VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

ROZŠÍŘENÍ ANALYZÁTORU STYLOVÝCH PŘEDPISŮ CSS3

CSS3 STYLE SHEET ANALYZER EXTENSIONS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **PETR MIKULÍK**
AUTHOR

VEDOUCÍ PRÁCE **Ing. RADEK BURGET, Ph.D.**
SUPERVISOR

BRNO 2019

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav informačních systémů (UIFS) Akademický rok 2018/2019

Zadání bakalářské práce

Student: **Mikulík Petr**
Program: Informační technologie
Název: **Rozšíření analyzátoru stylových předpisů CSS3**
CSS3 Style Sheet Analyzer Extensions
Kategorie: Web

Zadání:

1. Seznamte se s knihovnou jStyleParser a její architekturou.
2. Prostudujte nové vlastnosti CSS3 a identifikujte množinu vlastností, které dosud nejsou podporovány knihovnou jStyleParser.
3. Po dohodě s vedoucím zvolte vhodnou podmnožinu nových vlastností a navrhněte způsob jejich implementace.
4. Implementujte vybrané vlastnosti v knihovně jStyleParser.
5. Proveďte testování knihovny s využitím vhodných datových sad.
6. Zhodnoťte dosažené výsledky

Literatura:

- Dokumentace projektu jStyleParser: http://cssbox.sourceforge.net/jstyleparser/documentation.php
- Michálek, M.: Vzhůru do CSS3, e-kniha, 2015, https://www.vzhurudolu.cz/ebook-css3

Pro udělení zápočtu za první semestr je požadováno:

- Body 1 až 3

Podrobné závazné pokyny pro vypracování práce viz http://www.fit.vutbr.cz/info/szz/

Vedoucí práce: **Burget Radek, Ing., Ph.D.**
Vedoucí ústavu: Kolář Dušan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2018
Datum odevzdání: 15. května 2019
Datum schválení: 16. října 2018

Abstrakt

Práce rozšiřuje již existující open source projekt CSSBox přidáním podpory parseru jStyleParser o nové CSS3 vlastnosti. Konkrétně byla přidána podpora pro efekt stínu elementu, dále mřížkové rozložení a nakonec CSS animace a přechody. Práce je stejně jako původní projekt psána v jazyce Java a sestává ze série datových struktur, implementačních metod a testovacích tříd. Výsledkem práce je git pull request v systému Github.com nad původním repozitářem. Veškeré změny, související s touto prací, jsou již aplikovány do projektu a jsou tedy jeho součástí.

Abstract

Thesis extends an existing open source CSSBox project by adding support for new CSS3 properties to its jStyleParse parser. Specifically, support for box-shadow, grid layout and finaly CSS animations and transitions was added. Thesis is, just like the original project, writen Java and consists of a series of data structures, implementation methods and testing classes. The result is a git pull request on Github.com over the original project repository. All changes related to this thesis are applied to the project and are therefore part of it.

Klíčová slova

kaskádové styly, analyzátor stylových předpisů, CSS parser, jStyleParser, CSSBox, CSS3, blokový stín, mřížkové rozložení, animace, přechod, Java, Git

Keywords

Cascading Style Sheets, style rule analyzer, CSS parser, jStyleParser, CSSBox, CSS3, box-shadow, grid layout, animation, transition, Java, Git

Citace

MIKULÍK, Petr. *Rozšíření analyzátoru stylových předpisů CSS3*. Brno, 2019. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Radek Burget, Ph.D.

Rozšíření analyzátoru stylových předpisů CSS3

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Radka Burgeta. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Petr Mikulík
3. května 2019

Poděkování

Rád bych zde srdečně poděkoval svému vedoucímu práce panu Ing. Radku Burgetu za věnovaný čas, trpělivost a rady, jež mi umožnily práci úspěšně dokončit. Za to, že v průběhu tvorby byl vždy k dispozici a dokázal aktivně vysvětlit, vyřešit naskytlé problémy, se kterými jsem se postupně setkal. Děkuji.

Obsah

1 Úvod **3**

2 Kaskádové styly **4**
2.1 Syntaxe a gramatika 4

3 CSS syntaxe definice hodnoty **8**
3.1 Komponenty 8
3.2 Kombinátory 8
3.3 Multiplikátory 10

4 Shrnutí dosavadního stavu **11**
4.1 Význam subprojektu jStyleParser 13
4.2 Hierarchie datových struktur 13
4.3 Diagram volání 15
4.4 Aplikační rozhraní 17

5 Zhodnocení současného stavu **20**
5.1 Současná omezení projektu 20
5.2 Rozhodnutí o rozšíření 20

6 Návrh řešení **21**
6.1 Studium gramatiky 21
6.2 Programování řízené testy 21

7 Popis vlastní práce **22**
7.1 Rozmístění kódu 22
7.2 Testovací nástroje 23
7.3 První fáze – Efekt stínu 25
7.4 Druhá fáze – Mřížkové rozložení 27
7.4.1 Explicitní hranice 27
7.4.2 Implicitní hranice 28
7.4.3 Specifikace oblastí 31
7.4.4 Pozicování položek 32
7.4.5 Implicitní rozložení 33
7.4.6 Mezery mezi položkami 34
7.4.7 Zkrácený zápis 35
7.5 Třetí fáze – Přechody a animace 37
7.5.1 Přechod 37

7.5.2 Přechod – Ovlivněná vlastnost 38
7.5.3 Přechod – Délka 39
7.5.4 Přechod – Zpoždění startu 40
7.5.5 Přechod – Časová funkce 40
7.5.6 Animace 44
7.5.7 Animace – Identifikátor 45
7.5.8 Animace – Délka 46
7.5.9 Animace – Zpoždění startu 47
7.5.10 Animace – Počet opakování 47
7.5.11 Animace – Směr přehrávání 48
7.5.12 Animace – Stav přehrávání 49
7.5.13 Animace – Určení hodnoty před a po 50
7.5.14 Animace – Časová funkce 51

8 Závěr **52**

Literatura **53**

A Podporované vlastnosti **56**

B Nepodporované vlastnosti **59**

Kapitola 1

Úvod

Tento dokument pojednává o tvorbě rozšíření existujícího open source projektu CSSBox[1]. Projekt spojuje dvě v praxi spíše vzdálené oblasti, webové technologie a jazyk Java. V současné době je téměř všudypřítomně využívána standardní textová reprezentace pomocí HTML a CSS, což ovšem sebou nese své omezení ve snadnosti a efektivitě jejího zpracování, propagace úprav apod. Na rozdíl od toho způsob využitý v projektu CSSBox, umožňuje zachování efektivity typické práce s datovými strukturami uvnitř projektu i při komunikaci s jinými projekty podporujícími dané rozhraní.

Tento potenciál je možné pozorovat v podprojektu Pdf2Dom, zcela obcházející textovou formu dokumentu vygenerováním vnitřní reprezentace z PDF souboru. Bez jakéhokoli zásahu do zbytku projektu je tedy možné využít existující rozhraní, např. pro zobrazení dokumentu či vygenerování ekvivalentní HTML + CSS textové formy.

Přínos práce tkví v rozšíření podpory projektu o mnohé chybějící CSS3 styly při zachování jeho současných výhod a potenciálu. Protože jsou tyto pokročilé CSS vlastnosti již hojně využívány v běžných dokumentech dostupných na webu, přidání jejich podpory v projektu značně zvýší jeho celkovou využitelnost.

Text dokumentu se nejprve zabývá obecným uvedením do problematiky kaskádových stylů, dále způsobem vyjádření a validace jejich syntaxí. Následující sekce se zaměřuje na současný stav projektu, jeho význam a omezení. Dokument pokračuje návrhem na odstranění těchto omezení rozšířením projektu o chybějící styly. A v poslední sekci je popsán samotný způsob a průběh implementace nevržených změn.

[1] Veřejný repositář s projektem CSSBox – https://github.com/radkovo/jStyleParser

Kapitola 2

Kaskádové styly

Obrázek 2.1: Oficiální logo CSS3 [9]

Kaskádové styly neboli akronym CSS z anglického výrazu „Cascading Style Sheets“ je stylesheetový jazyk určený pro popis prezentace dokumentu napsaného v HTML nebo XML (včetně XML dialektů jako například SVG, MathML nebo XHTML). CSS vyjadřuje jak by elementy měly být vykresleny na obrazovku, na papír, v projevu nebo jiných médiích.

Jedná se o jeden z hlavních jazyků volného webu a je standardizován napříč webovými prohlížeči podle W3C specifikace[1]. Vyvíjen postupně, jeho první verze CSS1 je dnes již zastaralá, verze CSS2.1 je doporučená a nejnovější CSS3, která se rozděluje do menších modulů, se postupně standardem stává [11].

2.1 Syntaxe a gramatika

Základním úkolem jazyka CSS je umožnit prohlížeči vykreslit elementy dokumentu se specifickými vlastnostmi, jako barva, pozice nebo dekorace. Jeho syntaxe proto tento úkol reflektuje pomocí těchto stavebních bloků [10]:

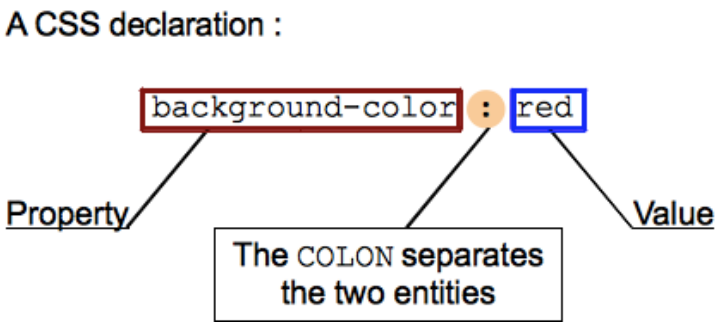
- **Vlastnost** (*property*) – identifikátor ve formě lidsky srozumitelného názvu, který definuje určitou vlastnost.
- **Hodnota** (*value*) – popis, jak má s vybranou vlastností prohlížeč naložit. Každá vlastnost má množinu validních hodnot definovaných formální gramatikou a také jejich sémantický význam pro implementaci.

[1] Cascading Style Sheets specifikace – https://www.w3.org/Style/CSS/#specs

CSS deklarace

Nastavování specifických hodnot pro dané vlastnosti je hlavní funkcí jazyka CSS. Pár vlastnosti a hodnoty nazýváme **deklarací** a prohlížeč při vykreslování musí vyhodnotit které deklarace se týkají kterého elementu.

Vlastnosti i hodnoty jsou ve výchozím stavu tzv. *case-insensitive*, neboli při specifikaci nezáleží na velikosti písmen. Pár je od sebe oddělen dvojtečkou `":"` a mezery před, mezi a za vlastnostmi jsou ignorovány.

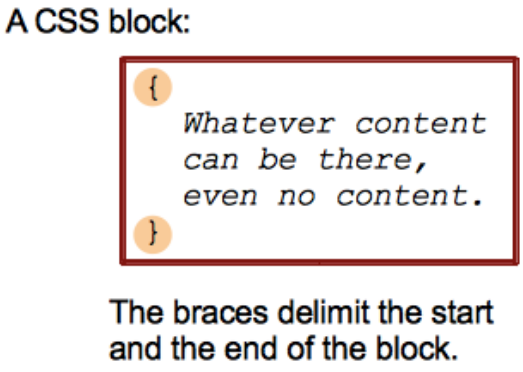


Obrázek 2.2: Ukázka syntaxe CSS deklarace [12]

Existuje více než 100 odlišných CSS vlastností a téměř nekonečno validních hodnot, které jim lze přiřadit. Pokud je vlastnosti přiřazena nevalidní hodnota je celá deklarace ignorována [10].

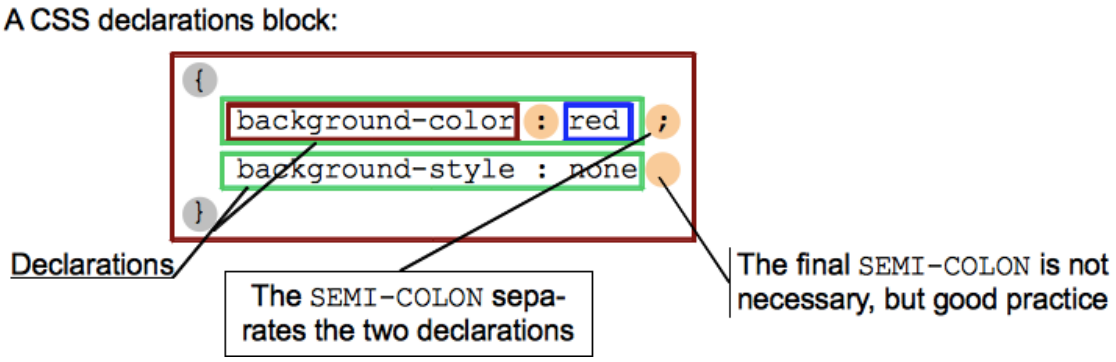
CSS deklarační blok

Deklarace jsou seskupeny do **bloků**, které jsou vyjádřeny složenými závorkami. Bloky se mohou do sebe zanořovat, ale pro zachování validity musí být všechny závorky uzavřeny.



Obrázek 2.3: Ukázka vnější syntaxe deklaračního bloku [13]

Tyto bloky jsou přirozeně nazývány **deklarační bloky** a obsažené deklarace jsou od sebe oddělovány středníkem `";"`. Bílé znaky mezi nimi jsou opět ignorovány. Středník za poslední deklarací není syntaxí vyžadován, pro přehlednost a zachování konzistence je ovšem doporučován [10].



Obrázek 2.4: Ukázka vnitřní syntaxe deklaračního bloku [14]

CSS selektor

Pro určení, kterých elementů se deklarační blok týká, se využívají **selektory**. Existuje několik typů:

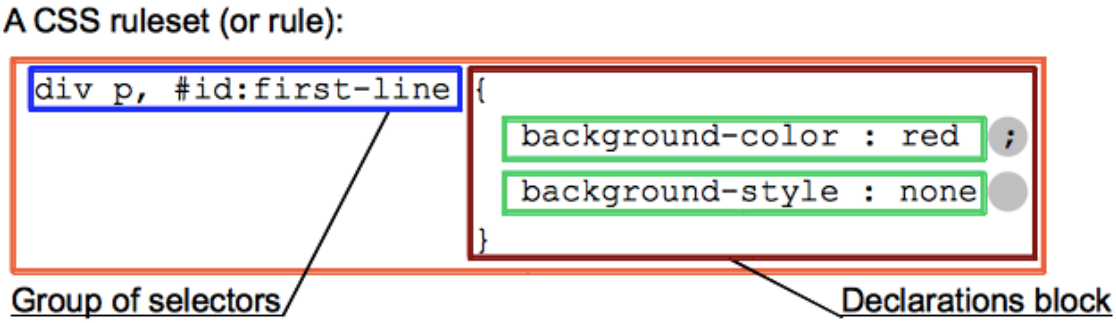
- **Typový selektor** – značený pomocí jména elementu;
  např. `"input"` pro selekci elementů `<input>`
- **Selektor třídy** – značený tečkou `"."`, následovanou názvem třídy elementu;
  např. `".clsName"` pro selekci elementů s touto třídou `<element class="clsName">`
- **Selektor ID** – značený mřížkou `"#"`, následovanou názvem identifikátoru elementu;
  např. `"#idName"` pro selekci elementů s identifikátorem `<element id="idName">`
- **Univerzální selektor** – značený pomocí hvězdičky `"*"` a vyjadřuje selekci jakéhokoliv elementu
- **Selektor atributu** – značený pomocí jména atributu a případně jeho hodnoty v hranatých závorkách; např. `"[attr=value]"` pro selekci `<element attr="value">` nebo `"[selected]"` pro selekci `<element selected>` apod.

Selektory lze také kombinovat s vyjádřením těchto vztahů:

- **Konjunkce** – selektory se napíší přímo za sebe;
  např. `"a.link"` pro selekci `<a class="link">`
- **Disjunkce** – selektory se oddělí čárkou `","`;
  např. `"a, p"` pro selekci odkazů `<a>` a odstavců `<p>`
- **Sousední sourozenec** – selektory se oddělí plusem `"+"`;
  např. `"p + a"` pro selekci odkazů `<a>` přímo za odstavcem `<p>`
- **Obecný sourozenec** – selektory se oddělí vlnovkou `"~"`;
  např. `"p ~ a"` pro selekci odkazů `<a>` následujících kdekoliv za odstavcem `<p>`
- **Přímý potomek** – selektory se oddělí znakem `">"`;
  např. `"p > a"` pro selekci odkazů `<a>` přímo uvnitř odstavce `<p>`
- **Obecný potomek** – selektory se oddělí mezerou;
  např. `"p a"` pro selekci odkazů `<a>` jakkoliv zanořených v odstavci `<p>`

CSS pravidlo

CSS umožňuje asociovat deklarační bloky pouze elementům vybraným selektorem. Každý (validní) deklarační blok je tak předcházen selektorem určujícím množinu relevantních elementů, na které se má blok aplikovat. Dvojice selektoru a deklaračního bloku se nazývá **pravidlo** (*ruleset*).



Obrázek 2.5: Ukázka syntaxe CSS pravidla [19]

Protože element dokumentu může vyhovovat několika selektorům a jeho vlastnosti tedy mohou být definovány v různých deklaracích jinak, CSS standard definuje která deklarace má přednost pomocí tzv. cascade[2] algoritmu. Zjednodušeně řečeno se pravidla aplikují v pořadí, v jakém jsou uvedeny ve zdrojovém kódu.

[2]Kaskáda a dědičnost – https://developer.mozilla.org/en-US/docs/Learn/CSS/Introduction_to_CSS/Cascade_and_inheritance

Kapitola 3

CSS syntaxe definice hodnoty

Pro vyjádření gramatik jednotlivých stylů v tomto dokumentu použiji *CSS syntaxi definice hodnoty* [21], což je formální gramatika pro definování množiny validních hodnot pro CSS vlastnosti nebo funkce.

Syntaxe popisuje které hodnoty jsou povoleny a interakce mezi nimi. Komponentami definic jsou *klíčové slovo*, *literál*, nebo hodnota vyjádřená *datovým typem* či jinou *CSS vlastností*. Je také možné pomocí kombinátorů vyjádřit vztah mezi komponentami nebo použitím multiplikátorů jejich očekávaný počet. *Entitou* pak označujeme komponentu nebo skupinu komponent, které mohou mít přiřazený multiplikátor.

3.1 Komponenty

- **Klíčové slovo** – slovo s předdefinovaným významem, je psáno přímo bez uvozovek; např. `auto`, `none` nebo `initial`.
- **Literál** – vyjadřuje výskyt znaku či sekvence znaků používaným *CSS syntaxí definice hodnoty* bez aplikace jeho významu, jsou obaleny uvozovkami;
  např. `'/'`, `','` a podobně.
- **Datový typ** – základní nebo odvozený datový typ, definovatelný pomocí *CSS syntaxe definice hodnoty*, je uzavřen do špičatých závorek;
  např. `<angle>`, `<string>`, `<název vlastnosti>` a podobně.

3.2 Kombinátory

- **Juxtapozice** - několik po sobě jdoucích entit oddělených pouze mezerami vyjadřuje tzv. juxtapozici, **všechny** entity se musí vyskytnout a to v **daném pořadí**;
  např. `"bold <length> , thin"`, čemuž vyhovují vstupy:

  – `bold 1em, thin`
  – `bold 0, thin`
  – `bold 2.5cm, thin`

  ale už ne:

  – `thin 1em, bold`

protože entity musí být v určeném pořadí a také:

  - `bold 1em thin`

protože všechny 4 entity jsou povinné a zde chybí čárka `","`.

- **Dvojitý ampersand** `&&` – oddělení dvou nebo více entit dvojitým ampersandem znamená, že **všechny** tyto entity se musí vyskytnout, ale nezáleží na jejich pořadí;
  např. `"bold && <length>"`, čemuž vyhovuje:

  - `bold 2em`
  - `0 bold`

  ale už ne:

  - `bold`

  protože chybí komponenta `"<length>"` a také:

  - `bold 1em bold`

  protože každá entita se smí vyskytnout pouze jednou.

- **Svislý oddělovač** `|` – oddělení dvou nebo více entit jedním svislým oddělovačem znamená, že **přesně jedna** z nich se musí vyskytnout, což se obvykle používá na výčet možných klíčových slov;
  např. `"<percentage> | <length> | left | right"`, čemuž vyhovuje:

  - `3%`
  - `left`
  - `10cm`

  ale už ne:

  - `right 3%`

  protože se může vyskytnou pouze jedna z entit a také:

  - `1em 2em`

  protože každá entita se smí vyskytnout maximálně jednou.

- **Dvojitý svislý oddělovač** `||` – oddělení dvou nebo více entit dvojitým svislým oddělovačem znamená, že **alespoň jedna** z nich se musí vyskytnout a nezáleží na jejich pořadí;
  např. `"<percentage> || <length> || inset"`, čemuž vyhovuje:

  - `inset`
  - `3% inset 10cm`
  - `10cm 3%`

  ale už ne:

  - `10cm 5cm`

  protože každý entita se smí vyskytnout maximálně jednou.

- **Hranaté závorky** `[]` – uzavření několika entit do hranatých závorek vyjadřuje transformaci do **jedné komponenty**, využívá se k zapouzdření a závorky je možné do sebe zanořovat;
  např. `"bold [ thin && <length> ]"`, čemuž vyhovují vstupy:
  - `bold thin 2em`
  - `bold 0 thin`

  ale už ne:
  - `thin bold 3em`

  protože `bold` je v juxtapozici se skupinou uvnitř `[]` a musí se tedy nacházet před ní.

3.3 Multiplikátory

- *Žádný* – neboli komponenta samotná, vyjadřuje výskyt **právě jednou**.
- **Mřížka** `#` – vyjadřuje výskyt entity **jednou nebo víckrát** s oddělením čárkami `","`.
- **Složené závorky** `{M, N}` – obsahující 2 celé čísla oddělené čárkou, vyjadřují výskyt entity **alespoň** `M`-krát a **maximálně** `N`-krát s oddělením mezerami.
- **Hvězdička** `*` – ekvivalentní s `{0,∞}`
- **Plus** `+` – ekvivalentní s `{1,∞}`
- **Otazník** `?` – ekvivalentní s `{0,1}`
- **Vykřičník** `!` – použití za skupinou vyjadřuje, že výraz musí vyprodukovat alespoň jednu hodnotu;
  např. `"[ bold? smaller? ]!"`, čemuž vyhovuje:
  - `bold`
  - `smaller`
  - `bold smaller`

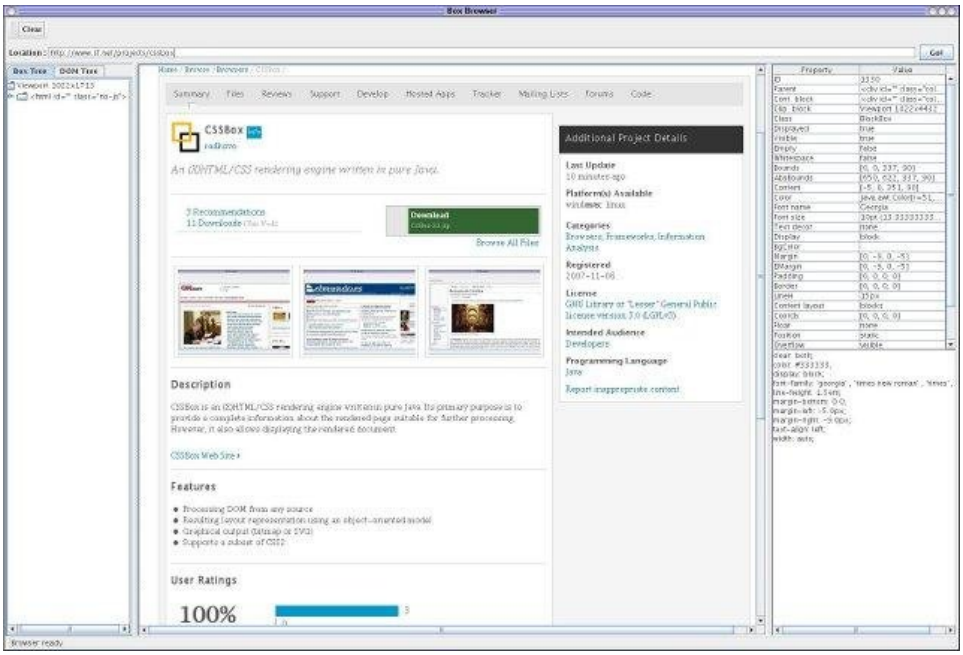
  ale už ne prázdný výraz.

Kapitola 4

Shrnutí dosavadního stavu

Obrázek 4.1: Logo projektu CSSBox [2]

Projekt CSSBox je (X)HTML/CSS renderovací engine napsaný v čisté Javě. Jedná se o open source projekt založený v roce 2007 Ing. Radkem Burgetem. Jeho hlavním účelem je poskytnutí kompletních a dále využitelných informací ohledně obsahu a rozložení renderované stránky. Mimo to ale může také být použit k prohlížení vykreslených dokumentů v Java Swingové aplikaci [2].

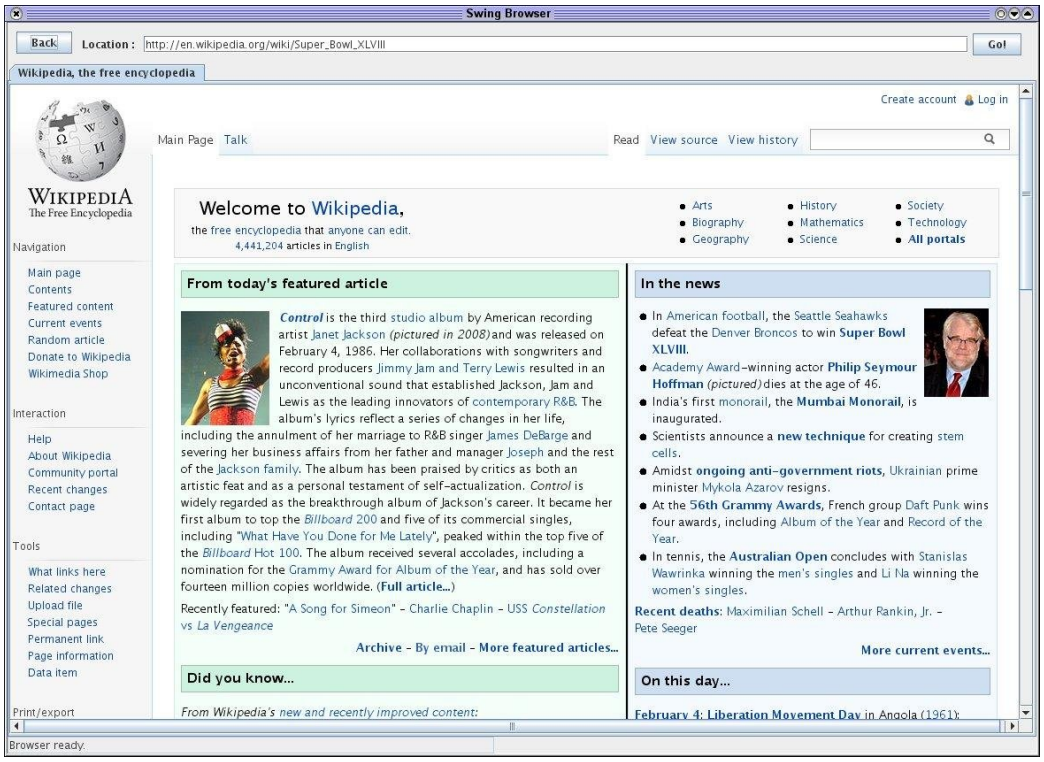


Obrázek 4.2: Uživatelské rozhraní zabudovaného prohlížeče [1]

CSSBox dále obsahuje 4 podprojekty s vlastním zaměřením na určitou oblast požadované funkcionality. Těmito sekcemi jsou:

SwingBox

Jedná se to Java Swingovou komponentu, která umožňuje zobrazení (X)HTML dokumentů včetně podpory CSS. Je navržena jako náhrada za `JEditorPane` s výrazně lepšími výsledky vykreslování. SwingBox je psán v čisté Javě a k vykreslení dokumentů využívá CSSBox renderovací engine [6].



Obrázek 4.3: Prostředí prohlížeče SwingBox [8]

Pdf2Dom

Pdf2Dom je PDF parser, který dokument převede do HTML DOM reprezentace. Výsledný DOM strom může poté být serializován do HTML souboru nebo dále zpracováván. Projekt je založen na knihovně Apache PDFBox™. K dosažení co nejvyšší vizuální podobnosti mezi převáděným PDF dokumentem a vygenerovanou HTML stránkou jsou použity inline CSS definice.

Součástí distribučního balíčku je nástroj pro konverzi PDF do HTML pro příkazový řádek. Pdf2Dom může být také využit jako nezávislá Java knihovna se standardním DOM rozhraním. V projektu CSSBox je využita jako alternativní parser, čímž mu umožňuje analyzovat dokumenty PDF [5].

WebVector

Projekt slouží ke konverzi HTML dokumentu do vektorového či bitmapového obrázku. K dispozici je převod do formátu SVG a PNG. Soubory SVG (Standard Vector Graphics) mohou samozřejmě být dále upravovány ve vektorových grafických editorech, např. Inkscape [7].

jStyleParser

CSS parser a analyzátor napsaný v čisté Javě. K dispozici je vlastní aplikační rozhraní, umožňující efektivní zpracovávání CSS stylů přímo v Javě, tím že mapuje hodnoty stylů na příslušné datové typy. Zanalyzované CSS styly je také možné aplikovat na DOM, reprezentující HTML nebo XML dokument a vygenerovat výsledné vlastnosti jednotlivých elementů.

V současné době jsou podporovány všechny vlastnosti z CSS 2.1 a také velká část nových z CSS3. Přínos této práce je zaměřen právě na rozšíření podpory vlastností tohoto analyzátoru [3].

4.1 Význam subprojektu jStyleParser

Úkolem projektu jStyleParser je propojit vnější data dokumentu v typické webové formě (HTML + CSS) a dále je předefinovaným rozhraním prezentovat zbytku projektu CSSBox, za pomocí vytvoření patřičných objektů.

Je třeba dokument načíst, zpracovat definice elementů dokumentu a dostupných kaskádových stylů, ve všech 3 formách, inline zápisu jako součást elementů, uvnitř elementů style i importovaných definic v externích souborech. Dále je otestována jejich validita a s ohledem na prioritu překrývajících se definic, podle jejich zdroje, jsou informace převedeny do vnitřní reprezentace, tak aby bylo možné jednoduše vyhodnotit styly jednotlivých elementů. Nedefinované a nevalidní vlastnosti jsou doplněny výchozími hodnotami.

Výpis podporovaných CSS vlastností je možné nalézt v příloze A.

4.2 Hierarchie datových struktur

CSSFactory

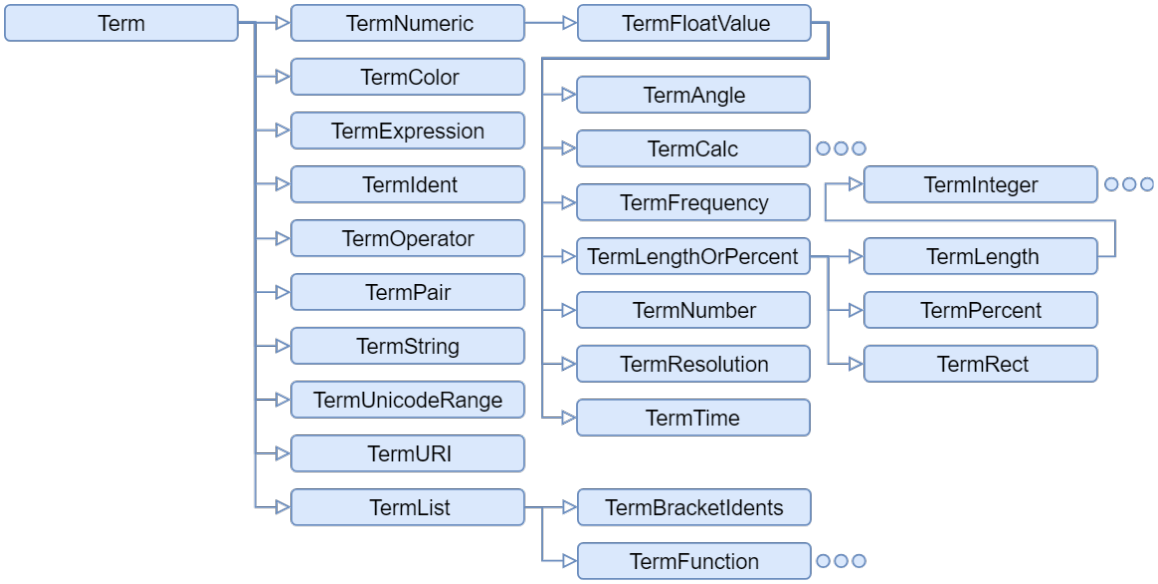
Globální statická třída a zároveň abstraktní předloha ke všem ostatní továrnám, zaměřujícím se na konkrétní problematiku. Obsahuje veřejné statické aplikační rozhraní (4.4), jak pro získání ostatních továren a dalších pomocných instancí, tak pro provedení všech globálních akcí projektu, a tedy zastřešuje práci s ním. Továrna zajišťuje vytvoření a navrácení pouze jediné instance pro každou pomocnou třídu.

ParserFactory a Parser

Třída zaměřující se na vytvoření a konfiguraci instance `CSSParseru` a následné vrácení produktu ve formě objektu `StyleSheet`. Samotná třída `CSSParser` je sice globálně dostupná, ovšem většinou není důvod s ní přímo manipulovat. Jejím úkolem je zjednodušeně řečeno převedení textové formy CSS pravidel, deklarací a selektorů na požadovaný výstup ve formě objektů `Rule`, `Declaration` a `Selector` a ty pak vrátit uvnitř `StyleSheet`.

TermFactory a Term

Tato továrna obsahuje metody pro vytvoření všech možných druhů termu. Samotný `Term` je parametrické rozhraní v nejobecnější formě, ke kterému jsou postupně přiřazovány vlastnosti a zodpovědnosti systémem vzájemné stromové implementace. Tímto způsobem se přes obecnější termy jako numerické hodnoty (`TermNumeric`) dostaneme až k velmi specifickým implementacím jako obdélník se 4 body pro vlastnost `clip`.



Obrázek 4.4: Diagram dědičnosti termů

RuleFactory, Rule, Selector a Declaration

Továrna pro vytváření pravidel, selektorů a deklarací, jakožto objektů reprezentující většinu CSS gramatiky. Pravidla reprezentují blok `RuleSet` objektů, které jednoduše spojují selektor a seznam relevantních deklarací, s potenciální podmínkou, zda se mají aplikovat či ne. Selektor obsahuje informace o tom, kterých elementů se následující deklarační blok týká. Deklarace pak vyjadřuje samotnou dvojici CSS vlastnosti a přiřazených termů.

DeclarationTransformer

Tento transformátor je zodpovědný za validaci syntaxe a převod deklarace na hodnoty vhodné pro strukturu `NodeData`. Třída mimo jiné obsahuje metody pro zpracování všech podporovaných CSS vlastností (A) definovaných uvnitř `SupportedCSS`.

StyleSheet, StyleMap a NodeData

Struktury `StyleSheet` a `StyleMap` obalují informace o CSS vlastnostech v různých fázích programu. Objekt StyleSheet obsahuje seznam zanalyzovaných CSS pravidel (`Rule`) a reprezentuje tedy všechny načtené CSS styly před jejich aplikací. Naopak `StyleMap` představuje již mapování mezi elementy dokumentu a jejich přiřazenými styly. Každý tento vztah je reprezentován objektem `NodeData`, obsahujícím vlastnosti ve formě `CSSProperty` a případně i přiřazené `Termy`.

SupportedCSS a Property

Třída obsahuje seznam podporovaných CSS vlastností (A) a jejich výchozích hodnot. K dispozici je také několik pomocných metod na získávání pořadí, jmen a vlastností samotných, využívaných převážně v testech.

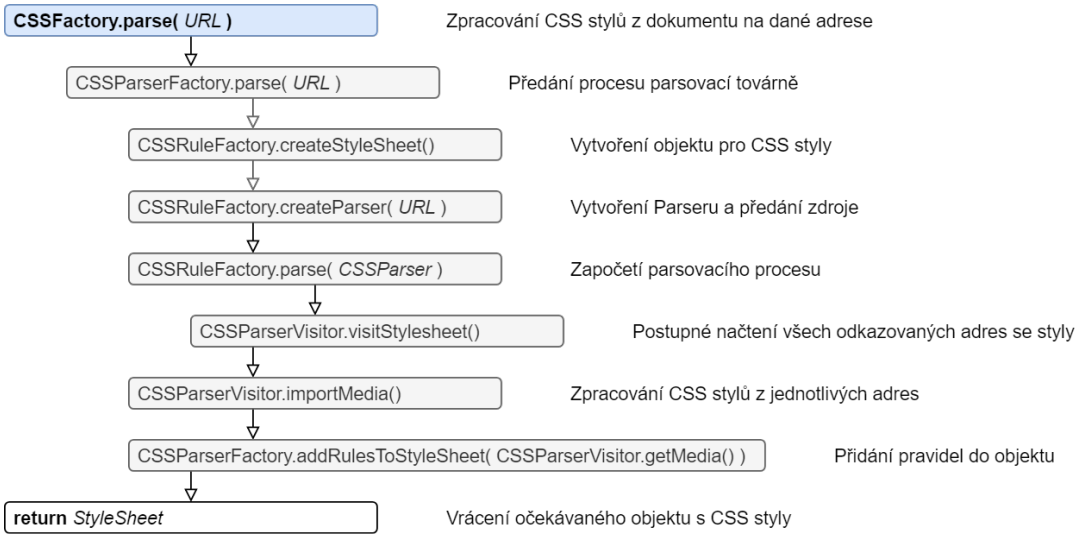
Každá vlastnost je reprezentována specializovanou implementací enumerace `CSSProperty` s nadefinovanými hodnotami pro všechny validní vstupy. K jednoduššímu

rozlišení mezi konstantami a proměnnými hodnotami je zavedeno pravidlo, že hodnoty velkými písmeny (UPPERCASE) jsou konstantní a hodnoty malými písmeny (lowercase) jsou proměnné s dodatečnou informací, dále získatelnou z `NodeData`.

4.3 Diagram volání

Průběh algoritmu vykonávajícího typické akce projektu jStylyParse je zcela deterministický. Postup předávání dat a zodpovědností 2 typických akcí je zde vyobrazen diagramem volání metod.

CSSFactory – parse()

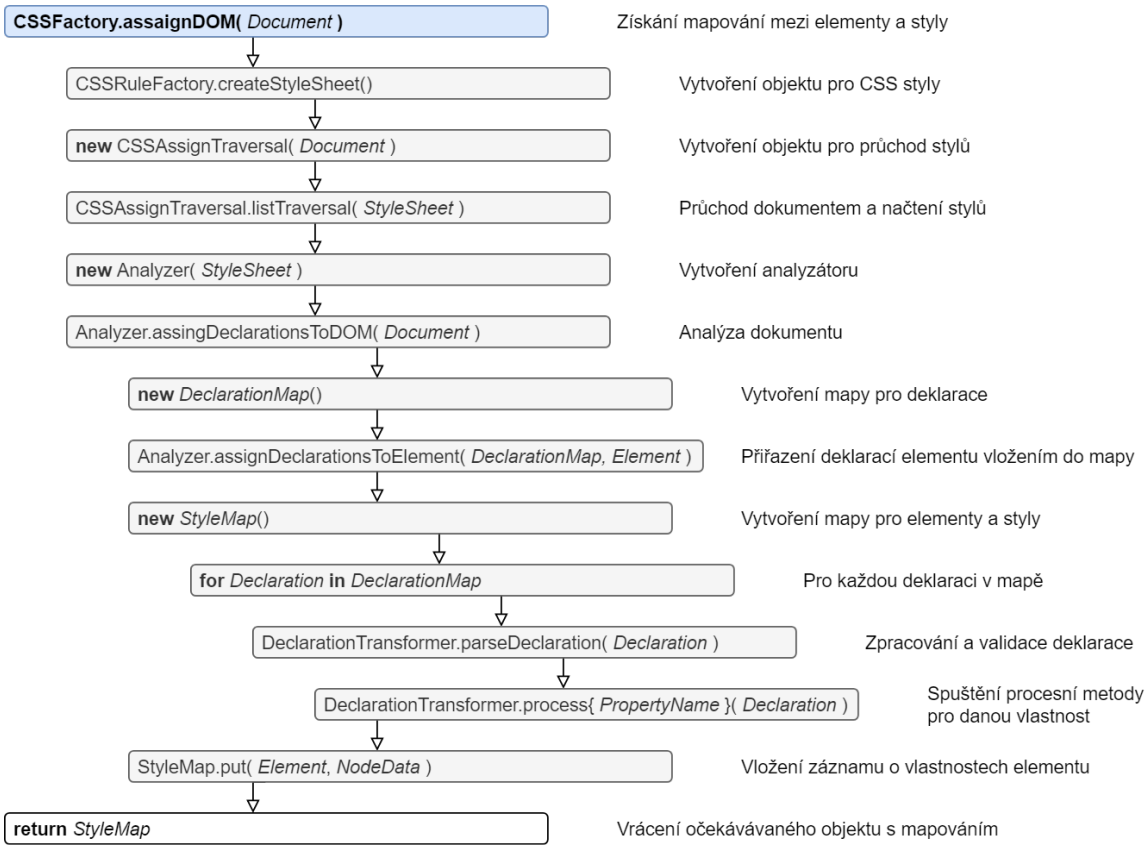


Obrázek 4.5: Diagram volání metody CSSFactory.parse()

Diagram znázorňuje zjednodušený životní cyklus vykonávání metody pro získání objektu `StyleSheet` s CSS styly z dokumentu na dané URL adrese.

1. `CSSFactory` nejprve předá proces parsovací továrně zodpovědné za podobné akce.
2. Ta vytvoří prázdný objekt `StyleSheet`, do kterého později budou přidány informace o stylech ve formě pravidel.
3. Vytvoří se objekt `CSSParser` a předá se mu URL adresa dokumentu.
4. Po zavolání jeho parsovací metody tzv. `Visitor` postupně projde všechny odkazované adresy na externí styly a ty rekurzivním voláním uloží.
5. Uložené adresy jsou pak lineárně načteny a obsažené styly naimportovány.
6. Následuje pak jen konverze do vnitřní reprezentace a vložení objektů do připraveného `StyleSheet`.
7. Výsledný objekt je nakonec vrácen externímu procesu.

CSSFactory – assaignDOM()



Obrázek 4.6: Diagram volání metody CSSFactory.assaignDOM()

Diagram znázorňuje zjednodušený životní cyklus vykonávání metody pro získání mapování mezi elementy a styly ve formě `StyleMap` ze zadaného dokumentu.

1. CSSFactory nejprve vytvoří prázdný objekt `StyleSheet`, do kterého později budou přidány informace o stylech.
2. Poté projde a načte styly z dokumentu pomocí třídy `CSSAssignTraversal` a naplní tak připravený `StyleSheet`.
3. Ten je předáním do konstruktoru použit k inicializaci `Analyzeru`.
4. Vytvoří se mapovací struktura `StyleMap`.
5. Spustí se analýza dokumentu, která podle selektorů nejprve přiřadí deklarace k elementům z dokumentu a následně otestuje jejich validitu pomocí procesních metod z `DeclarationTransformeru`.
6. Validní deklarace včetně jejich hodnot jsou vloženy do `NodeData` objektů, které jsou přidány do `StyleMap`.
7. Výsledná mapa je nakonec vrácena externímu procesu.

4.4 Aplikační rozhraní

Projekt jStyleParser poskytuje veřejné aplikační rozhraní určené k propojení s externími projekty. Komunikace sestává převážně z volání metod továrny `CSSFactory` vracejících patřičné datové struktury, které jsou buď přímo analyzovány, či předány dál.

Parsování stylů

Tato funkcionalita může být využita pro parsování individuálních stylů získaných ze vzdáleného či lokálního souboru nebo řetězce. Pro tento účel byly ve třídě `CSSFactory` definovány tyto statické metody:

```
StyleSheet parse(URL url, String encoding);
```

Nejobecnější metoda, spravuje data dostupné na dané URL adrese. Jako kódování se použije buď `encoding`, pokud je specifikováno, nebo výchozí,

```
StyleSheet parse(URL url, NetworkProcessor network, String encoding);
```

která má stejný efekt, jen je navíc specifikován `NetworkProcessor`,

```
StyleSheet parse(String fileName, String encoding);
```

která uvnitř transformuje `fileName` na URL a

```
StyleSheet parse(String css, URL base);
```

kterou je možné použít na parsování vnitřních CSS deklarací uvnitř `<style>` elementů.

Během parsovacího procesu jsou automaticky importovány veškeré odkazované styly pomocí `@import` pravidel. Toto chování se dá omezit nebo úplně deaktivovat pomocí metody

```
void setAutoImportMedia(MediaSpec media);
```

předáním `MediaSpec` specifikace. Např. `MediaSpec("screen")` importuje jen styly validní pro medium `"screen"`, `MediaSpecAll()` importuje vše a `MediaSpecNone()` deaktivuje automatický import úplně.

Pro získání naimportovaných stylů referencovaných pomocí jejich URL adres je možné poskytnout vlastní implementaci `NetworkProcesoru`, která zajistí získání `InputStreamu` z daných URL adres. Výchozí vystavěný `NetworkProcesor` je založen na standardní Java `URLConnection` infrastruktuře. Výsledkem parsování je `StyleSheet` objekt, který je v podstatě kolekce pravidel nalezených ve stylových definicích. Tento objekt může být použit jako vstup pro DOM analyzátor nebo manuálně zpracován za účelem získání obsažených pravidel a deklarací [4].

Získání stylů použitých v HTML dokumentu

Globální objekt `CSSFactory` obsahuje metodu, která zanalyzuje HTML či XML dokument reprezentovaný DOM stromem a vyparsuje všechny použité styly.

```
StyleSheet getUsedStyles(Document doc, String encoding, URL base, MediaSpec media);
```

Tato metoda zpracuje všechny odkazované styly, související se specifikovaným typem média, a vrátí jedinou `StyleSheet` strukturu, obsahující veškeré relevantní CSS pravidla. Parametr

URL base je potenciálně použit na překlad relativních adres uvnitř dokumentu a String `encoding` je opět použit pro určení výchozího kódování textu [4].

Analýza DOM stromu

Důvodem pro analýzu DOM stromu je aplikace získaných stylů na konkrétní HTML nebo XML dokument a vyhodnocení relevantních stylů jednotlivých elementů dokumentu. Výsledkem analýzy je mapování mezi elementy DOM stromu a odpovídajícími CSS deklaracemi. Toto mapování je možné použít pro vykreslení dokumentu nebo bližší analýzu jeho struktury.

CSS styly ve formě `StyleSheet` instance získané z `CSSParseru` je možné předat `Analyzeru` pomocí konstruktoru. Na vytvořené instanci `Analyzeru` je pak možné volat tyto metody:

```
StyleMap evaluateDOM(Document doc, String media, boolean inherit);
StyleMap evaluateDOM(Document doc, MediaSpec media, boolean inherit);
```

Ty vytvoří mapování mezi DOM elementy a jejich CSS deklaracemi podle specifikovaného média a dědičnosti.

Pro vyhodnocení stylů individuálních elementu bez vyhodnocení celého DOM stromu je také možné použít třídu `DirectAnalyzer`, která umožňuje vyhodnotit vlastnosti jednotlivých uzlů bez vytváření celé mapy. Jeho využití pro vyhodnocení velkého množství elementů ale zapříčiní znatelné snížení efektivity [4].

Zjednodušený přímý přístup

Globální objekt `CSSFactory` také poskytuje zjednodušený způsob pro zpracování (X)HTML dokumentu přímo do stylové mapy.

```
StyleMap assignDOM(Document doc, String encoding, URL base, MediaSpec media,
    boolean useInheritance);
```

Metoda automaticky stáhne a zpracuje všechny vnitřní i externí styly odkazované z dokumentu pro daný typ média a poté spustí proces mapování na jednotlivé elementy. K dispozici je také několik variant této metody s možností specifikace pseudo třídy pomocí `MatchCondition` a/nebo vlastního `NetworkProcesoru` pro získávání importovaných stylů [4].

Získání stylů pro DOM elementy

Po dokončení analýzy, zpracováním stylů, elementů dokumentu a vytvořením mapování mezi nimi, máme k dispozici datovou strukturu `StyleMap`. Struktura dědí standardní Javovské rozhraní `Map`, požadované styly pro daný DOM element jsou tedy získány metodou

```
NodeData get(Element elem);
```

Struktura `NodeData` obsahuje dvě základní metody

```
<T extends CSSProperty> T getProperty(String name, boolean includeInherited);
```

vracející vlastnosti ve formě `CSSProperty`. Přiřazená hodnota vlastnosti může být buď vyjádřena patřičnou konstantou, nebo proměnnou hodnotou s dodatečnou informací získatelnou pomocí metody

```
<T extends Term> T getValue(Class<T> clazz, String name, boolean includeInherited);
```

vracející hodnotu ve tvaru `Termu`. Pro obě tyto metody existují ekvivalentní alternativy

```
<T extends CSSProperty> T getSpecifiedProperty(String name);
<T extends Term> T getSpecifiedValue(Class<T> clazz, String name);
```

které při nenalezení validní definice místo `null` navracejí výchozí hodnoty pro danou CSS vlastnost [4].

Získání stylů pro pseudo-elementy

CSS specifikace umožňuje adresovat specifickou část již existujících DOM elementů. Styly těchto pseudo-elementů mohou být získány pomocí následujících metod objektu `StyleMap`:

```
boolean hasPseudo(Element elem, Selector.PseudoElementType pseudo);
```

otestuje, zda daný element obsahuje nějaké deklarace stylů pro vybraný pseudo-element.

```
NodeData get(Element elem, Selector.PseudoElementType pseudo);
```

vrátí požadované styly pro vybraný pseudo-element.

Příkladem `Selector.PseudoElementType` pseudo-elementu jsou hodnoty `BEFORE`, `AFTER` nebo `FIRST_LETTER`.

Příklad použití

```
// Získaní stylové mapy
StyleMap map = CSSFactory.assignDOM(doc, encoding, base, medium, true);
NodeData style = map.get(element);

// Získaní typu přiražené hodnoty
CSSProperty.Margin prop = style.getProperty("margin-top");
System.out.println("margin-top=" + prop);

// Vypsaní hodnoty proměnné nebo konstanty samotné
String margin;
if (prop == Margin.length || prop == Margin.percentage) {
    TermLength mtop = style.getValue(TermLength.class, "margin-top");
    margin = mtop.toString();
} else {
    margin = prop.toString();
}
System.out.println("element.margin-top = " + margin);
```

Kapitola 5

Zhodnocení současného stavu

Projekt v současné době je plně funkční a podporuje analýzu zpracování a aplikaci prakticky všech standardních kaskádových stylů verze CSS2.1 a také několika novějších z verze CSS3. Samotná logika a aplikační rozhraní je na dobré úrovni použitelnosti a potenciální využitelnost projektu ke zpracování moderních webových dokumentů lze navýšit rozšířením podpory o chybějící, převážně nové, styly.

5.1 Současná omezení projektu

Po prvotní analýze podporovaných stylů v projektu a srovnání s vývojářskými referencemi (Mozilla [20], W3Schools [28]) jsem identifikoval 78 chybějících stylů, kterým je doporučována všeobecná kompatibilita. Jejich výpis je možné nalézt v příloze B.

5.2 Rozhodnutí o rozšíření

Po prezentaci nalezené potenciální oblasti, k rozšíření projektu ve formě chybějící podpory pro některé CSS vlastnosti, vedoucímu práce Radku Burgetu, bylo rozhodnuto, že bude rozšířen tímto způsobem. Rozšíření nenaruší současnou funkcionalitu ani kompatibilitu a bude doplněno sadou testů zaručující funkčnost přidaného kódu při každém spuštění buildu projektu. V projektu jStyleParser bude postupně přidána podpora pro tyto dosud chybějící styly a to ve 3 fázích.

1. V první testovací fázi se seznámím s obecnou strukturou kódu a vyzkouším si rozšíření na implementaci stínu elementu (`box-shadow`) a příslušných testů. Výsledek poté prezentuji vedoucímu, který změny vyhodnotí, případně vyjádří nedostatky k opravení.
2. Druhá fáze zahrnuje implementaci všech vlastností souvisejících s mřížkovým rozložením elementů (`grid layout`), které budou v budoucnu také zapotřebí pro propojení se změnami v rendereru, přidávanými jiným studentem, implementující právě vykreslení tohoto rozložení. Gramatika těchto vlastností je značně komplexní a jejich vhodná implementace bude tedy tvořit jádro rozšíření.
3. V poslední fázi pak dojde k přidání většího množství jednodušších, často podobných, vlastností souvisejících s CSS animacemi (`animation`) a přechody `transition`. S poměrně malým vynaložením času se tak značně navýší počet podporovaných vlastností projektu.

Všechny přidané vlastnosti (33) jsou označeny zeleně v seznamu chybějících (B).

Kapitola 6

Návrh řešení

Tato sekce popisuje můj postup a myšlenkové pochody při navrhování implementace dohodnutého rozšíření. Bude třeba nejprve nastudovat gramatiky přidávaných vlastností, vyjadřující všechny validní hodnoty. Získané znalosti, nebo gramatiku samotnou, převést na způsob implementace následující vnitřní logiku projektu. A nakonec nalézt vhodná umístění ke vložení nového kódu.

6.1 Studium gramatiky

Jako zdroje pro studium gramatiky CSS vlastností jsem zvolil *MDN web docs* od Mozilly [20] a pro srovnání a často jednodušší pochopení také *W3Schools.com* [28]. Oba tyto materiály poskytují užitečné informace, přesné definice gramatiky (i když ne vždy se stejnou syntaxí) a také příklady použití pro každou[1] CSS vlastnost. Většinou je k dispozici také interaktivní zobrazovač typických kombinací hodnot s možností jejich úpravy a testu validace.

6.2 Programování řízené testy

Protože zadání kromě sekcí nespecifikuje přesné pořadí vypracování dílčích úkolů a protože tento způsob vývoje mi vyhovuje, kvůli logickému oddělení problematik a jednoduché průběžné evaluaci postupu, jsem se rozhodl pro vývoj použít přístup zvaný „Programování řízené testy“ (*Test-driven development* [29]).

Před započetím mých úprav tedy nejprve spustím všechny již existující testy a poznačím si současný stav, úspěšnost a výpisy do konzole. Poté vytvořím testy validující očekávanou funkčnost kódu, kterého plánuji přidat, a také jeho očekávané selhání. Následuje samotná implementace požadované logiky doprovázená pravidelným spouštěním předpřipravených testů. V průběhu psaní je také třeba periodicky kontrolovat výstupy ostatních testů a v případě selhání zjistit příčinu změny chování a tu co nejdřív opravit.

[1] Každou vlastnost související s touto prací. Databáze *MDN web docs* je ve srovnání s *W3Schools.com* o dost rozšířenější, ovšem pro účely této práce jsou oba zdroje naprosto dostatečné.

Kapitola 7

Popis vlastní práce

Kapitola popisuje realizaci navržených změn ze sekce 6. Obecný účel a rozmístění kódu. Analýzu přidávaných vlastností, jejich význam, parametry a gramatiku. Jakým způsobem jsou vlastnosti implementovány a je zajištěna jejich správná funkcionalita.

7.1 Rozmístění kódu

Testy

Již po krátké analýze bylo zřejmé, jakým způsobem se do projektu vkládají testy. Jak v kořenovém adresáři, tak balíčku (jemném prostoru), se nachází umístění `"test"` obsahující všechny testující třídy projektu. Ty mají běžnou strukturu jednotkových testů spouštěných externě přes anotace `@Test`, s případnou inicializací `@Before` a finalizací `@After`.

O úroveň níž v `"test/resources"` se pak nachází ostatní soubory pro testování, jako zdrojové HTML dokumenty, CSS styly, obrázky a podobně.

Deklarace stylů

Pro deklaraci stylu je nejprve nutné vytvořit novou implementaci `CSSProperty` ve formě enumerace uvnitř samotného `CSSProperty` rozhraní (z důvodu konzistence). V něm definovat 3 globální hodnoty `"inherit"`, `"initial"` a `"unset"`, plus vlastní konstanty a proměnné vhodně reprezentující validní kombinace hodnot vlastnosti. A z důvodu limitace jazyka Javy ve formě nemožné dědičnosti enumerací, také implementovat všechny metody rozhraní `CSSProperty`:

```
boolean inherited(); // Zda je daná vlastnost děděna po rodičích

boolean equalsInherit(); // Zda je přiřazena hodnota "inherit"

boolean equalsInitial(); // Zda je přiřazena hodnota "initial"

boolean equalsUnset(); // Zda je přiřazena hodnota "unset"

String toString(); // Převedení současné hodnoty na řetězec
```

Následně je pak nutné zaregistrovat styl uvnitř třídy `SupportedCSS3`, přidáním dvojice názvu a výchozí hodnoty, reprezentované právě vytvořenou implementací `CSSProperty`, do mapy podporovaných vlastností. V případě, že je výchozí hodnota reprezentována proměnnou, je navíc potřeba daný `Term` vložit do mapy výchozích hodnot.

Validace stylů a přiřazování termů

Přidáním stylu do mapy podporovaných vlastností dojde k jeho aktivaci, což znamená, že se `CSSParser` v průběhu analýzy, pokud na daný styl narazí, pokusí zavolat procesní metodu, zodpovědnou za validaci a přiřazení patřičné hodnoty. Tyto metody mají předepsaný formát a nacházejí se uvnitř implementace `DeclarationTransformeru`.

```
@SuppressWarnings("unused")
boolean process[název vlastnosti ve velbloudí notaci](Declaration d, Map<String,
    CSSProperty> properties, Map<String, Term> values);
```

Metoda v parametrech dostane patřičnou deklaraci obsahující `Termy` ke zvalidování a reference na mapy pro výsledné `CSSProperty` vlastnosti a `Term` hodnoty, které se později použijí k vytvoření `StyleMap` objektu.

Anotace pro umlčení varování o nepoužití těchto privátních metod není nutná, ale vzhledem k tomu, že jsou volány pouze přes `Method.invoke()`, je vhodné je zachovat, jinak je vývojové prostředí považuje za nikdy nepoužité.

Deklarace nových Termů

Posledním přidaným kódem jsou nově deklarované `Termy`, vytvořením nových a úpravou existujících rozhraní a tříd, souvisejících s jejich reprezentací. Je třeba identifikovat všechny potenciálně využívané termy z gramatik přidávaných stylů a zajistit, že je `CSSParser` zná a správně reprezentuje.

7.2 Testovací nástroje

Potom, co jsem zjistil, že v projektu dosud neexistuje žádný specializovaný nástroj na zjednodušení testování implementací vlastností a současné testy tohoto typu jsou buď psány pokaždé mírně odlišným způsobem, nebo chybí úplně, jsem se rozhodl si napsat vlastní. Věděl jsem, že je zapotřebí vytvořit testy implementace 33 CSS vlastností, které by měly být dostatečně podobné na to, aby se daly zobecnit, parametrizovat a implementovat externě.

Výsledkem této motivace je třída `TestUtils`, obsahující 2 statické třídy a 2 pomocné testovací metody.

TestData

Datová struktura `TestData` slouží k reprezentaci jednoho testu vlastnosti na elementu daném jeho CSS identifikátorem. Celkově se skládá z identifikátoru elementu, názvu testované CSS vlastnosti, její očekávané formy `CSSProperty` a potenciálně také hodnoty termu, zde reprezentované obecným objektem. Pro přehlednost jsou tedy definovány 2 konstruktory pro oba případy, s definovanou očekávanou hodnotou termu i bez ní.

```
TestData(String id, String propName, CSSProperty expProperty);
TestData(String id, String propName, CSSProperty expProperty, Object expTerm);
```

NameGenerator

Třída `NameGenerator` slouží k dalšímu usnadnění psaní testů generováním identifikátorů testovaných elementů. Při dodržení jednoduché konvence pojmenování identifikátorů ve zdrojovém dokumentu ve formátu `"{jméno}{n-tý element}"` můžeme toto jméno v testech specifikovat pouze jednou a jeho iterativní změny si nechat generovat touto třídou. Počáteční hodnota `n` je 0.

Použití `NameGeneratoru` je velmi jednoduché. Začíná vytvořením generátoru zavoláním jeho konstruktoru a předáním neměnné části identifikátoru.

```
NameGenerator(String idName);
```

Následně voláním metod `next()` pro následující a `curr()` pro současný generujeme jednotlivé iterace tvaru identifikátoru. Zavoláním `setName(String idName)` je také možné přenastavit jméno identifikátoru a vynulovat hodnotu `n`.

Metody

Třída `TestUtils` obsahuje také 2 statické pomocné metody, jednu pro spuštění a vyhodnocení testů ve formě kolekce struktur `TestData` a druhou k přehledné komparaci seznamů s identifikací nerovné délky nebo první nesouhlasné položky.

```
// Spustí, ohodnotí a vypíše výsledky všech testu z kolekce
static void runTests(Iterable<TestData> tests, URL testedSource);

// Otestuje rovnost seznamů srovnáním jejich délky a ekvivalencí položek
static void compareLists(List a, List b);
```

```
private static final TermFactory tf = CSSFactory.getTermFactory();
private final List<TestData> _tests = new ArrayList<>();

@Before
public void prepare() {
    NameGenerator ng = new NameGenerator("grid");
    TermList list;

    list = tf.createList();
    list.add(tf.createLength(1f, Unit.fr).setOperator(Operator.SLASH));
    _tests.add(new TestData(ng.next(), "grid-template-columns", GridTemplateRowsColumns.list_values, list));
    _tests.add(new TestData(ng.curr(), "grid-auto-flow", GridAutoFlow.ROW));
    _tests.add(new TestData(ng.curr(), "grid", Grid.component_values));
}

@Test
public void test() {
    TestUtils.runTests(_tests, getClass().getResource("/simple/grid-layout.html"));
}
```

Obrázek 7.1: Ukázka použití třídy `TestUtils`

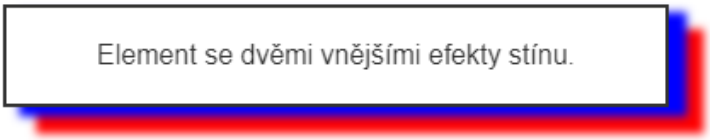
7.3 První fáze – Efekt stínu

V této fázi implementace byla do projektu přidána vlastnost `box-shadow` vyjadřující efekt stínu elementu. Definovatelnými parametry efektu jsou:

- odsazení v X-ové a Y-ové souřadnici
- míra rozmazání
- rádius rozpětí
- barva stínu
- a směr působení od okraje elementu (ven nebo dovnitř)

Obrázek 7.2: Vnitřní stín {`box-shadow: inset 0px 0px 20px blue;`}

Elementu je možné přiřadit i několik překrývajících se stínů, přičemž každý další se vykreslí pod předchozím.



Obrázek 7.3: Dva vnější stíny {`box-shadow: 10px 5px 5px blue, 20px 15px 5px red;`}

Gramatika

Gramatika validních hodnot vlastnosti `box-shadow` je definována takto (3):

```
none | <shadow>#

<shadow> = inset? && <length>{2,4} && <color>?
```

Definice pokračuje specifikací `length` a `color`, ovšem tato logika je již vyřešena použitím specializovaných termů `TermLenght` a `TermColor`.

Z gramatiky vyplývá, že minimální validní tvar je roven jedinému termu s klíčovým slovem `none`. Jinak je každý stín definován dvěma až čtyřmi délkovými termy a potenciálně jedním termem pro barvu a klíčovým slovem `inset`. Definice stínu je také možné řadit za sebe oddělené čárkami.

Testy

K zajištění správné implementace vlastnosti `box-shadow` byl vytvořen zdrojový dokument s testovanými kombinacemi hodnot a dále třída `BoxShadowTest`. Bylo vyzkoušeno správné

chování pro všechny typické validní kombinace pro jeden i více stínů, jako 2 až 4 hodnoty délky, barva, klíčové slovo `inset` a permutace mezi nimi. Dále několik nevalidních, u kterých se testovalo selhání a tedy vrácení výchozí hodnoty vlastnosti.

V této době ještě neexistovala třída `NameGenerator` a názvy identifikátoru místo iterací reflektují právě testovanou hodnotu.

Implementace

Pro reprezentaci této vlastnosti jsem vytvořil enumeraci `BoxShadow` implementující rozhraní `CSSProperty`. Kromě 3 globálních hodnot (7.1) je deklarována také konstanta `NONE("none")` toto klíčové slovo a dále proměnná `component_values("")` pro všechny ostatní validní kombinace termů.

Validace je zajištěna přímo procesní metodou `processBoxShadow()` ve třídě `DeclarationTransformer`. Metoda nejprve zavolá generický test na zpracování jedné konstanty od zadané `CSSProperty`.

V případě selhání se cyklicky testují a přiřazují termy do předem připraveného seznamu. Čítač délkových termů zajišťuje jejich správný počet a pomocí proměnných pro indexy posledního výskytu jednotlivých typů termů se testuje validní pořadí na vstupu. Definice jednoho stínu je ukončena buď skončením cyklu, nebo nalezením čárky ve formě operátoru na některém z termů, indikujícím začátek definice stínu dalšího. Před pokračováním ve validaci je tedy nutné vynulovat čítač i všechny indexové proměnné.

V případě chyby ve formě nalezení neočekávaného, či nenalezení očekávaného termu je proces ukončen s návratovou hodnotou `false`. Po úspěšném dokončení cyklu je do mapy vlastností vložena hodnota `BoxShadow.component_values` a do mapy hodnot seznam obsahující termy deklarace. Metoda vrací `true`.

7.4 Druhá fáze – Mřížkové rozložení

V druhé fázi implementace byly do projektu přidány vlastnosti týkající se mřížkového rozložení neboli *grid layout*. Jedná se o sérii CSS vlastností ovládajících rozmístění elementů na předefinované mřížce. Nastavením patřičné hodnoty do vlastnosti zobrazení některému elementu `"display: grid;"` nebo `"display: inline-grid;"` se z něj stává tzv. kontejner (*grid container*) a všechny synovské elementy jsou nyní jeho položkami (*grid item*).

Mřížka je daná protínajícími se vertikálními a horizontálními čarami. Jedny definují *sloupce* a druhé *řádky*. Elementy mohou být umístěny do této mřížky mezi tyto hranice.

7.4.1 Explicitní hranice

Pomocí `grid-template-columns` a `grid-template-rows` můžeme definovat sloupce a řádky mřížky. Z těchto jsou pak odvozené tzv. *hranice*, což jsou prostory mezi jakýmikoli dvěma čarami.

Pevná a přizpůsobivá velikost

Použitím jednotek délky s pevnou velikostí (např. pixely) je možné definovat mřížku o požadovaných rozměrech. Pro definici flexibilní mřížky, která se bude přizpůsobovat rozměru vnějšího elementu, je možné použít procenta nebo nově zavedenou jednotu `fr` (=*fraction* neboli zlomek), určenou právě pro tento účel [26].

Gramatika

Gramatika validních hodnot vlastností `grid-template-columns` a `grid-template-rows` je identická a je definována takto (3):

```
none | <track-list> | <auto-track-list>

<track-list> = [ <line-names>? [ <track-size> | <track-repeat> ] ]+ <line-names>?
<auto-track-list> =
               [ <line-names>? [ <fixed-size> | <fixed-repeat> ] ]* <line-names>?
               <auto-repeat>
               [ <line-names>? [ <fixed-size> | <fixed-repeat> ] ]* <line-names>?

<line-names> = '[' <custom-ident>* ']'
<track-size> = <track-breadth> | minmax( <inflexible-breadth> , <track-breadth> )
               | fit-content( [ <length> | <percentage> ] )
<track-repeat> = repeat( [ <positive-integer> ]
               , [ <line-names>? <track-size> ]+ <line-names>? )
<fixed-size> = <fixed-breadth> | minmax( <fixed-breadth> , <track-breadth> )
               | minmax( <inflexible-breadth> , <fixed-breadth> )
<fixed-repeat> = repeat( [ <positive-integer> ] ,
               [ <line-names>? <fixed-size> ]+ <line-names>? )
<auto-repeat> = repeat( [ auto-fill | auto-fit ] ,
               [ <line-names>? <fixed-size> ]+ <line-names>? )
```

```
<track-breadth> = <flex> | <length> | <percentage>
                | min-content | max-content | auto
<inflexible-breadth> = <length> | <percentage>
                | min-content | max-content | auto
<fixed-breadth> = <length> | <percentage>
```

Jak je vidět, definice je vcelku komplexní s potenciálním využitím vysoké abstrakce. Kromě běžného klíčového slova `none` je možné vlastnost definovat seznamem prvků, které mohou být vyjádřeny délkou, procentem, funkcemi `minmax()` a `fit-content()` zajišťujícími inteligentní vyjádření výšky či šířky na základě jejich parametrů i samotného obsahu buněk. Nebo místo opakující se formy seznamu použít funkci `repeat()`.

K tomu všemu lze ještě přidávat uživatelské identifikátory uvnitř hranatých závorek k pozdějšímu využití u jiných vlastností.

Testy

Testy pro vlastnosti `grid-template-columns` a `grid-template-rows` jsou umístěny ve třídě `GridLayoutTest`, zodpovědné za správnou funkci všech vlastností mřížkového rozložení. Díky využití nástrojů `TestUtils` otestování celkového rozsahu funkcionality zabralo jen 43 řádků.

Testuje se správná reprezentace nově přidaných klíčových slov (`auto`, `min-content`, `max-content`), funkcí s charakteristickými kombinacemi parametrů i uživatelskými identifikátory.

Implementace

Pro reprezentaci vlastností `grid-template-columns` a `grid-template-rows` jsem vytvořil enumeraci `GridTemplateRowsColumns`, implementující rozhraní `CSSProperty`. Kromě 3 globálních hodnot (7.1) jsou deklarovány také konstanty pro klíčová slova `NONE("none")`, `AUTO("auto")`, `MIN_CONTENT("min-content")` a `MAX_CONTENT("max-content")` a proměnná `list_values("")` pro všechny ostatní validní kombinace termů.

Validace je zajištěna metodou `processGridTemplateRowsColumns()` ve třídě `DeclarationTransformer` volanou z obou procesních metod pro sloupce i řádky. Metoda nejprve zavolá generický test na zpracování jedné konstanty od zadané `CSSProperty`.

V případě selhání se cyklicky testují a přiřazují termy do předem připraveného seznamu. Propouštěny jsou termy reprezentující identifikátor, jakožto potenciální klíčové slovo, identifikátor v hranatých závorkách, délku, procenta a nebo podporované funkce. Dvě boolovské proměnné zamezují vícenásobné použití funkce `repeat()` a po sobě jdoucí definici uživatelských identifikátorů.

V případě chyby ve formě nalezení neočekávaného, či nenalezení očekávaného termu je proces ukončen s návratovou hodnotou `false`. Po úspěšném dokončení cyklu je do mapy vlastností vložena hodnota `GridTemplateRowsColumns.list_values` a do mapy hodnot seznam obsahující termy deklarace. Metoda vrací `true`.

7.4.2 Implicitní hranice

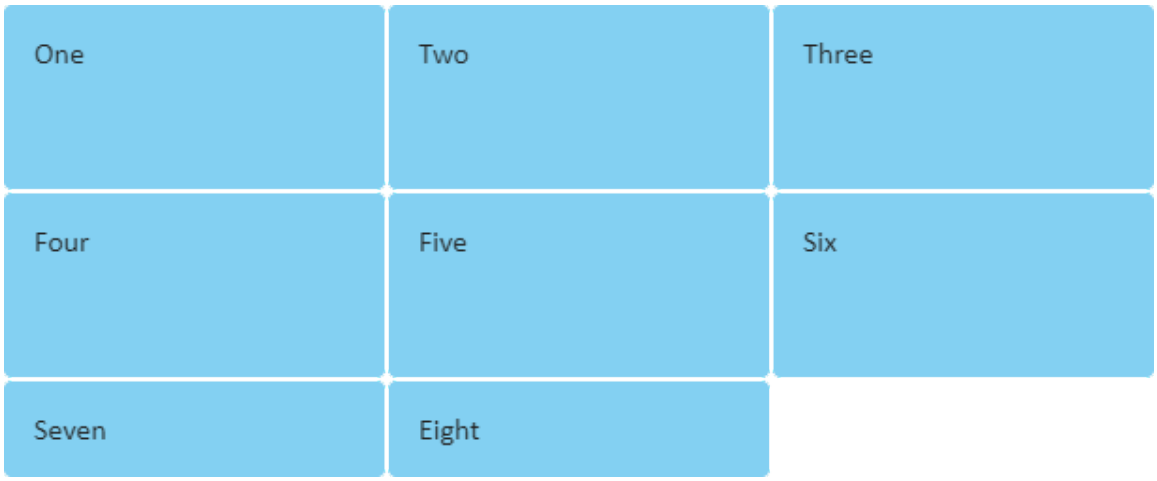
Pokud je obsah vložen mimo hranice explicitně definované mřížky, dogenerují se další sloupce a řádky a s nimi také tzv. implicitní hranice. Ve výchozím stavu mají tyto nové hranice automatický rozměr daný obsahem nebo navazujícími sloupci a řádky. Toto chování je

ale možné ovlivnit vlastnostmi `grid-auto-columns` a `grid-auto-rows` nastavujícími jejich implicitní šířku a výšku.

Následující příklad definuje 3 sloupce o šířce `1fr` (=1/3) a 2 řádky o výšce `100px`. Teoreticky do mřížky tedy vejde $2 \times 3 = 6$ buněk, ve skutečnosti je vloženo buněk 8. Přebytečné 2 buňky jsou tedy vloženy implicitně a jejich rozložení je tedy určeno automaticky, v tomto případně šířka podle buněk nad nimi a výška podle obsahu.

```
// CSS
.wrapper {
    display: grid;
    grid-template-columns: 1fr 1fr 1fr;
    grid-template-rows: 100px 100px;
}

// HTML
<div class="wrapper">
    <div>One</div>
    <div>Two</div>
    <div>Three</div>
    <div>Four</div>
    <div>Five</div>
    <div>Six</div>
    <div>Seven</div>
    <div>Eight</div>
</div>
```



Obrázek 7.4: Výsledek příkladu mřížkového rozložení

Gramatika

Gramatika validních hodnot vlastností `grid-auto-columns` a `grid-auto-rows` je identická a je definována takto (3):

```
<track-size>+

<track-size> = <track-breadth> | minmax( <inflexible-breadth> , <track-breadth> )
               | fit-content( [ <length> | <percentage> ] )

<track-breadth> = <flex> | <length> | <percentage>
               | min-content | max-content | auto
<inflexible-breadth> = <length> | <percentage>
               | min-content | max-content | auto
```

Vlastnosti lze definovat seznamem prvků, které mohou být vyjádřeny délkou, procentem nebo funkcemi `minmax()` či `fit-content()`, zajišťujícími inteligentní vyjádření výšky či šířky na základě jejich parametrů a obsahu buněk.

Testy

Testy pro vlastnosti `grid-auto-columns` a `grid-auto-rows` jsou umístěny ve třídě `GridLayoutTest`, zodpovědné za správnou funkci všech vlastností mřížkového rozložení. Díky využití nástrojů `TestUtils` otestování celkového rozsahu funkcionality zabralo jen 18 řádků.

Testuje se správná reprezentace klíčových slov (`auto`, `min-content` a `max-content`) i funkcí s charakteristickými kombinacemi parametrů.

Implementace

Pro reprezentaci vlastností `grid-auto-columns` a `grid-auto-rows` jsem vytvořil enumeraci `GridAutoRowsColumns` implementující rozhraní `CSSProperty`. Kromě 3 globálních hodnot (7.1) jsou deklarovány také konstanty pro klíčová slova `AUTO("auto")`, `MIN_CONTENT("min-content")` a `MAX_CONTENT("max-content")` a proměnná `list_values("")` pro všechny ostatní validní kombinace termů.

Validace je zajištěna metodou `processGridAutoRowsOrColumns()` ve třídě `DeclarationTransformer` volanou z obou procesních metod pro sloupce i řádky. Metoda nejprve zavolá generický test na zpracování jedné konstanty od zadané `CSSProperty`. Při selhání se dále volá generický test jedné kladné hodnoty délky či procenta.

V případě, že oba selžou se cyklicky testují a přiřazují termy do předem připraveného seznamu. Propouštěny jsou termy reprezentující identifikátor, jakožto potenciální klíčové slovo, délku, procenta a nebo podporované funkce.

V případě chyby ve formě nalezení neočekávaného, či nenalezení očekávaného termu je proces ukončen s návratovou hodnotou `false`. Po úspěšném dokončení cyklu je do mapy vlastností vložena hodnota `GridAutoRowsColumns.list_values` a do mapy hodnot seznam obsahující termy deklarace. Metoda vrací `true`.

7.4.3 Specifikace oblastí

Pomocí vlastnosti `grid-template-areas` je možné vyjádřit a pojmenovat požadované rozložení oblastí, které je poté možné využít k pozicování položek. Oblasti se zapisují jako série řetězců (reprezentující řádky) obsahující názvy oblastí oddělené mezerami (reprezentující sloupce). Pro zachování validity je nutné, aby oblasti měly unikátní pojmenování a jejich hranice vždy tvořily obdélníky.

```
// Příklad validní definice oblastí
grid-template-areas: "a a~a"
                     "b c c"
                     "b c c";

// Příklad chyby s opakujícím identifikátorem
grid-template-areas: "a a a"
                     "b b b"
                     "a a a";


// Příklad chyby s neobdélníkovou oblastí
grid-template-areas: "a a a"
                     "a c c"
                     "a c c";
```

Gramatika

Gramatika validních hodnot vlastnosti `grid-template-areas` je definována takto (3):

```
none | <string>+
```

Jak je vidět, gramatika je vcelku jednoduchá a skládá se pouze z klíčového slova `none` nebo série řetězců. Na řetězce definující oblasti jsou ovšem kladeny omezení popsané výše (7.4.3).

Testy

Testy pro vlastnost `grid-template-areas` jsou umístěny ve třídě `GridLayoutTest`, zodpovědné za správnou funkci všech vlastností mřížkového rozložení. Díky využití nástrojů `TestUtils` otestování celkového rozsahu funkcionality zabralo jen asi 20 řádků.

Testuje se správná reprezentace klíčového slova `none` a řetězců pro oblasti. Dále je kladen důraz na přesnou detekci validity specifikace oblastí.

Implementace

Pro reprezentaci vlastností `grid-template-areas` jsem vytvořil enumeraci `GridTemplateAreas` implementující rozhraní `CSSProperty`. Kromě 3 globálních hodnot (7.1) je deklarována také konstanta pro klíčové slovo `NONE("none")`, a proměnná `list_values("")` pro všechny ostatní validní kombinace termů.

Validace je zajištěna přímo procesní metodou `processGridTemplateAreas()` ve třídě `DeclarationTransformer`. Metoda nejprve zavolá generický test na zpracování jedné konstanty od zadané `CSSProperty`.

V případě selhání se cyklicky testují a přiřazují termy do předem připraveného seznamu. Propouštěny jsou pouze termy reprezentující řetězec, jakožto potenciální řádek oblasti. Tyto řetězce se průběžně parsují pomocnou metodou `ValidationUtils.getAreas()` a získané identifikátory se vkládají do dvourozměrného pole.

Po skončení cyklu a dotvoření reprezentace oblastí ve formě dvourozměrného pole řetězců je toto předáno druhé pomocné metodě `ValidationUtils.containsRectangles()`, která zkontroluje unikátnost identifikátorů a obdélníkový tvar oblastí.

V případě chyby ve formě nalezení neočekávaného, či nenalezení očekávaného termu je proces ukončen s návratovou hodnotou `false`. Po úspěšném dokončení je do mapy vlastností vložena hodnota `GridTemplateAreas.list_values` a do mapy hodnot seznam obsahující termy deklarace. Metoda vrací `true`.

7.4.4 Pozicování položek

Jedním ze způsobu specifikace rozložení položek je pozicování na hranice. Pomocí `grid-row-start`, `grid-row-end`, `grid-column-start` a `grid-column-end` lze specifikovat hranice oblasti, do které bude položka umístěna.

Přiřadit lze číslo reprezentující pořadí hranice od levého horního okraje mřížky, případně od protilehlé hranice, při přidání klíčového slova `span`, nebo identifikátor oblasti (7.4.3). Při zadání záporného čísla pořadí hranic počítá z druhé strany, nula není povolena.

Gramatika

Gramatika validních hodnot vlastností `grid-row-start`, `grid-row-end`, `grid-column-start` a `grid-column-end` je identická a je definována takto (3):

```
<grid-line>

<grid-line> = auto | <custom-ident> | [ <integer> && <custom-ident>? ]
              | [ span && [ <integer> || <custom-ident> ] ]
```

Každou z vlastností lze definovat klíčovým slovem `auto`, identifikátorem oblasti, numerickým vyjádřením pořadí hranice, potenciálně doplněným o identifikátor oblasti, nebo klíčovým slovem `span` s číslem hranice či identifikátorem oblasti.

Testy

Testy pro vlastnosti `grid-row-start`, `grid-row-end`, `grid-column-start` a `grid-column-end` jsou umístěny ve třídě `GridLayoutTest`, zodpovědné za správnou funkci všech vlastností mřížkového rozložení. Díky využití nástrojů `TestUtils` otestování celkového rozsahu funkcionality zabralo jen 21 řádků.

Testuje se správná reprezentace klíčových slov (`auto`, `span`) a dalších validních hodnot, dále propagace identifikátorů oblastí mezi vlastnostmi i očekávané selhání při krajních případech nevalidní deklarace.

Implementace

Pro reprezentaci vlastností `grid-row-start`, `grid-row-end`, `grid-column-start` a `grid-column-end` jsem vytvořil enumeraci `GridStartEnd` implementující rozhraní `CSSProperty`. Kromě 3 globálních hodnot (7.1) jsou deklarovány také konstanty pro klíčová

slova `AUTO("auto")` a `SPAN("span")` a proměnné `number("")` pro jedinou hodnotu pořadí hranice, `identificator("")` pro jediný identifikátor oblasti a `component_values("")` pro všechny ostatní validní kombinace termů.

Validace je zajištěna metodou `processGridStartEnd()` ve třídě `DeclarationTransformer` volanou ze všech 4 procesních metod. Metoda nejprve zavolá generický test na zpracování jedné konstanty od zadané `CSSProperty` či nenulové celočíselné hodnoty.

V případě selhání se cyklicky testují a přiřazují termy do předem připraveného seznamu. Propouštěny jsou termy reprezentující identifikátor, jakožto potenciální klíčové slovo nebo oblast, a celočíselnou hodnotu, pro pořadí hranice. Po nalezení termu dané kategorie je zapamatováno jeho místo ve formě indexu v deklaraci. Komparací těchto indexů je zajištěno správné pořadí a výskyt. Protože je nemožné kombinovat `SPAN("span")` se zápornými čísly, je hodnota hranice uložena pro případ, že toto klíčové slovo přijde až za ní.

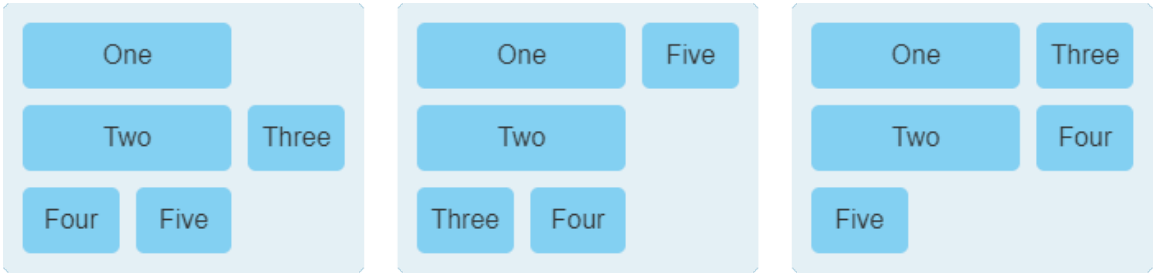
V případě chyby ve formě nalezení neočekávaného, či nenalezení očekávaného termu je proces ukončen s návratovou hodnotou `false`. Po úspěšném dokončení cyklu je do mapy vlastností vložena hodnota `GridStartEnd.component_values` a do mapy hodnot seznam obsahující termy deklarace. Metoda vrací `true`.

Kontrola překrývajícího obsahu

Do stejné oblasti v mřížce může být umístěno více položek a ty se tak mohou překrývat. Pořadí překrytých položek je možné ovlivnit vlastností `z-index` [26].

7.4.5 Implicitní rozložení

Ve výchozím stavu se položky do mřížky implicitně umisťují po řádcích. V případě, že je následující položka širší, než zbývající místo, je vložena na další řádek. Toto chování lze změnit nastavováním hodnoty `grid-auto-flow`. K dispozici je umisťování po řádcích nebo sloupcích a přidání hodnoty `dense` způsobí preferenci využití volného místa před zachováním pořadí položek.



Obrázek 7.5: Ukázka implicitního pozicování po řádcích, po sloupcích a hustě po řádcích

Gramatika

Gramatika validních hodnot vlastnosti `grid-auto-flow` je definována takto (3):

```
[ row | column ] || dense
```

Jak je vidět, gramatika je vcelku jednoduchá a skládá se pouze z klíčového slova `row` nebo `column`, potenciálně doplněná o `dense`.

Testy

Testy pro vlastnost `grid-auto-flow` jsou umístěny ve třídě `GridLayoutTest`, zodpovědné za správnou funkci všech vlastností mřížkového rozložení. Díky využití nástrojů `TestUtils` otestování celkového rozsahu funkcionality zabralo jen 5 řádků.

Testuje se správná reprezentace klíčových slov (`row`, `column`) samotných i v kombinaci s `dense` před nebo po.

Implementace

Pro reprezentaci vlastnosti `grid-auto-flow` jsem vytvořil enumeraci `GridAutoFlow` implementující rozhraní `CSSProperty`. Kromě 3 globálních hodnot (7.1) jsou deklarovány také konstanty pro klíčová slova `ROW("row")`, `COLUMN("column")` a `DENSE("dense")` a proměnná `list_values("")` pro všechny ostatní validní kombinace termů.

Validace je zajištěna přímo procesní metodou `processGridAutoFlow()` ve třídě `DeclarationTransformer`. Metoda nejprve zavolá generický test na zpracování jedné konstanty od zadané `CSSProperty`.

V případě selhání se cyklicky testují a přiřazují termy do předem připraveného seznamu. Propouštěny jsou pouze termy reprezentující identifikátor, jakožto potenciální klíčové slovo. Dvě boolovské proměnné zamezují vícenásobné použití `row` nebo `column` a příznaku `dense`.

V případě chyby ve formě nalezení neočekávaného, či nenalezení očekávaného termu je proces ukončen s návratovou hodnotou `false`. Po úspěšném dokončení je do mapy vlastností vložena hodnota `GridAutoFlow.component_values` a do mapy hodnot seznam obsahující termy deklarace. Metoda vrací `true`.

7.4.6 Mezery mezi položkami

Šířku hranic, neboli mezery mezi položkami, lze nastavit vlastnostmi `grid-row-gap`, pro mezery mezi řádky, a `grid-column-gap`, pro sloupce.

Gramatika

Gramatika validních hodnot vlastností `grid-row-gap` a `grid-column-gap` je identická a je definována takto (3):

```
normal | <length> | <percentage>
```

Jak je vidět, gramatika je vcelku jednoduchá a skládá se pouze z klíčového slova `normal`, délkové nebo procentuální hodnoty. Záporné hodnoty šířky nejsou dovoleny.

Testy

Testy pro vlastnosti `grid-row-gap` a `grid-column-gap` jsou umístěny ve třídě `GridLayoutTest`, zodpovědné za správnou funkci všech vlastností mřížkového rozložení. Díky využití nástrojů `TestUtils` otestování celkového rozsahu funkcionality zabralo jen 11 řádků.

Testuje se správná reprezentace klíčového slova `normal` a délkové nebo procentuální hodnoty.

Implementace

Pro reprezentaci vlastností `grid-row-gap` a `grid-column-gap` jsem vytvořil enumeraci `GridGap` implementující rozhraní `CSSProperty`. Kromě 3 globálních hodnot (7.1) je deklarována také konstanta pro klíčové slovo `NORMAL("normal")` a dále proměnné `length("")` pro jedinou délkovou hodnotu a `list_values("")` pro všechny ostatní validní kombinace termů.

Validace je zajištěna generickým testem `genericOneIdentOrLengthOrPercent()` na zpracování jedné konstanty od zadané `CSSProperty`, délkové nebo procentuální hodnoty volaným z obou procesních metod ve třídě `DeclarationTransformer`. Proces končí vrácením návratové hodnoty z tohoto testu.

7.4.7 Zkrácený zápis

Ke specifikaci většiny výše uvedených CSS vlastností existuje alternativní zkrácený zápis. Tyto tzv. *zkratové* vlastnosti slouží k nastavení hodnot do několika stylů najednou v jedné deklaraci. Jejich využití může přispět ke stručnosti a čitelnosti kódu. Mřížkové rozložení podporuje následující zkratkové vlastnosti:

`grid-template`

- Současně nastaví explicitní oblasti i rozměry hranic sloupců a řádků.
- Přepíše hodnoty `grid-template-areas`, `grid-template-columns` a `grid-template-rows`.
- Gramatika:

```
none | [ <'grid-template-rows'> / <'grid-template-columns'> ]
| [ <line-names>? <string> <track-size>? <line-names>? ]+
[ / <explicit-track-list> ]?

<line-names> = '[' <custom-ident>* ']'
<track-size> = <track-breadth>
               | minmax( <inflexible-breadth> , <track-breadth> )
               | fit-content( [ <length> | <percentage> ] )
<explicit-track-list> = [ <line-names>? <track-size> ]+ <line-names>?

<track-breadth> = <flex> | <length> | <percentage>
               | min-content | max-content | auto
<inflexible-breadth> = <length> | <percentage>
               | min-content | max-content | auto
```

`grid-row`

- Současně nastaví počáteční a konečnou hranici řádku položky.
- Přepíše hodnoty `grid-row-start` a `grid-row-end`.
- Gramatika:

```
<grid-line> [ / <grid-line> ]?
<grid-line> = auto | <custom-ident> | [ <integer> && <custom-ident>? ]
               | [ span && [ <integer> || <custom-ident> ] ]
```

`grid-column`

- Současně nastaví počáteční a konečnou hranici sloupce položky.
- Přepíše hodnoty `grid-column-start` a `grid-column-end`.
- Gramatika:

```
<grid-line> [ / <grid-line> ]?
<grid-line> = auto | <custom-ident> | [ <integer> && <custom-ident>? ]
              | [ span && [ <integer> || <custom-ident> ] ]
```

`grid-area`

- Současně nastaví počáteční a konečnou hranice oblasti položky.
- Přepíše hodnoty `grid-row-start`, `grid-row-end`, `grid-column-start` a `grid-column-end`.
- Gramatika:

```
<grid-line> [ / <grid-line> ]{0,3}
<grid-line> = auto | <custom-ident> | [ <integer> && <custom-ident>? ]
              | [ span && [ <integer> || <custom-ident> ] ]
```

`grid`

- Současně nastaví všechny explicitní i implicitní vlastnosti mřížkového rozložení.
- Přepíše hodnoty `grid-template-areas`, `grid-template-columns`, `grid-template-rows`, `grid-auto-rows`, `grid-auto-columns` a `grid-auto-flow`.
- Gramatika:

```
<'grid-template'>
| <'grid-template-rows'> / [ auto-flow && dense? ] <'grid-auto-columns'>?
| [ auto-flow && dense? ] <'grid-auto-rows'>? / <'grid-template-columns'>
```

S výjimkou vlastnosti `grid`, která ve své specifikaci zavedla nové klíčové slovo `auto-flow`, nebylo potřeba vytvářet nové implementace `CSSProperty`. Pro každou z nich ale byla přidána procesní metoda ve třídě `DeclarationTransformer` zajišťující validaci jejich deklarací. Všechny tyto procesní metody jsou vesměs krátké, protože vstupní hodnoty pouze parsují, vkládají do subdeklarací a poté předávají procesním metodám již konkrétních parametrů.

Testy obsažené v `GridLayoutTest` samozřejmě testují funkcionalitu i těchto zkratkových vlastností.

7.5 Třetí fáze – Přechody a animace

Ve třetí a poslední fázi implementace byly do projektu přidány vlastností týkající se CSS přechodů a animací. Ve výchozím stavu se změna hodnoty vlastnosti ihned propaguje a zobrazí. Velká část CSS stylů ovšem umožňuje plynulý přechod mezi jednotlivými hodnotami. Tohoto lze využít při definici přechodů a animací definujících, jakým způsobem jsou hodnoty v jednotlivých časech dopočítány.

7.5.1 Přechod

Přechod (`transition`) je jednodušší ze dvou způsobů definic určování hodnot vlastnosti v čase. Přechod je aplikován v momentě, kdy nějaká událost způsobí změnu hodnoty ovlivněného stylu. Definovatelnými parametry jsou:

- ovlivněná CSS vlastnost
- délka přechodu
- zpoždění startu
- a časová funkce určující průběh.

Elementu je možné přiřadit i několik přechodů s různými parametry, přičemž všechny budou prováděny paralelně. Parametry je možné definovat zvlášť nebo použitím tzv. zkratkové vlastnosti `transition` najednou v jedné deklaraci.

Gramatika

Gramatika validních hodnot zkratové vlastnosti `transition` je definována takto (3):

```
<single-transition>#

<single-transition> = [ none | <single-transition-property> ]
                  || <time> || <single-transition-timing-function> || <time>

<single-transition-property> = all | <custom-ident>
<single-transition-timing-function> = <single-timing-function>

<single-timing-function> = linear | <cubic-bezier-timing-function>
                  | <step-timing-function> | <frames-timing-function>

<cubic-bezier-timing-function> = ease | ease-in | ease-out | ease-in-out
                  | cubic-bezier(<number>, <number>, <number>, <number>)
<step-timing-function> = step-start | step-end
                  | steps(<integer>[, [ start | end ] ]?)
<frames-timing-function> = frames(<integer>)
```

Jak je vidět, gramatika se v podstatě rozpadá na gramatiky vlastností jednotlivých parametrů, s tím, že je možné je specifikovat v jakékoliv pořadí či úplně vynechat.

Testy

Testy pro zkratkovou vlastnost `transition` jsou umístěny ve třídě `TransitionTest`, zodpovědné za správnou funkci všech vlastností CSS přechodů. Díky využití nástrojů `TestUtils` otestování celkového rozsahu funkcionality zabralo jen 52 řádků. Testuje se správná reprezentace a propagace hodnot vlastnostem jednotlivých parametrů.

Implementace

Validace je zajištěna generickou procesní metodou `processPropertiesInList()` ve třídě `DeclarationTransformer`. Metoda kromě deklarace a map pro `CSSProperty` a hodnoty termů dostane na vstup seznam potenciálních vlastností, které se mohou v deklaraci vyskytnout. Vlastnosti jsou seřazeny podle priority takovým způsobem, že jednotlivé termy jsou přiřazeny první z nich, která jej přijme. Již nastavené parametry jsou přeskakovány dokud není nalezen čárkový operátor `","` indikující specifikaci dalšího přechodu.

V případě, že term nepřijme žádná z vlastností v seznamu je proces ukončen s návratovou hodnotou `false`. Jinak je vložen do subdeklarace pro danou vlastnost.

Po úspěšném zpracování a rozřazení všech termů do subdeklarací jsou tyto, pokud neprázdné, znovu předány do procesních metod vlastností, tentokrát ovšem všechny najednou.

V případě selhání je proces ukončen s návratovou hodnotou `false`. Jinak je pouze vráceno `true`. O přiřazení hodnot do jednotlivých vlastností parametrů se postaraly již jejich procesní metody.

7.5.2 Přechod – Ovlivněná vlastnost

Ve výchozím stavu jsou ovlivněny všechny podporující vlastnosti. Toto lze změnit nastavením vlastnosti `transition-property`. Možnými hodnotami jsou žádná, všechny či specifická CSS vlastnost identifikovaná jejím jménem.

Gramatika

Gramatika validních hodnot vlastnosti `transition-property` je definována takto (3):

```
none | [ all | <custom-ident> ]#
```

Je tedy možné ji definovat klíčovým slovem `none` či seznamem ovlivněných vlastností jednotlivými efekty přechodu.

Testy

Testy pro vlastnost `transition-property` jsou umístěny ve třídě `TransitionTest`, zodpovědné za správnou funkci všech vlastností CSS přechodů. Díky využití nástrojů `TestUtils` otestování celkového rozsahu funkcionality zabralo jen 9 řádků. Testuje se správná reprezentace jak klíčových slov (`none` a `all`), tak i jmen vlastností.

Implementace

Pro reprezentaci vlastnosti `transition-property` jsem vytvořil enumeraci `TransitionProperty` implementující rozhraní `CSSProperty`. Kromě 3 globálních hodnot (7.1) jsou deklarovány také konstanty pro klíčová slova `NONE("none")` a `ALL("all")` a proměnné

`custom_ident("")` pro jediný identifikátor vlastnosti a `list_values("")` pro všechny ostatní validní kombinace termů.

Validace je zajištěna přímo procesní metodou `processTransitionProperty()` ve třídě `DeclarationTransformer`. Metoda nejprve zavolá generický test na zpracování jedné konstanty od zadané `CSSProperty`.

V případě selhání se cyklicky testují a přiřazují termy do předem připraveného seznamu. Propouštěny jsou pouze termy reprezentující identifikátor, jakožto potenciální klíčové slovo či jméno vlastnosti.

V případě chyby ve formě nalezení neočekávaného, či nenalezení očekávaného termu je proces ukončen s návratovou hodnotou `false`. Po úspěšném dokončení je do mapy vlastností vložena hodnota `TransitionProperty.custom_ident`, pokud byl nalezen jediný identifikátor, nebo `TransitionProperty.list_values`, pokud termů bylo více. Stejně tak do mapy hodnot je vložen buď samotný term identifikátoru, nebo celý seznam. Metoda vrací `true`.

7.5.3 Přechod – Délka

Ve výchozím stavu je délka přechodu nastavena na 0 vteřin, tedy změna proběhne instantně. Pomocí vlastnosti `transition-duration` můžeme délku nastavit na jakoukoliv kladnou hodnotu času.

Gramatika

Gramatika validních hodnot vlastnosti `transition-duration` je definována takto (3):

```
<time>#
```

Je tedy možné ji definovat jednou či více hodnotami času v seznamu odděleném čárkami.

Testy

Testy pro vlastnost `transition-duration` jsou umístěny ve třídě `TransitionTest`, zodpovědné za správnou funkci všech vlastností CSS přechodů. Díky využití nástrojů `TestUtils` otestování celkového rozsahu funkcionality zabralo jen 9 řádků. Testuje se správná reprezentace jedné i více časových hodnot.

Implementace

Pro reprezentaci vlastnosti `transition-duration` jsem vytvořil enumeraci `TransitionDuration` implementující rozhraní `CSSProperty`. Kromě 3 globálních hodnot (7.1) jsou deklarovány také proměnné `time("")` pro jedinou časovou hodnotu a `list_values("")` pro všechny ostatní validní kombinace termů.

Validace je zajištěna přímo procesní metodou `processTransitionDuration()` ve třídě `DeclarationTransformer`. Metoda nejprve zavolá generický test na zpracování jedné kladné časové hodnoty.

V případě selhání se cyklicky testují a přiřazují termy do předem připraveného seznamu. Propouštěny jsou pouze termy reprezentující kladné hodnoty času.

V případě chyby ve formě nalezení neočekávaného, či nenalezení očekávaného termu je proces ukončen s návratovou hodnotou `false`. Po úspěšném dokončení je do mapy vlastností vložena hodnota `TransitionDuration.time`, pokud byl nalezen jediný čas, nebo

`TransitionDuration.list_values`, pokud jich bylo více. Stejně tak do mapy hodnot je vložen buď samotný term, nebo celý seznam. Metoda vrací `true`.

7.5.4 Přechod – Zpoždění startu

Ve výchozím stavu je zpoždění přechodu nastaveno na 0 vteřin, tedy změna proběhne ihned bez čekání. Pomocí vlastnosti `transition-delay` můžeme zpoždění nastavit na jakoukoliv kladnou hodnotu času.

Gramatika

Gramatika validních hodnot vlastnosti `transition-delay` je definována takto (3):

```
<time>#
```

Je tedy možné ji definovat jednou či více hodnotami času v seznamu odděleném čárkami.

Testy

Testy pro zkratkovou vlastnost `transition-delay` jsou umístěny ve třídě `TransitionTest`, zodpovědné za správnou funkci všech vlastností CSS přechodů. Díky využití nástrojů `TestUtils` otestování celkového rozsahu funkcionality zabralo jen 9 řádků. Testuje se správná reprezentace jedné i více časových hodnot.

Implementace

Pro reprezentaci vlastnosti `transition-delay` jsem vytvořil enumeraci `TransitionDelay` implementující rozhraní `CSSProperty`. Kromě 3 globálních hodnot (7.1) jsou deklarovány také proměnné `time("")` pro jedinou časovou hodnotu a `list_values("")` pro všechny ostatní validní kombinace termů.

Validace je zajištěna přímo procesní metodou `processTransitionDelay()` ve třídě `DeclarationTransformer`. Metoda nejprve zavolá generický test na zpracování jedné kladné časové hodnoty.

V případě selhání se cyklicky testují a přiřazují termy do předem připraveného seznamu. Propouštěny jsou pouze termy reprezentující kladné hodnoty času.

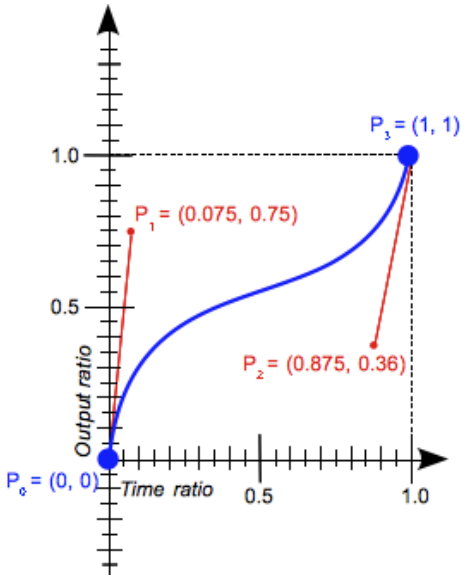
V případě chyby ve formě nalezení neočekávaného, či nenalezení očekávaného termu je proces ukončen s návratovou hodnotou `false`. Po úspěšném dokončení je do mapy vlastností vložena hodnota `TransitionDelay.time`, pokud byl nalezen jediný čas, nebo `TransitionDelay.list_values`, pokud jich bylo více. Stejně tak do mapy hodnot je vložen buď samotný term, nebo celý seznam. Metoda vrací `true`.

7.5.5 Přechod – Časová funkce

Časová funkce přechodu se nastavuje vlastností `transition-timing-function`. Ve výchozím stavu určuje časový průběh přechodu tzv. `ease` funkce, což je pojmenovaná kubická Beziérova křivka s hodnotami `cubic-bezier(0.25, 0.1, 0.25, 1.0)`.

`cubic-bezier(P1X, P1Y, P2X, P2Y)` je funkcionální notace kubické Beziérovy křivky. Jedná se o kontinuální křivku definovanou čtyřmi body P0, P1, P2 a P3. Nultý a třetí bod je dán fixně jako P0 = (0, 0) a P3 = (1, 1) a je tedy třeba doplnit pouze první a druhý.

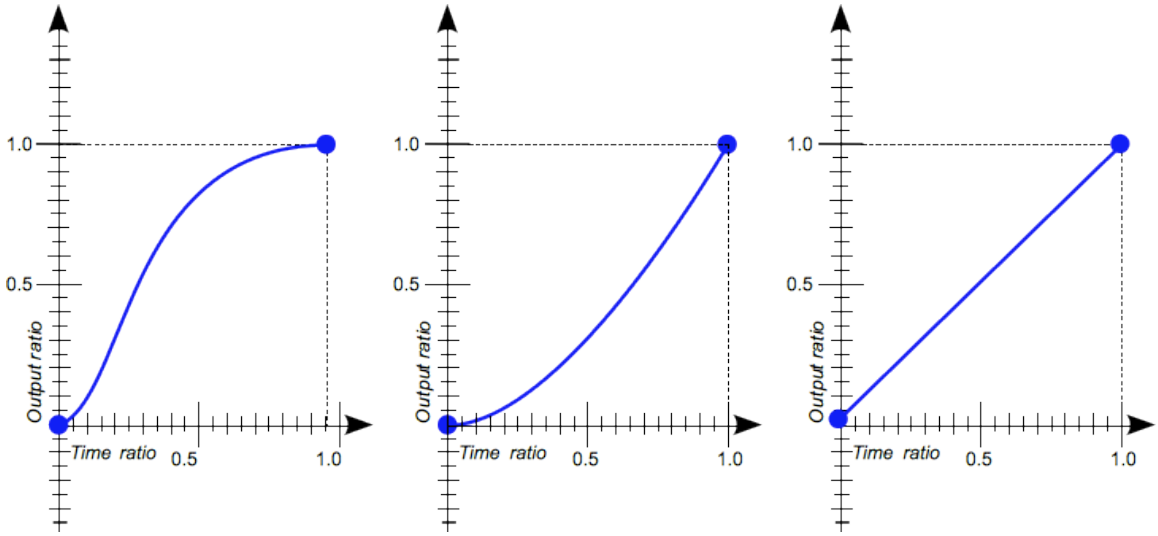
Ne všechny kubické Beziérovy křivky jsou vhodné pro vyjádření časové funkce. Je potřeba, aby pro každý čas definovala maximálně jeden bod výstupu [27].



Obrázek 7.6: Ukázka kubické Beziérovy křivky [18]

V konečném důsledku tedy *ease* funkce definuje pomalý start a zakončení s rychlým průběhem uprostřed. Dalšími pojmenovanými funkcemi jsou:

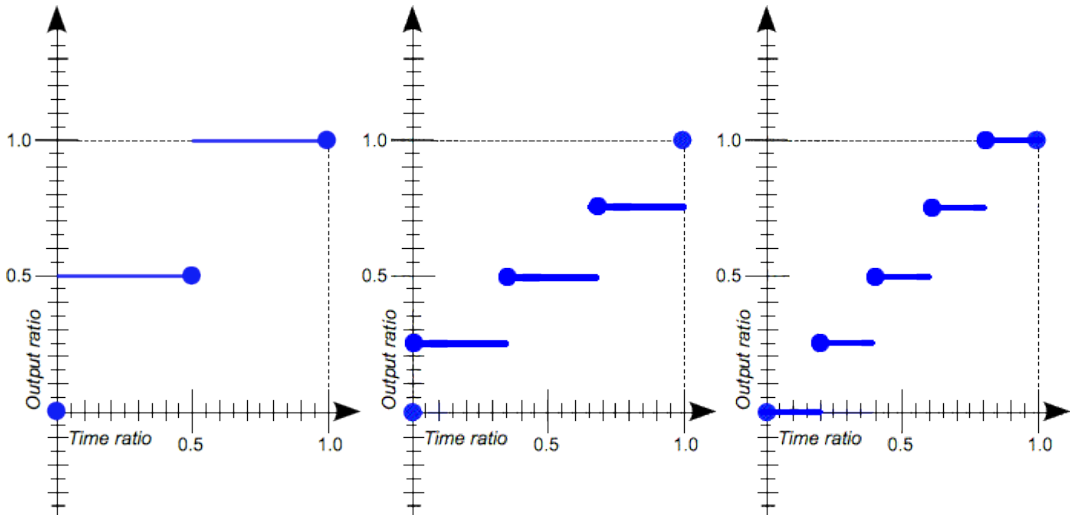
- `ease-in` – ekvivalentní `cubic-bezier(0.42, 0.0, 1.0, 1.0)`
- `ease-out` – ekvivalentní `cubic-bezier(0.0, 0.0, 0.58, 1.0)`
- `ease-in-out` – ekvivalentní `cubic-bezier(0.42, 0.0, 0.58, 1.0)`
- `linear` – ekvivalentní `cubic-bezier(0.0, 0.0, 1.0, 1.0)`



Obrázek 7.7: Průběh kubické Beziérovy křivky pro `ease` [22], `ease-in` [23] a `linear` [24]

Dalším způsobem definice časového průběhu je tzv. kroková funkce, která místo plynulého průběhu přechod zubatě odkrokuje definovaným počtem kroků, přičemž každému kroku věnuje stejné množství času. Funkce má 2 parametry `steps(n, direction)`:

- počet kroků
- a směr kontinuity funkce s možnostmi
  - `jump-start` – levá kontinuita se startem v prvním (nikoli nultém) kroku
  - `jump-start` – pravá kontinuita s koncem v předposledním kroku
  - `jump-none` – nic se nepřeskakuje, začátek v 0% a konec v 100%
  - `jump-both` – přeskočení nultého a posledního kroku
  - `start` – ekvivalentní `jump-start`
  - `end` – ekvivalentní `jump-end`



Obrázek 7.8: Průběh krokové funkce pro paramerty `(2, start)` [17], `(3, jump-both)` [15] a `(5, jump-none)` [16]

K dispozici jsou také 2 pojmenované kombinace této krokové funkce:

- `steps-start` - ekvivalentní `steps(1, jump-start)`, přeskok do konečného stavu ihned na začátku přechodu
- `steps-end` - ekvivalentní `steps(1, jump-end)`, zůstání v počátečním stavu po celou dobu přechodu

Posledním, dnes již v podstatě opuštěným, způsobem definice časového průběhu je snímková funkce `frames(n)` s jediným parametrem počtu snímků.

Gramatika

Gramatika validních hodnot vlastnosti `transition-timing-function` je definována takto (3):

```
<single-timing-function>#

<single-timing-function> = linear | <cubic-bezier-timing-function>
                         | <step-timing-function> | <frames-timing-function>

<cubic-bezier-timing-function> = ease | ease-in | ease-out | ease-in-out
                         | cubic-bezier(<number>, <number>, <number>, <number>)
<step-timing-function> = step-start | step-end | steps(<integer>[, start | end ]?)
<frames-timing-function> = frames(<integer>)
```

Je tedy možné ji definovat jednou nebo více časovými funkcemi ve seznamu odděleném čarkami.

Testy

Testy pro zkratkovou vlastnost `transition-timing-function` jsou umístěny ve třídě `TransitionTest`, zodpovědné za správnou funkci všech vlastností CSS přechodů. Díky využití nástrojů `TestUtils` otestování celkového rozsahu funkcionality zabralo jen 30 řádků.

Testuje se správná reprezentace všech časových funkcí ve tvaru klíčových slov i v doslovné formě s typickými validními kombinacemi parametrů. Dále také invalidace celé deklarace při detekci špatných parametrů funkce.

Implementace

Pro reprezentaci vlastnosti `transition-timing-function` jsem vytvořil enumeraci `TransitionTimingFunction` implementující rozhraní `CSSProperty`. Kromě 3 globálních hodnot (7.1) jsou deklarovány také konstanty pro klíčová slova `LINEAR("linear")`, `EASE_IN("ease-in")`, `EASE_OUT("ease-out")`, `EASE_IN_OUT("ease-in-out")`, `EASE("ease")`, `STEP_START("step-start")` a `STEP_END("step-end")` a proměnné `timing_function("")` pro jedinou časovou funkci a `list_values("")` pro všechny ostatní validní kombinace termů.

Pro reprezentaci časových funkcí jsem přidal rozhraní `TimingFunction`, rozšiřující `TermFunction`, představující obecnou časovou funkci a také 3 konkrétní (`CubicBezier`, `Steps` a `Frames`), rozšiřující toto nové rozhraní. Validaci parametrů zajišťují implementace těchto 3 rozhraní dědící od implementace obecné funkce `TermFunctionImpl`. Těmto jsou metodou `setValue(List<Term> value)` předány termy obsažené v jejich závorkách a ony pak rozhodnou, zda jsou parametry validní a jejich hodnoty si uloží. Validaci parametrů funkce lze zkontrolovat zavoláním `isValid()` deklarovaným v `TermFunction`, což se provádí např. při konverzi funkce z textové podoby.

Validaci vlastnosti zajišťuje procesní metoda `processTransitionTimingFunction()` ve třídě `DeclarationTransformer`. Metoda nejprve zavolá generický test na zpracování jedné konstanty od zadané `CSSProperty`.

V případě selhání se cyklicky testují a přiřazují termy do předem připraveného seznamu. Propouštěny jsou termy reprezentující identifikátor, jakožto potenciální klíčové slovo, nebo časovou funkci.

V případě chyby ve formě nalezení neočekávaného, či nenalezení očekávaného termu je proces ukončen s návratovou hodnotou `false`. Po úspěšném dokončení je do mapy vlastností vložena hodnota `TransitionTimingFunction.timing_function`, pokud byla nalezena jediná funkce, nebo `TransitionTimingFunction.list_values`, pokud jich bylo více. Stejně tak do mapy hodnot je vložen buď samotný term funkce, nebo celý seznam. Metoda vrací `true`.

7.5.6 Animace

Animace (`animation`) je druhý, komplexnější způsob definice určování hodnot vlastnosti v čase. S CSS animacemi také souvisí tzv. `@keyframes` pravidla, s jejichž pomocí lze místo jednoduchého přechodu přesněji definovat chování ovlivněných vlastností v mezikrocích animace. `@keyframes` budou implementovány separátně a nejsou tedy součástí této práce [25].

Animace se aplikuje ihned po jejím načtení, tedy většinou po úplném načtení dokumentu, a pokud je v aktivovaném stavu, se taky ihned spustí. Definovatelnými parametry jsou:

- jméno pro identifikaci
- délka animace
- zpoždění startu
- počet opakování
- směr přehrávání
- stav přehrávání
- určení hodnoty stylu před a po animaci
- a časová funkce určující průběh.

Gramatika

Gramatika validních hodnot zkratové vlastnosti `animation` je definována takto (3):

```
<single-animation>#

<single-animation> = <time> || <single-timing-function> || <time>
|| <single-animation-iteration-count> || <single-animation-direction>
|| <single-animation-fill-mode> || <single-animation-play-state>
|| [ none | <keyframes-name> ]

<single-timing-function> = linear | <cubic-bezier-timing-function>
                         | <step-timing-function> | <frames-timing-function>
<single-animation-iteration-count> = infinite | <number>
<single-animation-direction> = normal | reverse | alternate | alternate-reverse
<single-animation-fill-mode> = none | forwards | backwards | both
<single-animation-play-state> = running | paused
<keyframes-name> = <custom-ident> | <string>
```

```
<cubic-bezier-timing-function> = ease | ease-in | ease-out | ease-in-out
                      | cubic-bezier(<number>, <number>, <number>, <number>)
<step-timing-function> = step-start | step-end
                      | steps(<integer>[, [ start | end ] ]?)
<frames-timing-function> = frames(<integer>)
```

Jak je vidět, gramatika se v podstatě rozpadá na gramatiky vlastností jednotlivých parametrů, s tím, že je možné je specifikovat v jakémkoliv pořadí či úplně vynechat.

Testy

Testy pro zkratkovou vlastnost `animation` jsou umístěny ve třídě `AnimationTest`, zodpovědné za správnou funkci všech vlastností CSS animací. Díky využití nástrojů `TestUtils` otestování celkového rozsahu funkcionality zabralo jen 130 řádků. Testuje se správná reprezentace a propagace hodnot vlastnostem jednotlivých parametrů.

Implementace

Validace je zajištěna generickou procesní metodou `processPropertiesInList()`, popsanou již v sekci implementace zkratkové vlastnosti pro definici přechodů (7.5.1).

7.5.7 Animace – Identifikátor

Nastavením vlastnosti `animation-name` můžeme elementu přiřadit jednu nebo více animací definovaných `@keyframes` pravidly (7.5.6).

Gramatika

Gramatika validních hodnot vlastnosti `animation-name` je definována takto (3):

```
[ none | <custom-ident> ]#
```

Je tedy možné ji definovat jedním nebo více identifikátory, případně místo jména animace použít klíčové slovo `none`. V originální definici je ještě možnost definice řetězcem v uvozovkách, ale pro doslova nulovou podporu ze strany běžných prohlížečů jsem se rozhodl tuto možnost neimplementovat.

Testy

Testy pro vlastnost `animation-name` jsou umístěny ve třídě `AnimationTest`, zodpovědné za správnou funkci všech vlastností CSS animací. Díky využití nástrojů `TestUtils` otestování celkového rozsahu funkcionality zabralo jen 9 řádků.

Testuje se správná reprezentace jednoho i více identifikátorů nebo klíčových slov. Dále také invalidace deklarace při detekci špatného formátu identifikátoru.

Implementace

Pro reprezentaci vlastnosti `animation-name` jsem vytvořil enumeraci `AnimationName` implementující rozhraní `CSSProperty`. Kromě 3 globálních hodnot (7.1) je deklarována také

konstanta pro klíčové slovo `NONE("none")` a proměnné `custom_ident("")` pro jediný identifikátor a `list_values("")` pro všechny ostatní validní kombinace termů.

Validace je zajištěna přímo procesní metodou `processAnimationName()` ve třídě `DeclarationTransformer`. Metoda nejprve zavolá generický test na zpracování jedné konstanty od zadané `CSSProperty`.

V případě selhání se cyklicky testují a přiřazují termy do předem připraveného seznamu. Propouštěny jsou pouze termy reprezentující validní identifikátor.

V případě chyby ve formě nalezení neočekávaného, či nenalezení očekávaného termu je proces ukončen s návratovou hodnotou `false`. Po úspěšném dokončení je do mapy vlastností vložena hodnota `AnimationName.custom_ident`, pokud byl nalezen jediný identifikátor, nebo `AnimationName.list_values`, pokud jich bylo více. Stejně tak do mapy hodnot je vložen buď samotný term, nebo celý seznam. Metoda vrací `true`.

7.5.8 Animace – Délka

Ve výchozím stavu je délka animace nastavena na 0 vteřin, tedy změna proběhne instantně. Pomocí vlastnosti `animation-duration` můžeme délku nastavit na jakoukoliv kladnou hodnotu času.

Gramatika

Gramatika validních hodnot vlastnosti `animation-duration` je definována takto (3):

```
<time>#
```

Je tedy možné ji definovat jednou či více hodnotami času v seznamu odděleném čárkami.

Testy

Testy pro zkratkovou vlastnost `animation-duration` jsou umístěny ve třídě `AnimationTest`, zodpovědné za správnou funkci všech vlastností CSS animací. Díky využití nástrojů `TestUtils` otestování celkového rozsahu funkcionality zabralo jen 9 řádků. Testuje se správná reprezentace jedné i více časových hodnot.

Implementace

Pro reprezentaci vlastnosti `animation-duration` jsem vytvořil enumeraci `AnimationDuration` implementující rozhraní `CSSProperty`. Kromě 3 globálních hodnot (7.1) jsou deklarovány také proměnné `time("")` pro jedinou časovou hodnotu a `list_values("")` pro všechny ostatní validní kombinace termů.

Validace je zajištěna přímo procesní metodou `processAnimationDuration()` ve třídě `DeclarationTransformer`. Metoda nejprve zavolá generický test na zpracování jedné kladné časové hodnoty.

V případě selhání se cyklicky testují a přiřazují termy do předem připraveného seznamu. Propouštěny jsou pouze termy reprezentující kladné hodnoty času.

V případě chyby ve formě nalezení neočekávaného, či nenalezení očekávaného termu je proces ukončen s návratovou hodnotou `false`. Po úspěšném dokončení je do mapy vlastností vložena hodnota `AnimationDuration.time`, pokud byl nalezen jediný čas, nebo `AnimationDuration.list_values`, pokud jich bylo více. Stejně tak do mapy hodnot je vložen buď samotný term, nebo celý seznam. Metoda vrací `true`.

7.5.9 Animace – Zpoždění startu

Ve výchozím stavu je zpoždění animace nastaveno na 0 vteřin, tedy spustí se ihned bez čekání. Pomocí vlastnosti `animation-delay` můžeme zpoždění nastavit na jakoukoliv kladnou hodnotu času.

Gramatika

Gramatika validních hodnot vlastnosti `animation-delay` je definována takto (3):

```
<time>#
```

Je tedy možné ji definovat jednou či více hodnotami času v seznamu odděleném čárkami.

Testy

Testy pro vlastnost `animation-delay` jsou umístěny ve třídě `AnimationTest`, zodpovědné za správnou funkci všech vlastností CSS animací. Díky využití nástrojů `TestUtils` otestování celkového rozsahu funkcionality zabralo jen 9 řádků. Testuje se správná reprezentace jedné i více časových hodnot.

Implementace

Pro reprezentaci vlastnosti `animation-delay` jsem vytvořil enumeraci `AnimationDelay` implementující rozhraní `CSSProperty`. Kromě 3 globálních hodnot (7.1) jsou deklarovány také proměnné `time("")` pro jedinou časovou hodnotu a `list_values("")` pro všechny ostatní validní kombinace termů.

Validace je zajištěna přímo procesní metodou `processAnimationDelay()` ve třídě `DeclarationTransformer`. Metoda nejprve zavolá generický test na zpracování jedné kladné časové hodnoty.

V případě selhání se cyklicky testují a přiřazují termy do předem připraveného seznamu. Propouštěny jsou pouze termy reprezentující kladné hodnoty času.

V případě chyby ve formě nalezení neočekávaného, či nenalezení očekávaného termu je proces ukončen s návratovou hodnotou `false`. Po úspěšném dokončení je do mapy vlastností vložena hodnota `AnimationDelay.time`, pokud byl nalezen jediný čas, nebo `AnimationDelay.list_values`, pokud jich bylo více. Stejně tak do mapy hodnot je vložen buď samotný term, nebo celý seznam. Metoda vrací `true`.

7.5.10 Animace – Počet opakování

Ve výchozím stavu je animace přehrána právě jednou od začátku do konce. Toto chování můžeme změnit nastavením vlastnosti `animation-iteration-count`. Počet opakování lze vyjádřit nekonečnem nebo jakýmkoliv kladným číslem, včetně desetinných, značících přehrání pouze části animace v její poslední iteraci.

Gramatika

Gramatika validních hodnot vlastnosti `animation-iteration-count` je definována takto (3):

```
[ infinite | <number> ]#
```

Je tedy možné ji definovat jedním či více kladnými čísly nebo klíčovými slovy `infinite` v seznamu odděleném čárkami.

Testy

Testy pro vlastnost `animation-iteration-count` jsou umístěny ve třídě `AnimationTest`, zodpovědné za správnou funkci všech vlastností CSS animací. Díky využití nástrojů `TestUtils` otestování celkového rozsahu funkcionality zabralo jen 9 řádků. Testuje se správná reprezentace klíčového slova `infinite` i číselných hodnot.

Implementace

Pro reprezentaci vlastnosti `animation-iteration-count` jsem vytvořil enumeraci `AnimationIterationCount` implementující rozhraní `CSSProperty`. Kromě 3 globálních hodnot (7.1) je deklarována také konstanta `INFINITE("infinite")` a proměnné `number("")` pro jedinou číselnou hodnotu a `list_values("")` pro všechny ostatní validní kombinace termů.

Validace je zajištěna přímo procesní metodou `processAnimationIterationCount()` ve třídě `DeclarationTransformer`. Metoda nejprve zavolá generický test na zpracování jedné konstanty od zadané `CSSProperty` či kladné číselné hodnoty.

V případě selhání se cyklicky testují a přiřazují termy do předem připraveného seznamu. Propouštěny jsou termy reprezentující identifikátor, jakožto potenciální klíčové slovo, nebo kladnou číselnou hodnotu.

V případě chyby ve formě nalezení neočekávaného, či nenalezení očekávaného termu je proces ukončen s návratovou hodnotou `false`. Po úspěšném dokončení je do mapy vlastností vložena hodnota `AnimationIterationCount.number`, pokud byl nalezen jediný počet, nebo `AnimationIterationCount.list_values`, pokud jich bylo více. Stejně tak do mapy hodnot je vložen buď samotný term, nebo celý seznam. Metoda vrací `true`.

7.5.11 Animace – Směr přehrávání

Ve výchozím stavu je animace přehrávána normálně od začátku do konce. Nastavením vlastnosti `animation-direction` je ale možné určit jiný směr přehrávání. Přehrávat je možné také pozpátku nebo střídavě dopředu a dozadu s výběrem počátečního směru.

Gramatika

Gramatika validních hodnot vlastnosti `animation-direction` je definována takto (3):

```
[ normal | reverse | alternate | alternate-reverse ]#
```

Je tedy možné ji definovat jedním či více klíčovými slovy pro výběr směru v seznamu odděleném čárkami.

Testy

Testy pro vlastnost `animation-direction` jsou umístěny ve třídě `AnimationTest`, zodpovědné za správnou funkci všech vlastností CSS animací. Díky využití nástrojů `TestUtils` otestování celkového rozsahu funkcionality zabralo jen 9 řádků. Testuje se správná reprezentace jednoho i více klíčových slov (`normal`, `reverse`, `alternate` a `alternate-reverse`).

Implementace

Pro reprezentaci vlastnosti `animation-direction` jsem vytvořil enumeraci `AnimationDirection` implementující rozhraní `CSSProperty`. Kromě 3 globálních hodnot (7.1) jsou deklarovány také konstanty pro klíčová slova (`ALTERNATE("alternate")`, `ALTERNATE_REVERSE("alternate-reverse")`, `NORMAL("normal")` a `REVERSE("reverse")`) a proměnná `list_values("")` pro všechny ostatní validní kombinace termů.

Validace je zajištěna přímo procesní metodou `processAnimationDirection()` ve třídě `DeclarationTransformer`. Metoda nejprve zavolá generický test na zpracování jedné konstanty od zadané `CSSProperty`.

V případě selhání se cyklicky testují a přiřazují termy do předem připraveného seznamu. Propouštěny jsou pouze termy reprezentující identifikátor, jakožto potenciální klíčové slovo.

V případě chyby ve formě nalezení neočekávaného, či nenalezení očekávaného termu je proces ukončen s návratovou hodnotou `false`. Po úspěšném dokončení cyklu je do mapy vlastností vložena hodnota `AnimationDirection.list_values` a do mapy hodnot seznam obsahující termy deklarace. Metoda vrací `true`.

7.5.12 Animace – Stav přehrávání

Ve výchozím stavu je animace spuštěna a přehraje se tedy ihned po načtení. Nastavením vlastnosti `animation-play-state` je možné animaci pozastavit a spustit později, např. při nějaké události.

Gramatika

Gramatika validních hodnot vlastnosti `animation-play-state` je definována takto (3):

```
[ running | paused ]#
```

Je tedy možné ji definovat jedním či více klíčovými slovy `running` a `paused` v seznamu odděleném čárkami.

Testy

Testy pro vlastnost `animation-play-state` jsou umístěny ve třídě `AnimationTest`, zodpovědné za správnou funkci všech vlastností CSS animací. Díky využití nástrojů `TestUtils` otestování celkového rozsahu funkcionality zabralo jen 9 řádků. Testuje se správná reprezentace jednoho i více klíčových slov (`running` a `paused`).

Implementace

Pro reprezentaci vlastnosti `animation-play-state` jsem vytvořil enumeraci `AnimationPlayState` implementující rozhraní `CSSProperty`. Kromě 3 globálních hodnot (7.1) jsou deklarovány také konstanty pro klíčová slova (`running` a `paused`) a proměnná `list_values("")` pro všechny ostatní validní kombinace termů.

Validace je zajištěna přímo procesní metodou `processAnimationPlayState()` ve třídě `DeclarationTransformer`. Metoda nejprve zavolá generický test na zpracování jedné konstanty od zadané `CSSProperty`.

V případě selhání se cyklicky testují a přiřazují termy do předem připraveného seznamu. Propouštěny jsou pouze termy reprezentující identifikátor, jakožto potenciální klíčové slovo.

V případě chyby ve formě nalezení neočekávaného, či nenalezení očekávaného termu je proces ukončen s návratovou hodnotou `false`. Po úspěšném dokončení cyklu je do mapy vlastností vložena hodnota `AnimationPlayState.list_values` a do mapy hodnot seznam obsahující termy deklarace. Metoda vrací `true`.

7.5.13 Animace – Určení hodnoty před a po

Ve výchozím stavu není hodnota ovlivněné vlastnosti měněna, dokud se animace nespustí, tedy pokud neběží, je hodnota určena jinými CSS deklaracemi. Toto chování lze změnit nastavením vlastnosti `animation-fill-mode` na tyto hodnoty:

- `forwards` - ovlivněné vlastnosti zůstane přiřazena finální hodnota animace
- `backwards` - ovlivněné vlastnosti je ihned po aplikaci animace přiřazena její počáteční hodnota, relevantní při nastavení zpoždění (7.5.9)
- `both` - obě výše uvedené možnosti najednou

Gramatika

Gramatika validních hodnot vlastnosti `animation-fill-mode` je definována takto (3):

```
[ none | forwards | backwards | both ]#
```

Je tedy možné ji definovat jedním či více z daných klíčových slov v seznamu odděleném čárkami.

Testy

Testy pro vlastnost `animation-fill-mode` jsou umístěny ve třídě `AnimationTest`, zodpovědné za správnou funkci všech vlastností CSS animací. Díky využití nástrojů `TestUtils` otestování celkového rozsahu funkcionality zabralo jen 9 řádků. Testuje se správná reprezentace jednoho i více klíčových slov (`none`, `forwards`, `backwards` a `both`).

Implementace

Pro reprezentaci vlastnosti `animation-fill-mode` jsem vytvořil enumeraci `AnimationFillMode` implementující rozhraní `CSSProperty`. Kromě 3 globálních hodnot (7.1) jsou deklarovány také konstanty pro klíčová slova `NONE("none")`, `FORWARDS("forwards")`, `BACKWARDS("backwards")` a `BOTH("both")` a proměnná `list_values("")` pro všechny ostatní validní kombinace termů.

Validace je zajištěna přímo procesní metodou `processAnimationFillMode()` ve třídě `DeclarationTransformer`. Metoda nejprve zavolá generický test na zpracování jedné konstanty od zadané `CSSProperty`.

V případě selhání se cyklicky testují a přiřazují termy do předem připraveného seznamu. Propouštěny jsou pouze termy reprezentující identifikátor, jakožto potenciální klíčové slovo.

V případě chyby ve formě nalezení neočekávaného, či nenalezení očekávaného termu je proces ukončen s návratovou hodnotou `false`. Po úspěšném dokončení cyklu je do mapy vlastností vložena hodnota `AnimationFillMode.list_values` a do mapy hodnot seznam obsahující termy deklarace. Metoda vrací `true`.

7.5.14 Animace – Časová funkce

Časová funkce přechodu se nastavuje vlastností `animation-timing-function`. Tato parametrická vlastnost je identická s `transition-timing-function` popsanou v sekci 7.5.5.

Testy

Testy pro zkratkovou vlastnost `animation-timing-function` jsou umístěny ve třídě `AnimationTest`, zodpovědné za správnou funkci všech vlastností CSS animací. Díky využití nástrojů `TestUtils` otestování celkového rozsahu funkcionality zabralo jen 9 řádků.

Testuje se správná reprezentace všech časových funkcí ve tvaru klíčových slov i doslovné formě s typickými validními kombinacemi parametrů. Dále také invalidace celé deklarace při detekci špatných parametrů funkce.

Implementace

Pro reprezentaci vlastnosti `animation-timing-function` jsem vytvořil enumeraci `AnimationTimingFunction` implementující rozhraní `CSSProperty`. Kromě 3 globálních hodnot (7.1) jsou deklarovány také konstanty pro klíčová slova `LINEAR("linear")`, `EASE_IN("ease-in")`, `EASE_OUT("ease-out")`, `EASE_IN_OUT("ease-in-out")`, `EASE("ease")`, `STEP_START("step-start")` a `STEP_END("step-end")` a proměnné `timing_function("")` pro jedinou časovou funkcí a `list_values("")` pro všechny ostatní validní kombinace termů.

Implementace samotných časových funkcí je již popsána v sekci o jejich použití pro CSS přechody 7.5.5.

Validaci vlastnosti zajišťuje procesní metoda `processAnimationTimingFunction()` ve třídě `DeclarationTransformer`. Metoda nejprve zavolá generický test na zpracování jedné konstanty od zadané `CSSProperty`.

V případě selhání se cyklicky testují a přiřazují termy do předem připraveného seznamu. Propouštěny jsou termy reprezentující identifikátor, jakožto potenciální klíčové slovo, nebo časovou funkci.

V případě chyby ve formě nalezení neočekávaného, či nenalezení očekávaného termu je proces ukončen s návratovou hodnotou `false`. Po úspěšném dokončení je do mapy vlastností vložena hodnota `AnimationTimingFunction.timing_function`, pokud byla nalezena jediná funkce, nebo `AnimationTimingFunction.list_values`, pokud jich bylo více. Stejně tak do mapy hodnot je vložen buď samotný term funkce, nebo celý seznam. Metoda vrací `true`.

Kapitola 8

Závěr

Cílem práce bylo rozšířit již existující projekt CSSBox[1] přidáním podpory nových CSS3 vlastností. Doplněny byly vlastnosti týkající se efektu stínu, mřížkového rozložení, animací a přechodů. Rozšíření bylo úspěšné a její změny jsou již aplikovány do centrálního repositáře projektu.

Implementace vlastností mřížkového rozložení byla použita v práci související s jeho vykreslením. Testovací nástroje vytvořené v průběhu vývoje, na sjednocené a efektivní zajištění funkcionality implementace CSS vlastností, byly navíc úspěšně použity dalším autorem.

Přínos této práce tedy zvýšil použitelnost projektu a napomohl k možným budoucím rozšířením. Potenciál bych viděl např. v implementaci i ostatních chybějících CSS vlastností, uvedených v sekci „Současná omezení projektu“ (B), nebo přidání hlubší podpory `@keyframes` (7.5.6) pro možnost přesnější specifikace chování v průběhu CSS animace.

V konečném důsledku hodnotím celkovou zkušenost kladně, práce s open source projektem je značně odlišná od typických uzavřených projektů se kterými se setkávám v zaměstnání, což mi rozhodně rozšířilo obzory. Na jednu stranu má člověk větší volnost v rozhodování o detailech implementace, na druhé straně ale tato volnost zapříčiňuje oslabenou až neexistující štábní kulturu, které se vývojář může držet při počátečním seznamováním s projektem i pozdějším odstraňování chyb a ladění v kódu.

[1] Veřejný repositář s projektem CSSBox – https://github.com/radkovo/jStyleParser

Literatura

[1] Burget, R.: *CSSBox screenshot*. [Online; navštíveno 1.3.2019].
URL http://a.fsdn.com/con/app/proj/cssbox/screenshots/318271.jpg

[2] Burget, R.: *CSSBox* ©. [Online; navštíveno 1.3.2019].
URL http://cssbox.sourceforge.net

[3] Burget, R.: *CSSBox* © – *jStyleParser*. [Online; navštíveno 1.3.2019].
URL http://cssbox.sourceforge.net/jstyleparser

[4] Burget, R.: *CSSBox* © – *jStyleParser manuál*. [Online; navštíveno 1.3.2019].
URL http://cssbox.sourceforge.net/jstyleparser/manual.php

[5] Burget, R.: *CSSBox* © – *Pdf2Dom*. [Online; navštíveno 1.3.2019].
URL http://cssbox.sourceforge.net/pdf2dom

[6] Burget, R.: *CSSBox* © – *SwingBox*. [Online; navštíveno 1.3.2019].
URL http://cssbox.sourceforge.net/swingbox

[7] Burget, R.: *CSSBox* © – *WebVector*. [Online; navštíveno 1.3.2019].
URL http://cssbox.sourceforge.net/webvector

[8] Burget, R.: *SwingBox screenshot*. [Online; navštíveno 1.3.2019].
URL
http://a.fsdn.com/con/app/proj/swingbox.cssbox.p/screenshots/swing1.jpg

[9] Cutrell, C. J.: *CSS 3 Logo*. [Online; navštíveno 1.3.2019].
URL https://s3.amazonaws.com/clarityfm-production/attachments/2082/default/css3-markup.jpg

[10] Mozilla; individual contributors: *Cascading Style Sheets syntaxe*. [Online; navštíveno 1.3.2019].
URL https://developer.mozilla.org/en-US/docs/Web/CSS/Syntax

[11] Mozilla; individual contributors: *CSS: Cascading Style Sheets*. [Online; navštíveno 1.3.2019].
URL https://developer.mozilla.org/en-US/docs/Web/CSS

[12] Mozilla; individual contributors: *CSS deklarace*. [Online; navštíveno 1.3.2019].
URL https://mdn.mozillademos.org/files/3665/css%20syntax%20-%20declaration.png

[13] Mozilla; individual contributors: *CSS deklaračnı́ blok 1*. [Online; navštíveno 1.3.2019].
URL https://mdn.mozillademos.org/files/3666/css%20syntax%20-%20block.png

[14] Mozilla; individual contributors: *CSS deklarační blok 2*. [Online; navštíveno 1.3.2019].
URL https://mdn.mozillademos.org/files/3667/css%20syntax%20-%20declarations%20block.png

[15] Mozilla; individual contributors: *CSS kroková funkce jump-both*. [Online; navštíveno 1.3.2019].
URL https://mdn.mozillademos.org/files/16420/step3both.png

[16] Mozilla; individual contributors: *CSS kroková funkce jump-none*. [Online; navštíveno 1.3.2019].
URL https://mdn.mozillademos.org/files/16419/step5none.png

[17] Mozilla; individual contributors: *CSS kroková funkce start*. [Online; navštíveno 1.3.2019].
URL https://mdn.mozillademos.org/files/3436/steps(2,start).png

[18] Mozilla; individual contributors: *CSS kubická Beziérova křivka*. [Online; navštíveno 1.3.2019].
URL https://mdn.mozillademos.org/files/3433/cubic-bezier,%20example.png

[19] Mozilla; individual contributors: *CSS pravidlo*. [Online; navštíveno 1.3.2019].
URL
https://mdn.mozillademos.org/files/3668/css%20syntax%20-%20ruleset.png

[20] Mozilla; individual contributors: *CSS Reference*. [Online; navštíveno 1.3.2019].
URL
https://developer.mozilla.org/en-US/docs/Web/CSS/Reference#Keyword_index

[21] Mozilla; individual contributors: *CSS syntaxe definice hodnoty*. [Online; navštíveno 1.3.2019].
URL
https://developer.mozilla.org/en-US/docs/Web/CSS/Value_definition_syntax

[22] Mozilla; individual contributors: *CSS časová funkce ease*. [Online; navštíveno 1.3.2019].
URL https://mdn.mozillademos.org/files/3429/cubic-bezier,ease.png

[23] Mozilla; individual contributors: *CSS časová funkce ease-in*. [Online; navštíveno 1.3.2019].
URL https://mdn.mozillademos.org/files/3426/cubic-bezier,ease-in.png

[24] Mozilla; individual contributors: *CSS časová funkce linear*. [Online; navštíveno 1.3.2019].
URL https://mdn.mozillademos.org/files/3425/cubic-bezier,linear.png

[25] Mozilla; individual contributors: *Mozilla – @keyframes pravidla*. [Online; navštíveno 1.3.2019].
URL https://developer.mozilla.org/en-US/docs/Web/CSS/@keyframes

[26] Mozilla; individual contributors: *Mozilla – Základy konceptu mřížkového rozložení*. [Online; navštíveno 1.3.2019].
URL https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Grid_Layout/Basic_Concepts_of_Grid_Layout

[27] Mozilla; individual contributors: *Mozilla – Časové funkce*. [Online; navštíveno 1.3.2019].
URL https://developer.mozilla.org/en-US/docs/Web/CSS/timing-function

[28] W3Schools.com: *CSS Reference*. [Online; navštíveno 1.3.2019].
URL https://www.w3schools.com/cssref/default.asp

[29] Wikipedia; individual contributors: *Test-driven development*. [Online; navštíveno 1.3.2019].
URL https://en.wikipedia.org/wiki/Test-driven_development

Příloha A

Podporované vlastnosti

Před započetím této práce projekt CSSBox podporoval analýzu, validaci a reprezentaci těchto 141 CSS vlastností:

Zobrazení textu

- `color`
- `opacity`
- `font (-family, -size, -style, -variant, -weight, -decoration, -transform)`

Zobrazení dokumentu

- `background (-attachment, -color, -image, -position, -size, -repeat)`
- `visibility`
- `overflow (-x, -y)`

Rozložení textu

- `white-space`
- `text-align`
- `text-indent`
- `line-height`
- `word-spacing`
- `letter-spacing`
- `vertical-align`
- `direction`
- `unicode-bidi`
- `tab-size`

Rozložení dokumentu

- `margin (-top, -right, -bottom, -left)`
- `padding (-top, -right, -bottom, -left)`
- `border (-top, -right, -bottom, -left, -collapse, -spacing)`
- `border-width (-top-, -right-, -bottom-, -left-)`
- `border-style (-top-, -right-, -bottom-, -left-)`

- border-color (-top-, -right-, -bottom-, -left-)
- border-radius (-top-left-, -top-right, -bottom-right-, -bottom-left)
- width (min-, max-)
- height (min-, max-)
- clip
- box-sizing
- display
- position
- top, right, bottom, left
- float
- clear
- transform (-origin)
- flex-direction
- flex-wrap
- flex-basis
- flex-grow
- flex-shrink
- order
- jastify-content
- aling-content
- align-items
- align-self
- z-index

Ostatní

- list-style (-type, -position, -image)
- empty-cells
- table-layout
- caption-side
- content
- quotes
- counter-increment
- counter-reset
- filter (backdrop-)
- cursor
- outline (-width, -style, -color)
- page-break (-before, -after, -inside)
- widows
- orphans
- azimuth
- cue (-before, -after)
- elevation
- pause (-before, -after)
- pitch (-range)
- play-during
- richness
- speak (-header, -numeral, -punctuation)

- speech-rate
- stress
- voice-family
- volume

```
public enum Color implements CSSProperty {
    color(""), INHERIT("inherit"), INITIAL("initial"), UNSET("unset");

    private String text;

    private Color(String text) {
        this.text = text;
    }

    public boolean inherited() {
        return true;
    }

    public boolean equalsInherit() {
        return this == INHERIT;
    }

    public boolean equalsInitial() {
        return this == INITIAL;
    }

    public boolean equalsUnset() {
        return this == UNSET;
    }

    @Override
    public String toString() {
        return text;
    }
}
```

Obrázek A.1: Ukázka vnitřní reprezentace vlastnosti Color

Příloha B

Nepodporované vlastnosti

Po prvotní analýze podporovaných stylů v projektu a srovnáním s vývojářskými referencemi (Mozilla [20], W3Schools [28]) jsem identifikoval 78 chybějících stylů, kterým je doporučována všeobecná kompatibilita. Výpis 78 CSS vlastností, jimž před započetím této práce chyběla podpora v projektu CSSBox. Přidané styly jsou označeny zeleně.

- `all`
- `animation`
- `animation-delay`
- `animation-direction`
- `animation-duration`
- `animation-fill-mode`
- `animation-iteration-count`
- `animation-name`
- `animation-play-state`
- `animation-timing-function`
- `backface-visibility`
- `background-blend-mode`
- `background-clip`
- `background-origin`
- `box-shadow`
- `caret-color`
- `column-count`
- `column-fill`
- `column-gap`
- `column-rule (-color, -style, -width)`
- `column-span`
- `column-width`
- `columns`
- `flex`
- `font-kerning`
- `font-stretch`
- `grid (-area, -row, -column)`
- `grid-gap (-row-, -column-)`
- `grid-row-start`
- `grid-column-start`

- grid-row-end
- grid-column-end
- grid-template (-areas, -rows, -columns)
- grid-auto-flow
- grid-auto-rows
- grid-auto-columns
- hyphens
- isolation
- justify-items
- justify-self
- mix-blend-mode
- object-fit
- object-position
- overflow-wrap
- perspective (-origin)
- pointer-events
- resize
- scroll-behavior
- text-align-last
- text-combine-upright
- text-decoration-color
- text-decoration-line
- text-decoration-style
- text-justify
- text-overflow
- text-shadow
- transform-style
- transition
- transition-delay
- transition-duration
- transition-property
- transition-timing-function
- user-select
- word-break
- word-wrap
- writing-mode