

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

Diplomová práce

Vizuální reprezentace precedenčního grafu

Plzeň 2015

Bc. Michal Kacerovský

Prohlášení

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

V Plzni dne 25. června 2015

Bc. Michal Kacerovský

Abstrakt

Vizuální reprezentace precedenčního grafu

Po staletí jsou sbírány a archivovány historické záznamy. Dnes jich tedy existuje velké množství, na které je obtížné získat ucelený pohled. Tato práce se zabývá možnostmi, jakými lze data tohoto charakteru nejen zobrazit ve webovém prohlížeči, ale zároveň i opatřit vizualizací vztahů, jež mezi nimi existují. První část zkoumá historii grafické reprezentace časových dat a upozorňuje na problémy s ní spojené. Na jejich základě stanovuje požadavky pro implementaci ovládacího prvku časové osy a prezentuje některá existující řešení. Poslední kapitoly této práce se zabývají návrhem a realizací vlastní časové osy doplněné o testy a uživatelskou příručku.

Abstract

Visual representation of precedence graph

Over the centuries, historical records are being collected and archived. There is a huge amount of such data, thus it is difficult to get complex insight. This diploma thesis deals with solutions that can be used not only for its chronological displaying in a web browser but even for visualization of relationships that exist in between. The first part examines the history of time data graphical representation and points out related problems. Consequently, it determines requirements for timeline widget implementation and introduces some existent solutions. The last chapters are focused on designing and creating its own timeline widget accompanied by tests and user guide.

Poděkování

Při vytváření teoretické i praktické části této práce mi svými radami, podporou nebo jiným způsobem přispělo několik lidí, kterým bych rád vyjádřil vděčnost. Můj dík tedy patří

vedoucímu práce Ing. Richardu Lipkovi, Ph.D., za cenné rady, trefné připomínky a celkově příjemnou spolupráci,

Davidu Hrbáčkovi za konzultace všech možných problémů týkajících se nejen diplomové práce,

všem testerům, jmenovitě Martinu Altmanovi, Tomáši Le Vanovi, Kristýně Kacerovské, Olze Kacerovské, Antonínu Kacerovskému, Miroslavu Kubiskovi, Milanu Mrázovi, Janě Pilátové, Martinu Sedlákovi, Jiřímu Šmejkalovi a dalším účastníkům testování,

Mgr. Martinu Jakubíkovi za to, že si ve svém čase našel pár okamžiků, aby zkontroloval jazykové nedostatky této práce,

a v neposlední řadě rodině a příteli za podporu během studia.

Obsah

1	Úvod	1
2	Historie grafické reprezentace dat	2
2.1	Vizualizace	2
2.2	Časová osa	2
3	Data historických záznamů	5
3.1	Precedenční graf	5
3.2	Povaha dat	6
4	Poznámky k vizualizaci	8
4.1	Vizualizace na webu	8
4.2	Komplikace spojené s vizualizací	9
4.2.1	Rok 1 př. n. l.	9
4.2.2	Letní čas	10
4.2.3	Přesnost datování	10
4.2.4	Entita typu <i>místo</i>	11
5	Analýza požadavků	13
5.1	Technické požadavky	13
5.2	Funkční požadavky	14
5.3	Požadavky na uživatelské rozhraní	15
6	Analýza existujících řešení	17
6.1	TimelineJS	17
6.2	Dipity	18
6.3	Timeglider	19
6.4	vis.js	20
6.5	Timeline (SimileWidgets)	21
6.6	Shrnutí	22
7	Návrh nového widgetu	23
7.1	Obecná charakteristika	23
7.1.1	Rozsah projektu	23

7.1.2	Kontext widgetu	23
7.1.3	Omezení návrhu a implementace	24
7.2	Demo aplikace	24
7.3	Uživatelské rozhraní	25
7.4	Značení parametrů	25
7.5	Úrovně přiblížení, transformace	25
7.6	Komponenty widgetu a jejich rozvržení	27
7.6.1	Průhled (viewport)	28
7.6.2	Časová lišta (timebar)	28
7.6.3	Obálka (wrapper)	29
7.6.4	Pravítko (ruler)	29
7.6.5	Pás (band)	32
7.6.6	Skupina pásů (band group)	34
7.6.7	Položka pásu (band item)	35
7.6.8	Vrstva vztahů (relation viewer)	37
7.7	Základní interakce	39
7.7.1	Posun osy	39
7.7.2	Změna úrovně přiblížení	40
7.7.3	Zaostření	41
7.8	Načítání dat a jejich formát	42
7.8.1	Datový zdroj	42
7.8.2	Formát vstupních dat	43
8	Implementace	45
8.1	Použité knihovny jiných stran	45
8.1.1	jQuery a jQuery UI	45
8.1.2	RequireJS	45
8.1.3	moment.js	47
8.1.4	Snap.svg	48
8.1.5	Knihovny použité pro demo aplikaci	48
8.2	Vlastní podpůrné moduly	48
8.2.1	oop.js	48
8.2.2	moment.extension.js	52
8.3	Fyzická struktura aplikace	52
8.4	Konvence zdrojových textů	54
8.5	Stručný popis zdrojových souborů widgetu	54
8.5.1	Balík cz.kajda.common	55
8.5.2	Balík cz.kajda.data	55
8.5.3	Balík cz.kajda.timeline	56
8.5.4	Balík cz.kajda.timeline.band	57
8.5.5	Balík cz.kajda.timeline.render	58
8.6	Komponenty	59
8.6.1	Životní cyklus	59

8.6.2	Hierarchie v rámci widgetu	60
8.7	Vybrané problémy implementace	60
8.7.1	Rok 1 př. n. l	60
8.7.2	Letní čas	61
8.7.3	Přesnost datování	61
8.7.4	Entita typu <i>místo</i>	62
8.8	Demo aplikace	62
8.8.1	Implementace datových zdrojů	62
8.8.2	Pomocné třídy	63
8.8.3	Změna datového zdroje	63
9	Testování	64
9.1	Metody testování	64
9.2	Testování kompatibility	64
9.2.1	Výsledky testování kompatibility	64
9.3	Výkonnostní testy	65
9.3.1	Výsledky výkonnostních testů	66
9.4	Uživatelské testy	68
9.4.1	Charakteristika testerů	69
9.4.2	Výsledky testů	70
9.4.3	Uživatelské hodnocení	71
9.5	Význam výsledků pro další vývoj	73
9.5.1	Vizualizace vztahů	73
9.5.2	Označování záznamů	74
10	Závěr	75

1 Úvod

Každý den většina z nás používá při své práci nebo zábavě internet. Nemusíme pro každou informaci pospíchat do knihovny nebo zevrubně prohledávat tu největší encyklopedii, kterou máme ve své polici, tak jako před několika lety. Ačkoliv v síti nalezneme neskutečné množství dat a ve většině případů se i dopátráme kýženého výsledku, ne vždy můžeme s určitostí prohlásit, že jde o správný výsledek, a ne pokaždé jsme vůbec schopni nalezená data zpracovat.

Obvykle pracujeme s obyčejnými obrázky, tabulkami a texty navzájem provázanými pomocí hypertextových odkazů. Co když ale potřebujeme pohlížet na získané informace s větším odstupem, chceme-li znát spíš vztahy mezi nimi, než jejich obsah jako takový? Pokud budeme zkoumat například rodokmen do druhého kolene, jistě si s pomocí tužky a papíru poradíme. Co když ale potřebujeme jít ještě dál a chceme sledovat vztahy mezi historickými osobami v řádech stovek, nebo dokonce tisíců let. Na to jen s naší pamětí a představivostí nestačíme, obzvlášť máme-li je analyzovat podrobněji.

Předchozí odstavec již ukázal, že příkladem informací, které jsme schopni jen obtížně zpracovávat bez pomoci počítače, jsou právě historické záznamy. Ať už jde o válečné události, sledování generací královských rodů nebo po minutách monitorovaný teroristický útok, všechna tato data lze uspořádat jak chronologicky v čase, tak i v grafu, jehož hrany představují relace mezi jednotlivými událostmi a osobami. Ačkoliv člověk zvládne libovolné množství takových záznamů uspořádat v čase, s jeho transformací do zmíněného grafu si neporadí už jen proto, že graf takových rozměrů lze jen těžko zpracovat na papíře.

Tato diplomová práce se tedy zaměří na to, jak data historického charakteru vizualizovat, aby z nich byl snadno patrný jak jejich sled v čase, tak i vzájemné vztahy. Zřejmě není možné splnit oba tyto aspekty vizualizace na 100 %, a proto tato práce upřednostní chronologické uspořádání před grafovým. Uživatel však bude mít možnost získat informace o tom, v jakém vztahu s okolím je jím zvolený uzel.

Widget pro HTML stránku, který bude výstupem této práce, pak uživateli umožní přímo v jeho webovém prohlížeči procházet předzpracovaná data z externí databáze dostupná prostřednictvím REST rozhraní.

2 Historie grafické reprezentace dat

2.1 Vizualizace

Vizualizace je oborem, který má poměrně dlouhou historii, navzdory tomu ale funguje jako nezávislá oblast výzkumu na poli výpočetní techniky jen pár desítek let. V roce 1987 ji McCormick, americký vědec zabývající se počítačovými technologiemi, definoval následovně:

Definice 2.1. *Vizualizace* je metoda výpočtu. Transformuje symbolické objekty na geometrické, čímž výzkumným pracovníkům umožňuje sledovat jejich simulace a výpočty. Vizualizace nabízí způsob, jak spatřit nevídané detaily. Obohacuje proces vědeckého objevu a podporuje hluboké a neočekávané poznatky [1].

Tehdejším cílem se stalo spojení obrovských schopností lidského vnímání a síly výpočetní techniky, jež umožní uživatelům snáze data analyzovat a porozumět jim. V souvislosti s tímto záměrem vznikly tři základní podmínky, které by měla vizualizace splňovat [2]:

1. *významnost (expressiveness)*, schopnost vyjádřit přesně informaci, kterou data nesou – nic nevynechat ani nic nepřidat,
2. *efektivita (effectiveness)*, jež uvažuje úroveň lidského vnímání a další aspekty tak, aby bylo možné jednotlivé prvky vizualizace snadno rozpoznat a interpretovat,
3. *vhodnost (appropriateness)* zabývající se náklady nutnými k tomu, aby vizualizace dosáhla s ohledem na daný úkol požadovaných výsledků.

Mimo výše uvedené nás při vizualizaci rovněž zajímá, *co* a *proč* budeme zobrazovat – aby mohla být data vizuálně reprezentována tak, že jim člověk skutečně porozumí, je potřeba zcela pochopit jejich význam, podobu a důvod, proč vůbec jejich grafická podoba vzniká.

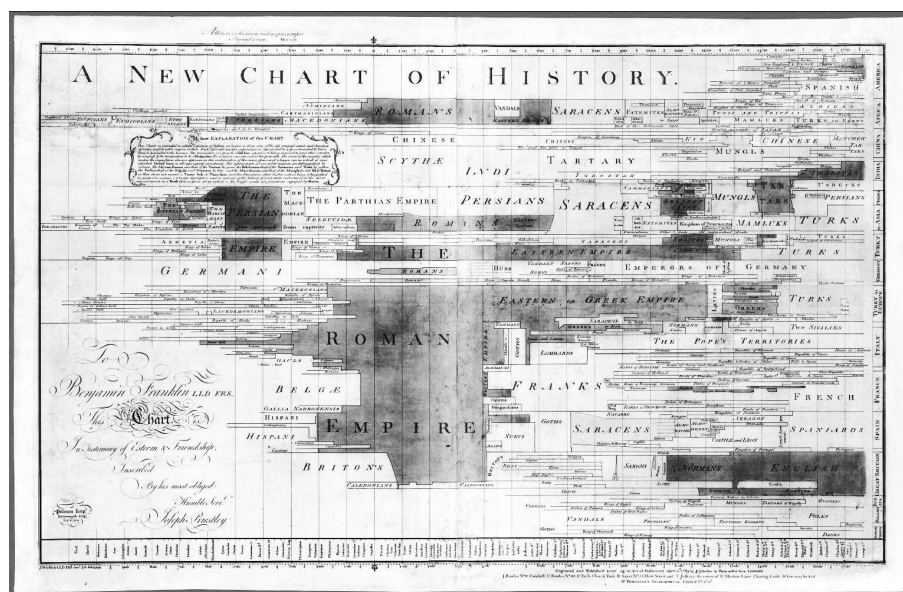
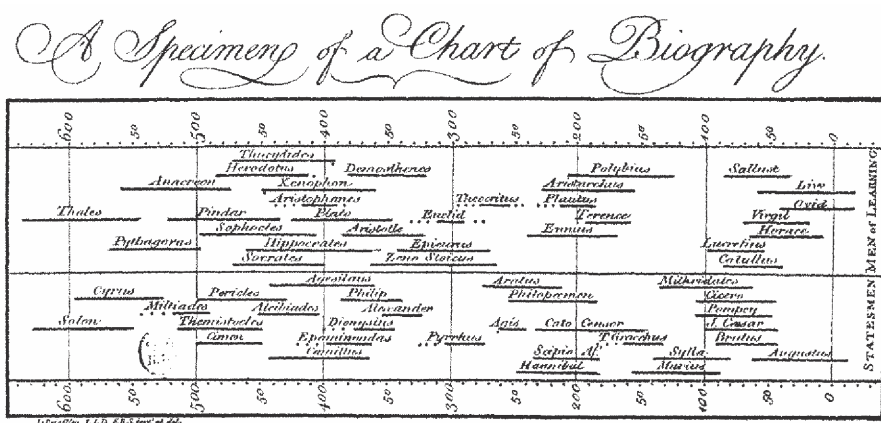
2.2 Časová osa

Definice 2.2. *Časová osa (timeline)* představuje způsob, jakým lze zobrazit skupiny událostí či záznamů uspořádaných chronologicky podle jejich časového zasažení [3] obvykle za pomoci *pravítka* (legendy) a jejich grafické reprezentace umístěné na odpovídajících pozicích.

Za první moderní (tj. do jisté míry interaktivní) časovou osu lze považovat *Carte chronographique* popsanou v roce 1753 Jacquesem Barbeu-Dubourgem. Šlo o více než šestnáctimetrový papír zobrazující v čase události od *stvoření světa* do období, v němž Dubourg žil, celý skrytý v rozkládatelném pouzdře

New Chart of History (1769), v němž můžeme pozorovat práci s velikostí písma v závislosti na prioritě dané informace [5].

Při pohledu do historie, který tato kapitola poskytla, zřetelně vidíme, že dnešní časové osy zcela vycházejí ze svých o několik stovek let starších předchůdců a zachovávají tehdy nastavené principy, jako je použití symbolů pro označení odlišností, velikost či změna řezu písma zdůrazňující priority apod.



Obrázek 2.2: Priestův A Chart of Biography (nahore) a New Chart of History (zdroj: commons.wikimedia.org, davidrumsey.com)

3 Data historických záznamů

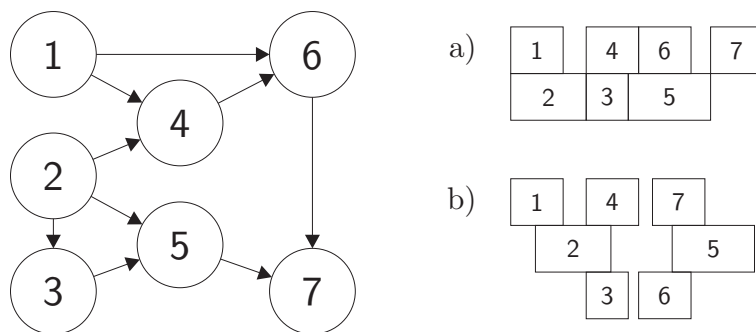
3.1 Precedenční graf

Definice 3.1. *Precedenční graf* (alternativně diagram) je orientovaný, acyklický graf G , jehož množina vrcholů $V(G)$ reprezentuje postupně uspořádané činnosti a každá hrana $\{i, j\} \in E(G)$ vedoucí z vrcholu $i \in V(G)$ do vrcholu $j \in V(G)$ vyžaduje, aby byla činnost reprezentovaná vrcholem i dokončena, než započne činnost charakterizovaná vrcholem j [6].

Obvykle jsme zvyklí pomocí takto definovaného precedenčního grafu popisovat plánování výrobního procesu nebo synchronizaci paralelních procesů. V této práci ovšem nemanipulujeme ani z jedním z uvedených, zabýváme se událostmi nebo záznamy určenými v čase. I na ty ale můžeme pohlížet podobně jako výše uvedená charakteristika precedenčního grafu. Na místo činností zasazujeme historické události, postavy, dokumenty nebo lokace – v této práci dále označované jako *záznamy* nebo *entity*. Mezi entitami definujeme *vztahy*, případně *relace*.

Podmínku precedence ale nelze brát při aplikaci na historická data tak, jak ji uvádí předchozí odstavec, např. u osob ve vztahu rodič-potomek by totiž znamenala, že rodič musí nejdříve zemřít a až tehdy se může narodit jeho potomek, což je samozřejmě nesmysl. Proto upravíme znění definice 3.1 do této podoby:

Definice 3.2. *Precedenční graf historických dat* je orientovaný, *pseudo-acyklický* graf G , jehož množina vrcholů $V(G)$ reprezentuje entity určené v čase a každá hrana $\{i, j\} \in E(G)$ vedoucí z vrcholu $i \in V(G)$ do vrcholu $j \in V(G)$ popisuje relaci mezi těmito vrcholy, přičemž tou může být například *účast na události*, *konání v místě*, *příbuzenský vztah* a další.



Obrázek 3.1: Rozdíly v interpretaci precedenčního grafu. Diagramy vpravo uvádějí sled záznamů v čase a) dle definice 3.1, b) dle definice 3.2

Vezmeme-li dva uzly, kde jeden reprezentuje otce a druhý syna, získáme mezi nimi dva vztahy odpovídající dvěma hranám se vzájemně opačnou orientací –

jednu ve významu *potomek* (ve směru od syna k otci) a druhou vyjadřující *rodičovství* (ve směru od otce k synovi). Ty zřejmě vytvoří v grafu cyklus, čímž při vycházení z definice 3.1 poruší podmínku acykličnosti. Zmíněné relace jsou ale v jistém smyslu symetrické, sice v každém směru popisujeme jejich význam jinými slovy, avšak ve skutečnosti vyjadřují jediný vztah. Takové páry hran tedy budeme vnímat jakou samotnou hranu směřující od vrcholu se starším datováním k mladšímu a graf, jehož součástí jsou, označíme jako *pseudo-acyklický*.

Takže zatímco podle definice 3.1 všechny činnosti předcházející té v současnosti aktivní musí být již dokončené a jejich graf lze označit za čistě precedenční, definice 3.2 tuto podmínku nevyžaduje, a dokonce připouští možnost, že entita, která existovala před současnou entitou, tuto „přežije“ – příkladem může být situace, kdy otec žije déle než jeho potomek. V takovém případě je graf precedenční z hlediska kauzality jeho vrcholů. Rozdíl mezi těmito dvěma způsoby interpretace ilustruje obrázek 3.1.

3.2 Povaha dat

Vzhledem k tomu, že se tato práce zabývá pouze vizualizační stránkou, a nachází se tedy na samém konci procesu prezentujícího data uživateli, sama o sobě ne-definuje podobu datových struktur. Jejich popis je součástí prací¹, na něž tato navazuje, a datové typy použité v rámci výsledné aplikace staví právě na něm. Nicméně, jak uvedla kapitola 2.1, musíme stanovit, co představuje cíl vizualizace, co je vlastně jejím předmětem.

Pro tento projekt byly na databázové úrovni stanoveny čtyři základní typy entit, z nichž vycházely datové struktury i na dalších vrstvách:

- *osoba* obvykle charakterizovaná jménem, příjmením nebo titulem,
- *objekt*, jímž může být například dokument nebo korunovační klenoty,
- *místo* popsané názvem, případně i GPS² souřadnicemi
- a *událost* reprezentující jakýkoliv okamžik, kdy něco vzniklo, zaniklo, probíhalo apod.

Každá entita disponuje mimo výše zmíněné vlastnosti také datem (a časem) začátku, tedy okamžiku, kdy vznikla, začala nebo se v případě lidské bytosti narodila. Dalo by se očekávat, že v sobě rovněž zahrnuje datum ukončení. To však není případ každé entity. Nelze popřít, že u entit typu *osoba* musí datum

¹Souběžně realizované diplomové práce, jejichž vydání je plánováno v roce 2015.
MERUNKO, D. *Generování a vizualizace časové osy*. ZČU, Plzeň.
HRBÁČEK, D. *Zpracování časových údajů pro jejich vizualizaci*. ZČU, Plzeň.

²*Global Positioning System* – globální vojenský družicový systém umožňující určení polohy kdekoli na Zemi

ukončení (nebo lépe úmrtí) existovat. Podívejme se ale například na korunovaci krále – můžeme vždy říct, v kolik hodin začala a v kolik skončila? Pokud nahlédneme do učebnic či jiné literatury, vyčteme pravděpodobně jediné datum či dokonce jen rok. Událost takového typu totiž není chápána jako časový interval, neuvádí se v žádném rozmezí, byť fyzicky nějakou dobu musela trvat. Definujeme ji následovně:

Definice 3.3. *Momentová entita* je historický záznam, který navzdory jeho skutečnému časovému trvání považujeme za jediný okamžik, jenž může být popsán jak malou (sekundy), tak i velkou (roky, století) časovou jednotkou.

Řekněme, že zmíněná korunovace proběhla v nějaký srpnový den roku 1220. Protože jde o momentovou entitu (korunovaci považujeme za jediný okamžik), budeme její čas uživateli prezentovat jako *srpen 1220*, přičemž vhodnou formou zobrazení naznačíme, že ve skutečnosti její trvání nedosáhlo 31 dnů, nýbrž jen několika hodin.

Při pohledu na definici 3.3 vidíme, že její podstata nepopisuje podobu entit typu *osoba* ani některých dalších entit jiných typů. Nikdy patrně nebudeme historické postavy časově charakterizovat jediným okamžikem, naopak zde přímo vyžadujeme, aby takový záznam disponoval jak informací o svém začátku, tak i konci. Proto zavádíme pojem *intervalové entity*.

Definice 3.4. *Intervalová entita* je historický záznam, jehož začátek i konec můžeme datovat dvěma odlišnými časovými údaji určenými v libovolně velkých časových jednotkách. Dobu uplynulou mezi datem začátku a konce záznamu pak označujeme jako *trvání entity*.

Mezi intervalové entity tedy budeme řadit jak všechny osoby, tak i války, obléhání, dobývání či jiné události s trváním.

4 Poznámky k vizualizaci

Jak již zmiňoval poslední odstavec úvodní kapitoly, výsledkem této práce bude widget¹ pro webovou aplikaci doplněný o vnější prostředí potřebné k jeho testování a demonstraci. Další kapitoly textu této diplomové práce se tedy budou zabývat výhradně vizualizací v prostředí webového prohlížeče.

4.1 Vizualizace na webu

Vizualizace spadá do oblasti počítačové grafiky, která řeší, jak prostorová data (ať už dvourozměrná nebo trojrozměrná) prezentovat uživateli přirozenou formou. I v případě tohoto projektu jde o to, jak databázi historických záznamů transformovat do takové podoby, která lidem umožní jejich snadné čtení.

Při vývoji grafických aplikací v běžných programovacích jazycích, kterými mohou být Java nebo C++, lze uplatnit princip *renderování grafických objektů*². Jedna vrstva aplikace mapuje data na objekty, další těmto objektům přiřazuje grafickou podobu a poslední pak rozhoduje o tom, co a jak má být vykresleno. Uživateli se tak prezentují pouze ty grafické informace, jež jsou z jeho pohledu v daný okamžik relevantní – jen ty jsou „přeměněny na pixely“ a zobrazeny prostřednictvím *viewportu*, části zobrazovacího zařízení, která funguje jako zorné pole.

Tento přístup lze zachovat i u webové aplikace za předpokladu, že se vývojář rozhodne použít tzv. *canvas*, který představuje alternativu zmíněného viewportu a jenž se stal součástí HTML5³ [7]. V takovém případě se však vzdává možnosti interakce s obsahem – jakmile je jednou do canvasu něco vykresleno, změní se veškeré informace na pouhé pixely. Proto se canvas hodí především pro realizaci her, úprav rastrových obrázků a obecně pro úlohy vyžadující manipulaci s pixely. Druhou alternativu představuje použití SVG⁴ [8], které pracuje s DOM⁵. Ten vývojáři poskytuje možnost přistupovat ke všem objektům, jež jsou součástí SVG dokumentu, a zachycovat události, které nad nimi vznikají.

S volbou SVG již tedy nelze mluvit o *renderování* v pravém slova smyslu, protože je vývojář odstíněn od samotného vykreslení pixelů (ve své podstatě operuje pouze na úrovni grafické reprezentace dat), a úloha vizualizace tak zcela mění svoji podobu.

¹ovládací prvek pro interakci aplikace s uživatelem, který obvykle slouží pro manipulaci s používanými daty

²proces tvorby reálného obrazu na základě počítačového modelu

³*HyperText Markup Language* – značkovací jazyk, který umožňuje sémanticky popisovat hypertextové dokumenty

⁴*Scalable Vector Graphics* – značkovací jazyk pro popis vektorové grafiky (vychází z XML)

⁵*Document Object Model* – objektová reprezentace HTML (obecně XML) dokumentu

4.2 Komplikace spojené s vizualizací

4.2.1 Rok 1 př. n. l.

Specifikace ISO 8601 [9] pro práci s datem, časem a časovými intervaly (v gregoriánském kalendáři) definuje několik standardů zápisu takových informací pomocí řetězce. Protože tento projekt bude pracovat i s daty před naším letopočtem, zaměříme se především na následující formát `+YYYYY-MM-DD`⁶, kde `+` (příp. `-`) definuje, zda jde o rok před naším letopočtem nebo našeho letopočtu.

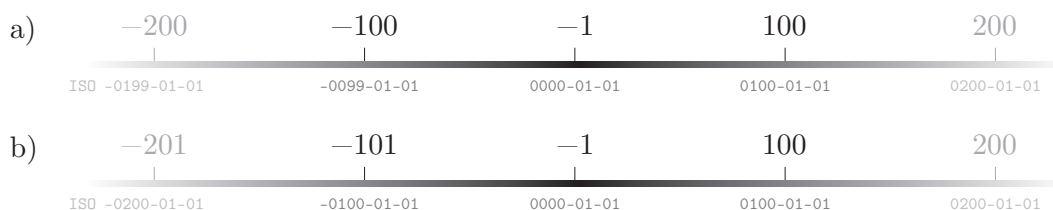
Zde ovšem dochází ke komplikaci. ISO norma shledává totiž následující zápisy

```
+00000-01-01
-00000-01-01
0000-01-01
```

standardními, což je v rozporu se skutečností, že rok 0 v gregoriánském kalendáři neexistuje. ISO 8601 používá astronomický systém číslování roků, jenž se právě liší tím, že rok nula akceptuje a reprezentuje jím gregoriánský rok 1 př. n. l. [9]. Jelikož literatura běžně používá pro zápis historických dat systém gregoriánského kalendáře, i tento projekt by měl uživateli zobrazovat veškeré informace v této formě. V takovém případě ale dochází k tomu, že v celé éře před naším letopočtem se ISO interpretace bude o rok lišit od gregoriánského zápisu.

To na první pohled nemusí působit jako problém, nicméně představme si tuto situaci: Na časové ose chceme zobrazit jednotlivá století kolem počátku našeho letopočtu. Protože rok nula neexistuje, zvolíme jako střed rok `-1` (tj. 1 př. n. l.). Pokud od tohoto středu pak vyznačíme sto let v obou směrech, v levé části osy budou popisky mezi stoletími končit jedničkou (obrázek 4.1), protože dle ISO jsme od roku 0 odečetli 100 a získali tak `-100`, což ale odpovídá roku 101 př. n. l. Vizualizace pak vyvolá dojem, že první století př. n. l. začalo již v tomto roce.

Problematicke *korekce roku nula* se bude později věnovat kapitola týkající se implementace.



Obrázek 4.1: Časová osa zobrazující století a) při úpravě 1. st. př. n. l. na trvání 99 let, b) bez úprav

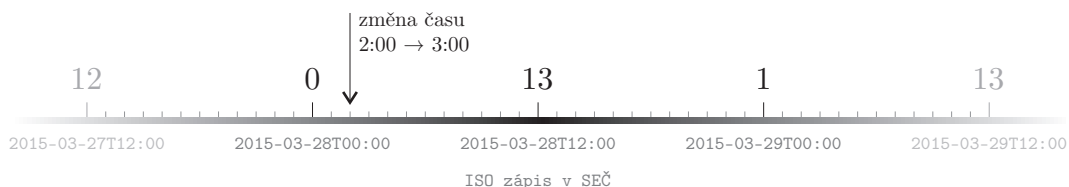
⁶Dle normy vyžaduje vyhrazení více číslic pro zápis roku (YYYYY) dohodu mezi stranami, které budou formát používat [9].

4.2.2 Letní čas

Potíže s vytvářením časové osy nezpůsobuje jen zmíněný problém s rokem nula, ale rovněž zavedení letního času, k němuž vedla snaha o větší využití světla za účelem úspory energie. Na území České republiky se pravidelně používá od roku 1979 [10].

Lze očekávat, že nástroje pro práci s časovými daty nebudou brát v potaz období, v němž se letní čas (středoevropský letní čas, dále jen SELČ) používal a kdy ne. Navíc se jeho užití liší i časovými pásmy, či dokonce jen zemí (např. v Německu a Británii akceptovali SELČ v jiných obdobích, než kdy byl používán v Čechách). Pokud se tedy podíváme blíže na několikrát zmiňovaný rok nula, kdy SELČ neexistoval, nástroje pro práci s časem nám budou i tak vracet hodnoty závislé na dnešních standardech časových pásem.

Ačkoliv přechod z běžného středoevropského času (SEČ) na SELČ fyzicky počítání času nezmění, tj. nedojde k „zahazení“ hodiny, způsobí na časové ose změny v interpretaci. Dílek, který by byl za normálních okolností označen jako 2. hodina ranní, nástroje přeznačí na 3. hodinu ranní. Při přiblížení na úroveň jednotlivých hodin tato skutečnost nehraje roli, pokud se však oddálíme na úroveň, kdy jeden dílek osy reprezentuje např. 12 hodin, odhalíme nepříjemný posun v popisu dílků, tento případ popisuje obrázek 4.2.



Obrázek 4.2: Ukázka posunu zapříčiněného změnou ze SEČ na SELČ

4.2.3 Přesnost datování

Problematika této oblasti úzce souvisí s charakterem historických dat popisovaným v kapitole 3.2. Definice 3.3 a 3.4 v souvislosti se způsobem datování záznamů zmiňují *libovolně velké časové jednotky*, za něž mohou být považovány například roky, staletí, ale i dny, hodiny či dokonce minuty. ISO norma [9], již se zabývala i kapitola 4.2.1, nám však neumožňuje popsat datum částečně, tj. třeba použitím výhradně roků. Prikazuje zaznamenávat datum v celé jeho podobě (případně doplněné o čas), což je samozřejmě i z pohledu aplikační logiky vhodnější – vyžaduje se jednotný formát datování pro všechny historické záznamy.

Zde tedy narážíme na potíže. Jestliže nám ISO norma i logická stránka aplikace prikazují zaznamenávat veškerá data a časy jednotně, jak odlišíme událost, o níž víme pouze to, že nastala v roce 1918, a záznam z konkrétního dne, například

1. ledna 1918? Hlavní problém představuje zápis samotného roku 1918. Norma vyžaduje uvedení měsíce a dne, ale ten neznáme. Použijeme tedy první leden, nebo den, jenž připadá na polovinu roku?

Ať už se přikloníme k první nebo druhé variantě, v obou případech se nám snadno stane, že některé události najednou budou disponovat stejnou časovou hodnotou, byť vybrané z nich datujeme s roční přesností a jiné zas s přesností na jednotlivé dny.

Druhou možností, jak k tomuto problému přistupovat, je uchovávat ve vrstvě modelu pro momentovou entitu popsanou pouze rokem začáteční i koncové datum. Tyto dvě hodnoty uložíme interně pro účely aplikace a navenek dále budeme propagovat pouze jediný časový údaj. Abychom pak zpětně získali informaci o tom, v jakých jednotkách datum formátovat, musíme zavést *přesnost data a času*. Pro snazší porozumění této alternativě uvádí tabulka 4.1 několik příkladů.

vnější formát		začátek přesnost	ISO	konec přesnost	ISO
M	1918	rok	1918-01-01T00:00	–	1918-12-31T23:59
I	1918–1919	rok	1918-01-01T00:00	rok	1919-12-31T23:59
M	2. ledna 1918	den	1918-01-02T00:00	–	1918-01-02T23:59
I	1917–5. ledna 1918	rok	1917-01-01T00:00	den	1918-01-05T23:59
M	20. století	století	1901-01-01T00:00	–	2000-31-12T23:59

Tabulka 4.1: Příklady uchování informací o datu odlišně časově zasazených záznamů (*M* – momentová entita, *I* – intervalová entita)

Z tabulky snadno vyčteme, že pro momentové entity sice uchováváme dvě data, ale přesnost uvádíme jen pro jedno. Zde totiž není potřeba zaznamenávat informaci o přesnostech obou časových údajů, protože jak koncové, tak i počáteční datum spadají do téhož časového období, které bude ve výsledku vyjádřeno vnějším formátem, tj. formátem prezentovaným uživateli (tento případ ilustruje první řádek tabulky). Toho, že na základě popsaného principu bude každá entita obsahovat dva časové údaje vymezující její (být jen přibližné) trvání, využijeme právě při vizualizaci v časové ose – jejím způsobem a dalšími podrobnostmi se zabývá kapitola týkající se implementace a návrhu aplikace.

4.2.4 Entita typu *místo*

Kapitola 3.2 popisuje 4 základní typy entit na základě toho, jaké historické záznamy reprezentují. Součástí tohoto výčtu je rovněž typ *místo*, jenž by měl fungovat jako reprezentace každé lokace, která se v historii vyskytla a na niž se relacemi vážou jiné entity, např. osoby či události.

Budeme-li uvažovat, že záznamy zobrazujeme v časové ose jako obdélníky různé šířky v závislosti na jejich trvání, narazíme právě u typu *místo* na problém. Většina historických lokací existovala po stovky let, nebo dokonce stále existuje. Pokud je tedy budeme chtít zmíněným způsobem reprezentovat, musíme v jednom

časovém okamžiku zobrazit spoustu takových záznamů. Ačkoliv předpokládáme, že výsledná aplikace bude pracovat s prioritami jednotlivých záznamů, pravděpodobně i tak nedokáže jejich počet snížit na takový, aby uživateli ukázala vše podstatné při zachování přehlednosti.

Jestliže záznamy reprezentující místo doplníme o GPS souřadnice (což lze očekávat), získáme novou možnost vizualizace – kombinaci časové osy a mapy. V takovém případě popisovaný problém zcela eliminujeme, protože pro zobrazení entit použijeme terčíky umístěné na odpovídající poloze v mapě. Při změně vizualizovaného období pak pouze upravíme viditelnost terčků historických oblastí. Tento přístup ale bohužel nelze uplatnit pro zbylé typy entit, tj. pro *osobu*, *událost* a *objekt*, už jen proto, že nemusí obsahovat informaci o GPS souřadnicích.

5 Analýza požadavků

Následující seznam požadavků byl sestaven pouze na základě konzultací s vedoucím práce a ostatními členy týmu realizujícího tento projekt, přičemž je doplněn o mé vlastní návrhy toho, jak by mohl výsledný widget vypadat či fungovat. Níže uvedené požadavky formulují vlastnosti prvotního konceptu, slouží však především jako kritéria pro případný výběr již existující knihovny – ať už pouze pro inspiraci nebo pro implementaci části této práce.

Knihovna nebo nástroj použitelný pro realizaci widgetu v rámci této práce by měly splňovat následující body.

5.1 Technické požadavky

Jazyk implementace

Uvážíme-li, že widget bude používán v rámci webového prohlížeče, a tedy musí být součástí webových stránek, pak můžeme výrazně zúžit okruh technologií, které lze pro jeho implementaci použít.

Jako první vezmeme v úvahu **Flash**, jenž byl svého času přítomen na 99 % zařízení používaných k prohlížení webu. Dominantní pozici si vydobyl díky absenci konkurence a tomu, že v porovnání s tehdejší kombinací HTML a JavaScriptu se osvobodil od různých omezení [11]. Takovou podporou se však nemůže pochlubit na mobilních zařízeních, která se stala majoritním bodem pro přístup k webu [12].

Novější obdobu zmíněné technologie představuje **SilverLight** vytvořený společností Microsoft. I zde se ale jedná o zásuvný modul, a jeho podpora tak není stoprocentní.

Z předchozích odstavců vyplývá, že použití HTML spolu s JavaScriptem skýtá nějaká omezení. Taková ovšem byla situace před několika lety, přesněji předtím, než organizace W3C oficiálně ohlásila dokončení standardu HTML5 (v říjnu 2014) [7] a CSS3 (postupně v letech 2011–2012). HTML5 spolu s JavaScriptem nabízí spoustu nových možností interakce, a eliminuje tak potřebu technologií, jako jsou výše zmíněný **Flash** nebo **Silverlight**. HTML5 rovněž nově disponuje tzv. *canvasem*, jenž umožňuje přímo v prohlížeči programově vykreslovat rastry.

Z úvodu práce plyne, že uživatel bude mít možnost zobrazit vztah vybraného záznamu a jeho okolí. Pokud si představíme vizualizaci chronologicky uspořádaných dat jako časovou osu, pak zřejmě vztahy můžeme zobrazit pomocí linek propojujících jednotlivé záznamy na ose. K tomu lze využít zmiňovaného canvasu. Měli bychom ovšem vzít v potaz, že zobrazení vztahů pravděpodobně budeme chtít stylovat jako ostatní prvky widgetu a rovněž s ním bude nutné manipulovat. Chceme tedy, aby se vizualizace vztahů mezi záznamy chovala jako ostatní prvky,

aby byla také součástí DOM. Pro takový případ se přímo nabízí SVG – XML popis vektorové grafiky, který lze vložit do HTML stránky.

Shrneme-li předchozí odstavce, výchozím požadavkem na technické zpracování bude implementace pomocí **HTML**, **Javascriptu** a **SVG**. V dalších odstavcích tyto jazyky doplní ještě datový formát **JSON** užitý pro přenos informace mezi datovou vrstvou a widgetem.

Responzivní zobrazení

Widget by měl být schopen reagovat na změnu rozměrů zobrazovacího zařízení a stejně tak i na jeho rozdílnou hustotu pixelů. Jde především o to, aby výsledek této práce mohl být umístěn kamkoliv do stránky – ať už do celé její plochy nebo jen vybrané části – a přitom si zachoval vlastnosti své grafické podoby.

Formát dat

Jeden z nejdůležitějších požadavků představuje formát dat, v němž bude widget přijímat informace k vizualizaci – tedy historické záznamy.

Vzhledem k tomu, že aplikace sama nebude mít přímý přístup k databázi, ale informace z ní získá přes mezivrstvu (která je předmětem jiné diplomové práce), musí být stanoven formát sloužící právě k výměně dat mezi aplikací a zmíněnou mezivrstvou.

Jako prostředník mezi widgetem a databází poslouží server. Vezmeme-li v potaz technologie zvolené pro implementaci widgetu, nabízí se jako ideální formát **JSON**. Už jen díky tomu, že vychází z JavaScriptu, a je tak i snadno zpracovatelný. Alternativou by mohlo být užití XML, které lze rovněž poměrně snadno zpracovat JavaScriptem. Jeho nevýhodou je však podstatně větší velikost (oproti JSON například kvůli syntaxi uzavírajících značek), tím pádem i větší velikost dat přenášených mezi serverem a widgetem [13].

5.2 Funkční požadavky

Paralelní zobrazení záznamů

Lze předpokládat, že rozdělení historických záznamů pouze na události a osoby nebude stačit. Existuje větší množství základních typů, do nichž můžeme data roztřídit. Při analýze historických záznamů zcela jistě odhalíme množinu označující místa nebo předměty.

Chceme-li, aby se uživatel v časové ose snadněji orientoval, musí widget podporovat oddělené zobrazení jednotlivých typů záznamů. Toho docílíme vizualizací dat v souběžných pásech, kde každý pás slouží pouze pro vybraný typ.

Chronologické uspořádání

Základním požadavkem, který zajistí to, že data budou vizualizována tak, jak pravděpodobně uživatel očekává, je jejich chronologické uspořádání v čase. Vzniká v podstatě automaticky, má-li vizualizace fungovat jako časová osa.

Vizualizace vztahu

Předchozím bodem by splňovala tato práce zadání jen z části. Nemá za cíl pouze uspořádat historické záznamy v čase, ale rovněž musí uživateli umožnit procházení vazeb mezi nimi. Protože není reálné zobrazit všechny vztahy mezi daty současně, umožní widget uživateli procházet aspoň jejich podmnožinu, a sice ty vazby, jež jsou spojeny se zvoleným (označeným) záznamem.

5.3 Požadavky na uživatelské rozhraní

Navigace

Uživatel, který pracuje s časovou osou, samozřejmě musí mít možnost měnit parametry jejího zobrazení. Potřebuje upravovat úroveň přiblížení a také časový rozsah výřezu osy, který pozoruje, tzv. *viewportu*.

Jednu z možností představuje ovládání pomocí hardwarových zařízení, tj. myši nebo klávesnicí. V případě myši příkazují konvence použít pro přiblížování rotační tlačítko a pro posun pak metodu drag&drop. U klávesnice lze očekávat, že změna úrovně přiblížení proběhne při stisku kláves + a -, zatímco posun osy zajistí kurzorové klávesy.

Widget by měl ale nabídnout i navigační tlačítka pro případ, kdy uživatel z nějakého důvodu nechce používat klávesnici a myš.

Filtry

Při tak velkém množství dat, které bylo několikrát zmiňováno v předcházejících odstavcích, můžeme očekávat, že bude chtít uživatel data filtrovat. Filtr nebude přímou součástí widgetu, protože jeho účelem je pouze data zobrazit a umožnit jejich procházení. Bude ale napojen na datový zdroj, z něhož položky k vizualizaci čerpá. Filtry jsou tedy spíše předmětem týkajícím se datového zdroje, zatímco widget musí zajistit to, že při změně datového zdroje dojde i k jeho přizpůsobení. V praxi může jít o to, že uživatel upraví filtry dat do takové podoby, kdy se změní počet pásů zobrazujících jednotlivé typy záznamů. Widget na to musí samozřejmě reagovat, tj. musí poskytnout odpovídající metody.

Z výše uvedeného plyne, že filtry budou součástí HTML stránky obsahující widget časové osy. Je tedy zcela na autorovi, jak formuláře filtrů navrhne, pouze pak k jejich aplikaci musí použít metody datového zdroje a widgetu.

Příklady filtrů

- Uživatel zvolí pouze jeden typ záznamů, o který se zajímá.
- Vybere časový rozsah období, které chce studovat.
- Stanovením hledaných hodnot pro vybrané atributy protřídí záznamy jen na ty, které splňují okruh jeho zájmů.

6 Analýza existujících řešení

Na základě požadavků stanovených v kapitole 5 lze již provést rešerši dostupných hotových řešení. Vzhledem k zvolené technologii (JavaScript a HTML) se okruh dostupných produktů zmenšil pouze na několik zástupců, které detailněji zkoumá tato kapitola.

6.1 TimelineJS

TimelineJS sice už kvůli tomu, že jde o službu a nikoliv o knihovnu, není možné použít, ale její podoba a funkce jsou velmi inspirativní. Uživatel této službě předá například jen tabulku vytvořenou v online nástrojích Google a ta nad ní vygeneruje časovou osu. Mimo obyčejné tabulky lze použít i formáty pro pokročilejší uživatele, jako je JSON, nebo ostatní webové služby a zdroje (Twitter, Flickr, Vimeo, ...).

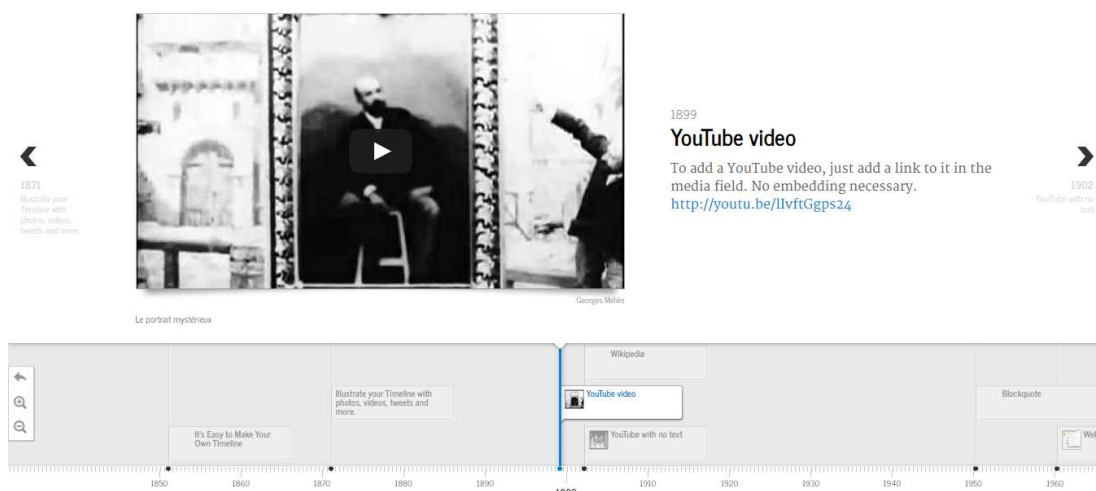
TimelineJS je nepochybně skvělým pomocníkem pro běžné uživatele, kteří chtějí zobrazit svá videa nebo fotografie chronologicky v čase, či pro zpravodajské servery k mapování vybrané skupiny událostí. Pro účely této práce však není použitelný, nedovoluje do zobrazení nijak zasahovat, načítat data průběžně z externého zdroje apod.

Poznátky

Popisovaná služba umožňuje přidat do objektu reprezentujícího záznam v časové ose fotografii (obrázek 6.1). To může uživateli usnadnit orientaci, na druhou stranu ale takový objekt zabírá v prostoru osy větší plochu. TimelineJS nijak neřeší překrývání těchto objektů, pouze je při podržení kurzoru nad jejich plochou přeneso do popředí. Pokud by se tedy vyskytlo více záznamů v krátkém časovém úseku, začne osa ztrácet svoji přehlednost.

Při testování bylo odhaleno, že úroveň přiblížení není možné měnit pomocí rolovacího tlačítka myši ani klávesami + a -. Naopak uživatelsky přívětivou možností je procházení záznamů prostřednictvím kurzorových kláves. Stiskem některé z nich se osa neposune o staticky danou vzdálenost (časový úsek), ale přesune svůj střed k začátku další události.

Vydavatel	North Western University Knight Lab
Domovská stránka	http://timeline.knightlab.com
Licence	Mozilla Public License 2.0
Poslední vydaná verze	2.35.6
Vydána dne	25. března 2015



Obrázek 6.1: Náhled služby TimelineJS

6.2 Dipity

Dipity funguje na podobném principu jako TimelineJS, narozdíl od něj ale umožňuje vytvoření časové osy pouze registrovaným uživatelům. Ačkoliv na své domovské stránce nabízí ukázkou toho, jak osa vygenerovaná touto službou vypadá, neposkytuje mnoho dalších informací.

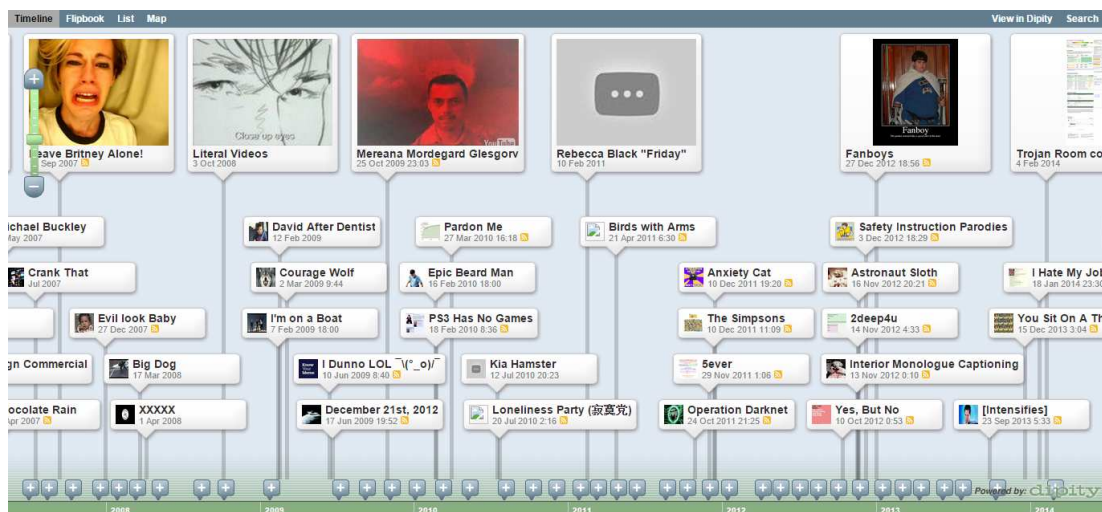
I zde jde o užitečnou pomůcku pro uživatele, kteří potřebují jednorázově vizualizovat svá data. Oproti předchozí alternativě však výrazně zaostává svým vzhledem (obrázek 6.2).

Poznátky

Díky široce využitému prostoru může Dipity vkládat do osy více informací – v ukázkovém příkladu videa, obrázky a datum. Po klepnutí na záznamy zobrazí lightbox, v němž prezentuje detailní informace o zvolené položce. Nevýhodou tohoto zobrazení ale je, že lightbox se nachází přímo v časové ose – je jejím potomkem.

Stejně jako v předchozím případě i Dipity nereaguje na kolečko myši a stisk kláves pro přiblížení. Oproti TimelineJS ale při použití navigačních tlačítek pro změnu úrovně přiblížení uživatele informuje o délce zobrazeného časového rozmezí. Podobně jako předchozí služba i tato dovoluje procházet záznamy pomocí kurzorových kláves, bohužel ale nemění pozici osy v závislosti na zobrazeném záznamu.

Vydavatel	Underlying, Inc.
Domovská stránka	http://www.dipity.com
Licence	vlastní podmínky užití (není distribuováno)



Obrázek 6.2: Náhled služby Dipity

6.3 Timeglider

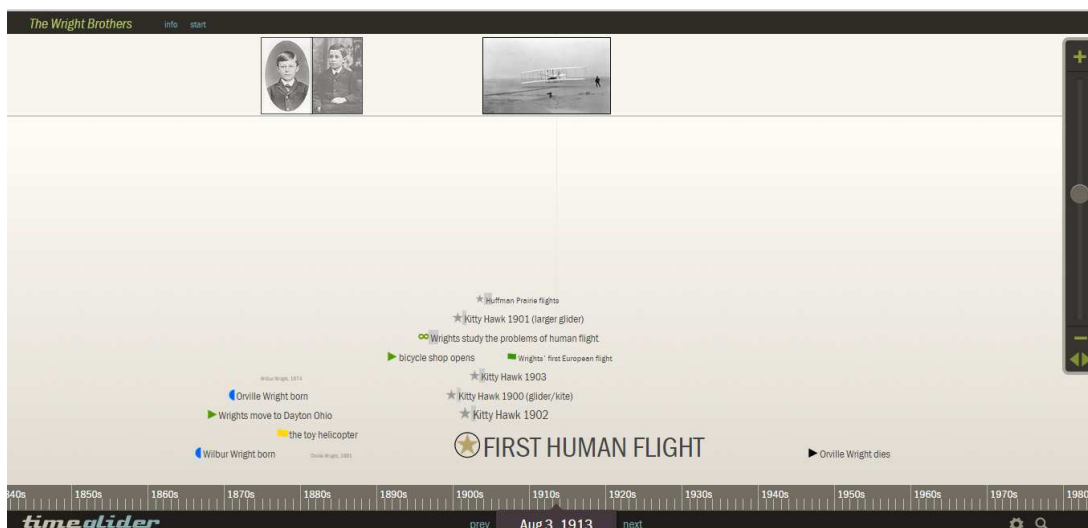
Timeglider se na první pohled sice prezentuje jako služba, avšak zároveň nabízí svoji časovou osu formou jQuery widgetu. Ten poskytuje přístup k základním metodám, jako je přidání záznamu nebo přiblížení. Součástí widgetu ale není přímá podpora načítání událostí pomocí technologie AJAX. Oproti předchozím variantám je Timeglider evidentně přizpůsobitelnější a není určen pouze běžným uživatelům.

Poznátky

Timeglider dovoluje uživateli využít celou plochu viewportu prohlížeče a získat tak spoustu prostoru pro zobrazení dat (obrázek 6.3). Stejně dobře lze ale widget umístit i do HTML prvku s omezenou velikostí. Díky dostatku místa nedochází k překryvu záznamů; pokud se dvě události časově prolínají, umístí je nad sebe.

Narozdíl od předchozí konkurence lze v případě tohoto widgetu použít kolečko myši pro změnu úrovně přiblížení. Timeglider rovněž pracuje s prioritou záznamů a podle ní rozhoduje, zda se má položka ještě zobrazit, či nikoliv.

Vydavatel	Mnemograph LLC
Domovská stránka	http://timeglider.com
Licence	vlastní podmínky užití
Cenový program	nekomerční (zdarma) omezený komerční (500 USD) OEM/SaaS (3 000 USD + 300 USD ročně)
Poslední vydaná verze	1.0.3
Vydána dne	28. března 2015



Obrázek 6.3: Náhled služby a jQuery widgetu Timeglider

6.4 vis.js

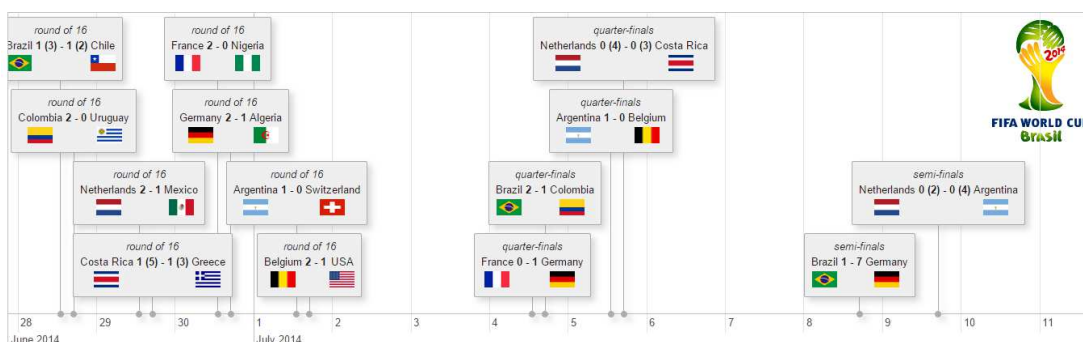
vis.js je knihovna, která nabízí širokou škálu vizualizací pro webové stránky; mezi jednu z nich patří i časová osa. Disponuje jednoduchou grafikou, takže působí zcela přehledně. vis.js již nelze považovat za alternativu pro úplné laiky. Použití této knihovny předpokládá znalost Javascriptu.

Poznátky

Jako první z testovaných produktů umožňuje vis.js přidávat do časové osy pásy shlukující položky se společnými vlastnostmi. Speciální vlastností této knihovny je podpora inline editace záznamů a šablony pro vykreslování jednotlivých událostí, které mohou obsahovat i HTML data (obrázek 6.4).

Domovské stránky tohoto widgetu nabízí řadu ukázek jeho použití a přizpůsobení doplněné o (ve srovnání s ostatními) rozsáhlou dokumentaci.

Vydavatel	Almende B. V.
Domovská stránka	http://visjs.org
Licence	Apache 2.0 nebo MIT
Poslední vydaná verze	3.11.0
Vydána dne	5. března 2015



Obrázek 6.4: Náhled widgetu vis.js

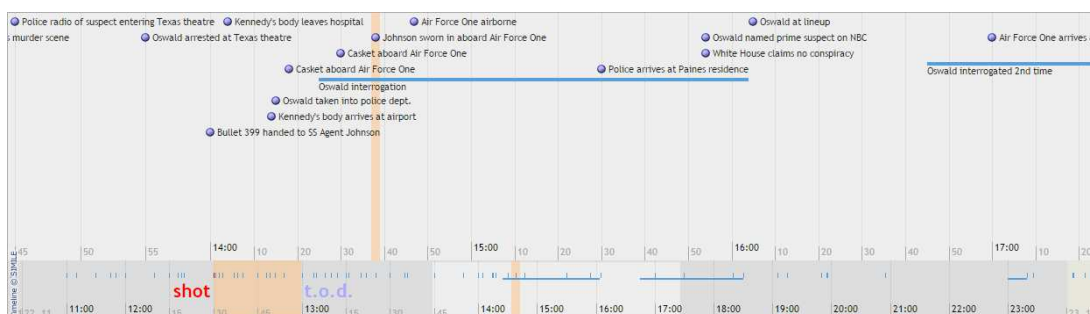
6.5 Timeline (SimileWidgets)

Timeline z balíku SimileWidgets (obrázek 6.5) je starší knihovnou, jejíž vývoj skončil poslední verzí v roce 2009. Od té doby neprodělala žádné změny a i její podpora pravděpodobně zanikla.

Poznátky

Již při prvních pokusech o testování byly odhaleny problémy znemožňující její úspěšné zprovoznění. Knihovna se pokoušela asynchronně načíst externí JS soubor (umístěním očekávaný na jejím domovském serveru), který však neexistoval (požadavek selhal se stavovým kódem 404). Zároveň do DOMu HTML stránky vkládala rámec (iframe), jenž se odkazoval na rovněž neexistující soubor. Dalším nedostatkem byl problém s kódováním – ačkoliv Timeline disponuje lokalizací do češtiny a automaticky ji podle prostředí použije, vybrané české znaky nezobrazí správně.

Vydavatel	MIT and Contributors
Domovská stránka	http://www.simile-widgets.org/timeline
Licence	BSD
Poslední vydaná verze	2.3.0
Vydána dne	březen 2009



Obrázek 6.5: Náhled widgetu z balíku SimileWidgets

6.6 Shrnutí

Průzkum ukázal, že produkty TimelineJS a Dipity lze vyřadit z možných řešení, protože jde pouze o webové služby, nikoliv použitelné knihovny. Ani Timeline z balíku SimileWidgets nezahrneme mezi potenciálně použitelné knihovny, protože od roku 2009 není dále vyvíjena a jak ukázaly testy, obsahuje několik chyb. Z uvedených kandidátů tedy zůstávají vis.js a Timeglider.

Obě knihovny splňují velkou část požadavků zmíněných v kapitole 5, ne ale všechny. U Timeglider nelze příliš ovlivnit parametry zobrazení. Jeho rozhraní poskytuje jen základní metody pro obsluhu widgetu, neumožňují však příliš zasahovat do jeho samotné funkčnosti. Naproti tomu vis.js nabízí širokou škálu možností, jak widget přizpůsobit či ovládat, a tak lze tuto knihovnu považovat za prakticky vyhovující. Nemá ale prostředky k tomu, aby jednotlivé záznamy propojila – vizualizovala vztahy mezi nimi. Tato vlastnost je stěžejní pro popisovaný projekt, proto v rámci této diplomové práce vznikne nový widget. Z velké části bude čerpat inspiraci ze zmíněných kandidátů, avšak rozšíří popsanou funkčnost o zbývající popsané požadavky.

7 Návrh nového widgetu

Žádné z existujících řešení popsaných na předchozích stránkách nesplňovalo podmínky uvedené v kapitole 5. Především však ani jedno nenabízí možnost vizualizace vztahů mezi historickými záznamy, jež je pro tento projekt klíčová. Z toho důvodu vznikne v rámci diplomové práce nový ovládací prvek, který bude čerpat inspiraci z existujících produktů, a navíc zobrazená data obohatí i o vizualizaci vztahů mezi nimi.

7.1 Obecná charakteristika

7.1.1 Rozsah projektu

Widget vyvíjený v rámci této diplomové práce umožní uživateli procházet historická data chronologicky uspořádaná v časové ose. K dispozici budou standardní prostředky navigace a interakce, jako je posun zobrazeného rozsahu, přiblížení či oddálení (tj. změna používaného měřítko časové osy). Oproti jiným ovládacím prvkům z této kategorie bude produkt této práce disponovat také vizualizací vztahů mezi historickými záznamy, a to na vyžádání uživatele (klepnutím na vybraný záznam).

Cílem této práce není vytvořit widget s kompletní funkcionalitou, nýbrž nastínit, jak takový prvek může fungovat, na jaká úskalí lze při jeho implementaci narazit, a tyto informace doplnit o funkční kostru, která splňuje požadavky uvedené v kapitole 5. V samotném závěru práce se pak čtenář dozví, o jakou další funkcionalitu by v budoucím vývoji mohl být widget obohacen.

7.1.2 Kontext widgetu

Výsledný widget bude implementován pro webové stránky či aplikace a jeho použití takřka na ničem nezávisí. Uživatel však musí zajistit zdroj dat dle později popsané specifikace v kapitole 7.8 tak, aby je mohl ovládací prvek osy v pořádku načítat. Souběžně s touto vznikají další dvě práce, jejichž vydání je plánováno v roce 2015 a na nichž tato závisí. Konkrétně se jedná o:

- MERUNKO, D. *Generování a vizualizace časové osy*. ZČU, Plzeň.
- HRBÁČEK, D. *Zpracování časových údajů pro jejich vizualizaci*. ZČU, Plzeň.

V jejich rámci je vyvíjena databáze spolu s aplikací, jež vytváří pro časovou osu potřebný kontext, tedy načítá a předzpracovává informace k použití ve widgetu.

7.1.3 Omezení návrhu a implementace

Vývoj widgetu je časově omezen termínem odevzdání diplomové práce a v jeho rámci vznikne pouze kostra výsledného widgetu. Zůstává zde však otevřena možnost, že vývoj bude pokračovat i po jejím obhájení.

Návrh a z něho vycházející implementace jsou omezeny především nekompatibilitou webových prohlížečů. Vzhledem ke komplexnosti widgetu zde zřejmě dojde k problémům nejen se zobrazením, ale také se samotným ovládáním prvku ve vybraných verzích prohlížečů. Podrobněji o tomto omezení pojednává následující poznámka.

Poznámka ke cross-browser kompatibilitě

Tato práce se věnuje tomu, jakým způsobem lze reprezentovat historická data ve webovém prohlížeči, a doplňuje popsané alternativy widgetem, který tyto principy implementuje, a demonstruje je tak v praxi. Vzhledem k tomu, že jde o první verzi tohoto projektu, nezaručuje se, že bude widget kompatibilní se všemi vydanými prohlížeči, a není ani cílem této práce 100% cross-browser kompatibilitu zajistit.

Výsledná aplikace není webovou stránkou, nepředpokládá se (aspoň v této její verzi) veřejný přístup z libovolného prohlížeče, a proto bude její implementace během vývoje testována prioritně v prohlížeči **Chrome** (63,9 % uživatelů k dubnu 2015 [21]), jenž je v současnosti nejpoužívanější aplikací pro procházení webových stránek. Nevylučuje se však, že bude aplikace zpětně kompatibilní i s jinými prohlížeči, např. **Mozilla Firefox** (21,6 % uživatelů k dubnu 2015 [21]) – tuto skutečnost ověří závěrečné testování kompatibility.

7.2 Demo aplikace

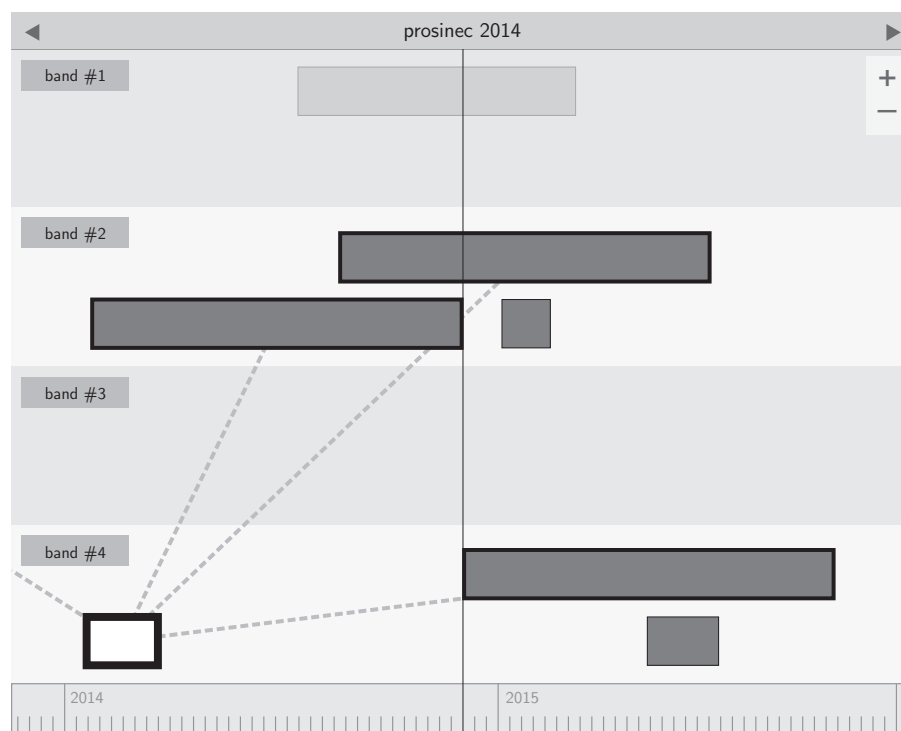
Součástí této práce bude aplikace, která demonstruje použití widgetu. Vlastní produkt sice představuje pouze ovládací prvek časové osy, ale k prokázání jeho použitelnosti v praxi musí být zasazen do aplikace, jež obstará externí záležitosti. Mezi ty můžeme zařadit například připojení ke vzdálenému serveru, reakci na kliknutí na entitu uvnitř widgetu a podobně. Časovou osu lze použít i samostatně, uživatele však ochudíme o pohodlnější získávání informací.

Z předchozího odstavce plyne, že widget neposkytne sofistikované prostředky pro prezentaci detailních informací spojených s entitami, dovolí ale vývojáři registraci posluchačů příslušných událostí, které osa při manipulaci s ní generuje, a při jejich vzniku veškerá data z entity účastníci se události předá obsluze. Ta má pak možnost je vizualizovat prostřednictvím *lightboxu*¹, zobrazit je v postranním panelu aplikace a podobně.

¹ovládací prvek zobrazující obrázek nebo galerii obrázků, popř. jiná data (např. HTML obsah) v prvku webové stránky překrývajícím její původní obsah

7.3 Uživatelské rozhraní

Widget časové osy se omezuje na vizualizaci historických dat chronologicky v čase, přičemž poskytuje základní možnosti, jak měnit zobrazené časové rozmezí, filtrovat prezentovaná data a jak mezi nimi zkoumat vazby. Uživatelské rozhraní odpovídající tomuto popisu a stejně tak požadavkům kapitoly 5.3 demonstruje obrázek 7.1.



Obrázek 7.1: Návrh uživatelského rozhraní widgetu časové osy

7.4 Značení parametrů

V následujících kapitolách se budeme setkávat se vzorci, které popisují, jakými způsoby probíhá výpočet pozic a rozměrů objektů. Z toho důvodu zavedeme značení podle tabulky 7.1.

7.5 Úrovně přiblížení, transformace

Časová osa umožňuje svůj obsah přibližovat a oddalovat, tedy měnit rozsah období, jež zobrazuje. Pro tento účel definuje několik úrovní přiblížení, které disponují následujícími atributy:

- doba vyjádřená jedním pixelem d_{px} ,
- doba, která představuje jeden dílek hlavní osy d_{maj} ,
- doba, která představuje jeden dílek vedlejší osy d_{min} ,
- formát pro popisek dílku hlavní osy a pro časovou lištu (kapitola 7.6.2).

Počet úrovní a jejich vlastnosti jsou zcela v rukou vývojáře, jenž widget použil. Pokud však žádné vlastní nedefinuje, použije časová osa vestavěné, jejichž popis uvádí tabulka 7.2.

Díky těmto jejich vlastnostem lze pak snadno provádět transformace z pozice vyjádřené v pixelech na časový moment a vice versa. Označíme-li časový rozsah pokrytý obálkou widgetu jako I , můžeme zavést dvě funkce:

$$T_x(t) = \frac{t - \min\{I\}}{d_{px}} \quad (7.1)$$

$$T_t(x) = x \cdot d_{px} + \min\{I\} \quad (7.2)$$

kde T_x vyjadřuje pro daný čas horizontální polohu uvnitř obálky widgetu a T_t pak čas pro danou polohu uvnitř obálky, jde tedy o inverzní funkci T_x^{-1} .

w_o	šířka objektu o
h_o	výška objektu o
x_o	horizontální pozice objektu o
y_o	vertikální pozice objektu o
e	entita / časový záznam
t_{e_s}, t_{e_e}	čas začátku a konce doby trvání entity
d_e	doba trvání entity
t_c	středový čas zobrazený ukazatelem widgetu
d_{px}	počet sekund odpovídajících jednomu pixelu při aktuální úrovni přiblížení
$item$	komponenta grafické reprezentace entity (kapitola 7.6.7)
$viewport$	komponenta průhledu (kapitola 7.6.1)
$wrapper$	komponenta obálky (kapitola 7.6.3)
$ruler$	komponenta pravítka (kapitola 7.6.4)
$band$	komponenta pásu (kapitola 7.6.5)

Tabulka 7.1: Značení parametrů a objektů

	d_{px}	d_{maj}	d_{min}		d_{px}	d_{maj}	d_{min}
1	10 let	1 000 let	100 let	14	3 hodiny	1 měsíc	1 den
2	5 let	1 000 let	100 let	15	1 hodina	14 dní	1 den
3	3 roky	1 000 let	100 let	16	30 minut	1 týden	1 den
4	1 rok	100 let	10 let	17	15 minut	3 dny	1 den
5	6 měsíců	100 let	10 let	18	10 minut	1 den	12 hodin
6	3 měsíce	50 let	5 let	19	5 minut	12 hodin	1 hodina
7	1 měsíc	25 let	1 rok	20	1 minuta	6 hodin	1 hodina
8	14 dní	10 let	1 rok	21	30 sekund	3 hodiny	1 hodina
9	1 týden	5 let	1 rok	22	15 sekund	1 hodina	15 minut
10	3 dny	1 rok	3 měsíce	23	10 sekund	30 minut	5 minut
11	1 den	1 rok	1 měsíc	24	5 sekund	15 minut	1 minuta
12	12 hodin	6 měsíců	1 měsíc	25	1 sekunda	5 minut	1 minuta
13	6 hodin	3 měsíce	1 měsíc	26	0,5 sekundy	1 minuta	15 sekund

Tabulka 7.2: Specifikace výchozích úrovní přiblížení

Analogicky pak mohou existovat funkce pro převod doby τ namísto jediného okamžiku:

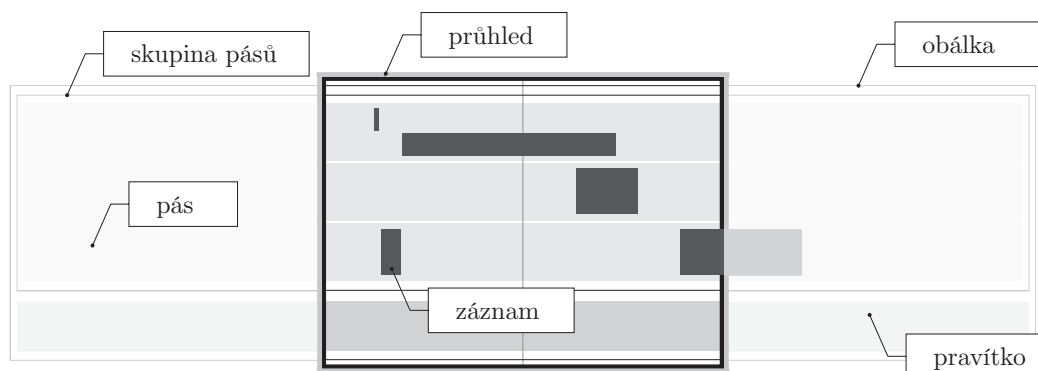
$$D_l(\tau) = \frac{\tau}{d_{px}} \quad (7.3)$$

$$D_\tau(l) = l \cdot d_{px} \quad (7.4)$$

kde D_l vrátí počet pixelů, jemuž odpovídá doba τ , a naopak D_τ pro počet pixelů l vrátí korespondující časový rozsah. Tyto funkce často využijeme při pozicování záznamů v rámci widgetu, generování pravítka a podobně.

7.6 Komponenty widgetu a jejich rozvržení

Na první úrovni lze widget rozdělit na dva základní prvky – časovou lištu (*timebar*), která souvisí s časovým ukazatelem (*time pointer*), a posuvnou obálku (*wrapper*). Časová lišta má jediný úkol, a sice informovat uživatele o tom, jaký časový úsek osa zobrazuje, kde v čase se právě nachází. Zároveň mu umožňuje pomocí navigačních šipek měnit zobrazený časový výřez. Obálka pak představuje zastřešení pro všechny komponenty widgetu, které může uživatel tahem myši posouvat – jde tedy o pravítko, samotný obsah, pásy. Následující kapitoly se zabývají specifikací jednotlivých komponent, jejich vlastností, vztahu k časové ose a funkčnosti.



Obrázek 7.2: Uspořádání komponent do vrstev a jejich zobrazení v průhledu

7.6.1 Průhled (viewport)

Jako viewport či průhled bývá v grafice označována plocha (na výstupu zobrazovacího zařízení), v níž se uživateli promítá scéna či obsah, který není možné zobrazit kompletně celý najednou. Jedná se tak o pohled hlavního hrdiny v akční hře, o výřez mapy na webovém portálu poskytujícím mapové podklady, ale i o pouhý rámeček textového pole, jehož obsah není možné zobrazit jednorázově, a tak jej musí uživatel rolovat.

I v případě časové osy jde o výřez celé vizualizace omezený velikostí zobrazovacího zařízení. Jedná se o kontejner s pevně stanovenou výškou a šířkou, který v sobě zahrnuje časovou lištu, časový ukazatel, wrapper a komponentu popisků páсů (štítky s číslováním **band #1** až **band #4** na obrázku 7.1). Způsob, jakým jsou tyto komponenty uspořádány a jak je průhled zobrazuje, ilustruje obrázek 7.2.

7.6.2 Časová lišta (timebar)

Časová lišta slouží k tomu, aby se uživatel snáze orientoval v rozsahu zobrazeném osou. Uprostřed lišty může vždy sledovat čas, který reprezentuje časový ukazatel. Tuto informaci zprostředkovává lišta ve formátu stanoveném úrovni přiblížení (kapitola 7.5). Pokud tedy uživatel sleduje ve widgetu období o délce stovek let, nepodsouvá mu lišta přesné datum, nýbrž jen století, kterým v daný okamžik prochází časový ukazatel.

Komponenta rovněž disponuje ovládacími prvky pro posun vizualizovaného rozmezí. Ačkoliv tak může uživatel učinit i pomocí klávesnice a myši (o interakci pomocí myši a klávesnice se zmiňuje kapitola 7.7.1), navigační tlačítka jsou jedinou na první pohled patrnou možností, jak časovou osu posunout. To, o jaké období se obálka s vizualizací záznamů posune, určuje jednak úroveň přiblížení a také koeficient definovaný uživatelem.

7.6.3 Obálka (wrapper)

Obálka sdružuje veškerý obsah widgetu, který lze posouvat – z podstaty časové osy plyne, že jde především o samotné pásy se záznamy a pravítko, které uživateli předává informaci o jejich časovém zasazení.

Aby uživatel viděl okamžitě při tažení osy (obálky) záznamy, které se doposud skrývaly mimo plochu průhledu, musí být s předstihem umístěny do obálky. Z toho důvodu se její šířka odlišuje od rozměrů viewportu, což orientačně ilustruje i obrázek 7.2. Ve skutečnosti by měla být šíře wrapperu přesně trojnásobná.

Vodící linky

Kapitola 7.6.4 uvádí, že součástí obálky je rovněž pravítko informující uživatele o přibližném časovém zasazení záznamů zobrazených v průhledu. Aby však osoba, která osu použije, získala větší povědomí o tom, do jakého časového úseku vybraná entita spadá, zobrazuje widget při podržení kurzoru nad záznamem vodící linky. Ty vedou od hraničních bodů entity (začátku a konce jejího trvání) až ke komponentě pravítka.

Linky jsou pozicovány absolutně vzhledem ke komponentě obálky, souřadnice x_{gl} a y_{gl} a délka h_{gl} levé vodící linky (vyznačující začátek doby trvání záznamu) se určí vztahy

$$x_{gl} = T_x(t_{e_s}) \quad (7.5)$$

$$y_{gl} = y_{band} + y_{item} + h_{item} \quad (7.6)$$

$$h_{gl} = h_{wrapper} - y_{gl} \quad (7.7)$$

a pro konec doby trvání záznamu pak zcela stejně pouze místo času začátku t_{e_s} použijeme čas konce t_{e_e} . Vodící linky doplňuje i funkce zvýraznění části pravítka (kapitola 7.6.4), kde funguje výpočet rozměrů na stejném principu.

7.6.4 Pravítko (ruler)

Pokud by časová osa nezahrnovala pravítko, jedinou možností, jak zjistit, kdy se která událost odehrála, by pro uživatele představovalo přímé vybírání záznamů a čtení u nich uvedených dat. Proto tato komponenta doplňuje v rámci obálky soubor pásů, a dává tak uživateli možnost odečítat časové informace přímo z ní.

Protože je pravítko součástí wrapperu, platí i pro něj požadavek na trojnásobně větší šířku než u průhledu. Abychom byli schopni zkontruovat jeho obsah, tedy jednotlivé dílky a popisky, musíme nejdříve zjistit, pro jaké období jej vlastně sestavujeme. To se zřejmě bude odvíjet od data reprezentovaného časovým ukazatelem, který je zároveň středem průhledu.

Generování dílků

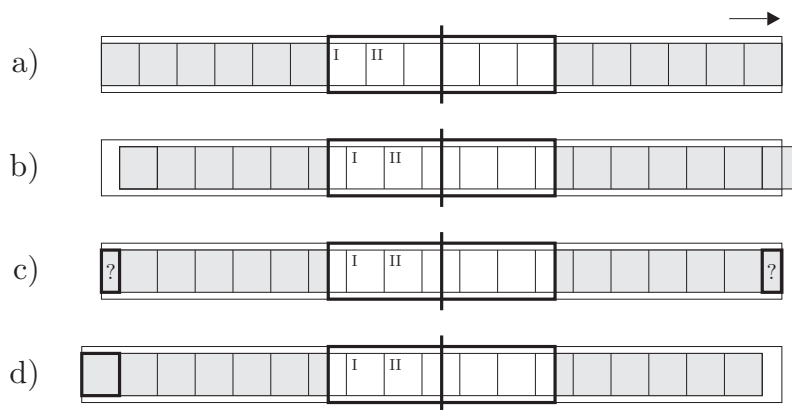
Pravítko se skládá z hlavní a vedlejší osy, přičemž jednotlivé dílky hlavní osy doplňují popisky. O jejich velikosti a časových jednotkách rozhoduje aktuální

úroveň přiblížení (kapitola 7.5). Vzhledem k tomu, že veškerá vizualizace widgetu probíhá v rámci webové stránky (HTML dokumentu), lze předpokládat, že jednotlivé dílky stejně jako celé pravítko nebudou ničím jiným než HTML prvkem v DOM. Pokud bychom chtěli striktně dodržet trojnásobnou šířku pravítka v porovnání s průhledem, mnohdy by to znamenalo, že jej musíme začít konstruovat například v polovině jednotky, a HTML prvek prvního dílku by tedy neměl být celý. Situaci ilustruje následující příklad s doprovodným obrázkem 7.3.

Mějme úroveň přiblížení takovou, že jeden dílek hlavní osy (omezíme se pouze na ni, protože na vedlejší osu lze aplikovat tentýž problém) odpovídá jednomu měsíci. V průhledu jsme schopni zobrazit současně období v délce šesti měsíců.

- Nastavili jsme časový ukazatel na přesný začátek měsíce (tj. půlnoc prvního dne), mimo průhled je tedy načteno dalších dvanáct měsíců (6 vlevo, 6 vpravo).
- Táhneme obálku doprava o polovinu měsíce, čímž měníme zobrazený časový rozsah. Na levé straně pravítka kus dílku chybí, na pravé pak přebývá.
- Narážíme na problém, jak doplnit jen zlomek dílku?

Z parametrů widgetu by samozřejmě bylo možné zjistit požadované rozměry necelého dílku a ty aplikovat. Pravděpodobně by ale tento postup musel být aplikován i na vedlejší ose – můžeme totiž předpokládat, že dojde-li k „useknutí dílku“ hlavní osy, nastane podobná situace i na vedlejší. Z toho důvodu nebudeme s velikostí dílku manipulovat, pouze upravíme jejich počet uvnitř pravítka (bod d) obrázku 7.3).



Obrázek 7.3: Stavy komponenty pravítka při posunu obálky

Korekce pravítka

Abychom mohli určit, z jakého rozsahu budeme záznamy vizualizovat, a tedy i pro jaký musíme vykreslit pravítko, potřebujeme nejdříve znát šířku průhledu –

označíme ji $w_{viewport}$. Z předešlých kapitol (7.6.3) již víme, že šířka pravítka (potažmo obálky) w_{ruler} by měla být trojnásobná v porovnání s průhledem. Využijeme-li funkce D_τ (vzorec 7.4, str. 27), jež převádí počet pixelů na dobu (počet sekund), dokážeme s pomocí času vyznačeného časovým ukazatelem (dále jen *středový čas*) t_c určit rozsah zobrazený ve wrapperu:

$$I' = \left\langle t_c - D_\tau \left(\frac{w_{ruler}}{2} \right); \quad t_c + D_\tau \left(\frac{w_{ruler}}{2} \right) \right\rangle \quad (7.8)$$

I' zde vyjadřuje rozsah před korekcí pravítka, to znamená takový, kdy by si proces generování mohl vyžádat vytvoření necelého dílku. Proto provedeme korekci – začátek i konec intervalu upravíme tak, abychom mohli místo části dílku vložit celý a přitom zachovali minimálně stejně velký rozsah pravítka, jaký byl určen vzorcem 7.8. To znamená, že začátek intervalu zaokrouhlíme na celé jednotky aktuální úrovně přiblížení směrem dolů a konec intervalu naopak nahoru. Pokud mělo dojít k začátku generování pravítka například 10. května a aktuální úroveň přiblížení definuje, že jeden dílek hlavní osy odpovídá jednomu měsíci, pak tuto hodnotu zaokrouhlíme na celé měsíce směrem dolů a dostaneme 1. května. V případě, kdy by šlo o konec intervalu, získali bychom datum 1. června. Zaokrouhlení vyžaduje následující postup:

1. Zvolíme na časové ose pevný bod, například půlnoc 1. ledna roku 1 n. l.
2. Zjistíme dobu odpovídající jednomu dílku z aktuální úrovně přiblížení.
3. Určíme, kolikrát se tato doba vejde do intervalu vymezeného pevně stanoveným bodem a zaokrouhlovaným datem.
4. Tento počet zaokrouhlíme dolů/nahoru na celé jednotky.
5. Zaokrouhlenou hodnotu zpět přičteme k pevně stanovenému bodu.

Příklad Mějme datum 23. srpna roku 2 n. l., které chceme zaokrouhlit směrem dolů. Úroveň přiblížení přiřazuje jednomu dílku odpovídající dobu jednoho měsíce. Do intervalu od 1. ledna roku 1 do 23. srpna roku 2 se tato doba vejde přibližně 20,742krát. Hodnotu zaokrouhlíme směrem dolů (20) a přičteme zpět k 1. lednu roku 1 – přičteme 20 měsíců. Získáme tak datum 1. srpna roku 2.

S použitím zaokrouhlování již můžeme přesně určit, jak bude vypadat časový rozsah widgetu přizpůsobený pravítku. Přitom budeme vycházet ze vzorce 7.8.

$$I = \left\langle \left\lfloor t_c - D_\tau \left(\frac{w_{ruler}}{2} \right) \right\rfloor; \quad \left\lceil t_c + D_\tau \left(\frac{w_{ruler}}{2} \right) \right\rceil \right\rangle \quad (7.9)$$

7.6.5 Pás (band)

Předcházející kapitoly již několikrát zmiňovaly, že historická data lze bezpochyby kategorizovat. Pokud by měl uživatel pracovat s časovou osou, která slučuje události, osobnosti i další objekty do jedné plochy, brzy by se přestal orientovat. Proto se plocha, v níž widget vizualizuje záznamy, rozděluje na jednotlivé pásy, kde každý obsahuje pouze vybraný typ historických dat.

Načtení záznamů

Každý pás si uchovává seznam všech záznamů, které do něj díky svému typu spadají. Při překreslení časové osy však nedochází k zobrazení záznamů (a jejich fyzickému vložení do DOM), protože z předchozích kapitol už víme, že velikost obálky odpovídá přibližně trojnásobku šíře průhledu. Je tedy zbytečné vkládat do DOM všechny záznamy, když nemá uživatel šanci je spatřit.

Proto při manipulaci s widgetem komponenta pásu zjišťuje, které ze zaregistrovaných záznamů má vložit do objektového modelu a zobrazit je tak uživateli. Záznam může být svou polohou v několika různých vztazích vůči časovému intervalu wrapperu – popisuje je obrázek 7.4 a následující definice.

Definice 7.1. (a) Entita e *vyhovuje* (*fits*) časovému intervalu I , kde I_s je jeho začátek a I_e konec, pokud

$$t_{e_s} \geq I_s \quad \wedge \quad t_{e_e} \leq I_e.$$

V případě, že jde o momentovou entitu, pak platí, že $t_{e_e} = t_{e_s}$.

Definice 7.2. (b) Intervalová entita e *prostupuje* (*protrudes*) do časového intervalu I , pokud

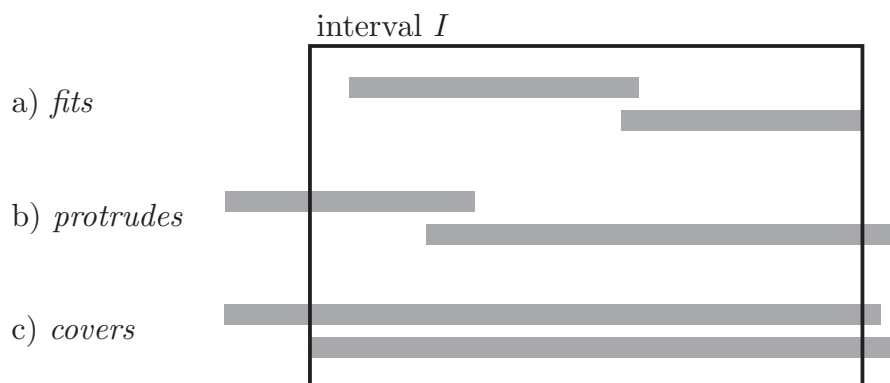
$$(t_{e_s} < I_s \wedge t_{e_e} < I_e \wedge t_{e_e} > I_s) \quad \oplus \quad (t_{e_s} > I_s \wedge t_{e_e} > I_e \wedge t_{e_s} < I_e).$$

Definice 7.3. (c) Intervalová entita e *pokrývá* (*covers*) časový interval I , pokud

$$t_{e_s} \leq I_s \wedge t_{e_e} \geq I_e.$$

Definice 7.4. Entita *je přítomna* v časovém intervalu, pokud mu *vyhovuje* nebo do něj *prostupuje* nebo jej *pokrývá*.

Komponenta pásu při vykreslení prochází jednotlivé záznamy, jež jsou k ní registrovány, a ověřuje, zda jsou podle definice 7.4 *přítomny* v intervalu aktuálně zobrazeném komponentou obálky. Pokud ano, přidá je do pásu, stanou se tedy součástí DOM.



Obrázek 7.4: Popis vztahů mezi dobou trvání záznamu a časovým intervalem

Řešení kolizí záznamů

V historických datech pravděpodobně velmi často narazíme na případ, kdy se dva a více záznamů překrývá v čase. Tyto překryvy nelze předvídat, protože pás nemá prostředky k tomu, aby předběžně zjistil, které ze záznamů vzájemně kolidují. To způsobuje především fakt, že počet záznamů přítomných v DOM se s každým překreslením mění, a navíc je závislý i na aktuální úrovni přiblížení, která jej může na základě priority entit rovněž upravovat.

Tento problém řeší *lane algoritmus*. Předpokladem pro jeho úspěšné použití je chronologické řazení záznamů. Protože pás všechny jemu příslušející entity registruje už při inicializaci časové osy, může je již v tom okamžiku seřadit, a při neomezeném počtu vykreslení pak už nebude potřeba tuto akci provádět znovu² – tím pádem proces řazení entit nijak neovlivní čas vykreslení.

Lane algoritmus

Jako *lane* označujeme jeden řádek záznamů v rámci pásu (např. řádek č. 1 na obrázku 7.5).

Mějme uspořádanou množinu $L = \emptyset$, která uchovává horizontální souřadnici pravého konce³ posledního záznamu pro jednotlivé pruhy, a proměnnou $i = 0$. Pro každou komponentu uvnitř pásu:

1. Pokud $i = |L|$, zařadíme komponentu do i -tého pruhu, do množiny L přidáme prvek $l_i = x_{item} + w_{item}$ a pokračujeme bodem 4.
2. Pokud $x_{item} > l_i \in L$, pak zařadíme komponentu do i -tého pruhu, nastavíme $l_i = x_{item} + w_{item}$ a pokračujeme bodem 4.
3. Nastavíme $i = i + 1$ a pokračujeme bodem 1.

²Ovšem pouze do okamžiku, kdy se změní data poskytnutá zdrojem. Taková akce totiž invaliduje seřazený seznam záznamů registrovaných v rámci pásu.

³tj. $x_{item} + w_{item}$

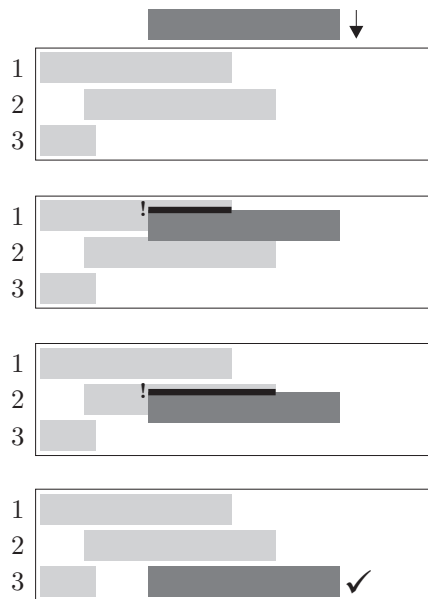
4. Nastavíme komponentě vertikální pozici $y_{item} = i \cdot h_{item}$, přičemž předpokládáme stejnou výšku všech komponent.

V případě, že si algoritmus nárokuje větší počet pruhů, než se do pásu vejde, dojde k jeho přetečení – uživatel záznamy umístěné v pruzích, které přetékají, neuvidí a časová osa na to upozorní generováním příslušné události.

Nastane-li nejhorší případ, vznikne pro n komponent stejný počet pruhů, přičemž s každou zpracovávanou komponentou budeme muset projít o jeden pruh více než u předchozí. Pro první komponentu projdeme jedním pruhem, druhá musí projít dva, třetí tři atd. Počet takových průchodů tedy bude $1 + 2 + 3 + \dots + n$. Jde o aritmetickou posloupnost, kde $d = 1$ a $a_1 = 1$, jejíž součet můžeme vyjádřit jako [22]

$$S_n = n \cdot \frac{a_1 + a_n}{2} = n \cdot \frac{1 + n}{2} = \frac{n^2 + n}{2}$$

Asymptotická složitost algoritmu odpovídá $O(\frac{n^2+n}{2}) \sim O(n^2)$, kde n je počet komponent. Naopak ve chvíli, kdy mezi komponentami nedochází k žádné kolizi, platí, že $m = 1$, a tedy $O(n)$.



Obrázek 7.5: Příklad postupu při zařazení záznamu do pásu pomocí lane algoritmu

7.6.6 Skupina pásů (band group)

Jednotlivé pásy se sdružují s historickými záznamy do skupiny, tj. do jedné komponenty, která je pak vložena do obálky. Tím zajistíme, že se při tažení pohybují všechny komponenty pásů současně a stejně dobře tak máme jistotu, že i výpočty pro transformaci času na pozici v pixelech a opačně budou v pořádku.

Tato komponenta rovněž zodpovídá za distribuci volného prostoru v průhledu mezi jednotlivé pásy. Při každé manipulaci s obsahem widgetu tedy musí skupina pásů zjistit, jaká je výška volného prostoru v průhledu a tu rozdělit mezi pevně stanovený počet pásů.

7.6.7 Položka pásu (band item)

Položka pásu představuje pouhou grafickou abstrakci reálného historického záznamu. Jde o komponentu, která je svázaná s objektem nesoucím skutečná data, a funguje tak jako jejich reprezentace – sama o sobě žádnou informaci neuchovává.

Aby se mohl uživatel mezi záznamy časové osy snáze orientovat, musí se položky pásu nějakým způsobem odlišovat. To, jakým způsobem se položka zobrazí v pásu, neurčuje přímo sama, využívá k tomu objekt, jenž na základě informací o historickém záznamu sám rozhoduje například o barvě či velikosti. Tento objekt budeme nazývat *renderer*.

Vykreslení

Renderer při vykreslování položky volí její finální grafickou podobu na základě toho, zda jde o momentovou či intervalovou entitu. Rozdíl mezi těmito dvěma typy by měl být pro uživatele patrný na první pohled. Zatímco intervalový záznam popisuje dlouhodobý stav⁴, průběh události nebo život člověka, a lze tedy vyznačit jeho trvání, momentová entita se odehrává v jednom okamžiku (byť jím může být i rok), a proto pro ni zvolí vizualizaci bodem. Stejně tak by měl uživatel grafickou podobou záznamu informovat, není-li časové rozmezí záznamu přesné (více v kapitole 4.2.3), například postupným vymizením okrajů do ztracena. Ve všech případech ale musí výslednou komponentu umístit na pozici odpovídající jejímu časovému zasazení. Toho docílí použitím funkcí ze vzorce 7.1 (str. 26):

$$x_{item} = T_x(t_{e_s})$$

U komponenty reprezentující intervalovou entitu je pak potřeba nastavit i šířku prvku reprezentujícího dobu trvání:

$$w_{dur_{item}} = D_l(d_e)$$

Pokud však entita *překrývá* interval obálky nebo do něj *prostupuje*, musí renderer její hodnoty x_{item} a $w_{dur_{item}}$ upravit:

$$x_{item} = \max\{0, x_{item}\}$$

$$w_{dur_{item}} = \min\{w_{dur_{item}}, w_{wrapper}\}$$

⁴Rozdíly mezi intervalovou a momentovou entitou podrobněji popisuje kapitola 3.2.

Vykreslení popisku




Součástí každé položky pásu je i její popisek, který uživateli poskytuje základní možnost orientace. Forma popisku závisí na podobě zbylé části komponenty, všechny následující alternativy ilustruje obrázek 7.6.

- a) Popisek nelze umístit do prvku reprezentujícího dobu trvání entity, protože $w_{label} \geq D_l(d_e)$, kde w_{label} je šířka prvku s popiskem.
- b) Popisek lze umístit do prvku reprezentujícího dobu trvání; $w_{label} < D_l(d_e)$ a jeho pozice je relativní vůči komponentě záznamu.
- c) Komponenta vizualizuje momentovou entitu, popisek se vždy nachází vedle.

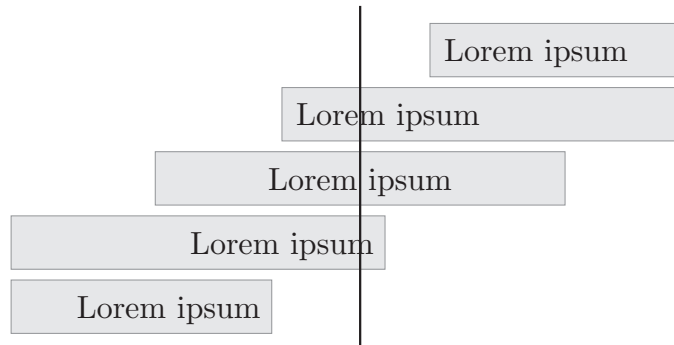
Případ b) můžeme dále rozdělit (obrázek 7.7). Protože uživatel zřejmě soustředí svoji pozornost do středu osy, tedy do okolí časového ukazatele, snaží se renderer pozicovat popisky uvnitř prvku reprezentujícího trvání entity tak, aby se maximálně blížily k centru osy. A to podle následujících pravidel:

- Pokud $x'_{item} + w_{item} < \frac{1}{2} \cdot (w_{viewport} + w_{label})$, pak $x_{label} = w_{item} - w_{label}$.
- Pokud $x'_{item} > \frac{1}{2} \cdot (w_{viewport} + w_{label})$, pak $x_{label} = 0$.
- V ostatních případech $x_{label} = \frac{1}{2} \cdot (w_{viewport} - w_{label}) - x'_{item}$,

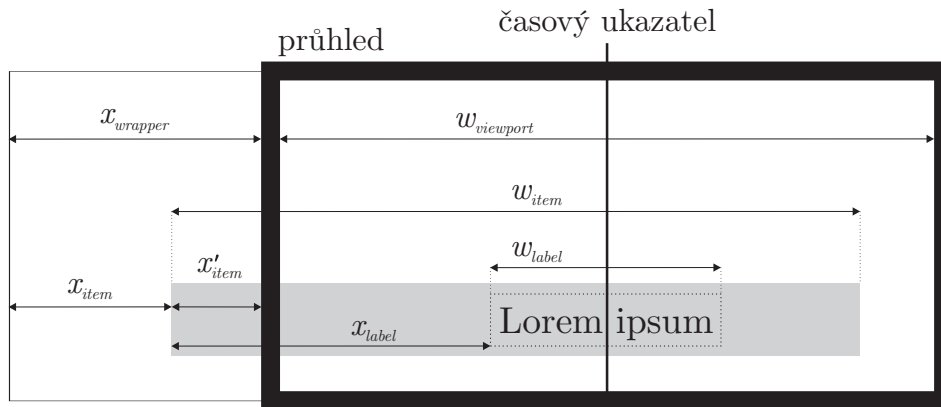
přičemž $x'_{item} = x_{wrapper} + x_{item}$ a vyjadřuje buď počet pixelů šířky komponenty, které jsou skryty (vlevo vyčnívají z průhledu), nebo počet pixelů, o něž je záznam odsazen od levého okraje průhledu. Význam všech použitých proměnných ilustruje obrázek 7.8

- a)  Lorem ipsum
- b)  Lorem ipsum
- c)  Lorem ipsum

Obrázek 7.6: Možnosti pozicování popisku komponenty záznamu



Obrázek 7.7: Centerování popisků umístěných uvnitř prvku reprezentujícího dobu trvání záznamu

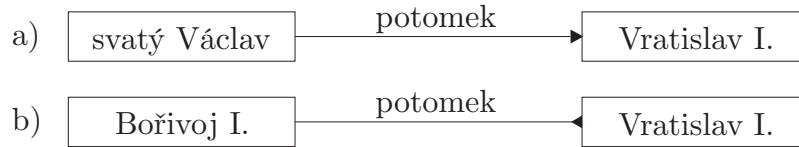


Obrázek 7.8: Ilustrace významu proměnných použitých při centerování popisku

7.6.8 Vrstva vztahů (relation viewer)

Widget časové osy vytvářený v rámci této práce je unikátní tím, že mezi vizualizovanými záznamy dokáže zobrazit i vybrané vztahy. Stejně jako entity i jejich vzájemné relace se rozlišují podle typu a obsahují název (popisek). Pokaždé v sobě nesou také informaci o tom, z kterého záznamu do kterého směřují. Díky takové charakteristice můžeme pro jejich grafickou reprezentaci použít šipku, přičemž význam vztahu budeme číst v jejím směru. Příklad naznačuje obrázek 7.9.

Zatímco případ a) čteme jako „Svatý Václav je potomek Vratislava II.“, druhou možnost, kde šipka směřuje zprava doleva, přečteme „Vratislav I. je potomek Bořivoje I.“.



Obrázek 7.9: Ilustrace odlišnosti významu vztahu při opačném směru šipky

Výběr vztahů

Komponenta vztahů nemůže současně zobrazit všechny relace mezi entitami – uživatel by zcela ztratil veškerou orientaci. Z toho důvodu relation viewer před samotným vykreslením šipek rozhoduje o jejich počtu na základě následujících podmínek.

- Zobrazí se pouze ty relace, které vychází z označené položky pásu nebo do ní vstupují.
- Komponenta záznamu účastnícího se vztahu musí být přítomna v objektovém modelu widgetu.
- Komponenta, do/z níž vede záznam z/do vybrané položky, musí být viditelná v rámci pásu, do něhož patří. To znamená, že nesmí být v pruhu, jenž přetekl (kapitola 7.6.5, část Řešení kolizí). Platí pro ni tedy, že

$$y_{item} + h_{item} \leq h_{band} ,$$

kde $band$ je nadřazená komponenta pro $item$.

Vykreslení

Splňují-li dva záznamy výše uvedené podmínky, lze mezi nimi vykreslit relaci. Vrstva vztahů se přitom snaží, aby šipka vždy směřovala do (resp. vycházela ze) středu viditelné části prvku, který reprezentuje dobu trvání. Souřadnice tohoto bodu označíme x_{citem} a y_{citem} a bude pro ně platit:

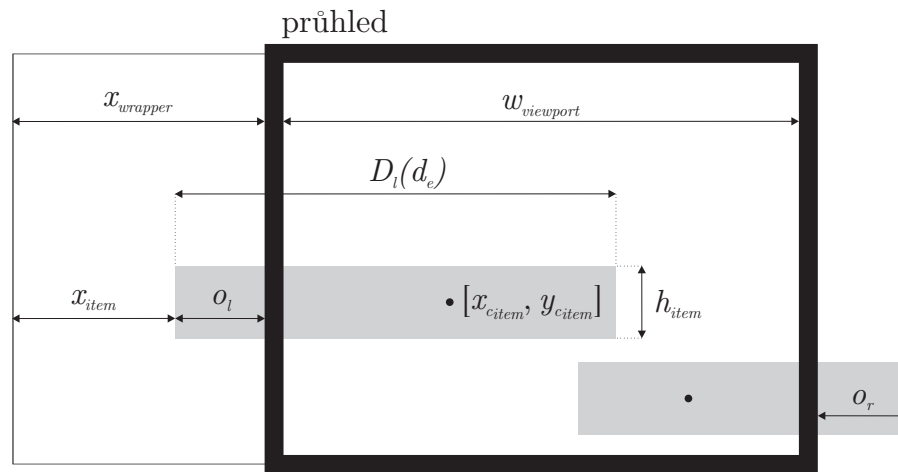
$$x_{citem} = \frac{D_l(d_e) - o_l + o_r}{2} + x_{item} \quad (7.10)$$

$$y_{citem} = y_{band} + y_{item} + \frac{h_{item}}{2} \quad (7.11)$$

kde o_l (resp. o_r) vyjadřuje velikost části prvku, která vyčnívá z průhledu vlevo (resp. vpravo) (obrázek 7.10), a určí se jako

$$o_l = \min\{0, x_{wrapper} + x_{item}\},$$

$$o_r = \max\{0, x_{item} + D_l(d_e) + x_{wrapper} + w_{viewport}\}.$$



Obrázek 7.10: Ilustrace významu proměnných použitých při hledání viditelného středu záznamu

Poznámka Převod doby trvání záznamu d_e na počet pixelů v tomto případě používáme místo w_{item} z toho důvodu, že je-li prvek reprezentující trvání entity příliš malý, její popis, který je rovněž součástí komponenty *item*, bude umístěn vedle, a tím zvýší hodnotu w_{item} . My však chceme znát viditelný střed grafické reprezentace doby trvání, nikoliv celé komponenty.

7.7 Základní interakce

Kapitola popisující základní interakce, jež lze nad widgetem časové osy provádět, uvádí některé vzorce potřebné k zajištění správného posunu osy či změny úrovně přiblížení.

7.7.1 Posun osy

Jedná se o interakci, o níž se několikrát zmiňují předcházející kapitoly a která umožňuje uživateli prostřednictvím průhledu pohodlně procházet záznamy.

Nejrychlejším a zároveň nejpřesnějším způsobem, jak měnit prohlížený časový rozsah, je posun obálky metodou *drag&drop* – jde o její uchopení stiknutím levého tlačítka myši a následné tažení doleva nebo doprava (vertikální směr v tomto případě není povolen, nemá smysl). Po upuštění komponenty pak časová osa provede operace nutné k tomu, aby uživatel viděl všechny záznamy z nově zvoleného období.

Druhou možnost představuje použití navigační šipek v oblasti časové lišty nebo kurzorových kláves. V takovém případě však uživatel přímo neovlivní veli-

kost posunu – tu stanovuje úroveň přiblížení a vývojář, jenž má možnost určit koeficient definující právě míru posunutí.

7.7.2 Změna úrovně přiblížení

Větší množství záznamů, které jsou zasazeny do stejného časového období, nelze současně zobrazit v čitelné podobě při určitých úrovních přiblížení. Z toho důvodu má uživatel možnost tuto úroveň přizpůsobit.

K přiblížení nebo oddálení může uživatel použít tlačítka + a – navigačního panelu umístěného vpravo nahoře nebo jim odpovídající klávesy. Při tomto způsobu manipulace s úrovní přiblížení nedochází k žádnému posunu, zůstává zachován původní středový čas časové osy.

K jinému chování však dochází, použije-li uživatel k přibližování kolečko myši. Tehdy totiž bere widget v potaz také pozici kurzoru, při níž k vyvolání změny úrovně došlo. Uživatel pravděpodobně očekává, že podrží-li kurzor nad konkrétním záznamem během této události, bude jej widget považovat za střed přibližování. Jinými slovy, čas, jemuž odpovídá pozice kurzoru před změnou úrovně přiblížení, by měl zůstat stejný i pro změně.

Pro zachování středu přibližování potřebujeme vědět, jakému času v ose odpovídá pozice kurzoru. Musíme tedy zjistit polohu kurzoru vůči poloze wrapperu – budeme ji označovat x_z a určíme ji jako

$$x_z = x_{event} - x_{wrapper} ,$$

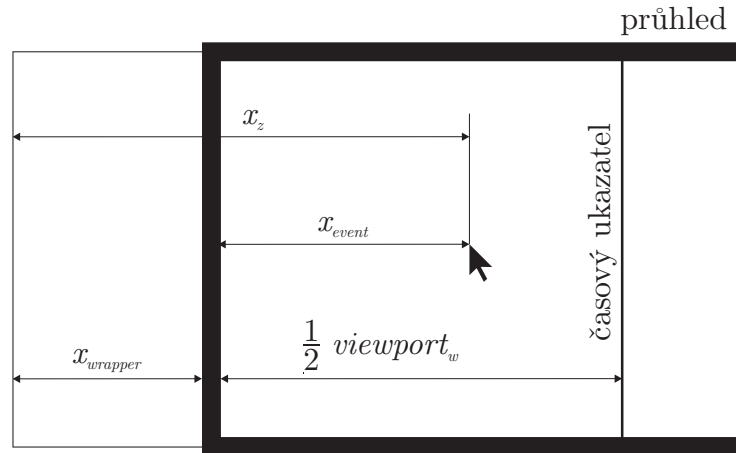
kde x_{event} je horizontální pozice kurzoru při vzniku události vztažená k umístění widgetu a $x_{wrapper}$ pak horizontální pozice obálky vzhledem k widgetu (vždy záporná) (význam proměnných ilustruje obrázek 7.11). Díky znalosti funkce T_t (vzorec 7.2, str. 26) můžeme z pozice x_z určit odpovídající čas

$$t_z = T_t(x_z) .$$

Čas odpovídající pozici, v níž vznikl požadavek na přiblížení, potřebujeme znát především proto, že samotná horizontální pozice přestane mít v rámci obálky význam ve chvíli, kdy úroveň změním – měním s ní totiž i parametr d_{px} , který měřítko osy ovlivňuje.

Po provedení operací nutných k zajištění změny úrovně přiblížení musíme nastavit nový středový čas widgetu. Pokud by zůstal stejný, entita, nad níž uživatel držel při rolování kurzor, by pravděpodobně zmizela z průhledu, protože se měřítko mohlo změnit i o desítky let. Nový středový čas t'_c získáme tak, že k času t_z odpovídajícímu pozici, ve které požadavek na událost vznikl, přičteme dobu, která koresponduje se vzdáleností v pixelech mezi x_z a časovým ukazatelem, jenž se nachází uprostřed widgetu:

$$t'_c = t_z + D_\tau \left(\frac{w_{viewport}}{2} - x_z \right) \quad (7.12)$$



Obrázek 7.11: Význam proměnných použitých ve vzorcích pro vystředění podle pozice kurzoru po změně úrovně přiblížení

Doba, kterou vrátí funkce D_τ (vzorec 7.4, str. 27) jako výsledek, může být i záporná, protože D_τ zachovává znaménko svého argumentu, tedy předaného počtu pixelů. K takové situaci dochází, pokud uživatel inicioval přiblížení s kurzorem umístěným napravo od časového ukazatele.

7.7.3 Zaostření

Zaostření je speciální funkcí, která kombinuje výše popsaný posun osy a změnu úrovně přiblížení. Umožňuje totiž přizpůsobit obsah průhledu tak, aby se v něm vybraný záznam zobrazil celý a v co největších rozměrech.

V případě, že je funkce zaostření zapnuta, zjistí widget, jaké je trvání zvolené entity, a projde dostupné úrovně přiblížení. Přitom se pokusí najít takovou, pro jejíž parametr d_{px} platí, že je nejmenší možný a zároveň

$$d_{px} \cdot w_{viewport} > d_e ,$$

kde d_e představuje trvání historického záznamu. Nalezenou úroveň pak widget nastaví jako aktuální a zároveň obálku posune tak, aby zvolenou entitu vystředil. Nalezne tedy čas t_{c_e} ve středu trvání entity jako

$$t_{c_e} = t_{s_e} + \frac{d_e}{2} ,$$

kde t_{s_e} je čas začátku historického záznamu, a nastaví jej jako středový čas widgetu.

7.8 Načítání dat a jejich formát

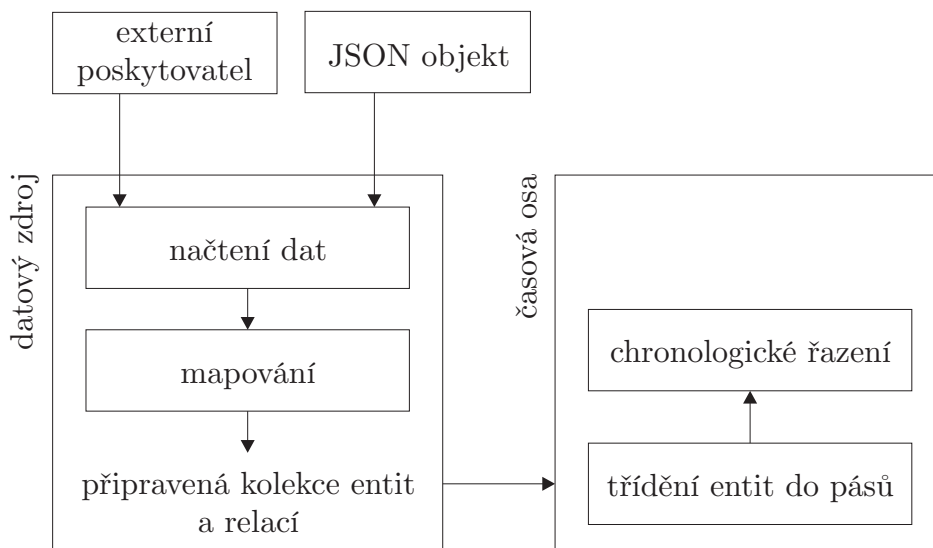
V této práci bylo již několikrát zmíněno, že v určitých ohledech vychází z jiných souběžně zpracovávaných diplomových prací. Ty výrazně ovlivnily právě formát dat a rozhodly po vzájemné konzultaci o způsobu jejich dodání. To, jak lze standardně klasifikovat historické záznamy, již popisovala kapitola 3.2. Tato se věnuje principu, jakým je widget získává a jak mají být strukturována.

7.8.1 Datový zdroj

Objekt datového zdroje hraje v této práci roli prostředníka mezi widgetem časové osy a poskytovatelem historických záznamů, jež mají být vizualizovány. Jeho pozici lépe přibližuje obrázek 7.12.

Datový zdroj je zodpovědný za načtení dat, a to ať už se jedná o data získaná ze vzdáleného serveru (což je případ souběžné diplomové práce Bc. Davida Hrbáčka) nebo o statický JSON soubor. Protože jde pouze o abstrakci, nepředepisuje žádný konkrétní formát – poskytovatel může data zaslat v libovolné podobě za předpokladu, že na straně klienta existuje implementace datového zdroje, jež takovému formátu dat rozumí.

Načtené záznamy spolu s relacemi musí objekt datového zdroje transformovat na formát používaný časovou osou – namapuje entity a vztahy na odpovídající objektové reprezentace. Další kroky jsou již záležitostmi widgetu, datový zdroj v tuto chvíli slouží jen jako úložiště, z něhož časová osa data získává podle potřeby.



Obrázek 7.12: Diagram toku dat

7.8.2 Formát vstupních dat

Ačkoliv předchozí kapitola uvádí, že formát vstupních dat objektu zdroje může být prakticky libovolný, v rámci diplomové práce vznikne i konkrétní implementace datového zdroje, která bude na vstupu vyžadovat množinu záznamů a vztahů ve formátu JSON, tak jak uvádí požadavky v kapitole 7.8. Podobu této reprezentace uvádí obrázek 7.13.

Z obrázku zároveň plynou základní požadavky na objekt záznamu a objekt vztahu. Oba zmíněné musí disponovat

- unikátním číselným identifikátorem,
- označením (stereo)typu
- a názvem.

Entita musí obsahovat i informaci o začátku jejího trvání ve formátu dle normy ISO 8601 [9], volitelně pak (v případě, že jde o momentovou entitu) čas konce trvání rovněž dle normy. Reprezentace vztahu vyžaduje identifikátory záznamů, které se ho účastní.

Atribut `properties` slouží k předání dalších libovolných informací, jejichž zpracování je pak zcela v rukou vývojáře. Za zmínku však stojí speciální vlastnosti entity `startPrecision` a `endPrecision`, které umožňují definovat, s jakou přesností má být čas začátku, resp. konce, prezentován. Pokud nastavíme čas začátku trvání záznamu na `1956-05-27T23:10:43` a hodnota `startPrecision` bude `decade`, pak se uživateli informace o čase zobrazí ve formě řetězce *50. léta 20. st.* V rámci této práce akceptuje entita jako platné přesnosti času hodnoty uvedené v tabulce 7.3.

hodnota atributu <code>precision</code>	výstup
<code>century</code>	20. st.
<code>decade</code>	50. léta 20. st
<code>year</code>	1956
<code>month</code>	květen 1956
<code>day</code>	27. květen 1956
<code>none</code>	27. květen 1956 23.10:43

Tabulka 7.3: Příklady výstupů pro jednotlivé hodnoty vlastnosti `precision` (vzorový čas `1956-05-27T23:10:43`)

```
{
  "nodes" : [
    {
      "id" : <Number>,
      "stereotype" : <String>,
      "name" : <String>,
      "begin" : <String ISO8601>,
      "end" : <String ISO8601>, // nepovinné
      "description" : <String>, // nepovinné
      "properties" : { ... } // nepovinné
    }, ...
  ],
  "edges" : [
    {
      "id" : <Number>,
      "stereotype" : <String>,
      "from" : <Number>,
      "to" : <Number>,
      "name" : <String>,
      "properties" : { ... } // nepovinné
    }, ...
  ]
}
```

Obrázek 7.13: Formát dat pro reprezentaci množiny záznamů a vztahů vyžadovaný standardní implementací datového zdroje časové osy

8 Implementace

8.1 Použité knihovny jiných stran

8.1.1 jQuery a jQuery UI

Hlavní účel často užívané knihovny jQuery představuje zjednodušení psaní aplikací v JavaScriptu, což plně vystihuje i její motto *write less, do more* (napiš méně, udělej více). Běžné operace, které v čistém JavaScriptu vyžadují několik řádek zdrojového textu, zabaluje do jednoduchých a stručných metod, které jsou navíc funkční napříč všemi prohlížeči – řeší tedy interně i stránku *cross-browser* kompatibility [14].

jQuery UI je rozšiřující knihovnou jQuery, která vývojáři usnadňuje práci především tím, že doplňuje standardní možnosti o nové způsoby interakce a řadu ovládacích prvků [15]. V její nabídce tak lze nalézt widget pro výběr data, indikátor postupu ale rovněž i implementaci interakce typu tažení, změna velikosti nebo přeskupení prvků DOM.

Uvedené charakteristiky jsou dostatečným důvodem pro použití obou knihoven v této práci. jQuery výrazně zjednoduší zpracovávání AJAX požadavků a práci s DOM, jež představuje jednu z majoritních činností widgetu. U jQuery UI pak bude využito poskytované implementace pro interakci typu tažení (draggable) – to umožní získat kontrolu nad tím, jak uživatel manipuluje s časovou osou, a adekvátně na tyto události reagovat.

Vydavatel	jQuery Foundation
Domovská stránka	http://jquery.com • http://jqueryui.com
Licence	MIT license
Použitá verze	2.1.4 (jQuery) • 1.11.2 (jQuery UI)
Vydána dne	28. dubna 2015 (jQuery) • 6. února 2015 (jQuery UI)

8.1.2 RequireJS

Jednoduché webové stránky velmi často používají jen knihovny třetích stran a samy nevyžadují velké množství dalšího kódu JavaScriptu. Jedná-li se však o aplikaci, lze poměrně snadno očekávat, že velikost JavaScriptových souborů poroste – v případě, že se vyvojář snaží o přehlednost, stoupne i jejich počet.

Pokud nevkládáme JavaScriptový zdrojový text do HTML stránky přímo, musíme jej do dokumentu zahrnout direktivou¹

```
<script src="[cesta_k_JS]"></script>
```

a tu umístit před ukončovací značku pro `body` – zlepší se tak způsob načítání stránky (soubory JavaScriptu nebudou svým stahováním zdržovat stažení obrázků či samotného obsahu dokumentu) [16]. Když uvážíme výše popsanou situaci, kdy aplikace pracuje s mnohdy až desítkami takových souborů, znamenalo by to uvést je všechny v HTML dokumentu, a rovněž je tedy i všechny najednou načíst při jeho stahování. Často však nejsou všechny potřeba.

Popsanou situaci řeší využití **RequireJS** – knihovna, která usnadňuje načítání souborů a modulů JavaScriptu a zlepšuje rychlost a kvalitu kódu – jednak jeho organizací a také tím, že konkrétní modul (soubor) načítá až ve chvíli, kdy je skutečně vyžadován [17]. Díky elegantně navrženému zápisu se tak zahrnutí externího modulu ve zdrojovém textu JavaScript souboru (obrázek 8.1) podobá použití direktiv `import` nebo `include`, na něž jsme zvyklí v jiných programovacích jazycích.

```
define([
  "PrvniModul", /* relativní cesta k externímu modulu */
  "DruhyModul",
], function(
  Prvni,        /* alias externího modulu */
  Druhy         /* pro použití v definici tohoto modulu */
) {
  return {
    ... new Prvni() ...
  }
})
```

Obrázek 8.1: Struktura JavaScript modulu za použití RequireJS

Vydavatel	The Dojo Foundation
Domovská stránka	http://requirejs.org
Licence	The New BSD license / MIT license
Použitá verze	2.1.17
Vydána dne	31. března 2015

¹Starší publikace uvádí také atribut `type`, ten však není vyžadován, neboť je JavaScript výchozím skriptovacím jazykem pro HTML dokument [16].

8.1.3 moment.js

Práce s datem bývá oříškem ve většině jazyků a velmi často na ni nestačí jejich standardní vybavení, programátor tedy musí sáhnout po knihovnách třetích stran. Nejinak je tomu i v případě JavaScriptu, zde se jedná o obzvlášť obtížnou práci.

Základní objekty tohoto jazyka sice umožňují s datem pracovat, jde však o jednoduché operace, např. zformátování data či jeho převod z reprezentace časovým otiskem na člověkem čitelný údaj. Na problém narazíme ve chvíli, kdy potřebujeme počítat diferenci mezi dvěma daty či datum modifikovat třeba posunem o den.

Každé datum je v JavaScriptu v rámci objektu `Date` uchováváno jako počet milisekund od půlnoci 1. ledna 1970 dále označovaný jako *časová hodnota*, přičemž přestupné sekundy jsou ignorovány [18]. Vrátime-li se k uvedenému příkladu s přičtením dne k aktuálnímu datu, můžeme naivně předpokládat, že o den posunuté datum získáme prostým přičtením počtu milisekund jednoho dne ($8,64 \cdot 10^7$). Ve většině případů bude výsledek správný, ne však pokud půjde o datum, při němž se mění SEČ na SELČ. Ačkoliv je samozřejmé, že při této změně času nedochází k žádné ztrátě milisekund mezi dvěma dny, interpretace času je upravena – právě toto JavaScript neřeší a zřejmě i z toho důvodu neposkytuje u objektu `Date` metody `add` nebo `subtract` jako jiné jazyky (např. PHP). Časovou aritmetiku neimplementuje.

Mimo tento vybraný problém neposkytuje JavaScript pro práci s datem ani spoustu dalších funkcí, které může vývojář očekávat – ať už jde o sofistikované metody formátování, jeho lokalizaci či manipulaci s reprezentací časového úseku (*duration*).

Proto tato práce využije knihovnu `moment.js`. Mimo to, že výše uvedené nedostatky JavaScriptu skvěle řeší, její použití je navíc velmi jednoduché. Instance data se vytváří mnohonásobně přetíženým konstruktorem `moment`, existuje tedy řada způsobů, jak datum specifikovat – časovou hodnotou, otiskem nebo i řetězcem ve stanoveném formátu [19].

`moment.js` podporuje veškeré formáty stanovené normou ISO 8601 [9, 24], vstup časových údajů i jejich výstup je tedy standardizován. Widget pracuje s daty, která nejsou nijak spojena s časovým pásmem, díky čemuž se vyhýbá problémům se změnami ze SEČ na SELČ – `moment.js` výrazně zjednodušuje vypuštění časových zón svojí metodou `utc`. Ta začne data interně interpretovat pomocí koordinovaného světového času (UTC).

Vydavatel	Iskren Iovov Chernev
Domovská stránka	http://momentjs.com
Licence	MIT license
Použitá verze	2.10.3
Vydána dne	13. května 2015

8.1.4 Snap.svg

Vrstva vztahů používá pro vykreslení šipek SVG, přičemž důvody jeho upřednostnění před použitím canvasu uvádí kapitola 5.1. Ačkoliv JavaScript poskytuje základní metody pro práci s SVG (to je totiž součástí DOM, proto s ním lze pracovat až na vybrané detaily velmi podobně), existují knihovny, jež činí manipulaci s ním mnohem jednodušší. Jednou z takových je i `Snap.svg`, která pochází od autora rozšířené knihovny `Raphaël`, jež rovněž pracuje SVG. Rozdíl mezi nimi je však v podpoře prohlížeči. Zatímco `Raphaël` se soustředí na maximální podporu prohlížeči na úkor nabízených funkcí, `Snap.svg` se zaměřuje na moderní prohlížeče, díky čemuž může nabídnout pokročilejší možnosti [25].

Vydavatel	Dmitry Baranovskiy
Domovská stránka	http://snapsvg.io
Licence	Apache License 2.0
Použitá verze	0.4.1
Vydána dne	13. dubna 2015

8.1.5 Knihovny použité pro demo aplikaci

Demo aplikace (nebo také podpůrná aplikace) slouží v této práci pouze k tomu, aby demonstrovala použití widgetu. Není tedy hlavním produktem, a proto se jí ani text práce nebude příliš věnovat. Nicméně stojí za zmínku, že mimo výše popsané knihovny používané ovládacím prvkem časové osy, pracuje navíc pouze s CSS frameworkem `Bootstrap`, jež definuje výchozí vzhled některých prvků HTML stránky, doplňuje je o možnost interakce a celkově usnadňuje práci se styly.

V podpůrné aplikaci jej doplňuje ještě CSS/JavaScript rozšíření `Bootstrap dialog`, které usnadňuje vytváření dialogových oken.

8.2 Vlastní podpůrné moduly

8.2.1 oop.js

JavaScript je interpretovaný objektově orientovaný jazyk s datovými typy, operátory a několika výchozími objekty a metodami. Jeho syntaxe vychází z jazyků Java a C, stejně jako byla spousta principů těchto jazyků převzata do JavaScriptu [20]. Navzdory tomu se JavaScript od zmíněných jazyků výrazně odlišuje v zásadních bodech:

- nepracuje s třídami – ty jsou nahrazeny objektovými prototypy,
- funkce jsou interpretovány jako objekty, nesou spustitelný kód, který může být předán jinému objektu [20].

Uvedené vlastnosti JavaScriptu vedly k tomu, že vznikla jednoduchá knihovna, která zmíněné odlišnosti zakrývá tak, aby bylo možné vytvářet kód co do největší míry podobný principům OOP, jak je známe – tedy s třídami, rozhraními, děděním apod.

Closures

Definice 8.1. *Closure* je datová struktura, která uchovává kód funkce společně s daty či prostředím (*scope*), v němž má být spouštěna [23].

V případě JavaScriptu představuje *scope* to, na co se uvnitř funkce odkazujeme klíčovým slovem `this`. Closures lze využít mnoha způsoby, v této práci však slouží především k vytváření *callbacků* – funkcí volaných v příhodnou dobu. Widget časové osy je používá především k obsluze událostí.

Novou closure vytvoříme následovně:

```
new Closure(scope, function, args);
```

příčemž zavolat ji pak můžeme pomocí standardních method JavaScriptu, a to `call` nebo `apply`. Tyto metody se liší pouze tím, jak předávají cílové funkci argumenty. Například:

```
var obj = {
  x : 5,
  sum : function(a, b) {
    this.x = a + b;
  }
}

var clr = new Closure(obj, obj.sum);
clr.apply(null, [1, 2]); alert(obj.x);
clr.call(null, 1, 2); alert(obj.x);
```

Dialogové okno volané na posledních dvou řádcích ukázky zobrazí v obou případech výsledek 3. Prvním argumentem obou metod je právě *scope*, my však voláme closure, která sama o sobě žádný kontext nepotřebuje, a proto můžeme předat `null` (nebo jinou libovolnou hodnotu – nebude využita).

Funkci `function`, která je v rámci closure spouštěna, můžeme předat dodatečné parametry, které definujeme již při inicializaci closure. Ty se pak sloučí s těmi, které předává objekt volající closure. Mějme příklad, kdy jedno tlačítko provede nad objektem součet jeho dvou členů a druhé rozdíl, přičemž obě operace lze uskutečnit stejnou metodou, pouze s odlišným argumentem.

```
var obj = {
  a : 1,
  b : 2,
  perform : function(e, op) {
    this.a = this.a + op * this.b;
    alert(this.a);
  }
}

document.getElementById("btn_add")
  .onclick = new Closure(obj, obj.perform, +1);
document.getElementById("btn_sub")
  .onclick = new Closure(obj, obj.perform, -1);
```

Podstatnou roli zde hraje pořadí argumentů, kterému musí vývojář věnovat svoji pozornost – parametry předávané volajícím objektem (v tomto případě objekt události *e* vyvolaný jedním z tlačítek) obdrží funkce uvnitř closure jako první a až za nimi pak nalezne parametry předané při inicializaci closure (v tomto případě *op*, který je buď *+1*, nebo *-1*).

Pseudo-třídy s Class

`Class` slouží k vytvoření iluze, že v kódu JavaScriptu definujeme třídu tak, jak jsme zvyklí například u jazyka Java. Tento speciální objekt předává jako návratovou hodnotu funkci opatřenou objektovým prototypem – tato funkce se tedy chová jako konstruktor, a jejím voláním společně s klíčovým slovem `new` tak vytvoříme nový objekt (instanci).

`Class` navíc umožňuje jednoduchým způsobem dědit od jiných pseudo-tříd, kontroluje, zda jsou implementovány metody ze všech rozhraní, která pseudo-třída používá, a umožňuje pohodlnější ladění. Pro zjednodušení budeme dále v textu pro označení pseudo-třídy používat pouze výraz *třída*. Její definice vypadá následovně:

```
MyClass = new Class("MyClass", {
  _extends : AnotherClass,
  _implements : [IInterface1, IInterface2],
  _constructor : function(arg) {
    this.myArg = arg;
    this.myArray = [];
  },

  myArg : null,
  myArray : null,
```

```
MY_CONST : 5,

  getArray : function() {
    if(this.myArray == null) this.__exception("err", "...");
    return this.myArray;
  }

  myAbstractMethod : function() {
    this.__abstract("myAbstractMethod");
  }
});
```

Je důležité poznamenat, že členské proměnné třídy **musí** být inicializovány uvnitř konstruktoru (nebo jinde, každopádně však před jejich použitím). Zápis `myArray : null` totiž ve skutečnosti nedefinuje členskou, nýbrž interní² proměnnou, a slouží tak pouze k zpřehlednění kódu – dává jasně najevo, že instance disponuje členem `myArray`.

Každá třída vytvořená pomocí `Class` navíc automaticky dědí od speciální třídy `_Core`, která poskytuje prostředky pro vytvoření abstraktní metody nebo sofistikovaný záznam výjimky. Také první parametr konstruktoru `Class` – název třídy – má usnadnit orientaci v případných chybách vypsáných do konzole.

Dědění s `Class`

Předchozí kapitola již poznamenala, že s pomocí `Class` lze používat i principy dědičnosti. Prvním krokem je uvedení předka nepovinným atributem `_extends`.

Velmi často se setkáváme s tím, že konstruktor obsahuje parametry. V takovém případě je nutné zavolat i rodičovský konstruktor. U jazyků, jako je Java, jsme zvyklí, že k tomuto účelu použijeme klíčové slovo `super`. Zde tomu tak není, musíme přímo volat konstruktor rodiče, avšak v kontextu potomka:

```
_constructor : function(arg1, arg2) {
  MyParentClass.call(this, arg1, arg2);
}
```

V ostatních metodách instance už pak můžeme rodičovské metody volat pomocí automaticky generované členské proměnné `__super`, například

```
this.__super.someMethod.call(this, arg);
```

Poznámka Mimo členskou proměnnou `__super` disponuje každá instance také proměnnou `__className`, která může usnadnit práci při ladění.

²Interní proměnná je společná pro všechny instance třídy, pouze ty k ní mohou přistupovat – chová se tedy jako privátní statická.

Rozhraní

Stejně jako třídy lze definovat pomocí `oop.js` i rozhraní. Ve své podstatě se jedná o jednoduché objekty s prázdnými funkcemi. Ty jsou pak při implementaci rozhraní třídou předefinovány. Pokud však vývojář některou z prázdných funkcí zapomene předefinovat, kontrolní mechanismus jej na to při vytváření instance třídy upozorní.

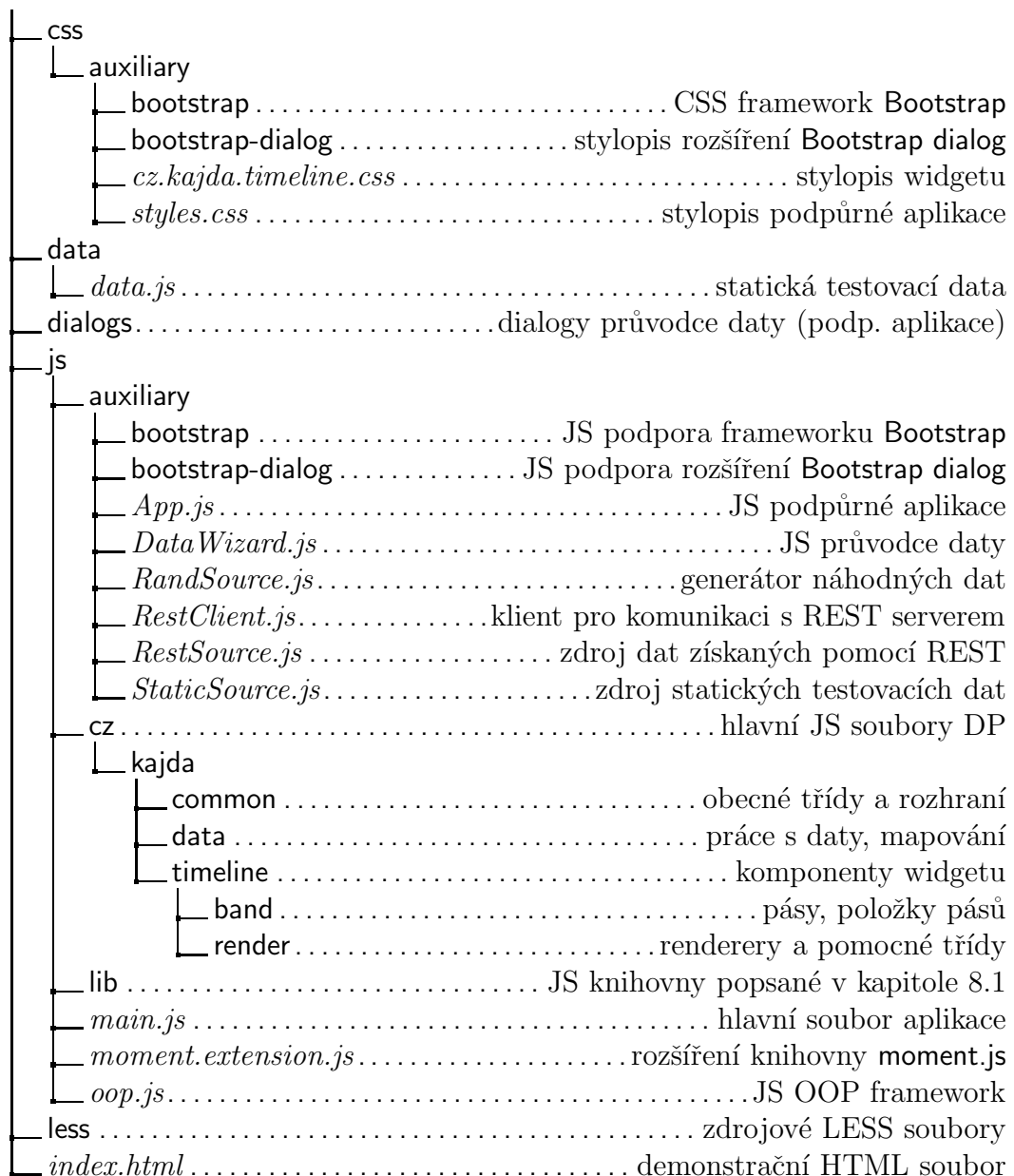
8.2.2 `moment.extension.js`

Rozšíření `moment.extension.js` doplňuje knihovnu `moment.js` o dodatečné funkce vyžadované widgetem časové osy. Jde především o úpravu formátování, původní knihovna totiž neposkytuje formátovací řetězce, jež by umožnily získat z časových dat informaci o století či desetiletí, do něhož časový okamžik spadá. Stejně tak neumožňuje lokalizovat výstup pro data před naším letopočtem (výstupem je v takovém případě pouze záporné číslo).

Poznámka `moment.js` je poskytnut s MIT licencí, lze jej tedy bez omezení upravovat za předpokladu, že s ním nadále bude znění licence distribuováno. Doplnky implementované v rámci popsaného rozšíření by se tedy mohly stát přímo součástí knihovny, to by si však vyžádalo porozumění její stavbě a konvencím zdrojového textu, což není předmětem této práce.

8.3 Fyzická struktura aplikace

To, jakým způsobem je aplikace fyzicky strukturována, ilustruje obrázek 8.2. V zásadě jde o rozdělení na tři základní části, a sice data, styly a soubory JavaScriptu. Zdrojové texty widgetu (složka `js/cz`) jsou uspořádané podobně jako balíky aplikace v jazyce `Java` a jejich struktura je zachována i při notaci tříd definovaných v rámci widgetu.



Obrázek 8.2: Fyzická struktura aplikace

8.4 Konvence zdrojových textů

Pro větší přehlednost uspořádání a obsahu zdrojových souborů, stanovuje diplomová práce několik základních konvencí, jež její programová část dodržuje.

- Třídy a rozhraní jsou sdružovány do balíků, přičemž jejich fyzické umístění balíkovou strukturu kopíruje.
- Každá třída nebo rozhraní je umístěno ve vlastním JS souboru.
- Názvy privátních členských proměnných a metod obsahují prefix `_`.
- Celky jednotlivých definic, např. členské proměnné, gettery a přepsané metody, se sdružují do *code-folds*, speciálních direktiv, které umožňuje část kódu sbalit do jednoho řádku. Zdrojový text vznikl v aplikaci Netbeans, která je jako součást komentáře definuje následovně:

```
//<editor-fold defaultstate="collapsed" desc="getters">  
...  
//</editor-fold>
```

- Názvy proměnných a dalších objektů používají formu *camelCase* (např. `getAbsolutePosition`), třídy a rozhraní pak speciálně *CamelCase* (např. `RelationViewer`). Jedinou výjimku tvoří názvy, které představují jméno matematické proměnné s indexem, např. `x_t` pro vyjádření x_t .
- Při definici třídy je v první řadě uváděna informace o implementaci rozhraní `_implements` a dědění rodičovské třídy `_extends` následována konstruktorem `_constructor`. Za ním pokračuje seznam členských proměnných a až poté deklarace a definice metod.
- Dokumentační komentáře používají z převážné většiny direktivy definované nástrojem JsDoc 3 a stejně tak používají i jím stanovené konvence pro zápis jmenných cest (`namepaths`). Konkrétně jde v případě widgetu o rozlišení zápisu statické (`Class.member`) a členské proměnné (`Class#member`) [26].

8.5 Stručný popis zdrojových souborů widgetu

Následující podkapitoly pouze velmi stručně informují o funkci jednotlivých tříd či rozhraní v rámci widgetu. Podrobnější informace lze získat z dokumentace uvedené přímo ve zdrojových textech, které jsou důsledně komentovány, a to včetně privátních metod a členských proměnných.

Poznámka Následující text používá výrazy *třída*, *rozhraní*, *abstraktní*, *implementovat* a další. Jejich význam má v tomto případě pouze informativní charakter, jelikož se fyzicky stále jedná o jednoduché funkce a objekty, které jsou pomocí OOP frameworku popsaného v kapitole 8.2.1 uspořádány a formátovány tak, aby připomínaly syntaxi standardních objektových jazyků.

8.5.1 Balík `cz.kajda.common`

Sdružuje obecně používané třídy a rozhraní, a to jak v rámci widgetu, tak i v demo aplikaci.

<code>Identifiable</code>	Rozhraní implementované třídami, jejichž instance mají být jedinečné svého typu. Unikátnost zajišťuje numerický identifikátor, jež má entita vrátet při volání metody <code>getId</code> , kterou právě toto rozhraní deklaruje.
<code>Observable</code>	Abstraktní třída, která umožňuje instancím oddělených tříd generovat události a těmi notifikovat objekty, které se metodou <code>addListener</code> zaregistrovaly k jejich poslechu.

8.5.2 Balík `cz.kajda.data`

Spojuje třídy, které jsou zodpovědné za práci s daty – uchovávají je, reprezentují a načítají. Jde o vrstvu mezi poskytovatelem dat a časovou osou.

<code>AbstractDataSource</code>	Abstrakce datového zdroje podrobněji popsána v kapitole 7.8.1. Deklaruje potřebné metody, přičemž mezi ty nejpodstatnější patří <code>loadData</code> a <code>_map</code> . <code>loadData</code> slouží vnějším objektům k vyvolání požadavku na získání dat. <code>_map</code> mapuje obdržená data na objekty používané widgetem časové osy, které jsou datovému zdroji předány při inicializaci.
<code>AbstractEntity</code>	Abstraktní třída popisující historický záznam. Vyžaduje pouze jeho identifikátor, o další podobě dat s ním spojených nic neříká.
<code>AbstractRelation</code>	Abstraktní třída popisující vztah mezi záznamy. Vyžaduje pouze jeho identifikátor, o další podobě dat s ním spojených nic neříká.
<code>Collection</code>	Kolekce, která pracuje s instancemi tříd implementujících <code>Identifiable</code> . Umožňuje tedy na základě identifikátoru hledat položku kolekce a při vkládání zabránit vzniku duplicit.

Entity	Konkrétní implementace objektu historického záznamu. Při inicializaci vyžaduje data ve formátu specifikovaném v kapitole 7.8. Řeší problémy přesnosti datování (4.2.3), rozlišuje momentovou a intervalovou entitu (3.2).
Relation	Konkrétní implementace vztahu mezi historickými záznamy. Při inicializaci vyžaduje data ve formátu specifikovaném v kapitole 7.8.

8.5.3 Balík `cz.kajda.timeline`

Sdružuje komponenty widgetu, které zajišťují hlavní funkce časové osy.

AbstractItem	Abstrakce položky časové osy připravena pro budoucí využití. V této práci ji používá pouze třída <code>BandItem</code> .
Component	Rodičovská třída pro všechny komponenty časové osy i ji samotnou. Komponenta může být vložena do DOM, lze ji překreslit, může generovat události (díky dědění od <code>Observable</code>) a může přidávat a spravovat podřízené komponenty.
NavBar	Komponenta navigační lišty umístěná v horní části widgetu. Informuje uživatele o aktuální středovém času a poskytuje mu možnost jej měnit pomocí tlačítek.
Projection	Pomocná třída pro provádění transformací na základě úrovně přiblížení popsaných v kapitole 7.5.
RelationViewer	Vrstva pro prezentaci vztahů mezi záznamy. Sama o sobě nevykresluje vztahy , pouze rozhoduje o tom, zda vůbec a které vykreslit na základě pravidel uvedených v kapitole 7.6.8.
Ruler	Komponenta pravítka, jež uživateli pomáhá s orientací v čase. Zajišťuje vykreslování dílků podle aktuální úrovně přiblížení, tak jak to popisuje kapitola 7.6.4.

Timeline	Hlavní komponenta widgetu zastřešující ostatní součásti. Obsluhuje všechny z vnějšku přicházející události a zároveň také generuje vlastní. Poskytuje rozhraní pro komunikaci s okolním prostředím, díky čemuž umožňuje jiným objektům měnit její vybrané vlastnosti (např. středový čas nebo označený záznam).
Wrapper	Obálka časové osy sdružující ty komponenty, které při posunu osy rovněž změni svoji pozici, např. pravítko, pás záznamů. Wrapper generuje události informující o tažení osy a předává je hlavní komponentě ke zpracování.
ZoomBar	Ovládací prvek pro změnu úrovně přiblížení dostupný v pravém horním rohu widgetu.
ZoomLevel	Objektová reprezentace úrovně přiblížení. Uchovává charakteristiky popsané v kapitole 7.5 potřebné pro poziční a proporční výpočty v rámci časové osy.

8.5.4 Balík `cz.kajda.timeline.band`

Spojuje dohromady komponenty časové osy, které jsou zodpovědné za reprezentaci pásů záznamů, jejich slučování a identifikaci.

Band	Komponenta pásu, která si uchovává seznam záznamů do ní spadajících a na základě podmínek v kapitole 7.6.5 tyto záznamy pak prezentuje uživateli. Stejná kapitola rovněž uvádí i způsob, kterým tato komponenta řeší kolize záznamů.
BandGroup	Slučuje pásy do jedné skupiny tak, aby se při posunu osy pohybovaly všechny současně a nedocházelo tak mezi nimi k časovým rozporům. <code>BandGroup</code> je zároveň zodpovědná za rozdělení dostupné výšky průhledu mezi jednotlivé pásy.
BandItem	Komponenta reprezentující historický záznam jako položku pásu, jde o grafickou podobu instancí třídy <code>Entity</code> . Položka pásu o svojí podobě nerozhoduje, tu obstarává renderer, jehož instanci obdrží při inicializaci.

<code>BandLabelContainer</code>	Speciální komponenta, která je umístěna mimo wrapper do popředí widgetu časové osy, a to z toho důvodu, aby při posunu obálky neměnila svoji pozici a zůstala pozicována fixně v levé části. Uživatele informuje o názvu jednotlivých pásů vložených do skupiny.
---------------------------------	--

8.5.5 Balík `cz.kajda.timeline.render`

Komponenty sloužící k prezentaci dat popsané v předchozích bodech ve většině případů samy nezodpovídají za svoji podobu. Ta je totiž dána renderery – objekty, které na základě vlastností dat rozhodnou o tom, jak mají být zobrazena uživateli. Balík `render` takové třídy spojuje.

<code>AbstractItemRenderer</code>	Abstraktní třída pro vykreslování záznamu na časové ose. Deklaruje metodu <code>render</code> , která vytvoří fyzický HTML prvek, jenž může být přidán do DOM, a <code>redraw</code> , jež podle potřeby mění jeho podobu či pozici.
<code>AbstractRelationRenderer</code>	Abstraktní třída pro vykreslování vztahů mezi záznamy na časové ose. Na rozdíl od <code>AbstractItemRenderer</code> nevytváří HTML prvky, pouze přidává elementy do SVG plátna komponenty <code>RelationViewer</code> , a to jedinou metodou <code>drawRelation</code> .
<code>BandItemRelationRenderer</code>	Konkrétní implementace rendereru pro vyobrazení vztahů mezi položkami pásu časové osy.
<code>BandItemRenderer</code>	Konkrétní implementace rendereru pro položky pásu časové osy.
<code>Color</code>	Pomocná třída pro objektovou reprezentaci barev. Díky tomu, že nejde o pouhý hexadecimální řetězec, umožňuje měnit jas barvy nebo hodnotu jejího alfa kanálu. Toho využívají renderery ve chvíli, kdy potřebují vytvořit rám okolo položky pásu tak, že je o několik stupňů tmavší než barva jejího pozadí.

8.6 Komponenty

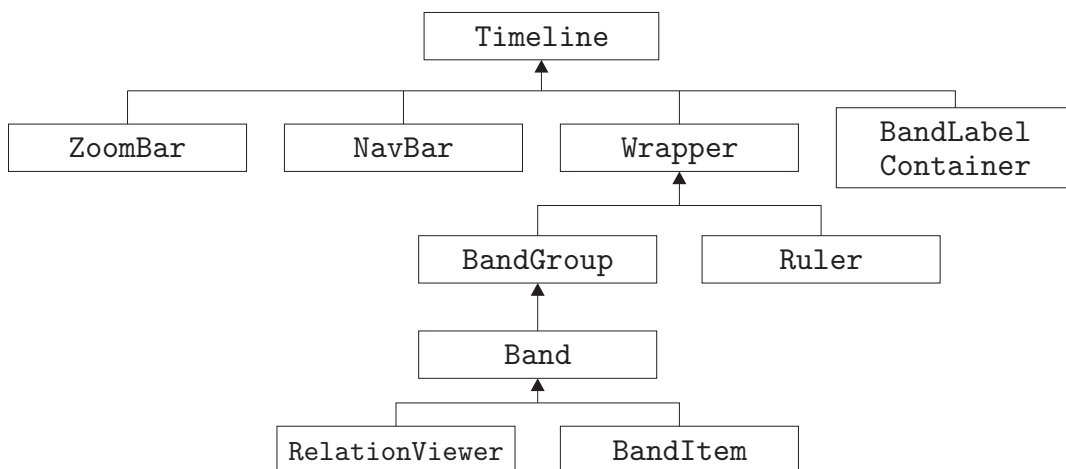
Tato kapitola podrobněji popisuje charakteristiky komponent uvnitř widgetu, především se pak zabývá vztahem komponenty a HTML prvku, překreslováním a prací s pozicí.

8.6.1 Životní cyklus

V případě HTML aplikace se komponenty chovají odlišně, než jak jsme zvyklí například u jazyka **Java**. Není je potřeba překreslovat při každé změně, jelikož nejde o proces renderování ve svém standardním slova smyslu. Prvky *překresluje* pouze v okamžiku, kdy přímo vyžadujeme jejich změnu, a tehdy cíleně vyvoláváme požadavek na vykreslení.

Následující kroky přibližují životní cyklus komponenty uvnitř widgetu:

1. Komponenta **newbie** je iniciována voláním konstruktoru. V tuto chvíli existuje pouze v paměti, není však žádným způsobem přítomna v DOM.
2. Komponentu **newbie** přidává jiný objekt **parent** jako svoji subkomponentu voláním `parent.addComponent(newbie)`.
 - **parent** se nastaví komponentě **newbie** jako její rodič.
 - Zavolá nad **newbie** metodu **build**, která vytvoří HTML prvek a v podobě jQuery objektu jej vrátí.
 - Ten vloží **parent** do svého HTML prvku.
 - **parent** přidá **newbie** do seznamu subkomponent.
3. Komponentu **newbie** lze nyní standardně používat. Nachází se v DOM widgetu, což lze ověřit metodou `isInDOM`. Rovněž všechny další metody týkající se její HTML reprezentace, jako jsou `getWidth` nebo `getPosition`, můžeme nyní použít.
4. Sama komponenta nebo jiný objekt (obvykle rodič) si může vyžádat překreslení voláním **redraw**. Ve standardní implementaci tím způsobí pouze to, že komponenta zavolá **redraw** nad všemi svými subkomponentami. Obvykle ale většina objektů uvnitř časové osy tuto metodu přepisuje a provádí v ní vlastní potřebné úkony k zajištění správného vykreslení. Například objekt časové osy při překreslení přepočítá aktuální středový čas.
5. Kterýkoliv objekt může komponentu cíleně vyjmout z DOM voláním metody **undraw**. Ta vyjme z objektového modelu widgetu HTML prvek reprezentující komponentu (a následná volání metody `isInDOM` pak vracejí negativní odpověď) a zároveň také zruší její registraci u rodiče. Komponenta však nadále existuje v paměti.



Obrázek 8.3: Komponentový strom

- Objekt komponenty lze zcela odstranit použitím klíčového slova `delete`, které je standardní výbavou JavaScriptu. Předtím však lze *život* komponenty obnovit tak, že ji opět přidáme jako subkomponentu některé z existujících – v takovém případě se vrátíme do bodu 2.

8.6.2 Hierarchie v rámci widgetu

Komponenty uvnitř prvku časové osy zachovávají předem dané stromové uspořádání, které popisuje obrázek 8.3. Ačkoliv dle popsané implementace třídy `Component` mají přístup výhradně k nadřazené komponentě a svým subkomponentám, instance třídy `Timeline` si speciálně uchovává referenci i na vybrané komponenty, které nejsou jejím přímým potomkem, např. `Ruler`.

8.7 Vybrané problémy implementace

V kapitole 4.2 popisuje práce některé problémy spojené s vizualizací historických dat. Následující odstavce se zabývají tím, jaké postupy byly použity pro jejich eliminaci, ať už byla jakkoliv úspěšná.

8.7.1 Rok 1 př. n. l

Kapitola 4.2.1 pojednávala o rozporu specifikace ISO8601 a gregoriánského kalendáře v záležitosti roku nula, respektive 1 př. n. l. Protože použitá knihovna `moment.js` pracuje rovněž s uvedenou normou ISO, promítá se tento problém i do časové osy vznikající v rámci diplomové práce.

První krok ke korekci roku nula znamenal úpravu formátování, a to až už přímo v knihovně `moment.js` nebo jejím doplněném rozšíření `moment.extension.js`. Cílem bylo upravení hodnoty roku u všech dat, která předcházejí 1. lednu roku 1 n. l. Taková úprava však sama o sobě nepomohla při vykreslování pravítka osy, naopak problém zhoršila, protože při velikosti dílku odpovídající jednomu století pak popisky končily číslicí jedna. To nepůsobí přehledně.

Po provedení této korekce zřejmě dojdeme k závěru, že zásadní problém vytváří chybějící nula v číselné řadě, tedy že mezi rokem 1 př. n. l. a 1 n. l. uběhl čas jednoho roku, byť $1 - (-1) = 2$. Proto budeme v komponentě pravítka před jejím vykreslením ověřovat, zda vkládaný dílek nepokrývá počátek našeho letopočtu, a pokud ano, zmenšíme jeho trvání o 1 rok. Tím zkrátíme první století před naším letopočtem a vyrovnáme tak chybějící nulu v číselné řadě pravítka (let).

Tak ovšem situaci nevyřešíme úplně. Pokud posuneme obálku a upravíme úroveň přiblížení do takové podoby, kdy se celý načtený časový interval nachází před začátkem našeho letopočtu, ke korekci nedojde. Tento problém se ale bohužel vyřešit nepodařilo.

8.7.2 Letní čas

Problém vznikající při zobrazení časového intervalu, v němž dochází ke změně ze SEČ na SELČ a vice versa, popisovala kapitola 4.2.2. Jeho řešení však bylo výrazně jednodušší než u komplikací s rokem nula. Díky tomu, že knihovna `moment.js` nabízí funkci `utc`, která interně určuje, že se s datem bude zacházet jako se světově koordinovaným časem, se tento problém okamžitě eliminoval. Data zpracovávaná tímto způsobem tedy `moment.js` připraví o jakékoliv informace o časové zóně, kterou právě i SELČ je.

8.7.3 Přesnost datování

Kapitola 4.2.3 uváděla problém týkající se toho, jak od sebe odlišit entity, kde jedna začíná 1. ledna 1918 a o druhé víme pouze to, že někdy v roce 1918 začala. ISO norma nám neumožňuje uchovávat pouze část data, řetězec vyhovující standardu musí vždy nést minimálně informace o dni, měsíci a roku.

Tento problém widget řeší zavedením přesnosti data (vlastnosti entity `startPrecision` a `endPrecision`), která říká, jak velkým časovým úsekem chceme konec či začátek trvání entity popsat. Záměrem widgetu však bylo informovat uživatele o přesnosti data nejen textově, jak popisuje kapitola 4.2.3, ale i vizuálně, vzhledem entity.

V rámci implementace byl tedy upraven renderer položky pásu `BandItemRenderer`, který se pomocí speciální CSS vlastnosti pokoušel vyjádřit nepřesnost začátku či konce záznamu barevným přechodem od standardní barvy pozadí záznamu k průhledné barvě. Při větším množství položek v pásu ale byly zpo-

rovány problémy s časem vykreslení, který se znatelně prodloužil právě díky nutnosti vykreslení gradientů. Mimo to se při dlouho trvajících záznamech zobrazoval barevný přechod nestandardně. Z těchto důvodů byla implementace vizuální reprezentace časových nepřesností zrušena.

8.7.4 Entita typu *místo*

Kapitola 4.2.4 se zabývala problémem vizualizace entity se stereotypem *místo*. U záznamů tohoto typu lze obvykle velmi obtížně rozhodnout o jejich trvání a i v případě, kdy jsme schopni takový údaj stanovit, půjde pravděpodobně o velké časové rozmezí. Pokud pak bude v pásu přítomno více takových entit, uživatel prakticky neustále uvidí jen dlouhé pásy, což postrádá význam.

Widget vytvořený v rámci diplomové práce tento problém neřeší a takové záznamy interpretuje stejně jako ostatní. Díky obecnému návrhu však nabízí možnost vytvořit vlastní způsob vizualizace položek osy, který může nahradit zobrazení vlastních pásů, a to jen s minimálním zásahem do logiky widgetu.

8.8 Demo aplikace

Současně se zdrojovými soubory widgetu je dodána i demo aplikace, která prezentuje jeho použití. Zároveň poslouží i k provedení uživatelského testování. Tato práce se jí nebude zabývat podrobně, protože není jejím předmětem, uvede ale v následujících odstavcích aspoň základní informace o propojení s ovládacím prvkem časové osy.

8.8.1 Implementace datových zdrojů

Podpůrná aplikace nabízí tři odlišně implementované datové zdroje, přičemž každý z nich vznikl za jiným účelem testování. Všechny níže uvedené třídy dědí od abstraktní `cz.kajda.data.AbstractDataSource`.

<code>RandSource</code>	Slouží pro výkonnostní testování widgetu. Na základě konfigurace generuje náhodně záznamy i vztahy mezi nimi a ty poté předává časové ose. Uživatel u něj má možnost stanovit počet i časový rozsah generovaných entit.
<code>RestSource</code>	Komunikuje s REST serverem (produkt dip. práce Bc. Davida Hrbáčka), kterému při žádosti o data poskytne uživatelem nastavené priority jednotlivých stereotypů entit a relací. Server pak odpoví ohodnocenými záznamy a vztahy získanými z databáze.

StaticSource	Poskytuje statická data uložená v souboru JavaScriptu jako JSON objekt. Tato data nejsou ohodnocená, ale díky jejich rozsáhlosti a rozmanitosti nad nimi lze provést uživatelské testování widgetu.
---------------------	---

8.8.2 Pomocné třídy

Podpůrná aplikace využívá mimo soubory widgetu a datových zdrojů ještě několik vlastních tříd umístěných v adresáři `js/auxiliary`.

App	Sdružuje funkce podpůrné aplikace, převážně pak postranního panelu použitého při uživatelském testování. Demonstruje komunikaci s widgetem prostřednictvím vnějšího rozhraní.
DataWizard	Pomocí rozšíření <code>Bootstrap dialog</code> vytváří průvodce pro volbu priorit záznamů a vztahů při použití datového zdroje <code>RestSource</code> .
RestClient	Zjednodušuje vytváření požadavku pro REST server a jeho následnou obluhu. <code>RestClient</code> je používán datovým zdrojem <code>RestSource</code> při vyžádání načtení dat.

8.8.3 Změna datového zdroje

Při testování má uživatel možnost libovolně měnit datový zdroj použitý widgetem. Stačí, když v hlavním souboru JavaScriptu `js/main.js` na posledních řádcích odkomentuje jeden ze řádků:

- `app.createSource()` – vytvoří datový zdroj ze statického souboru dat (`data/data.js`),
- `app.startDataWizard()` – spustí průvodce nastavením priorit a následně vytvoří datový zdroj z odpovědi REST serveru,
- `app.createRandSource({...})` – v rámci datového zdroje vygeneruje náhodná data na základě předané konfigurace³.

Poznámka Při změně datového zdroje mějte na paměti, že se při obnovení webové aplikace v prohlížeči projeví jen tehdy, nenačítá-li prohlížeč soubory JavaScriptu z cache. Takovému chování lze obvykle předejít obnovou stránky klávesovou zkratkou `Ctrl+F5` nebo zapnutím vývojářských nástrojů.

³Popis konfiguračního objektu je zdokumentován v rámci interní proměnné `RandSource~DEFAULTS`.

9 Testování

Jedním z bodů zadání této diplomové práce bylo provedení testování implementovaného prvku ovládací osy, přičemž důraz má být kladen obzvláště na uživatelskou přívětivost a použitelnost. Následující podkapitoly uvádí způsoby, jimiž lze funkčnost a výkonnost vzniklého řešení testovat, a doplňují je i o konkrétní získané výsledky.

9.1 Metody testování

U widgetu časové osy můžeme za zásadní považovat dva aspekty – uživatelskou přívětivost a výkon. Zatímco míru toho, jak je použití ovládacího prvku pro uživatele příjemné a jednoduché, lze ověřit konstrukcí scénáře procházejícího jednotlivé funkce widgetu, pro prokázání dostatečné výkonnosti musíme jednak nejdříve určit, jaká úroveň představuje dostačující výkon, a pak také jak ji lze prověřit.

Pro softwarový produkt vzniklý v rámci této práce postupně provedeme následující testy:

- testy kompatibility, které navzdory proklamovanému omezení podpory pouze na prohlížeč **Chrome** (kapitola 7.1.3) ověří funkčnost i v jiných programech pro procházení HTML obsahu,
- výkonnostní testy, jež odhalí časovou náročnost vybraných výpočtů logiky a vykreslování pro různou velikost vstupních dat,
- a uživatelské testy dle scénáře, jimiž práce dokládá použitelnost widgetu.

9.2 Testování kompatibility

Jak již zmiňovala poznámka ke cross-browser kompatibilitě v kapitole 7.1.3, vývoj a průběžné testování funkčnosti widgetu probíhaly výhradně v prohlížeči **Chrome**. Přesto byli pro ověření kompatibility zvoleni tři hlavní zástupci na poli prohlížečů, a sice **Google Chrome**, **Mozilla Firefox** a **Internet Explorer** [21]. Ve všech třech aplikacích proběhly testy nad statickým zdrojem dat **StaticSource**, z nichž vznikl seznam potenciálně kritických oblastí, ve kterých může dojít k nekompatibilitě. Výsledky uvádí tabulka 9.1 a podrobněji je rozebírá následující kapitola.

9.2.1 Výsledky testování kompatibility

Tabulka 9.1 uvádí poměrně překvapivé výsledky, vezmeme-li v potaz, že při vývoji procházel widget testy výhradně v prohlížeči **Chrome**. Přesto se napříč zvolenými aplikacemi objevují drobné problémy s kompatibilitou, které buďto zapříčiňují

	Chrome 43.0	Firefox 38.0	IE 11.0
Ovládání			
změna úrovně přiblížení	●/●/●	! ¹ /-/●	! ¹ /●/●
posun	●/●/●	●/●/●	●/-/-
zrušení výběru záznamu	●/●/-	●/●/-	●/●/-
výběr záznamu	●	●	●
přechod po vztahu	●	●	-
Ostatní			
náhledy záznamů a vztahů	●	●	●
vodicí linky	●	●	●
zobrazení vztahů	●	●	●
řešení kolizí záznamů	●	! ²	! ²
vykreslení a vzhled	●	●	●

Tabulka 9.1: Tabulka výsledků testování kompatibility (položky s více symboly charakterizují různé způsoby ovládání, a to v pořadí myš / klávesy / tlačítka widgetu)

úplnou absenci vybrané funkce (-) nebo její odlišné chování (!), kde uvedený index odpovídá těmto případům:

1. Při rolování kolečkem myši za účelem změny úrovně přiblížení dochází v označených prohlížečích k opačnému efektu – při rolování směrem dopředu prohlížeč Chrome obsah přibližuje, kdežto Internet Explorer a Mozilla Firefox jej oddalují. Problém lze snadno řešit úpravou kódu.
2. Internet Explorer a Mozilla Firefox v jistých případech špatně vyřeší kolize záznamů, a ty se tak překrývají. Tento problém nevzniká chybou lane algoritmu (kapitola 7.6.5), nýbrž špatným určováním šířky záznamu w_{item} , jehož popisek je absolutně pozicován uvnitř.

9.3 Výkonnostní testy

Widget časové osy při svém překreslení provádí řadu úkonů, které mohou při větším množství vstupních dat výrazně zvýšit jeho časovou náročnost. Jde například o vykreslení záznamů do pásů nebo řešení kolizí mezi nimi.

Pro testování výkonu zvolíme zdroj náhodných dat (RandSource), jež je rovněž součástí implementace, jako časový rozsah období generovaných entit určíme 1. ledna 2001 př. n. l až 31. prosince 2000 a nastavíme vlastnosti jeho konfiguračního objektu podle tabulky 9.2.

Měření vždy proběhne při maximálním oddálení časové osy – tím docílíme toho, že spousta záznamů se bude kvůli velikosti popisků překrývat, a my tak

vlastnost	význam	hodnota
count	počet generovaných entit	$n \in \{10, \dots, 1000\}$
durationDist	trvání entity (d_e)	$\sim N(100, 30)$ [roky]
relationDist	počet vztahů k entitě	$\sim N(20, 5)$
priorityDist	priorita entit	$\sim N(100, 0)$
momentProbability	pravděpodob. vzniku momentové entity	0,4

Tabulka 9.2: Parametry zdroje náhodných dat pro testování výkonnosti

zatížíme lane algoritmus, který se tyto kolize snaží řešit. Jak je z obrázku 9.2 patrné, při maximálním oddálení se ose podaří entity v rozmezí přibližně 4 tisíc let rozdělit jen na čtyři *sloupečky*.

Postupně provedeme tři měření pro 10, 50, 100, 250, 500, 750 a 1 000 vygenerovaných záznamů, jejichž prioritu jsme jednotně nastavili na 100, abychom měli jistotu, že se osa pokusí zobrazit všechny najednou (byť část z nich *vyteče* z pásu). Informace o naměřených časech získáme z konzole prohlížeče, kam je komponenty widgetu zasílají (obrázek 9.1). Tyto informace může sledovat i běžný uživatel, jsou-li po načtení knihovny `oop.js` nastaveny proměnné `_OOP.debug` a `_OOP.time-Info` na hodnotu `true` (standardně tomu tak je).

▼ Timeline widget redraw stats	oop.js:39
▼ [person] band redraw	oop.js:39
Band item DOM insertation [0.32 s]	oop.js:328
Band item redraw [0.692 s]	oop.js:328
Compute overlaps [3.77 s]	oop.js:328
Redraw required by solving overlaps [0.579 s]	oop.js:328
└ Lane crossings: 34097, lane count: 165	oop.js:78
Generating ruler [0.076 s]	oop.js:328
└ Total time [5.525 s]	oop.js:328

Obrázek 9.1: Ukázka výstupu o měření časů v jednotlivých komponentách widgetu

9.3.1 Výsledky výkonnostních testů

Tabulka 9.3 ukazuje závislost času vyžadovaného překreslením widgetu na počtu záznamů, které zobrazuje. Uvedené hodnoty jsou získány třemi měřeními pro každé n a jednotlivé sloupce mají následující význam:



Obrázek 9.2: Ukázka vykreslení velkého počtu záznamů při maximálním oddálení osy

- *DOM* – čas vyžadovaný na registraci komponenty u rodiče, vytvoření odpovídajícího HTML prvku a jeho vložení do HTML prvku rodiče,
- *redraw* – čas vyžadovaný na výpočet a změnu pozice záznamu dle jeho časového zasazení a na výpočet umístění popisku,
- *lane algo* – čas nutný k realizaci lane algoritmu včetně času potřebného k získání šířky a pozice prvku,
- *overlap* – čas potřebný na úpravu pozic záznamů podle lane algoritmu,
- *total* – celková doba překreslení časové osy včetně všech komponent.

Je zřejmé, že hranici použitelnosti widget překračuje přibližně při vizualizaci 250 až 500 záznamů. Musíme však vzít v úvahu, že v tomto testu jsme zobrazovali všechny záznamy najednou. V praxi pravděpodobně nenastane situace, kdy by měla časová osa zobrazit současně 500 položek, protože jejich autor data ohodnotí (či použije hodnotící mechanismy implementované v rámci práce Bc. Davida Hrbáčka) tak, že při maximálním oddálení budou viditelné pouze ty položky, jejichž význam je v onom měřítku relevantní. Díky tomu sníží n při minimálním přiblížení třeba na pouhých 20.

V rámci měření widget zároveň zaznamenával i počet přesunů záznamů mezi pruhy (C) při běhu lane algoritmus. Podle kapitoly 7.6.5 odhadujeme jeho asymptotickou složitost na $O(\frac{n^2+n}{2})$, což odpovídá kvadratické složitosti $O(n^2)$. Tabulka 9.4 doplněná grafem 9.3 poukazuje na to, že naměřené počty přesunů mezi pruhy se jen zdaleka přibližují nejhoršímu odhadu, i přesto že jsme vstupní data modelovali tak, aby cíleně lane algoritmus vytížila.

n	DOM	redraw	lane algo	overlap	total
10	0,014±0,002	0,018±0,003	0,010±0,001	0,019±0,002	0,164±0,005
50	0,049±0,003	0,081±0,007	0,101±0,032	0,055±0,005	0,363±0,049
100	0,114±0,030	0,149±0,008	0,214±0,026	0,101±0,007	0,664±0,013
250	0,210±0,006	0,340±0,027	1,018±0,010	0,280±0,016	1,940±0,026
500	0,426±0,019	0,819±0,034	3,798±0,091	0,704±0,025	5,859±0,128
750	0,640±0,029	1,358±0,025	8,223±0,111	1,280±0,006	11,614±0,070
1000	0,900±0,018	2,150±0,054	14,421±0,111	1,962±0,080	19,559±0,141

Tabulka 9.3: Tabulka výsledků měření časů [sekundy] jednotlivých úkonů při n záznamech

n	10	50	100	250	500	750	1000
C	16	366	1427	8769	34 231	75 706	135 143
C_w	55	1 275	5 050	31 375	125 250	281 625	500 500
L	5	20	36	84	162	237	306

Tabulka 9.4: Tabulka počtu přesunů záznamů mezi pruhy C , nejhoršího odhadu tohoto počtu C_w a počtu vzniklých pruhů L

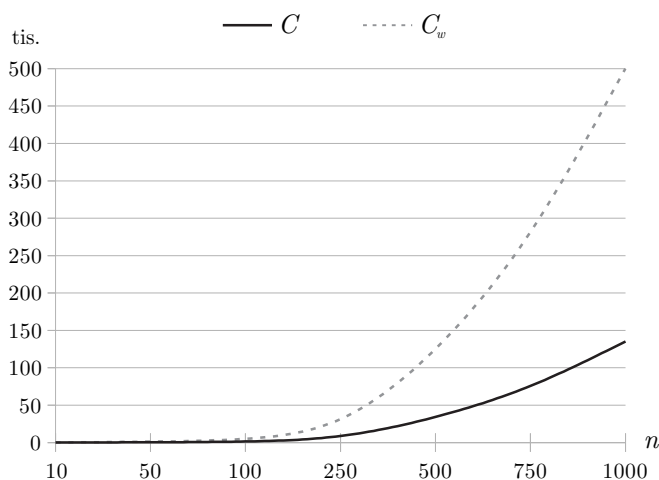
9.4 Uživatelské testy

Cílem uživatelského testování je prověřit, zda se ovládací prvek časové osy dobře používá, nezpůsobuje uživatelům problémy s orientací a především plní svoji funkci tak, jak se od něj očekává. Tyto aspekty pomůže ověřit testovací scénář, jež zúčastněné osoby s widgetem seznámí, vysvětlí jim účel testování a této práce a nakonec prověří funkce časové osy tím, že požádá uživatele o vyřešení několika úkolů. Na konci testování má pak autor vyplněného scénáře možnost poskytnout zpětnou vazbu, a to jak hodnocením, tak i prostřednictvím otevřeného komentáře.

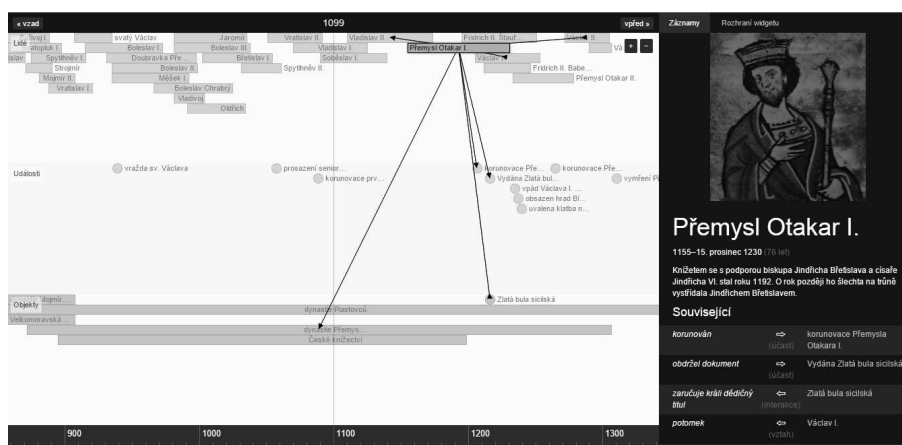
Pro uživatelské testování zvolíme historické záznamy z období Českého knížectví, přesněji od počátku Sámovy říše až po vymření rodu Přemyslovců, přičemž hlavními daty budou informace o panovnících, jejich příbuzenských vztazích a na vedlejší úrovni pak vybrané události z doby jejich vlády. Tyto záznamy poskytneme aplikaci formou statického datového zdroje `StaticSource`¹.

Samotný scénář testování zpřístupníme jako `Google Forms` dotazník, který umožňuje pohodlný sběr dat. Kompletní přepis testovacího scénáře obsahuje příloha A. Obrázek 9.4 pak prezentuje podobu widgetu spolu s podpůrnou aplikací, tak jak jej zobrazí účastníci testu.

¹`StaticSource` je v aplikaci již připraven a není potřeba jej nijak inicializovat. Stačí pouze na konci souboru `js/main.js` odkomentovat řádek `app.createSource()`.



Obrázek 9.3: Graf závislosti počtu přesunů záznamů mezi pruhy (lanes) na celkovém počtu záznamů



Obrázek 9.4: Náhled widgetu a podpůrné aplikace, tak jak jej zobrazí tester

9.4.1 Charakteristika testerů

Testování se zúčastnilo 15 osob. Součástí průvodce scénářem byla i část, která od uživatelů získávala základní demografické údaje a informace o jejich znalosti dějin a práce s počítačem. Na základě získaných dat můžeme průměrnou osobu, která

odeslala testovací scénář této diplomové práce, charakterizovat jako někoho, kdo

- je ve věku 30–50 let,
- své znalosti českého dějepisu považuje za běžné,
- zná klávesové zkratky na běžné nebo nadprůměrné úrovni,
- umí standardně pracovat s webovým prohlížečem
- a má nadprůměrné zkušenosti se zpracováním dat.

9.4.2 Výsledky testů

Testovací scénář požadoval po uživateli splnění tří úkolů, a sice:

- *Čtení časových údajů* – uživatel měl prostřednictvím podpůrné aplikace a widgetu dohledat datování vybraných osob či událostí, přičemž v některých případech nebylo jméno uvedeno explicitně, a tester jej tak musel dohledat čtením vztahů.
- *Jaký byl vztah?* – výchozí text uváděl dvě jména, mezi nimiž měl uživatel určit odpovídající vztah.
- *Mohli se potkat?* – úkolem testera bylo bez znalosti konkrétních časových údajů, pouze na základě vizualizace časovou osou rozhodnout, zda se dvě osoby mohly setkat, či nikoliv.

Čtení časových údajů

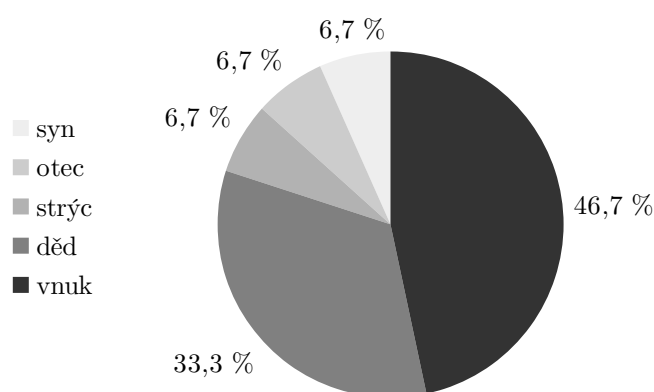
Při čtení časových údajů, ať už z widgetu či podpůrné aplikace, většina testerů odpověděla správně. Výjimku v tomto ohledu tvořila otázka *Kolika let se dožil sourozenec Vratislava I.*, u níž se testeré rozdělili do čtyř skupin, přičemž jen jedna odpověděla správně. Chybu ostatních zřejmě zapříčinila špatná interpretace vztahů čtených z widgetu nebo nepozornost, kvůli které mohli například zaměnit Vratislava za Vladislava.

Jaký byl vztah?

Podobně jako u předchozího úkolu i zde většina uživatelů odpovídala správně. Zcela odlišné odpovědi však testeré uváděli u otázky *Břetislav I. je ____ Boleslava II.*, jejíž správnou odpovědí je *vnuk*. Ta ale byla zastoupena v takřka stejném počtu jako špatná možnost *děd* (obrázek 9.5).

Hlavní problém zde pro uživatele zřejmě představovalo nalezení správného vztahu přes *prostředníka*. Relaci *praprotomek* (děd ↔ vnuk) nelze z časové osy vyčíst přímo, místo toho je reprezentována dvěma vztahy *potomek* (otec ↔ syn) – otec (v případě této otázky Oldřich) je tedy prostředníkem. Nicméně toho většina

testerů identifikovala správně, pouze pak vztah obrátila navzdory tomu, že z widgetu je zcela patrné pozdější narození Břetislava I. oproti Boleslavu II. K podobné chybě, nikoliv v takové míře, docházelo i u otázky *Přemysl Otakar II. je ____ Přemysla Otakara I.*



Obrázek 9.5: Výsledky otázky Břetislav I. je ____ Boleslava II.

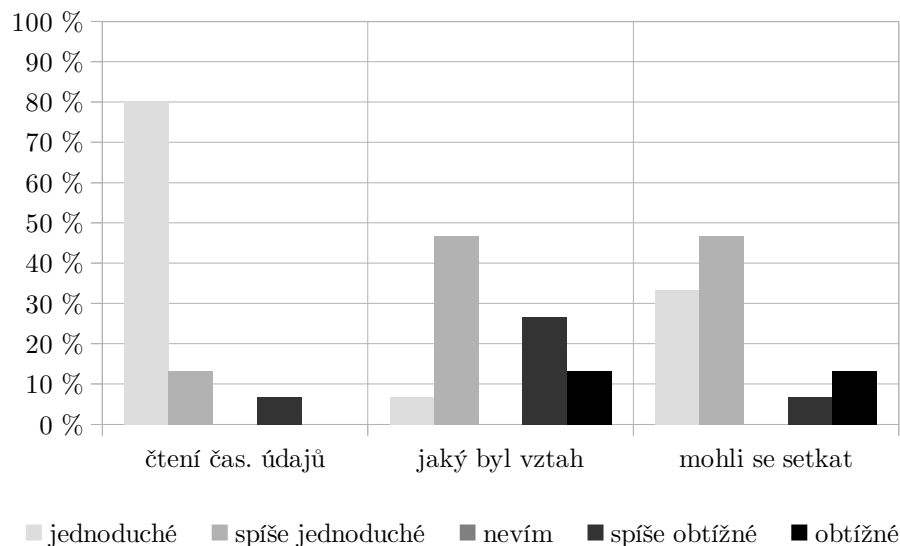
Mohli se potkat?

V posledním úkolu měli testéři rozhodnout o tom, zda se dvě osobnosti mohly potkat, a to na základě jejich reprezentace v časové ose (tedy průniku jejich životů). Většinu otázek vyřešili uživatelé správně, problém pro ně představovala pouze otázka setkání Oldřicha a jeho nejmladšího vnuka. U té totiž museli pracovat nejen se vztahy (určení vnuka pomocí prostředníka), ale také s pozicí grafické reprezentace v časové ose (výběr nejmladšího). Při řešení této otázky selhala jedna třetina uživatelů.

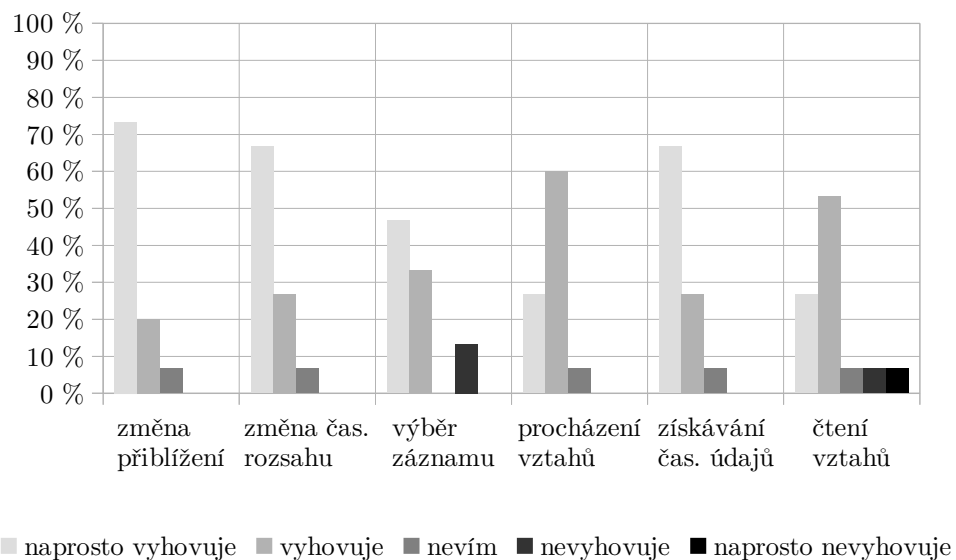
9.4.3 Uživatelské hodnocení

V poslední kroku testování uživatelé hodnotili celkový průběh testů. Pomocí pětiúrovňové stupnice měli rozhodnout o náročnosti jednotlivých úkolů a také o tom, jak jim vybrané funkční oblasti widgetu vyhovovaly. Výsledky shrnují grafy 9.6 a 9.7.

Mimo hodnocení pomocí výběru dostali testéři také prostor k písemnému vyjádření, ať už šlo o zdůraznění negativ či pozitiv. Většina uživatelů této možnosti využila a popsala své zkušenosti, které během testování získala.



Obrázek 9.6: Hodnocení jednotlivých úkolů



Obrázek 9.7: Hodnocení uživatelské přívětivosti operací widgetu

Jako negativum většina testerů uváděla způsob vizualizace vztahů. Popis pomocí šipek považovali za matoucí, obzvlášť velké problémy jim činil směr čtení vztahu (kapitola 7.6.8) bez ohledu na to, zda pracovali přímo s widgetem nebo s podpůrnou aplikací. Za obtěžující pak považovali operaci rušení výběru záznamu (což je patrné i z grafu 9.7) – použití pravého tlačítka pro zrušení výběru označili jako neintuitivní, případně postrádali možnost označit více záznamů najednou tak, aby mohli sledovat cestu vztahů (například pokrevní linii).

Naopak popularitu u uživatelů získala podpůrná aplikace, která sice není přímým produktem této diplomové práce, na druhou stranu ale demonstruje, že vhodnou implementací doplňující interaktivity, která využívá vnější rozhraní widgetu, lze dosáhnout lepšího ohlasu u uživatelů. Někteří rovněž ocenili rychlost, jakou časová osa reaguje na změnu úrovně přiblížení, a klasifikaci záznamů pomocí pásů. Zajímavostí je, že jeden z testerů považoval za užitečné zobrazení vztahu pomocí šipek, jež bylo ostatními ve většině případů označováno jako matoucí či náročné na pochopení. Tento uživatel jej však považoval za efektivní pomůcku při učení:

Líbí se mi práce s daty a jmény. Špatně se mi pamatují pojmy bez práce s nimi, tj. strojově. Určitě bych nezavádal označení vztahů, při přemýšlení nad tím, kdo je kdo podle šipek, se dobře zapamatovaly. Kdyby byly přímo napsané, nemělo by to ten efekt.

9.5 Význam výsledků pro další vývoj

Tato kapitola shrnuje výsledky všech testování a na jejich základě uvádí možnosti, jakými lze widget dále vyvíjet.

9.5.1 Vizualizace vztahů

Uživatelská část testování ukázala, že zobrazení relací mezi entitami pomocí pouhé šipky, kde pozice jejího konce (špičky) zásadně ovlivňuje význam vztahu, představuje pro člověka problém. Naneštěstí neexistuje mnoho jiných alternativ, které by mohly šipky nahradit a přitom úspěšně charakterizovat vztah mezi záznamy.

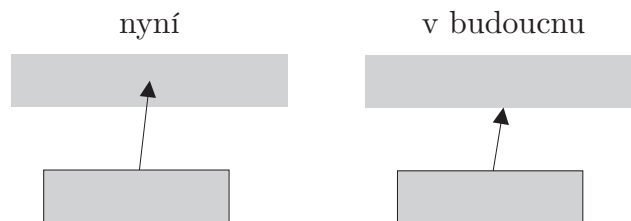
Výraznou pomocí pro zmateného uživatele by však mohla být specifikace názvu vztahu pro oba jeho významové směry. V této verzi widgetu očekává objektová reprezentace relace jediný řetězec názvu, který má kompletně vyjadřovat její smysl. Pokud by v budoucnu došlo k rozšíření relace o alternativní název vystihující opačný směr vztahu, uživatel by se orientoval snáze. Příkladem může být několikrát opakovaná relace *potomek*. Ať ji uživatel prochází ve směru od otce nebo od syna, pokaždé uvidí pouze popisek *potomek*, a to, kdo je koho synem, vyčte až ze směru šipky. V případě, kdy by vztah disponoval alternativním názvem, by uživatel při klepnutí na záznam otce uviděl u vztahu popisek *otec*

směřující k jeho potomkovi. Při klepnutí na potomka by pak relace zobrazila titulek *potomek* vedoucí směrem k otci.

9.5.2 Označování záznamů

Pro spoustu uživatelů je aktuální řešení označování záznamů nepraktické, a to především z toho důvodu, že pro výběr jiné položky musí nejdříve označení záznamu zrušit pomocí pravého tlačítka nebo klávesy **Escape**. To je ovšem vyžadováno pouze v případě, kdy widget aktivně používá vrstvu vztahů. Při její deaktivaci lze výběr záznamu změnit překliknutím na jiný. Tato možnost není při použití relation vieweru dostupná, protože jeho komponenta překrývá položky pásů, aby mohla zobrazit kompletně všechny šipky a nedošlo k jejich skrytí pod záznamy (pouze označená položka je předsunuta před vrstvu vztahů).

Jako možné řešení této situace se nabízí úprava vykreslovacího algoritmu vrstvy vztahů tak, že nepovedeme šipky do středového bodu prvku, který zobrazuje dobu trvání entity, ale ukončíme je na jeho okraji (obrázek 9.8). V takovém případě bychom i nadále určovali středové body, jen bychom pomocí odchylky vektorů (vektor vztahu a vektor vzdálenosti mezi středy položek účastnících se vztahu) určili průsečík šipky s okrajem cílového záznamu.



Obrázek 9.8: Rozdíl mezi současným a případným budoucím zobrazením vztahu

10 Závěr

Tato diplomová práce si kladla za cíl vytvořit ovládací prvek časové osy pro stránku zobrazitelnou v HTML prohlížeči, který by umožnil uživatelům pohodlně a přehledně procházet ohodnocené historické záznamy, sledovat jejich datování, popis a především pak vzájemné vztahy.

Ve své první části čtenáře stručně seznamuje s historií vizualizace dat, prezentuje přístupy, kterými se lidé dříve snažili záznamy o osobách a událostech uspořádat, přičemž poukazuje na to, že princip časové osy zůstal ve velmi blízké podobě zachován dodnes. Zároveň se zabývá tím, jak mohou být taková data reprezentována pomocí precedenčního grafu a jakým způsobem je můžeme klasifikovat podle jejich povahy.

Poté, co je čtenář seznám s charakterem historických záznamů a se způsoby, jak rozlišovat jejich typ, přináší tato práce základní informace o tom, co je to vizualizace a jak se odlišuje její realizace ve webovém prohlížeči od standardních principů. Současně také upozorňuje na problémy, které při zobrazování časových záznamů mohou vzniknout. Zabývá se například problematikým rokem nula, jenž je používám standardy ISO, nikoliv ale naším gregoriánským kalendářem, či způsoby, jak prezentovat nepřesnosti datování událostí.

V návaznosti na poodhalenou problematiku vizualizace historických záznamů stanovuje základní požadavky na výsledný produkt – ovládací prvek časové osy. Definuje, jakým způsobem má být koncipováno uživatelské rozhraní, jaké technologie budou použity pro jeho implementaci a především pak popisuje to, jak má časová osa zobrazovat vzájemné vztahy mezi entitami.

V další části představuje tato práce vybrané knihovny, které nabízí hotovou implementaci časové osy. Uvádí jejich základní vlastnosti, informace o podpoře a zejména pak posuzuje do jaké míry se shodují s definovanými požadavky na finální řešení.

Jelikož žádná z analyzovaných knihoven nenabízí možnost vizualizace vztahů mezi daty, je náplní další části rozsáhlý popis zcela nového ovládacího prvku, který se snaží vyhovět všem stanoveným podmínkám. Kapitola návrhu detailně rozebírá algoritmy a vzorce, které výsledná aplikace použije k výpočtu ideální pozice záznamů v časové ose. Na teoretické principy pak navazuje implementační část, jež čtenáře seznamuje s podobou již realizovaného ovládacího prvku.

V samotném závěru se pak práce zabývá tím, jak lze nově vzniklý produkt adekvátně otestovat, a to jak z hlediska uživatelského, tak i výkonnostního. Prověřuje kompatibilitu napříč nejpoužívanějšími prohlížeči, popisuje zátěžové testy a prezentuje jejich výsledky. V poslední řadě se věnuje zpětné vazbě získané od uživatelů, upozorňuje na oblasti, které testery příjemně překvapily, ale zároveň zmiňuje i jejich připomínky, na jejichž základě pak doporučuje možné úpravy realizovatelné při budoucím vývoji.

Literatura

- [1] McCORNICK, B. H. *Visualization in Scientific Computing*.
Computer Graphics, 21, 6 (listopad 1987).
ACM SIGGRAPH, New York City, New York, 1987.
- [2] AIGNER, W. – MIKSCH, S. – SCHUMANN, H. – TOMINSKI, Ch.
Visualization of Time-Oriented Data.
Springer Science & Business Media, New York City, New York, 2011.
ISBN10: 0857290797, ISBN13: 9780857290793.
- [3] GRAFTON, A. – ROSENBERG D.
Cartographies of Time: A History of the Timeline.
Princeton Architectural Press, Princeton, New Jersey, 2013.
ISBN10: 1568987633, ISBN13: 9781568987637.
- [4] FERGUSON, S. *The 1753 Carte chronographique of Jacques Barbeu-Dubourg*.
Friends of the Princeton University Library, Princeton, New Jersey, 1991.
- [5] TRETTEIEN, W. *Timeline Visualizations: A Brief and Incomplete Teleological History* [část I] [online].
Hyperstudio, MIT, Boston, Massachusetts, 4. 11. 2009 [cit. 18. 4. 2015].
Dostupné z: <http://hyperstudio.mit.edu/blog/blog-research/timeline-visualizations-a-brief-and-incomplete-teleological-history-part-1>
- [6] INGARGIOLA, G. *CIS 307: Precedence Graphs, Concurrency Grain, Fork and Join, CoBegin CoEnd* [online].
CIS, Temple University, Philadelphia, Pensylvánie [cit. 19. 4. 2015].
Dostupné z: <http://www.cis.temple.edu/~giorgio/old/cis307s96/readings/precedence.html>
- [7] *HTML5* [online].
© W3C, 2014 [cit. 6. 3. 2015]
Dostupné z <http://w3.org/TR/html5>

- [8] SUCAN, M. *Choosing Between the Two* [online].
©2006–2015, Opera Software ATA, 16. 6. 2010 [cit. 20. 4. 2015].
Dostupné z: <https://dev.opera.com/articles/svg-or-canvas-choose>
- [9] *ISO 8601:2004 – Date and time format*.
© ISO, 19. 6. 2013 [cit. 20. 4. 2015].
- [10] POUPA, M. *Časoměrný systém SEČ a přestupná sekunda* [online].
ZČU, Plzeň, 2008 [cit. 27. 4. 2015].
Dostupné z: <http://home.zcu.cz/~poupa/sec.html>
- [11] SIMPSON, K. *JavaScript and HTML5 Now*.
O'Reilly Media, Inc., Newton, Massachusetts, 2012. 1. vydání.
ISBN10: 1449339069.
- [12] BOSOMWORTH, D. *Mobile marketing statistics 2015* [online].
© SmartInsights, Ltd, 15. 1. 2015 [cit. 28. 3. 2015].
Dostupné z: <http://smartinsights.com/mobile-marketing/mobile-marketing-analytics/mobile-marketing-statistics>
- [13] Kogent Solutions Inc. *Ajax Black Book, New Edition*.
Dreamtech Press, Nové Dillí, Indie, 2008.
ISBN10: 8177228382, ISBN13: 9788177228380.
- [14] *jQuery Introduction*. W3Schools [online].
© Refsnes Data, 2015 [cit. 30. 5. 2015].
Dostupné z: http://www.w3schools.com/jquery/jquery_intro.asp
- [15] *Getting Started with jQuery UI* [online].
© The jQuery Foundation, 2015 [cit. 30. 5. 2015].
Dostupné z: <http://learn.jquery.com/jquery-ui/getting-started>
- [16] *JavaScript Where To*. W3Schools [online].
© Refsnes Data, 2015 [cit. 30. 5. 2015].
Dostupné z: http://www.w3schools.com/js/js_where_to.asp
- [17] BURKE, J. *RequireJS* [online] [cit. 30. 5. 2015].
Dostupné z: <http://requirejs.org>
- [18] *ECMAScript® Language Specification* [online].
© Ecma International, Ženeva, Švýcarsko, 2011 [cit. 30. 5. 2015].
Kapitola 15.9. Dostupné z:
<http://www.ecma-international.org/ecma-262/5.1/Ecma-262.pdf>

- [19] PATTON, T. *Moment.js simplifies working with date values in JavaScript*. Software Engineer [online].
 © CBS Interactive, San Francisco, Kalifornie, 8.7.2013 [cit. 30.5.2015].
 Dostupné z: <http://www.techrepublic.com/blog/software-engineer/momentjs-simplifies-working-with-date-values-in-javascript>
- [20] *A re-introduction to JavaScript* [online].
 © Mozilla Developer Newtwork and individual distributors, posl. úprava 26.4.2015 [cit. 31.5.2015].
 Dostupné z: https://developer.mozilla.org/cs/docs/Web/JavaScript/A_re-introduction_to_JavaScript
- [21] *Browser Statistics* [online].
 © Refsnefs Data, 2015 [cit. 31.5.2015].
 Dostupné z: http://www.w3schools.com/browsers/browsers_stats.asp
- [22] VOJÁČEK, J. *Aritmetická posloupnost a aritmetická řada* [online].
 © maths.cz, 2010 [cit. 24.6.2015].
 Dostupné z: <http://maths.cz/clanky/aritmeticka-posloupnost-a-aritmeticka-rada.html>
- [23] SOSHNIKOV, D. *Closures*. ECMA-262-3 in detail [online].
 © Dmitry A. Soshnikov, 2012 [cit. 7.6.2015].
 Dostupné z:
<http://dmitrysoshnikov.com/ecmascript/chapter-6-closures>
- [24] *Docs*. Moment.js [online].
 © Tim Wood, Iskren Chernev, Moment.js contributors, 2015 [cit. 21.6.2015].
 Dostupné z: <http://momentjs.com/docs>
- [25] *Why Snap*. Snap.svg [online].
 Dmitry Baranovskiy, 2015 [cit. 22.6.2015].
 Dostupné z: <http://snapsvg.io/about>
- [26] *Using namepaths with JSDoc 3*. Use JSDoc [online].
 © The contributors to the JSDoc 3 doc. project [cit. 23.6.2015].
 Dostupné z: <http://usejsdoc.org/about-namepaths.html>

Přehled použitých zkratek a pojmů

AJAX	<i>(Asynchronous JavaScript and XML)</i> technologie přenosu dat na pozadí webové stránky prostřednictvím skriptovacího jazyka JavaScript a XML
canvas	(plátno) plocha, do níž lze vykreslovat bitmapu
CSS	<i>(Cascade Style Sheet)</i> jazyk pro popis formátování HTML (obecně XML) dokumentů
DOM	<i>(Document Object Model)</i> objektová reprezentace HTML (obecně XML) dokumentu
GPS	<i>(Global Positioning System)</i> globální vojenský družicový systém umožňující určení polohy kdekoli na Zemi
HTML	<i>(HyperText Markup Language)</i> značkovací jazyk sémanticky popisující hypertextové dokumenty
JSON	<i>(JavaScript Object Notation)</i> formát zápisu dat vycházející ze syntaxe JavaScriptu
renderování	proces tvorby reálného obrazu na základě počítačového modelu
lightbox	ovládací prvek zobrazující obrázek nebo galerii obrázků, popř. jiná data (např. HTML obsah) v prvku webové stránky překrývajícím její původní obsah
REST	<i>(Representational State Transfer)</i> rozhraní v rámci protokolu HTTP navržené pro distribuované prostředí
SEČ	<i>(středoevropský čas)</i> střední sluneční čas středoevropského poledníku posunutý o +1 hodinu
SELČ	<i>(středoevropský letní čas)</i> střední sluneční čas středoevropského poledníku posunutý o +2 hodiny

SVG	<i>(Scalable Vector Graphics)</i> značkovací jazyk pro popis vektorové grafiky (vychází z XML)
UTC	<i>(Universal Time Coordinated)</i> koordinovaný světový čas založený na atomových hodinách
viewport	(průhled) část zobrazovacího zařízení, která funguje jako zorné pole
widget	ovládací prvek pro interakci aplikace s uživatelem, který obvykle slouží pro manipulaci s používanými daty
XML	<i>(eXtensible Markup Language)</i> značkovací jazyk sémanticky popisující dokumenty různého typu

Přílohy

- A Testovací arch (přepis)
- B Uživatelská příručka
- C Obsah přiloženého CD

A Testovací arch (přepis)

Úvodní poznámka Příloha představuje přepis interaktivního online formuláře, který byl použit pro sběr zpětné vazby od testerů.

Úvod

Dobrý den,
předem Vám děkuji za pomoc při testování mé diplomové práce. Jejím cílem je vytvořit interaktivní časovou osu, která umožňuje snadné procházení většího množství historických záznamů a zároveň uživateli dovoluje zobrazovat i některé vztahy, jež mezi nimi existují.

Než začnete plnit úkoly v rámci testování, ujistěte se, že máte možnost spustit testovanou aplikaci v prohlížeči Chrome. Jiné prohlížeče momentálně nejsou podporovány. Pokud jej tedy nemáte nainstalovaný ani instalaci neplánujete, v testování nepokračujte.

Při plnění úkolů se prosím pokuste potlačit Vaše dějepisné znalosti a vycházejte pouze z informací získaných prostřednictvím testované aplikace.

Díky.

Michal Kacerovský

Demografické údaje

V prvním kroku prosím vyplňte potřebné demografické údaje, které mi umožní snáz porozumět datům získaným během testování.

Následují formulářové prvky pro získání informací o věkové kategorii, znalosti klávesových zkratk, znalosti práce s prohlížečem, zkušenostech se zpracováním dat a případně jménu a příjmení.

Seznámení s časovou osou

Aplikaci naleznete na adrese: <http://home.zcu.cz/~kacerov2/timeline>

Hlavním předmětem testování je ovládací prvek časové osy v levé části obra-

zovky – tomu věnujte pozornost především. Postranní panel představuje podpůrnou aplikaci, do níž je časová osa zasazena a s níž také komunikuje. Při testování budete pracovat s oběma částmi.

Časovou osu lze ovládat myší, klávesnicí i pomocí tlačítek, která jsou její součástí. Přiblížení/oddálení:

- posun rolovacího kolečka myši
- tlačítka + a –
- klávesy + a – na numerické části klávesnice

Změna časového rozsahu:

- klepnutím a tažením osy levým tlačítkem myši
- tlačítka vpřed a vzad
- kurzorovými klávesami

Označení záznamu:

- Klepnutím levého tlačítka myši na záznam

(Jakmile v časové ose označíte některý ze záznamů, v postranním panelu se zobrazí jeho detailní popis a v časové ose pak vztahy označeného záznamu k jiným nejbližším.)

Zrušení označení záznamu:

- klávesa Escape
- klepnutí pravým tlačítkem myši kamkoliv mimo záznam

Vycentrování záznamu:

- klepnutí pravým tlačítkem myši na záznam

(Najde takové přiblížení, kdy je záznam největší a zároveň zobrazen celý. Rovněž jej posune na střed osy.)

Zmíněné interakce si vyzkoušejte a poté pokračujte dalším krokem.

Vztahy mezi záznamy

Časová osa vyvíjená v rámci této práce se od ostatních liší tím, že zobrazuje mimo samotné záznamy i vztahy mezi nimi. Ty lze navíc efektivně procházet. Vztahy jsou v rámci osy charakterizovány šipkou, přičemž její směr určuje i význam vztahu. Ten vždy čteme od záznamu, z něhož šipka vychází, k záznamu, do něhož směřuje.

Příklad

Michal Kacerovský — potomek → Antonín Kacerovský
čtěme jako *Michal Kacerovský je potomek Antonína Kacerovského*,
opačně by takový vztah vypadal:
Antonín Kacerovský — potomek ← Michal Kacerovský

Obdobu můžete pozorovat například u Břetislava I. a Spytihněva II. přímo v ose. V detailu záznamu pak získáte přehled o všech vztazích, jichž se účastní, navíc doplněný o záznamy, které tento vztah rovněž utváří. Klepnutím na jejich název je označíte v ose a zobrazíte jejich podrobnosti. Stejněho efektu dosáhnete i klepnutím na šipku přímo uvnitř osy. Takto můžete například procházet celou pokrevní linií mezi panovníky.

Vyzkoušejte si procházení vztahů a práci s nimi.

Hledání záznamů

Protože nejsou v časové ose na první pohled vidět všechny záznamy, můžete využít vyhledávací pole, jež se zobrazuje v postranním panelu pokaždě, když není označen žádný záznam v ose. Zkuste vyhledat třeba Oldřicha.

Čtení časových údajů

S použitím aplikace odpovězte na následující otázky.

1. Kdy vymřeli Přemyslovci po meči?
2. Kdy zemřel Měšek I.?
3. Kdy zemřel bratr Vladislava I.?
4. Kolika let se dožil sourozenec Vratislava I.?
5. Kdy byl korunován Vratislav II. českým králem?
6. V jakém roce se narodil Oldřichův děd?

Jaký byl vztah?

Pomocí časové osy určete vztah mezi následujícími událostmi a osobnostmi.

1. Břetislav I. je (syn/otec/děd/vnuk/strýc/synovec/praděd/pravnuk) Boleslava II.
2. Boleslav Chrabrý je (Piastovec/Přemyslovec/Mojmírovec) .
3. Svatého Václava zavraždil jeho (otec/syn/děd/bratr/synovec/strýc).

4. (Václav I. / Fridrich II. Štaufský / Fridrich II. Babenberský
Přemysl Otakar I./Přemysl Otakar II.) obsadil hrad Bítov.
5. Přemysl Otakar II. je (syn/otec/děd/vnuk/strýc/synovec/praděd/pravnuk)
Přemysla Otakara I.

Mohli se setkat?

Pokuste se z časové osy (bez znalosti konkrétních dat narození a úmrtí) určit, zda se mohly vybrané dvojice osob setkat (tj. žily ve stejné době).

1. Břetislav I. a Boleslav III.
2. Strojník a Rostislav
3. Soběslav I. a první český král
4. Vladivoj a vrah svatého Václava
5. Vladislav II. a držitel Zlaté buly sicilské
6. Oldřich a jeho nejmladší vnuk

Hodnocení

Testy jsou za Vámi! Teď je na čase, abyste ohodnotili, jak se Vám s aplikací pracovalo.

Následující prvky formuláře pro ohodnocení testovaného widgetu. Posuzované aspekty podrobněji popisuje kapitola 9.4.3.

B Uživatelská příručka

Tato příručka má poskytnout podrobnější informace těm, kteří chtějí widget časové osy používat ve své aplikaci nebo snaží jej rozšířit o další funkce. Dokumentuje metody vnějšího rozhraní a popisuje také konfigurační objekt časové osy.

Integrace widgetu do HTML aplikace

Ovládací prvek vzniklý v rámci práce, jejíž přílohou je tato dokumentace, závisí na několika knihovnách popsaných v kapitolách 8.1 a 8.2, které jsou k dispozici na připojeném médiu. Obzvláště podstatnou je pak `RequireJS`, jež zajišťuje asynchronní načtení všech potřebných tříd, a framework `oop.js`, který definice tříd umožňuje. Tyto dva moduly musí být načteny v HTML stránce, proto umístíme do hlavičky následující úsek kódu¹:

```
<script src="js/oop.js"></script>
<script data-main="js/main" src="js/lib/require/require.js">
</script>
```

Aby se osa korektně zobrazovala, neměla by chybět ani definice jejích CSS stylů. Do hlavičky HTML souboru aplikace tedy přidáme ještě následující řádky:

```
<link href="css/cz.kajda.timeline.css" type="text/css"
rel="stylesheet" />
```

Všimněte si speciálního atributu `data-main` u prvku odkazujícího na skript knihovny `RequireJS`. Uvádí cestu k hlavnímu JavaScript souboru (bez přípony), jehož příkazy jsou při načtení aplikace provedeny jako první – jde vlastně o jakousi alternativu *main class* v Javě.

Hlavní skript main.js

V rámci hlavního skriptu definujeme aliasy pro používané knihovny a zároveň také uvádíme jejich vzájemné závislosti. V hlavní funkci pak voláme všechny operace potřebné ke spuštění aplikace. Soubor `src/js/main.js` již tuto konfiguraci

¹Umístění odkazu na externí JavaScript soubory v tomto případě nijak načtení stránky nezpomalí, jelikož ta potřebuje ke své inicializaci, aby byly oba skripty načteny hned při startu.

obsahuje, při integraci widgetu do jiné aplikace tedy doporučuji z něj kompletní podobu níže uvedených řádků zkopírovat. Více informací o konfiguraci RequireJS lze získat přímo na stránkách vydavatele requirejs.org.

```
require.config({
  // výchozí cesta k root adresáři JS souborů
  baseUrl: "js",

  // cesty k jednotlivým knihovnám (bez přípony) a jejich aliasy
  paths: {
    jquery : "lib/jquery/jquery-2.1.4.min",
    jqueryui : "lib/jqueryui/jquery-ui.min",
    ...
  },

  // závislosti a další konfigurace
  shim: {
    "jqueryui" : {
      export: "$" ,
      deps: ['jquery']
    },
    ...
  }
});

requirejs([...], function(...) {
  // volání inicializačních funkcí
});
```

Konfigurace časové osy

Časová osa je konfigurována okamžitě při její inicializaci, spousta jejích vlastností lze však měnit i při běhu aplikace, do níž je integrována. Widget inicializujeme voláním konstruktoru třídy `cz.kajda.timeline.Timeline`, jemuž předáme jednak jQuery objekt nesoucí HTML prvek dokumentu, do něhož má být osa vložena, ale také konfigurační objekt popsany v následující podkapitole.

```
var config = {...};
var timeline = new Timeline($("#timelineContainer"), config);
```

Konfigurační objekt

Následující tabulka uvádí jednotlivé atributy konfiguračního objektu a jejich výchozí hodnoty.

<code>bands</code>	<code>Object</code> <code>({})</code> Popisuje vlastnosti jednotlivých pásů (více informací o struktuře v podkapitole Konfigurace pásu).
<code>bandAssignMethod</code>	<code>function(timeline, entity)</code> <code>(function(){})</code> <ul style="list-style-type: none">– <code>cz.kajda.timeline.Timeline</code> <i>timeline</i>– <code>cz.kajda.data.AbstractEntity</code> <i>entity</i>– <code>return cz.kajda.timeline.band.Band</code> Na základě vlastností <i>entity</i> rozhodne, do kterého pásu bude zařazena, a ten vrátí.
<code>cssPrefix</code>	<code>String</code> <code>(timeline)</code> Prefix společný pro všechny komponenty widgetu.
<code>defaultZoomLevel</code>	<code>Number</code> <code>(0)</code> Výchozí úroveň přiblížení.
<code>defaultTime</code>	<code>moment</code> <code>(moment().utc())</code> Výchozí středový čas.
<code>data</code>	<code>cz.kajda.data.AbstractDataSource</code> <code>(null)</code> Instance zdroje dat, je-li ve chvíli inicializace osy připravena k použití.
<code>events</code>	<code>Object</code> <code>({})</code> Objekt výchozích posluchačů událostí generovaných časovou osou (více informací v podkapitole Události widgetu).
<code>locale</code>	<code>Object</code> <code>({...})</code> Řetězce překladu widgetu (více informací v podkapitole Lokalizace).
<code>maxDataPriority</code>	<code>Number</code> <code>(100)</code> Stanovuje maximální možnou prioritu záznamů, přičemž minimální je pevně nastavena na 1.
<code>popoverDelay</code>	<code>Number</code> <code>(500)</code> Určuje, s jakým zpožděním (milisekundy) mají být zobrazovány náhledy entit a vztahů.

<code>popoverOffset</code>	Number	(10)
	Posun náhledu relace či entity o daný počet pixelů vertikálně i horizontálně.	
<code>popoverTemplateFactory</code>	Object	(<code>{}</code>)
	Definuje funkce, pomocí nichž jsou vykreslovány náhledy entit a relací (více informací v podkapitole Náhledy dat).	
<code>safeZoomLevel</code>	Number	(5)
	Index úrovně přiblížení, při níž by již měly být viditelné všechny záznamy bez ohledu na jejich prioritu.	
<code>showBandLabels</code>	Boolean	(<code>true</code>)
	Říká, zda mají být zobrazeny popisky pásů osy.	
<code>showGuidelines</code>	Boolean	(<code>true</code>)
	Říká, zda se mají při najetí kurzoru nad záznam zobrazit vodící linky.	
<code>showItemPopovers</code>	Boolean	(<code>true</code>)
	Říká, zda se má při podržení kurzoru nad záznamem zobrazit jeho náhled.	
<code>showRelationPopovers</code>	Boolean	(<code>true</code>)
	Říká, zda se má při podržení kurzoru nad vztahem zobrazit jeho náhled.	
<code>showRelations</code>	Boolean	(<code>true</code>)
	Říká, zda se má při výběru záznamu v ose zobrazovat vrstva vztahů.	
<code>showTimePointer</code>	Boolean	(<code>true</code>)
	Říká, zda se má v ose zobrazit ukazatel středového času.	
<code>slideCoefficient</code>	Number	(0.2)
	Udává, o jakou část šířky průhledu se má jím zobrazené časové období posunout při použití tlačítek či kláves.	
<code>zoomLevels</code>	Array	(<code>[...]</code>)
	Pole úrovní přiblížení (od nejmenší k největší), tj. pole instancí třídy <code>cz.kajda.timeline.ZoomLevel</code> .	

Konfigurace pásu

Popisuje atributy objektu, jenž stanovuje parametry pásu osy.

<code>color</code>	<code>String</code> Barva pozadí pásu.
<code>id</code>	<code>String</code> Řetězcový identifikátor pásu.
<code>itemRenderer</code>	<code>cz.kajda.timeline.render.AbstractItemRenderer</code> Renderer položek pásu.
<code>label</code>	<code>String</code> Lokalizovaný nadpis pásu.

Události widgetu

Časová osa generuje několik typů událostí, při nichž posluchači předává vybrané parametry.

<code>dataChanged</code>	(<code>cz.kajda.data.AbstractDataSource data</code>) Došlo ke změně dat zobrazených v ose (předána nová <i>data</i>).
<code>itemClick</code>	(<code>cz.kajda.data.AbstractEntity entity</code>) Klepnutí na záznam reprezentující <i>entity</i> v ose.
<code>itemEnter</code>	(<code>cz.kajda.data.AbstractEntity entity</code>) Vstup kurzoru na záznam reprezentující <i>entity</i> v ose.
<code>itemLeave</code>	(<code>cz.kajda.data.AbstractEntity entity</code>) Odsunutí kurzoru ze záznamu reprezentujícího <i>entity</i> v ose.
<code>itemFocus</code>	(<code>cz.kajda.data.AbstractEntity entity</code>) Označení záznamu reprezentujícího <i>entity</i> v ose. Událost je generována před <code>itemClick</code> .
<code>itemBlur</code>	(<code>cz.kajda.data.AbstractEntity entity</code>) Zrušení označení záznamu reprezentujícího <i>entity</i> v ose.
<code>relationClick</code>	(<code>cz.kajda.data.AbstractRelation rel</code>) Klepnutí na reprezentaci vztahu <i>rel</i> v ose.
<code>relationEnter</code>	(<code>cz.kajda.data.AbstractRelation rel</code>) Vstup kurzoru nad reprezentaci vztahu <i>rel</i> v ose.

<code>relationLeave</code>	(<code>cz.kajda.data.AbstractRelation rel</code>) Odsunutí kurzoru z reprezentace vztahu <i>rel</i> v ose.
<code>resize</code>	(<code>cz.kajda.timeline.Timeline timeline</code>) Vznikne, pokud byl vynucen přepočítání rozměrů widgetu.
<code>timeChanged</code>	(<code>moment newTime</code> , <code>moment.duration offset</code>) Došlo ke změně středového času na novou hodnotu <i>newTime</i> , která se od předchozí liší o dobu <i>offset</i> .
<code>zoomChanged</code>	(<code>Number dir</code> , <code>Number nLevel</code> , <code>Number offset</code>) Došlo ke změně úrovně přiblížení. Argument <i>dir</i> určuje o kolik úrovní a v jakém směru bylo přiblížení změněno, <i>nLevel</i> informuje o nově nastavené úrovni a <i>offset</i> udává počet pixelů od levého okraje widgetu k bodu, kde bylo přiblížení inicializováno.

Lokalizace

Widget časové osy používá několik řetězců, které by měly být lokalizovány do prostředí integrující aplikace. Následující tabulka uvádí identifikátory frází a jejich výchozí hodnotu.

<code>btnSlideBack</code>	<i>back</i>	(tlačítko posunu vzad)
<code>btnSlideForward</code>	<i>forward</i>	(tlačítko posunu vpřed)
<code>btnZoomIn</code>	<i>zoom in (Num +)</i>	(tlačítko pro přiblížení)
<code>btnZoomOut</code>	<i>zoom out (Num -)</i>	(tlačítko pro oddálení)

Náhledy dat

Při podržení kurzoru nad záznamem či vztahem zobrazí widget standardně náhled. Ten je konstruován pomocí speciálních funkcí, přičemž uživatel má možnost stanovit pro entity a relace s odlišnými stereotypy různé šablony náhledu.

Šablona je tvořena funkcí, která přijímá jako argument objekt entity či vztahu, pro nějž má náhled vzniknout. Navrací pak jQuery objekt obsahující DOM reprezentaci náhledu. Všechny funkce generující náhledy sdužuje atribut konfiguračního objektu časové osy `popoverTemplateFactory`, jehož struktura je následující:

```

{
  "entity" : {
    "nazev-stereotypu" : function(entity) { ... },
    ...
  },
  "relation" : {
    "nazev-stereotypu" : function(entity) { ... },
    ...
  }
}

```

přičemž místo názvu stereotypu lze použít zástupný symbol *, který widget informuje, že konkrétní funkce má být použita pro vykreslení náhledu jakéhokoliv typu vztahu či záznamu, který nemá definovanou šablonu pro vlastní typ.

Vnější rozhraní

Časová osa poskytuje navenek několik užitečných metod, pomocí nichž lze ovlivnit její chování či vzhled.

<code>addListener</code>	<p><code>void (String <i>eName</i>, Closure <i>handler</i>)</code></p> <p>Zaregistruje pro událost <i>eName</i> posluchače <i>handler</i>.</p>
<code>adjustTo</code>	<p><code>Boolean (cz.kajda.data.AbstractEntity <i>entity</i>[, Boolean <i>zoom</i>])</code></p> <p>Umístí grafickou reprezentaci předané entity do středu průhledu a je-li <i>zoom</i> nastaveno na <code>true</code> provede maximální možné přiblížení tak, aby byl záznam viditelný celý. Vratí <code>true</code>, pokud se akce podařila.</p>
<code>blur</code>	<p><code>void</code></p> <p>Zruší označení záznamu.</p>
<code>focusItem</code>	<p><code>Boolean (cz.kajda.data.AbstractEntity <i>entity</i>[, Boolean <i>adjust</i>][, Boolean <i>zoom</i>])</code></p> <p>Označí předanou entitu. Je-li <i>adjust</i> nastaveno na <code>true</code>, provede její vystředění, a pokud je i <i>zoom</i> nastaveno na <code>true</code>, pohled na entitu maximálně přiblíží. Vratí <code>true</code>, pokud se akce podařila.</p>
<code>getFocusedItem</code>	<p><code>cz.kajda.data.AbstractEntity ([Boolean <i>itemNeeded</i>])</code></p> <p>Vrátí aktuálně označený záznam. Je-li <i>itemNeeded</i> nastaveno na <code>true</code>, vrátí přímo jeho grafickou reprezentaci.</p>

<code>goTo</code>	<p><code>void (moment <i>nTime</i>)</code></p> <p>Změní středový čas osy.</p>
<code>slide</code>	<p><code>void (Number <i>direction</i>)</code></p> <p>Posune časový interval zobrazený v průhledu, přičemž o směru posunu rozhodne <i>dir</i> (+1/-1).</p>
<code>zoom</code>	<p><code>void (Number <i>direction</i>[, Number <i>offset</i>[, Boolean <i>specificLevel</i>]])</code></p> <p>Na základě argumentu <i>dir</i> (+1/-1) provede přiblížení nebo oddálení. Je-li stanoven <i>offset</i>, vezme jej v potaz, v opačném případě použije střed osy jako výchozí bod přibližování. Pokud je <i>specificLevel</i> nastaveno na <code>true</code>, pak se místo argumentu <i>direction</i> očekává index konkrétní úrovně přiblížení.</p>

C Obsah přiloženého CD

Obsah disku

bin	
└─ timeline-rest.war.....	WAR soubor REST serveru
src	zdrojové soubory widgetu a vzorová data (kapitola 8.3)
text	
└─ src.....	TEX soubory textu práce
└─ text.pdf.....	kompletní text práce včetně příloh

Zprovoznění

- Pro prohlížení vzorových dat použitých při testování (období Českého knížectví) nebo pro generování náhod. dat stačí otevřít soubor `src/index.html`.
- Úpravou (zakomentováním/odkomentováním) posledních řádků souboru `src/js/main.js` můžete rozhodnout o tom, který datový zdroj bude použit pro získání záznamů (více v kapitole 8.8.3).
- Chcete-li testovat komunikaci s REST rozhraním, nasad'te WAR soubor umístěný ve složce `bin` na server (testováno na Apache Tomcat 7.0.61). Soubor je produktem práce Bc. Davida Hrbáčka (Zpracování časových údajů pro jejich vizualizaci. 2015, ZČU, Plzeň.), která poskytuje podrobnější informace. Při změně řádku v souboru `src/js/main.js` nezapomeňte na stejném místě upravit také výchozí URL REST požadavků v závislosti na umístění nasazeného WAR souboru.
- **Upozornění:** Při použití datového zdroje načítajícího data prostřednictvím REST rozhraní je nutné, aby i sama aplikace s widgetem časové osy běžela na stejném serveru, popř. aby byl v prohlížeči povolen *cross-origin mode*.