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

Bakalářská práce

Implementace testovacího nástroje pro časovou osu

Plzeň 2019 Michal Fiala

Místo této strany bude zadání práce.

Prohlášení

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

V Plzni dne 2. dubna 2019

Michal Fiala

Abstract

The text of the abstract (in English). It contains the English translation of the thesis title and a short description of the thesis.

Abstrakt

Text abstraktu (česky). Obsahuje krátkou anotaci (cca 10 řádek) v češtině. Budete ji potřebovat i při vyplňování údajů o bakalářské práci ve STAGu. Český i anglický abstrakt by měly být na stejné stránce a měly by si obsahem co možná nejvíce odpovídat (samozřejmě není možný doslovný překlad!).

Obsah

1 Úvod **3**

2 Úvod do vizualizace **4**
2.1 Vizualizace 4
2.2 Časová osa 4
2.3 Řešení zobrazení časové osy 5
2.3.1 Implementace 5
2.3.2 Uživatelské rozhraní 5
2.3.3 Pás (band) 6
2.3.4 Položka pásu (band item) 6
2.3.5 Renderer 6
2.3.6 Základní interakce 7

3 Analýza možných řešení **8**
3.1 Lifeline 8
3.2 Perspektivní zeď 9
3.3 Komponenta na práci se slovy 9
3.4 Časově nezávislé skládání 10
3.5 Filtrovací strom 11
3.6 Redukce vizuálního šumu (Visual clutter reduction) 11
3.6.1 Nahrazení hran 12
3.6.2 Shlukování uzlů 12

4 Analýza požadavků **14**
4.1 Technické požadavky 14
4.1.1 Formát dat 14
4.1.2 Změna určování rozdělení 14
4.2 Funkční požadavky 14
4.2.1 Renderery pro zobrazení pod entit 14
4.3 Testovací nástroj 15

5 Realizace **16**
5.1 Formát dat 16
5.1.1 Změna datového formátu 16
5.1.2 Zpracování nového datového formátu 18
5.1.3 Základy pro vizualizaci 18

5.2 Změna určování rozdělení 19
5.3 Nové renderery 21
5.3.1 Počítání pozice *SubItem* 21
5.3.2 Implementace rendererů 22
5.4 Testovací nástroj 23

6 Diskuze **24**

7 Závěr **25**

1 Úvod

V dnešní době se můžeme setkat s vizualizací dat téměř každodenně a to například v práci, kde vytváříme grafy znázorňující počty vyrobených výrobků nebo na internetu, kde hledáme nějakou informaci, či vytváříme rozvrh na další týden. Už jen vytváření rozvrhu lze považovat za znázorňování informace v čase. Z těchto vizualizací získáváme data, se kterými dále pracujeme.

Obvykle však pracujeme se základní vizualizací, jakou jsou například tabulky nebo jednoduché grafy. Problém nastává tehdy, když chceme zobrazit vztahy mezi daty. Dokázat znázornit a získat data například ze spojitosti mezi uživateli sociální sítě v časové ose tak, abychom mohli lehce zjistit, kdo s kým má jaký vztah, je na rozsáhlé množině nelehký úkol. Je to z toho důvodu, že tak velké množství dat nedokáže náš mozek zpracovat najednou. Avšak analýza takových a jimi podobných dat je velmi užitečná, například při tvorbě politických průzkumů, v marketingu a dalších odvětvích.

Zobrazování rozsáhlých, na sobě závisejících dat naivním způsobem ve většině případů nevedou k čitelnému a analyzovatelnému řešení. Tato bakalářská práce bude navazovat na vytvořený nástroj pro zobrazování dat v časové ose viz [1] a dále se zaměří na to, jak efektivně v tomto nástroji vizualizovat historická data, aby z nich bylo patrné, jaký je mezi nimi vztah a v jakém časovém sledu se události nacházely. Tato práce se zaměří i na rozšíření uživatelského prostředí tohoto nástroje, které bude pomáhat uživateli práci s daty.

Výslednou práci nakonec otestuji oproti předchozí verzi, která je již funkční a kterou budu vylepšovat o efektivnost zobrazení. Testování proběhne na vybrané množině lidí a bude se porovnávat rychlost, snadnost a průběh nalezení odpovědi na dané otázky týkající se zobrazené datové množiny.

2 Úvod do vizualizace

Tato bakalářská práce se zaobírá především vizualizací dat v časové ose, aby bylo dále z textu zřejmé, co jaký pojem znamená, je třeba popsat pár základních pojmů.

2.1 Vizualizace

Slovo vizualizace je široce používaný pojem, který vyjadřuje postup, při kterém popisujeme nějaké informace, data nebo hodnoty pomocí obrazu. Vizualizace má dlouhou a úctyhodnou historii. Avšak teprve třicet let uběhlo od doby, kdy se stala vizualizace nezávislou oblastí výzkumu.

Cílem této oblasti výzkumu je integrace vynikajících schopností lidského vizuálního vnímání a enormní výpočetní síly počítačů, které uživatelům pomáhají analyzovat, porozumět a komunikovat s těmito daty. Aby bylo dosaženo tohoto cíle, je třeba splnit následující tři kritéria:

- Vyjádřitelnost
- Efektivita
- Vhodnost

Vyjádřitelnost odkazuje na požadavek přesně zobrazovat informace obsažené v datech. Efektivita především zvažuje, do jaké míry se vizualizace zaměřuje na kognitivní schopnosti lidského mozku. Cílem je získat intuitivně rozpoznatelné a reprezentovatelné data. Vhodnost zahrnuje vyvážení ceny zobrazení vůči rychlosti dosažení úkolu [8].

2.2 Časová osa

Časová osa je množina událostí uspořádaných chronologicky podle jejich časového zasazení. Je to typicky grafický návrh, který zobrazuje dlouhý pruh označený daty vedle sebe a zobrazení samotných událostí nad tímto pruhem. Časové osy lze dělit do několika základních typů [9].

Typy časových os:

- Textové
- Číselné
- Interaktivní, s možností přiblížení

Jako řešení časové osy v přecházející práci [1] byl využit třetí typ.

2.3 Řešení zobrazení časové osy

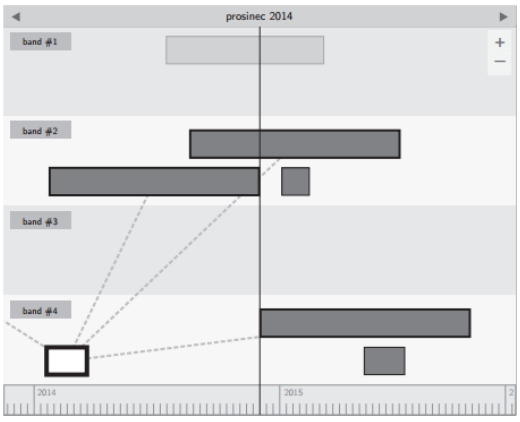
V roce 2015 byl [1] publikován nástroj pro zobrazení časové osy, na který budu touto prací navazovat. V této sekci tento nástroj stručně popíši.

2.3.1 Implementace

Tato časová osa byla implementována pro webové stránky a její využití je tedy nezávislé na platformách. Pro zobrazování slouží elementy HTML a pro práci s těmito elementy jsou použity javascriptové soubory.

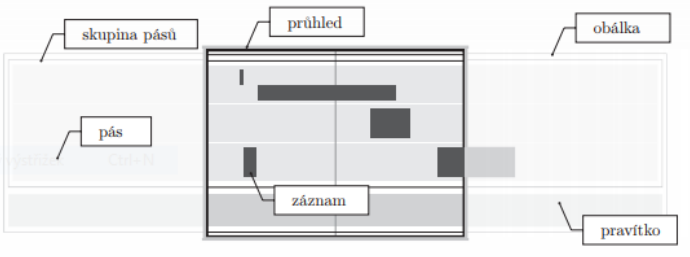
2.3.2 Uživatelské rozhraní

Uživatelské prostředí je složeno z časové osy, která je umístěna dole. A zbytek okna tvoří zobrazení historických událostí viz Obrázek 2.1.



Obrázek 2.1: Uživatelské rozhraní [1].

Obrázek 2.1 zobrazuje pouze část časové osy. Celý kontext můžeme vidět na Obrázku 2.2, kde je tučně zvýrazněno aktuálně zobrazené okno v kontextu s časovou osou.



Obrázek 2.2: Celý kontext okna [1].

2.3.3 Pás (band)

Pás je komponenta, do které se zařazují záznamy podle určitého typu. Každý pás má jeden jediný možný typ dat. Tato komponenta zajišťuje například schování záznamů pokud nejsou v zorném poli časové osy, dále zajišťuje metodu pro řešení kolize záznamů.

2.3.4 Položka pásu (band item)

Položka pásu slouží pro abstrakci reálného záznamu. Tato komponenta uchovává skutečná data a funguje jako jejich reprezentace. Jakým způsobem se tato komponenta vykresluje určuje objekt, který se nazývá *renderer* [1].

2.3.5 Renderer

Tato komponenta určuje jak má položka pásu nebo hrana grafu vypadat. Zajišťuje vytvoření HTML obálky, která je uložena v položce pásu. Dále obsahuje metody pro úpravy rozměrů a pozice této HTML obálky.

2.3.6 Základní interakce

V této části je popis základních interakcí implementované časové osy.

Posun osy

Způsob jakým se uživatel může posouvat po ose je drag&drop. Jde o uchopení stisknutím levého tlačítka myši a následné tažení doleva nebo doprava. Po upuštění časová osa zobrazí data ve zvoleném období.

Změna úrovně přiblížení

Uživatel může časovou osu přibližovat a oddalovat použitím tlačítka + a -. Při této manipulaci s osou nedochází k žádnému posunu, středový čas zůstává zachován. Pokud použije uživatel k přiblížení kolečko myši, dojde k jinému chování. Tehdy je totiž brána v potaz pozice kurzoru a osa se přiblíží k myší určenému místu.

Zaostření

Tato funkce kombinuje výše popsané interakce. Umožňuje totiž přizpůsobit obsah průhledu tak, aby se v něm vybraný záznam zobrazil celý a v co největších rozměrech.

3 Analýza možných řešení

V této části jsou popsány postupy pro změnu zobrazení časové osy tak, aby se zajistila co největší efektivita práce s daty. Z analytické části poté vyberu postupy, které budu implementovat do stávajícího projektu časové osy. Implementované řešení později otestuji na testovací skupině lidí tak, abych zjistil, zda-li mělo řešení nějaké přínosy pro orientaci a vyhledávání v datech.

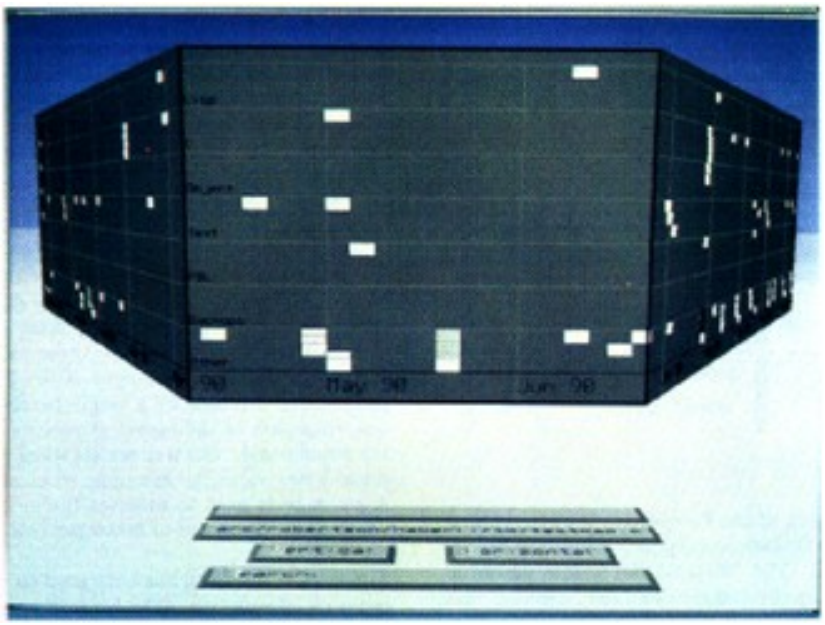
3.1 Lifeline

První ze způsobů zefektivnění zobrazených dat je Lifeline. Postup pro zobrazení dat vypadá následovně. Obrazovka je horizontálně rozdělena do regionů, kde každý region je obarven jinou barvou a reprezentuje určitý typ dat. Data jsou poté zařazena do regionu odpovídajícího typu dat. Určité vizuální podněty jako jsou barvy, tloušťky čar nebo zvýraznění označují význam určitých informací a vztahy mezi událostmi [2].

Tento způsob přináší lepší přehlednost v datech zvýrazněním důležitých dat a lepším filtrováním dat podle regionů.

3.2 Perspektivní zeď

Tento způsob využívá zkreslovacích technik, pro zobrazení velkého množství dat na obrazovce, zajišťující větší efektivitu prohledávání. Tato technika lépe využívá dostupný prostor na obrazovce integrací detailního a kontextového okna. Řešením je zkreslit zobrazení tak, že data mimo hlavní zorné pole jsou zmenšena a zkreslena. Tato technika se jinak nazývá "Rybí oko"[2].



Obrázek 3.1: Ukázka perspektivní zdi [4].

3.3 Komponenta na práci se slovy

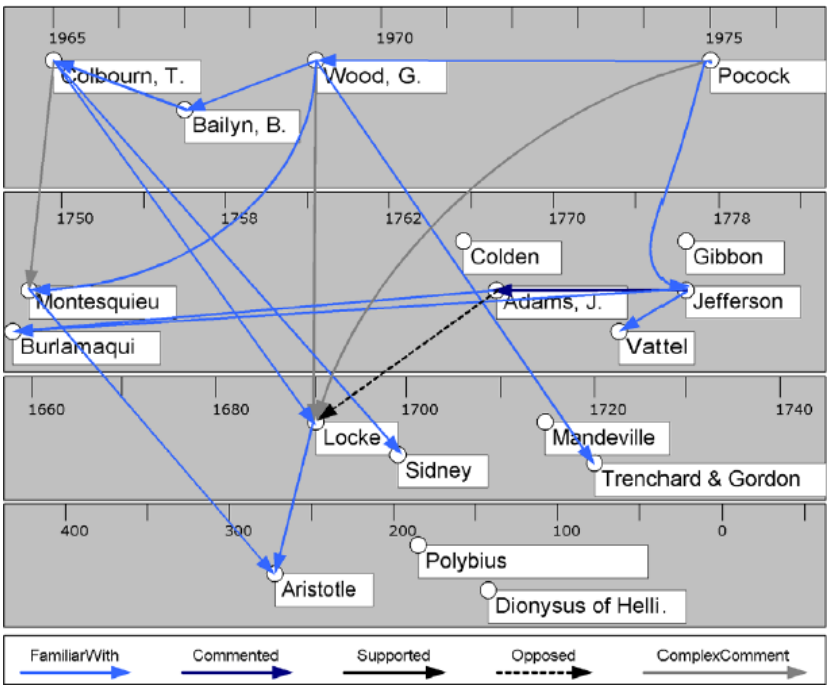
Je mnoho způsobů jak interagovat s časovou osou. V této sekci jsou popsány způsoby jak pracovat s vyhledáváním podle slov.

Toto vyhledávání je zaměřeno na nalezení slova nebo četnosti nejblíže se podobajících slov v textu zobrazených událostí. Toto umožní uživateli hledat v textu podle slova tak, že pokud je slovo nalezeno, označí se všechny události obsahující toto slovo a okno se zaměří na místo, kde je první nalezená událost. Toto zaměření zahrnuje posun osy na tuto instanci tak, aby byla instance uprostřed. Tento přesun by měl být inicializován animací, aby se uživatel orientoval v přesunu. Pokud není zadáno slovo, vypíšeme seznam nejbližších slov nalezených v textu seřazených sestupně podle četnosti slov a po kliknutí na jedno z těchto slov opakujeme akci jako u nalezení slova.

Není-li zadané žádné slovo, vypíše se seznam nejfrekventovanější slov v textu [3].

3.4 Časově nezávislé skládání

Jako například Lifeline dokáže rozdělit zobrazená data podle jejich typů, tak tento způsob umožňuje rozdělit obrazovku na více nezávislých časových os. Čas na těchto osách může být nastaven nezávisle na ostatních. Tento způsob pomáhá zobrazovat vztah mezi dlouho vzdálenými instancemi viz Obrázek 3.2.



Obrázek 3.2: Časově nezávislé skládání [4].

Každý prvek může být považován za pod-časovou osu a takováto osa může být zabalena nebo rozbalena. Pokud je osa zabalená, ostatní prvky odkazující na prvky shluknuté v pod-ose odkazují na tuto zabalenou osu, aby byla zřejmá spojitost mezi prvky.

Shlukováním prvků do pod-os a jejich zobrazováním jen pokud je to nutné, lze zajistit zobrazení velkého množství dat a schopnosti rozeznat spojitost mezi těmito daty.[4]

3.5 Filtrovací strom

Tento způsob zahrnuje přidání bloku obsahující různé aspekty, které mohou být přidány jako filtry. Dále filtrační panel, který zobrazuje aktuální strom filtrování viz levý panel Obrázek 3.3.

Obrázek 3.3: Filtrační strom [5].

Filtrační strom reprezentuje skupinu dotazů, které jsou využity pro zobrazení událostí nad množinou. Každá cesta od kořene k listu je jeden dotaz. To znamená, že je formován cestou do listu, zatímco se provádí filtrování podle aktuálního vrcholu.

Rozhraní začíná s prázdným filtračním stromem a zobrazením celé množiny časové řady. Uživatel může přidat aspekt pomocí výběru z boxu aspektů a poté je časová osa rozdělena do více časových os s položkami, které odpovídají atributu aspektu. Dále má uživatel možnost přidat podtyp aspektu, následně je osa znovu rozdělena podle aspektů a filtrována.[5]

3.6 Redukce vizuálního šumu (Visual clutter reduction)

S vizuálním šumem (visual clutter) se setkáme na každém kroku našeho života, je ho třeba uvážit při vyvíjení uživatelských rozhraních a vizualizaci informací. Abychom mohli uvést metody pro redukci vizuálního šumu, je třeba si říci co tento pojem znamená, můžeme si ho definovat následovně:

Definice: Šum je stav, ve kterém přebytečné položky, nebo jejich reprezentace a nebo organizace, vede k degradaci výkonu u některých úkolů [6].

Spojení mezi vizuálním šumem a reprezentací nebo organizací informací v zobrazení bylo zaznamenáno mnoha designéry a výzkumníky. Pro návrhy webových stránek je doporučováno zaměřit se na organizování textové a grafické informace, abychom zredukovali vizuální šum. Pro zobecněné mapy je třeba se zaměřit na změnu reprezentace informací při změně měřítka mapy, aby se snížil šum a zvýšila použitelnost [6].

Především u velkých grafů se můžeme setkat s pojmem vizuální šum, abychom tento problém vyřešili uvedu zde metody, které slouží k redukci vizuálního šumu.

3.6.1 Nahrazení hran

Touto metodou se snažíme nahradit hrany grafu spline křivkami (spojitá množina souřadnic bodů) a tím zredukovat počet protínání hran, jelikož je považováno za hlavní příčinu vizuálního šumu. Tento způsob lze především použít u grafů, které mají předdefinované souřadnice uzlů například geografické pozice. Avšak minimalizování počtu protínání hran není dobrý přístup a to z toho důvodu, že minimalizování počtu protnutí je NP-úplný problém, a zároveň tento způsob potřebuje hodně času na nalezení řešení, proto je velmi nevhodné použít tento způsob pro interaktivní grafy.

Další způsob jak zmírnit protínání hran je vykreslovat protnutí konfluentně (stékajíc se). Tento způsob přímo neřeší zredukování počtu protnutí hran, ale vykresluje hrany jako křivky, které přecházejí hladce do místa protnutí, a udělá je více přirozené pro pozorovatele viz Obrázek 3.4 [7].

Obrázek 3.4: Ukázka konfluentního vykreslení [7]

3.6.2 Shlukování uzlů

Shlukování uzlů (reprezentace dat) je známo pod mnoha jmény jako například seskupování, klasifikace a rozpoznávání vzorů. S vizuálním šumem znamená tento pojem, rozdělení celé struktury do několika pod-grafů a tyto

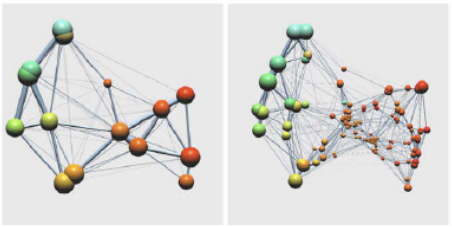
pod-grafy poté vykreslovat jako jednotlivé uzly grafu. Tímto shlukováním podobných a nebo odlišných uzlů uvolníme velké množství prostoru a tím snížíme vizuální šum [7].

Pro shlukování uzlů existují různé způsoby. Hlavní rozdíl mezi nimi je definice vzdálenosti a nebo podobnosti mezi dvěma položkami v datech. O podobnosti dat existují dva základní přístupy [7].

První z přístupů je založen na obsahu a využívá sémantiku (význam) dat pro shlukování uzlů. Ačkoli tento přístup může produkovat velmi smysluplné shlukování, není tolik prostudován, protože je velmi závislý na aplikaci a nepřináší obecné řešení [7].

Druhý z přístupů je založen na struktuře, tento přístup je více obecný a používá pouze informaci o struktuře (relační informace například spojení a nebo hierarchické). Ve shlucích založených na struktuře se dobře definované shluky obvykle vztahují ke komponentám grafu, které mají více spojení uvnitř než-li k vnějším uzlům. Různé techniky můžou docílit shlukování podle struktury a mohou být rozděleny do tří metodik [7]:

- **Grafově teoretické:** Grafově teoretické algoritmy se opírají o podobnostní matici reprezentující podobnost mezi jednotlivými uzly. Shluky jsou tvořeny úzce souvisejícími uzly.
- **Jedno průchodové:** Jedno průchodové algoritmy dokáží vytvářet shluky jejich růstem od jednotlivých datových bodů zvané shluková semínka. Nejjednodušší způsob funguje tak, že se vybere startovní uzel a pak ostatní uzly jeden po druhém přidává do shluku, dokud není shluk dosti velký.
- **Iterativní algoritmy:** Iterativní algoritmy mohou používat generované shluky jinými shlukovými algoritmy. Při realizaci této metody existují tři hlavní kroky a to zhrubnutí, dělení a promítání grafu.



Obrázek 3.5: Dvě různé metody shlukování pro stejný graf [7].

4 Analýza požadavků

V této části je popsán seznam požadavků pouze na základě konzultací s vedoucím práce a týmu spolupracujícím na tomto projektu. Každý uvedený požadavek níže byl zrealizován v rámci této práce.

4.1 Technické požadavky

4.1.1 Formát dat

Základním požadavkem je změna aktuálního formátu dat tak, aby každý prvek dat mohl obsahovat podmnožinu prvků, každý z těchto prvků má mít podobné chování jako původní entita, tzn. zobrazení momentu nebo nějakého trvání. S touto podmnožinou bude umět aplikace pracovat a náležitě jí zobrazit.

Dále, aby byla možnost ke každému prvku přidat třídy kaskádových stylů, které se projeví při zobrazení v aplikaci. Tyto třídy bude možné definovat v souboru speciálně pro tyto třídy.

4.1.2 Změna určování rozdělení

V původní aplikaci jsou pruhy definovány přímo v kódu a ne každý uživatel by byl schopen je měnit. Proto dalším požadavkem je přesunutí definice pruhů do samostatného souboru, kde uživatel bude moci určit jak se jaký typ dat bude zobrazovat. Tyto pruhy pak mohou obsahovat více různých datových typů to znamená, že pruhy půjde sloučit dohromady.

4.2 Funkční požadavky

4.2.1 Renderery pro zobrazení pod entit

Nejdůležitějším požadavkem na práci je vytvoření nových rendererů - objekty, které rozhodnou jak mají být data vyobrazena uživateli.

První je renderer, který bude umět zobrazovat podmnožinu prvků entity a to tak, aby uměl zobrazit jak moment tak dobu nějakého trvání. Další vlastností rendereru bude, že při určitém zobrazení bude umět podmnožinu schovat a nebo naopak zviditelnit.

Druhý je renderer, který bude umět taktéž zobrazovat podmnožinu prvků entity, ale v jiném formátu. Prvek typu moment se zobrazí jako kruh a prvek typu doba trvání se bude zobrazovat jako obdélník. Dále bude vykreslovat spojovací čáru mezi vykreslenými prvky.

4.3 Testovací nástroj

Poslední z požadavků je požadavek na testovací nástroj. Tento nástroj bude sloužit pro otestování zda-li došlo ke zlepšení výsledků oproti předchozímu času. Bude obsahovat testovací otázky a bude sledovat aktivitu uživatele a časy odpovědí, po skončení testu tyto zaznamenané aktivity ve vhodném formátu dá k dispozici testujícímu. Časy poté budou porovnány s výsledky z předchozí verze aplikace.

5 Realizace

5.1 Formát dat

Požadavkem je na změnu formátu dat, tak aby každý prvek dat mohl obsahovat podmnožinu datových prvků, které budou vykazovat podobné chování jako původní entita (reprezentace dat v grafu).

5.1.1 Změna datového formátu

Aby každý prvek dat mohl obsahovat podmnožinu dalších datových prvků, je třeba definovat pole, které bude obsahovat tuto podmnožinu.

Můžeme si zde prohlédnout jak jeden datový prvek ve formátu JSON vypadá a co jednotlivé vlastnosti znamenají:

```
{
        "id":<Number>,
        "stereotype":<String>,
        "name":<String>,
        "begin":<String ISO8601>,
        "end":<String ISO8601>,
        "description":<String>,
        "properties": { ... }
}
```

- **id:** Vyjadřuje identifikaci prvku v grafu, díky tomuto můžeme prvek v grafu najít a nebo ho spojit s jiným prvkem pomocí hrany.
- **stereotype:** Vyjadřuje typ prvku, podle toho typu dokážeme poznat jaký použít *renderer* pro jeho vykreslení.
- **name:** Obsahuje hlavní název prvku, který se zobrazí uživateli.
- **begin:** Vyjadřuje datum začátku prvku v grafu.
- **end:** Vyjadřuje datum konce prvku v grafu. Pokud tento údaj neuvedeme, považujeme prvek za moment.
- **description:** Nepovinný údaj, který obsahuje popis prvku.

- **properties:** Nepovinný údaj, který obsahuje doplňující informace o prvku.

Proto, aby prvek obsahoval podmnožinu dalších prvků, jsem přidal do tohoto stávající formátu jednu vlastnost ("subItems"), která bude obsahovat pole všech pod-prvků viz:

```
{
        ...
        "properties": { ... },
        "subItems": { ... }
}
```

Pro možnost zobrazení pod-prvků je třeba definovat jejich formát. Formát by se měl podobat původnímu prvku a vypadá následovně:

```
{
        "id":<Number>,
        "name":<String>,
        "begin":<String ISO8601>,
        "end":<String ISO8601>,
        "css":<String>
}
```

- **id:** Vyjadřuje identifikaci pod-prvku v grafu, díky tomuto můžeme pod-prvek v grafu najít a nebo ho spojit s jiným prvkem pomocí hrany. Toto identifikační číslo musí být unikátní v celé množině dat (nemůže ho obsahovat ani žádný nad-prvek).
- **name:** Obsahuje hlavní název pod-prvku, který se zobrazí uživateli.
- **begin:** Vyjadřuje datum začátku pod-prvku v grafu.
- **end:** Vyjadřuje datum konce pod-prvku v grafu. Pokud tento údaj neuvedeme, považujeme pod-prvek za moment.
- **css:** Nepovinný údaj, který obsahuje doplňující kaskádové styly, které se přiřadí pod-prvku při vykreslování. Pro definování tříd jsem vytvořil soubor (*../src/css/customStyles.css*), ve kterém by se měly styly objevovat, je to doporučeno pro lepší přehlednost a údržbu stylů aplikace.

5.1.2 Zpracování nového datového formátu

Stávající aplikace obsahovala definici třídy *Entity*. Tato třída slouží jako reprezentace jednoho datového prvku a když je volán její konstruktor, jsou jí předána data, která se mají v této třídě uchovat. Z těchto dat lze jednoduše vyčíst atributy, které obsahují, a pak uložit do struktur třídy. Proto, aby tato třída dokázala uchovávat pod-prvky, bylo přidáno pole *subEntities*, do kterého se pod-prvky vkládají.

Pro reprezentaci těchto pod-prvků byla vytvořena třída *SubEntity*, která má podobné vlastnosti jako *Entity*, avšak nemůžeme tuto třídu přímo dědit kvůli chování v kontextu, kde nechceme interagovat přímo s každým pod-prvkem, ale nadřazeným prvkem, který je typu *Entity*. Proto třída *SubEntity*, dědí pouze od třídy *AbstractEntity*, abych získal vlastnosti jako je pozice v ose a přístup k základním informacím jako je identifikační číslo, priorita, typ a další.

Zpracování těchto pod-prvků zprvu probíhalo přímo ve třídě *Entity*, ale po další implementaci jsem zjistil, že je třeba tyto nové pod-prvky nějakým způsobem registrovat u zdroje dat *StaticSource*, proto veškeré zpracovávání probíhá právě v tomto zdroji dat. Do stávající doby tento zdroj obsahoval slovník *entities* pro uchovávání instancí *Entity*, proto jsem přidal nový slovník *allMappedEntities* pro mapování všech všech prvků, které se v grafu nachází.

Tímto je zajištěno veškeré zpracování nového formátu dat, dále se budeme zabývat jak tyto data zobrazit.

5.1.3 Základy pro vizualizaci

V předchozí části jsem si připravil data pro vizualizaci, teď je potřeba vytvořit obálku, která by uchovávala definovaná data a informace o zobrazení v grafu. Ve stávající aplikaci tuto obálku zastupovala třída *BandItem*, která dědí od abstraktní třídy *AbstractItem*, tato třída uchovává mnoho informací, ale nejdůležitější z nich jsou informace o datech, pozici, popisku a zásadní je informace o elementu, který reprezentuje tyto data. S tímto elementem se dále pracuje při zobrazování prvku v grafu nebo taktéž při zobrazení spojení mezi prvky v grafu.

Abych dokázal tedy takovou obálku vytvořit, je třeba definovat si podobnou ne-li stejnou třídu jako je *BandItem*. Mám zde několik možností jak tohoto docílit.

První ze způsobů je novou třídu *SubItem* navrhnout tak, aby dědila všechny vlastnosti od *BandItem*, tímto bych vyřešil veškeré problémy s reprezentací, avšak nastal by problém, že pro každý druh pod-prvku by musel

existovat vlastní renderer. Další a závažnější problém je, že by aplikace pracovala s touto instancí jako stejně rovnou s *BandItem* instancí, to by vedlo k nechtěnému chování pod-prvků a k velkému množství ošetřování kódu ve stávající aplikaci.

Druhý ze způsobů je novou třídu navrhnout tak, aby dědila pouze od *AbstractItem*, tím získáme obálku pro základní práci jako je přístup k datum, elementu, pozici a velikosti. Tímto způsobem renderery zůstanou pouze pro *BandItemy* a do těchto rendererů bude třeba přidat pouze jednu metodu pro vyrenderování a překreslení *SubItemů*. Jednoduše poté přidám těmto instancím požadované chování.

Nakonec jsem použil druhý ze způsobů a přidal jsem každé instanci *BandItem* vlastní pole *subItems* pro uchovávání instancí třídy *SubItem*. Tyto instance třídy *SubItem* jsou přidávané při přidávání *BandItem* do instance *Band*, která zastupuje jednotlivý pás zobrazení.

Tímto máme definovanou obálku pro data a další zpracovávání této obálky uvedu v části s renderery.

5.2 Změna určování rozdělení

Z Obrázek 2.1 vidíme, že je aplikace rozdělená na více pásů. Stávající řešení bylo takové, že každý z těchto pásů je vytvořen pouze pro jeden typ prvku dat. Původní definice pásů v *main.js* vypadala následovně:

```
bands : [
{
        id: "place",
        label: "Mista",
        itemRenderer: new BandItemRenderer("#7DD968"),
        color: "#f5f5f5"
},
{
        id: "person",
        label: "Lide",
        itemRenderer: new BandItemRenderer("#FFB182"),
        color: "#fafafa"
}
]
```

- **id:** Vyjadřuje jaký *stereotype*(typ) dat do pásu patří viz Formát dat.

- **label:** Obsahuje popisek pásu
- **itemRenderer:** Určuje jakým rendererem se mají prvky v pásu zpracovávat.
- **color:** Jakou barvou budou prvky v pásu obarveny.

Tímto způsobem jsem nemohl zajistit to, aby jeden pás dokázal obsahovat více typů dat. Proto jsem implementoval nové řešení rozdělení a definice pásů.

Definice pásů se nyní odehrává v třídě *BandsDistribution* a vypadá následovně:

```
bands : [

{
        id: "person",
        label: "Lide",
        itemRenderer: new BandItemRenderer("#FFB182"),
        color: "#fafafa"
},
{
        id: "Sjednoceny pruh",
        label: "Sjednoceni",
        types: [
          {
          id: "event",
          itemRenderer: new BandItemRenderer("#F2BC53"),
          color: "#f5f5f5"
          },
          {
          id: "item",
          itemRenderer: new BandItemRenderer("#78B4FF"),
          color: "#fafafa"
          }
        ]
}
]
```

Pásy nyní lze definovat dvěma způsoby. Prvním je původní způsob viz pás definovaný jako "Lide". Druhý je nový způsob, ve kterém je přidáno nové pole *types*, které definuje jaké typy dat lze v tomto pásu zobrazovat. Tyto typy lze definovat jako původní pás, ale bez potřeby definice popisu *label*.

Abych zajistil zpracování této nové definice pásu bylo potřeba zajistit několik úprav ve stávající aplikaci. Ve třídě *BandGroup*, bylo zapotřebí definice nového slovníku *bandTypes*, který obsahuje pro každý typ dat přiřazený pás instance *Band*. Pak následné hledání pásů a entit probíhá nejdříve v nově definovaném slovníku *bandTypes* a až v původním slovníku *bands*, tímto jsem dosáhl kompatibility s původní definicí pásů. Dále bylo zapotřebí definovat ve třídě *Band* nový slovník *types*, který obsahuje ke každému typu dat renderer, kterým se mají data vyobrazit, při vytváření instance *BandItem* nebo *SubItem* lze zjistit jaký renderer jim přiřadit podle jejich typu.

5.3 Nové renderery

Nyní po implementaci výše zmíněných věcí, může pod-prvky zobrazovat jakýkoliv renderer ve stávající aplikaci, jedinou podmínkou pro toto zobrazení je implementace dvou hlavních metod, pro práci s těmito pod-prvky. Tyto metody jsou *renderSubItem* a *correctProtrusionSubItem*, funkčnosti těchto metod budou popsány níže.

První z metod *renderSubItem* určuje jakým způsobem bude generována HTML obálka instance *SubItem*. Této obálce nastaví základní atributy jako je identifikační číslo v kontextu, třídy kaskádových stylů definované u dat. Tato metoda je volána pouze jednou a to při generování grafu.

Druhá z metod *correctProtrusionSubItem* určuje jakým způsobem se počítá pozice a šířka výše generované obálky. Tato metoda je volána při jakémkoliv překreslení grafu. Dále je potřeba implementovat způsob jakým se bude určovat pozice *SubItem*.

5.3.1 Počítání pozice *SubItem*

Jelikož jsou všechny prvky v grafu určeny absolutní pozicí, je třeba definovat tuto pozici vztahující se k nadřazenému prvku *BandItem*. Proto jsem přidal do třídy *SubItem* atribut *leftPositionToParent*, která určuje vzdálenost zleva k nadřazenému prvku. Tuto vzdálenost lze počítat 3 různými způsoby.

První ze způsobů jak získat vzdálenost zleva je určen vztahem:

$l = w \times \frac{d}{d_e}$, kde l je počítaná vzdálenost, w je šířka nadřazeného prvku, d je rozdíl trvání nadřazeného prvku a podřazeného prvku, d_e je doba trvání nadřazeného prvku.

Druhý ze způsobů jak získat vzdálenost zleva je určen vztahem:

$l = px(s_s - s_e)$, kde l je počítaná vzdálenost, px je funkce pro převod na pixely, s_s je začátek podřazeného prvku, s_e je začátek nadřazeného prvku.

Třetí ze způsobů jak získat vzdálenost zleva je určen vztahem:

$l = px(s_s) - l_e$, kde l je počítaná vzdálenost, px je funkce pro převod na pixely, s_s je začátek podřazeného prvku, l_e je vzdálenost zleva nadřazeného prvku.

Pro řešení byl zvolen třetí způsob počítání vzdálenosti a to proto, že dokázal správně počítat vzdálenost, i když prvek nebyl přímo zobrazený v zorném poli grafu.

5.3.2 Implementace rendererů

SplitBandItemRenderer

Požadavek na první renderer je takový, aby uměl zobrazovat podmnožinu prvků entity. Tato podmnožina prvků může obsahovat jak typ moment, tak dobu nějakého trvání. Další vlastností tohoto rendereru bude, aby uměl tuto podmnožinu schovat a zobrazit za určitých podmínek.

Abych tento požadavek splnil, vytvořil jsem třídu *SplitBandItemRenderer*. Tato třída dědí od původního *BandItemRenderer*, tím získává metody pro práci s *BandItem*. Tento nový renderer musí obsahovat metody pro práci s instancemi *SubItem* popsány viz výše. Dále je třeba přetížit metodu pro překreslení *redraw*, tato metoda slouží k volání metod pro přepočet pozice a šířky podprvků. Tuto metodu jsem zachoval podobnou jako původní definovanou v *BandItemRenderer*, ale přidal jsem do ní volání metody *correctProtrusionSubItem*, která pro každou instanci *SubItem* opraví pozici a šířku. Dále v této metodě *redraw* byl přidán kód, který zajišťuje schování a zobrazení podmnožiny prvků za určité velikosti instance *BandItem*. Těmito změnami jsem dosáhl požadovaného chování viz požadavek. Výsledek tohoto rendereru si můžete prohlédnout na Obrázku 5.1.

Obrázek 5.1: Výsledek rendereru

DumbbellItemRenderer

Požadavek na druhý renderer je takový, aby uměl taktéž podmnožinu prvků, ale v jiném formátu. Prvek typu moment se zobrazí jako kruh a prvek typu doba se vykreslí jako obdélník. Tyto prvky budou spojeny spojovací čárou, aby bylo zřejmé, že k sobě patří.

Abych tento požadavek splnil, vytvořil jsem třídu *DumbbellItemRenderer*. Tato třída opět dědí od původního *BandItemRenderer*, avšak z dů-

vodu jiného způsobu vykreslování bylo třeba přetížit více základních metod. Abych vyřešil spojení podmnožiny prvků čárou, přetížil jsem metodu *renderContinuous*, v této metodě jsem přidal vložení čáry do obálky celého *BandItem*. Avšak nastal problém s tím, že původní *BandItem* měl text podle pozice v grafu buď uvnitř nebo vedle této obálky. Proto bylo třeba přetížit metodu *redrawLabel*, která upravuje pozici textu popisku vůči obálce. Tuto metodu jsem upravil tak, aby text popisku byl vždy vedle obálky *BandItem*, tímto bylo vyřešeno "přeškrtnutí textu". Dále bylo třeba definovat nový atribut *type* dat proto, aby byl tato instance měla vždy nějaký konec a začátek. Tento atribut dat vypadá následovně:

```
{"id":5",stereotype":"dumbbell-entity","name":"TestDumbbell",
"begin":"2010-09-27T00:00:00","end":"2150-06-21T23:59:59",
"properties":{"startPrecision":"day"},
"subItems":
[
  {"id":6,"name":"King","type":"start"},
  {"id":7,"name":"Dumbell","type":"end"},
  {"id":8,"name":"Queen","begin":"2020-09-27T00:00:00"},
]
},
```

Tímto typem řekneme jaký prvek je považován za start a jaký za konec, není třeba dodávat atribut *begin*, protože se tomto prvku přiřadí buď to začátek nebo konec nad-prvku. Metoda *redraw* je taktéž přetížena a doplněna o volání přepočítání všech pod-prvků a taktéž za určité podmínky schová všechny pod-prvky a nebo je zobrazí. Tímto způsobem jsem dosáhl požadovaného chování viz požadavek. Výsledek tohoto rendereru si můžete prohlédnout na Obrázku 5.2.

Obrázek 5.2: Výsledek rendereru

5.4 Testovací nástroj

6 Diskuze

7 Závěr

Literatura

[1] LIPKA, R., KACEROVSKÝ, M., HRBÁČEK, D. *Timeline visualisation tool.* Západočeská univerzita v Plzni 2016.

[2] Sônia Fernandes Silva and Tiziana Catarci *Visualization of Linear Time-Oriented Data.* Agora Telematica Spa, Roma, Italia

[3] Paul Craig, Néna Roa-Seïler *A Vertical Timeline Visualization for the Exploratory Analysis of Dialogue Data.* Universidad Tecnológica de la Mixteca, Edinburgh Napier University

[4] Matt Jensen *Visualizing Complex Semantic Timelines.* NewsBlip 239 Bellevue Ave. E.

[5] Jian Zhao1, Steven M. Drucker, Danyel Fisher, Donald Brinkman *TimeSlice: Interactive Faceted Browsing of Timeline Data.* Department of Computer Science, University of Toronto. Microsoft Research, Redmond

[6] Ruth Rosenholtz, Yuanzhen Li, Jonathan Mansfield, and Zhenlan Jin *Feature Congestion: A Measure of Display Clutter* . MIT Dept. of Brain and Cognitive Sciences 3 Cambridge Center, NE20-447, Cambridge, MA 02139, USA

[7] Weiwei Cui *A Survey on Graph Visualization.* Hong Kong University of Science and Technology Clear Water Bay, Kowloon, Hong Kong

[8] Wolfgang Aigner, Silvia Miksch, Heidrun Schumann, Christian Tominski *Visualization of Time-Oriented Data.*

[9] https://en.wikipedia.org/wiki/Timeline *Timeline.*

[10] J.D. Mackinlay, G.G. Robertson and S.K. Card *The Perspective Wall: Detail and Context Smoothly Integrated.*