jako je aktuální granularita, datum a čas, kde se nachází kurzor myši a další.

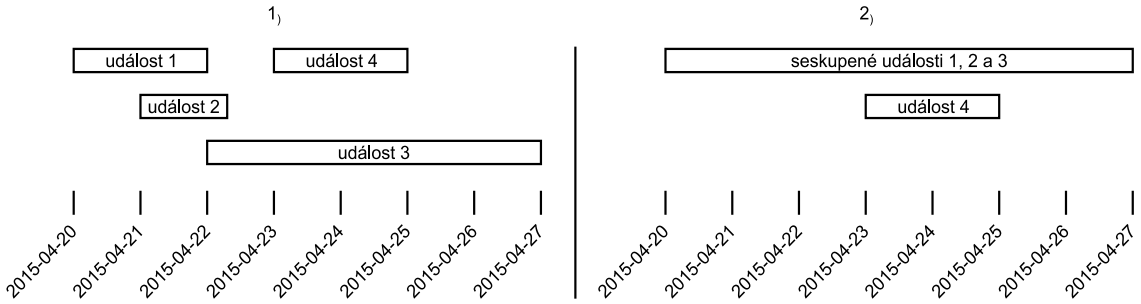
6.8 Skrývání událostí

Podobně jako při výběru událostí, které uživatele zajímají, jsou v grafu nepochybně i události a okamžiky, které uživatel při zkoumání historických událostí vidět nechce. Proto je dobré přidat možnost skrytí vrcholů, které nejsou v daném okamžiku nijak zajímavé. Skrývání může probíhat automaticky podle předem dané logiky (například vrcholy bez hran budou vždy skryté) nebo podle ohodnocení událostí, viz skrývání události při zoomu 6.3. Další možností je skrytí podle uživatelského výběru. Uživatel by mohl provést selekci vrcholů grafu, které bude chtít skrýt nebo naopak vykreslit.

Nastává zde ale otázka, zda počítat se skrytými vrcholy při výpočtu rozložení grafu, nebo tyto skryté události ignorovat. Pokud budou skryté vrcholy ignorovány, bude výpočet layoutu nedeterministický (bude záležet, jaké vrcholy jsou skryté v daný okamžik). Naopak, pokud se pokaždé při výpočtu layoutu bude počítat i se skrytými událostmi, bude tento výpočet deterministický a výsledné rozložení vždy stejné. Nejlepším řešením se tedy může zdát přidat konfiguraci výpočtu rozložení a nechat o daném problému rozhodnout uživatele na základě požadovaného výsledku.

6.9 Seskupování

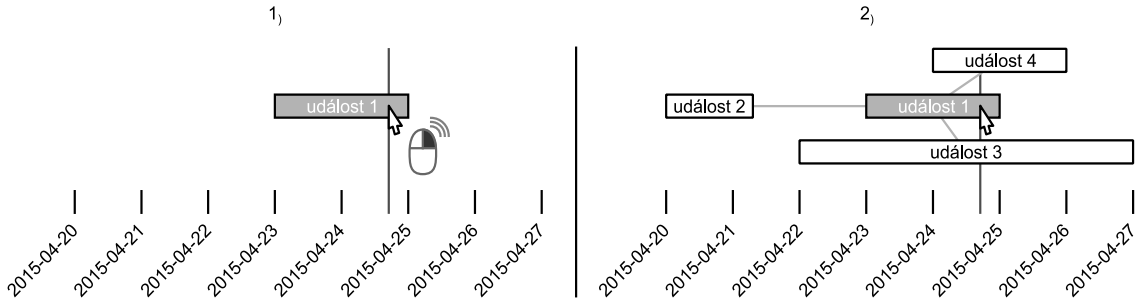
Pokud spolu události vzájemně souvisí a tvoří společně nějakou historickou událost, je možné je seskupit dohromady a vytvořit tak jednu událost komplexní. Takové chování se může hodit zejména, pokud graf obsahuje velké množství událostí, které by bylo pro uživatele nepřehledné. Poté je možné v závislosti na úrovni zoomu události automaticky seskupovat nebo naopak rozbalovat, nebo můžeme dovolit uživateli seskupení, respektive rozbalení na vyžádání. Ukázka, jak může vypadat seskupení událostí, viz obrázek 6-6.



Obrázek 6-6: Seskupení událostí 1, 2 a 3; 1) před seskupením, 2) výsledek

6.10 Náhled okolí

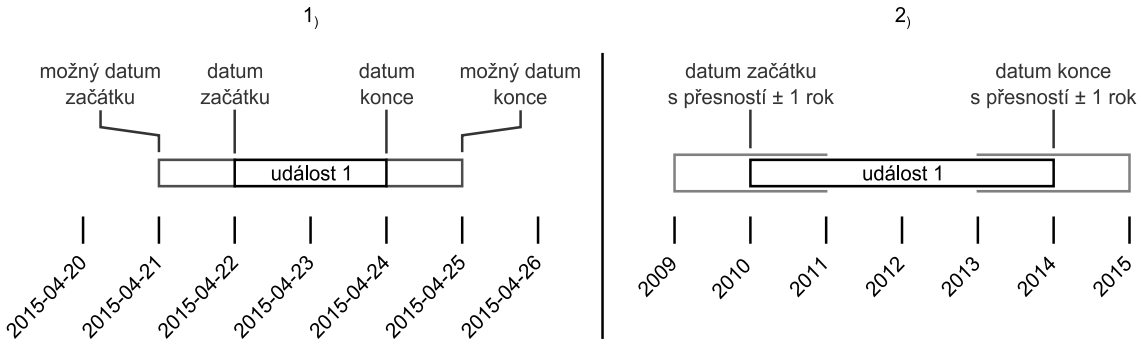
Mírně souvisí se zvýrazněním (kapitola 6.5). Jedná se o zobrazení událostí, které jsou přímo spojeny s událostí, nad kterou se nachází kurzor, ale jsou skryté, nebo jsou mimo viditelnou oblast. Náhled by se neměl zobrazit ihned po najetí kurzoru myši nad požadovaný vrchol, ale měl by se zobrazit se zpožděním několika vteřin (konfigurovatelný parametr), nebo na vyžádání uživatele například při stisku pravého tlačítka myši. Obrázek 6-7 zobrazuje, jak by mohl takový náhled okolí události vypadat v praxi.



Obrázek 6-7: Náhled okolí události 1 při pravém kliknutí na vrchol; 1) před kliknutím, 2) náhled po kliknutí

6.11 Zobrazení nepřesnosti začátku a konce události

Ne o všech historických událostech víme přesné údaje o tom, kdy začaly nebo skončily. Čím starší taková událost je, tím nepřesnější většinou bývá datum a čas, který tuto událost popisuje. Proto může být u události řečeno a následně ve vizualizaci zobrazeno, s jakou přesností událost začala nebo naopak skončila. Nepřesnost může být uvedena ve formě dalšího data nebo úrovně granularity. V prvním případě musí mít událost u každého data dva záznamy (možný začátek, začátek, konec a možný konec). Při nepřesnosti na úrovni granularity specifikujeme, s jakou přesností granularity událost začala nebo skončila. Obrázek 6-8 zobrazuje vizualizaci obou případů.



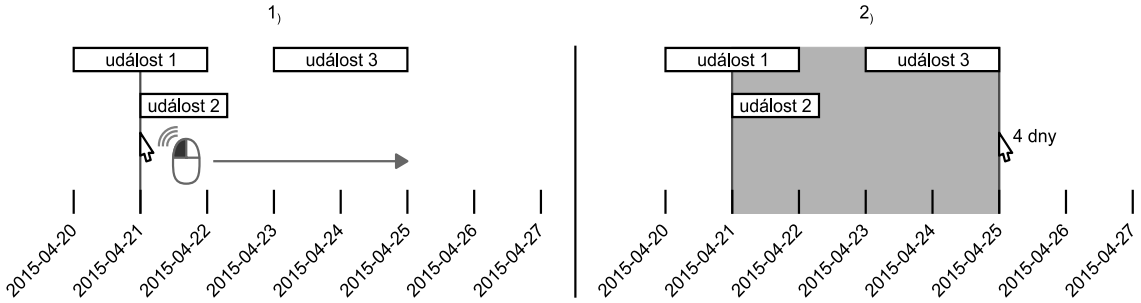
Obrázek 6-8: Zobrazení nepřesnosti začátku a konce události; 1) dvě data začátku události a 2 data konce události, 2) nepřesnost na úrovni granularity roků

6.12 Odměřování doby trvání

Při zobrazení grafu událostí je důležitým aspektem, jak dlouho události trvaly, kolik času uběhlo mezi událostmi nebo jak dlouho se události překrývaly. Samozřejmě lze všechny tyto informace dopočítat, pokud máme k dispozici data začátků a konců událostí. Nelze ale jednoduše rozhodnout, jakou informaci budeme chtít zobrazit, když vybereme dvě nebo více událostí v grafu. Bude to celková doba jejich trvání, čas, po který se události překrývaly, nebo naopak čas kdy k překryvu nedošlo?

Elegantním řešením je nechat rozhodnutí na uživateli a poskytnout mu pouze možnost vybrat dva body na časové ose a spočítat dobu trvání mezi nimi. Uživatel si poté může libovolně vybírat, mezi kterými časy bude chtít změřit dobu trvání a co má tato doba znamenat.

Obrázek 6-9 naznačuje, jak by mohlo měření délky času fungovat v praxi. Nejprve stisknutím levého tlačítka myši vybereme začátek, odkud budeme chtít měřit (levá část obrázku) a následně posunem myši se dostaneme na požadované místo konce, v našem ukázkovém případě nás zajímá doba od začátku události 2 do konce události 3. Již od stisknutí levého tlačítka myši a posunu se nad kurzorem začne vypisovat délka trvání označeného výběru (pravá část obrázku).



Obrázek 6-9: Změření doby trvání mezi dvěma časovými body; 1) stisknutí levého tlačítka výběr začátku a posun myši a 2) zobrazení délky trvání nad kurzorem myši

6.13 Lokální kopie dat, uložení stavu

Pokud bude v budoucnu jedna centrální databáze s historickými událostmi (nebo více) a budeme lokálně provádět nějaké změny (rozmístění událostí, skrývání vrcholů, seskupování událostí a tak dále), nebudeme chtít tyto změny propagovat do společného úložiště, ale budeme si je chtít uchovat lokálně. Program by mohl obsahovat metody pro export a následně import nastavení a dat, které si uživatel přizpůsobil podle své potřeby.

6.14 Vkládání, mazání a úprava dat

Poslední ze zde uvedených možností interakce je přidávání, mazání a úprava dat grafu. Pokud by měl sloužit výsledný program jako celkový vizualizační a editační nástroj pro správu historických dat, poté se bez těchto funkcí neobejde. Základní metody pro přidávání, mazání a změnu poskytuje jak datová vrstva [1] tak API nad ní postavené [2]. Program by musel následně obsahovat přidávací a editační formuláře, které by sloužily uživateli k zadávání nebo změně dat.

7 Vlastní implementace

Vizualizační program je vytvořen v programovacím jazyce Java. Pro řízení projektu a jeho sestavení je použitý nástroj Maven a dávkové soubory systému Microsoft Windows.

7.1 Prostředí a použité knihovny

Vývojové prostředí:

- Programovací jazyk Java 1.7
- Sestavovací nástroj Apache Maven a dávkové soubory

Knihovny:

- SwiXML 1.4 [26] definice GUI (Apache-styl licence)
- Joda-Time 2.7 [9] reprezentace dat a časů (Apache 2.0 licence)
- log4j 2 2.1 [27] logovací nástroj (Apache 2.0 licence)
- graph 1.0 [2] reprezentace grafu událostí
- graph-ranking 1.0 [2] ohodnocení grafu událostí
- db-graph-ranking 1.0 [2] ohodnocení grafu událostí z databáze
- TimelineMaven 0.0.1 [1] datová vrstva s grafem událostí

7.2 Struktura projektu

Balíky programu jsem rozdělil podle jednotlivých komponent do následující struktury s třídami a rozhraními (class diagram viz *Příloha A - Class diagram*):

- `cz.zcu.fav.kiv.timeline.visualization`
  - hlavní balík programu
  - třídy:
    - `Main` – hlavní třída programu, vstupní bod
    - `Utils` – pomocná třída s metodami a globálními konstantami
- `cz.zcu.fav.kiv.timeline.visualization.graph`
  - popis elementů grafu, napojení na zdroj dat
  - třídy:
    - `EdgeWrapper` – obalová třída hran grafu
    - `GraphManager` – graf událostí, zdroj dat
    - `VertexWrapper` – obalová třída vrcholů grafu
- `cz.zcu.fav.kiv.timeline.visualization.gui`
  - grafické uživatelské rozhraní programu
  - třídy:
    - `FileChooserFileFilter` – filtr souborů
    - `MainWindow` – hlavní okno programu
    - `OpenCSVDialog` – dialog pro načtení CSV souborů
    - `OpenDatabaseDialog` – dialog pro připojení k databázi
    - `StatusBar` – informační panel
- `cz.zcu.fav.kiv.timeline.visualization.gui.graph`
  - vykreslování grafu
  - třídy a rozhraní:
    - `GraphView` – grafická komponenta grafu událostí
    - `IGraphLayout` – rozhraní pro výpočet rozložení grafu
    - `TimeBasedLayout` – časově orientované rozložení grafu
- `cz.zcu.fav.kiv.timeline.visualization.gui.listeners`
  - posluchači událostí a delegát interakce s programem
  - třídy a rozhraní:
    - `DialogCloseListener` – posluchač zavření dialogového okna
    - `GraphViewListener` – posluchač interakce s grafem
    - `IInteractionListener` – rozhraní posluchače interakcí
    - `InteractionDelegate` – delegát interakcí
    - `PanningListener` – posluchač posunu časové osy a grafu
    - `WindowListener` – posluchač změn okna programu
    - `ZoomListener` – posluchač zoomu časové osy a grafu
- `cz.zcu.fav.kiv.timeline.visualization.gui.timeline`
  - zobrazení časové osy
  - třídy:
    - `Timeline` – grafická komponenta časové osy
    - `TimelineUtils` – pomocná třída s metodami a konstantami
- `resources`
  - složka s konfiguračními a dalšími soubory programu

7.3 GUI

Grafické uživatelské rozhraní je stěžejní částí pro vizualizaci. Z důvodu lepší udržitelnosti a přehlednosti jsem pro základní definování struktury a rozložení GUI zvolil knihovnu SwiXML [26].

7.3.1 SwiXML

SwiXML je malá open-source knihovna, sloužící ke generování GUI Java aplikací a appletů. Aktuálně se nachází ve verzi 2.4 a je stále aktivně vyvíjena. Grafické uživatelské rozhraní je definované v samostatných XML dokumentech, které jsou za běhu programu načteny a vykresleny podle definic jako `javax.swing` objekty. Knihovna nabízí načítání XML souborů jak lokálně, tak i ze vzdáleného serveru. Díky tomu mohou být výsledné aplikace nezávislé a jejich vzhled lze jednoduše globálně měnit.

Definování grafického uživatelského rozhraní v separátních souborech přináší řadu výhod. GUI nemusí být přímo zakomponované v kódu aplikace, což umožňuje bez většího zásahu libovolnou změnu vzhledu. Dále lze dynamicky za běhu programu načítat různé XML soubory a tím měnit vzhled aplikace například podle uživatelské role, kdy uživatel s vysokým oprávněním má mít větší nabídku funkcí programu než uživatel s oprávněním nízkým.

Knihovna se nesnaží programátora odstínit od základních `javax.swing` komponent, ale spíše poskytnout, zjednodušit, urychlit vývoj a zredukovat pokud možno co nejvíce kódu, který by musel programátor napsat.

Definování grafických uživatelských rozhraní v separátních souborech se využívá v mnoha programovacích jazycích. Jako příklad lze uvést XAML pro .NET Framework od Microsoftu nebo definování GUI v XML souborech pro operační systém Android. Pro programovací jazyk Java existují knihovny JavaFX, Thinlet, XUL, XULUX, SwingML.

Mezi hlavní výhody SwiXML patří:

- zaměřuje se čistě na `javax.swing`,
- programátoři se znalostmi SWING mohou ihned začít používat knihovnu pro popis grafického uživatelského rozhraní,
- knihovna je velice rychlá a malá,
- pouze generuje GUI a o dynamické chování a logiku se musí postarat programátor sám přímo v kódu.

7.3.1.1 XML definice

Jak již bylo napsáno, grafické uživatelské rozhraní je definované v XML souboru, viz ukázka kódu 7-1 (okno `frame`, které obsahuje popisek `label` a tlačítko `button`). Definovat lze všechny základní SWING komponenty, jako jsou okna, dialogy, tlačítka, vstupní pole a tak dále. Jejich kompletní soupis lze nalézt na stránkách nástroje.

```
<?xml version="1.0" encoding="UTF-8"?>
<frame size="300,100">
    <label constraints="BorderLayout.CENTER" text="Hello World"/>
    <button id="quit" constraints="BorderLayout.SOUTH" text="Quit"/>
</frame>
```

Ukázka kódu 7-1: SwiXML HelloWorld

V XML lze zároveň definovat vlastní komponenty (např. `timeline`), vlastní parametry komponent (`height`, `backgroundColor`, `foregroundColor`), název properties souboru s lokalizací `bundle` a definice jazyka `locale`, obslužné metody pro aktivní prvky `action` (například tlačítka), klávesové zkratky `accelerator` a `mnemonic` a další, viz ukázka kódu 7-2.

Lokalizace grafického uživatelského rozhraní pracuje na způsobu definice klíče v hodnotách parametrů (ukázka kódu 7-2, komponenta `menu` a klíč „`mus_File`“, komponenta `menuitem` a klíče „`mis_OpenCSV`“, „acc_OpenCSV“ a „mn_OpenCSV“). Pokud je klíč obsažen v lokalizaci definované parametry `bundle` a `locale`, bude použita jeho hodnota z properties souboru (ukázka kódu 7-3). Pokud by se v lokalizaci daný klíč nenacházel, bude použit jako text na komponentě právě tento klíč.

```
<?xml version="1.0" encoding="UTF-8"?>
<frame id="frame" bundle="visualization" locale="cs">
    <menubar name="menubar">
        <menu text="mus_File">
            <menuitem name="mi_openCSV" Text="mis_OpenCSV" action="openCSVAction"
                      accelerator="acc_OpenCSV" mnemonic="mn_OpenCSV" />
        </menu>
    </menubar>
    <timeline id="timeline" constraints="BorderLayout.SOUTH" height="50"
              backgroundColor="#FFFFFF" foregroundColor="#404040" />
</frame>
```

Ukázka kódu 7-2: SwiXML lokalizace, obslužné akce, vlastní komponenta a parametry

```
mus_File = Soubor
mis_OpenCSV = Otev\u0159ít CSV soubory
acc_OpenCSV = control O
mn_OpenCSV = VK_O
```

Ukázka kódu 7-3: Ukázka souboru „visualization_cs.properties“ s lokalizací

7.3.1.2 Vytvoření GUI

Obslužný kód, který podle XML definice vygeneruje grafické uživatelské rozhraní, zobrazuje ukázka kódu 7-4. SwiXML knihovna při vytváření GUI přes reflexi nastaví instance grafických komponent k objektům, pokud mají v XML definici nastavené `id`, které udává jméno objektu (ukázka kódu 7-4 objekt `JButton quit` a definice `id="quit"` u komponenty `button`, viz ukázka kódu 7-1.

```
public class HelloWorld {
	// tlacitko konec
	public JButton quit;

	public HelloWorld() throws Exception {
		// zobrazeni debugovacich zprav SwiXML
		SwingEngine.DEBUG_MODE = true;

		// nacteni XML definice a zobrazeni GUI
		new SwingEngine(this).render("hello_world.xml").setVisible(true);

		// osetreni kliknuti na tlaciko konec
		quit.addActionListener(new ActionListener() {
			public void actionPerformed(ActionEvent evt) {
				System.exit(0);
			}
		});
	}

	public static void main(String[] args) throws Exception {
		// vytvoreni instance
		new HelloWorld();
	}
}
```

Ukázka kódu 7-4: Obslužný kód k vytvoření GUI podle XML definice k příkladu HelloWorld, viz ukázka kódu 7-1

Pokud v XML definici používáme vlastní komponenty (ukázka kódu 7-2 komponenta `timeline`), je nutné před vygenerováním GUI přidat tagy těchto komponent a provázat je se třídami, které reprezentují, viz ukázka kódu 7-5.

```
// vytvoreni SwiXML renderovaciho enginu
swingEngine = new SwingEngine(this);

// definice vlastniho tagu a provazani s tridou
swingEngine.getTaglib().registerTag("timeline", Timeline.class);

// nacteni XML definice
swingEngine.render("definice_gui.xml");

// zobrazeni GUI
swingEngine.getRootComponent().setVisible(true);
```

Ukázka kódu 7-5: Definice vlastních tagů, viz ukázka kódu 7-2

7.3.1.3 Ošetření akcí

Knihovna se sama stará o ošetření uživatelských akcí s aktivními komponentami jako jsou tlačítka a menu. Pokud k nim chceme přidat další aplikační logiku při jejich interakci (stisknutí tlačítka, výběr v menu), je nutné v XML definici komponentě nastavit, jaká obslužná akce se má vykonat (parametr `action="nazev_akce"`, viz ukázka kódu 7-2). Knihovna poté přes reflexi zavolá při interakci s komponentou definovanou obslužnou akci (ukázka kódu 7-6).

```
// obsluzna akce "openCSVAction"
public Action openCSVAction = new AbstractAction() {
	@Override
	public void actionPerformed(ActionEvent e) {
		// obslouzeni po vyvolani akce
	}
};
```

Ukázka kódu 7-6: Definice obslužné akce pro GUI komponentu menuitem, viz ukázka kódu 7-2

7.3.1.4 Vlastní parametry

Jednotlivé grafické komponenty v XML definici mohou obsahovat parametry, díky kterým lze upravovat jejich vzhled, vykreslování a chování. Pokud má komponenta definovaný parametr s hodnotou, hledá knihovna při generování GUI v této komponentě (respektive v její třídě) přes reflexi `set` metodu podle názvu parametru (např. pro parametr `velikost` bude zavolána metoda `setVelikost`) a předá do ní hodnotu parametru. Hodnota parametru v XML definici musí odpovídat datovému typu parametru definovanému v set metodě, jinak nebude nastaven.

Ukázka kódu 7-2 definuje u komponenty `timeline` vlastní parametry `height`, `backgroundColor` a `foregroundColor`. Třída reprezentující tuto komponentu musí obsahovat settery, které bude knihovna volat, viz ukázka kódu 7-7.

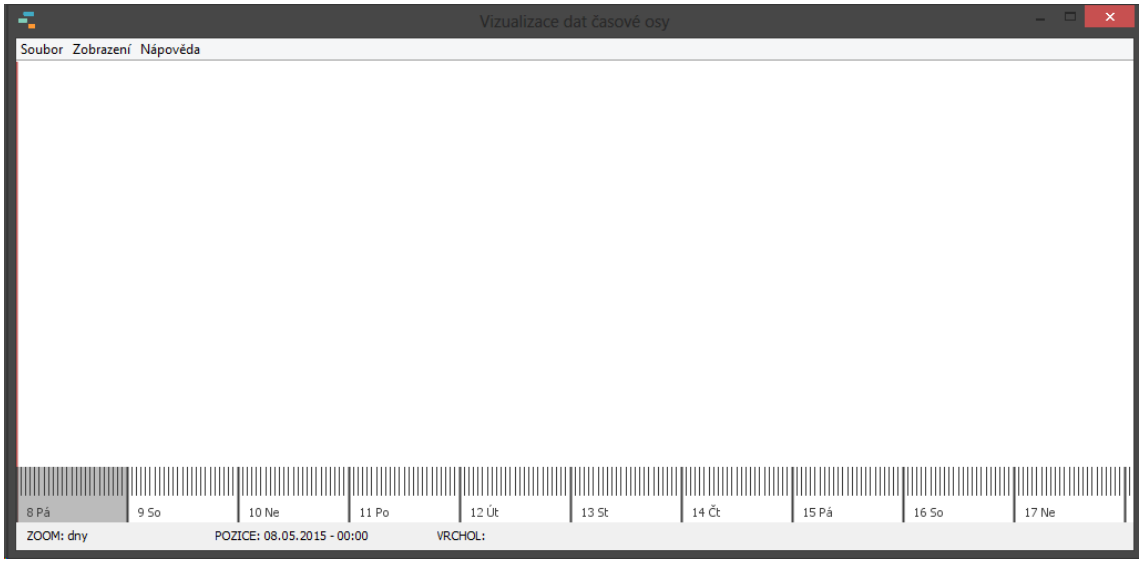
```
// trida komponenty timeline
public class Timeline {

	// set metoda pro parametr height
	public void setHeight(int height) {}

	// set metoda pro parametr backgroundColor
	public void setBackgroundColor(String backgroundColor) {}

	// set metoda pro parametr foregroundColor
	public void setForegroundColor(String foregroundColor) {}
}
```

Ukázka kódu 7-7: Set metody pro ošetření vlastních parametrů komponenty

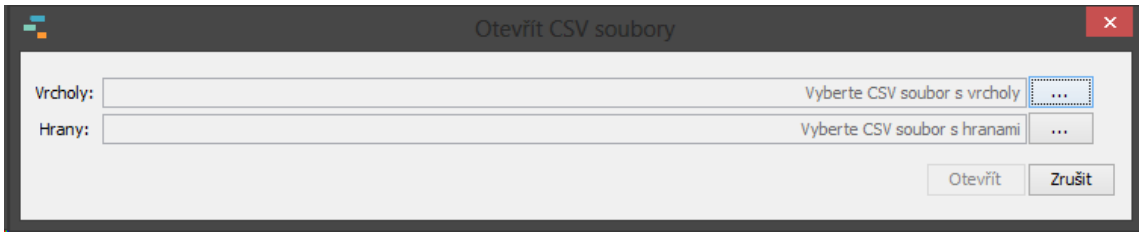
7.3.1.5 Seznam XML definic programu

V programu jsou použité celkem tři XML definice grafického uživatelského rozhraní a jeden properties soubor s lokalizací:

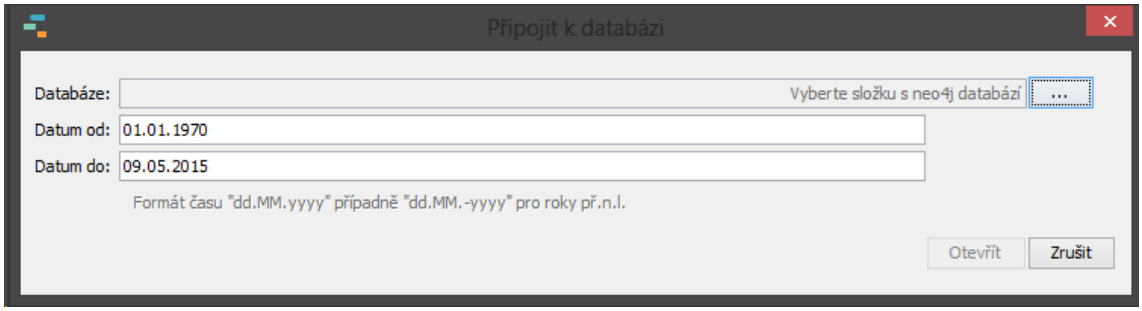
- `visualization_main_window.xml` pro třídu `MainWindow` (obrázek 7-1),
- `visualization_open_csv_dialog.xml` pro třídu `OpenCSVDialog` (obrázek 7-2),
- `visualization_open_database_dialog.xml` pro třídu `OpenDatabaseDialog` (obrázek 7-3) a
- `visualization_cs.properties` s lokalizací GUI.



Obrázek 7-1: Hlavní okno programu (visualization_main_window.xml)



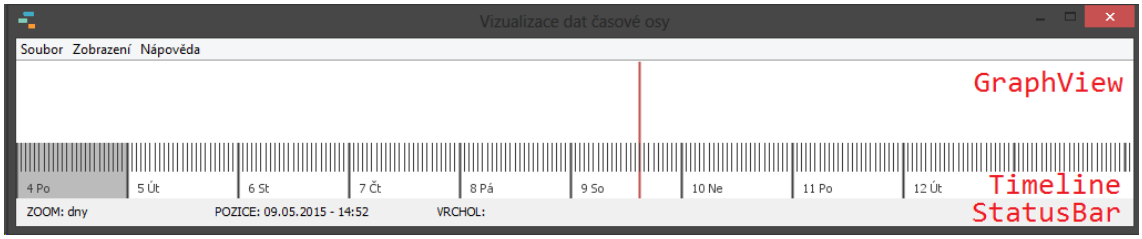
Obrázek 7-2: Dialog pro otevření CSV souborů (visualization_open_csv_dialog.xml)



Obrázek 7-3: Dialog pro připojení k databázi (visualization_open_database.xml)

7.3.2 Hlavní okno aplikace

Hlavní okno aplikace `MainWindow` je definované SwiXML souborem `visualization_main_window.xml`. Kromě menu se skládá ze tří základních komponent: reprezentace grafu `GraphView`, časové osy `Timeline` a informačního panelu `StatusBar`. Zobrazení hlavního okna s popisem jednotlivých komponent ukazuje obrázek 7-4.



Obrázek 7-4: Komponenty hlavního okna

V konstruktoru třídy se spouští načtení definice z XML souboru, spuštění vykreslovacího vlákna grafu (7.3.4), inicializace delegáta (7.6) a přidání posluchačů jak k delegátovi, tak i grafickým komponentám (7.5). Dále třída obsahuje metody pro získání lokalizovaného řetězce `getLocalizedString` podle zadaného klíče z properties souboru, který je definovaný v XML definici, metodu spuštění inicializace aplikace `initializeApplication` a obslužné akce prvků menu.

7.3.3 Časová osa

Třídy `Timeline` a pomocná třída `TimelineUtils` tvoří dohromady vlastní komponentu časové osy. Komponenta je založena na reprezentaci času knihovnou Joda-Time přesněji řečeno její třídou `DateTime`. `DateTime` používá spodní granularitu času milisekundy reprezentované celým číslem `long`. Referenční datum, od kterého se čas odvíjí, je půlnoc 1. ledna 1970 (vyjádřeno přesněji: `00:00:00 1. 1. 1970`). Díky tomu lze zobrazit data přibližně od roku `-2 147 483 648` až do `2 147 483 647`. Pomocná třída `TimelineUtils` obsahuje výčet podporovaných granularit a globální konstanty. Třída `Timeline` se stará pouze o vykreslení komponenty.

7.3.3.1 TimelineUtils

Základní části třídy jsou definice jedné úrovně granularity `ITimeSegment` (ukázka kódu 7-8), výčet všech podporovaných úrovní granularit `ZOOM_LEVEL` ve formě `enum` datového typu, který implementuje `ITimeSegment` (ukázka kódu 7-9 a ukázka kódu 7-10) a globální konstanty a metody. Prozatím jsou v práci definované úrovně: hodiny, dny, měsíce, roky, desetiletí, staletí, tisíciletí a desítky tisíc let.

```
// predpis urovne granularity
public interface ITimeSegment {
	// nalezeni nejblizsiho mensiho zacatku casoveho useku dane granularity
	// od zadaneho data.
	DateTime findNearestBeginningOfTimeSegment(long fromTime);

	// nalezeni nejblizsiho vetsiho zacatku casoveho useku dane granularity
	// od zadaneho data.
	DateTime nextTimeSegmentDate(DateTime dateTime);

	// textovy popis zadaneho casu pro vypis do casove osy
	String getTimeSegmentIdentifier(DateTime dateTime);
}
```

Ukázka kódu 7-8: Předpis úrovně granularity

Každá úroveň granularity musí obsahovat jedinečný celočíselný identifikátor, minimální šířku časového segmentu a počet dní, který je obsažený v dané úrovni. Tyto informace musí být definovány v konstruktoru při vytvoření. Výčet úrovní dále obsahuje definice jednotlivých úrovní a metody pro získání informací o jednotlivých úrovních.

```
// uroven granularity DEN
DAY(zoomLevelCount++, 40d, 1d) {
	@Override
	public DateTime findNearestBeginningOfTimeSegment(long fromTime) {
		DateTime dateTime = new DateTime(fromTime);
		return dateTime.withTimeAtStartOfDay();
	}
	@Override
	public DateTime nextTimeSegmentDate(DateTime dateTime) {
		return dateTime.plusDays(1).withTimeAtStartOfDay();
	}
	@Override
	public String getTimeSegmentIdentifier(DateTime dateTime) {
		int day = dateTime.getDayOfMonth();

		return (day != 1) ? (day + " " +
		MainWindow.getLocalizedString(DAYS_SHORT_NAMES[dateTime.getDayOfWeek
		() - 1])) : (ZOOM_LEVEL.MONTH.getTimeSegmentIdentifier(dateTime));
	}
}
```

Ukázka kódu 7-9: Definice jedné úrovně granularity „den“, definice úrovní jsou napsané na začátku výčtu ZOOM_LEVEL, viz ukázka kódu 7-10

```
// vycet granularit
public static enum ZOOM_LEVEL implements ITimeSegment {
	//
	// VYCET GRANULARIT
	//

	// ciselny identifikator granularity
	private int id;
	// minimalni sirka casoveho useku granularity
	private double minimalTimeSegmentWidth;
	// pocet dni obsazenych v jednom casovem useku dane granularity
	private double timeSegmentDayCount;

	// privatni konstruktor urvone granularity
	private ZOOM_LEVEL(int id, double minimalTimeSegmentWidth, double
	timeSegmentDayCount) {
		this.id = id;
		this.minimalTimeSegmentWidth = minimalTimeSegmentWidth;
		this.timeSegmentDayCount = timeSegmentDayCount;
	}

	// ziskani identifikatoru urovne
	public int getId() {
		return id;
	}

	// ziskani minimalni sirky
	public double getMinimalTimeSegmentWidth() {
		return minimalTimeSegmentWidth;
	}

	// ziskani poctu dni v urovni granularity
	public double getTimeSegmentDayCount() {
		return timeSegmentDayCount;
	}

	// ziskani urovne podle ciselneho identifikatoru
	public static ZOOM_LEVEL getById(int id) {
		for (ZOOM_LEVEL zoomLevel : ZOOM_LEVEL.values()) {
			if (zoomLevel.getId() == id) {
				return zoomLevel;
			}
		}
		return ZOOM_LEVEL.DAY;
	}
}
```

Ukázka kódu 7-10: Výčet úrovní granularit

7.3.3.2 Timeline

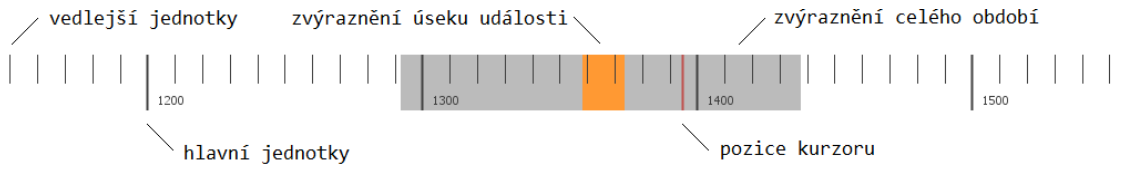
Grafická reprezentace časové osy je založena na úrovních granularity definovaných v `TimelineUtils`. Třída obsahuje metody pro ošetření interakce s časovou osou: posun `panning`, přiblížení `zoomIn`, oddálení `zoomOut`, výpočet aktuální úrovně granularity `setZoomLevel` a dále metody pro konverzi času na horizontální souřadnici v časové ose `transformDateToXCoordinate` a také naopak souřadnici na časové ose na čas, který se na dané souřadnici nachází `transformXCoordinateToDate`.

Hlavní úlohou třídy je vykreslení časové osy, které probíhá ve více fázích. Vstupním bodem do vykreslování je přepsaná metoda `paintComponent` komponenty `JPanel`, od které dědí časová osa. Do panelu je vždy vykreslována jen viditelná část časové osy. V každé fázi je vykreslována jiná část celkové časové osy, výsledné zobrazení ukazuje obrázek 7-5. Pořadí fází vykreslení je následující:

1. vykreslení zvýrazněného časového úseku (minimální až maximální datum všech událostí) `drawPeriodHighlight`,
2. pokud je komponenta inicializovaná a je načtený graf událostí:
    a. zvýraznění časových úseků vybraných událostí `drawSelectedVertices`,
    b. zvýraznění časového úseku události `drawHoverVertex`, nad kterou se nachází kurzor myši,
3. vykreslení identifikátoru pozice myši `drawMousePosition`,
4. vykreslení vedlejších jednotek časové osy bez popisků `drawTimeline` a
5. vykreslení hlavních jednotek časové osy s popisky `drawTimeline`.

Zaměřil bych se zde na dvě poslední fáze, tedy vykreslení hlavních a vedlejších jednotek časové osy. Obě fáze využívají stejnou metodu `drawTimeline`, pouze metodu volají s rozdílnými hodnotami parametrů. Parametry metody jsou třída `Graphics2D g2D`, přes kterou se bude časová osa vykreslovat do panelu, úroveň granularity, respektive aktuální úroveň zoomu `zoomLevel`, příznak zda se mají vypisovat popisky intervalů `withIdentifires` a výška `height`. Ve fázi 4 se volá metoda s úrovní zoomu o jedna menší než je aktuální (pokud je aktuální úroveň den, budou vedlejší jednotky hodiny), příznak `withIdentifires` je nastaven na `false` (nechceme vypisovat textové popisky vedlejších jednotek) a s poloviční výškou `height` než ve fázi `5`. Naopak ve fázi `5` se volá metoda s aktuální úrovní zoomu, výpisem popisků a výškou definovanou v konfiguračním souboru. Postup kroků, které se provádí při vykreslení úseků časové osy, je následující:

1. zjištění souřadnice, od které se bude časová osa vykreslovat,
2. vykreslování úseků do maximální šířky panelu:
    a. načtení textového popisu časového úseku,
    b. nastavení stylu vykreslení,
    c. vykreslení časového úseku,
    d. vykreslení popisku, pokud má být vidět a
    e. nalezení dalšího časového úseku.



Obrázek 7-5: Časová osa a její části

7.3.4 Graf

Po časové ose je zobrazení grafu nejdůležitější částí vizualizace. Jak již bylo napsáno v kapitolách 4.3.2 a 5, nejprve jsem pro reprezentaci grafu používal knihovnu JGraphX, která se nakonec ukázala jako nevyhovující (viz 5.4) a proto jsem se rozhodl naprogramovat vizualizaci grafu vlastní, která částečně vychází a inspiruje se právě knihovnou JGraphX.

Hlavní třída komponenty je `GraphView`, rozložení grafu je popsáno rozhraním `IGraphLayout` a implementací `TimeBasedLayou`, vrcholy grafu jsou v obalové třídě `VertexWrapper` a hrany v obalové třídě `EdgeWrapper`.

7.3.4.1 Vykreslovací smyčka

Vykreslování grafu probíhá aktivním renderováním. Jedná se o techniku, kdy vykreslování probíhá stále dokola v řízené smyčce v separátním vlákně. Vždy se vykresluje do neaktivního bufferu a následně po vykreslení všech objektů do bufferu se neaktivní buffer prohodí s aktivním a tak stále dokola. Výhodou aktivního renderování je plynulost animací a možnost řízení snímkové frekvence FPS (frames per second). V programu je snímkovací frekvence omezena na hodnotu 60. Vlákno vykreslování je spuštěno ihned po startu aplikace a zastaveno je až při jejím ukončení.

Akce prováděné ve vykreslovací smyčce (třída `GraphView` metoda `run`) jsou:

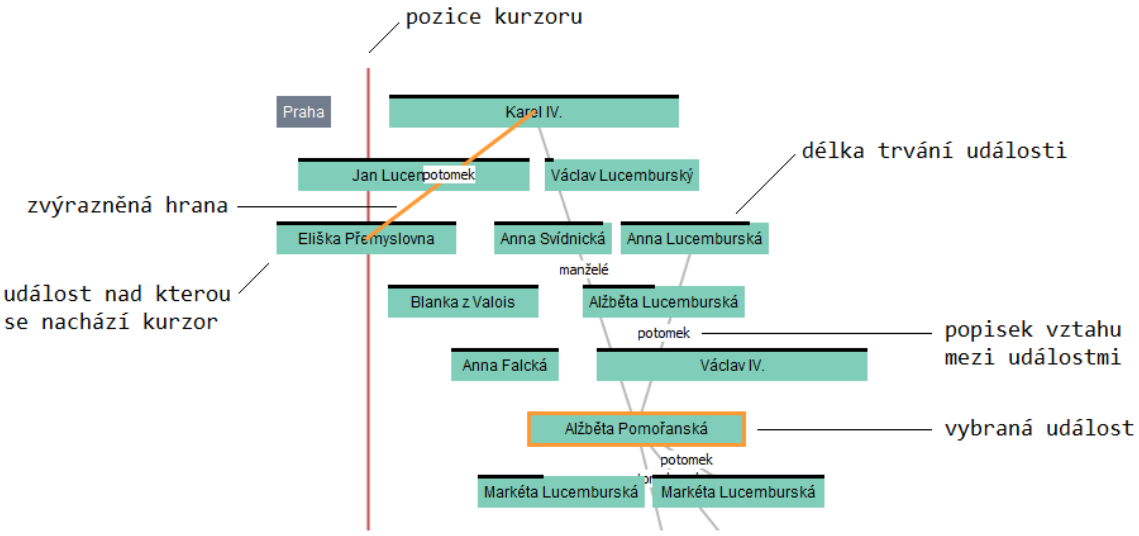
1. inicializace proměnných,
2. inicializace bufferů,
3. vykreslovací smyčka:
    a. výpočet FPS,
    b. vytvoření bufferu pro vykreslení,
    c. vykreslení grafu (viz 7.3.4.2),
    d. výpis FPS,
    e. prohození bufferů,
    f. uspání vlákna a
    g. ošetření chyb.

7.3.4.2 Vykreslení grafu

Vykreslení grafu a jeho elementů (vrcholů a hran) probíhá v metodě `drawGraph` ve třídě `GraphView`. Stejně tak jako v případě časové osy je i graf vykreslován v několika fázích:

1. vykreslení pozice kurzoru,
2. pokud je komponenta inicializovaná a je načtený graf událostí:
    a. vykreslení všech hran (podmíněné),
    b. vykreslení hran vybraných vrcholů (podmíněné),
    c. vykreslení zvýrazněných hran (podmíněné),
    d. vykreslení vrcholů a
    e. vykreslení zvýrazněných hran (podmíněné).

Některé fáze vykreslování jsou podmíněné uživatelskou interakcí s GUI nebo logikou vykreslování. Například vykreslení hran v jakékoli fázi je podmíněné akcí zoom. Pokud uživatel provádí zoom, jsou veškeré hrany skryté a začnou se vykreslovat až po ukončení této akce. Další podmínkou je například to, zda se mají zvýrazněné hrany vykreslovat nad všemi ostatními objekty nebo nikoli. Zvýrazněné hrany se tedy vykreslují buď ve fázi *c* anebo *e*. Výslednou reprezentaci grafu zobrazuje obrázek 7-6.



Obrázek 7-6: Graf událostí a jeho části

7.3.4.3 Vrcholy grafu

Vrchol grafu VertexWrapper je obalová třída pro reprezentaci vrcholu, respektive události v API [2]. Vrchol obsahuje popis vlastní události, vstupní a výstupní hrany. Událost je popsaná datem začátku a konce a dále uchovává informace o události (název, popis, typ a další). Druhy událostí `NodeSterotype` jsou: událost `EVENT`, osoba `PERSON`, místo `PLACE` a předmět `ITEM`.

Obalová třída obsahuje definici začátku a konce události, identifikátor, vstupní a výstupní hrany. Kromě těchto informací definuje také proměnné nutné pro vykreslení vrcholu: `x` a `y` souřadnice vrcholu, šířku vrcholu `widht`, renderer `vertexRenderer` a příznaky, zda je vrchol vybraný `selected` a viditelný `visible`.

V konstruktoru `VertexWrapper` jsou z vrcholu uloženy základní informace do lokálních proměnných včetně vrcholu samotného. Dále je inicializován renderer vrcholu a nastavena jeho barva pozadí podle typu vrcholu `NodeSterotype`, který reprezentuje, a barva popředí která se počítá podle barvy pozadí tak, aby byly barvy kontrastní (metoda `getForegroundColor` z pomocné třídy `Utils`). Důležitou metodou třídy je metoda `refresh` s parametrem `fullRefresh`. Metoda provádí změnu velikosti vykreslovaného vrcholu podle obsahu, který je vykreslován anebo podle doby, jak dlouho událost trvala. Pokud je parametr metody `true`, vygeneruje se i text, který bude ve vrcholu vykreslován (metoda `buildBasicNodeRenderedString` z třídy `Utils`).

Samotné vykreslení vrcholu probíhá následovně:

1. test, zda je vrchol ve vykreslované (viditelné) oblasti,
2. ořez vrcholu pouze na viditelnou část, pokud je vykreslován podle délky trvání,
3. posun plátna na pozici vrcholu,
4. vykreslení vrcholu (`vertexRenderer`),
5. zobrazení délky trvání na horní hraně vrcholu, pokud je vykreslován podle délky trvání,
6. zvýraznění vrcholu pokud je vybraný a
7. posun plátna zpět.

Díky tomu, že je ve druhém kroku vrchol oříznut pouze na viditelnou část, je zaručeno, že bude vždy vidět text události. Pokud by k oříznutí nedocházelo, mohl by text zmizet mimo vykreslovanou oblast, pokud by byla délka události delší než šířka vykreslované oblasti.

7.3.4.4 Hrany grafu

Hrana grafu `EdgeWrapper` funguje na podobném principu jako obalová třída vrcholu. Obsahuje instanci počátečního a koncového vrcholu společně s rendererem. Vykreslení hrany probíhá v následujících krocích:

1. zjištění středu počátečního vrcholu,
2. zjištění středu koncového vrcholu,
3. vytvoření přímky mezi počátečním a koncovým vrcholem,
4. ořez přímky pouze na viditelnou část,
5. vykreslení hrany,
6. vykreslování popisků hran:
    a. posun plátna na střed přímky,
    b. vykreslení popisku hrany a
    c. posun plátna zpět.

Stejně tak jako vrchol je i hrana oříznuta pouze na viditelnou část. Díky tomu je zaručen dobrý výkon při vykreslování a také to, že bude vždy vidět popisek hrany.

7.3.4.5 Rozložení grafu

Rozložení grafu je nedílnou součástí při jeho zobrazování. Rozhraní `IGraphLayout` předepisuje, jaké metody musí výpočet layoutu obsahovat. Základní metody jsou:

- `execute` – spuštění výpočtu rozložení grafu,
- `isExecutionComplete` – identifikace, zda výpočet doběhl,
- `stopExecution` – zastavení výpočtu,
- `getMinY` – získání minimální `Y` souřadnice v grafu a
- `getMaxY` – získání maximální `Y` souřadnice v grafu.

Třída `TimeBasedLayout`, reprezentuje časové rozložení grafu (viz 6.2) a implementuje rozhraní `IGraphLayout`. Výpočet pozic vrcholů grafu probíhá následovně:

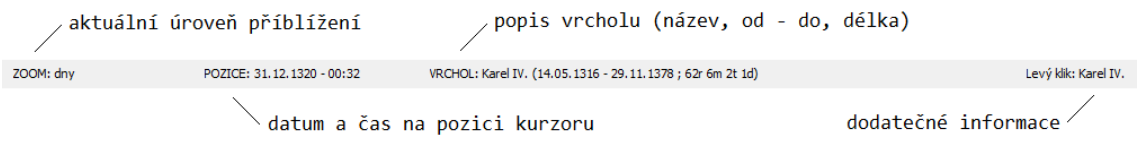
1. inicializace proměnných,
2. načtení všech vrcholů,
3. pro každý vrchol `i`:
    a. výpočet `X` souřadnice podle data začátku události, `Y` souřadnice `0`,
    b. kontrola, zda má být výpočet zastavený (pokud ano skok na bod 3),
    c. výpočet šířky vrcholu, pokud se mají zobrazit jako délka události,
    d. pro každý již umístěný vrchol `j < i`:
        i. posun vrcholu `i` dolů, pokud se překrývá s vrcholem `j`,
    e. zjištění maximální souřadnice `Y`.

Vytvoření a spuštění layoutu je kompletně řešené ve třídě `GraphView`. Inicializace layoutu je prováděna při inicializaci grafového zobrazení v metodě `initialize`. Spuštění přepočtu rozložení grafu je v metodě `executeLayout` a provádí se, pokud došlo k inicializaci grafového zobrazení, resetování pohledu nebo zoom akci (přiblížení a oddálení).

Jelikož je výpočet rozložení grafu časově náročný pro velký počet vrcholů grafu, je výpočet vždy spuštěn v separátním vlákně `layoutThread`. Pokud je zavolaná metoda `executeLayout`, ale rozložení se stále ještě počítá z předešlého volání, program zastaví vlákno výpočtu metodou `stopLayoutExecution` (výpočet doběhne, ale dopočítají se pouze `X` souřadnice vrcholů, viz zastavovací podmínka b) a spustí ho znovu od začátku.

7.3.5 Informační panel

Informační panel slouží k zobrazení základních informací o vizualizaci a interakci s časovou osou a grafem událostí. Zobrazuje informace o aktuální úrovni přiblížení, datu a času na pozici kurzoru, zvýrazněné události a dodatečné informace, viz obrázek 7-7.



Obrázek 7-7: Informační panel a jeho části

První tři informace jsou vypisované automaticky podle interakce s grafem a časovou osou. Dodatečné informace slouží například k popisu, co program aktuálně provádí za operace: otevření souboru, připojení k databázi, resetování pohledu a tak dále. Dodatečné informace mohou být nastaveny odkudkoli z programu aniž by měl volající objekt instanci informačního panelu, jelikož je metoda `setAdditionalText` statická. Text se zobrazuje po dobu 5 vteřin a poté automaticky zmizí (čas definovaný konstantou `SHOW_MESSAGE_DURATION`).

7.4 GraphManager

Třída `GraphManager` slouží jako prostředník mezi aplikací a datovou vrstvou, respektive jejím API. Obsahuje metody pro načtení dat z CSV souborů `loadDataFromCSV` a databáze `loadDataFromDatabase`. Dále uchovává celý načtený graf `graph` a všechny jeho vrcholy `vertices` a hrany `edges` včetně vrcholů vybraných `selectedVertices` a vrcholu zvýrazněného `hoverVertex`.

Při výběru nebo odznačení vrcholu grafu se upravuje seznam vybraných vrcholů `selectedVertices` tak, aby byl vždy aktuální. Stejně tak pokud kurzor myši přejede nad vrcholem, upravuje se proměnná `hoverVertex`.

7.4.1 Zdroj dat grafu událostí

Aplikace může zobrazit data ze dvou zdrojů a to CSV souborů nebo grafové databáze neo4j [28]. Po získání grafu z CSV souborů nebo databáze je spuštěna metoda `loadDataFromGraph`, která vytvoří pro každý vrchol a hranu obalovou třídu `VertexWrapper` nebo `EdgeWrapper` a najde minimální a maximální datum události v grafu.

7.4.1.1 CSV soubory

Při načítání dat z CSV souborů je nutné specifikovat dva tyto soubory (viz obrázek 7-2). Prvním z nich je CSV soubor s definicemi vrcholů grafu a druhým CSV soubor s definicemi závislostí mezi jednotlivými vrcholy. O načtení grafu ze souborů se stará metoda `loadFromCSVfiles` z třídy `GraphCSVLoader`, která je součástí API (projekt `graph-ranking`).

Oba CSV soubory mají přesně definovaný formát dat, který musí být dodržen, jinak dojde při čtení souborů k chybě. Ukázka kódu 7-11 a ukázka kódu 7-12 popisují na jednoduchém příkladu Karla IV., jak by mohly CSV soubory vypadat. Popis formátů obou souborů je na následující straně.

```
1|PERSON|Karel IV.|král, císař, markrabě a hrabě|1316-05-14|1378-11-29
2|PERSON|Jan Lucemburský|král a hrabě|1296-08-10|1346-09-26
3|PERSON|Eliška Přemyslovna|královna|1292-01-20|1330-09-28
```

Ukázka kódu 7-11: CSV soubor s definicí vrcholů

```
1001|RELATIONSHIP|potomek|2|1
1002|RELATIONSHIP|potomek|3|1
1003|RELATIONSHIP|manželé|2|3
```

Ukázka kódu 7-12: CSV soubor s definicí vztahů/hran

Formát CSV souboru s vrcholy/událostmi:

- `id|stereotype|nazev|popis|datum_zacatku|datum_konce`

Popis jednotlivých parametrů (tučné parametry povinné):

- **`id`** – jedinečný celočíselný identifikátor vrcholu,
- **`stereotype`** – typ vrcholu, možnosti (case-insensitive): `event`, `person`, `place` a `item`,
- **`nazev`** – název události,
- `popis` – popis události,
- `datum_zacatku` – datum začátku události ve formátu `yyyy-MM-dd` (pro roky před naším letopočtem `-yyyy-MM-dd`) a
- `datum_konce` – datum začátku události ve formátu `yyyy-MM-dd` (pro roky před naším letopočtem `-yyyy-MM-dd`).

Formát CSV souboru s hranami/závislostmi:

- `id|stereotype|nazev|pocatecni_vrchol|koncovy_vrchol`

Popis jednotlivých parametrů (tučné parametry povinné):

- **`id`** – jedinečný celočíselný identifikátor hrany,
- **`stereotype`** – typ hrany, možnosti (case-insensitive): `relationship`, `interaction`, `participation`, `creation`, `cause`, `part_of` a `takes_place`,
- `nazev` – název závislosti,
- **`pocatecni_vrchol`** – číselný identifikátor počátečního vrcholu a
- **`koncovy_vrchol`** – číselný identifikátor koncového vrcholu.

7.4.1.2 Databáze

Jako datové úložiště grafu událostí může být použita grafová databáze neo4j. Nad datovou vrstvou [1] je vytvořeno API [2], které poskytuje metody pro připojení k databázi, vypočtení ohodnocení vrcholů pagerank algoritmem a další. Pro otevření databáze je nutné specifikovat cestu k adresáři, ve kterém je databáze uložena. Dále se naváže spojení s databází a provede se výběr dat na základě minimálního a maximálního data, které se zadávají při připojení, viz obrázek 7-3.

Aby bylo možné provést ohodnocení přes pagerank algoritmus, je nutné mít specifikované priority jednotlivých typů vrcholů a hran, které se ve výpočtu použijí. K tomuto účelu v aplikaci slouží soubor `priorities.properties`, ve kterém jsou tyto priority definované. Algoritmus pagerank nad událostmi by mohl být použitý i při načítání dat z CSV souborů, ale vizualizace zatím s ohodnocením nijak nepracuje.

7.5 Posluchači

V aplikaci je implementováno více posluchačů různých událostí:

- `DialogCloseListener` – posluchač zavření dialogového okna,
- `GraphViewListener` – posluchač interakce s grafem,
- `IInteractionListener` – rozhraní posluchače interakcí,
- `PanningListener` – posluchač posunu časové osy a grafu,
- `WindowListener` – posluchač změn okna programu a
- `ZoomListener` – posluchač zoomu časové osy a grafu.

7.5.1 DialogCloseListener

Ošetření zavření dialogového okna pro otevření CSV souborů nebo připojení k databázi. Dialogová okna jsou tvořena komponentou `JDialog`. Knihovna SwiXML automaticky přidává akci skrytí dialogového okna při stisku klávesy `ESCAPE`. Posluchač při zavření okna dialogu `windowClosed` nebo při jeho skrytí `componentHidden` volá abstraktní metodu `dialogIsClosing`, která je ve třídách dialogů implementovaná a nastavuje `boolean` příznak `opened` na hodnotu `false`.

V konstruktoru dialogu se pak provádí test, zda již není dialogové okno otevřeno `opened == true`. Pokud již je dialogové okno otevřeno, neprovede se žádná akce. Pokud ještě není, vytvoří se a zobrazí uživateli. Díky tomu je zaručeno, že nebude zobrazeno více stejných dialogů najednou.

7.5.2 GraphViewListener

Jedná se o posluchače posunu a interakce myši nad komponentou grafu `GraphView`. Při pohybu myši posluchač kontroluje, zda se kurzor nenachází nad některým z vrcholů grafu a pokud ano, informuje o tom ostatní komponenty prostřednictvím delegáta 7.6 a jeho metody `vertexHover`. Dále posluchač reaguje na akce kliknutí levým nebo pravým tlačítkem myši na vrchol grafu a také o tom informuje ostatní komponenty prostřednictvím delegáta a metody `vertexClicked`.

7.5.3 IInteractionListener

Rozhraní posluchače interakcí s časovou osou a grafem událostí. Definuje základní metody interakce, na které mohou implementující třídy reagovat. Seznam definovaných metod:

- `getName` – vrací jméno posluchače,
- `initialize` – inicializace,
- `zoomIn` – přiblížení,
- `zoomOut` – oddálení,

- `zoomEnd` – ukončení zoom akce,
- `panning` – posun v časové ose nebo grafu událostí,
- `mousePosition` – změna pozice kurzoru myši,
- `windowSizeChanged` – změna velikosti okna aplikace,
- `refresh` – pokyn k překreslení komponent,
- `reset` – návrat do defaultního zobrazení,
- `vertexHover` – přejetí myši nad vrcholem grafu,
- `vertexClicked` – kliknutí levým nebo pravým tlačítkem myši na vrchol grafu,
- `selectAll` – výběr všech vrcholů grafu a
- `clearSelection` – zrušení výběru.

Aby byli posluchači informovaní o interakcích, musí být přidaní do delegáta, který interakce zprostředkovává. V delegátu k přidání posluchače slouží metoda `addInterationDelegate` (více v kapitole 7.6).

7.5.4 PanningListener

Posluchač posunu časové osy a grafu událostí táhnutím myši. Stejný posluchač je nastaven jak komponentě `Timeline` tak i komponentě `GraphView`. Díky tomu lze pohybovat s časovou osou a grafem událostí nezávisle na tom, nad kterou komponentou se kurzor myši nachází.

7.5.5 WindowListener

Stará se o ošetření změny velikosti nebo zavření okna programu `MainWindow` (respektive jeho SwiXML kořenové komponenty). Při změně velikosti je tato skutečnost přes delegáta rozšířena mezi ostatní komponenty metodou `windowSizeChanged`. Pokud je okno zavřeno, zavolá se metoda `closeProgram` z hlavní třídy programu `Main`, která se stará o správné ukončení celé aplikace a hlavně odpojení od databáze, pokud bylo spojení navázáno.

7.5.6 ZoomListener

Podobně jako PanningListener je i ZoomListener použitý jak pro časovou osu tak i graf událostí. Jeho účelem je zpracovat interakci při zoom akci (přiblížení, oddálení a ukonční zoom akce) a informovat o tom ostatní komponenty přes delegáta metodami `zoomIn`, `zoomOut` a `zoomEnd`. Zoom akce probíhá při stisku klávesy `CTRL` nebo `SHIFT` a otočení kolečka myši. Zoom s klávesou `CTRL` probíhá s menším krokem než v případě `SHIFT`. Hodnoty kroků jsou definované ve třídě `TimelineUtils` v konstantách `ZOOM_STEP_SMALL` a `ZOOM_STEP_LARGE`. Ukončení zoom akce je komponentám oznámeno, pokud je klávesa `CTRL` nebo `SHIFT` puštěna.

7.6 Delegát

Delegát je hlavní třídou, která se stará o informování komponent o změnách a interakci s časovou osou a grafem událostí. Komponenty, které chtějí být delegátem informované, musí implementovat rozhraní `IInteractionListener` a dále musí být přidány do delegáta metodou `addInteractionListener`. Výjimkou jsou pouze komponenty `GraphManager` a `Timeline`, které musí být předány již do konstruktoru, když se delegát vytváří a jsou automaticky přidány jako posluchači. Důvod je ten, že třída `GraphManager` poskytuje data pro graf a z časové osy `Timeline` se získávají souřadnice událostí.

Prozatím jsou v aplikaci čtyři komponenty, které implementují rozhraní `IInteractionListener` a jsou přidané jako posluchači do delegáta. Jedná se o třídy: `GraphManager`, `Timeline`, `GraphView` a `StatusBar`. Delegát je inicializován při vytváření hlavního okna aplikace `MainWindow` v jejím konstruktoru a zároveň zde jsou přidáni i posluchači. Toto ale není podmínkou a další posluchači mohou být libovolně vytvářeni a přidáváni do delegáta.

7.7 Resources

Součástí aplikace jsou i konfigurační a další soubory, které jsou uloženy ve složce `resources`. Složka obsahuje konfigurační soubor logovací knihovny log4j2 [27] `log4j2.xml`, soubor `priorities.properties` s definicí priorit pro výpočet pagerank algoritmu nad grafem událostí.

Dále obsahuje lokalizační soubor aplikace `visualization_cs.properties`, ikonu aplikace `visualization_icon.png` a SwiXML definice GUI: `visualization_main_window.xml`, `visualization_open_csv_dialog.xml` a `visualization_open_database_dialog.xml`.

7.8 Tipy pro budoucí rozšíření

Zde bych chtěl krátce uvést a popsat, jaké kroky by bylo nutné provést při úpravě nebo rozšíření stávající implementace a kde by bylo dobré začít.

7.8.1 Změna vykreslování grafu událostí

Vykreslování grafu událostí začíná ve třídě `GraphView` v metodě `drawGraph`. Samotné vykreslení grafu probíhá v několika fázích, viz 7.3.4.2. Pokud bychom chtěli upravit jednotlivé fáze, je metoda `drawGraph` dobrým začátečním bodem.

7.8.2 Změna vykreslování vrcholů grafu

Metoda pro vykreslování vrcholů `render` se nachází v obalové třídě `VertexWrapper`. Pokud chceme upravit pouze barvu jednotlivých typů událostí nebo barvu zvýraznění vrcholů, změníme příslušné hodnoty parametrů u komponenty `graphView` v XML definici hlavního okna `visualization_main_window.xml`. Pozor, barvy musí být zapsány v hexadecimálním formátu.

Parametry vrcholů v XML definici:

- `highlightColor` – barva zvýraznění vrcholu,
- `nodeEventColor` – barva vrcholu typu událost,
- `nodeItemColor` – barva vrcholu typu předmět,
- `nodePersonColor` – barva vrcholu typu osoba,
- `nodePlaceColor` – barva vrcholu typu místo a
- `nodeDefaultColor` – výchozí barva události.

7.8.3 Změna vykreslování hran grafu

Metoda pro vykreslování hran render se nachází v obalové třídě EdgeWrapper. Pokud chceme upravit pouze barvu hran a jejich zvýraznění, změníme příslušné hodnoty parametrů u komponenty `graphView` v XML definici hlavního okna `visualization_main_window.xml`.

Parametry hran v XML definici:

- `edgeColor` – barva hrany a
- `highlightColor` – barva zvýrazněné hrany (stejná jako v případě vrcholu).

7.8.4 Úprava layoutu grafu

V případě, že implementované rozložení grafu `TimeBasedLayout` nebude vyhovující, lze buď přímo přepsat výpočet rozložení v metodě `execute`, nebo vytvořit vlastní třídu implementující rozhraní `IGraphLayout` a napsat logiku rozložení vlastní. Dále je nutné změnit ve třídě `GraphView` v metodě `initialize` vytvoření nového layoutu místo stávajícího. Tím se bude rozložení grafu počítat přes nově vytvořený layout (viz popis v kapitole 7.3.4.5).

7.8.5 Rozšíření hlavního okna o nové menu položky

Přidání nové položky do menu hlavního okna vyžaduje splnění tří kroků:

1. přidání lokalizovaných řetězců (7.3.1.1 a 7.3.1.5),
2. přidání komponenty do XML definice hlavního oka (s využitím lokalizovaných řetězců, 7.3.1.1 a 7.3.1.5) a
3. implementace aplikační logiky ve třídě `MainWindow` (7.3.1.3).

7.8.6 Rozšíření o boční informační panel

Rozšíření aplikace o boční panel, který by zobrazoval detailní informace o vybraných událostech, by spočívalo ve splnění následujících kroků:

1. vytvoření nové grafické komponenty `InformationPanel` implementující rozhraní `IInteractionListener` a registrace této komponenty jako posluchače do delegáta (viz 7.6), tím bude zaručeno, že bude komponenta informovaná o jakékoli změně a interakci s časovou osou a grafem událostí,
2. přidání komponenty, respektive tagu `informationPanel` do XML definice hlavního okna `visualization_main_window.xml` a
3. přiřazení komponenty `informationPanel` ke třídě `InformationPanel` v konstruktoru třídy `MainWindow` ještě před generováním samotného grafického uživatelského rozhraní (viz 7.3.1.2).

8 Otestování vizualizace

Důležitou částí při implementaci bylo i ověření výsledného programu hlavně z hlediska zátěžových testů, tedy zjištění s kolika daty může program spolehlivě pracovat a dále ověření výsledné vizuální reprezentace a zjištění možných problémů nebo nedostatků, které by mohly být vylepšeny.

8.1 Testování výkonu aplikace

K účelům výkonnostního testování jsem napsal jednoduchý generátor CSV souborů, který podle zadaných parametrů vygeneruje daný počet událostí. Následně jsem vygenerované soubory použil jako zdroj dat pro testování.

8.1.1 Testovací prostředí

Výkonnostní testování probíhalo na osobním počítači s následující konfigurací:

- Microsoft Windows 8 x64,
- Intel® Core™ i7-4700MQ CPU @ 2.40GHz (až 3.40 GHz), 4 jádra (8 vláken) a
- 16 GB RAM.

8.1.2 Generátor dat

Každá událost má pevně nastavenou délku trvání na jeden měsíc a události na sebe těsně navazují. Každá událost je zároveň propojena hranou se svým bezprostředním předchůdcem a následníkem, počet vztahů je tedy o jedna menší než počet událostí. Ukázka kódu 8-1 a ukázka kódu 8-2 ukazuje, jak vypadají vygenerované CSV soubory se třemi událostmi.

```
1|Event|event 1|description 1|1000-01-01|1000-02-01
2|Event|event 2|description 2|1000-02-01|1000-03-01
3|Event|event 3|description 3|1000-03-01|1000-04-01
```

Ukázka kódu 8-1: Vygenerovaný CSV soubor s událostmi; 3 události

```
4|Cause|edge 1|1|2
5|Cause|edge 2|2|3
```

Ukázka kódu 8-2: Vygenerovaný soubor se vztahy; 2 vztahy

8.1.3 Měření

Pro testování jsem použil vygenerovaná data s počtem událostí od `10` až do `15000`. Veškeré měření hodnot (časy a FPS) probíhalo minimálně ve třech opakováních, ze kterých jsem posléze vypočetl průměrné hodnoty výsledků.

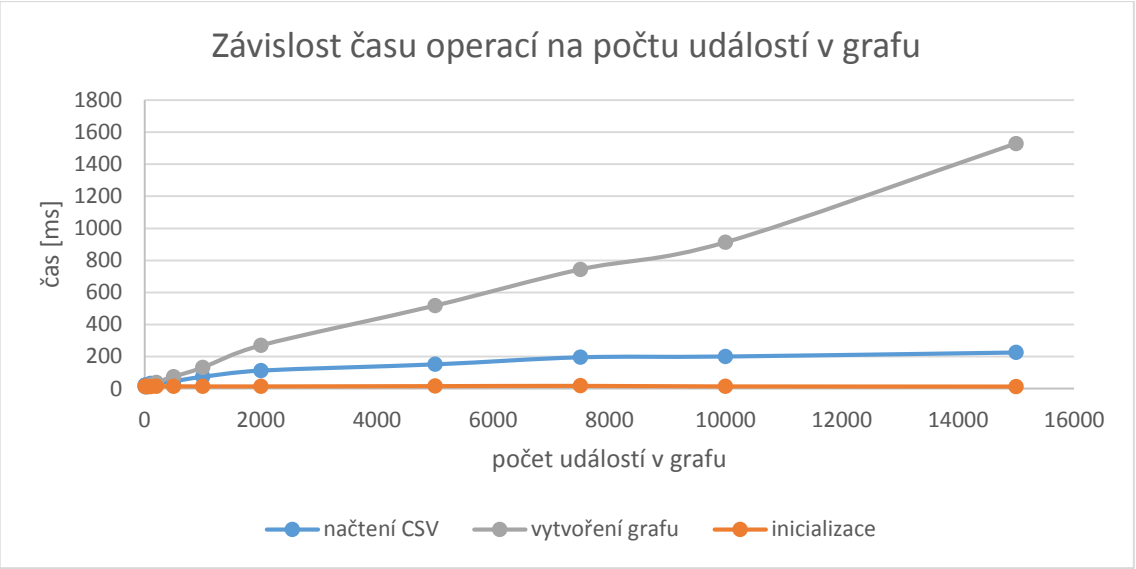
V prvním měření jsem se zaměřil na časy načtení CSV souborů, vytvoření reprezentace grafu ve třídě `GraphManager`, inicializaci všech komponent, výpočet rozložení grafu a celkového času od stisknutí tlačítka otevřít CSV soubory do konce výpočtu layoutu. Tabulka 8-1 obsahuje výsledky těchto měřených časů a graf 8-1 a graf 8-2 zobrazují závislosti mezi počtem událostí a časy výpočtů a operací.

Počet událostí	Časy [ms]				
	Načtení CSV	Vytvoření grafu	Inicializace	Výpočet layoutu	Celkem
10	22	12	14	1	49
20	19	11	14	1	46
50	20	12	12	1	46
100	32	23	12	5	72
200	35	39	15	12	101
500	47	75	15	27	163
1000	74	132	14	74	295
2000	113	270	14	706	1074
5000	152	518	16	13796	14483
7500	196	744	18	56510	57468
10000	201	913	14	124563	125692
15000	226	1529	13	637698	639466

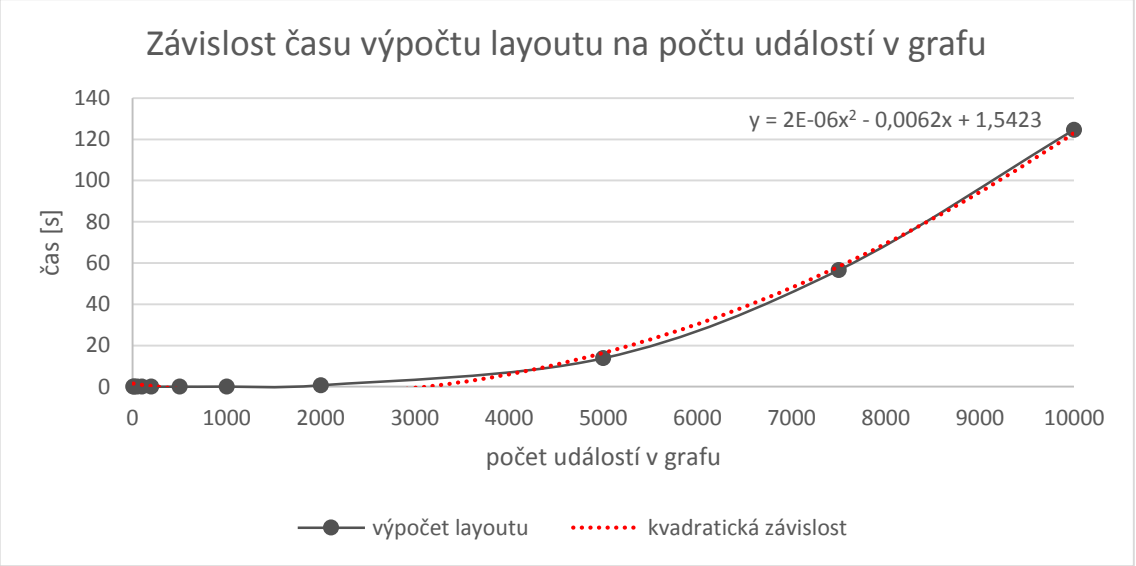
Tabulka 8-1: Závislost času výpočtů a operací na počtu událostí

Z tabulky naměřených hodnot (tabulka 8-1) a také grafu (graf 8-1) lze vyvodit závěry, že načtení CSV souborů a vytvoření grafu je lineárně závislé na počtu událostí, což odpovídá realitě, jelikož se musí pro každý vrchol provést určité operace. Dalším zjištěním je, že inicializace komponent (příprava vizuální reprezentace grafu událostí, časové osy a informačního panelu) nijak nezávisí na počtu načtených dat.

Čas výpočtu layoutu v tomto měření znamená kompletní vypočtení rozložení grafu po otevření souboru bez jakékoli interakce uživatele. Při otevření souboru (nebo databáze) je nastaven výchozí stav takový, že se celé časové období, které je v souboru (nebo databázi) zobrazí na viditelné časové ose. To znamená, že pro `10` událostí je toto období `10` měsíců, ale pro `15000` událostí je již toto období `1250` let. Je jasné, že všechny události spadají do tohoto období a tudíž musí být v tomto úseku rozmístěny. Jak již bylo napsáno v kapitolách 6.2 a 7.3.4.5, výpočet layoutu probíhá tak, že se vrcholy umisťují na pozici začátku události a pokud se překrývají s jinou událostí, jsou posunuty dolů ve vertikálním směru, tak aby se překrytí předešlo. S narůstajícím počtem událostí tedy dochází ke zpomalení výpočtu layoutu (výška rozložení se stále zvyšuje, tak aby se události nepřekrývaly). Mezi počtem událostí grafu a časem výpočtu existuje kvadratická závislost, kterou zobrazuje graf 8-2.



Graf 8-1: Závislost času načtení CSV, vytvoření grafu a inicializace na počtu událostí

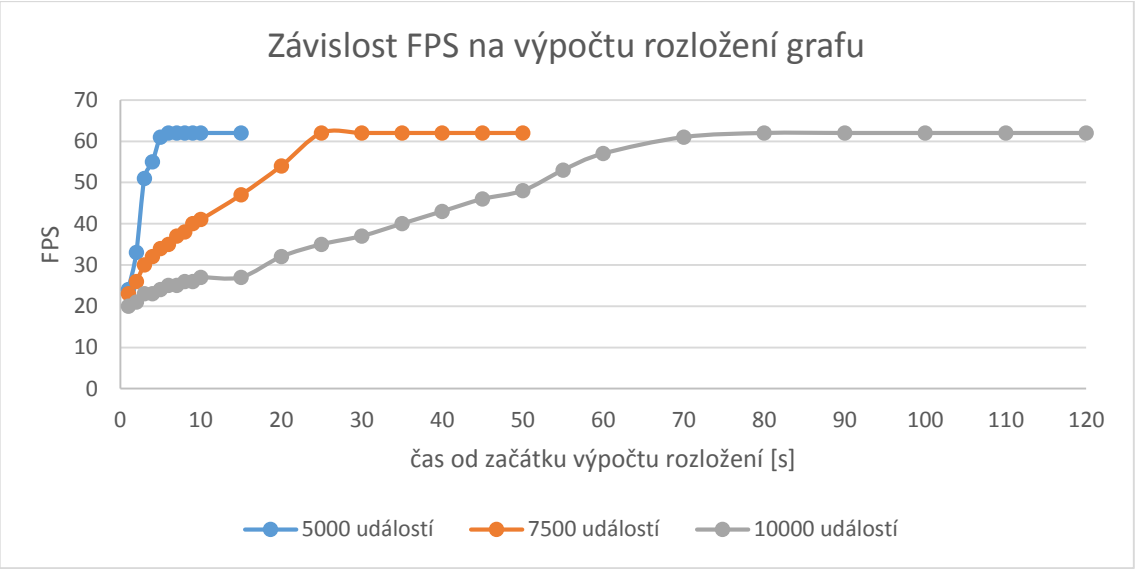


Graf 8-2: Závislost času výpočtu rozložení grafu na počtu událostí

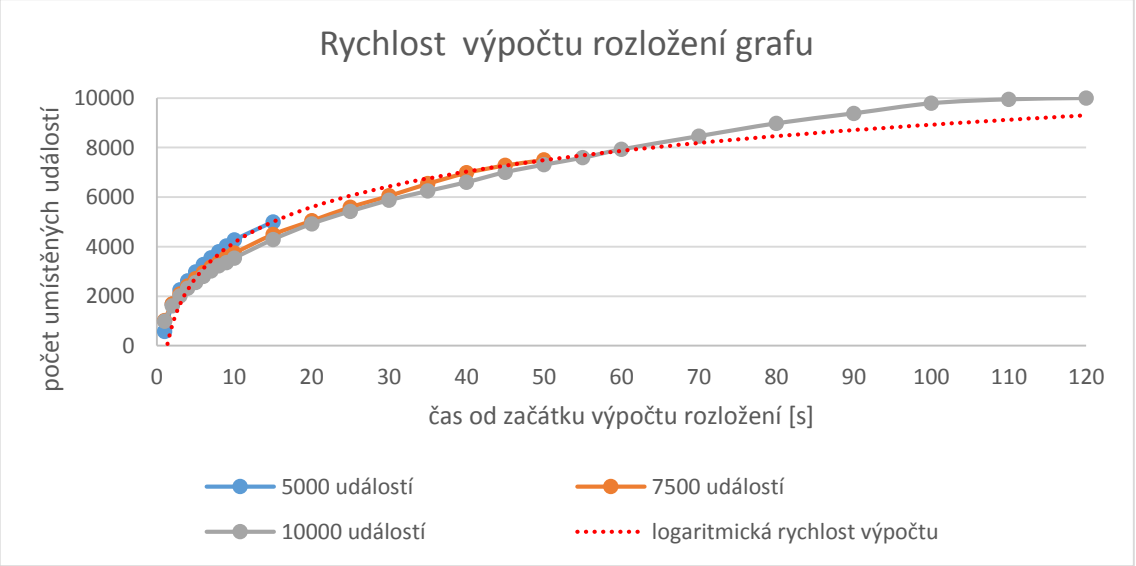
V druhém měření jsem se zaměřil na závislost FPS při výpočtu rozložení grafu, které je náročnou operací při narůstajícím počtu událostí. FPS je v programu omezeno na hodnotu 60 a může dosahovat mírně vyšších hodnot (až 63) díky nepřesnému plánování a uspávání vykreslovacího vlákna. Zároveň jsem měřil rychlost, s jakou se rozložení grafu počítá, tedy kolik vrcholů je umístěno na jejich finální pozice za jednotku času. Měření probíhalo od začátku spuštění výpočtu rozložení grafu až do jeho ukončení. Výsledky měření zobrazuje tabulka 8-2, závislost FPS na době výpočtu rozložení graf 8-3 a závislost rychlosti výpočtu graf 8-4.

Čas od začátku výpočtu rozložení [s]	5000 událostí		7500 událostí		10000 událostí	
	FPS	Umístěných událostí	FPS	Umístěných událostí	FPS	Umístěných událostí
1	24	569	23	1017	20	986
2	33	1674	26	1679	21	1605
3	51	2250	30	2090	23	2008
4	55	2615	32	2411	23	2321
5	61	2971	34	2711	24	2553
6	62	3279	35	2951	25	2795
7	62	3549	37	3172	25	3016
8	62	3785	38	3364	26	3222
9	62	4026	40	3572	26	3352
10	62	4273	41	3738	27	3530
15	62	5000	47	4489	27	4283
20			54	5044	32	4916
25			62	5594	35	5422
30			62	6050	37	5871
35			62	6543	40	6244
40			62	6981	43	6598
45			62	7283	46	7003
50			62	7500	48	7307
55					53	7588
60					57	7923
70					61	8460
80					62	8978
90					62	9380
100					62	9787
110					62	9948
120					62	10000

Tabulka 8-2: Závislost FPS a rychlost výpočtu rozložení grafu



Graf 8-3: Závislost FPS na době výpočtu rozložení grafu



Graf 8-4: Rychlost výpočtu rozložení grafu

O FPS, v závislosti na době výpočtu rozložení grafu, lze tvrdit, že se od počáteční hodnoty 20 FPS stále zvyšuje až do maximální hodnoty 60 FPS, které dosáhne přibližně v polovině z celkového času nutného na vypočtení rozložení grafu a poté se již této hodnoty drží a neklesá, viz graf 8-3.

Rychlost výpočtu rozložení, respektive počet umístěných vrcholů za jednotku času, je podle výsledků měření logaritmická, viz graf 8-4. Je to způsobeno tím, že se ve výpočtu rozložení snažíme eliminovat překrývající události a tedy s přibývajícím počtem událostí přibývá také množství porovnání.

8.2 Testování vizuální reprezentace

Pro otestování vizuální reprezentace a celé aplikace jsem připravil dotazník (viz *Příloha D – Dotazník*) a dal jsem aplikaci k dispozici dobrovolníkům k otestování.

8.2.1 Testovací data

Pro testování vizualizace jsem zvolil dvě historická témata: Domažlice a život Karla IV. Téma „Domažlice“ se zabývá historií vzniku města a událostmi, které s tímto městem souvisí. Celkem se toto téma skládá z 25 událostí a 30 závislostí mezi jednotlivými událostmi. Druhé téma „život Karla IV.“ je zaměřeno na jeho osobní vztahy (rodiče, manželství, potomci), korunovace a důležité události s ním spojené. Celkem toto téma obsahuje 27 událostí a 38 vztahů mezi nimi.

8.2.2 Výsledky, názory a připomínky

Příloha E – Vyplněné dotazníky obsahuje vyplněné dotazníky od dobrovolníků, kteří aplikaci testovali. Způsob hodnocení byl zvolen stejný jako v případě známkování ve škole: 1 – výborný až 5 – nedostatečný. Tabulka 8-3 a tabulka 8-4 obsahují průměrné výsledky hodnocení aplikace a vizualizace.

Kritérium	Průměrné hodnocení
1 Ovládání aplikace	1,67
2 Orientace v grafu událostí	1,67
3 Orientace v časové ose	1
4 Možnosti zobrazení	1,33
5 Možnosti interakce s časovou osou	1
6 Možnosti interakce s grafem událostí	2
7 Rozložení GUI	2

Tabulka 8-3: Výsledky globálního hodnocení aplikace

Téma	Průměrné hodnocení
1 Domažlice	1,67
2 Karel IV.	1,67

Tabulka 8-4: Výsledky hodnocení vizualizace a přehlednosti historických témat

Z výsledků lze usoudit, že dobrovolníci byli spokojení s ovládáním aplikace, orientací v grafu událostí a časové ose, možnostmi zobrazení a výsledkem rozložení grafu. Mírná nespokojenost byla s možnostmi interakce s grafem událostí. K tomuto názoru se také osobně přikláním. Obě testovaná témata se zdála dobrovolníkům přehledná a neměli k nim žádné připomínky.

Dobrovolníci byli také požádáni k sepsání případných připomínek k aplikaci jako celku. Jako hlavní připomínku bych zde zmínil umožnění posunu grafu událostí a časové osy levým tlačítkem myši a jejím táhnutím (původně se posun prováděl při stisku kolečka myši) a dále přidání historie otevřených souborů (případně databáze), kde by si mohl uživatel vybrat, které soubory z posledních otevřených chce opět otevřít.

8.3 Zhodnocení výsledků

Ze zátěžových testů je patrné, že program nemá problém se zobrazením většího počtu událostí najednou. Výpočet rozložení grafu se sice s přibývajícím množstvím událostí zpomaluje (očekávané chování), ale při praktickém použití zřejmě nebude aplikace zobrazovat tisíce událostí najednou, nebo zobrazovat stovky událostí na krátkém časovém úseku, kde by mohlo potencionálně docházet k častým překryvům událostí a tedy ke zpomalení výpočtu.

Z praktického testování vyšlo najevo, že je aplikace přehledná, ale měly by být vylepšeny a přidány možnosti práce a interakce s grafem událostí. Prozatím je v aplikaci implementovaná jen základní interakce (viz 9), která by měla být dále rozšířena. Velkým přínosem by bylo například skrývání událostí (viz 6.8), což by výrazně přispělo ke zpřehlednění zobrazovaných událostí. Dále by bylo přínosem i rozšíření aplikace o informační panel s detailními informacemi o vybraných událostech.

9 Závěr

Cílem mojí práce bylo seznámit se s problémem vizualizace časových os a grafů. Speciálně jsem se zaměřil na zobrazení grafu událostí s přidanými závislostmi. Dále jsem shrnul teoretické informace o existujících nástrojích a seznámil se s nimi. Součástí zadání bylo vytvoření aplikace, která dokáže vizualizovat časovou osu a graf a zároveň bude rozšířena o interaktivní prvky, které budou vylepšovat a doplňovat samotnou vizuální reprezentaci.

Výsledná aplikace je napsaná v programovacím jazyce Java a její součástí je datová vrstva grafu událostí [1] a API poskytující operace nad daty [2]. Celá aplikace je šířena pod licencí GNU LGPL verze 3 (text licence viz *Příloha B – Licence programu*). Program se skládá z několika komponent (časové osy, grafu událostí, manažeru vstupních dat a dalších). Grafické komponenty časové osy a grafu událostí jsem vytvořil vlastní. Z počátku jsem pro zobrazení grafu využíval knihovnu JGraphX, ale kvůli nesplnění funkčních a výkonnostních požadavků (viz 5.4) jsem se rozhodl pro implementaci své vlastní komponenty. Díky tomu je v programu zaručena plná kontrola kódu a také běhu aplikace.

Interakce s aplikací je v programu řešena přes delegáta, který informuje jednotlivé komponenty o právě probíhající interakci, na kterou mohou komponenty podle jejich účelu adekvátně reagovat. V kapitole 6 jsou popsané různé druhy úkolů a interakce, z nichž přibližně polovina je ve výsledné aplikaci zakomponovaná a další mohou sloužit jako možné budoucí rozšíření. Aplikace prozatím podporuje interakci ve formě: výpočtu časově orientovaného rozložení grafu, možnosti přiblížení a oddálení v časové ose (změna granularity času), posunu v časové ose a grafu událostí (nahlížení do minulosti a budoucnosti), vykreslení událostí jako časových bodů (délka trvání události je ignorována) nebo podle jejich skutečné délky trvání, zvýraznění událostí (v grafu a na časové ose), výpis a zvýraznění závislostí mezi událostmi a nakonec výpis informací o požadovaných událostech. Mezi hlavní rozšíření, které by aplikaci přineslo zpřehlednění zobrazovaných dat, patří možnost skrývání dat, případně jejich filtrace nebo seskupování.

Z výkonnostního testování aplikace je patrné, že aplikace bude dobře pracovat i s velkým počtem zobrazovaných dat (tisíce událostí). S přibývajícím počtem událostí sice stoupá čas potřebný na výpočet rozložení grafu, vždy ale bude záležet na jejich charakteru, zda jsou události shromážděné na krátkém časovém úseku nebo rozloženy do delšího období.

Testování vizuální reprezentace a aplikace jako celku dopadlo pozitivně. Vizualizace časové osy i grafu událostí se zdála dobrovolníkům přehledná, mírná nespokojenost zde byla pouze s rozsahem možností interakce s grafem událostí. S tímto názorem také souhlasím. Některé z dalších připomínek se týkaly aplikace samotné a ovládání. Ovládání aplikace jsem podle připomínek upravil a musím uznat, že nyní bude uživatelsky lepší.

Přehled zkratek

API	application programming interface
CG	computer graphics
CSS	cascading style sheets
CSV	comma-separated values
CT	computer tomography
FPS	frames per second
GUI	graphical user interface
HTML	hypertext markup language
HW	hardware
JSON	javascript object notation
MRI	magnetic resonance imaging
MVC	model view controller
RAM	random-access memory
SVG	scalable vector graphics
TOC	theory of constraints
URL	uniform resource locator
UTC	coordinated universal time
XML	extensible markup language

Literatura

[1] **Merunko, David.** *Generování a vizualizace časové osy.* Plzeň : David Merunko, 2015. diplomová práce, zatím neobhájená.

[2] **Hrbáček, David.** *Zpracování časových údajů pro jejich vizualizaci.* Plzeň : David Hrbáček, 2015. diplomová práce, zatím neobhájená.

[3] **Owen, G. Scott.** History of Visualization. *ACM SIGGRAPH.* [Online] 11. 2. 1999. [Citace: 11. 4. 2015.] http://www.siggraph.org/education/materials/HyperVis/visgoals/visgoal3.htm.

[4] **Ptolemaios, Klaudios.** Harley 7182 Map of the world. *The British Library.* [Online] 27. 7. 2014. [Citace: 11. 4. 2015.] obrázek. http://www.bl.uk/catalogues/illuminatedmanuscripts/ILLUMIN.ASP?Size=mid&IllID=28894.

[5] **Minard, Charles Joseph.** Minard - Napoleon's disastrous Russian campaign of 1812. *Wikipedia.org.* [Online] 17. 1. 2008. [Citace: 11. 4. 2015.] obrázek. http://en.wikipedia.org/wiki/Charles_Joseph_Minard#/media/File:Minard.png.

[6] **Ward, Matthew O., Grinstein, Georges G. a Keim, Daniel.** *Interactive data visualization: foundations, techniques, and applications.* místo neznámé : A K Peters/CRC Press, 2010. ISBN 978-1-56881-473-5.

[7] **Aigner, Wolfgang, a další.** *Visualization of Time-Oriented Data.* místo neznámé : Springer, 2011. Human-Computer Interaction Series. ISBN 978-0-85729-078-6.

[8] **Frank, Andrew U.** *Spatial and Temporal Reasoning in Geographic Information Systems.* New York : Oxford University Press, 1998. stránky 40-61. kapitola: Different Types of "Times" in GIS. ISBN-13: 978-0195103427.

[9] **Joda-Time.** Joda-Time. [Online] joda.org, 2015. [Citace: 11. 5. 2015.] Apache 2.0 licence. http://www.joda.org/joda-time/.

[10] **Bostock, Mike.** D3.js. *Data-Driven Documents.* [Online] 2013. [Citace: 21. 4. 2015.] URL adresa autora: http://bost.ocks.org/mike/, BSD licence. http://d3js.org/.

[11] **Cook, Peter.** F1 Timeline. *Animated Data.* [Online] 2014. [Citace: 21. 4. 2015.] http://charts.animateddata.co.uk/f1/.

[12] **Huynh, David Francois.** Timeline. *Simple Widgets.* [Online] Massachusetts Institute of Technology, 2009. [Citace: 21. 4. 2015.] BSD licence. http://www.simile-widgets.org/timeline/.

[13] **Knight Foundation a Robert R. McCormick Foundation.** Timeline JS - Beautifully crafted timelines that are easy, and intuitive to use. *NORTHWESTERN UNIVERSITY knight lab.* [Online] Knight Foundation, 2013. [Citace: 22. 4. 2015.] Mozilla Public Licence, v. 2.0.. http://timeline.knightlab.com/.

[14] **vis.js.** vis.js - A dynamic, browser based visualization library. *vis.js.* [Online] 2015. [Citace: 22. 4. 2015.] Apache 2.0 a MIT licence. http://visjs.org/.

[15] **jaret.de.** jaret timebars - jaret timebars (gantt chart and timeline widget/control for SWT and Swing). *jaret.de.* [Online] 2013. [Citace: 22. 4. 2015.] GNU Public licence. http://jaret.de/timebars/.

[16] **Lemmermann, Dirk.** DLSC. [Online] DLSC, 2015. [Citace: 22. 4. 2015.] http://dlsc.com/.

[17] **Žáček, Jakub.** *Knihovna pro analýzu grafových struktur.* Plzeň : Jakub Žáček, 2014. bakalářská práce.

[18] **Čada, Roman, Kaiser, Tomáš a Ryjáček, Zdeněk.** *Diskrétní matematika.* [Dokuement] Plzeň : Katedra matematiky FAV, Západočeská Univerzita v Plzni, 2004. skripta. dostupné z: http://www.cam.zcu.cz/~ryjacek/students/DMA/skripta/dma.pdf.

[19] **Telea, Alexandru C.** *Data Visualization: Principles and Practice.* místo neznámé : A K Peters/CRC Pess, 2007. ISBN-13: 978-1568813066.

[20] **JGraphX.** JGraphX. *jgraph/jgraphx - GitHub.* [Online] JGraph Ltd, 2015. [Citace: 26. 4. 2015.] BSD licence. https://github.com/jgraph/jgraphx.

[21] **GraphStream.** GraphStream - A Dynamic Graph Library. *graphstream-project.org.* [Online] GraphStream Team, 2013. [Citace: 26. 4. 2015.] CeCILL-C a GNU LGPL licence. http://graphstream-project.org/.

[22] **mxGraph.** JavaScript Diagraming. *jgraph.com.* [Online] JGraph Ltd, 2015. [Citace: 26. 4. 2015.] komerční nástroj. https://www.jgraph.com/.

[23] **draw.io.** draw.io. [Online] JGraph Ltd, 2015. [Citace: 27. 4. 2015.] online vizualizační nástroj. https://www.draw.io/.

[24] **JGraph Forum.** mxGraph Technical Support Forum. *jgraph connecting the dots.* [Online] JGraph Ltd. [Citace: 1. 5. 2015.] http://forum.jgraph.com/.

[25] **JGraph Stackoverflow.** *stackoverflow.com.* [Online] [Citace: 1. 5. 2015.] http://stackoverflow.com/questions/tagged/mxgraph.

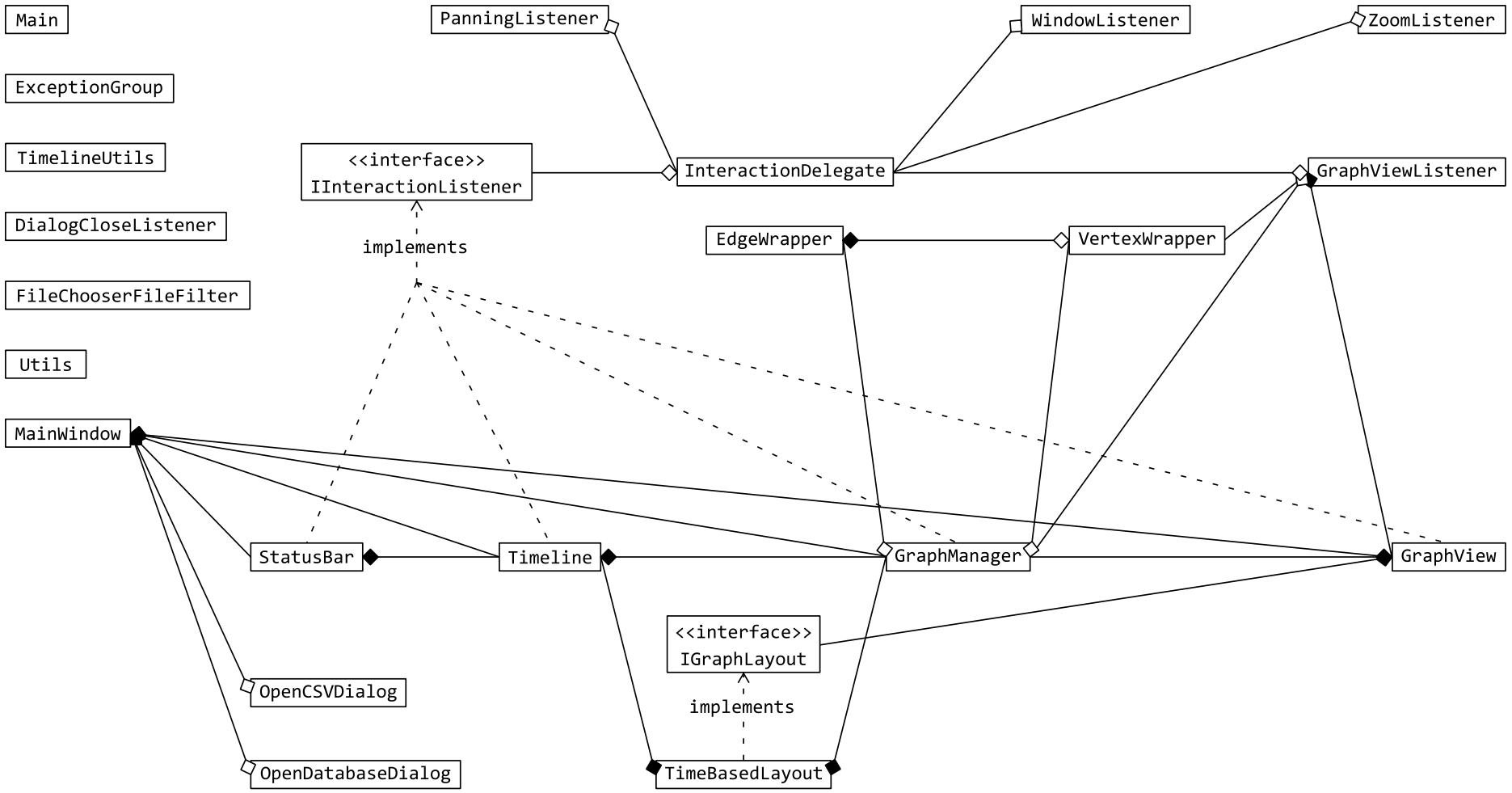
[26] **SwiXML.** SWIXML - Generate javax.swing at runtime based on XML descriptors. *swixml.org.* [Online] 2015. [Citace: 4. 5. 2015.] Apache-styl licence, open-source. http://www.swixml.org/.

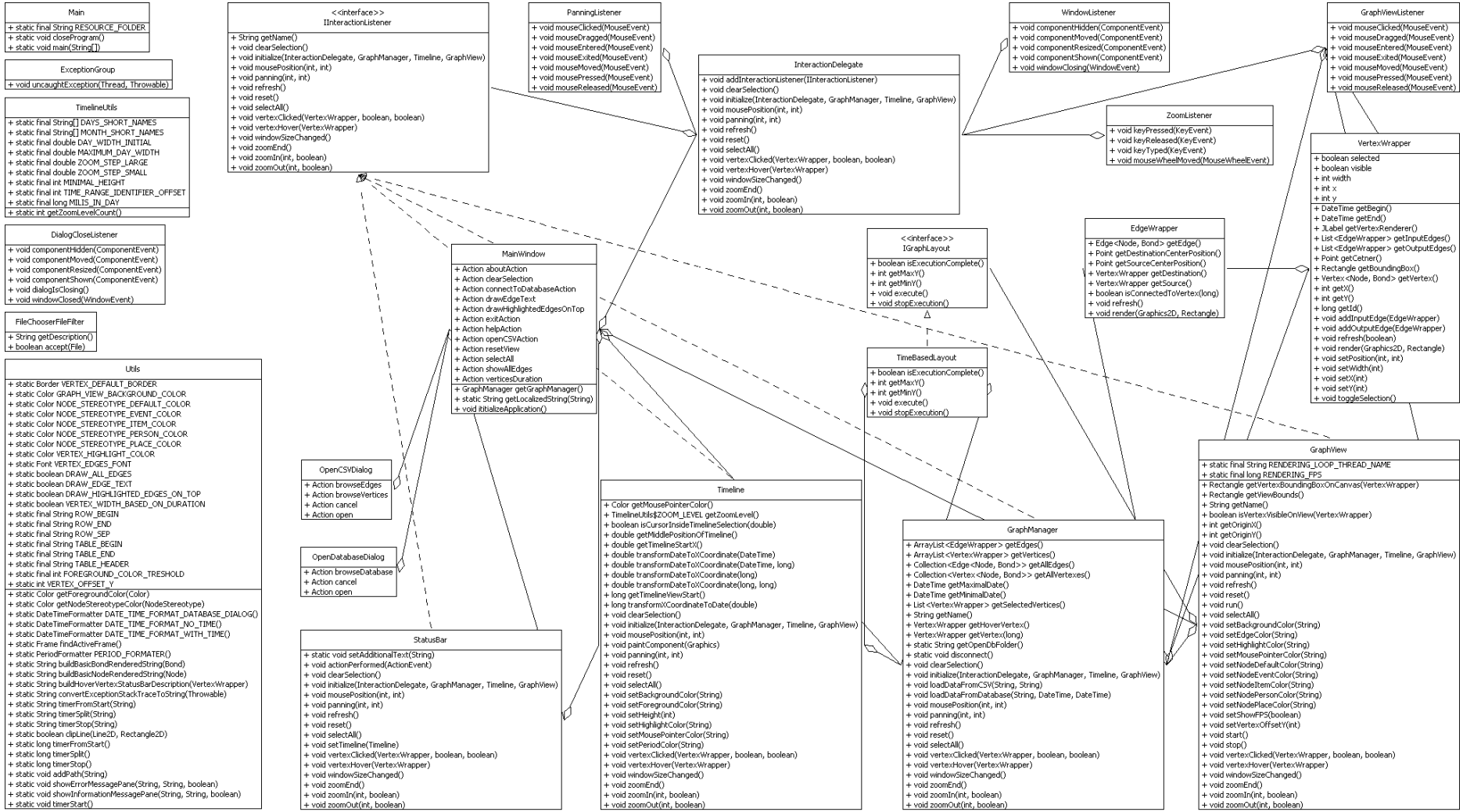
[27] **log4j2.** Log4j 2 Guide - Apache Log4j 2. *Logging Services.* [Online] Apache, 2015. [Citace: 7. 5. 2015.] Apache 2.0 licence. http://logging.apache.org/log4j/2.x/.

[28] **neo4j.** Neo4j, the World's Leading Graph Database. *neo4j.* [Online] Neo Technology, Inc., 2015. [Citace: 7. 5. 2015.] http://neo4j.com/.

Přílohy

Příloha A - Class diagram


Main
ExceptionGroup
TimelineUtils
DialogCloseListener
FileChooserFileFilter
Utils
MainWindow
PanningListener
WindowListener
ZoomListener
<<interface>>
IInteractionListener
InteractionDelegate
GraphViewListener
implements
EdgeWrapper
VertexWrapper
StatusBar
Timeline
GraphManager
GraphView
<<interface>>
IGraphLayout
implements
OpenCSVDialog
OpenDatabaseDialog
TimeBasedLayout


Main
+ static final String RESOURCE_FOLDER
+ static void closeProgram()
+ static void main(String[])
ExceptionGroup
+ void uncaughtException(Thread, Throwable)
TimelineUtils
+ static final String[] DAYS_SHORT_NAMES
+ static final String[] MONTH_SHORT_NAMES
+ static final double DAY_WIDTH_INITIAL
+ static final double MAXIMUM_DAY_WIDTH
+ static final double ZOOM_STEP_LARGE
+ static final double ZOOM_STEP_SMALL
+ static final int MINIMAL_HEIGHT
+ static final int TIME_RANGE_IDENTIFIER_OFFSET
+ static final long MILIS_IN_DAY
+ static int getZoomLevelCount()
DialogCloseListener
+ void componentHidden(ComponentEvent)
+ void componentMoved(ComponentEvent)
+ void componentResized(ComponentEvent)
+ void componentShown(ComponentEvent)
+ void dialogIsClosing()
+ void windowClosed(WindowEvent)
FileChooserFileFilter
+ String getDescription()
+ boolean accept(File)
Utils
+ static Border VERTEX_DEFAULT_BORDER
+ static Color GRAPH_VIEW_BACKGROUND_COLOR
+ static Color NODE_STEREOTYPE_DEFAULT_COLOR
+ static Color NODE_STEREOTYPE_EVENT_COLOR
+ static Color NODE_STEREOTYPE_ITEM_COLOR
+ static Color NODE_STEREOTYPE_PERSON_COLOR
+ static Color NODE_STEREOTYPE_PLACE_COLOR
+ static Color VERTEX_HIGHLIGHT_COLOR
+ static Font VERTEX_EDGES_FONT
+ static boolean DRAW_ALL_EDGES
+ static boolean DRAW_EDGE_TEXT
+ static boolean DRAW_HIGHLIGHTED_EDGES_ON_TOP
+ static boolean VERTEX_WIDTH_BASED_ON_DURATION
+ static final String ROW_BEGIN
+ static final String ROW_END
+ static final String ROW_SEP
+ static final String TABLE_BEGIN
+ static final String TABLE_END
+ static final String TABLE_HEADER
+ static final int FOREGROUND_COLOR_TRESHOLD
+ static int VERTEX_OFFSET_Y
+ static Color getForegroundColor(Color)
+ static Color getNodeStereotypeColor(NodeStereotype)
+ static DateTimeFormatter DATE_TIME_FORMAT_DATABASE_DIALOG()
+ static DateTimeFormatter DATE_TIME_FORMAT_NO_TIME()
+ static DateTimeFormatter DATE_TIME_FORMAT_WITH_TIME()
+ static Frame findActiveFrame()
+ static PeriodFormatter PERIOD_FORMATER()
+ static String buildBasicBondRenderedString(Bond)
+ static String buildBasicNodeRenderedString(Node)
+ static String buildHoverVertexStatusBarDescription(VertexWrapper)
+ static String convertExceptionStackTraceToString(Throwable)
+ static String timerFromStart(String)
+ static String timerSplit(String)
+ static String timerStop(String)
+ static boolean clipLine(Line2D, Rectangle2D)
+ static long timerFromStart()
+ static long timerSplit()
+ static long timerStop()
+ static void addPath(String)
+ static void showErrorMessagePane(String, String, boolean)
+ static void showInformationMessagePane(String, String, boolean)
+ static void timerStart()
<<interface>>
IInteractionListener
+ String getName()
+ void clearSelection()
+ void initialize(InteractionDelegate, GraphManager, Timeline, GraphView)
+ void mousePosition(int, int)
+ void panning(int, int)
+ void refresh()
+ void reset()
+ void selectAll()
+ void vertexClicked(VertexWrapper, boolean, boolean)
+ void vertexHover(VertexWrapper)
+ void windowSizeChanged()
+ void zoomEnd()
+ void zoomIn(int, boolean)
+ void zoomOut(int, boolean)
PanningListener
+ void mouseClicked(MouseEvent)
+ void mouseDragged(MouseEvent)
+ void mouseEntered(MouseEvent)
+ void mouseExited(MouseEvent)
+ void mouseMoved(MouseEvent)
+ void mousePressed(MouseEvent)
+ void mouseReleased(MouseEvent)
InteractionDelegate
+ void addInteractionListener(IInteractionListener)
+ void clearSelection()
+ void initialize(InteractionDelegate, GraphManager, Timeline, GraphView)
+ void mousePosition(int, int)
+ void panning(int, int)
+ void refresh()
+ void reset()
+ void selectAll()
+ void vertexClicked(VertexWrapper, boolean, boolean)
+ void vertexHover(VertexWrapper)
+ void windowSizeChanged()
+ void zoomEnd()
+ void zoomIn(int, boolean)
+ void zoomOut(int, boolean)
WindowListener
+ void componentHidden(ComponentEvent)
+ void componentMoved(ComponentEvent)
+ void componentResized(ComponentEvent)
+ void componentShown(ComponentEvent)
+ void windowClosing(WindowEvent)
ZoomListener
+ void keyPressed(KeyEvent)
+ void keyReleased(KeyEvent)
+ void keyTyped(KeyEvent)
+ void mouseWheelMoved(MouseWheelEvent)
GraphViewListener
+ void mouseClicked(MouseEvent)
+ void mouseDragged(MouseEvent)
+ void mouseEntered(MouseEvent)
+ void mouseExited(MouseEvent)
+ void mouseMoved(MouseEvent)
+ void mousePressed(MouseEvent)
+ void mouseReleased(MouseEvent)
VertexWrapper
+ boolean selected
+ boolean visible
+ int width
+ int x
+ int y
+ DateTime getBegin()
+ DateTime getEnd()
+ JLabel getVertexRenderer()
+ List<EdgeWrapper> getInputEdges()
+ List<EdgeWrapper> getOutputEdges()
+ Point getCetner()
+ Rectangle getBoundingBox()
+ Vertex<Node, Bond> getVertex()
+ int getX()
+ int getY()
+ long getId()
+ void addInputEdge(EdgeWrapper)
+ void addOutputEdge(EdgeWrapper)
+ void refresh(boolean)
+ void render(Graphics2D, Rectangle)
+ void setPosition(int, int)
+ void setWidth(int)
+ void setX(int)
+ void setY(int)
+ void toggleSelection()
MainWindow
+ Action aboutAction
+ Action clearSelection
+ Action connectToDatabaseAction
+ Action drawEdgeText
+ Action drawHighlightedEdgesOnTop
+ Action exitAction
+ Action helpAction
+ Action openCSVAction
+ Action resetView
+ Action selectAll
+ Action showAllEdges
+ Action verticesDuration
+ GraphManager getGraphManager()
+ static String getLocalizedString(String)
+ void ititializeApplication()
<<interface>>
IGraphLayout
+ boolean isExecutionComplete()
+ int getMaxY()
+ int getMinY()
+ void execute()
+ void stopExecution()
EdgeWrapper
+ Edge<Node, Bond> getEdge()
+ Point getDestinationCenterPosition()
+ Point getSourceCenterPosition()
+ VertexWrapper getDestination()
+ VertexWrapper getSource()
+ boolean isConnectedToVertex(long)
+ void refresh()
+ void render(Graphics2D, Rectangle)
TimeBasedLayout
+ boolean isExecutionComplete()
+ int getMaxY()
+ int getMinY()
+ void execute()
+ void stopExecution()
OpenCSVDialog
+ Action browseEdges
+ Action browseVertices
+ Action cancel
+ Action open
OpenDatabaseDialog
+ Action browseDatabase
+ Action cancel
+ Action open
StatusBar
+ static void setAdditionalText(String)
+ void actionPerformed(ActionEvent)
+ void clearSelection()
+ void initialize(InteractionDelegate, GraphManager, Timeline, GraphView)
+ void mousePosition(int, int)
+ void panning(int, int)
+ void refresh()
+ void reset()
+ void selectAll()
+ void setTimeline(Timeline)
+ void vertexClicked(VertexWrapper, boolean, boolean)
+ void vertexHover(VertexWrapper)
+ void windowSizeChanged()
+ void zoomEnd()
+ void zoomIn(int, boolean)
+ void zoomOut(int, boolean)
Timeline
+ Color getMousePointerColor()
+ TimelineUtils$ZOOM_LEVEL getZoomLevel()
+ boolean isCursorInsideTimelineSelection(double)
+ double getMiddlePositionOfTimeline()
+ double getTimelineStartX()
+ double transformDateToXCoordinate(DateTime)
+ double transformDateToXCoordinate(DateTime, long)
+ double transformDateToXCoordinate(long)
+ double transformDateToXCoordinate(long, long)
+ long getTimelineViewStart()
+ long transformXCoordinateToDate(double)
+ void clearSelection()
+ void initialize(InteractionDelegate, GraphManager, Timeline, GraphView)
+ void mousePosition(int, int)
+ void paintComponent(Graphics)
+ void panning(int, int)
+ void refresh()
+ void reset()
+ void selectAll()
+ void setBackgroundColor(String)
+ void setForegroundColor(String)
+ void setHeight(int)
+ void setHighlightColor(String)
+ void setMousePointerColor(String)
+ void setPeriodColor(String)
+ void vertexClicked(VertexWrapper, boolean, boolean)
+ void vertexHover(VertexWrapper)
+ void windowSizeChanged()
+ void zoomEnd()
+ void zoomIn(int, boolean)
+ void zoomOut(int, boolean)
GraphManager
+ ArrayList<EdgeWrapper> getEdges()
+ ArrayList<VertexWrapper> getVertices()
+ Collection<Edge<Node, Bond>> getAllEdges()
+ Collection<Vertex<Node, Bond>> getAllVertexes()
+ DateTime getMaximalDate()
+ DateTime getMinimalDate()
+ List<VertexWrapper> getSelectedVertices()
+ String getName()
+ VertexWrapper getHoverVertex()
+ VertexWrapper getVertex(long)
+ static String getOpenDbFolder()
+ static void disconnect()
+ void clearSelection()
+ void initialize(InteractionDelegate, GraphManager, Timeline, GraphView)
+ void loadDataFromCSV(String, String)
+ void loadDataFromDatabase(String, DateTime, DateTime)
+ void mousePosition(int, int)
+ void panning(int, int)
+ void refresh()
+ void reset()
+ void selectAll()
+ void vertexClicked(VertexWrapper, boolean, boolean)
+ void vertexHover(VertexWrapper)
+ void windowSizeChanged()
+ void zoomEnd()
+ void zoomIn(int, boolean)
+ void zoomOut(int, boolean)
GraphView
+ static final String RENDERING_LOOP_THREAD_NAME
+ static final long RENDERING_FPS
+ Rectangle getVertexBoundingBoxOnCanvas(VertexWrapper)
+ Rectangle getViewBounds()
+ String getName()
+ boolean isVertexVisibleOnView(VertexWrapper)
+ int getOriginX()
+ int getOriginY()
+ void clearSelection()
+ void initialize(InteractionDelegate, GraphManager, Timeline, GraphView)
+ void mousePosition(int, int)
+ void panning(int, int)
+ void refresh()
+ void reset()
+ void run()
+ void selectAll()
+ void setBackgroundColor(String)
+ void setEdgeColor(String)
+ void setHighlightColor(String)
+ void setMousePointerColor(String)
+ void setNodeDefaultColor(String)
+ void setNodeEventColor(String)
+ void setNodeItemColor(String)
+ void setNodePersonColor(String)
+ void setNodePlaceColor(String)
+ void setShowFPS(boolean)
+ void setVertexOffsetY(int)
+ void start()
+ void stop()
+ void vertexClicked(VertexWrapper, boolean, boolean)
+ void vertexHover(VertexWrapper)
+ void windowSizeChanged()
+ void zoomEnd()
+ void zoomIn(int, boolean)
+ void zoomOut(int, boolean)

Příloha B – Licence programu

GNU LESSER GENERAL PUBLIC LICENSE
Version 3, 29 June 2007

This version of the GNU Lesser General Public License incorporates the terms and conditions of version 3 of the GNU General Public License, supplemented by the additional permissions listed below.

0. Additional Definitions.

As used herein, "this License" refers to version 3 of the GNU Lesser General Public License, and the "GNU GPL" refers to version 3 of the GNU General Public License.

"The Library" refers to a covered work governed by this License, other than an Application or a Combined Work as defined below.

An "Application" is any work that makes use of an interface provided by the Library, but which is not otherwise based on the Library. Defining a subclass of a class defined by the Library is deemed a mode of using an interface provided by the Library.

A "Combined Work" is a work produced by combining or linking an Application with the Library. The particular version of the Library with which the Combined Work was made is also called the "Linked Version".

The "Minimal Corresponding Source" for a Combined Work means the Corresponding Source for the Combined Work, excluding any source code for portions of the Combined Work that, considered in isolation, are based on the Application, and not on the Linked Version.

The "Corresponding Application Code" for a Combined Work means the object code and/or source code for the Application, including any data and utility programs needed for reproducing the Combined Work from the Application, but excluding the System Libraries of the Combined Work.

1. Exception to Section 3 of the GNU GPL.

You may convey a covered work under sections 3 and 4 of this License without being bound by section 3 of the GNU GPL.

2. Conveying Modified Versions.

If you modify a copy of the Library, and, in your modifications, a facility refers to a function or data to be supplied by an Application that uses the facility (other than as an argument passed when the facility is invoked), then you may convey a copy of the modified version:

a) under this License, provided that you make a good faith effort to ensure that, in the event an Application does not supply the function or data, the facility still operates, and performs whatever part of its purpose remains meaningful, or

b) under the GNU GPL, with none of the additional permissions of this License applicable to that copy.

3. Object Code Incorporating Material from Library Header Files.

The object code form of an Application may incorporate material from a header file that is part of the Library. You may convey such object code under terms of your choice, provided that, if the incorporated material is not limited to numerical parameters, data structure layouts and accessors, or small macros, inline functions and templates (ten or fewer lines in length), you do both of the following:

a) Give prominent notice with each copy of the object code that the Library is used in it and that the Library and its use are covered by this License.

b) Accompany the object code with a copy of the GNU GPL and this license document.

4. Combined Works.

You may convey a Combined Work under terms of your choice that, taken together, effectively do not restrict modification of the portions of the Library contained in the Combined Work and reverse engineering for debugging such modifications, if you also do each of the following:

a) Give prominent notice with each copy of the Combined Work that the Library is used in it and that the Library and its use are covered by this License.

b) Accompany the Combined Work with a copy of the GNU GPL and this license document.

c) For a Combined Work that displays copyright notices during execution, include the copyright notice for the Library among these notices, as well as a reference directing the user to the copies of the GNU GPL and this license document.

d) Do one of the following:

0) Convey the Minimal Corresponding Source under the terms of this License, and the Corresponding Application Code in a form suitable for, and under terms that permit, the user to recombine or relink the Application with a modified version of the Linked Version to produce a modified Combined Work, in the manner specified by section 6 of the GNU GPL for conveying Corresponding Source.

1) Use a suitable shared library mechanism for linking with the Library. A suitable mechanism is one that (a) uses at run time a copy of the Library already present on the user's computer system, and (b) will operate properly with a modified version of the Library that is interface-compatible with the Linked Version.

e) Provide Installation Information, but only if you would otherwise be required to provide such information under section 6 of the GNU GPL, and only to the extent that such information is necessary to install and execute a modified version of the Combined Work produced by recombining or relinking the Application with a modified version of the Linked Version. (If you use option 4d0, the Installation Information must accompany the Minimal Corresponding Source and Corresponding Application Code. If you use option 4d1, you must provide the Installation Information in the manner specified by section 6 of the GNU GPL for conveying Corresponding Source.)

5. Combined Libraries.

You may place library facilities that are a work based on the Library side by side in a single library together with other library facilities that are not Applications and are not covered by this License, and convey such a combined library under terms of your choice, if you do both of the following:

a) Accompany the combined library with a copy of the same work based on the Library, uncombined with any other library facilities, conveyed under the terms of this License.

b) Give prominent notice with the combined library that part of it is a work based on the Library, and explaining where to find the accompanying uncombined form of the same work.

6. Revised Versions of the GNU Lesser General Public License.

The Free Software Foundation may publish revised and/or new versions of the GNU Lesser General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Library as you received it specifies that a certain numbered version of the GNU Lesser General Public License "or any later version" applies to it, you have the option of following the terms and conditions either of that published version or of any later version published by the Free Software Foundation. If the Library as you received it does not specify a version number of the GNU Lesser General Public License, you may choose any version of the GNU Lesser General Public License ever published by the Free Software Foundation.

If the Library as you received it specifies that a proxy can decide whether future versions of the GNU Lesser General Public License shall apply, that proxy's public statement of acceptance of any version is permanent authorization for you to choose that version for the Library.

Příloha C – Uživatelská příručka

Překlad a sestavení programu

Pro překlad programu je nutné mít na počítači nainstalovaný sestavovací nástroj Apache Maven. Lze jej stáhnout z webové adresy https://maven.apache.org/. Jelikož je tento nástroj kompletně vytvořen v programovacím jazyce Java, je nutné mít na počítači nainstalované její vývojové prostředí v minimální verzi 1.7. Vývojové prostředí programovacího jazyka Java lze stáhnout z adresy: http://www.oracle.com/technetwork/java/javase/downloads/index.html.

Projekt `Visualization` je závislý na dalších projektech: `db-graph-ranking`, `graph`, `graph-ranking` a `TimelineDatabase`. Všechny projekty jsou společně uloženy ve verzovacím systému SVN, který je dostupný na adrese: https://subversion.assembla.com/svn/timeline_generating/.

Následující seznam zobrazuje strukturu projektů s vyznačením důležitých souborů:

- svn/timeline_generating/
  - branches
    - node_ranking_hrbacek
      - **db-graph-ranking**
        - mvnPackAndInstall.bat
        - pom.xml
      - **graph**
        - mvnPackAndInstall.bat
        - pom.xml
      - **graph-ranking**
        - mvnPackAndInstall.bat
        - pom.xml
    - **TimelineDatabase**
      - mvnPackAndInstall.bat
      - pom.xml
    - visualization_jmoulis
      - **Visulization**
        - lib
          - swixml-2.4.jar
        - resources
          - log4j2.xml
          - priorities.properties
          - visualization_cs.properties
          - visualization_icon.png
          - visualization_main_window.xml
          - visualization_open_csv_dialog.xml
          - visualization_open_database_dialog.xml
        - build.bat
        - pom.xml

Veškeré projekty jsou řízeny a sestavovány nástrojem Maven a obsahují tedy vlastní `pom.xml` soubor. Zároveň každý ze závislých projektů má vedle `pom.xml` také dávkový soubor `mvnPackAndInstall.bat`, který má na starost spuštění překladu projektu a jeho instalaci do lokálního Maven repositáře. Projekt `Visualization` obsahuje také vlastní `pom.xml` a zároveň dávkový soubor `build.bat`, který má na starost spuštění překladu závislých projektů a následně sestavení projektu `Visualization` do spustitelného souboru `JAR`.

Překlad závislých projektů díky závislostem mezi nimi samotnými musí probíhat v následujícím pořadí:

1. `graph`,
2. `graph-ranking`,
3. `TimelineDatabase` a
4. `db-graph-ranking`.

Programy jsou dále závislé na externích knihovnách (práce s časem, logování událostí a tak dále), které jsou definované v `pom.xml` souborech a jsou stahované z online Maven repositářů. Projekt `Visualization` je ale závislý také na knihovně SwiXML (verze 2.4), která se v online Maven repositářích nenachází. Proto je knihovna umístěná ve složce projektu ve složce `lib`, odkud je při sestavení nainstalovaná do lokálního repositáře.

Vlastní sestavení programu probíhá spuštěním dávkového souboru `build.bat`. Po jeho spuštění je uživatel dotázán, zda si přeje sestavit i závislé projekty nebo jen samotný projekt `Visualization`. Po úspěšném doběhnutí překladu a sestavení se v projektu `Visualization` vytvoří složka `build`, ve které bude spustitelný soubor programu `visualization.jar` a složka `resources` se soubory, které jsou nutné pro správný běh programu (složka `resources` se nachází přímo v kořenovém adresáři projektu `Visualization` a její obsah společně s ní je zkopírován do složky `build`).

Spuštění programu

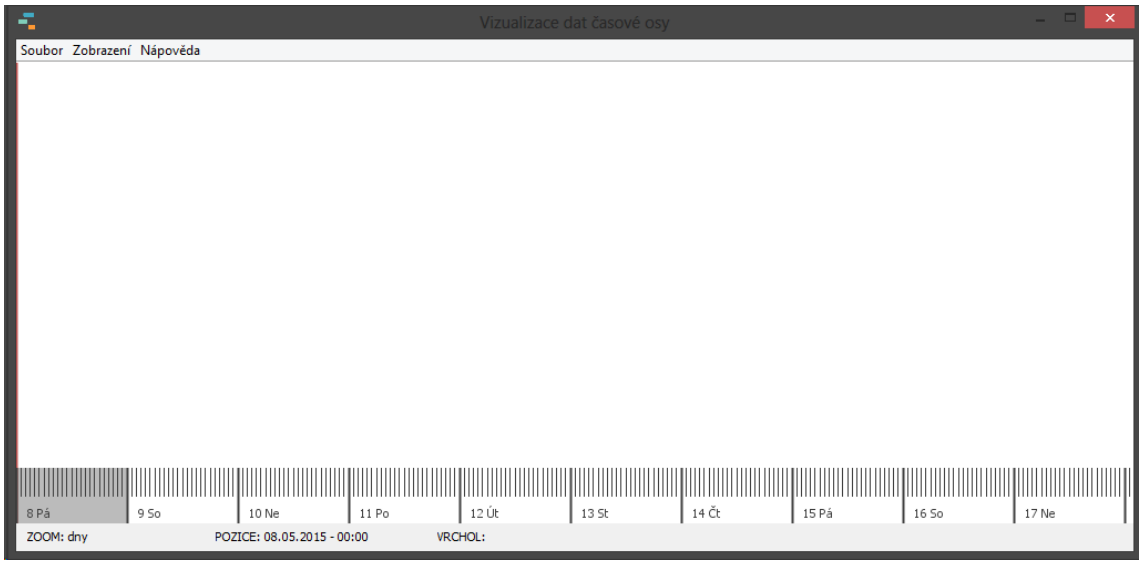
Program lze spustit jak pod operačním systémem Windows, tak i pod operačním systémem Linux. Jeho vývoj byl prováděn na systému Windows 8, testování probíhalo na systémech Windows 8 a Ubuntu. Při spuštění na operačním systému Linux může docházet k chybám ve výpisu textů, způsobených odlišným zobrazováním fontů.

Program je napsán v programovacím jazyce Java a pro jeho spuštění je nutné mít na počítači nainstalované její běhové prostředí v minimální verzi 1.7. Pokud toto běhové prostředí na počítači nainstalované nemáme, je nutné ho stáhnout z webové adresy http://www.oracle.com/technetwork/java/javase/downloads/index.html
a nainstalovat jej.

Další nutnou podmínkou pro úspěšné spuštění programu je mít ve složce se souborem `visualization.jar` umístěnou složku `resources` se soubory programu: `log4j2.xml`, `priorities.properties`, `visualization_cs.properties`, `visualization_icon.png`, `visualization_main_window.xml`, `visualization_open_csv_dialog.xml`, `visualization_open_database_dialog.xml`. Bez těchto souborů nemusí program fungovat správně.

Spuštění programu je poté velice jednoduchou záležitostí. Stačí v souborovém manažeru otevřít složku, ve které je program umístěn a spustit buď přímo soubor `visualization.jar` dvojím kliknutím, nebo spustit soubor `visualization.jar` přes příkazový řádek zadáním příkazu „`java -jar visualization.jar`“.

Po úspěšném spuštění uvidíme hlavní okno programu, viz obrázek 1.

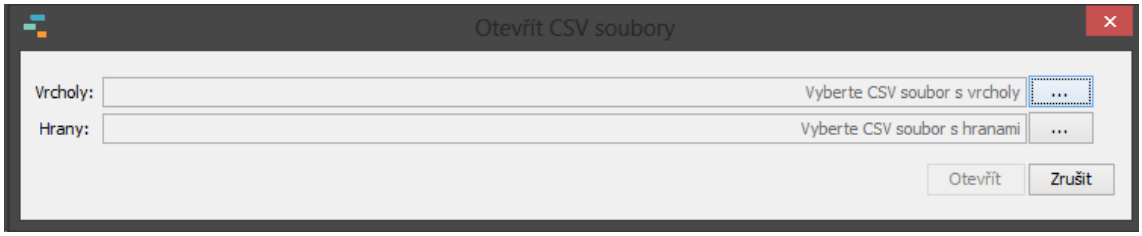


Obrázek 1: Hlavní okno programu

Práce s programem

Načtení CSV souborů

Načtení CSV souborů (`Soubor – Otevřít CSV soubory`, `CTRL+O`) probíhá přes dialogové okno (viz obrázek 2), kde uživatel specifikuje dva CSV soubory: soubor s událostmi (vrcholy) a soubor se závislostmi (hranami).



Obrázek 2: Načtení CSV souborů

Formát CSV souborů

Soubory CSV mají specifický formát, který musí být dodržen, jinak dojde při načítání souborů k chybě. Jako oddělovač se v souborech používá znak roury '|'. CSV soubory by měly být kódované ve formátu UTF-8, aby bylo zaručeno správně čtení a zobrazení českých znaků. Zároveň nesmí soubory obsahovat takzvaný BOM (Byte Order Mark) znak na začátku souboru, jinak dojde k chybě při čtení souboru. Příklad obsahu souborů viz ukázka 1 a ukázka 2.

Formát CSV souboru s vrcholy/událostmi:

- `id|stereotype|nazev|popis|datum_zacatku|datum_konce`

Popis jednotlivých parametrů (tučné parametry povinné):

- **`id`** – jedinečný celočíselný identifikátor vrcholu,
- **`stereotype`** – typ vrcholu, možnosti (case-insensitive): `event`, `person`, `place` a `item`,
- **`nazev`** – název události,
- `popis` – popis události,
- `datum_zacatku` – datum začátku události ve formátu `yyyy-MM-dd` (pro roky před naším letopočtem `-yyyy-MM-dd`) a
- `datum_konce` – datum začátku události ve formátu `yyyy-MM-dd` (pro roky před naším letopočtem `-yyyy-MM-dd`).

Formát CSV souboru s hranami/závislostmi:

- `id|stereotype|nazev|pocatecni_vrchol|koncovy_vrchol`

Popis jednotlivých parametrů (tučné parametry povinné):

- **`id`** – jedinečný celočíselný identifikátor hrany,
- **`stereotype`** – typ hrany, možnosti (case-insensitive): `relationship`, `interaction`, `participation`, `creation`, `cause`, `part_of` a `takes_place`,
- `nazev` – název závislosti,
- **`pocatecni_vrchol`** – číselný identifikátor počátečního vrcholu a
- **`koncovy_vrchol`** – číselný identifikátor koncového vrcholu.

```
1|PERSON|Karel IV.|král, císař, markrabě a hrabě|1316-05-14|1378-11-29
2|PERSON|Jan Lucemburský|král a hrabě|1296-08-10|1346-09-26
3|PERSON|Eliška Přemyslovna|královna|1292-01-20|1330-09-28
```

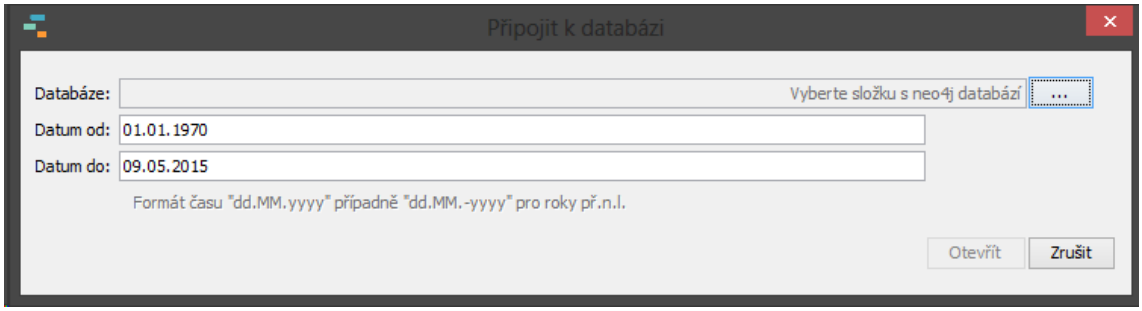
Ukázka 1: CSV soubor s definicí událostí/vrcholů

```
1001|RELATIONSHIP|potomek|2|1
1002|RELATIONSHIP|potomek|3|1
1003|RELATIONSHIP|manželé|2|3
```

Ukázka 2: CSV soubor s definicí vztahů/hran

Připojení k databázi

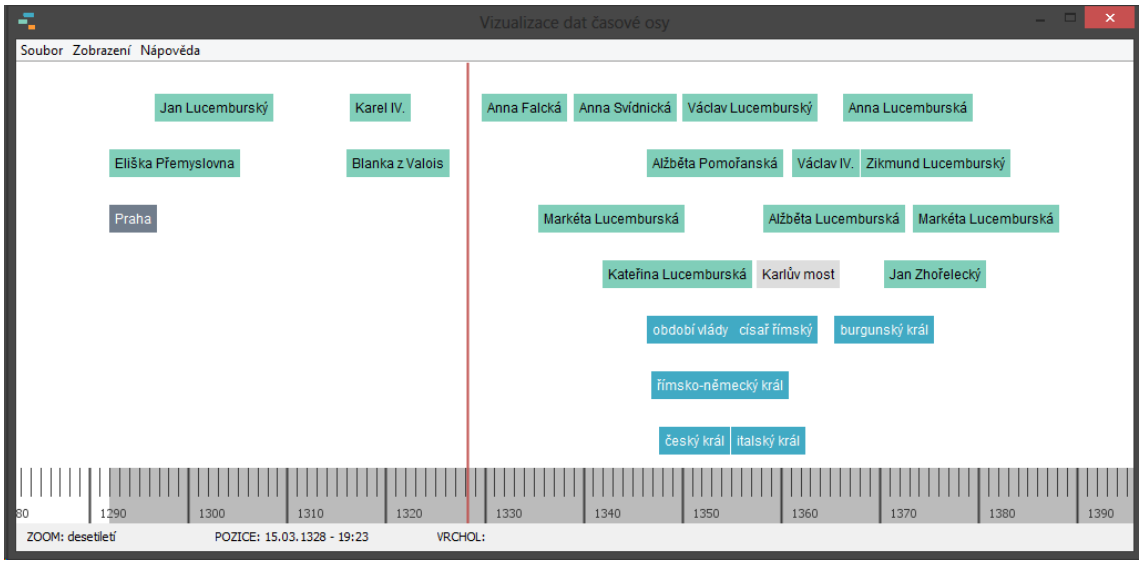
Připojení k databázi (`Soubor - Připojit k databázi`, `CTRL+D`) probíhá přes dialogové okno (viz obrázek 3), kde uživatel specifikuje cestu k adresáři s databází a specifikuje časové rozmezí, podle kterého se vyberou data z databáze.



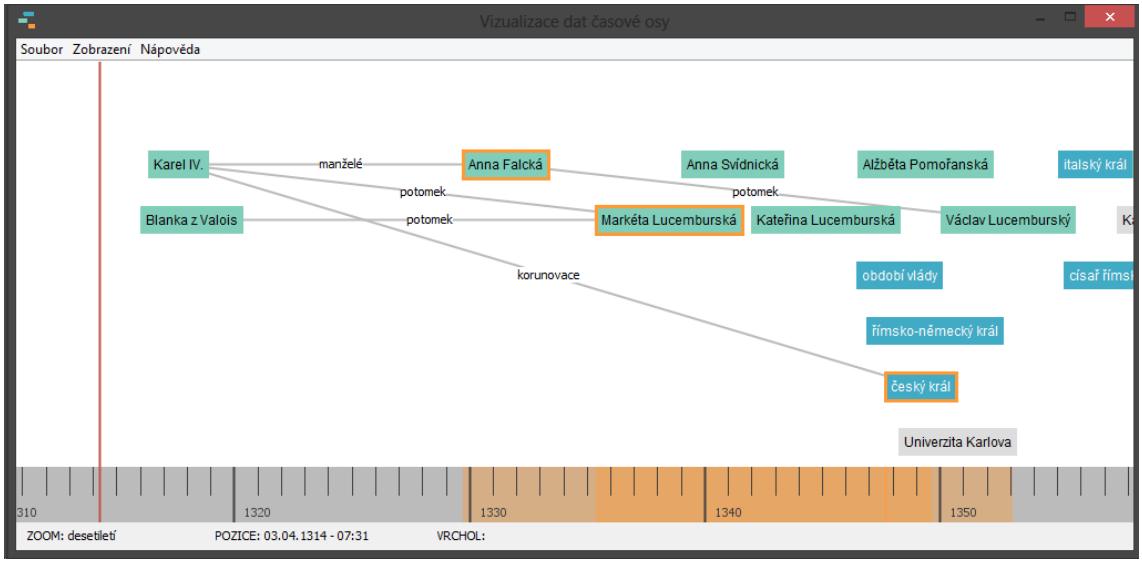
Obrázek 3: Připojení k databázi

Ovládání a popis programu

Po otevření CSV souborů nebo připojení k databázi se v hlavní části vykreslí události, jak ukazuje obrázek 4. V časové ose se lze pohybovat táhnutím myši, kterou následuje vertikální indikátor. Přibližovat a oddalovat lze otáčením kolečka myši společně se stisknutou klávesou `CTRL` (pomalý zoom) nebo klávesou `SHIFT` (rychlý zoom). Výběr události lze provést levým kliknutím na požadovanou událost, po výběru se tato událost zvýrazní a zároveň se na časové ose zobrazí doba trvání, viz obrázek 5.

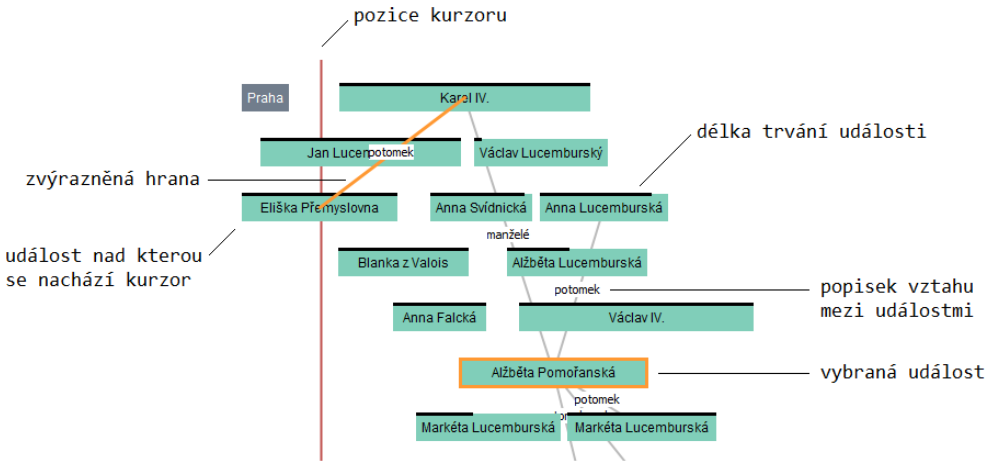


Obrázek 4: Vizualizace událostí; téma: Karel IV.

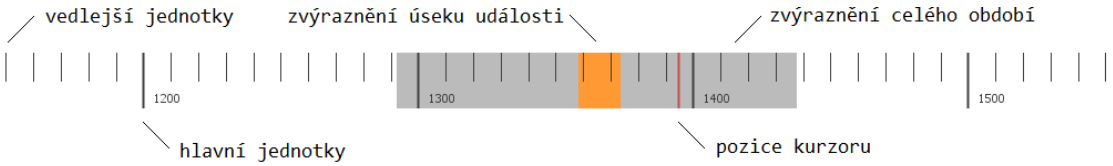


Obrázek 5: Výběr a zvýraznění událostí

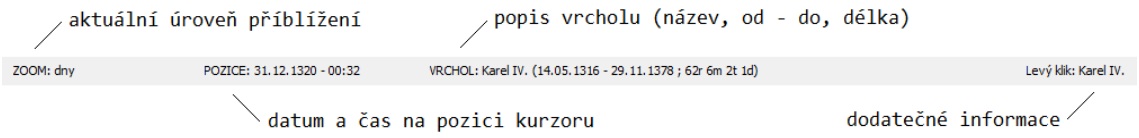
Jednotlivé komponenty grafického uživatelského rozhraní a jejich části jsou popsané na následujících obrázcích: obrázek 6, obrázek 7 a obrázek 8. U informačního panelu a položky „Vrchol“ jsou zobrazené jen ty informace, které jsou v události uloženy. Pokud například nemá událost informaci o svém konci, nebude tato informace zobrazena a zároveň nebude vypsána ani délka trvání události.



Obrázek 6: Graf událostí a jeho části



Obrázek 7: Časová osa a její části



Obrázek 8: Informační panel a jeho části

Možnosti interakce a zobrazení

Interakce s časovou osou a grafem událostí:

- přejetí kurzoru myši nad událostí:
  - zvýraznění hran vedoucích z/do vrcholu,
  - zvýraznění délky trvání události na časové ose,
  - textový výpis informací o události v informačním panelu,
- výběr události levým kliknutím:
  - zvýraznění události,
  - vykreslování hran vedoucích z/do vrcholu a
  - zvýraznění délky trvání události na časové ose a průniků vybraných událostí.

Zobrazení (menu – Zobrazení):

- `Resetovat zobrazení` – zobrazení celého časového úseku
- `Vybrat všechny události` – vybrání všech událostí v grafu
- `Zrušit výběr` – zrušení všech vybraných událostí
- `Šířka událostí podle délky trvání` – zapnutí/vypnutí vykreslování událostí podle jejich délky trvání
- `Zobrazit všechny hrany` – zapnutí/vypnutí vykreslování všech hran grafu
- `Zobrazit informace o hraně` – zapnutí/vypnutí vykreslování popisu hran
- `Zobrazit zvýrazněné hrany na vrchu` – zapnutí/vypnutí vykreslování zvýrazněných hran na vrchu nad všemi ostatními objekty

Změna parametrů konfiguračních souborů

Uživateli je umožněno měnit parametry konfiguračních souborů, které jsou umístěné ve složce `resources` a přizpůsobit tak aplikaci svým potřebám. Změna parametrů je možná pouze, pokud je program vypnutý. V případě úpravy jakékoli konfigurace je nutné dbát na správný formát hodnot, které uživatel mění. Jednotlivé parametry v konfiguračních souborech jsou pojmenovány anglickými názvy, které vyjadřují jejich význam.

Seznam konfiguračních souborů:

- `log4j2.xml` (ukázka 3) – konfigurace logování,
- `priorities.properties` (ukázka 4) – priority pro výpočet pagerank algoritmu,
- `visualization_cs.properties` (ukázka 5) – česká lokalizace programu,
- `visualization_main_window.xml` (ukázka 6) – definice hlavního okna a jeho komponent,
- `visualization_open_csv_dialog.xml` (ukázka 7) – dialog otevření CSV souborů (doporučeno neměnit tento soubor!) a
- `visualization_open_database_dialog.xml` (ukázka 8) – dialog připojení k databázi (doporučeno neměnit tento soubor!).

Změna úrovně logování

Defaultně je nastaveno logování do konzole a do soboru. Do konzole se logují všechny zprávy od úrovně `info` výše. Do souboru se logují pouze chybové zprávy `error`. Pokud budeme chtít například do konzole vypisovat zprávy o interakci s programem, musíme změnit úroveň logování `level` na `debug`.

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration status="WARN">
	<Appenders>
		<Console name="Console" target="SYSTEM_OUT">
			<PatternLayout pattern="%d{HH:mm:ss.SSS} [%t] %-5level
			%logger{36} - %msg%n" />
		</Console>
		<File name="File" fileName="visualization.log">
			<PatternLayout pattern="%d{HH:mm:ss.SSS} [%t] %-5level
			%logger{36} - %msg%n" />
		</File>
	</Appenders>
	<Loggers>
		<Root level="info">
			<AppenderRef ref="Console" level="info" />
			<AppenderRef ref="File" level="error" />
		</Root>
	</Loggers>
</Configuration>
```

Ukázka 3: log4j2.xml; konfigurace logování

Změna priorit pro výpočet pagerank algoritmu

Soubor obsahuje pro každý typ vrcholu a hrany její ohodnocení, se kterými pracuje pagerank algoritmus při svém výpočtu (při načtení dat z databáze). Hodnoty priorit musí být kladná celá čísla!

```
#############################################################################
#                          NODE and BOND priorities
#############################################################################

#
# Default value
#
DEFAULT = 5

#
# NODE
#
EVENT   = 10
PERSON = 8
PLACE   = 5
ITEM    = 3

#
# BONDS
#
RELATIONSHIP          = 1
INTERACTION           = 1
PARTICIPATION         = 10
CREATION              = 1
CAUSE                 = 5
PART_OF               = 10
TAKES_PLACE           = 1
```

Ukázka 4: priorities.properties; priority pro výpočet pagerank algoritmu

Úprava lokalizace

Při úpravě lokalizace celé aplikace, její části nebo přejmenování nějakého z popisků upravíme odpovídající řetězec podle našeho přání. Pozor na české znaky, nemusí se v aplikaci zobrazovat správně, pokud budou v souboru zapsané klasicky! Pro úpravu speciálních znaků použijte native2ascii.

```
################################################################################
#                          Localization
################################################################################

app_Title = Vizualizace dat \u010Dasové osy

#
# ZOOM levels
#
HOURS                = hodiny
DAY                  = dny
WEEK                 = týdny
MONTH                = m\u011Bsíce
YEAR                 = roky
DECADE               = desetiletí
CENTURY              = století
MILLENNIUM           = tisíciletí
MILLENNIUM_DECADE    = desetitisíciletí

#
# Months
#
# short names
month_s_January      = Led
month_s_February     = Únr
month_s_March        = B\u0159e
month_s_April        = Dub
month_s_May          = Kv\u011B
month_s_June         = \u010Cvn
month_s_July         = \u010Cvc
month_s_August       = Srp
month_s_September    = Zá\u0159
month_s_October      = \u0158íj
month_s_November     = Lis
month_s_December     = Pro

#
# Days
#
# short names
day_s_Monday         = Po
day_s_Tuesday        = Út
day_s_Wednesday      = St
day_s_Thursday       = \u010Ct
day_s_Friday         = Pá
day_s_Saturday       = So
day_s_Sunday         = Ne
```

Ukázka 5: visualization_cs.properties; česká lokalizace programu (pouze část)

Změna parametrů aplikace

V konfiguračním souboru aplikace lze měnit její vzhled, rozložení a další parametry.

Změnu barvy vrcholů (platí i pro ostatní barvy) provedeme úpravou hodnoty u požadovaného parametru. Například při změně barvy události osoba změníme hodnotu parametru `nodePersonColor` u komponenty `graphView`. Pozor, barvy musí být zadané v hexadecimálním formátu „`#XXXXXX`“, kde `X` jdou hexadecimální hodnoty červené, zelené a modré barvy!

Zapnutí výpisu FPS provedeme změnou hodnoty parametru `showFPS` na `true`.

```
<?xml version="1.0" encoding="UTF-8"?>
<frame id="frame" name="mainframe" size="1024,500" title="app_Title" IconImage="app_Image"
bundle="visualization" locale="cs" plaf="com.sun.java.swing.plaf.windows.WindowsLookAndFeel"
defaultCloseOperation="JFrame.EXIT_ON_CLOSE" extendedState="JFrame.MAXIMIZED_BOTH">

        <menubar name="menubar">
                <menu text="mus_File">
                        <menuitem name="mi_openCSV" Text="mis_OpenCSV" action="openCSVAction"
                        accelerator="acc_OpenCSV" mnemonic="mn_OpenCSV" />
                        <menuitem name="mi_connectToDatabase" Text="mis_ConnectToDatabase"
                        action="connectToDatabaseAction" accelerator="acc_OpenDatabase"
                        mnemonic="mn_OpenDatabase" />
                        <menuitem name="mi_exit" Text="mis_Exit" action="exitAction"
                        accelerator="acc_Exit" mnemonic="mn_Exit" />
                </menu>
                <menu text="mus_View">
                        <menuitem name="mi_ResetView" Text="mis_ResetView" action="resetView" />
                        <menuitem name="mi_SelectAll" Text="mis_SelectAll" action="selectAll" />
                        <menuitem name="mi_ClearSelection" Text="mis_ClearSelection"
                        action="clearSelection" />
                        <separator />
                        <checkbox name="mi_VerticesDuration" Text="mis_VerticesDuration"
                        action="verticesDuration" />
                        <checkbox name="mi_ShowAllEdges" Text="mis_ShowAllEdges"
                        action="showAllEdges" />
                        <checkbox name="mi_DrawEdgeText" Text="mis_DrawEdgeText"
                        action="drawEdgeText" />
                        <checkbox name="mi_DrawHighlightedEdgesOnTop"
                        Text="mis_DrawHighlightedEdgesOnTop" action="drawHighlightedEdgesOnTop" />
                </menu>
                <menu text="mus_Help">
                        <menuitem name="mi_help" text="mis_Help" action="helpAction"
                        accelerator="acc_Help" mnemonic="mn_Help" />
                        <menuitem name="mi_about" text="mis_About" action="aboutAction" />
                </menu>
        </menubar>

        <panel layout="BorderLayout">
                <graphView id="graphView" constraints="BorderLayout.CENTER" showFPS="false"
                backgroundColor="#FFFFFF" mousePointerColor="#C0504D" edgeColor="#BBBBBB"
                highlightColor="#FF9933" nodeEventColor="#41AAC4" nodeItemColor="#DDDDDD"
                nodePersonColor="#80CEB9" nodePlaceColor="#717D8C" nodeDefaultColor="#FFFFFF"
                vertexOffsetY="25" />
                <timeline id="timeline" constraints="BorderLayout.SOUTH" height="50"
                backgroundColor="#FFFFFF" foregroundColor="#404040" mousePointerColor="#C0504D"
                periodColor="#BBBBBB" highlightColor="#FF9933" />
        </panel>

        <statusBar id="statusBar" constraints="BorderLayout.SOUTH" />
</frame>
```

Ukázka 6: visualization_main_window.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<dialog title="mis_OpenCSV" IconImage="app_Image" size="750,150"
defaultCloseOperation="JDialog.DISPOSE_ON_CLOSE" bundle="visualization" locale="cs"
layout="GridBagLayout">
        <label text="csv_vertices">
                <gridbagconstraints id="label" gridx="0" gridy="0" insets="5,10,5,5" />
        </label>
        <label text="csv_edges">
                <gridbagconstraints use="label" gridy="1" />
        </label>
        <textfield id="vertices" text="csv_choose_vertices_file" horizontalAlignment="4" enabled="false"
        editable="false">
                <gridbagconstraints id="textField" gridx="1" gridy="0"
                fill="GridBagConstraints.HORIZONTAL" weightx="1.0" />
        </textfield>
        <textfield id="edges" text="csv_choose_edges_file" horizontalAlignment="4" enabled="false"
        editable="false">
                <gridbagconstraints use="textField" gridy="1" />
        </textfield>
        <button text="browse_file" action="browseVertices">
                <gridbagconstraints id="browseButton" gridx="2" gridy="0"
                anchor="GridBagConstraints.LINE_START" />
        </button>
        <button text="browse_file" action="browseEdges">
                <gridbagconstraints use="browseButton" gridy="1" />
        </button>
        <button id="openButton" text="open" action="open" enabled="false">
                <gridbagconstraints id="bottomButton" gridx="1" gridy="2"  insets="10,0,0,0"
                anchor="GridBagConstraints.LINE_END" />
        </button>
        <button text="cancel" action="cancel">
                <gridbagconstraints use="bottomButton" gridx="2" gridy="2"
                anchor="GridBagConstraints.LINE_START" insets="10,0,0,10"/>
        </button>
</dialog>
```

Ukázka 7: visualization_open_csv_dialog.xml; dialog otevření CSV souborů

```
<?xml version="1.0" encoding="UTF-8"?>
<dialog title="mis_ConnectToDatabase" IconImage="app_Image" size="750,200"
defaultCloseOperation="JDialog.DISPOSE_ON_CLOSE" bundle="visualization" locale="cs"
layout="GridBagLayout">
        <label text="database">
                <gridbagconstraints id="label" gridx="0" gridy="0" insets="5,10,5,5" />
        </label>
        <label text="database_date_from">
                <gridbagconstraints use="label" gridy="1" />
        </label>
        <label text="database_date_to">
                <gridbagconstraints use="label" gridy="2" />
        </label>
        <label text="database_date_format_description" enabled="false">
                <gridbagconstraints use="label" gridx="1" gridy="3" gridwidth="3"
                fill="GridBagConstraints.HORIZONTAL" weightx="1.0" />
        </label>
        <textfield id="database" text="database_choose" horizontalAlignment="4" enabled="false"
        editable="false">
                <gridbagconstraints id="textField" gridx="1" gridy="0" gridwidth="2"
                fill="GridBagConstraints.HORIZONTAL" weightx="1.0" />
        </textfield>
        <textfield id="dateFrom" text="" horizontalAlignment="2">
                <gridbagconstraints id="dateField" gridx="1" gridy="1" gridwidth="1"
                fill="GridBagConstraints.HORIZONTAL" weightx="1.0" />
        </textfield>
        <textfield id="dateTo" text="" horizontalAlignment="2">
                <gridbagconstraints use="dateField" gridy="2" />
        </textfield>
        <button text="browse_file" action="browseDatabase">
                <gridbagconstraints id="browseButton" gridx="3" gridy="0"
                anchor="GridBagConstraints.LINE_START" />
        </button>
        <button id="openButton" text="open" action="open" enabled="false">
                <gridbagconstraints id="bottomButton" gridx="2" gridy="4" insets="10,0,0,0"
                anchor="GridBagConstraints.LINE_END" />
        </button>
        <button text="cancel" action="cancel">
                <gridbagconstraints use="bottomButton" gridx="3" anchor="GridBagConstraints.LINE_START"
                insets="10,0,0,10" />
        </button>
</dialog>
```

Ukázka 8: visualization_open_database_dialog.xml; dialog připojení k databázi

Seznam ovládacích prvků a klávesových zkratek

táhnutí myši	posun časové osy
CTRL + otočení kolečka myši	pomalý zoom
SHIFT + otočení kolečka myši	rychlý zoom
levé kliknutí	výběr události
CTRL + O	otevření CSV souborů
CTRL + D	připojení k databázi
CTRL + Q	ukončení programu
F1	nápověda programu

Příloha D – Dotazník

Hodnocení vizualizace časové osy a grafu událostí

Dotazník číslo	
Věk	
Dosažené vzdělání	
Obor vzdělání	

Hodnocení globálně						
Prosím oznámkujte jako ve škole (1 – výborný, 5 nedostatečný)		1	2	3	4	5
1	Ovládání aplikace					
2	Orientace v grafu událostí					
3	Orientace v časové ose					
4	Možnosti zobrazení					
5	Možnosti interakce s časovou osou					
6	Možnosti interakce s grafem událostí					
7	Rozložení GUI					
Připomínky k vizualizaci a aplikaci						

Hodnocení vizualizace tématu „Domažlice“						
Prosím oznámkujte jako ve škole (1 – výborný, 5 nedostatečný)		1	2	3	4	5
1	Přehlednost vizuální reprezentace historických událostí					
Připomínky						

Hodnocení vizualizace tématu „Karel IV.“						
Prosím oznámkujte jako ve škole (1 – výborný, 5 nedostatečný)		1	2	3	4	5
1	Přehlednost vizuální reprezentace historických událostí					
Připomínky						

Další návrhy a připomínky

Příloha E – Vyplněné dotazníky

Hodnocení vizualizace časové osy a grafu událostí

Dotazník číslo	1
Věk	26
Dosažené vzdělání	Vysokoškolské - Magisterské
Obor vzdělání	Informační technologie

Hodnocení globálně						
Prosím oznámkujte jako ve škole (1 – výborný, 5 nedostatečný)		1	2	3	4	5
1	Ovládání aplikace		X			
2	Orientace v grafu událostí	X				
3	Orientace v časové ose	X				
4	Možnosti zobrazení	X				
5	Možnosti interakce s časovou osou	X				
6	Možnosti interakce s grafem událostí	X				
7	Rozložení GUI		X			
Připomínky k vizualizaci a aplikaci						

Hodnocení vizualizace tématu „Domažlice“						
Prosím oznámkujte jako ve škole (1 – výborný, 5 nedostatečný)		1	2	3	4	5
1	Přehlednost vizuální reprezentace historických událostí	X				
Připomínky						

Hodnocení vizualizace tématu „Karel IV.“						
Prosím oznámkujte jako ve škole (1 – výborný, 5 nedostatečný)		1	2	3	4	5
1	Přehlednost vizuální reprezentace historických událostí		X			
Připomínky						

Další návrhy a připomínky
• Udělat samostatnou položku pro výběr checkboxů na záložce Zobrazení • Posun pomocí stlačení levého tlačítka myši • Přidat nedávno otevřené soubory, aby je uživatel nemusel po každém spuštění vybírat znovu • Není vidět délka trvání, pokud je událost vybraná • Pokud je nainstalován Bing Toolbar, nelze aplikaci ukončit zkratkou CTRL+Q

Hodnocení vizualizace časové osy a grafu událostí

Dotazník číslo	2
Věk	25
Dosažené vzdělání	Maturita
Obor vzdělání	IT

Hodnocení globálně						
Prosím oznámkujte jako ve škole (1 – výborný, 5 nedostatečný)		1	2	3	4	5
1	Ovládání aplikace		x			
2	Orientace v grafu událostí		x			
3	Orientace v časové ose	x				
4	Možnosti zobrazení	x				
5	Možnosti interakce s časovou osou	x				
6	Možnosti interakce s grafem událostí		x			
7	Rozložení GUI		x			
Připomínky k vizualizaci a aplikaci						

Hodnocení vizualizace tématu „Domažlice“						
Prosím oznámkujte jako ve škole (1 – výborný, 5 nedostatečný)		1	2	3	4	5
1	Přehlednost vizuální reprezentace historických událostí		x			
Připomínky						

Hodnocení vizualizace tématu „Karel IV.“						
Prosím oznámkujte jako ve škole (1 – výborný, 5 nedostatečný)		1	2	3	4	5
1	Přehlednost vizuální reprezentace historických událostí		x			
Připomínky						

Další návrhy a připomínky

Hodnocení vizualizace časové osy a grafu událostí

Dotazník číslo	3
Věk	26
Dosažené vzdělání	Bc
Obor vzdělání	Software development

Hodnocení globálně						
Prosím oznámkujte jako ve škole (1 – výborný, 5 nedostatečný)		1	2	3	4	5
1	Ovládání aplikace	x				
2	Orientace v grafu událostí		x			
3	Orientace v časové ose	x				
4	Možnosti zobrazení		x			
5	Možnosti interakce s časovou osou	x				
6	Možnosti interakce s grafem událostí			x		
7	Rozložení GUI		x			
Připomínky k vizualizaci a aplikaci						

Hodnocení vizualizace tématu „Domažlice“						
Prosím oznámkujte jako ve škole (1 – výborný, 5 nedostatečný)		1	2	3	4	5
1	Přehlednost vizuální reprezentace historických událostí		x			
Připomínky						

Hodnocení vizualizace tématu „Karel IV.“						
Prosím oznámkujte jako ve škole (1 – výborný, 5 nedostatečný)		1	2	3	4	5
1	Přehlednost vizuální reprezentace historických událostí	x				
Připomínky						

Další návrhy a připomínky