VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

AUTOMATIZACE WEBOVÉHO PROHLÍŽEČE

WEB BROWSER AUTOMATION

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. VOJTĚCH BASTL**
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 diplomové práce

Student: **Bastl Vojtěch, Bc.**
Program: Informační technologie Obor: Informační systémy
Název: **Automatizace webového prohlížeče**
Web Browser Automation
Kategorie: Web

Zadání:

1. Prostudujte existující technologie pro ovládání webových prohlížečů z aplikací třetích stran. Zaměřte se na řešení PhantomJS, Selenium WebDriver a další.
2. Seznamte se s detaily reprezentace zobrazené webové stránky v prohlížeči a souvisejícím aplikačním rozhraním jednotlivých prohlížečů.
3. Navrhněte architekturu aplikace, která na základě předem daného konfiguračního souboru provede posloupnost operací ve webovém prohlížeči. Uvažujte možnost vyplňování formulářů pomocí dodaných dat, klikání na odkazy a další.
4. Po dohodě s vedoucím navrhněte vhodnou reprezentaci vstupní konfigurace a implementujte navržené řešení pomocí vhodných technologií.
5. Proveďte testování vytvořeného řešení.
6. Zhodnoťte dosažené výsledky.

Literatura:

- Swicegood, T.: Programming Node.js, O´Reilly, 2012
- Žára, O.: JavaScript - Programátorské techniky a webové technologie, Computer Press, 2015

Při obhajobě semestrální části projektu 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: 22. května 2019
Datum schválení: 23. října 2018

Abstrakt

Tato práce se zabývá automatizací webového prohlížeče – nástroji, umožnující programové ovládání programu pro prohlížení webových stránek. Nejprve diskutuje existující řešení, s důrazem na nástroje z rodiny Selenium Suite a PhantomJS. Následně je probrána interní reprezentace webové stránky v renderovacích jádrech prohlížečů Gecko a WebKit. Práce se poté zaměří na aplikační rozhraní webového prohlížeče, které nabízí klientským skriptovacím jazykům. Zároveň zde budou zmíněny standardy, podporující tyto rozhraní. Jádro práce tvoří návrh a implementace nástroje, jež umožní, pomocí knihovny Selenium WebDriver, ovládat webový prohlížeč a provézt získání dat o webové stránce. Práce ukazuje vnitřní uspořádání, popisuje vstupní konfigurační soubor a aplikační rozhraní. Také se zabývá problematikou získání dat o stránce a jejich převod na jednotný strukturovaný výstup. Zároveň demonstruje funkčnost pomocí jednotkových testů a ovládání reálných webových stránek.

Abstract

This work deals with the automation of a web browser – the tools that allow programmatic control of the program for browsing the web pages. First, it discusses the existing solutions with focus on the tools from the Selenium Suite family and PhantomJS. Further, the internal representation of the web pages in the Gecko and WebKit browser engines is discussed. The work then focuses on the web browser application interface available for client-side scripting. The relevant standards are discussed as well. The core part of the thesis is dedicated to the design and implementation of a tool that allows to control a browser using the Selenium WebDriver tool and to extract data about the targert web page. The work presents an internal architecture, configuration files and the application interface of the designed tool. The topic of extracting detailed data about the page and its transformation to a unified structured description is covered as well. Finally, the performed unit tests and tests on real web pages are described.

Klíčová slova

Selenium, WebDriver, DOM, BOM, CSSOM, automatizace webového prohlížeče, renderovací jádro prohlížeče

Keywords

Selenium, WebDriver, DOM, BOM, CSSOM, Web Browser Automation, Browser engine

Citace

BASTL, Vojtěch. *Automatizace webového prohlížeče*. Brno, 2019. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Radek Burget, Ph.D.

Automatizace webového prohlížeče

Prohlášení

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

.....
Vojtěch Bastl
20. května 2019

Poděkování

Chtěl bych poděkovat Ing. Radku Burgetovi, Ph.D. za vedení a ochotu při řešení této práce. Dále bych chtěl poděkovat své rodině za podporu a trpělivost.

Obsah

Kapitola 1

Úvod

Jedním z velkých trendů 21. století je přesun klasických aplikací do webového prostředí. Takové aplikace se vyskytují ve všech představitelných formách, od malých, vyvíjených jedním člověkem či několika lidmi, po obrovské (komerční aplikace) vyvíjené velkými zkušenými týmy. Mohou svojí složitostí směle konkurovat klasickým desktopovým aplikacím. Některé webové aplikace zcela zastínili svůj klasický protějšek, jiné služby jsou v podvědomí lidí spjaté pouze s webovou platformou. Vzrůstající popularita je důsledkem benefitů, které přináší jak pro uživatele, tak pro vývojáře. V případě uživatelů se může jednat o okamžitou dostupnost (odpadá nutnost získání média a instalace), podporu všech zařízení či bezplatnou formu. V případě vývojářů přináší vznik aplikace ve webovém prostředí spoustu výhod: snadná distribuce nových verzí a záplat, rychlý vývoj nebo dostupnost po celém světě. I přes zjevné výhody musí vývojář na této platformě čelit několika výzvám:

- Programů schopných zobrazit webový obsah je velké množství a zároveň je nutné podporovat jejich starší verze.
- Často neexistující standardy (např. Browser Object Model nebo uživatelské rozhraní prohlížeče) či standardy, vznikající až jako reakce na zavedené technologie. Existují také situace, kdy standardů popisující stejnou technologii je větší počet a žádný z nich není určen jako autoritativní.
- Celé spektrum zařízení (s různým uživatelským rozhraním, velikostí a rozlišením obrazovky) obsahující webový prohlížeč.

Kvůli těmto problémům musí být záruka, že se aplikace chová korektně a je zobrazena očekávaným způsobem. Pro dosažení záruky funkčnosti slouží testování a z důvodu množství konfigurací je nutné využít testování automatizované. Jednou z nejdůležitějších komponent automatizovaného testování webových aplikací je programové řízení webového prohlížeče.

S automatizací webového prohlížeče je úzce spjato téma této práce. Jejím hlavním cílem je vznik nástroje, s jehož pomocí bude možné získat co nejvěrnější popis grafické podoby webové stránky. K dosažení požadované stránky je využito právě automatizace webového prohlížeče. Programové ovládání prohlížeče umožňuje získat obsah, který je jinak skryt za uživatelskými úkony (přihlášení uživatele, vyplnění formuláře atd.). Po získání požadovaného pohledu bude proveden sběr informací. V kontextu práce je tím myšlen zisk strukturovaného textového popisu prvků stránky. Pro splnění cíle je vytyčeno hned několik úkolů:

- Diskutování problematiky nástrojů pro programové ovládání webových prohlížečů.

- Seznámení se s reprezentací zobrazené webové stránky v prohlížeči.
- Popis aplikačních rozhraní, jež prohlížeč nabízí klientským skriptovacím jazykům.
- Rozbor architektury a vznik prototypu nástroje pro automatizaci webového prohlížeče a sběr informací o internetových stránkách.
- Rozšíření prototypu o další funkčnost a rozbor vzniklého nástroje.
- Důkladné otestování aplikace i v reálných podmínkách.
- Shrnutí poznatků.

V kapitole 2 se práce nejprve zaměří na srovnání testování ve webovém prostředí oproti testování klasickému. Následně se již text soustředí na automatizaci webového prohlížeče, její historii a existující nástroje. Velký důraz bude kladen na nástroje z rodiny Selenium Suite, jak z důvodu jejich popularity, tak proto, že byl nástroj z této rodiny vybrán do diplomové práce. Mimo jiné dojde ke zmínce o standardu WebDriver, jež je velice důležitým počinem ve snaze sjednocení ovládání webových prohlížečů.

Kapitola 3 popisuje kroky a použité struktury při vykreslování a zobrazení webové stránky. Hlavní důraz je kladen na renderovací jádra WebKit a Gecko, z důvodů jejich významu, Open-source povahy a dostupnosti materiálů.

V kapitole 4 text rozebírá aplikační rozhraní prohlížeče a standardy, jež jsou k němu vázány.

Následující kapitoly již popisují samotné řešení zadaného problému. Hlavním důvodem vzniku těchto kapitol je cíl, aby byl čtenář po jejich přečtení schopen nejen aplikaci využívat, ale v případě potřeby i rozšířit.

Práce se nejprve zaměří (viz kapitola 5) na rozbor cílů a architektonický návrh nástroje pro získání informací o webové stránce, jehož vznik je hlavním tématem této práce. Bude zde také probrán návrh konfiguračního souboru a množina operací s webovým prohlížečem. Výstup kapitoly je prototyp a poznatky, které poslouží jako základní kámen dalších kapitol.

Vzhledem k návrhu aplikace je jedním z mála světu vystavených částí aplikační rozhraní, a proto musí být jasně definováno. Tímto rysem se text zabývá v kapitole 6, kde jsou rozebrány jednotlivé skupiny funkcí a jejich chování při nekorektním použití.

Jádrem práce je kapitola 7. Zde jsou rozebrána témata, která se přímo dotýkají ovladače webových stránek a také nástroje obalující tuto funkčnost. Text popisuje možnosti spuštění aplikace skrz příkazovou řádku, postupy pro získání objektu, obsahující aplikační rozhraní, parametrizaci vzniku ovladače a jeho životní cyklus. Kapitola se následně zaměří na rozbor operací s webovým prohlížečem, zvláštní pozornost je věnována hromadnému vyplnění formuláře. Závěrem jsou ukázány praktiky, díky kterým došlo k zapouzdření knihovny Selenium WebDriver, a tudíž byla v budoucnosti možnost záměny za jinou implementaci.

Text se v kapitole 8 soustředí na postupy jimiž jsou získána data o webové stránce. Je zde rozebrán princip průchodu DOM stromu, uchování vztahů prvků stránky a předně popis získání dat o každém elementu webové stránky.

Korektní chování výsledné aplikace a demonstrace schopností se bude zabývat kapitola 9, zaměřující se na testování nástroje. Text je zde rozdělen do dvou hlavních celků: jednotkové testy a reálné testy. Jednotkové testy ověřují funkčnost přesně vytyčených částí nástroje, testy v reálném prostředí, ověřující vhodnost aplikace pro její určení. Valná část testování je sestavena z jednotkových testů, a to z důvodu, že jsou webové stránky velice živé a vytvořené konfigurační skripty přestanou být brzy aktuální.

Kapitola 2

Automatizované ovládání webového prohlížeče

Se vzrůstající popularitou webových aplikací vyvstala zároveň potřeba záruky správného chování i tohoto typu softwaru. Ať už je tento trend způsoben dostupností média či bezplatné formy, požadavky na bezpečnost, rozšiřitelnost, spolehlivost a dostupnost, získávají na čím dál větší důležitosti. Z těchto příčin přirozeně vznikla nutnost testování. I přesto, že je testování softwaru, běžícím ve webovém prostředí, blízké obvyklému přístupu při testování aplikací, má svá specifika a pouhé rozšíření klasického přístupu není lehké ani přímočaré. Webové prostředí obsahuje celou řadu výzev, jež u klasických aplikací neexistují nebo nedosahují takového významu [17, s. 220] [5, s. 2]:

- Webové prohlížeče, na nichž fungují zmíněné aplikace, jsou navrženy především pro použití člověkem, nikoli automatizovaným strojem.
- Obsahují velké množství vzájemně propletených technologií, které obstarávají jednotlivé aspekty webového světa a mají diametrálně odlišný syntax a sémantiku. HTML dodává obsah, CSS vzhled, Javascript [12] chování apod.
- Díky rozšiřující se dostupnosti internetu a, ve většině případů, bezplatné formě je velice přístupná. Toto ústí v konkurentní používaní velkým počtem lidí z celého světa.
- Softwaru pro zobrazení webové stránky, tedy webových prohlížečů, je celá řada. Mohou mít odlišnosti v tom, jak se chovají k webové stránce, a je nutné, aby aplikace korektně fungovala při použití jakéhokoli z nich.
- Dynamicky mění svůj stav a komponenty za běhu programu jako reakci na uživatelské akce.
- Neposledním rysem webových aplikací může být široké spektrum zařízení, která jsou schopna připojit se k internetu a otevřít webovou stránku. Zařízení, jež mají bohaté variace velikosti zobrazované stránky, různý hardware, operační systém i diametrálně jiné ovládání.

Všechny výše zmíněné aspekty musí být zohledněny při validaci a verifikaci programu. Tyto odlišnosti velice ztěžují jeden z přístupů k testování, a to manuální. Velké množství prostředí a zařízení, na kterých musí aplikace korektně fungovat, spolu s často komplikovaným grafickým vstupem a výstupem, nahrává k chybám způsobeným člověkem. Zároveň musí

být aplikace testována regresivně stejnou sadou testů, kdy by se stalo manuální testování časově neúnosným a vyčerpávajícím.

Z těchto a dalších důvodů musela vzniknout automatizace testování webových aplikací. Jedná se o velmi komplexní téma s technikami převzatými z konvekčního testování a s různými pohledy. Následující text se zaměří na jednu odlišnost od klasického testování. Jedná se o jedno z nejdůležitějších součástí automatického testování ve webovém prostředí, ovládání webového prohlížeče. Tedy sekvencí příkazů simulovat akce uživatele na navštívené webové stránce. Nástroje, jež slouží tomuto účelu, je možné posuzovat dle několika měřítek [21]:

Robustnost

Jak již z účelu nástroje vyplývá, měl by být schopen všech akcí, jež provádí uživatel s webovým prohlížečem při přístupu a používání webové stránky. Toto zahrnuje mimo jiné: vyplnění textového pole, odeslání formuláře, kliknutí na odkaz. Robustnost udává, které tyto operace nástroj zvládá.

Univerzálnost

Možnost použití nástroje s více různými prohlížeči. Do této metriky také spadá, zda je nástroj schopen ovládat některý z moderních prohlížečů, který využívají potenciální uživatelé webové aplikace. Druhá možnost, tedy vlastní renderovací jádro prohlížeče, simulovaný prohlížeč či využití již existujícího renderovací jádro je méně vhodná, a to z toho důvodu, že korektní chování je požadováno v reálném prostředí, oproti kterému může tento přístup produkovat rozdílné výsledky [5, s. 10].

Rychlost vykonávání

Rychlost provádění akcí s prohlížečem, např. nástroje, jež využívají pro vykonávání příkazů nad prohlížečem skriptovací jazyk Javascript, mohou být až příliš pomalé pro efektivní testování webové aplikace. Rychlost vykonávání je silně provázána s metrikou Univerzálnosti. Nástroj, využívající ke své činnosti pouze renderovací jádro a nikoli reálný prohlížeč je sice rychlý, ale může produkovat rozdílné výsledky. Příkladem může být PhantomJS, jež využívá renderovací jádro WebKit.

Integrace

Množina jazyků, ve kterých je možnost s nástrojem pracovat a posuzovat, jaká musí být vyvinuta snaha pro jeho zabudování do automatických testů webových aplikací. Integrace je většinou provedena pomocí API, jež nástroj k danému jazyku nabízí.

2.1 Historie

Jako u valné většiny vznikajících nástrojů, knihoven či aplikací i u ovládání webového prohlížeče lze pozorovat určitý vývoj v čase a postupné upřesnění požadavků na jeho funkčnost.

Na začátku testování webových aplikací byl často používán nástroj HttpUnit. Tester vytvořil HTTP požadavek na zkoumanou stránku a prozkoumal odpověď. I když může připadat tento nástroj jako již zastaralý, stále může nabídnout výhody u specifického okruhu testování aplikace. V práci *Agile Security Testing of Web-Based Systems via HTTPUnit* [23]

je ukázán jeden takový příklad zaměřený do oblasti bezpečnosti. HttpUnit lze nastavit tak, aby ignoroval veškerou validaci formulářů na straně klienta. Tímto způsobem může být otestována serverová část aplikace na náchylnost k nesprávným hodnotám poslaných při odeslání vyplněného uživatelského formuláře.

Za další stupeň vývoje by se dal označit posun z protokolové úrovně testování na uživatelsky přívětivější nástroje, kdy k ovládání prohlížeče slouží jazyk Javascript. Jedním ze zástupců může být nástroj Selenium Core, kdy jsou pokyny popisovány v proprietárním jazyce Selenese, který je následně na stránce vykonán v Javascriptu. Poněkud pokročilejším zástupcem jsou nástroje využívající známý programovací jazyk, např. Selenium RC, jež poskytuje API širokému spektru programovacích jazyků. Na druhé straně leží PhantomJS, jehož pokyny jsou psány přímo v jazyce Javascript.

I toto řešení má však své nedostatky. Hlavním je, že PhantomJS nevyužívá skutečný prohlížeč, ale pouze renderovací jádro WebKit. Tento problém byl částečně odbourán vznikem headless Chrome a headless módem ve Firefox, ale bohužel měl každý z nich jiné metody pro interakci se stránkou. Dalším může být rychlost provádění a dynamické stránky používající AJAX. Tyto nedostatky daly za vznik WebDriver Internet standard, který popisuje API k jednotnému ovládání prohlížeče. Ve spojení s dříve zmíněným Seleniem vznikl nástroj Selenium WebDriver, jehož metody přímo ovládají prohlížeč, odbourávají mezivrstvu v jazyce Javascript, a tím se zbavují problémů s jeho využíváním.

2.2 Selenium Suite

Název Selenium zastřešuje celou rodinu Open-source produktů k automatizaci testování webového softwaru. Jedná se o sadu nástrojů, jež využívá více než 20 tisíc společností a v oblasti automatizovaného testování webových aplikací je jednoznačně nejhojněji používanou [1]. S podporou všech hlavních operačních systémů, nejoblíbenějších webových prohlížečů a velké škály programovacích jazyků, není divu, že tyto nástroje využívají organizace jako MIT, Google, Fitbit a další[2]. Některé důvody, proč je Selenium tak oblíbené[3][4]:

+ Pestrá škála podporovaných programovacích jazyků

Jednou z měřítek využitelnosti testovacího nástroje je množina jazyků, v nichž se dá využít. V tomto ohledu je Selenium vysoce flexibilní. Podporuje: Python, Perl, Ruby, Java, C#, PHP a další [5].

+ Nezávislost na programovacím jazyce aplikace

Skripty či programy určené pro danou webovou stránku nebo aplikaci mohou být psané v libovolném podporovaném programovacím jazyce, jež Selenium podporuje.

[1] Selenium Use https://idatalabs.com/tech/products/selenium

[2] Companies using Selenium https://stackshare.io/selenium/in-stacks#/

[3] The good and bad of selenium https://www.altexsoft.com/blog/engineering/the-good-and-the-bad-of-selenium-test-automation-tool/?fbclid=IwAR091X4YRupY5M7ENseU9T1MrTTfn4c4jppSqVYEFz7BswfLAXaXjnrGnTc

[4] 10 topmost reasons https://softwaretestingtips2.wordpress.com/2013/06/11/10-topmost-reasons-why-you-should-use-selenium/

[5] programming languages https://www.seleniumhq.org/about/platforms.jsp#programming-languages

+ Podpora mnoha prohlížečů

Zejména Selenium WebDriver je schopno využít a ovládat nejpoužívanější webové prohlížeče. Tento bod je velice důležitý vzhledem k potřebě, aby webová aplikace běžela korektně v hlavních prohlížečích. Nebýt této podpory, muselo by se využít více různých nástrojů pro pokrytí prohlížečů.

+ Otevřený software zdarma

Rodina nástrojů z rodiny Selenium je k dostání zdarma a jedná se o software s otevřeným zdrojovým kódem.

+ Simultánní provádění

Možnost simultánního provádění testů na více zařízeních. Tímto lze lehce docílit zkrácení doby provádění testů nebo simulovat konkurentní přístup k webové aplikaci.

+ Podpora nejdůležitějších platforem

Selenium nabízí podporu platforem Windows, Linux, Mac, a díky dodatečným nástrojům i Android a IOS. Tyto nástroje jsou dva: Selendroid (podporující Android i IOS) a Appium (pouze IOS).

+ Integrace do různých vývojových platforem

Jako jsou Jenskins, Maven, TestNG a jiné.

+ Je využíván rozsáhlou skupinou společností a jednotlivců

Jak již bylo zmíněno, do svých projektů začlenilo Selenium Suite obrovské množství společností a vývojářů. Toto znamená, že většina problémů je již zodpovězena a ověřena. Existuje množství kurzů a také návodů.

V neposlední řadě vzniklo z této oblíbenosti nesčetně doplňků a rozšíření, a také nástroje, jež obalují Selenium, a tím rozšiřují jeho funkčnost.

I přes tyto výhody lze najít i některé nedostatky či nevýhody:

- Programové testování

Jedná se o testování, při němž je nutno psát testovací skripty či programy. Důsledek tohoto je, že tester musí ovládat minimálně jeden podporovaný jazyk Selenia.

- Zachycení a vyhodnocení snímku aplikace

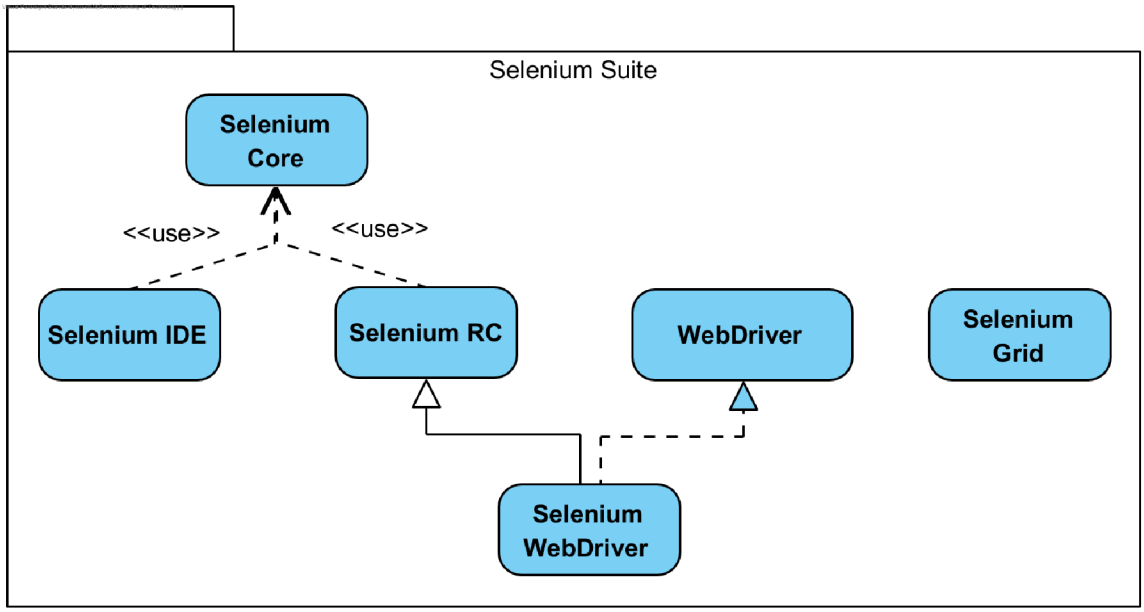
Jednou z užitečných praktik při testování webových aplikací či stránek je podpora porovnání referenčního snímku aplikace se snímkem zachycením v průběhu provádění testů. Selenium tuto funkcionalitu v základu neobsahuje, lze ji ale získat použitím nástroje třetích stran. Jedním z nich může být Sikuli.

2.2.1 Nástroje

Jak bylo již zmíněno, Selenium Suite sdružuje hned několik nástrojů, konkrétně se jedná o [24]:

- Selenium Core – vykonávání akcí v prohlížeči pomocí Javascriptu.
- Selenium IDE – doplněk do webového prohlížeče. Nahrává uživatelské akce a následně je, pomocí Selenium Core, zopakuje kdykoli, kdy je potřeba.
- Selenium RC – možnost volání funkcí Selenium Core v několika jazycích.
- Selenium WebDriver – přímé ovládání webového prohlížeče pomocí ovladače dodaného vývojáři prohlížeče.
- Selenium Grid – možnost provádění paralelních testů na více zařízeních.

Vztahy mezi těmito nástroji je naznačen na obrázku 2.1.



Obrázek 2.1: Naznačení vztahů mezi nástroji Selenium Suite.

2.2.2 Selenium Core

Jedná se o nástroj, jež ovládá hostitelský webový prohlížeč pomocí aplikace v jazyce Javascript běžící v prohlížeči. K nejzákladnějšímu popisu akcí, které mají být provedeny slouží proprietární jazyk Selenese. Příkazy tohoto jazyka jsou sdruženy v HTML tabulce se sloupci pro druh příkazu, prvek stránky a popřípadě hodnotu [4]. Příkazy jazyka Selenium je možno roztřídit do tří kategorií [9]:

- Akce – manipulují se stavem aplikace. Jako příklad může sloužit zaškrtnutí `checkbox` nebo kliknutí na odkaz. Pokud akce selže, vykonávání testu je ukončeno.
- Přístupové – získávají stav aplikace, např. `storeText(locator,variableName)`, pro získání textu elementu.
- Utvrzovací – obdobné přístupovým příkazům, ale potvrzují, že stav aplikace je stejný jako očekávaný. Příklad takového příkazu může být: Ujisti se, že je název stránky X.

2.2.3 Selenium IDE

Slouží jako nástroj pro prototypování a tvorbu testovacích skriptů. Existuje ve formě doplňku pro prohlížeče Firefox a Chrome. Pro vznik testu uživatel provádí na dané stránce akce, Selenium IDE tyto akce nahrává. Následně je možnost s využitím Selenium Core akce zopakovat. I přes možnost uložit provádění testu do tabulkového formátu, a jeho následné znovupoužití, vznikl tento nástroj pouze pro rychlé prototypování a není tedy určen pro vznik všech automatických testů, které budou potřeba. V důsledku toho Selenium IDE neumožňuje podmínky ani cykly.

2.2.4 Selenium RC

Selenium Remote Control (či pod jiným názvem Selenium 1.0), přidává možnost psát testovací skripty v jazyku dle testerovi preference. Odpadá tedy nutnost použití jazyka Selenese nebo provádění uživatelských akcí. Oproti Selenium IDE se již jedná o plnohodnotný testovací nástroj, který byl dlouhou dobu hlavním projektem Selenia. Selenium RC se skládá ze dvou hlavních částí:

- Selenium Server – obdrží příkazy pomocí HTTP GET/POST požadavků, vykoná je a vrátí programu výsledky. Jeho hlavní součástí je Selenium Core, jež se skládá z množiny funkcí v jazyce Javascript, které interpretují a provádějí Selenium příkazy, přičemž využívají zabudovaný interpret Javascriptu ve webovém prohlížeči. Core je automaticky vložen do prohlížeče v okamžiku kdy testovací program otevře okno prohlížeče.
- Klientské knihovny – poskytují rozhraní mezi každým programovacím jazykem a Selenium Serverem.

Hlavní nevýhodou Selenium RC je, že používá k ovládání prohlížeče stále Selenium Core, tedy Javascript. Vykonávání Javascript kódu může být až příliš pomalé. Webové prohlížeče na tento kód aplikují zásady bezpečnosti a Javascript se jen těžko vyrovnává se stránkami s dynamickou povahou, tedy stránkami využívajícími AJAX volání. I když mohou být některé z těchto omezení odstraněny [25], dala za vznik nástroje Selenium 2.0.

2.2.5 Selenium WebDriver

Vznikl díky spojení dvou technologií. Selenium 2.0 (rozhraní vycházející ze Selenium RC) a WebDriver (standard World Wide Web Consortium pro rozhraní na ovládání webového prohlížeče). Díky spojení vznikl nástroj, který odstraňuje většinu největších nedostatků Selenium RC jako jsou: rychlost, nemožnost nahrávání a stahování souborů, vyskakovací okna a dialogy, Same-origin policy[6] a AJAX volání. Jak rozebírá práce pana Gojare et. al [10] mohou kolem tohoto nástroje vzniknout komplexní nástroje pro testování.

WebDriver

Poskytuje jazykově a platformě nezávislý síťový protokol, jež umožňuje programům zpřístupnit vzdálené ovládání prohlížeče [22]. Při vzniku specifikací bylo přihlédnuto k již dlouhodobě žijícímu projektu Selenium [1, s. 57]. Vzhledem k délce jeho trvání a rozšíření na trhu

[6]Same-origin policy https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin_policy

se již v jeho standardu odráží očekávaná funkčnost a syntax. WebDriver protokol slouží pro komunikaci mezi dvěma uzly: Local end a Remote end.

Local end v tomto případě reprezentuje klientskou stranu protokolu. Ve většině případů se jedná o jazykově specifickou knihovnu, poskytující API k příkazům z protokolu.

Remote end (serverová strana protokolu) se může vyskytovat ve dvou formách. Intermediary node, jež se chová pouze jako proxy, implementuje Local End i Remote End protokolu, ale neprovádí koncové úkony a Endpoint node, poslední článek v řetězu Remote End uzlů, jež není Intermediary node. Endpoint node je implementován v rámci webových prohlížečů. Všechny typy Remote End musí být implementovány jako černá skříňka a nesmí být od sebe rozeznatelné.

Komunikace s využitím WebDriver protokolu probíhá pomocí jednotlivých příkazů, které mají formu HTTP koncových bodů. Rozeznávány jsou HTTP metody GET, POST a DELETE. Pokud je třeba dalších parametrů nebo návratové hodnoty, využije se JSON.

Pro udržení kontinuity mezi local a remote end je využito techniky Session. Sezení je ekvivalentní jedné instanci webového prohlížeče. Každému ze sezení je přiřazeno unikátní Session ID, jež slouží k jejich vzájemnému rozlišení. Tento koncept zajišťuje, že jeden HTTP server může ovládat více než jeden webový prohlížeč. Téměř všechny příkazy z protokolu WebDriver (výjimku tvoří příkaz pro vytvoření nového Session a příkazu Status) obsahují Session ID, které jednoznačně určí, pro jaké sezení je příkaz určen. Každý Remote end uzel má asociační tabulku aktivních sezení, jež je seznam všech sezení, které jsou aktuálně započaté. Asociační tabulka Endpoint uzlu obsahuje maximálně jeden záznam.

2.3 Watir

Obdobně jako Selenium, prodělal projekt Watir bouřlivý a (v měřítku informačního světa) dlouhý vývoj. Na počátku, v roce 2001, byla knihovna `cliec`, jejímž autorem byl Chris Morris. Tento kód napsaný v programovacím jazyce Ruby, měl za úkol ovládat prohlížeč Internet Explorer. Ovládání docílil odlišnou technikou než obdobné knihovny, jež ke stejné funkčnosti využívaly skriptovací jazyk Javascript či HTTP protokol. Watir používal schopností Ruby a Internet Exploreru. Ruby dodalo zabudovanou schopnost využít Object Linking & Embedding, zkráceně OLE. OLE je technologie vyvinutá firmou Microsoft, dovolující dynamicky propojit soubory a aplikace dohromady. Internet Explorer nabízí rozhraní Component Object Model, jež umožňuje skoro úplnou kontrolu nad prohlížečem, toto zahrnuje i objekty webové stránky. Díky těmto technologiím byl Watir schopen velice efektivně programově řídit Internet Explorer [8, s. 336]. Nad touto technologií vznikly i některé knihovny, využívající ke své funkci Watir. Příklad může být nástroj WET [9, s. 1913], který kromě funkcionality společné s Watir, nabízí simulovaný prohlížeč, nahrávání uživatelských akcí a jejich opakování.

Hlavní nedostatky nástroje Watir je možno velice jednoduše určit: možnost ovládat pouze Internet Explorer a nutnost vzniku skriptů v programovacím jazyce Ruby. V přechodném období vznikaly, za účelem udržení kroku s nástrojem Selenium, projekty pro specifické prohlížeče (např. FireWatir či OperaWatir) a také pro další programovací jazyky (Watij pro programovací jazyk Java a Watin s podporou. Net frameworku). S příchodem WebDriver standartu bylo rozhodnuto využít tuto technologii, paralelně však byla stále udržována verze pro Internet Explorer. Od této verze bylo definitivně upuštěno ve chvíli, kdy Microsoft představil prohlížeč Edge a prohlásil, že končí s vývojem nových technologií pro Internet Explorer. V této chvíli nemá být nástroj Watir brán jako konkurence Selenium,

ale spíše nástroj pro odlišný účel. Selenium, jako knihovna pro řízení prohlížeče a Watir, jakožto knihovna pro automatizaci testů, která ke svému chodu využívá kód Selenia.

2.4 PhantomJS

Životní cyklus PhantomJS začal již na jaře 2011, kdy ho Ariya Hidayat, po několika letech vývoje, zveřejnil a započal tak komunitní snahu, o vývoj a údržbu nástroje pro programové řízení Headless prohlížeče, trvající 7 let. Do této činnosti se zapojilo více než 130 vývojářů[7]. I takové množství přispěvatelů nestačilo a 3. 3. 2018 oznámil sám zakladatel Ariya z důvodu malého množství příspěvků archivaci projektu, a tím de facto ukončení jeho vývoje[8].

Ve své podstatě je PhantomJS pouhé specializované sestavení renderovacího jádra WebKit. Díky použití WebKit mohou mít vývojáři jistotu, že testy jsou prováděné v moderním renderovacím jádře. Tuto záruku posiluje fakt, že WebKit současně využívá populární moderní prohlížeč Safari, a také Firefox a Chrome pro platformu iOS. K tomuto renderovacímu jádru PhantomJS poskytuje API v jazyce Javascript, které propůjčuje plně programovatelné ovládání prohlížeče.

Headless browser

Jedná se o prohlížeč, jež nemá žádné grafické uživatelské rozhraní a který je ovládán z příkazové řádky, po síti či s použitím API v programovacím jazyku. Headless přístup má určité benefity. Ať už se jedná o rychlost, lehkou až neexistující konfiguraci, nebo menší nároky na výpočetní výkon. Díky těmto klíčovým výhodám je často použit u testování webových aplikací, automatizace úkonů s prohlížečem nebo sbírání dat z webových stránek. S rychlejším přístupem na webovou stránku však přicházejí i negativní důsledky. Zde lze uvést její zneužití v podobě útoků na odepření webové služby, Denial of Service (DOS) či Distributed Denial od Service (DDOS)[9]. Zároveň však z jeho názvu vyplývá hlavní nedostatek. Není to prohlížeč, jež používá typický uživatel a nehodí se tedy na všezahrnující testy webových aplikací na nejpopulárnějších webových prohlížečích, místo toho může být použit jako rychlé ověřovací testy základní funkčnosti. Možná nejznámější využití headless prohlížeče je nástroj PhantomJS, ale mimo něho nabízí toto rozhraní i populární prohlížeče typu Chrome, Firefox či Opera.

[7] PhantomJS GitHub https://github.com/ariya/phantomjs

[8] Archivace https://github.com/ariya/phantomjs/issues/15344

[9] 150 Hour DDOS attack https://www.business2community.com/tech-gadgets/headless-browser-botnet-used-150-hour-ddos-attack-0688767

Kapitola 3

Reprezentace webové stránky

Práce *A reference architecture for Web browsers* [11] uvádí, že se Webový prohlížeč skládá z několika důležitých bloků. Jeden z nich, který zde bude podrobněji probrán, se nazývá renderovací jádro prohlížeče. Mimo jádro zde může být zmíněn také interpret jazyka Javascript, modul pro práci se sítí, uživatelské rozhraní či persistence dat. Renderovací jádro prohlížeče slouží k načtení a zpracování informací ze souborů ve značkovacím jazyce (HTML, XML), informacích o vzhledu stránky (CSS, XSL) a dalších objektů. Následně zobrazí naformátovaný obsah do okna webového prohlížeče. V dnešní době webových aplikací nabývá na důležitosti i život stránky po jejím načtení. Důležitá je rychlost reakce webové stránky na uživatelské akce a skriptovací jazyky, jež mohou zahrnovat posun stránky, změnu velikosti stránky, přidání obsahu na stránku apod. Všechny akce musí být navíc plynulé pro uživatele, tedy běžet v 60 snímcích za sekundu, z čehož vyplývá 16 ms na výpočet a vykreslení jedné stránky. Z tohoto důvodu renderovací jádro klade důraz na efektivní interní uložení webové stránky. Naopak je vyvíjena snaha dosáhnout vysokého stupně abstrakce ve formě jednotného aplikačního rozhraní všech struktur, jež jsou dostupné uživatelským skriptům. Mezi nejpoužívanější renderovací jádra prohlížečů patří:

- WebKit – použit v prohlížeči Safari na platformě macOS.
- Gecko – renderovací jádro zabudováno ve webovém prohlížeči Mozilla Firefox.
- Blink – jedná se o Fork[1] modulu WebCore renderovacího jádra WebKit. Blink je využitý v prohlížeči Google Chrome od verze 28 a Opera.
- EdgeHTML – renderovací jádro prohlížeče Microsoft Edge. Nejmladší z výčtu, původní verze vyšla v roce 2015. V prosinci 2018 ohlásila firma Microsoft přerušení jeho vývoje a nahrazením projektem Chromium, tedy jádrem Blink [2].
- Trident – proprietární jádro použito v prohlížeči Internet Explorer.

Renderovací jádra využívají odlišné techniky pro proces, při kterém z HTML dokumentu vznikne grafický výstup. Rozdíly se nicméně týkají převážně snahy snížit paměťovou a výpočetní náročnost a tedy hlavní myšlenkový tok může být popsán. V textu níže bude popsán takový proces se zaměřením na renderovací jádra Gecko a WebKit a to z důvodu jejich Open-source povahy.

[1] Fork https://help.github.com/articles/fork-a-repo/

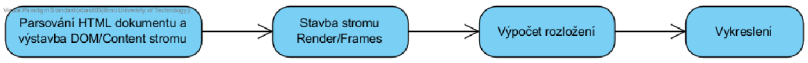
3.1 Obecný průchod

Po přijetí dat ze sítě začíná proces zobrazení webové stránky [7]. Průchod začíná parsováním HTML souboru a jeho transformací do podoby DOM stromu, některá renderovací jádra využívají názvu Content Tree. V rámci HTML souboru se typicky vyskytují informace o vzhledu v jazyce CSS. Tvary, v jakých se mohou tyto informace vyskytovat jsou: externí stylový soubor, styly zapsané jako atribut u HTML značky a stylová pravidla ve značce `style`. CSS je ve všech svých podobách zpracováváno simultánně s tvorbou DOM stromu.

Spojením Content stromu a stylových informací, vznikne následně další stromová struktura, Render Tree, který je v rámci jádra Gecko nazýván Frames Tree[2]. Render Tree jsou již obdélníky s vlastnostmi jako barva pozadí a font. Uzly ve stromě jsou zároveň ve správném uspořádání pro vykreslení.

Po dokončení konstrukce Render Tree pokračuje výpočet rozložením jeho uzlů, tedy uspořádáním všech obdélníků v rámci plochy webového prohlížeče.

Posledním krokem je vykreslení již rozložených prvků. Přehled obecných kroků nutných pro zpracování a zobrazení webové stránky je zobrazen na obrázku 3.1.



Obrázek 3.1: Typické kroky renderovacího jádra

Nakonec je nutné podotknout, že proces zobrazení webové stránky není lineární. Renderovací jádro nečeká se stavbou Render Tree, ani s následujícími kroky, až skončí krok předchozí. Dokonce nečeká na načtení celého dokumentu z webového serveru. V zájmu lepšího uživatelského zážitku začne co nejdříve dodávat grafický obsah. Části stránky tedy mohou být zobrazeny ještě před úplným zpracováním HTML dokumentu.

3.2 Render Tree/Frames Tree

Tato stromová struktura představuje spojení DOM stromu a informací o stylech. Další pohled může být jako na strom grafických obdélníku, jejichž umístění v rámci stromu odpovídá pořadí, v jakém budou vykresleny. Renderovací jádro Gecko nazývá uzly takového stromu Frames, WebKit využívá pojmů Renderer či Render Object. Každý Renderer odpovídá za obdélníkovou oblast srovnatelnou s CSS2 specifikací pro Box model[3]. Další vlastností uzlu je schopnost vykreslit sebe a své děti.

Podobně jako v stromu DOM mají objekty třídní hierarchii, ta může být určena např. CSS vlastností `display` nebo typem značky.

3.2.1 Vztah k DOM stromu

Jak již bylo řečeno, strom Render vychází z reprezentace dokumentu pomocí DOM stromu. Stromy jsou podobného tvaru, nicméně nejsou totožné. Rozdíl tkví v počtech uzlů Render stromu, které odpovídají uzlu stromu DOM. Některým uzlům DOM stromu odpovídá více uzlů v Render stromu. Naopak prvku stromu DOM nemusí odpovídat žádný objekt Renderer.

[2]Gecko Layout https://wiki.mozilla.org/Gecko:Overview#Layout
[3]Box model https://www.w3.org/TR/2011/REC-CSS2-20110607/box.html#box-model

Prvky stromu DOM, které nejsou vizuálního charakteru, nemají odpovídající Renderer. Příkladem takového prvku může být značka `script`. Další DOM uzel, jež nemá svůj protějšek, je takový, co má CSS vlastnost `display` nastavenou na `none`, naopak prvek s vlastností `visibility` s nastavením `hidden` bude do stromu Render zařazen. Z druhé strany spektra existují DOM prvky, kterým odpovídá větší počet uzlů ve stromě Render Tree. Tento případ nastává s prvky, které mají složitou strukturu a nemohou být popsány jedním obdélníkem. Tomuto typu odpovídá značka `table`, která v sobě obsahuje další obdélníky či ovládací prvky formulářů. Více Render uzlů může mít i element, který je obsažen přes více stránek.

Další případ, kdy strom Render neodpovídá stromu Content, jsou uzly, které sice jsou ve stromu Render, ale na odlišném místě než ve stromu DOM. Tento případ nastane při plovoucím a absolutním pozicování, kdy je prvek vyjmut z toku a vložen na jiné místo dokumentu.

3.2.2 Výpočet stylů

Každý uzel v rámci Render stromu musí znát své vizuální vlastnosti. Toto je zajištěno spočtením všech stylových vlastností pro každý prvek stránky a následně jejich přiřazení k odpovídajícímu objektu Renderer. Efektivní výpočet stylů obnáší některá úskalí:

1. Obsáhlost – stylů může být velké množství napříč soubory.
2. Hledání odpovídajících pravidel – hledat shodu pravidla oproti každému prvku je bez jakýchkoli optimalizací velice nákladné. Selektory mohou být velice komplexní a jejich podoba může zapříčinit, že začátek porovnávání s určitou větví DOM stromu vypadá velice slibně a až po určité době se pravidlo vyřadí.
3. Uplatnění pravidel kaskády – pokud pro některou z CSS vlastností existuje více definovaných hodnot, je nutnost uplatnit prioritu kaskády a pravidel CSS jazyka[4] pro určení priorit hodnot.
4. Výpočet konkrétních hodnot – hodnoty CSS vlastností je nutno převézt z relativních jednotek na jednotky fixní a absolutní.

Z těchto důvodů a také proto, že v dnešní době je kladen velký důraz na život stránky po načtení a její interakce s uživatelem a skripty, bylo nutné zavést některé paměťové a výkonnostní optimalizace. Mimo jiné se jedná o sdílení struktur se styly mezi prvky stránky, zavedení dalších struktur pro lehčí určení platných pravidel a výpočet stylů [6]. Tyto optimalizace již nejsou předmětem této práce.

3.3 Výpočet rozložení

Výstupem fáze Render je strom vizuálních komponentů, které však neznají svoji pozici ani rozměry. Fáze výpočtu geometrie jsou nazývány Layout (jádro WebKit) a Reflow (jádro Gecko). Výpočet rozložení využívá souřadnicového systému s počátkem na souřadnicích 0,0 v levém horním rohu viditelného okna prohlížeče.

Výchozí HTML model rozložení je založen na toku[5]. Tok propůjčuje vlastnost, díky které prvky později v toku pravděpodobně neovlivní dřívější prvky. Ve většině případů

[4] Cascade and inheritance https://developer.mozilla.org/en-US/docs/Learn/CSS/Introduction_to_CSS/Cascade_and_inheritance

[5] Normal Flow https://developer.mozilla.org/en-US/docs/Learn/CSS/CSS_layout/Normal_Flow

je tedy možné spočíst geometrii všech prvků na první průchod. Výpočet rozložení tedy může probíhat zleva–doprava a shora–dolů. Jednou z výjimek může být tabulka HTML, potřebující více než jeden průchod.

Vlastní výpočet rozložení startuje v kořenovém uzlu stromu Render. Tento uzel odpovídá značce `html`, je umístěn na souřadnicích 0,0 a má velikost stejnou jako viditelné okno prohlížeče. Výpočet následně rekurzivně pokračuje stromem Render a spočte geometrii všech uzlů, jež to vyžadují.

3.3.1 Postup výpočtu rozložení

Aby bylo možné správně rozložit prvky na obrazovce prohlížeče, je nutné určit jejich výšku a šířku. Šířku stránky je Renderer schopen určit sám, ale zjištění výšky se musí účastnit i potomci uzlu. Až si každý potomek spočte svoji hodnotu, Renderer si z těchto hodnot a hodnot svých odsazení určí výšku.

3.4 Vykreslení

V této je procházen strom Render a volána metoda objektů Renderer pro vykreslení. Metoda dokáže vykreslit obsah na obrazovku pomocí UI infrastruktury jednotlivých komponentů.

Kapitola 4

Alikační rozhraní webové stránky

Jeden z pohledů na webovou stránku může být jako na množinu aplikačních rozhraní, jež nabízí webový prohlížeč pro použití skriptovacími jazyky. Pomocí těchto API je klientský skript schopen řídit webovou stránku během jejího načítání, i její život po načtení. Obecně lze říci, že má každé takové rozhraní odkrýt interní strukturu prohlížeče tak, aby se s touto vysoce optimalizovanou reprezentací webové stránky pracovalo pohodlně a soudržně napříč prohlížeči. Bohužel toto kritérium nebylo vždy splněno. Ať již z důvodů, že některé prohlížeče splňovaly pouze části standardů, o standard určující API se staralo více skupin či se standard odklonil od reality.

4.1 Browser Object Model

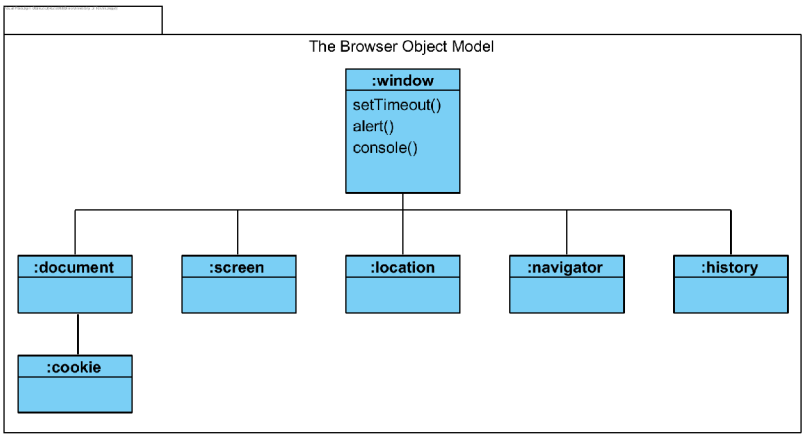
Nejdůležitější z rozhraní prohlížeče jsou Document Object Model, CSS Object Model a API pro komunikaci s webovým prohlížečem. Tyto rozhraní a další funkce, pomocí nichž skriptovací jazyk komunikuje s webovým prohlížečem, jsou sdruženy pod názvem Browser Object Model. I přesto, že nebyl dotyčný model nikdy standardizován, moderní prohlížeče poskytují, až na výjimky, stejné metody a funkce. V případě potřeby podporovat starší prohlížeče či starší verze prohlížečů, musí být k faktu neexistujícího standardu přihlédnuto. BOM je typicky dostupný skrz globální objekt, jež webový prohlížeč dodá skriptovacímu jazyku. V jazyce Javascript, při použití ve webovém prohlížeči, se jedná o globální objekt s názvem `window`[1]. Mimo rozhraní webové stránky a webového prohlížeče funguje také jako globální rámec, tedy jsou skrz něj dostupné programátorsky globálně definované funkce a proměnné. Stručný přehled komponentů objektu `window` je zobrazen na obrázku 4.1, jejich význam je následující:

- `document` – kořen HTML dokumentu, tato komponenta bude blíže rozebrána v kapitole 4.2.
- `screen` – obsahuje informace o uživatelské obrazovce.
- `location` – slouží k získání adresy současné webové stránky či přesměrování na novou stránku.
- `navigator` – zpřístupňuje informace o prohlížeči. Zda jsou povoleny Cookies, jeho název či název renderovacího jádra a další.

[1] The Window Object https://www.w3schools.com/jsref/obj_window.asp

- `history` – umožňuje přístup k historii prohlížeče. Vzhledem k zabezpečení soukromí uživatele obsahuje pouze dvě metody: `back()` a `forward()`. Tedy zpět a vpřed.
- `cookie` – umožňuje vytvářet, mazat a editovat uživatelské Cookie.

Všechny výše vypsané komponenty jsou také dostupné pod svým názvem bez prefixu `window`. Kromě komponentů nabízí některé metody a atributy samotný objekt `window`. Za zmínku stojí funkce `alert()`, `console()`, metody pro práce s časovači nebo atributy pro zjištění rozměrů webové stránky a prohlížeče.



Obrázek 4.1: Browser Object Model a komponenty objektu `window`

4.2 Document Object Model

Pod pojmem Document Object Model (dále DOM) si lze představit aplikační rozhraní pro validní HTML a správně strukturovaný XML soubor[2]. Definuje logickou strukturu dokumentu a to, jak je k němu přistupováno včetně modifikace, přidávání a mazání obsahu. Důležitým cílem DOM, jako specifikace W3C, je poskytnutí standardního programovacího rozhraní, které je možno použít napříč prostředími a platformami. Aby bylo možno splnit tento cíl, je specifikace definována v OMG IDL[3], čímž je docíleno možnosti navázání na jakékoli programovací prostředí. Kromě této specifikace dodalo W3C také konkrétní navázání na jazyky ECMAScript a JScript. Jak již název napovídá, jedná se o objektový model v tradičním smyslu objektově orientovaného návrhu. Dokumenty jsou modelovány s užitím objektů a model zahrnuje nejenom strukturu dokumentu, ale také chování dokumentu a objektů, z nichž je složen. Jako objektový model DOM rozpoznává:

- Rozhraní a objekty použité k reprezentaci a manipulaci s dokumentem.
- Význam těchto rozhraní a objektů.
- Vztahy a vazby mezi rozhraními a objekty.

Dokumenty, k nimž je přistupováno pomocí DOM, mají logickou strukturu odpovídající stromu, přesněji Lesu. Každý takový dokument obsahuje nejvýše jeden Doctype uzel, jeden

[2] well-formed https://www.w3.org/TR/REC-xml/#sec-well-formed/
[3] Interface Definition Languagehttps://www.omg.org/spec/IDL/4.0/About-IDL/

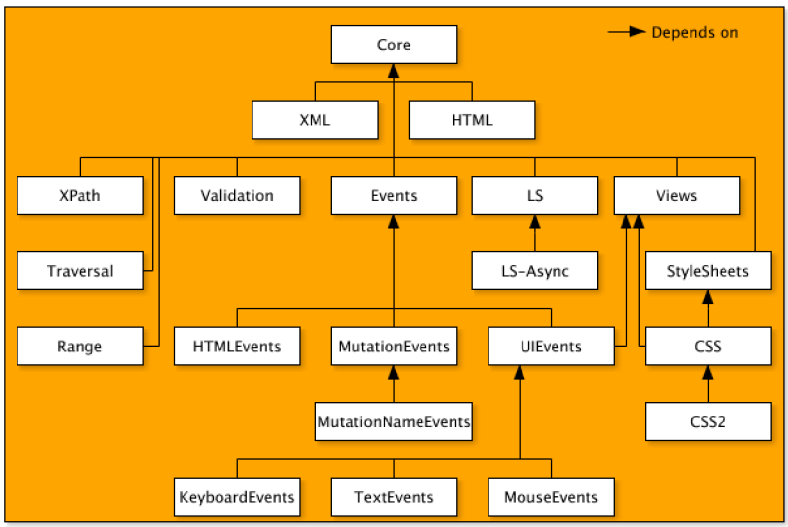
Document uzel a také dobrovolné uzly s komentáři a výpočetními instrukcemi. Uzel Document slouží jako kořen stromu elementů dokumentu. I přes tyto skutečnosti neříká DOM nic o samotné implementaci, jedná se pouze o logický model, jež může být implementován jakýmkoli způsobem, který je příhodný. V konečném důsledku se nemusí jednat ani o strom.

4.2.1 Architektura

Standard DOM je rozdělen mezi množinu různých API, jež ho společně formují. Každá taková specifikace definuje jeden či více modulů a každý modul odpovídá jedné konkrétní funkčnosti, např. specifikace s názvem DOM Core definuje dva moduly:

- Core Module – základní rozhraní, jež musí implementovat každá implementace. Je spojena s funkčností „Core“.
- XML Module – musí být implementována všemi konkrétními implementacemi, jež chtějí splnit XML DOM požadavky. Asociována je s názvem „XML“.

Konkrétní DOM implementace musí implementovat minimálně Core modul. Zda implementuje některé další závisí již na konkrétním použití. Celkový počet modulů (k standardu DOM level 3) je 16, jejich vzájemné vztahy jsou zobrazeny na obrázku 4.2



Obrázek 4.2: Moduly a jejich vztahy. Převzato z: https://www.w3.org/TR/2004/REC-DOM-Level-3-Core-20040407/introduction.html

4.2.2 Vývoj a standardy

Jednotlivé standardy Document object modelu se nazývají úrovně[4]. Začaly vznikat v době pozdních 90. letech , tzv válek prohlížečů[5], které probíhaly mezi prohlížeči Netscape Navigator a Microsoft Internet Explorer, a také mezi skriptovacími jazyky Javascript a JScript.

[4]DOM Levels https://developer.mozilla.org/fr/docs/DOM_Levels
[5]Browser wars https://en.wikipedia.org/wiki/Browser_wars

DOM level 0

Za předchůdce standardů může být považována omezená schopnost skriptovacích jazyků Javascript a JScript detekovat uživatelské události a měnit HTML obsah. Tento historický mezník se stal v roce 1995. Později byl znám jako „DOM level 0“ a nevznikl k němu žádný nezávislý standard.

DOM level 1

První verze standardu skupiny W3C byla vydána v roce 1998. Její specifikace je rozdělena do dvou částí. HTML a Core. Core nabízí nízkoúrovňovou množinu rozhraní, která mohou reprezentovat jakýkoli strukturovaný dokument. Část HTML popisuje rozhraní vyšší úrovně, která jsou použita pro praktičtější pohled na HTML dokument. Rozhraní v DOM1 zahrnují mimo jiné i Document, Attr a Element.

DOM level 2

Uvedena roku 2000. Obsahuje 6 různých specifikací, jež rozšiřují a přidávají funkčnost k částem představených v DOM1. DOM level 2 mimo jiné specifikuje: DOM CSS, DOM2 Core či Dom2HTML.

DOM level 3

Specifikace z roku 2004. Představuje dalších 5 různých specifikací. Dom3 XPath, Dom3 Load and Save apod.

DOM level 4

Převzato roku 2015 z WhatWG Living Standard. Přidává např. `MutuatonObservers` a `MutuatonEvents`. Tato verze, a následující vývoj, je doménou dvou skupin: W3C [14] a WhatWG [13]. Přičemž aktivní vývoj probíhá v DOM Living Standard skupiny WhatWG a skupina W3C vyvíjí snahy převzít tento vývoj, při minimálních změnách, a vytvořit stálý standard. Rozdíly by tedy měly být minimální. Standard W3C je podmnožinou standardu WhatWG. Pouze části, které zatím nejsou implementovány byly odebrány.

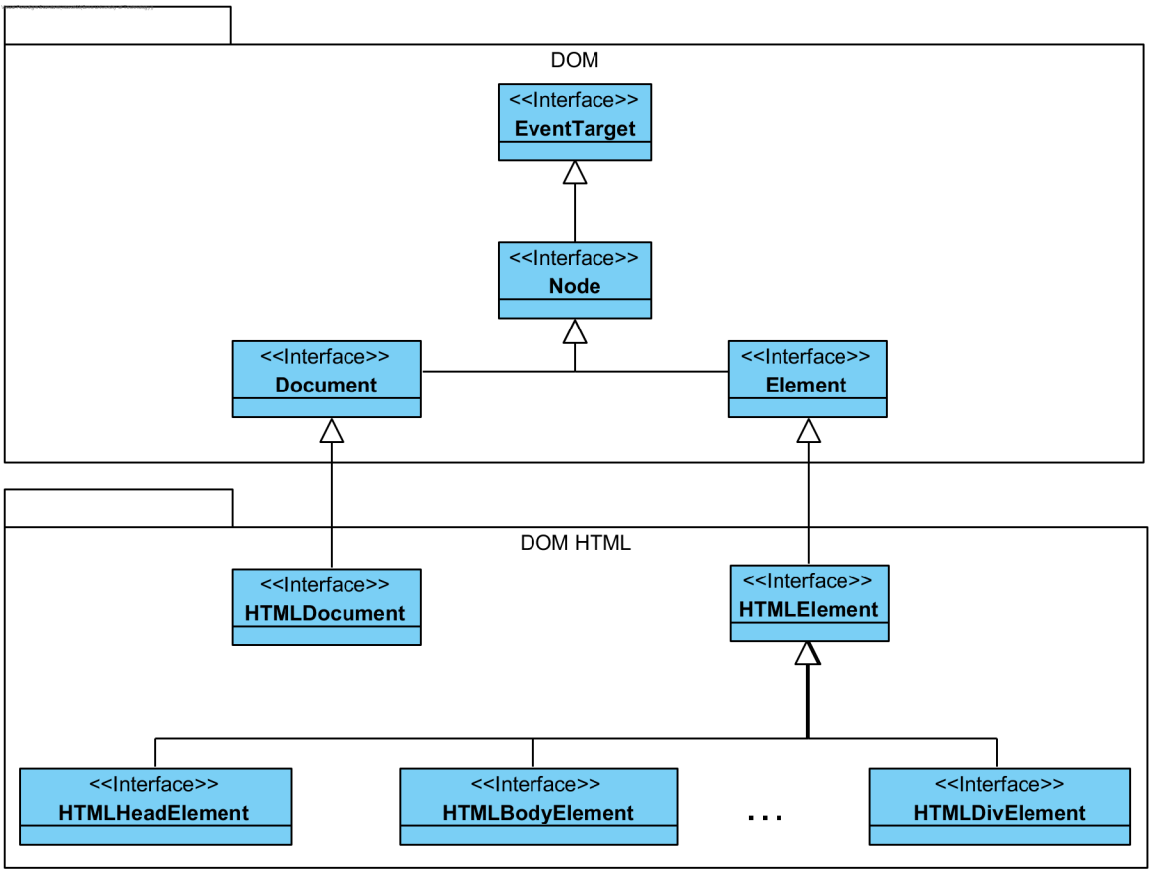
4.2.3 DOM a HTML

V rámci HTML dokumentu je využito rozšíření obecného Document Object Model s název DOM HTML, toto API je popsáno ve standardu HTML[6]. Pomocí odvozených tříd rozšiřuje třídní hierarchii rozhraní DOM, tak jak je naznačeno na obrázku 4.3. Toto rozšíření není nezbytné pro práci s HTML dokumentem, ale umožňuje efektivně pracovat se specifickými vlastnostmi HTML prvků. Každý z těchto typů rozšiřuje schopnosti svého předka, umožňuje přístup k specifickým atributům či akcím:

- `EventTarget`[7] – kořen stromu dědičnosti. Umožňuje cíli možnost reakce na událost.
- `Node`[8] – typ propůjčující základní práci se stromem. Poskytuje metody a atributy pro průchod a úpravu stromu.

[6] HTML DOM https://html.spec.whatwg.org/multipage/dom.html
[7] Event Target https://dom.spec.whatwg.org/#eventtarget
[8] Node https://dom.spec.whatwg.org/#interface-node

- `Element`[9] – základní rozhraní použité pro uzly DOM stromu. Poskytuje vyhledávací metody typu `getElementsByTagName` či atributy id a class.
- `HTMLElement`[10] – bázový typ definovaný v HTML DOM. Obsahuje atributy společné pro všechny HTML elementy, např `title` a vlastnost `hidden`. Jeho potomci jsou již specifické HTML prvky typu `HTMLHeadElement`, `HTMLBodyElement`, `HTMLDivElement` aj.



Obrázek 4.3: Hierarchie rozhraní HTML DOM

Skriptovací jazyk Javascript je s tímto modelem webové stránky provázán skrz globální objekt `document`. Nejvýznačnější atributy tohoto objektu jsou: `documentElement` (představující kořenový prvek `html`), `body` (tělo HTML dokumentu) a také `head`, obsahující hlavičku dokumentu. Nejčastější operace pomocí tohoto objektu v sobě zahrnují: navigaci po DOM stromu, tvoření nových uzlů, navázání zpětných volání na události a úpravu uzlů stromu.

4.3 CSS Object Model

CSSOM definuje aplikační rozhraní pro CSS selektory a Media Queries [19]. Popisuje manipulaci s nimi, tvar a také navázání do dokumentů. V rámci skriptovacích jazyků použitých

[9] Element https://dom.spec.whatwg.org/#interface-element
[10] HTML Element https://html.spec.whatwg.org/multipage/dom.html#htmlelement

na webových stránkách je model např. použit pro dynamickou úpravu vzhledu elementů a pro získání vypočtených stylů elementu.

4.3.1 Atribut style

CSSOM je použit i v kontextu DOM API. Konkrétně se jedná o atribut `style`, jež je dostupný každému objektu typu `HTMLElement`. V tomto atributu, implementující rozhraní `CSSStyleDeclaration`[11], jsou obsaženy informace o stylu objektu. Při přístupu přes `style` atribut je možná pouze změna stylu. Toto omezení je z důvodu, že tento objekt neví nic o CSS kaskádě, a v důsledku ani o vypočteném stylu elementu.

4.3.2 Vypočtený styl

Pro získání vypočteného stylu konkretního element je k dispozici metoda globálního objektu `window` s názvem `getComputedStyle`[12]. Název metody je bohužel z historických důvodů zavádějící. Při vzniku jejího názvu bylo zamýšleno, že získá "Computed", tedy vypočtený styl CSS, což je styl, jaký se získal po aplikaci pravidel kaskády CSS. Taková hodnota stylu může být v relativních hodnotách typu procent. V dnešní době je však žádána a také dodána hodnota "Resolved", což je hodnota spočtena prohlížečem a aplikována na element. Taková hodnota je v jednotném a absolutním tvaru (např. pixel). Stejně jako v případě atributu `style` je i zde použito rozhraní `CSSStyleDeclaration`. Navrácený objekt je však určen pouze ke čtení. Se čtením hodnoty je spojena další vlastnost navráceného objektu. Jeho živost. Pokud je provedena úprava stylů a styl aplikovaný na element byl změněn, projeví se tato změna i v objektu získaném pomocí `getComputedStyle`.

Speciálním případem, kdy nelze použít vypočtené styly, je geometrie a umístění prvku. Konkrétně pro šířku prvku nelze vypočtené styly použít ze dvou hlavních důvodů:

1. CSS vlastnost `box-sizing`[13] může změnit, co je chápáno jako šířka elementu, zda je do ní započteno odsazení a rámeček.

2. Navrácená hodnota nemusí být v absolutní jednotkách, např. řetězec `auto`, jež je, v určitých případech, vrácen pro značku `span`.

Z těchto důvodů je bezpečnější použít jiné vlastnosti dostupné rozhraní `Element`. Například atribut `clientWidth`[14].

[11] CSSStyleDeclaration https://developer.mozilla.org/en-US/docs/Web/API/CSSStyleDeclaration
[12] getComputedStyle https://developer.mozilla.org/en-US/docs/Web/API/Window/getComputedStyle
[13] CSS box-sizing Property https://www.w3schools.com/cssref/css3_pr_box-sizing.asp
[14] Element.clientWidth https://developer.mozilla.org/en-US/docs/Web/API/Element/clientWidth

Kapitola 5

Návrh řešení

V rámci této kapitoly budou rozebrány hlavní cíle a požadavky aplikace. Zároveň budou zmíněna hlavní architektonická rozhodnutí v návrhu nástroje. Cílem kapitoly je vznik modelů a prototypu, jež umožnily vznik aplikace popsané v následujících kapitolách

5.1 Analýza cílů

Nástroj, vznikající v rámci této práce, má za cíl získat informace o webové stránce. Za informace se, v tomto případě, považuje strukturovaný popis vzhledu stránky: geometrie jednotlivých prvků stránky, jejich písma, umístění na stránce, odsazení aj. Do informací naopak nespadá textový obsah prvků a jejich sémantický význam. Aby bylo možno získat požadovaný výstup, je zapotřebí další funkčnost nástroje: automatizaci webového prohlížeče se zaměřením na ovládání webové stránky. Pro splnění cíle musí být aplikace schopna většiny akcí s webovou stránkou, které obvykle provádí uživatel. Tyto akce umožní získat pohledy na webovou stránkou, jež jsou jinak schovány za úkony uživatele. Tato část internetu se nazývá Deep web a může být skryta mimo jiné za přihlášením či vyplněním formuláře. Mezi další, neméně důležité cíle, patří výběr webového prohlížeče, nastavení specifické konfigurace prohlížeče a možnost exportu pouze části stránky. Využití této funkcionality je závislé na možnostech konfiguračního souboru. Ten by proto měl být dostatečně pružný, aby je dokázal pospat. Vzhledem k rychlosti, jakou se webové technologie vyvíjejí, musí být nástroj dále rozšiřitelný a udržovatelný.

5.1.1 Analýza operací

Nástroj by měl být schopný takových akcí, aby bylo možné procházet libovolnou stránku a toto procházení popsat co nejkratším, přehledným konfiguračním souborem. Pro splnění tohoto cíle byla určena množina operací:

- Otevření a ukončení webového prohlížeče.
- Přechod na konkrétní URL.
- Kliknutí na prvek stránky.
- Vyplnění a vymazání textového pole.
- Zvolení z výběrového prvku.

- Odeslání formuláře.
- Vyplnění formuláře pomocí externího souboru.
- Kontrola, zda se prvek nachází na stránce.
- Kontrola, zda prvek obsahuje danou hodnotu.
- Přijmutí dialogových oken.
- Pozastavení chodu ovladače.
- Čekání na splnění podmínky s udáním maximální doby čekání.
- Možnost určité časové tolerance při hledání prvků stránky.

Nadále je nutné říct, že může být množina operací obohacena o další, pokročilejší, akce. Za jednu z pokročilejších akcí by bylo možné považovat vyplnění formuláře pomocí externího souboru. Jedná se o akci, která by šla simulovat posloupností základních příkazů, ale jež byla zavedena z důvodu zjednodušení a zpřehlednění konfiguračního souboru. Externí soubor pro vyplnění formuláře se bude skládat z dvojic klíč–hodnota, přičemž v konfiguračním skriptu dojde k jeho přiřazení ke konkrétnímu formuláři.

5.1.2 Analýza výstupu

Ohledně výstupu aplikace lze vytyčit některé hlavní požadavky. Mezi nejzákladnější patří to, že hodnoty zjištěných atributů o prvku stránky, musí být v absolutních jednotkách. Zároveň, kde je to možné, by měly být jednotky stejného tvaru. Tímto se myslí např. to, že i když jsou oba hexadecimální a RGB zápisy barev absolutního tvaru, výstup by měl používat pouze jeden z nich. Z výstupu by mělo být možné určit pozici prvku v hierarchii stránky.

5.1.3 Analýza konfiguračního souboru

Účel konfiguračního souboru je jasný: musí být schopen plně ovládat vzniklý nástroj. Zároveň by měl být dostatečně obecný, aby skrz něj byla možnost i pokročilých konstrukcí. Z těchto požadavků vyplývají dvě možná řešení:

1. Značkovací jazyk se sekvencí příkazů – u tohoto přístupu převažují spíše negativa: musí vzniknout interpret souboru, výrazové schopnosti vstupu jsou striktně omezeny interpretem a musí vzniknout dokumentace těchto příkazů.

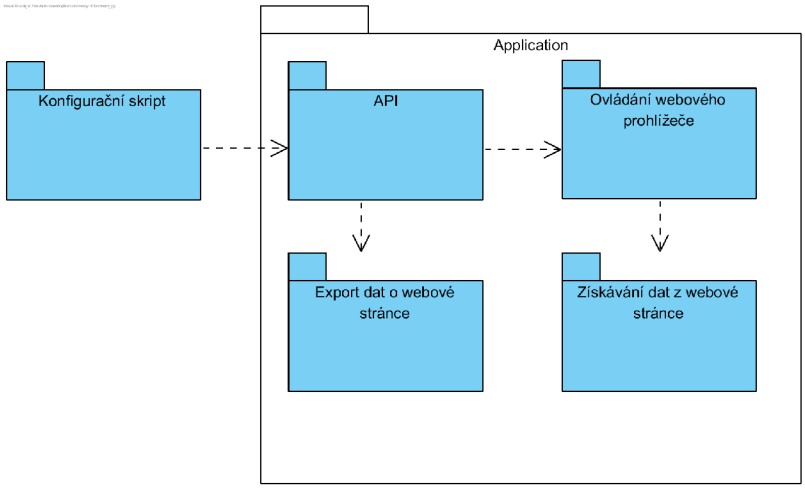
2. Využití aplikačního rozhraní nástroje – odbourává většinu nevýhod předchozího přístupu. Jediná zřejmá nevýhoda je, že aplikace musí dodržet zapouzdření a odhalit pouze metody, které mají být přístupny a signatura těchto metod by měla být neměnná.

5.2 Architektura

Obecně lze říci, že se architektura řešení skládá z bloků, jež plní určitou funkcionalitu aplikace. Jedná se o blok pro ovládání webového prohlížeče, získávání dat z webové stránky,

veřejné rozhraní a export dat o webové stránce (obrázek 5.1). S tím, že by měly mít jednotlivé bloky co nejtenčí vazby, a tedy by měla být možnost výměny některého bloků bez nutnosti změny bloku jiného. Této architektury bylo docíleno s využitím objektově orientovaného návrhu, rozhraní a návrhových vzorů. Použitím těchto zásad bylo dosaženo některých benefitů:

1. K procházení stránek lze využít více prohlížečů.
2. Možnost využít charakteristické vlastnosti prohlížečů (headless mode u prohlížeče Chrome a Firefox, profil uživatele u prohlížeče Chrome apod.).
3. I přes využití konkrétního nástroje k ovládání prohlížeče je možnost záměny bez změny konfiguračních souborů a zbytku aplikace.
4. Dobrá rozšiřitelnost (např. u získávaní dat o webové stránce).
5. Silně univerzální vstupní konfigurační soubor. Možnost cyklů, podmínek apod.



Obrázek 5.1: Hlavní bloky aplikace.

5.2.1 Aplikační rozhraní

Deleguje požadované akce uživatele na dva bloky aplikace: ovládání webového prohlížeče a export dat o webové stránce. I přesto, že konfigurační skript volá metody, jež mají za následek akce s prohlížečem, jedná se pouze o abstrakci v návrhovém vzoru Most. Abstrakce dále deleguje akce s prohlížečem na konkrétní implementaci, která v tomto případě obaluje nástroj Selenium WebDriver. Abstrakce nesmí využívat žádné funkcionality Selenium WebDriver, aby bylo snadné vyměnit implementaci. Při požadavku o export stránky je akce také pouze delegována, v tomto případě na konkrétní třídu implementující rozhraní `IWriter`. Ve výchozím stavu implementuje rozhraní pro export třída, jejíž objekty exportují stránky do formátu XML.

5.2.2 Ovládání webového prohlížeče

Jak již bylo řečeno, jedná se o implementaci v návrhovém vzoru Most. I přes tuto abstrakci komunikuje API pouze s abstraktní třídou, která implementuje metody společné pro

všechny ovladače a nabízí pouze definice metod, tam kde je akce specifická pro každý ovladač. Třídy, jež dědí tuto abstraktní třídu, již odpovídají konkrétním webovým prohlížečům.

5.2.3 Získávání dat z webové stránky

Získávání dat probíhá pomocí jazyka Javascript, jež je vložen a vykonán v kontextu aktuální webové stránky. Vložení kódu a získávání návratových hodnot má na starosti modul Ovládání webového prohlížeče.

5.2.4 Export dat o webové stránce

Export může provézt jakýkoli objekt, jehož třída implementuje potřebné rozhraní. Výchozí export je proveden do souboru ve formátu XML a správnost jeho obsahu je kontrolována oproti šabloně v jazyce XSD.

Pro splnění požadavku na schopnost určit kde se prvek nacházel v rámci hierarchie webové stránky, je XML soubor uspořádán do stromu. Tvar stromu je totožný s tvarem stromu DOM a skoro každý jeho uzel je prvkem stránky.

5.2.5 Konfigurační soubor

Pro zápis konfiguračního vstupu byl zvolen skriptovací jazyk Javascript. A to tím způsobem, že je blízce propojen se samotným nástrojem, jež je v jazyce Java. Může vytvářet veřejné objekty a využívat jejich rozhraní, může volat veřejné třídní metody a využívat výčtových typů. Této funkčnosti bylo dosaženo pomocí javascriptového enginu Nashorn. Díky využití programovacího jazyka jako vstupu bylo oproti značkovacímu jazyku dosaženo několika klíčových výhod:

- Nástroj nemusí obsahovat parser a interpret vstupního souboru. Vstupní skript jednoduše použije veřejné rozhraní.
- Odpadá nutnost dokumentace tvaru konfiguračního souboru, postačí mít řádně dokumentované API.
- Konfigurační skript může využít vlastnosti Turingovské úplnosti programovacího jazyka. Vytvářet funkce, cykly aj.

I přes přínosy lze najít negativa. Jako hlavní negativum lze vytknout nutnost alespoň základní znalosti programování a jazyka Javascript.

Pro pohodlnější využití konfiguračního skriptu je k němu připojen Javascriptový soubor, jež slouží jako inicializační soubor pro stále se opakující konstrukce konfiguračního zápisu, a také pro navázání potřebných typů z jazyka Java. Najdeme v něm např. definici funkcí pro selekci prvků v HTML dokumentu a jejich přetvoření na instance tříd tak, jak to požaduje API Java nástroje (viz algoritmus 5.1). Typy jsou dodány pomocí konstrukce `Java.type`[1]. Získaná hodnota se chová velice obdobně jako třída v Javě. Můžeme vytvářet nové objekty, přistupovat na statické metody, získávat vnořené výčtové typy atd. V enginu Nashorn existuje ještě druhá varianta. V jazyce Java (při vzniku enginu) je možnost využít metody `put`[2]. Metoda `put` slouží primárně pro vkládání již existujících instancí a konstant,

[1] Nashorn API https://docs.oracle.com/javase/8/docs/technotes/guides/scripting/nashorn/api.html

[2] ScriptEngine put https://docs.oracle.com/javase/7/docs/api/javax/script/ScriptEngine.html#put(java.lang.String,%20java.lang.Object)

ale dovoluje vkládat i typy. Bohužel, využití takto vložených typů, má příliš zdlouhavou syntaxi, a ta se již nepodobá klasické práci s typy. Z těchto důvodů je využito provázání pomocí inicializačních skript.

Algoritmus 5.1: Překlad hledání prvku pomocí jeho názvu do tvaru, jež požaduje rozhraní aplikace. Selektor, nechť je třída v jazyce Java.

```
var Selector = Java.type('site.Selector');
/**
 * výběr elementů na stránce pomocí názvu elementu
 * @param selector {string}
 * @return {object} Java Objekt třídy Selector
 */
function name(selector){
    return new Selector(Selector.SelectorType.NAME, selector);
}
```

Příklad použití konfiguračního souboru je ukázán v algoritmu 5.2. Typ `Factory` je dodán v inicializačním skriptu, protože publikuje pouze statické metody, a tedy není jeho vznik třeba parametrizovat.

Algoritmus 5.2: Ukázka použití konfiguračního souboru. Vstup způsobí, že bude exportována stránka výsledku hledání slova "google".

```
driver = Factory.getDriver({
    driver: 'chrome'
});
driver.loadPage("https://www.google.com/");
if (driver.checkIfElementExists(name("q"))){
    driver.inputText(name("q"), "google");
    driver.sendForm(name("f"));
    driver.exportPage("google.xml")
}
driver.close();
```

Kapitola 6

Aplikační rozhraní nástroje

Vzhledem k programové povaze konfiguračního souboru se stává aplikační rozhraní jednou z nejdůležitějších částí nástroje. Je zde nutno nabídnout všechny schopnosti aplikace, ale zároveň skrýt a zabezpečit části, které by mohly ohrozit bezpečnost, anebo vyjevit vnitřní implementaci. Jako součást rozhraní aplikace je možné chápat i různé výčtové typy, které slouží pro omezení chyb při zadávání převážně řetězcových konstant, např. je dodán výčet názvů všech jmen atributů a vlastností prvku. Za část aplikačního rozhraní lze považovat i třídy, jejichž objekty slouží jako argumenty metod aplikačního rozhraní. Identifikace prvků na stránce nebo objekty podmínek explicitního čekání, spadají do této skupiny. Další částí rozhraní mohou být typy, které obsahují statické metody, např. třída `Factory`, která slouží pro získání instancí ovladače. Dále jsou nedílnou součástí důkladné dokumentační komentáře každé publikované metody a třídy. Zde je nutno podotknout, že komentáře budou, při využívání API, sloužit jako hlavní zdroj informací o využívaných metodách. S dokumentačními komentáři úzce souvisí metoda pro identifikaci prvků v DOM stromu. Napříč celým rozhraním je pro identifikaci použito objektů třídy `Selector`. Vzhledem k tomu, že může výběr touto třídou určit více prvků DOM stromu, je v dokumentačních komentářích jasně popsáno chování aplikace v takovém případě. U některých metod nejednoznačný výběr skončí změnou stavu více prvků stránky, či exportu několika větví DOM stromu. Jiná taková volání skončí výjimkou a koncem běhu programu. Pokud se výjimka týká tohoto či jiného případu, který souvisí se špatnými argumenty (např. příliš mnoho nebo naopak žádný identifikovaný prvek či špatný typ značky) je vyvolán jeden z podtypů výjimky `BadArgumentException` z balíčku `site.exceptions`.

Metody zahrnuté do aplikačního rozhraní lze rozdělit do šesti hlavních skupin:

1. Ovládání prohlížeče a stránek – do této skupiny spadají metody pro přechod na stránku, kliknutí na prvek, odeslání formuláře aj. Jejich signatury jsou ukázány v příloze C.1.

2. Manipulace se stavem prvků – lze do ní zařadit zadávání textu či jeho mazání z textových polí, změna stavu `checkbox` atd.

3. Hromadné vyplnění formuláře.

4. Ujišťovaní se o stavu prvku – metody využité, pokud se chce uživatel ujistit, zda daný prvek existuje na stránce či má vstup očekávaný stav.

5. Získání dat o webové stránce – mezi hlavní metody této skupiny patří různé druhy exportu stránky do XML souboru.

6. Čekací operace – do skupiny spadá trojice operací, které určitým způsobem pozastavují provádění konfiguračního skriptu.

6.1 Manipulace se stavem prvků stránky

V této části rozhraní lze najít velké množství metod, jež slouží pro individuální změnu stavu vstupů formuláře. Metod je větší množství ze dvou hlavních důvodů. První spočívá ve speciálním způsobu, kterým se mění stav určitého vstupu. Toto je případ vstupů: `checkbox`, `select` a `radio`. Druhá skupina se skládá ze vstupů, které sice lze zadat jednotným způsobem, ale nebyla by možná žádná kontrola dodaného nového stavu, ani správnosti typu prvku. Tam lze zařadit např. málo využívané typy prvku `input`: `date`, `week`, `color` apod. Zároveň je nutno říci, že každý druh prvku se většinou skládá ze dvou hlavních metod: nastavení stavu a vynulování stavu. Ukázka některých signatur pro změnu stavu prvku je vyobrazen v příloze C.3

6.2 Hromadné vyplnění formuláře

Do této oblasti spadají dvě metody (příloha C.4). První slouží pro vyplnění vstupů vybraného formuláře dodaným slovníkem identifikátorů a požadovaných stavů. Druhá pak pro stejný úkon, nicméně je slovník dodán jako externí soubor ve formátu JSON.

6.3 Ujištění se o stavu

Do této části rozhraní lze zařadit např. ověření existence prvku (příloha C.5). Tento nenápadný úkon může posloužit k ujištění, že se ovladač prohlížeče nachází na očekávané stránce či je přihlášen uživatel. Naproti tomu sem lze zařadit i metody pro porovnání stavu vstupů s očekávaným. Řetězec v textovém vstupu, potvrzený `checkbox` aj. Zvláštností ověření očekávaného stavu vstupu je, že může být pomocí selektoru vybráno více prvků. V takovém případě musí být v očekávaném stavu všechny vybrané prvky.

6.4 Získání dat z webové stránky

Všechny metody z této skupiny mají společné, že je jejich výstupem je XML reprezentace webové stránky. Liší se pouze v tom, zda bude proveden export celého stromu DOM či pouze některých větví. Zároveň je možnost omezit množinu exportovaných dat. Prvního zpřesnění lze docílit voláním metod určeným pro částečný export. Tyto metody mají prefix `partially` a lze s nimi určit nový výchozí kořen stromu nebo dokonce více kořenů čili les. Omezení druhé je zadáváno pomocí seznamu `whitelist`, lze jej zadat u přetížených metod pro export, a to jak pro atributy, tak i vlastnosti prvků. Ve výsledném exportu jsou pak vytisknuty pouze povolené entity. Deklarace části metod pro export je ukázána v příloze C.6.

6.5 Operace pro pozastavení běhu programu

Do této oblasti rozhraní spadají tři operace: `wait`, `setUpImplicitlyWait` a `waitUntil` (příloha C.2). Přičemž první dvě mají velice přímočaré použití. Třetí metoda má složitější

signaturu, ale také lze využít v širší množině případů. Jako svůj první argument přijímá instance tříd implementující rozhraní `IConditionExpected`. Dodaný objekt určuje do splnění jaké podmínky bude program pozastaven. Druhý argument určuje maximální dobu prodlevy. Všechny dostupné podmínky a rozhraní, jež implementují jsou k použití v balíčku `site.conditions`.

Kapitola 7

Ovladač webového prohlížeče

Vzhledem k využití knihovny Selenium WebDriver, není jádro automatizace samotná implementace operací pro přímé ovládání prohlížeče, ale spíše využití sekvence příkazů ovladače dodané knihovnou Selenium pro realizaci těchto operací. Příkazy musí být využívány bezpečným způsobem tak, aby bylo předejito chybám, a v případě nesprávného použití uživatelem byla vyvolána reakce již na straně aplikace. Kapitola popisuje různé možné použití aplikace. Jmenovitě jde o konzolovou aplikaci, která interpretuje konfigurační skript, jež je navázán na API aplikace, a na druhé straně přímé využití API jinou aplikací v jazyce Java. Následně je popsán postup vzniku důležitých objektů při požádání aplikace o nový ovladač webového prohlížeče včetně samotné instancializace ovladače z knihovny Selenium WebDriver. U něhož text vysvětluje způsob, jak ovlivnit parametry vzniku, a také kdy zaniká. Poté již přichází na řadu jádro ovladače, kterým jsou implementace abstraktní třídy `AWebDriver`, poskytující operace k ovládání webového prohlížeče a prostředky pro změnu stavu webové stránky. Změna stavu může nastat vyplněním vstupu, odesláním formuláře, kliknutím na prvek či přechodem na jinou stránku. Poté je věnována pozornost hromadnému vyplnění formuláře stránky a také praktikám, díky nimž bylo dosaženo abstrakce od knihovny Selenium. Zapouzdření knihovny Selenium je posíleno využitím výjimek, spadající do namespace aplikace a objektů třídy Selektor pro identifikaci prvků na stránce.

7.1 Konzolová aplikace

Konzolová aplikace představuje hlavní vstupní bod celého nástroje. Při startu aplikace přes příkazovou řádku dochází pouze k několika základním krokům, po kterých je již interpretován konfigurační soubor. Interpretování předchází nastavení úrovně logování a také vložení pomocných definic v jazyce Javascript do enginu Nashorn, které lze následně využít v rámci konfiguračního souboru. Start aplikace je tedy velice přímočarý proces, při kterém nevznikne žádný zásadní objekt aplikace. Vytvoření objektu reprezentujícího aplikační rozhraní dojde až v rámci konfiguračního souboru. Tímto krokem je zajištěna volnost nastavení nově vznikajícího ovladače, využití více ovladačů, a je také silně omezený počet a složitost argumentů konzolové aplikace. Aplikace přijímá tyto argumenty:

- `-i`, `-input` – povinný argument, reprezentuje cestu ke konfiguračnímu souboru.

- `-l`, `-loging` – určuje úroveň logování. Existuje pět úrovní, které jsou ve vztahu: `trace` < `debug` < `info` < `warning` < `error`[1]. Při vynechání argumentu je nastavena úroveň `info`. Tedy jsou zaznamenány pouze záznamy `info`, `warning`, `error`.
- `-s`, `-schema` – udává umístění souboru se schématem výstupu. Pro základní implementaci se jedná o dodaný XSD soubor. Pokud není argument zadán, je výchozí hodnota: `http://www.stud.fit.vutbr.cz/~xbastl03/outputSchema.xsd`.

7.1.1 Interpretace konfiguračního souboru

Pro interpretaci konfiguračního skriptu je využit Javascriptový engine Nashorn. Již v průběhu vzniku práce byl Nashorn označen jako deprecated [16] a v příštích verzích Javy je plánováno jeho odstranění, ale GraalVM[2], jakožto přijatelná náhrada, nebyl ještě zcela připraven a např. nefungoval na platformě Windows. Proto, i přes jeho nepřenositelnost na novější verze Javy, byl zvolen Nashorn. Před samotným konfiguračním souborem jsou v Nashorn přeloženy dva pomocné soubory: `inputInitial.js` a `AttrParDefinition.js`. Inicializační soubor vkládá do kontextu enginu funkce, které slouží pro snadnější využití třídy `Selector`, a také tříd pro podmíněné čekání. V budoucnu může posloužit např. pro vznik výchozího ovladače, který se bude moci bez dalších příkazů využít v konfiguračním souboru. Druhý soubor obsahuje výčtové typy pro usnadnění použití seznamů `whitelist`.

7.2 Vznik ovladače

Při vzniku ovladače musí brát aplikace ohled na některá hlediska. Ovladač musí jít vytvořit jak z jazyka Java, tak z konfiguračního souboru, přičemž musí být brán zřetel na styl programování v různých jazycích[3]. Ovladačů může být větší počet, a to i s různými prohlížeče. Uživatel nesmí poznat konkrétní knihovnu na ovládání prohlížeče a konkrétní implementace musí být lehce vyměnitelná. Z těchto důvodu vzniklo několik vrstev abstrakce a proces vzniku ovladače není triviální záležitost.

7.2.1 Možnosti vzniku ovladače

Vznik ovladače ovlivňují implementace třídy `IDriverOptions` (viz rozhraní 7.2), přednostně každý z podtypů ovlivňuje, jaký prohlížeč bude ovládán. Pro zdědění rozhraní je navíc nutné implementovat metody pro určení cesty k ovladači a nastavení headless verze prohlížeče. Dále se již rozsah možností liší podle konkrétní implementace. Ovladač webového prohlížeče Chrome nabízí chod s uživatelským profilem nebo ponechání spuštěného prohlížeče po ukončení ovladače. Firefox pouze chod v headless režimu a HtmlUnit nerozšiřuje základní implementaci, naopak zakazuje volání metod pro headless mód a nastavení cesty. Žádná z těchto rozšiřujících vlastností není povinná a každá z nich má určitou výchozí hodnotu, kterou lze najít jako statické členy tříd implementujících rozhraní `IDriverOptions`, např. ovladač prohlížeče Chrome je ve výchozím nastavení hledán v pracovní složce pod název `chromedriver.exe`, nepracuje v headless módu, nezůstane otevřen a není navázán na uživatelský profil.

Vzhledem k počtu argumentů startu prohlížečů [3] byl dodán další mechanismus. Každá implementace nabízí pole argumentů, do kterého se mohou vkládat textové řetězce. Při

[1] logging https://tinylog.org/logging
[2] GraalVM https://www.graalvm.org/
[3] Programming style https://en.wikipedia.org/wiki/Programming_style

spouštění jsou pak prvky pole zadány jako argumenty startu ovladače. Tento mechanismus již nemá další kontroly správnosti zadaných argumentů a správnost jeho využití je zcela v rukou uživatele.

Start ovladače s uživatelským profilem

Možnost přístupná pouze ovladači Chrome a to pouze za určitých podmínek. Hlavní z nich je, že může existovat pouze jediná instance s tímto profilem [26]. Do tohoto omezení spadá i uživatelsky spuštěný Chrome. Druhá podmínka je, že prohlížeč bude spuštěn bez rozšíření.

7.2.2 Tvorba z jazyka Java

V případě jazyka Java lze přeskočit jednu vrstvu abstrakce a přímo vytvořit instanci třídy `WebControl`, která slouží jako aplikační rozhraní. Konstruktor této třídy vyžaduje jeden z objektů tříd, implementující rozhraní `IDriverOptions`. Typ objektu určuje, jaký prohlížeč bude ovládán, zároveň je mu možné dodat parametry, jež budou uplatněny při startu prohlížeče, např. zda má pracovat v headless režimu. Objekt `WebControl` je zároveň abstrakce v návrhovém vzoru Most a své akce pouze deleguje na konkrétní implementaci. O implementaci se stará potomek abstraktní třídy `AWebDriver`, jehož má instance `WebControl` jako svou komponentu. Konkrétní objekt získá voláním statické metody třídy `AWebDriver` `newInstance`. Tato tovární metoda, přijímající jako svůj argument `IDriverOptions`, rozhodne, z jaké z trojice tříd `ChromeWeb`, `FirefoxWeb` či `HTMLUnitWeb` vznikne objekt a ten navrátí. Všechny akce API budou delegovány právě na tento objekt. Vybraný objekt si zároveň jako svoji komponentu vytvoří a uschová objekt pro ovládání konkrétního prohlížeče z knihovny Selenium WebDriver. Diagram tříd popisující výše načrtnuté vztahy obsahuje příloha B.3.

Algoritmus 7.1: Vznik ovladače z jazyka Java. Postup je využit např. v rámci jednotkových testů.

```
ChromeDriverOptions options = new ChromeDriverOptions();
options.setHeadless(true);

WebControl driver = new WebControl(options);
```

7.2.3 Tvorba z konfiguračního souboru

V zájmu zachování stylu programování v jazyce Javascript není objekt `WebControl` v konfiguračním souboru vytvářen přímo. Místo toho aplikace nabízí třídu `Factory` se svou statickou metodou `getDriver`, která jako svůj argument přijímá slovník klíč–hodnota. V těle metody následně vznikne ze slovníku konkrétní instance třídy implementující rozhraní `IDriverOptions`. Tento objekt je naplněn a následně dodán jako argument konstruktoru třídy `WebControl`, instance třídy je následně vrácena konfiguračnímu souboru. Příloha A.3 obsahuje sekvenční diagram postupu vzniku ovladače. Nutno dodat, že při vyplňování `IDriverOptions` nástroj kontroluje, zda jsou všechny klíče ze slovníku platné, a pokud ne, dojde k upozornění uživatele. Příklad syntaxe vzniku ovladače z konfiguračního souboru je zobrazen v algoritmu 5.2.

Jak již bylo řečeno metoda `getDriver` ze třídy `Factory` přijímá slovník, kdy množina klíčů je omezena a jejich hodnota nese význam v rámci nastavení ovladače. Mezi tyto řetězce patří:

- `driver` – klíč, jehož hodnota určuje, jaký prohlížeč bude ovládán. Přípustné hodnoty jsou: `chrome`, `firefox` a `htmlunit`. Jediná povinná položka slovníku.
- `headless` – nese pravdivostní hodnotu, zda bude prohlížeč pracovat v headless módu. Platné pouze pro Chrome a Firefox.
- `location` – cesta k ovladači. Chrome a Firefox implementují tuto funkcionalitu.
- `stay-open` – zda prohlížeč zůstane otevřen po dokončení interpretace konfiguračního souboru. Implementuje ovladač Chrome.
- `user-profile` – start s uživatelským profilem. Pouze Chrome.
- `other-options` – hodnota musí být pole řetězců, kde každý představuje argument příkazové řádky, který bude využit při spouštění ovladače.

Algoritmus 7.2: Rozhraní, jehož potomci slouží pro dodání parametrů ovladače.

```
public interface IDriverOptions {

    WebControl.WebDriverType getType();

    void setDriverLocation(String path) throws FileNotFoundException;
    String getDriverLocation();

    boolean isHeadless();
    void setHeadless(boolean headless);

    void addOtherOption(String otherOption);
    List<String> getOtherOptions();
}
```

7.3 Uzavření Selenium ovladače

Uzavření ovladače, a tím pádem i webového prohlížeče, může nastat dvěma způsoby. Přímé volání metody `close` aplikačního rozhraní, nebo skončení interpretace konfiguračního souboru. Důležitým faktem je také skutečnost, že po uzavření ovladače již nelze volat žádnou z jeho ostatních metod. Při snaze o volání metody po uzavření ovladače dojde k vyvolání výjimky.

Při konci interpretace konfiguračního souboru provede třída `Main` snahu o uzavření všech otevřených ovladačů. Seznam vzniklých objektů třídy `WebControl` získá díky spolupráci s třídou `Factory`, která řídí vznik každého objektu `WebControl` v konfiguračním skriptu, a tím pádem je schopna na ně uschovat referenci. Pokud si uživatel přeje zachovat okno prohlížeče i po uzavření ovladače, musí tak říct při jeho vzniku skrz rozhraní `IDriverOptions`.

Při přímém využití objektů třídy `WebControl` není možnost hlídat vznikající instance, a tudíž nedochází k automatickému uzavírání.

7.4 Operace pro ovládání prohlížeče

Vzhledem k přímočaré povaze je většina operací implementována jako pouhé delegování akce na Selenium WebDriver. Odlišnosti lpí v použití třídy `Selector` pro identifikaci prvků stránky, předřazení kontroly prvků pro akce, které lze provézt pouze s jediným prvkem a kontrola typu prvku. Další kontrola musí být provedena u operace pro kliknutí na prvek stránky. Pokud je prohlížeč spuštěn ve své headless verzi, nesmí se využít funkce z knihovny Selenium. Pro tuto metodu je takový prvek neviditelný a dojde k vyvolání výjimky. Namísto toho je proveden klik pomocí příkazu v jazyce Javascript. Sekvenční diagram pro operaci načtení webové stránky je ukázán v příloze A.1.

7.5 Operace pro čekání

Možnost čekat na splnění určité podmínky je při ovládání moderních webových stránek nezbytnou schopností. Webové stránky hojně využívají AJAX, provádějí transformace DOM stromu pomocí skriptů či generují časově závislé akce. Všechny tyto akce trvají určitou dobu a ovládání stránky bez prodlev by vedlo k akcím nad prvky, které buď neexistují, ještě nejsou připojeny k DOM stromu, nebo provádějí transformaci. Na první pohled se může zdát, že by byl dostatečný jediný druh čekání. Pozastavení běhu skriptu na předem danou dobu. Příkaz pozastavení by se vkládal mezi jednotlivé akce se stránkou a nedocházelo by k výše popsaným problémům. Takový přístup sebou nese určité obtíže, přičemž některé jsou zcela kritické a nelze je pomocí předem přesně vymezeného čekání vyřešit. Předně by bylo nutné zjišťovat optimální prodlevy mezi příkazy. Již toto hledisko činí z pouze jednoho druhu čekání nepřípustnou variantu. Pouze jeden druh čekání by také vedl na zbytečně dlouhé prodlevy, protože musí být počítáno s proměnlivou latencí webové stránky. Nedeterministická latence stránky, zbytečně dlouhé prodlevy a obtížný vývoj skriptů jsou důvody k zavedení třech skupin operací pro čekací:

1. Bezpodmínkové čekání – operace, která zastaví chod aplikace na určité časové kvantum. Tento druh čekání není podporován ze strany knihovny Selenium WebDriver, ale existují situace ve kterých neexistuje žádná jiná alternativa. Pro implementaci čekání využívá nástroj knihovní funkci `Thread.sleep`. Funkce `sleep` může být bezpečně využita z důvodu, že nástroj není paralelní a nevyužívá sdílené zdroje [15].

2. Implicitní čekání – udává prodlevu po kterou ovladač čeká než ohlásí neúspěch při hledání prvku stránky. Pokud dojde do vypršení této doby k připojení hledaného prvku ke stránce, pokračuje nástroj v činnosti, v opačném případě je vyvolána odpovídající výjimka. Jakmile je čekání nastaveno, platí pro každý další příkaz aplikace. Funkčnost v totožné podobě nabízí knihovna Selenium [20], a tedy byla využita v nezměněné podobě.

3. Explicitní čekání – ovladač čeká dokud není splněna jasně definovaná podmínka. Čekání je shora omezené pomocí časovače, jež po vypršení vyvolá výjimku. Podmínky se, kromě výjimečných případů, váží na stav prvků stránky. Z toho např. vyplývá, že může být pozastaven běh skriptu dokud není prvek viditelný, nebo dokud stránka neobsahuje dostatečný počet elementů definovaných selektorem.

7.5.1 Explicitní čekání

První dvě oblasti čekání jsou díky své minimální konfiguraci velice přímočaré, a to jak při použití v konfiguračním skriptu, tak vzhledem k implementaci. I v rámci třetí skupiny je implementace, vzhledem k podpoře ze strany knihovny Selenium WebDriver, pouhé delegování. Na rozdíl od ostatních skupin však mají tyto operace velice různorodou konfiguraci a k této skutečnosti musí být přihlédnuto při návrhu použití v konfiguračním skriptu, a také při snaze o zachování abstrakce od knihovny Selenium.

Všechna explicitní čekání jsou delegována na funkci `waitUntil` objektu `WebDriverWait`[4], kterému je při vzniku řečena maximální doba, po kterou může čekat na splnění podmínky. Podmínky čekání jsou udány pomocí statických metod třídy `ExpectedConditions`[5], které přijímají určitý počet parametrů pro definici podmínky. Pro ověření viditelnosti je např. zapotřebí jediný argument – selektor pro určení prvku na webové stránce. Ověření počtu prvků má již argumenty dva: selektor a požadovaný počet prvků. V rámci nástroje tedy vyvstala potřeba správně a vhodně reprezentovat volání statických metod. Toho bylo docíleno hierarchií tříd v balíčku `site.conditions`, s počátkem v rozhraní `IConditionExpected`. Každá implementace rozhraní reprezentuje jednu ze statických metod, argumenty metody jsou uchovány v parametrech objektu a jsou zadány konstruktorem. Vzhledem k tomu, že je tento balíček přístupný z rozhraní aplikace, nesmí být přímo vázaný na knihovnu Selenium WebDriver a samotné volání funkce `waitUntil` musí být provedeno mimo tyto třídy. Tento problém byl vyřešen návrhovým vzorem Návštěvník, kdy rozhraní `IConditionExpected` zadává povinnost implementace metody `accept`, přijímající objekt třídy `ConditionVisitor`. Metoda `accept` má jediný úkol, zavolat metodu `visit` objektu `ConditionVisitor`, která je přetížená pro každý typ implementace rozhraní a která již ve svém těle vykoná čekání. Oproti návrhovému vzoru došlo k odebrání rozhraní na straně návštěvníka. Konkrétní implementace existuje pouze jedna, rozhraní je tedy zbytečné, a to i v případě náhrady knihovny Selenium. Obrázek 7.1 zobrazuje zjednodušený třídní diagram pro tento návrhový vzor. Operace pro čekání na zviditelnění prvku je zobrazena v sekvenčním diagramu A.2.

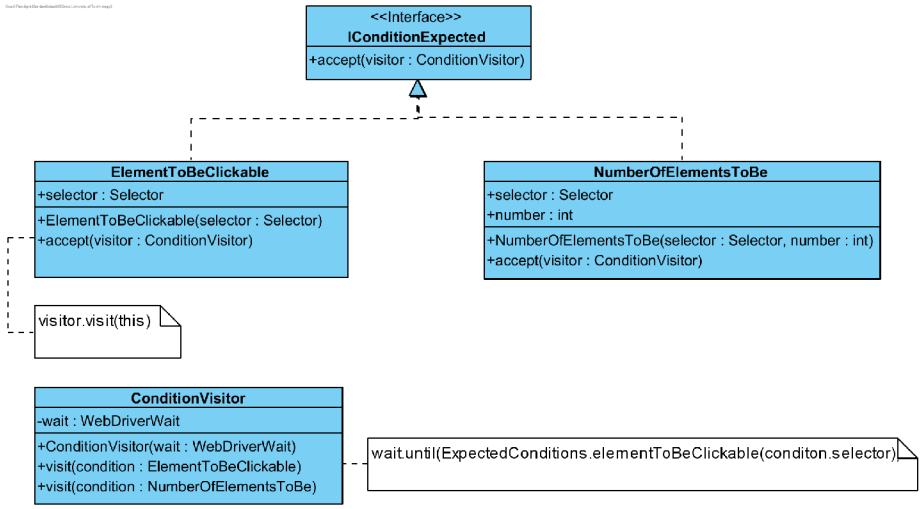
Použití v konfiguračních souborech (algoritmus 7.3) je skrz metodu rozhraní `waitUntil`. Tato metoda má vždy dva parametry: definici podmínky a `timeout`. Definice podmínky je obdobná jako v případě výběru prvků stránky, tedy pomocí funkce, která obaluje vytváření objektů rozhraní `IConditionExpected`. Tímto je zajištěn programovací styl bližší jazyku Javascript a rovněž možnost proměnlivého počtu parametrů podmínek.

7.6 Operace s prvky pro vstup

I zde dochází k přesměrování operací na knihovnu Selenium WebDriver. Jejich vztah již není tak přímočarý jako u operací s prohlížečem. Jedna akce může být přeložena na více různých metod knihovny a to buď v případě, že bylo na stránce identifikováno více prvků, které splňují selekci, nebo akci nelze provézt pouze přímým voláním Selenium. První případ zahrnuje větší množství operací, kde má, v případě nejednoznačného výběru, smysl použít akci na všechny prvky. Jako příklad mohou posloužit akce: zadání vstupu do textového pole, zjištění, zda je zatrhlý `checkbox` či výběr možnosti v prvku `select`. Akce záměny

[4] WebDriverWait https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/support/ui/WebDriverWait.html

[5] ExpectedConditions https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/support/ui/ExpectedConditions.html



Obrázek 7.1: Návrhový vzor Návštěvník využitý pro reprezentaci podmínek čekání.

obsahu vstupu `textarea` může posloužit za příklad netriviálního volání knihovny Selenium WebDriver, využívající posloupnosti tří volání: zaslání klávesy TAB, vyčištění obsahu a zaslání obsahu.

Valná většina operací využije jako svůj první příkaz jednu ze dvou funkcí:

- `listCheck(Selector selector)` – využijí operace, kterým nezáleží, zda jsou provedeny s více než jedním prvek. Metoda je mimo jiné schopna vyvolat výjimku `NoSuchElementException`, pokud nebyl nalezen ani jeden prvek v dokumentu a varovat v případě výběru více prvků. Zároveň navrací seznam objektů `WebElement`. Nad těmito objekty, patřící do balíčku Selenium WebDriver, je vykonána požadovaná operace.

- `oneAndOnlyOne` – přetížený název metod. Definice s osamoceným argumentem přijímá objekt třídy `Selector`. Přetížená definice přidává ještě druhý argument typu `WebElement`, sloužící jako kontext, ve kterém bude provedeno hledání prvku. Metoda bez kontextu určí jako svůj kontext celý dokument a dále již se s implementací spolehne na kontextovou metodu. Hlavní rozdíl od akce `listCheck` spočívá v zaměření operací, které ji využijí. A to operace, které lze provézt pouze s jediným elementem. V Těle metody tedy může kromě výjimky `NoSuchElementException` vzniknout, při nalezení více než jednoho prvku v kontextu určeným argumentem, výjimka `TooMuchElementsException`. V případě, že není nutná žádná z výjimek, metoda vrátí jediný objekt `WebElement`.

Operace lze dle účelu rozdělit do tří hlavních skupin:

1. Nastavení stavu elementu – slouží pro manipulaci se stavem vstupů formuláře. Před změnou je zkontrolováno, zda se pracuje s korektním typem prvku. Za příklad může posloužit metoda pro změnu textového vstupu, skupina operací jejichž stav musí být změněn pomocí příkazů Javascriptu nebo výběr prvku `radiogroup`.

2. Ujištění o stavu prvku – metody začínající prefixem `check`. Ověření existence elementu, porovnání hodnoty textového vstupu oproti řetězci, je vybrána očekávaná možnost v `select`, nebo je zaškrtnutý `checkbox`.

Algoritmus 7.3: Ukázka definice podmínky čekání a její použití v konfiguračním souboru. Pokud nelze na prvek stránky s id `clickable` kliknout, ovladač počká maximálně 5s.

```
var ElementToBeClickable = Java.type('site.conditions.ElementToBeClickable');

/**
 * Podmínka explicitního čekání. Podmínka je splněna, pokud lze na prvek stránky
 ↪  s~daným selektorem kliknout.
 * Tedy je přítomný v~DOM stromu, je viditelný a nemá nulové rozměry.
 * @param selector výběr pvku stránky
 * @returns {object} Java Objekt třídy ElementToBeClickable
 */
function elementToBeClickable(selector){
    return new ElementToBeClickable(selector);
}

//Použití v~konfiguračím souboru
var clickableSel = css("#clickable");
driver.waitUntil(elementToBeClickable(clickableSel), 5);
driver.click(clickableSel);
```

3. Uvedení do výchozího stavu – vymazání textového pole či černá barva ve vstupu `color`.

7.7 Hromadné vyplnění formuláře

Před samotným řešením je nutné nastínit hlavní požadavky pro hromadné zadání formuláře. Samotný zápis pro identifikace prvků formuláře a požadovaného stavu těchto prvků musí být co nejkompaktnější. Zároveň však nesmí být nejednoznačný a být dostatečně univerzální pro identifikace jakéhokoli prvku formuláře a uvedení ho do všech možných stavů. Identifikace prvků musí probíhat v kontextu požadovaného formuláře, nikoli na celé stránce. Při snaze vybrat neexistující prvek nebo při zadání nevalidní hodnoty je jako další hledisko požadována zpětná vazba.

V zájmu kompaktnosti se jako řešení nabízí slovník dvojic, kde klíč referuje na atribut prvku s názvem `name`, jež musí být přítomný u všech vstupů. Tato forma zápisu tedy má schopnost určit jakýkoli prvek formuláře. Tímto nejjednodušším zápisem však není splněn požadavek na jednoznačnost. I když je atribut `name` v rámci jednoho formuláře ve valné většině případů unikátní, nemusí tomu tak být, např. u prvku `checkbox`, který často tvoří skupiny, je sdílena stejná hodnota atributu `name`. Proto musel být identifikátor rozšířen speciální syntaxí. Ta umožňuje zadat řetězec, který se skládá ze dvou částí oddělených dvojtečkou. První část stále popisuje požadovanou hodnotu atributu `name`, druhá část značí, jaké hodnoty musí vybraný prvek nabývat v atributu `value`. Pokud toto zpřesnění není potřeba, nemusí být využito. Sdílení atributu `name` a rozšíření identifikátoru je vidět na výčtu 7.1.

Stav, jakého má vstup nabýt určuje druhý prvek dvojice. Zde není striktně omezen jeho typ, protože se odvíjí od typu prvku. Pro textový vstup se jedná o řetězec, pro `checkbox` o pravdivostní hodnotu atd. To, zda je typ hodnoty přípustný, rozhoduje aplikace až při vyplňování, při kterém proběhne rezoluce typu vstupu. Postup, při kterém nástroj uvede prvek formuláře do požadovaného stavu, se odvíjí od jeho druhu. Textovému vstupu aplikace

Výčet 7.1 Nejednoznačnost atributu `name`. Pro její řešení musí být použito speciální syntaxe, např. pro výběr písmena "b"se jedná o: `alphabet[]:b`

```
<form>
    <label><input type="checkbox" name="alphabet[]" value="a"/>a</label>
    <label><input type="checkbox" name="alphabet[]" value="b"/>b</label>
    <label><input type="checkbox" name="alphabet[]" value="c"/>c</label>
</form>
```

jednoduše smaže obsah a posléze mu nahraje nový, `checkbox` obslouží simulací kliknutí. Některé prvky je nutno vyplnit pomocí příkazů v jazyce Javascript a pro `select` existuje sekvence příkazů knihovny Selenium WebDriver.

Jak již bylo předesláno, další důležitý požadavek představuje určení kontextu prvků. Tohoto již není docíleno v rámci slovníku hodnot, ale pomocí API. Metody pro vyplnění formuláře mají argument, v němž je povinnost identifikovat konkrétní formulář na stránce. Metody zároveň přijímají data pro vyplnění formuláře, a to ve dvou formách. První způsob přijímá přímo slovník hodnot, druhý vyžaduje řetězec, který popisuje cestu k souboru obsahující data ve formátu JSON. Této funkcionality je docíleno přetížením metody s názvem `fillForm`, jejíž signatury jsou ukázány v příloze C.4

Pomocí metod k vyplnění formuláře nedochází k automatickému odeslání. Toto musí být explicitně řečeno dalším příkazem z API, `sendForm`. Další omezení tkví v odlišném chování při identifikaci prvků stránky, každý selektor ve slovníku musí striktně vybrat jeden prvek formuláře. Je to tedy odlišné chování od změny stavů prvků pomocí příslušných metod, kdy je, v určitých případech, dovolen několikanásobný výběr.

7.8 Zpětná vazba

V rámci nástroje je, ve výchozím nastavení, zcela výlučně využita negativní zpětná vazba. Aplikace uživatele tedy upozorní pouze na podezřelé a nesmyslné akce. Upozornění na podezřelé akce je prováděno pomocí informativních zpráv na konzoly. V takovém případě nástroj dále interpretuje konfigurační soubor. V druhém případě již nemá smysl příkaz provézt a vyvolá se výjimka. Uživatel se v takovém případě musí rozhodnout, zda se bude snažit z výjimky zotavit (ať už v konfiguračním souboru, či při přímém použití API) nebo chod aplikace skončí.

7.8.1 Informační zprávy

Jak již bylo řečeno, ve výchozím nastavení jsou informační zprávy použity v případě, kdy si nástroj umí s danou situací poradit, ale existuje zde nejistota, zda byla akce zamýšlena. Informační zpráva je mimo jiné vygenerována při vyplnění textového vstupu, kdy je vybrán větší počet prvků stránky. Taková situace sice má řešení – vyplnění všech nalezených vstupů – ale nemusela být zamýšlena. Uživatel mohl neúmyslně použít málo konkrétní selektor. Jako odezva je pouze vypsána informační zpráva popisující s kolika elementy byla daná akce vykonána.

Pro generování zpráv nástroj využívá knihovnu `tinylog`[6], která umožňuje několik stupňů důležitosti zpráv. V rámci aplikace je využito pět stupňů zpráv. `Debug` a `Trace` pro ladění

[6] tinylog https://tinylog.org/

aplikace. `Info`, které zaznamenává obecný průchod programem. `Warn`, jež značí potenciálně chybné akce a `Error`, pro chyby použití aplikace. Ve výchozím nastavení se zobrazují pouze zprávy `Info`, `Warn` a `Error`. Zobrazení zpráv `Debug` a `Trace`, či naopak skrytí zbylých tří, lze nastavit při spouštění aplikace využitím argumentů aplikace.

7.8.2 Výjimky

Vyvolání výjimek slouží pro situace, kdy další provádění akce postrádá smysl a mohlo by dojít k neočekávaným důsledkům. Jako příklad lze uvézt snahu o kliknutí na více prvků či zadání hodnoty do prvku, který není vstupem. Valné množství výjimek použitých v nástroji je odvozeno od předka `IllegalArgumentException`. A to ze dvou důvodů. Za prvé má tato výjimka za předka `RuntimeException`, který pochází z rodiny výjimek `unchecked`. Dle článku *Unchecked Exceptions — The Controversy* [18] byla tato rodina zvolena z několika důvodů. Výjimky vyvolané nástrojem vznikají nesprávným použitím a uživatel se z těchto situací bude velice těžko zotavovat. Druhý důvod spočívá v situacích, kdy jsou výjimky vyvolány – špatné vstupní argumenty metod, tedy `IllegalArgumentException`. V zájmu začlenění do API, jsou konkrétní výjimky odvozeny od třídy `BadArgumentException`. Ta je sice abstraktní a nelze být vyvolána, ale dovoluje jednotné zachycení vzniklých výjimek a také nahrazení konkrétních výjimek v signatuře metod, kde by byla signatura již příliš dlouhá. Hierarchie tříd výjimek je vyobrazena v příloze B.1

7.9 Identifikace prvků

Pro dosažení nezávislosti na využité knihovně k ovládání prohlížeče vznikla třída `Selector`, sloužící na identifikaci prvků na stránce. Konstruktor této třídy má dva argumenty. První určuje metodu, podle které se bude prvek identifikovat. Identifikátor metody náleží typu `SelectorType`, výčtového typu, patřící třídě `Selector` (viz příloha B.2). Druhý parametr je již řetězec pro identifikaci.

Objekty třídy jsou využívány napříč celým aplikačním rozhraním, a také, k dosažení vyšší abstrakce, u veřejných metod třídy `AWebDriver`. Překlad na interní reprezentaci, jež používá Selenium WebDriver probíhá až v metodách `AWebDriver`, a to pomocí soukromé metody `translateSelectorToBy`, která vrací instance podtypů abstraktní třídy `By`. Objekty tříd odvozených od `By` již patří do knihovny Selenium WebDriver.

Pro využití, které se více blíží stylu programování v jazyce Javascript, jsou v konfiguračních souborech dodány funkce vytvářející objekty třídy `Selector` (algoritmus 5.1).

Kapitola 8

Získání a export dat z webové stránce

Data lze v kontextu nástroje rozdělit do dvou odlišných skupin. Předně se jedná o textový popis grafické podoby prvku webové stránky čili HTML značky. Úplnost, správnost a jednotnost informace o grafické podobě prvku, lze řadit mezi hlavní cíle aplikace, a tudíž na něj byl kladen velký důraz při implementaci a ověření korektnosti chování implementace. Vznik standardů v oblasti programového zjištění vypočtených stylů ulehčil získání a další práci s touto informací, ale i přesto budou popsány metody a problémy v získávání grafické podoby prvku.

Druhým typem jsou atributy prvku. Na tento druh dat, vzhledem k zaměření práce, již byl kladen menší důraz, ale i přesto bude postup získání uveden.

V poslední části bude probrán průběh průchodu stromu DOM, zpracování jednoho uzlu, a také vnitřní reprezentace uzlu. Následný text se zaměří na poslední aspekt získávání informací, export do externího souboru a jeho strukturu, která musí zaručit, aby nedošlo ke ztrátě informací o umístění prvku v ekvivalentním HTML souboru.

8.1 Vlastnosti prvku

Vlastností prvku jsou v tomto kontextu myšleny informace, které lze programově získat o daném prvku. Přednostně sem patří informace graficky popisující element stránky. To znamená: velikost obdélníku, odsazení, použité písmo, barva pozadí atd. Z dosavadního výčtu lze vypozorovat, že se jedná o informace nastavované skrz styly v jazyce CSS. Z tohoto faktu vychází i programové získání většiny vlastností prvků, které probíhá skrz objekt `Computed styles` v jazyce Javascript. V rámci nástroje, k usnadnění syntaxe pro získání vypočtených stylů, je využito knihovny JQuery a její metody `css`. Pro získání všech vlastností o prvku je definován slovník klíč–hodnota, kde klíč značí název vlastnosti a hodnota určuje funkci, která se zavolá pro její získání. Následně je slovník iterován a, při respektování pole `whitelist`, získána každá vlastnost prvku.

Většina funkcí pro získání informace se skládá z pouhého zavolání funkce `css` a předáním návratové hodnoty. Některé informace tímto stylem získat nelze či mají nevhodný tvar, poté musí vzniknout unikátní rezoluce hodnoty pro danou vlastnost. V takovém případě vynikne často zdlouhavý a nadbytečný způsob volání funkcí, které pouze zavolají metodu `css`. Při zjištění nevhodnosti navrácené hodnoty dojde k úpravám pouze konkrétní funkce. A do hlavní iterace nebude zasaženo. Tento přístup se ověřil i v pozdějších fázích práce, kdy

byly, kvůli odstranění redundance, vynechány vlastnosti, které nabyly výchozí hodnoty. Tedy muselo být zasaženo skoro do každé funkce pro získání vlastnosti.

8.1.1 Získání vlastního textu

Vlastním textem se myslí text, který je přímo přítomný v dané značce. Není tedy zanořen v žádné značce dceřinné, ale může např. následovat po definici této dceřinné značky, nebo mezi dvěma potomky. Text prvku sice nepatří mezi grafické vlastnosti prvku, ale je jeho důležitou a nedílnou součástí.

Jedná se o jednu z mála vlastností, která není získána pomocí skriptu v jazyce Javascript, místo toho je, skrz knihovnu Selenium WebDriver, získán atribut `innerHTML`. Ten následně dodá aplikace knihovně Jsoup[1] a metodou `ownText` získá výsledný text.

8.1.2 Získání použité rodiny písma

Určení fontu, s nímž webový prohlížeč vykreslil konkrétní text, je jedna z požadovaných charakteristik, u které již nestačí využít vypočtené styly. Ty nám poskytnou pouze obecnou představu ve tvaru seznamu rodin písem, ze které prohlížeč vybíral. Přičemž v některých případech je využit i font mimo tento seznam.

Logika uplatnění písma

Informace, s níž dále operuje webový prohlížeč, je typicky získána z kaskádových stylů. Preferovaná písma jsou zadána jako seznam názvů[2]. Příklad zadání takové informace je zobrazen ve výčtu 8.1. Priorita písem klesá směrem doprava. Prohlížeč se tedy prvně snaží aplikovat začátek seznamu a s každým nezdařeným pokusem klesá doprava na méně preferované fonty. Seznam je doporučováno ukončit generickým názvem sad písem. Generické názvy pouze specifikují druh písma (např. patkové či bezpatkové) a výběr konkrétního fontu je již v moci webového prohlížeče. Font využitý v tomto případě se proto může lišit napříč webovými agenty, platformami, a dokonce i na stejné platformě. Poslední příklad nastane v případě, kdy není prohlížečem preferovaný generický font dostupný a musí využít jiný, méně vhodný. Každý webový agent je povinen dodat alespoň jeden vhodný font na každý z generických názvů.

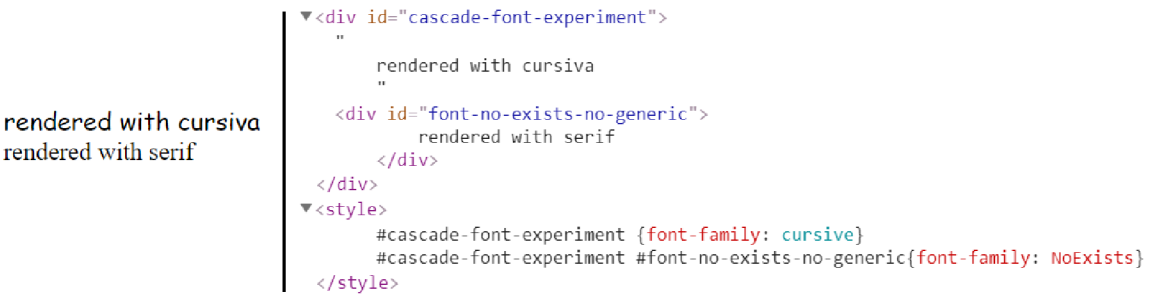
Výčet 8.1 Typický příklad přiřazení rodin písma elementu.

```
div {font-family: BebasNeueBold, ClarikaGeometry, cursive}
```

Pokud není možné uplatnit žádnou hodnotu ze seznamu, je rozhodnutí, jaké písmo využít na prohlížeči. Nutno poznamenat, že v takovém případě již nejsou zohledněna pravidla kaskády a dědičnosti, jak je zobrazeno v experimentu 8.1.

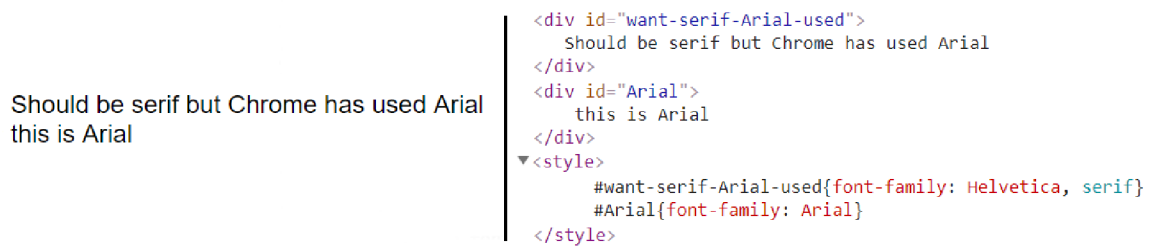
[1] Jsoup https://jsoup.org/apidocs/org/jsoup/Jsoup.html
[2] propdef-font-family https://www.w3.org/TR/CSS2/fonts.html#propdef-font-family

Experiment 8.1 Ignorování kaskády. Vlevo vykreslený text. Vpravo odpovídající HTML kód. Preferovaný font zanořeného prvku neexistuje, místo aby prohlížeč použil generické písmo: `cursive`, rozhodne o použitém písmu sám. Experiment uskutečněn v Chrome verze 72.



Další případ, kdy prohlížeč ne zcela respektuje standard, je ukázán v experimentu 8.2, kdy v případě nepřítomného písma `Helvetica` nevyužije následující písmo v seznamu, ale využije velice obdobné písmo `Arial`.

Experiment 8.2 Ignorování priorit. Vlevo vykreslený text. Vpravo odpovídající HTML kód. Preferovaný font `Helvetica` neexistuje, místo aby prohlížeč použil generické písmo `serif`, použije písmo `Arial`. Experiment uskutečněn v Chrome verze 72.



Při programovém zjišťování vykresleného písma musí být přihlédnuto ke všem těmto skutečnostem a k nestandardnímu chování prohlížečů.

Programové zjištění rodiny písma

Jak již bylo řečeno, z vypočtených stylů se pouze dozvíme seznam kandidátů na vykreslené písmo. Proces, při kterém dojde ke zjištění použitého písma, je možno rozložit do dvou hlavních kroků: zjištění, zda písmo existuje a následné ujištění, že písmo bylo použito. Přičemž oba kroky využívají rozhodovací proces prohlížeče pro výběr vykresleného písma, který vybere vhodný font z dodaného seznamu. Hlavní myšlenka tedy spočívá v průchodu seznamu kandidátů – dodaného z vypočtených stylů – od nejvíce preferovaných písem a kontrole, zda platí zmíněná pravidla. První písmo, které splní obě dvě podmínky, je písmo požadované.

Pro svůj chod využívá algoritmus pláten. Na ty jsou vždy vykresleny řetězce písem s tím, že všechny plátna vykreslují totožný text a jsou jim dodány stejné stylové informace. V čem se tyto plátna liší, je jaký seznam písem je mu dodán.

První podmínka, existence písma, je ověřena pomocí dvou pláten. Jednomu z nich je dodán seznam písem obsahující kontrolovaný font a za něj připojený font generický, patkový. Druhému plátnu je dodán seznam písem v němž, kromě zjišťovaného fontu, figuruje font bezpatkový. Následně je obsah pláten porovnán pixel po pixelu. Můžeme říct, že pokud mají obě plátna totožný obsah, font existuje. Tato jistota pramení z úvahy, že pokud písmo neexistuje, prohlížeč využije méně preferované písmo v seznamu. První plátno vykreslí pomocí patkového písma a na druhé použije písmo bezpatkové. Tyto písma nikdy nejsou totožná.

Ujištění, zda bylo písmo použito při vykreslení je obdobné. Použijeme třetí plátno a tomu zadáme seznam písem získaný z vypočtených stylů, přičemž prohlížeč sám rozhodne, jaké písmo ze seznamu použít. Následně toto plátno porovnáme s libovolným z existujících pláten. Pokud se rovnají, nalezli jsme hledaný font.

Pro upřesnění použitého fontu je do algoritmu přidán ještě následující mechanizmus: v případě, že nebyl nalezen žádný vhodný font nebo byl vrácen generický název, je vyvinuta snaha určit písmo ze seznamu známých, bezpečných a písem použitých prohlížeči v případě generického názvu.

8.1.3 Absolutní jednotky získaných dat

Po získaných informací o webové stránce je požadováno, aby byly jejich hodnoty v jednotném tvaru. Navíc musí být tento tvar v absolutních jednotkách. Jak již bylo řečeno v kapitole 4.3.2 objekt s aplikovanými styly toto pravidlo ve většině případů splňuje. Valná většina hodnot délky je dodána v pixelech a barvy jsou v tvaru RGB složek, obdobně je to pro jiné veličiny. I z tohoto pravidla však existují výjimky. Například u hodnot délky jsou ojedinělé případy kdy, pokud je hodnota zadána v relativní jednotce procenta, objekt s aplikovanými styly dodá hodnotu v této jednotce. Tento rys sdílí vlastnosti: `text-indent`[3], `border-radius`[4] či `background-size`[5] a `background-position`[6]. Pro každou z nich musí být vytvořen speciální postup převodu na absolutní jednotku. Před samotným převodem musí být zmíněny tři rozdílné rozměry elementu:

- `content-box` – rozměr prvku bez rámečku a vnitřního odsazení.
- `padding-box` – `content-box`, ke kterému je zahrnuto vnitřní odsazení.
- `border-box` – `padding-box` s rozměry rámečku.

Také je nutno zmínit vlastnost obdélníku `box-sizing`[7]. Ta určuje, jakou roli hrají vlastnosti `width` a `height` ve výsledných rozměrech elementu. Pokud je její hodnota řetězec `border-box`, velikost obdélníku je roven těmto vlastnostem. V případě řetězce `content-box` je však situace jiná. `Width` a `height` pouze definují velikost obsahu. Výsledné rozměry dále určuje vnitřní odsazení a rámeček. K `box-sizing` tedy musí být přihlédnuto při určování třech výše zmíněných rozměrech prvku. A to z důvodu, že jsou rozměry dodané objektem vypočtených stylů pouze převedené na absolutní jednotky a `box-sizing` není brán v úvahu. Výsledné velikosti obdélníku odpovídají pouze v případě, že `box-sizing` je nastaven na `border-box`.

[3] text-indent https://developer.mozilla.org/en-US/docs/Web/CSS/text-indent
[4] border-radius https://developer.mozilla.org/en-US/docs/Web/CSS/border-radius
[5] background-size https://developer.mozilla.org/en-US/docs/Web/CSS/background-size
[6] background-position https://developer.mozilla.org/en-US/docs/Web/CSS/background-position
[7] box-sizing https://developer.mozilla.org/en-US/docs/Web/CSS/box-sizing

text-indent

Výpočet absolutní hodnoty odsazení textu probíhá z šířky obdélníku. Šířka je zde vždy chápána jako `content-box`. Výpočet je tedy velice přímočarý, jedinou zvláštností může být možnost záporných hodnot odsazení textu.

border-radius

Nejprve je nutné si představit logiku za zakřivením rámečku obdélníku. Každé zakřivení rohu je určeno elipsou, která přesně určuje tvar. Z této premisy vyplývá, že každý roh musí být určen dvěma hodnotami. V rámci zjednodušení lze zakřivení rohu určit pouze jednou hodnotou. V takovém případě je druhá hodnota totožná a elipsa se degraduje na kruh. V rámci kaskádových stylů má `border-radius` mnoho možných syntaxí zápisu[8]. S využitím vypočtených stylů dojde k oproštění od konkrétních zápisů a pro každý roh je vrácen řetězec jedné či dvou hodnot oddělených mezerou. Jak již bylo řečeno, v případě jedné hodnoty je druhá totožná. V případě, kdy jsou jednotky hodnoty procenta, musí dojít k přepočtu na pixely. První rozměr elipsy je vztažen k `border-box` šířce obdélníku. Druhá hodnota je relativní k `border-box` výšce prvku.

background-size

V případě velikosti obrázku na pozadí je nejnáročnější úkon určit k jaké oblasti bude obrázek relativní. Velikost takového obdélníku ovlivňuje hned několik vlastností. Předně `background-attachment`[9]. V momentě, kdy tato vlastnost obsahuje řetězec `fixed` je obdélník úměrný ploše zobrazované prohlížečem. Pokud hodnota není řetězec `fixed`, zasahuje vlastnost `background-origin`[10]. Ta může nabývat hodnot: `padding-box`, `content-box` a `border-box`. Přičemž význam je stejný, jako je nastíněn ve výčtu 8.1.3. Do výpočtu navíc zasahuje vlastnost `background-origin`, určující, kde bude počátek souřadnic obrázku, a tedy i to, k jaké ploše bude relativní jeho velikost. Samotná velikost obrázku se skládá z jedné až dvou hodnot, které popořadě určují šířku a výšku pozadí. V případě dodání pouze jedné hodnoty, nabývá druhá řetězce `auto`. Po získání velikosti boxu je převod již velice přímočarý. První hodnota je relativní k jeho šířce a druhá k výšce boxu.

Nutno dodat, že každý prvek může mít hned několik obrázků na pozadí, počet těchto pozadí určuje výčet ve vlastnosti `background-image`[11] a případné chybějící hodnoty v ostatních vlastnostech jsou doplněny opakováním.

background-position

Souřadnice počátku obrázku na pozadí se určují hned z několika veličin. Předně mezi ně patří již dříve představené `background-origin` a `background-attachment`, k těmto vlastnostem dále přibyla hrana, k níž je vzdálenost vztažena. Na horizontální ose je možnost souřadnici spřáhnout k levé či pravé hraně. Obdobně vertikální vzdálenost je možno určit k horní nebo dolní hraně. Nutno říci, že při neurčení je využita levá a horní hrana. Všechny tyto veličiny musí být zahrnuty do výpočtu absolutních hodnot, přičemž vypočtené styly dodají jednu z těchto tří variant:

[8] W3 border-radius https://www.w3.org/TR/css-backgrounds-3/#the-border-radius

[9] background-attachment https://developer.mozilla.org/en-US/docs/Web/CSS/background-attachment

[10] background-origin https://developer.mozilla.org/en-US/docs/Web/CSS/background-origin

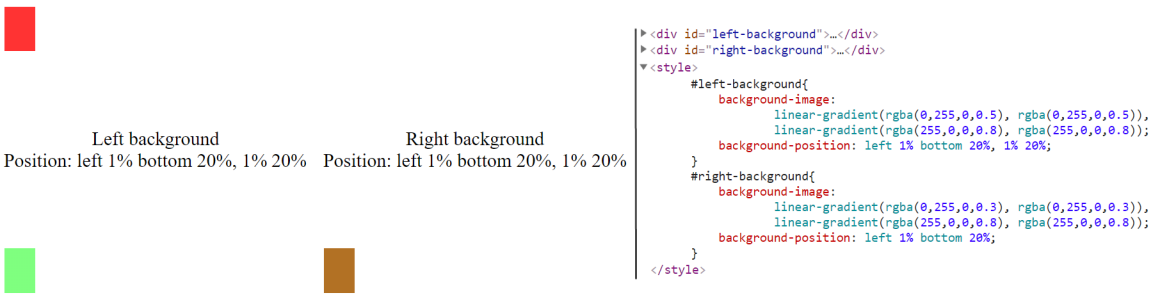
[11] background-image https://developer.mozilla.org/en-US/docs/Web/CSS/background-image

- Dvě hodnoty – každá hodnota určuje vzdálenost od výchozí hrany.
- Tři hodnoty – jedna z hodnot má určenou hranu, druhá využije výchozí.
- Čtyři hodnoty – každá z hodnot je spřažena s explicitně řečenou hranou.

V rámci nástroje je vyžadováno, aby byla vždy navrácena možnost první. Tento požadavek přidává potřebu výpočtu i při pozici, která je již určená pixely. I v tomto případě totiž mohou být pixely vztaženy k jiné, než výchozí, hraně.

Převod pozice obrázku na pozadí na absolutní hodnoty v sobě zahrnuje chybu, kterou nelze odstranit. Není totiž v algoritmu, ale v dodaných vypočtených stylech, jež jsou v určitém případě nesprávné. Situace nastane za splnění několika předpokladů: element má více obrázků na pozadí, explicitně zadaných pozic je méně než počet obrázků, a je využito nevýchozích hran. Při splnění výše uvedených podmínek prohlížeč obrázek správně umístí, ale dodá chybný vypočtený styl. Tento styl je nesprávný v tom, že u chybějících deklarací pozic, jež byly správně doplněny opakováním, již není určena hrana, ke které jsou uvedené hodnoty spjaty. Situace je nastíněna na experimentu 8.3

Experiment 8.3 Chyba vypočtených stylů. Vlevo dva prvky a každý z nich obsahuje dva obdélníky na pozadí. Oba mají totožný vypočtený styl pro vlastnost `background-position`, ale pouze obdélníky vpravo se překrývají. Oddíl vpravo ukazuje HTML kód. Experiment uskutečněn v Chrome verze 72.



Ztráta informací

Získání jednotných absolutních hodnot je zcela určitě přínos pro jakýkoli automatický nástroj či uživatelský skript, který s nimi pracuje. Ale i tento přístup sebou nese drobné ústupky. Jedním z nich může být drobná ztráta přesnosti, a tím pádem informace, např. v případě, kdy je barva zadána v HSL reprezentaci, dojde při jejím mapování na prostor RGB ke ztrátě přesnosti. `HSL(22, 85%, 57%)` je namapováno na `RBG(239, 120, 57)`, ale zpětným převodem již nevyjde původní odstín, nýbrž hodnota `HSL(21, 85%, 58%)`. Tento drobný rozdíl odstínu je lidským okem nepostřehnutelný, nicméně automatické nástroje s ním mohou mít problémy.

8.2 Atributy prvku

Atributy značí informace zapsané přímo v hlavičce značky. V HTML dokumentu hrají význačnou úlohu a mohou posloužit pro identifikaci prvku v rámci dokumentu, specifikaci informace, uvedení prvku do nějaké stavu či získání role v rámci HTML dokumentu. Funkci

identifikace plní zejména atributy `class` a `id`. Za příklad získání role v rámci HTML dokumentu může posloužit atribut `name`, který vstup přiřadí nadřazenému formuláři. Pro změnu stavu lze využít atribut `checked`. Specifikace informací patří mezi nejrozsáhlejší skupinu atributů. Za příklad lze uvést atribut `action`, určující akci při odeslání formuláře nebo skupinu atributu s prefixem `data-`, které slouží výlučně skriptovacím jazykům a webový prohlížeč je zcela ignoruje. Jiné rozdělení atributů značek může být na atributy, které se mohou vyskytovat u každé značky a atribut použitelný pouze u skupiny značek. První skupina nese pojmenování Globálních atributů a patří do ní např. atributy: `class`, `id` a `data-`.

Získání atributů prvku nástroj provádí pomocí JQuery funkce `attr` a to tím způsobem, že je definováno pole všech možných názvů atributů a u každého je zjištěno, zda prvek atribut obsahuje, pokud ano, hodnota atributu je navrácena.

Jediné výjimky tvoří atributy:

- `checked` – u prvku `checkbox` nahrazuje hodnotu atributu `value`. Zjištěn pomocí funkce JQuery `prop`, která vrátí pravdivostní hodnotu.
- `class` – návratová hodnota funkce `attr` (řetězec názvů tříd) je dále zpracována.
- Atributy s prefixem `data-` – získané iterací návratové hodnoty JQuery funkce `data`. Nutno poznamenat, že je získána pouze první úroveň a zanořené `data-` atributy jsou ignorovány.

Při získávání musí být zároveň respektován dodaný seznam `whitelist`.

8.3 Průchod DOM stromem

Počátek průchodu DOM stromem je vybrán pomocí objektu třídy `Selector`. Vzhledem ke své povaze musí aplikace brát zřetel na situaci, že objekt může identifikovat více než jeden prvek DOM stromu. Proto samotnému průchodu předchází metoda, která jej provede pro každou vybranou větev. Samotný průchod jedné větve odpovídá již klasickému `Preorder` rekurzivnímu průchodu stromem. Metoda vytvoří každému uzlu jeho reprezentaci `WebsiteRepresantion` a následně jsou pro uzel nalezeni jeho potomci. Na každého potomka přistoupí, a provede kroky:

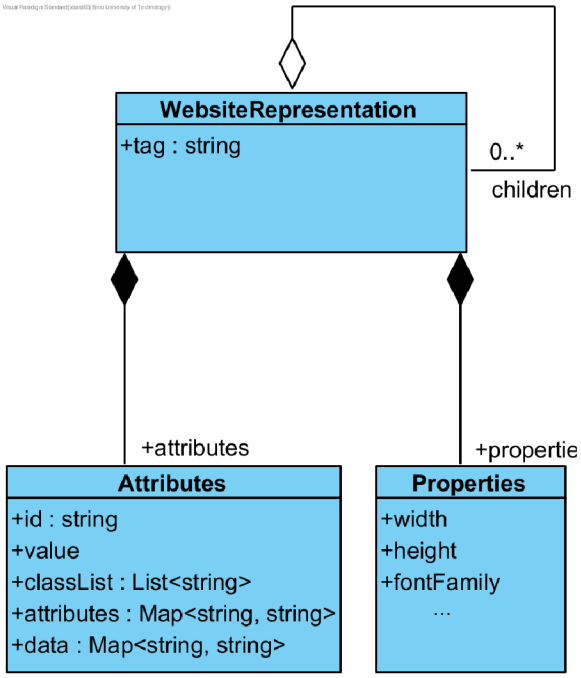
1. Zjištění, zda je potomek viditelný. Pokud ne, dále není pokračováno.
2. Zavolána rekurzivní funkce pro potomka.
3. Metoda připojí potomkovu reprezentaci k rodiči.

Těmito třemi kroky je navrácen seznam stromů objektů `WebsiteRepresentation` čili les. Každý objekt již v sobě uchovává zjištěná data o prvku stránky a může být použit pro výstup.

8.4 Tvorba objektu `WebsiteRepresantion`

Objekty třídy `WebsiteRepresantion` slouží jako vnitřní reprezentace uzlů DOM stromu. O tvorbu a naplnění instancí se stará soukromá metoda `createWebRepresentation`. Ta nejprve vytvoří nový prázdný objekt `WebsiteRepresantion` a naplní jeho atributy:

- `tag` – řetězcová informace uschovávající název značky. Její hodnota je získána pomocí útržku jazyka Javascript.
- `properties` – obsahuje objekt třídy `Properties`, který shromažďuje zjištěné vlastnosti o elementu stránky. Objekt `Properties` získá metoda `getElementProperties`, jež nejprve zavolá metodu v jazyce Javascript, která byla dříve vložena na stránku. Návratovou hodnotu následně konvertuje na objekt `Properties` pomocí knihovny `jackson-databind`[12].
- `attributes` – uschovává instanci třídy `Attributes`, sloužící pro uschování získaných informací o atributech prvku stránky. Malé množství atributů je přímo prvkem instance a to z důvodu, že jsou použity pro informační zprávy. Objekt je získán pomocí metody `getElementAttributes` a to automaticky z návratové hodnoty Javascript skriptu.



Obrázek 8.1: Třída jejichž objekty s komponentami slouží pro vnitřní reprezentaci DOM uzlů.

8.5 Vložení kódu Javascript do webové stránky

Aplikace pro svůj chod vyžaduje vložení určitého množství kódu Javascript do navštívené webové stránky. V průběhu pouhého procházení stránek a změny stavů vstupů je vkládáno minimální množství kódu Javascript, např. pro vyplnění jiného, než textového vstupu či zjištění typu značky `input`. K vkládání většího množství kódu dochází až při žádosti o získání informací o prvku stránky. V takovém případě jsou v předem daném pořadí nahrané soubory sloužící pro získání informací o prvcích stránky a různé pomocné knihovny

[12] jackson-databind, https://github.com/FasterXML/jackson-databind

a funkce. S přihlédnutím k podpoře enginu HtmlUnit, odpovídá úroveň jazyka Javascript standardu ECMAScript 5.1 s určitými odchylkami.

Všechen kód nástroj vkládá pomocí objektu `JavascriptExecutor` z knihovny Selenium WebDriver a jeho metody `executeScript`, která na aktuálně navštívenou stránku vloží řetězec v jazyce Javascript a vykoná ho v rámci anonymní funkce. Důležitý detail je, že mezi jednotlivým voláním funkce nejsou zachovány žádné lokální proměnné, a tedy ani definice funkcí. Mít všechen potřebný kód v jednom souboru je však nepraktické, navíc by pro získání informací o každém prvku muselo dojít k opakovanému vložení kódu. Proto jsou všechny vkládané funkce a definice připojeny do objektu `document.webdriver`. Využití objektu `webdriver` dojde k minimalizaci šance přepsání již definovaných funkcí, které mohou být k objektu `document` připojeny.

Při vkládání větších úseků kódů, než jsou krátké útržky, přestává být řetězec přípustnou a přehlednou metodou. Větší úseky kódu jsou proto vkládány pomocí externích souborů. Celý soubor aplikace načte a následně využitím metody `executeScript` vloží na právě navštívenou stránku. Nejprve musí zajistit přítomnost knihovny `JQuery`, využívané pro lehčí získání vypočtených stylů. Ujištění spočívá v algoritmu, který zjistí, zda je již přítomná na stránce. Pokud ne, nahraje ji z lokálního souboru. Poté nahraje zbytek skriptů, které slouží pro získání atributů a jiných informací o prvcích stránky.

8.6 Export dat o webové stránce

O export dat se starají implementace rozhraní `IWriter` (viz rozhraní 8.1). Takové třídy musí implementovat dvě metody: `createRepresentation` a `outputToFile`. První metoda slouží pro vytvoření cílové reprezentace ze vstupu, který představuje objekt `WebsiteRepresantion`. Důležitá vlastnost vytvořených objektů je, že si při provádění exportu drží vnitřní stav. Implementace `XMLWriter` při každém volání metody `createRepresentation` připojí další XML větev k dokumentu XML. A až při volání `outputToFile` dojde k vypsání do souboru a znovu inicializaci stavu. Proto musí být při každém exportu dodržena posloupnost volání metod nebo vždy vytvořen nový objekt rozhraní `IWriter`.

Jak již bylo předesláno, v rámci nástroje je implementován export do XML souboru pomocí třídy `XMLWriter`. Objekty této třídy využívají pro svoji činnost knihovny `jdom2`[13]. `XMLWriter` obsahuje celou řadu soukromých pomocných metod, starajících se o vznik různých podob XML značek:

- `createSimpleElement` – vytvoří jednoduchou značku. Jejíž název odpovídá názvu vlastnosti a obsah značky tvoří hodnota vlastnosti, např. vlastní text prvku.
- `valueAsAttribute` – přidá hodnotu jako atribut značky dodané v parametrech. Používá se pro jednoduché vlastnosti a atributy HTML prvku.
- `createElementFromDict` – metoda, které musí být dodán slovník klíč–hodnota. Jako návratová hodnota vznikne nová značka s názvem vlastnosti, která ve svých atributech sdružuje dvojice ze slovníku. Dekorace textu vznikne touto metodou.
- `createDictElementsFromList` – stejné jako předchozí případ, ale sdružujících značek je větší množství, např. stíny a rámečky.
- `createDataRecords` – metoda používaná v případě zpracování atributu data. Vytvoří značku `data-` a každou dvojici název–hodnota vyplní jako atribut značky.

[13] jdom, http://www.jdom.org/

Algoritmus 8.1: Rozhraní, jehož potomci slouží export dat.

```
/**
 * Rozhraní pro jednotný vznik a výpis webové reprezentace.
 */
public interface IWriter {
    /**
     * Připojení jedné reprezentace z~vnitřní webové reprezentace.
     * @param rootNode kořenový uzel.
     */
    void createRepresentation(WebsiteRepresentation rootNode);

    /**
     * Výpis načtených dat do souboru.
     * @param path cesta k~souboru.
     * @throws IOException Výjimka, pokud cesta neexistuje.
     */
    void outputToFile(String path) throws IOException;
}
```

- `createRootTag` – bezparametrová metoda, vytvářející kořenovou značku. Metoda v sobě zahrnuje nastavení cesty k XSD šabloně pro kontrolu korektnosti výsledného dokumentu a výchozí namespace, což je `masterThesis-namespace`. Výchozí namespace slouží ke kontrole zanoření HTML značek a definici možných atributů. Zároveň metoda definuje sekundární namespace `complexProperties`, do něhož spadají informace, které nemohou být v atributech HTML značky. Značky z namespace `complexProperties` využívají prefix `x` a jejich správnost je kontrolována oproti jiné XSD šabloně.

Zpracování jednoho kořenového objektu `WebsiteRepresantion` využívá právě výše zmíněných metod. Strom objektů je rekurzivně procházen a komponenty aktuálního uzlu jsou přetvořeny na XML reprezentaci. Prvek `tag` poslouží za název nově vznikajícího XML elementu. Komponenty objektu `Properties` (popisující získaná data o prvku stránky) jsou, pomocí metod pro transformaci, vyplněny do XML dokumentu. Ty, které je možno popsat řetězcem jsou zadány jako atributy XML elementu. Složitější mají vlastní značku s atributy obsahujícími získaná data. Značka je následně připojena k XML elementu a je rozlišena odlišným namespace. Posléze jsou zpracovány hodnoty z objektu `Attributes`, které reprezentují získané atributy HTML prvku. Krom výjimek jsou zadány jako atributy hlavního prvku. Výjimku tvoří např. atribut `data-`, který je připojen jako nová značka. Vzniklý XML element je posléze připojen k elementu odpovídajícímu rodiči v stromu objektů `WebsiteRepresantion`. Po rekurzivním průchodu připojí aplikace kořenový XML element ke kořenu XML dokumentu.

Kromě exportu dat do externího souboru nabízí aplikace metody pro získání konkrétní informace pro prvek stránky. Lze tedy získat libovolný argument nebo vlastnost pro prvek identifikovaný pomocí objektu třídy `Selector`. V návratové hodnotě těchto metod je nutné od sebe odlišit tři případy:

1. Je požadována neexistující informace o prvku – v takovém případě metody vrací hodnotu `null`.
2. Navrácená hodnota by nabyla výchozí hodnoty pro danou informaci – metody vrátí prázdný řetězec, prázdný slovník či prázdné pole. Přesný typ je určen dle komplexnosti informace.
3. Hodnota byla získána – v tomto případě je vrácena hodnota informace.

Kapitola 9

Testování

Ověření korektnosti chování představuje důležitou součást životního cyklu aplikace. S přihlédnutím k faktu, že aplikace pracuje s webovými stránkami, tato potřeba pouze sílí. Webové stránky jsou velice heterogenní, dosahují různé úrovně složitosti a prodělávají velice rychlý vývoj, kdy standardy vznikají až poté, co je technologie implementována v rámci webových prohlížečů. Odlišné webové prohlížeče navíc určité informace zpracovávají nejednotným způsobem a občas nepodporují standardy v celém rozsahu. Dalším specifikem aplikace, a tudíž testování, je podpora hned tří webových agentů, jež navíc v určitých případech dosahují rozdílných výsledků ve svém headless módu. V této situaci se manuální testování stává příliš pracné a potřeba regresivních testů tuto skutečnost pouze posiluje. Proto hlavní část testů sestává z automatizovaných testů a pouze jedno odvětví testů musí být manuální. Testování lze rozdělit na dvě skupiny: automatické jednotkové testy a manuální testy v reálném prostředí.

9.1 Jednotkové testy

Pro posílení důvěry ve správné a očekávané chování aplikace vzniklo velké množství jednotkových testů, které pokrývají hlavní aspekty aplikace pro každý z ovladačů. V případě testování konfiguračních skriptů je použito existujících souborů v jazyce Javascript ve složce `test/java/resources`. Další okruhy testování vyžadují webové stránky, přičemž je nevhodné využít pro jednotkové testování reálné webové stránky. Ty jsou velice proměnlivé a jednotkové testy by po určité době ztrácely smysl. Proto byl vytvořen korpus stránek, které jsou zaměřeny na jednotlivé aspekty testování. Obsah každé stránky je minimální pro otestování dané funkcionality. Tyto webové stránky musí být nasazeny na lokální webový server s adresou `http://localhost/diplomka`. Soubor stránek lze získat ve složce `test/resources/web`. Valná část jednotkových testů je navázána přímo na aplikační rozhraní, ale existují i skupiny testů využívající konfigurační skripty, např. kontrola interpretace skriptů, spouštění aplikace z příkazové řádky a získání ovladače.

Za implementaci jednotkových testů byla zvolena knihovna `junit`[1] s využitím knihovny `jupiter`[2] pro případy kontroly vyvolání výjimky při plnění konkrétního příkazu. Knihovna `junit` sice v základu také nabízí kontrolu vyvolané výjimky, ale pouze v kontextu celého jednotkového testu a to je v určitých případech nedostatečně přesné. Existující jednotkové testy lze zařadit do několika kategorií:

[1] junit5 https://junit.org/junit5/
[2] jupiter https://junit.org/junit5/

Argumenty nástroje

Vzhledem k minimálnímu počtu argumentů aplikace, sestává testování z malého počtu, velice přímočarých jednotkových testů. Aplikace může mít pouze tři parametry: povinná cesta ke konfiguračnímu skriptu, dobrovolný argument udávající stupeň četnosti logování a udání cesty k schématu výstupu. Testováno je tedy nepřítomnost povinného argumentu a také špatně zadaná cesta. U logování je pak otestován legální a nelegální vstup.

Vznik ovladače a jeho parametry

V této části dochází k testování vytvoření samotného ovladače stránky. Pokaždé je vytvořen jeden druh ovladače prohlížeče, vyzkoušena základní akce a prohlížeč uzavřen. Zároveň je zde otestováno jiné umístění ovladače, než je domovský adresář aplikace a další parametry ovladače. Také je v této sekci otestován vznik několika ovladačů různých prohlížečů.

Akce s prohlížečem

Zde je otestováno korektní chování ovladače při ovládání prohlížeče, např. správně vyvolaná výjimka při snaze o klik na žádný či více prvků, odeslání formuláře bez výběru aj.

Změna stavů vstupů

Sekce testování, která pod sebe sdružuje větší množství jednotkových testů, související se změnou stavu formulářových prvků. Testy provádí změnu stavu u textových vstupů, prvku `checkbox`, `select`, `textarea`, `radio` a také u méně známých typů prvku `input`. Dále jsou zde kontrolovány reakce na vícenásobný výběr vstupů, nelegální změnu stavu, vynulování stavu a snaha zadat textový vstup do značky `div`.

Získávání atributů

Testy zaměřené na získávání atributů prvků stránky. Zahrnují testy získání všech atributů o prvku, získání `data-` a také tříd.

Informace o prvku

V této oblasti dochází k testování získaných informacích o prvku. Kromě získání všech informací bez vyvolání výjimky, jsou zde tyto oblasti:

- Pole `whitelist`. Chování při dodání prázdného pole, null a ověření totožnosti dvou souborů. Přičemž první vznikl dodáním všech znalostí do pole `whitelist` a druhý vynecháním pole `whitelist`.
- Zjištění vykresleného písma.
- Jednotná podoby navrácené hodnoty při různých formátech hodnot barvy.
- Testování absolutních hodnot pro hodnoty míry.
- Převod odsazení písma, zaoblení hran rámečku, velikosti obrázku na pozadí a pozice obrázku na pozadí.
- Test získání rozměrů prvku.

Nutno dodat, že část testů vznikla za účelem ověření korektního chování vypočtených stylů. I přesto byly tyto testy nutností, díky nimž byly odhaleny vypočtené styly, jež nejsou vždy navráceny v požadovaném tvaru.

Třída `Selector`

Jednotkové testy zkoumají korektnost chování objektů třídy `Selector` ve spojení s metodou `partiallyExportPage`.

Hromadné vyplňování formuláře

Testy zaměřené na chování metod pro vyplnění formuláře. Ověření, že u pokusů o výběr prvku, který není značkou `form`, výběr neexistujícího formuláře a výběr více než jednoho formuláře, dojde k vyvolání korektních výjimek. Dále otestování všech typů vstupů a ověření funkčnosti identifikace prvku, pokud stránka obsahuje v různých formulářích prvek se stejným atributem `name`. Obsahuje také testy pro výběr prvku pomocí rozšířeného identifikátoru a vyplnění z externího JSON souboru.

Konfigurační soubor

Testy vytvoření různých ovladačů a jejich schopností přímým překladem konfiguračních souborů.

9.2 ceur-ws.org

Vzhledem k spíše jednoduché grafické povaze stránky, nepřítomnosti skriptů a svému rozsahu, byla tato stránka vybrána k otestování základních schopností nástroje v reálném prostředí. V průběhu prací s touto stránkou byly zjištěny některé poznatky. Aplikace sice byla schopna dokončit průchod a export dat, ale ukázalo se, že výstupní soubor nemůže zůstat ve stávající podobě. Při exportu celého DOM stromu byl výsledný soubor neúnosné velikosti. Přesněji obsahoval 250 MB dat na pěti miliónech řádků. Přičemž většina dat byla redundantních.

Jako reakci na testování s touto stránkou byl významně upraven tvar výstupního souboru. Předně došlo k vynechání všech vlastností prvku s výchozí hodnotou (černé písmo, žádné odsazení, písmo o velikosti 16px). Tento krok znamenal, zvláště na stránkách s jednoduchým grafickým výstupem, markantní snížení velikosti výstupu. K dosažení dalších úspor byla nadále zjednodušena struktura výstupu následovně:

1. Vlastnosti, které lze pospat jedním řetězcem a také získané atributy byly přesunuty do atributů značky prvku stránky.
2. Byly zcela vynechány značky sdružující atributy a vlastnosti prvku. Hodnoty, jež sdružovaly byly přesunuty do atributů či jako přímý potomek hlavní značky.
3. Značka pro uchování potomků byla odstraněna a potomci jsou přímými potomky.
4. Třídy HTML elementu byly zpracovány do řetězce. V předchozím přístupu měla každá třída svoji značku.
5. `data-` byly sdruženy do atributů jedné značky, namísto značky pro každý `data-` atribut.

Těmito kroky bylo dosaženo zhruba desetinásobnému zmenšení velikosti souboru a výraznému omezení počtu řádků. Zároveň je nutno říci, že nedošlo k žádné ztrátě informací, ale pouze k odstranění redundance.

9.3 novinky.cz

Internetová stránka novinky.cz patří k jednomu z nejnavštěvovanějších zpravodajských serverů českého internetu [3]. Vyznačuje se stále spíše jednoduchým grafickým vzhledem, ale již obsahuje skripty a časově závislé akce.

V rámci testování s tímto webem vyvstala nutnost implementace podmíněného čekání. Konkrétně se jednalo o přítomnost okna s nabídkou notifikací. Okno sice zcela neznemožňuje práci se stránkou, ale může působit určité potíže. Proto musí být počkáno, dokud není viditelné, odmítnuto, a až pak skript pokračuje. Na potvrzení okna nevystačil obvyklý způsob kliknutí skrz Selenium WebDriver, a proto byla přidána možnost kliknout příkazem v jazyce Javascript. Příloha D.1 obsahuje ukázku konfiguračního souboru.

9.4 czc.cz

Web, který lze zařadit mezi největší české e-shopy s elektronikou. Obsahuje velké množství skriptů a AJAX volání, zároveň je responsivní, tedy přizpůsobivý aktuálnímu rozlišení webového prohlížeče. S poslední vlastností souvisí požadavek, který v průběhu testování vyvstal. Otevřít webový ovladač v konkrétním rozlišení, aby byla jistota, že bude prvek přítomný v takové podobě, v jaké ho očekáváme. Ovladač webového prohlížeče chrome tuto možnost nabízí skrz argument při startu. Do aplikace byla tedy přidána možnost zadat při získávání ovladače pole argumentů, které bude předáno ovladači při jeho vzniku. Zároveň nastala situace, kdy bylo třeba čekat, ale nebyla odhalena žádná vyhovující podmínka, na kterou by se dalo počkat. Byla tedy implementováno bezpodmínkové čekání po určité časové kvantum. Příloha D.2 nastiňuje základní práci s touto stránkou.

9.5 9gag.com

Populární sociální platforma pro sdílení uživatelsky vytvořeného obsahu. Byla vybrána pro styl, jakým je řešeno stránkování obsahu. Po dosažení konce obsahu je automaticky nahrán další obsah, přičemž je připojen k aktuálnímu dokumentu, starý obsah je skryt. Pro umožnění navigace i po tomto stylu stránek byla implementována operace pro zaslání klávesy prvkům, které nejsou vstupy. V případě této stránky se jedná o klávesu Page Down zaslanou prvku `body` (příloha D.3). Dále byl přidán výčtový typ pro snadnější použití kláves, které nelze zadat v textovém řetězci.

[3] Návštěvnost český stránek https://www.kurzy.cz/~nr/netmonitor/cz-media/home/

Kapitola 10

Závěr

V rámci této práce jsem nejdříve prostudoval již existující nástroje pro automatizaci webového prohlížeče. Nejprve jsem zmínil jejich přínos pro testování webových aplikací a porovnal ho oproti testování klasickému. Dále již následoval rozbor jednotlivých nástrojů. Zde jsem kladl důraz na nástroje z rodiny Selenium Suite. A to z důvodu jejich oblíbenosti, pokroku ve standardizaci ovládání prohlížeče a také proto, že jsem nástroj Selenium WebDriver vybral pro ovládání webových prohlížečů ve vzniklé aplikaci. Selenium WebDriver byl vybrán díky podpoře hlavních prohlížečů, možnosti navázat na jazyk Java (v němž vznikl nástroj diplomové práce) a také rozsahem příkazů pro webový prohlížeč.

Poté jsem prostudoval detaily reprezentace zobrazené webové stránky v prohlížeči, použité interní struktury a průběh zobrazení webové stránky.

Následně jsem se zaměřil na popis API webových prohlížečů. Při popisu jsem důkladně probral rozhraní, jež souvisí s webovou stránkou a jejím navázáním na skriptovací jazyky. Kvůli tématu diplomové práce jsem rozebral rozhraní DOM a získání vypočtených stylů, vše spolu s omezeními, které musí mít programátor na paměti.

V rámci diplomové práce jsem vytvořil nástroj s jehož pomocí je možné ovládat webový prohlížeč, vyplňovat formuláře a provádět další akce. Následně, na požádání, vygeneruje soubor ve formátu XML, obsahující reprezentaci požadované webové stránky. Soubor má tvar stromu HTML značek a u každé značky je popsána její geometrie, atributy a CSS vlastnosti. XML soubor představuje hlavní výstup, a proto je kladen velký důraz na jeho správný tvar a také jednotný tvar hodnot vypočtených CSS vlastností a geometrie. Pro podporu kontroly a lehčí navázání na budoucí nástroje jsem vytvořil soubor s definicí schématu v jazyku XSD.

Na vznik, schopnosti a podrobnosti implementace nástroje jsem se zaměřil v jádru textu práce. V textu se zabývám technikami, díky nimž lze ovládat webový prohlížeč za použití knihovny Selenium WebDriver při určitém stupni abstrakce. V textu jsem rozebral návrhová rozhodnutí, která napomohou v situaci, kdy vznikne potřeba vyměnit implementaci ovládání prohlížeče. Velký důraz jsem kladl na popis využití nástroje, tedy spuštění konzolové aplikace, operace aplikačního rozhraní a tvorbu konfiguračních skriptů. Jako vedlejší výstup testování vznikla celá řada konfiguračních skriptů, na kterých lze vidět obecný styl použití, ale i jednotlivé operace aplikačního rozhraní. Tyto soubory, spolu s jednotkovými testy a dokumentačními komentáři, jsou hlavní prostředky pro hlubší pochopení funkcionality nástroje. V rámci práce jsem také rozebral techniky pro získání informací o jednotlivých prvcích stránky. Tyto informace musí být v jednotném a pevně daném tvaru, a proto jsem zde popsal transformace informací do požadovaného tvaru.

Testování jsem rozdělil do dvou hlavních etap. První, která probíhala přímo při vývoji aplikace, se skládá výlučně z jednotkových testů. Tyto testy prověřují jednotlivé aspekty aplikace a je díky nim možnost ověřit, zda nedošlo při dalším vývoji k poškození jiné funkcionality aplikace.

V rámci vzniku jednotkových testů vyvstalo několik závažných nedostatků, které ztěžují využití headless módu a také prohlížeče HtmlUnit. Prohlížeč Chrome se ve svém headless módu chová odlišně než s grafickým výstupem. A to tak, že v některých případech vykreslí text jinou rodinou písma. Další problém představuje prohlížeč HtmlUnit. Tento prohlížeč, bez grafického výstupu, nabízí slabší podporu skriptovacího jazyka Javascript (přibližně ECMAScript 5.1), a tedy nemusí zvládat stránky, které neberou v potaz zpětnou podporu skriptovacího jazyka. Dalším hrubým nedostatkem je nedokonalost při získávání dat. Prohlížeč HtmlUnit není schopen získat tolik, a tak přesná data, jako prohlížeč Chrome. Zároveň HtmlUnit nezvládá v plném rozsahu využít operace nástroje, např. již nedokáže zkontrolovat stav prvku `checkbox` a `radiobutton` po změně jejich stavu v jazyce Javascript. HtmlUnit tedy doporučuji použít pouze ve spojení s jednoduchými webovými stránkami. Pro složitější webové stránky je vhodné využít prohlížeče Chrome nebo Firefox, které se chovají obdobně a korektně.

Testování s reálnými webovými stránkami jsem zahájil graficky velice jednoduchou stránkou ceur-ws.org. Stránka je vhodná k prvotnímu testování také z důvodu, že se na ní nenachází žádný skriptovací kód, a její ovládání tedy je velice usnadněno. V průběhu prací s *CEUR Workshop Proceedings* jsem odhalil, že množství redundancí ve výstupu aplikace je nepřípustné, a jako reakci provedl korekce výstupu, díky nimž došlo ke snížení redundance na minimum, a tedy k výraznému zmenšení velikosti výstupního souboru. Další na řadě byl oblíbený zpravodajský server novinky.cz. Při práci s touto webovou stránkou vyvstala potřeba pozastavení běhu programu na určité časové kvantum, nebo do chvíle, kdy stránka přejde do konkrétního stavu. Jako reflexi na tento problém jsem do nástroje zakomponoval dva různé druhy pozastavení běhu konfiguračního skriptu. Poslední ze série reálných testů jsem provedl s internetovým obchodem czc.cz. Zde nastala situace, kdy jsem nebyl schopen určit podmínku explicitního čekání. Proto přibylo třetí pozastavení – bezpodmínkové.

Na základě testování v reálném prostředí mohu konstatovat, že je možno aplikaci použít v reálném prostředí.

Nástroj by bylo dále možno rozšířit několika směry. Jedním ze směrů by mohl být paralelní průchod a získávání informací o stránce. Existovala by určitá množina procesů, které by postupně paralelně zpracovaly každou větev HTML dokumentu. Zde by nastal problém s různou hloubkou a délkou větví a musela by existovat vhodná heuristika, kdy na větev použít nový proces. Další rozšíření by mohlo spočívat v paměťové optimalizaci. Nyní je v paměti ukládán celý strom, který reprezentuje stránku, její prvky a získané informace. Tento přístup by bylo možné nahradit iterátorem nad DOM stromem, kdy by se každý uzel zpracoval a ihned zapsal do souboru. Při takovém přístupu by nejspíše došlo k problémům, jak správně rozhodnout, do kterého místa výstupu uzel připojit. S přihlédnutím k faktu, že na nástroj nejsou kladena vysoká časová a paměťová omezení, jsem se těmito rozšířeními nezabýval.

Literatura

[1] Antonova, A.; Shanovskiy, B. : RESEARCH AND ANALYSIS OF APPLICATION OF AUTOMATED TESTING IN WEB APPLICATIONS. *Avtomatizaciâ Tehnologičeskih i Biznes-Processov*, roč. 10, č. 1, 2018: s 55–58, ISSN 2312-3125, 10.15673/atbp.v10i1.882. Dostupné z: <https://doaj.org/article/c56d8286aa5149dea11a9264ca16d8e9>

[2] Belfiore, J. *Microsoft Edge: Making the web better through more open source collaboration* [online]. Microsoft, 2018 [cit. 2019-01-04]. Dostupné z: <https://blogs.windows.com/windowsexperience/2018/12/06/microsoft-edge-making-the-web-better-through-more-open-source-collaboration/#K9XF88Xm0yqDxju5.97>.

[3] Beverloo, P. *List of Chromium Command Line Switches* [online]. @2019 [cit. 2019-03-16]. Dostupné z: <https://peter.sh/experiments/chromium-command-line-switches/>.

[4] Bruns, A.; Kornstadt, A.; Wichmann, D. : Web Application Tests with Selenium. *Software, IEEE*, roč. 26, č. 5, 2009: s 88–91, ISSN 0740-7459, 10.1109/MS.2009.144. Dostupné z: <https://ieeexplore.ieee.org/abstract/document/5222802>

[5] Chaney, B. I. *Automated Web Application Testing Using Selenium* Worcester: Worcester Polytechnic Institute, 2017. Dostupné z: <https://digitalcommons.wpi.edu/mqp-all/1803/>.

[6] Clark, L. *Inside a super fast CSS engine: Quantum CSS (aka Stylo)* [online]. Mozilla Hacks, 2017 [cit. 2018-12-21]. Dostupné z: <https://hacks.mozilla.org/2017/08/inside-a-super-fast-css-engine-quantum-css-aka-stylo/>.

[7] Garsiel, T.; Irish, P. *How Browsers Work: Behind the scenes of modern web browsers* [online]. html5rocks, 2011 [cit. 2019-01-04]. Dostupné z: <https://www.html5rocks.com/en/tutorials/internals/howbrowserswork/>.

[8] Gogna, N. : Study of Browser Based Automated Test Tools WATIR and Selenium. *International Journal of Information and Education Technology*, roč. 4, č. 4, 2014: s 336–339. Dostupné z: <http://www.ijiet.org/papers/425-T0026.pdf>

[9] Gogna, N.; Kumari, R. : COMPARATIVE STUDY OF BROWSER BASED OPEN SOURCE TESTING TOOLS WATIR AND WET. *International Journal on Computer Science and Engineering*, roč. 3, č. 5, 2011: s 1910–1923, ISSN 0975-3397. Dostupné z: <http://search.proquest.com/docview/926311914/>

[10] Gojare, S.; Joshi, R.; Gaigaware, D. : Analysis and Design of Selenium WebDriver Automation Testing Framework. *Procedia Computer Science*, roč. 50, 2015: s 341–346, ISSN 18770509, 10.1016/j.procs.2015.04.038. Dostupné z: <https://linkinghub.elsevier.com/retrieve/pii/S1877050915005396>

[11] Grosskurth, A.; Godfrey, M. A reference architecture for Web browsers. *21st IEEE International Conference on Software Maintenance (ICSM'05)*. 2005, s. 661-664. Dostupné z: <http://ieeexplore.ieee.org/document/1510168/>. 10.1109/ICSM.2005.13.

[12] Kantor, I. *The Modern Javascript Tutorial* [online]. 2018 [cit. 2019-01-01]. Dostupné z: <http://javascript.info/>.

[13] van Kesteren, A. *DOM : Living Standard* [online]. WHATWG, 2018 [cit. 2018-12-28]. Dostupné z: <https://dom.spec.whatwg.org/>.

[14] van Kesteren, A.; Gregor, A.; Russell, A.; aj. *W3C DOM4* [online]. MIT, ERCIM, Keio, Beihang: W3C, 2015 [cit. 2018-12-28]. Dostupné z: <https://www.w3.org/TR/dom/>.

[15] Kumar, P. *Thread.sleep() in Java – Java Thread sleep* [online]. JournalDev, 2013 [cit. 2019-03-30]. Dostupné z: <https://www.journaldev.com/1020/thread-sleep-java>.

[16] Laskey, J. *JEP 335: Deprecate the Nashorn JavaScript Engine* [online]. Oracle, @2019 [cit. 2019-03-16]. Dostupné z: <https://openjdk.java.net/jeps/335>.

[17] Mendes, E.; Mosley, N. *Web Engineering*. New York: Springer, c2006. ISBN 978-3-540-28196-2.

[18] Oracle and/or its affiliates. *Unchecked Exceptions — The Controversy* [online]. @1995-2019 [cit. 2019-03-09]. Dostupné z: <https://docs.oracle.com/javase/tutorial/essential/exceptions/runtime.html>.

[19] Pieters, S.; Glazman, D. *CSS Object Model (CSSOM)* [online]. MIT, ERCIM, Keio, Beihang: W3C, 2016 [cit. 2018-12-28]. Dostupné z: <https://www.w3.org/TR/cssom-1/>.

[20] Pujari, C. *Implicit, Explicit, & Fluent Wait in Selenium WebDriver* [online]. Guru99, @2019 [cit. 2019-03-30]. Dostupné z: <https://www.guru99.com/implicit-explicit-waits-selenium.html>.

[21] Singh, I.; Tarika, B. : Comparative Analysis of Open Source Automated Software Testing Tools: Selenium, Sikuli and Watir. *International Journal of Information & Computation Technology*, roč. 4, č. 15, 2014: s 1507–1518, ISSN 0974-2239. Dostupné z: <https://bit.ly/2Uch2et>

[22] Stewart, S.; Burns, D. *WebDriver* [online]. MIT: W3C, © 2018 [cit. 2018-12-17]. Dostupné z: <https://www.w3.org/TR/2018/REC-webdriver1-20180605/>.

[23] Tappenden, A.; Beatty, P.; Miller, J.; aj. Agile security testing of Web-based systems via HTTPUnit. *Agile Development Conference (ADC'05)*. 2005, s. 29-38. Dostupné z: <http://ieeexplore.ieee.org/document/1609802/>. 10.1109/ADC.2005.11.

[24] Uppal, N.; Chopra, V. : Design and Implementation in Automatic Testing Tool Selenium IDE, RC and Web Driver. *International Journal of Systems, Algorithms & Applications*, roč. 2, č. 5, 2012: s 43–46. Dostupné z: <http://search.proquest.com/docview/1357564863/>

[25] Uppal, N.; Chopra, V. : Enhancement and Elimination of Roadblocks in Automation Testing Tool Selenium RC. *International Journal of Computer Applications*, roč. 53, č. 5, 2012: s 12–14, ISSN 09758887, 10.5120/8416-0892. Dostupné z: <http://search.proquest.com/docview/1237151384/?pq-origsite=primo>

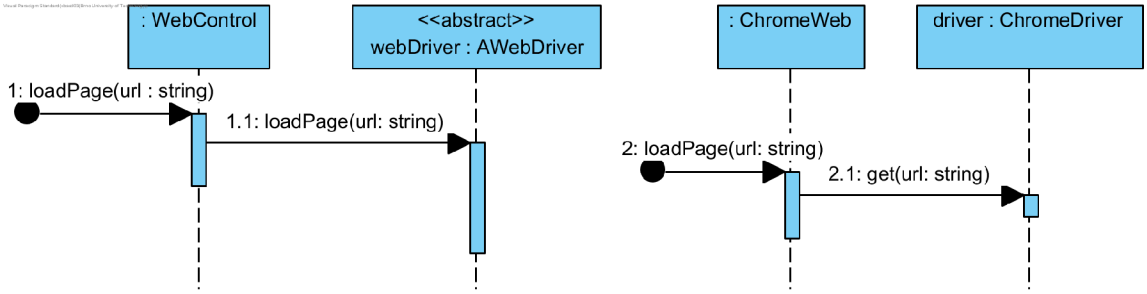
[26] Veverke. *Adding user-data-dir option to ChromeDriver makes it not work and timeout only* [online]. Stackexchange, 2015 [cit. 2019-04-07]. Dostupné z: <https://sqa.stackexchange.com/questions/15311/adding-user-data-dir-option-to-chromedriver-makes-it-not-work-and-timeout-only>.

Přílohy

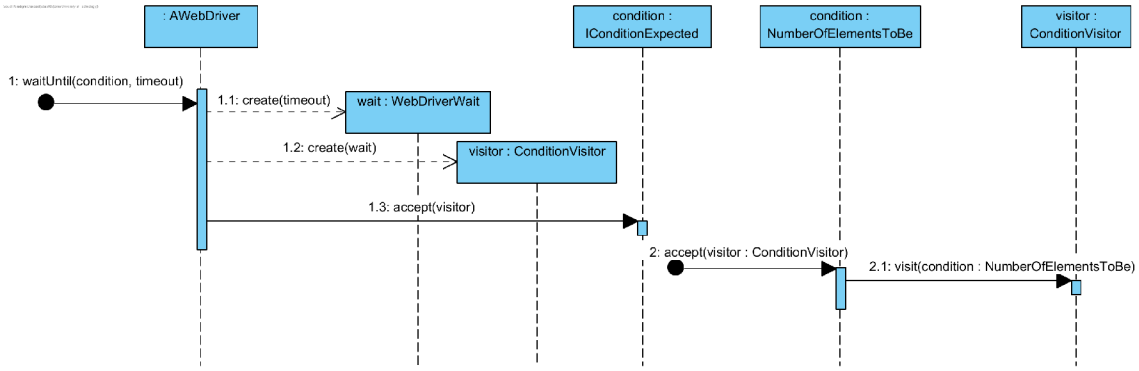
Příloha A

Diagramy Sekvence

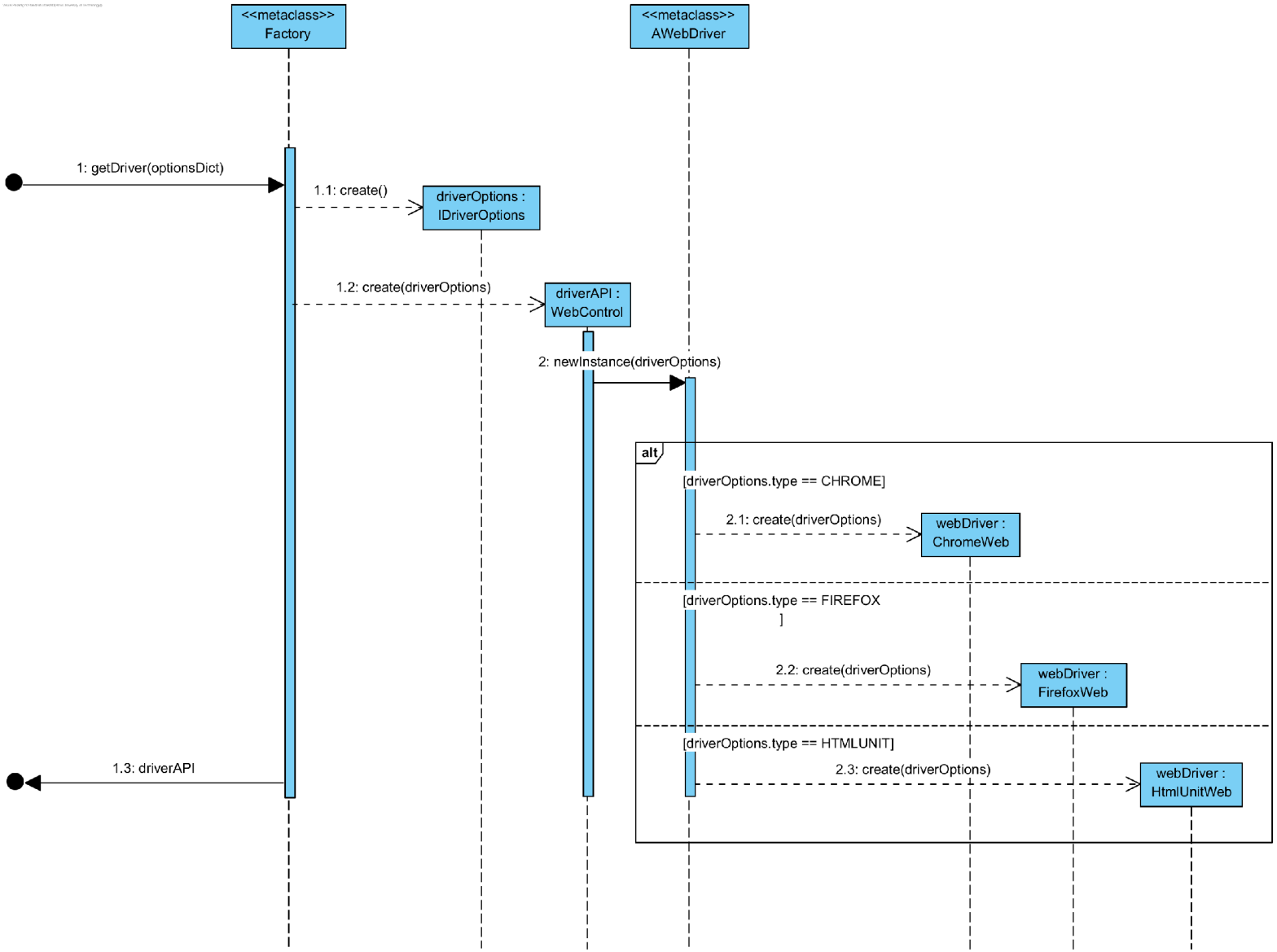
Součástí této přílohy budou vybrané sekvenční diagramy aplikace.



Obrázek A.1: Sekvenční diagram pro načtení stránky. Třída `ChromeDriver` je již součástí knihovny Selenium Webdriver.



Obrázek A.2: Sekvenční diagram pro začátek čekání na viditelnost prvku stránky. Třída `WebDriverWait` je již součástí knihovny Selenium Webdriver.

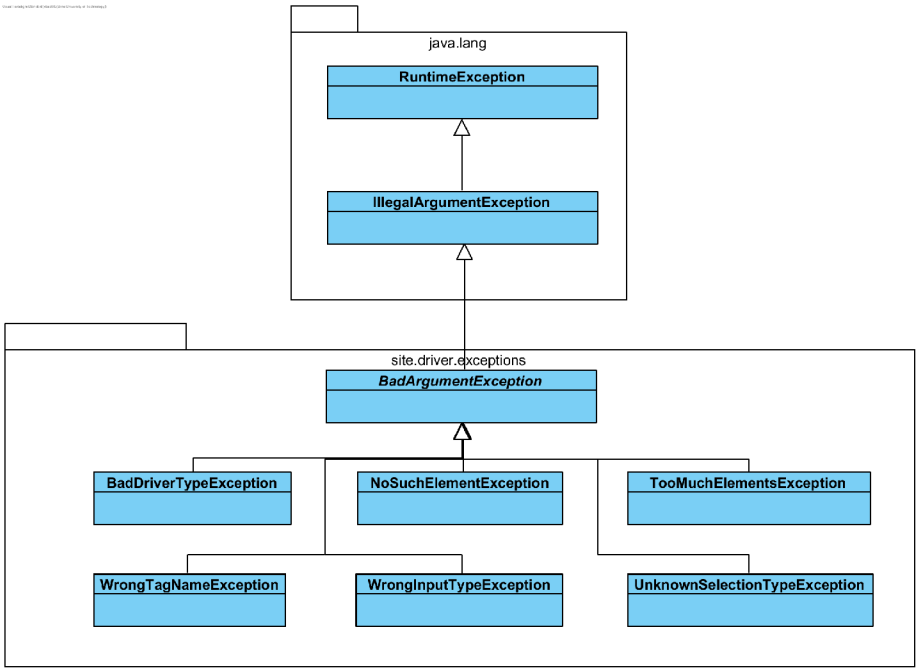


Obrázek A.3: Sekvenční diagram vzniku ovladače

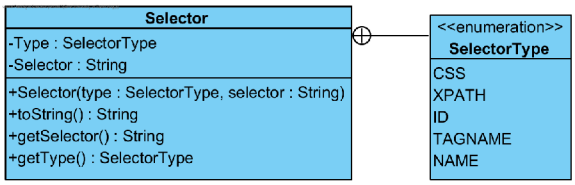
Příloha B

Diagramy Tříd

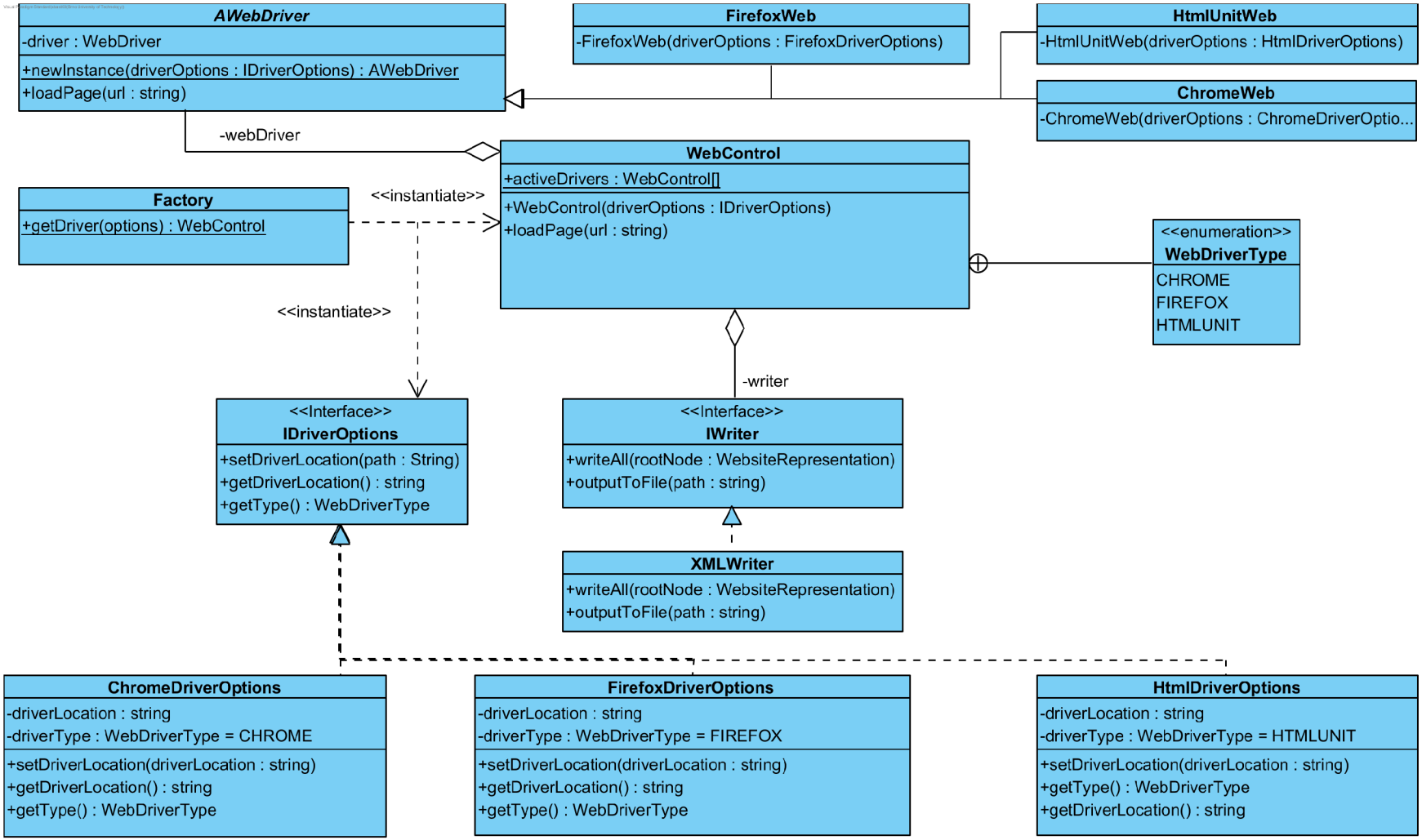
Součástí této přílohy budou vybrané diagramy tříd aplikace.



Obrázek B.1: Třídní hierarchie `unchecked` výjimek a jejich začlenění do balíčků



Obrázek B.2: Třída `Selector` a vnořený výčtový typ. Instance třídy slouží pro identifikaci prvku na HTML stránce.



Obrázek B.3: Rámcový diagram tříd

Příloha C

Metody aplikačního rozhraní

Vybrané signatury metod aplikačního rozhraní nástroje.

```
/**
 * Provedení akce přechodu na webovou stránku.
 * @param url požadované stránky, url musí být zcela kvalifikováno, včetně protokolu.
 */
public void loadPage(String url);

/**
 * Kliknutí na element na aktuální stránce.
 * @param selector selektor pro určení elementu na stránce.
 */
public void click(Selector selector) throws BadArgumentException;

/**
 * Potvrzení vyskakovacího okna. Doporučeno kombinovat s metodou {@link #waitUntil} a
↪  podmínkou alertIsPresent.
 */
public void acceptAlert();

/**
 * Zavření ovladače. Pokud není zavoláno bude zavřen dle nastavení ovladače.
 * V případě již zavřeného ovladače nebude akce provedena.
 * Po zavření se již nesmí ovladač využívat, pokud se použije bude vyvolána výjimka.
 */
public void close();

/**
 * Odeslání formuláře.
 * @param selector selektor pro určení formuláře na stránce na stránce.
 */
public void sendForm(Selector selector) throws BadArgumentException;
```

Obrázek C.1: Signatury metod pro ovládání prohlížeče.

```java
/**
 * Pozastavení provádění skriptu.
 * @param milliseconds doba na kterou bude program pozastaven.
 * @throws InterruptedException pokud je pozastavení přerušeno.
 */
public void wait(int milliseconds) throws InterruptedException;

/**
 * Nastavení časové tolerance pro identifikaci pruku na stránce. Tolerance platí pro každý
 ↪  následující příkaz.
 * @param seconds tolerance v sekundách.
 * Pokud není element nalezen do konce časového limitu, je vyvolána výjimka {@link
 ↪  NoSuchElementException}.
 */
public void setUpImplicitlyWait(int seconds);

/**
 * Podmíněné čekání. Pokud dojde k splnění podmíky do doby dané hodnotou timeout, program
 ↪  pokračuje. V opačném případě je vyvolána výjimka
 * @param condition podmínka, která musí být splněna.
 * @param timeout časový limit v sekundách.
 */
public void waitUntil(IConditionExpected condition, int timeout);
```

Obrázek C.2: Operace pro čekání.

```java
/**
 * Zaslání hodnoty do textového input pole aktuální webové stránky. Input je vybrán pomocí
 ↪  selektoru.
 * Může být vybráno i více elementů. Mohou být zadány typy viz. {@link
 ↪  AWebDriver#allowedInputAsText}.
 * @param selector selektor, jež vybere input.
 * @param value hodnota, jež bude zadána do vstupu.
 */
public void inputText(Selector selector, String value) throws BadArgumentException;

/**
 * Vyčištění input. Může být vybráno i více elementů. Mohou být zadány typy viz. {@link
 ↪  AWebDriver#allowedInputAsText}
 * @param selector selektor, jež vybere input
 */
public void clearTextInput(Selector selector) throws BadArgumentException;

/**
 * Zaslání hodnoty barvy do input typu color. Input je vybrán pomocí selektoru.
 * Může být vybráno i více elementů.
 * @param selector selektor, jež vybere input.
 * @param color platný řetězec typu color.
 */
public void inputColor(Selector selector, String color) throws BadArgumentException;

/**
 * Vynulování input s type color. Může být vybráno i více elementů.
 * @param selector identifikace prvku.
 */
public void clearColor(Selector selector) throws BadArgumentException;

/**
 * Změna stavu vstupu checkbox na stav state. Může být vybráno i více pruků.
 * @param selector identifikace prvků stránky.
 * @param state požadovaný stav.
 */
public void changeCheckbox(Selector selector, boolean state) throws BadArgumentException;
/**
 * Výběr konkrétního přepínače z pole RadioButton. Musí být vybrán právě jeden.
 * @param selector identifikace prvku na stránce.
 */
public void chooseRadioButton(Selector selector, String value) throws BadArgumentException;
```

Obrázek C.3: Změna stavu vstupů. První dvě dvojice metod slouží pro nastavení stavu textového pole a pole pro barvu. předposlední metoda pak pro změnu stavu `checkbox` a poslední pro výběr možnosti v prvku `select`

```java
/**
 * Vyplnění formuláře pomocí dodaného slovníku.
 * @param formSel identifikátor formuláře. Určuje kontext ve kterém budou nalezeny pruky.
 * @param dictOfValues slovník dvojic identifikátor-hodnota. Přičemž identifikuje se pomocí
 ↪  dvou metod:
 *<ol>
 *  <li>Identifikace jménem. identifikátor je pak řetězec který přímo určuje hodnotu
 ↪  atributu Name.
 *  <li>Identifikace jménem a hodnotou.  Identifikátor se skládá ze dvou částí oddělených
 ↪  dvojtečkou. První část určuje hodnotu atributu Name. Druhá hodnotu atributu Value.
 *</ol>
 *<p>
 * Na identifikaci je dále uplatněno omezení, že může vybrat pouze jeden prvek v daném
 ↪  kontextu formuláře.
 */
public void fillForm(Selector form, Map<String, Object> dictOfValues) throws
 ↪  BadArgumentException;

/**
 * Vyplnění slovníku pomocí obsahu souboru na cestě filePath.
 * @param formSel identifikátor formuláře. Určuje kontext ve kterém budou nalezeny pruky.
 * @param filePath cesta k souboru jehož obsah je ve tvaru JSON. Tvar obsahu je popsán v
 ↪  {@link #fillForm(Selector, Map)}.
 */
public void fillForm(Selector form, String filePath) throws IOException, ParseException,
 ↪  BadArgumentException;
```

Obrázek C.4: Signatury metod pro vyplnění formuláře.

```java
/**
 * Kontrola, zda element existuje na stránce.
 * @param selector objekt třídy Selector. Pokud dojde k výběru více pruků stránky, bude
 ↪  oznámen počet.
 * @return zda takový element existuje na stránce.
 */
public boolean checkIfElementExists(Selector selector);

/**
 * Kontrola, zda vstupy vybrané selektorem splňují podmínku.
 * Může být vybráno i více inputů.
 * @param selector výběr pruku input.
 * @param value hodnota, jež se má zkontrolovat.
 * @return zda všechny vstupy s daným selektorem obsahují správnou hodnotu.
 */
public boolean checkInputValue(Selector selector, String value) throws
 ↪  BadArgumentException;

/**
 * Kontrola, zda je checkbox určený pomocí selektoru zatrhlý. V případě vícenásobného
 ↪  výběru je určen průnik stavů.
 * @param selector identifikace pruků stránky.
 * @return zda je zaškrtlý.
 */
public boolean checkCheckbox(Selector selector) throws BadArgumentException;
```

Obrázek C.5: Některé signatury metod sloužící pro ujištění stavu stránky. První metoda slouží pro ověření existence prvku, druhá porovná textový vstup s očekávanou hodnotou. Poslední metodě stačí pouze jeden parametr, zjišťuje totiž,s zda je zatrhlý `checkbox`.

```java
/**
 * Export aktuální stránky. Po průchodu je do souboru zapsána její grafická textová
 ↪  reprezentace.
 * @param exportPath cesta k souboru včetně názvu. Cesta může být relativní nebo absolutní.
 */
public void exportPage(String exportPath);

/**
 * Export aktuální stránky. Po průchodu je do souboru zapsána její grafická textová
 ↪  reprezentace.
 * @param exportPath cesta k souboru včetně názvu.
 * @param propWhitelist bílá listina exportovaných properties. Pokud je hodnota null,
 ↪  export všech. Pokud je hodnota prázdné pole, export ničeho.
 * @param attrWhitelist bílá listina exportovaných atributů. Pokud je hodnota null, export
 ↪  všech. Pokud je hodnota prázdné pole, export ničeho.
 */
public void exportPage(String exportPath, List<String> propWhitelist, List<String>
 ↪  attrWhitelist);

/**
 * Částečný export stránky. Z aktuální stránky vybere pouze tu větev jejíž výběr je dán
 ↪  selektorem. Po průchodu je do souboru zapsána její grafická textová reprezentace.
 * @param selector objekt z třídy Selector. Slouží pro nalezení pruku na stránce.
 * @param exportPath cesta k souboru včetně názvu.
 */
public void partiallyExportPage(Selector selector, String exportPath);

/**
 * Několikanásobný částečný export stránky. Z aktuální stránky vybere pouze ty větve
 ↪  jejichž výběr je dán selektory. Po průchodu je do souboru zapsána jejich grafická
 ↪  textová reprezentace.
 * @param selectors pole objektů z třídy Selector. Slouží pro identifikaci pruku na
 ↪  stránce.
 * @param exportPath cesta k souboru včetně názvu.
 */
public void partiallyExportPage(Selector[] selectors, String exportPath);

/**
 * Několikanásobný částečný export stránky. Z aktuální stránky vybere pouze ty větve
 ↪  jejichž výběr je dán selektory. Po průchodu je do souboru zapsána jejich grafická
 ↪  textová reprezentace.
 * @param selectors pole objektů z třídy Selector. Slouží pro nalezení pruku na stránce.
 * @param exportPath cesta k souboru včetně názvu.
 * @param propWhitelist bílá listina exportovaných properties. Pokud je hodnota null,
 ↪  export všech. Pokud je hodnota prázdné pole, export ničeho.
 * @param attrWhitelist bílá listina exportovaných atributů. Pokud je hodnota null, export
 ↪  všech. Pokud je hodnota prázdné pole, export ničeho.
 */
public void partiallyExportPage(Selector[] selectors, String exportPath, List<String>
 ↪  propWhitelist, List<String> attrWhitelist);
```

Obrázek C.6: Signatury metod pro export stránky.

Příloha D

Konfigurační skripty

V této příloze budou ukázány některé z konfiguračních skriptů, vzniklých především při testování v reálném prostředí.

Algoritmus D.1: Ukázka použití konfiguračního souboru k ovládání stránky novinky.cz. Nejprve je odmítnuto vyskakovací okno a exportována hlavní strana. Následně je proveden přechod na Hlavní zprávu dne a proveden export jejího nadpisu a těla. S tím, že jsou exportovány všechny získané informace, ale z atributů pouze třída a id.

```
driver = Factory.getDriver({
    driver: 'chrome'
});

driver.loadPage("https://www.novinky.cz/");
var dialogPopUp = css("#onesignal-popover-dialog");

driver.waitUntil(visibilityOfElement(dialogPopUp), 3);
driver.clickWithJs(css("#onesignal-popover-cancel-button"));
driver.waitUntil(invisibilityOfElement(dialogPopUp), 3);

driver.exportPage("output/novinky/mainPage.xml");

driver.click(css("#TopStory .topArticle h2 a"));

driver.partiallyExportPage([id("articleHeaderBig"),
↪  css("#articleBody")], "output/novinky/topStory.xml", null,
↪  [attributesEnum.id, attributesEnum.class]);
```

Algoritmus D.2: Ukázka použití konfiguračního souboru k ovládání stránky czc.cz. Nejprve je zkontrolováno, zda je již přihlášeno ke stránce. Pokud tomu tak není, je provedeno kliknutí na přihlášení, počkáno na odhalení přihlašovacího formuláře a vyplněny přihlašovací údaje z json souboru. Z důvodu přesměrování je poté počkáno 2 sekundy. Následně jsou vyhledány produkty, které obsahují řetězec "lampa"a proveden export každé strany.

```
driver = Factory.getDriver({
    driver: 'chrome',
    'other-options': ['--window-size=1920,1080']
});

driver.loadPage("https://www.czc.cz/");

if (driver.checkIfElementExists(id("login"))){
    driver.click(id("login"));
    driver.waitUntil(visibilityOfElement(css("#popup-login")), 2);
    driver.fillForm(css(".login-form"), 'credentials.json');
    driver.sendForm(css(".login-form"));

}

driver.wait(2000);
driver.inputText(id("fulltext"), "lampa");
driver.sendForm(css(".fulltext"));

i=1;
while(driver.checkIfElementExists(css(".btn.show-next"))){

    driver.click(css(".btn.show-next"));
    driver.exportPage("czc-product-name-page-"+i+".xml");

    i++;
}
driver.exportPage("czc-product-name-page-"+i+".xml");
```

Algoritmus D.3: Ukázka použití konfiguračního souboru k ovládání stránky 9gag.com. Nejprve je potvrzen dialog s Cookies a načteno pět stránek obsahu. Poté již dojde k exportu stránky.

```
driver =  Factory.getDriver({
    driver: 'chrome'
});


driver.loadPage("http://9gag.com");

//accept cookies
driver.waitUntil(visibilityOfElement(css(".popup_content--2JBXA")),5);
driver.clickWithJs(css(".intro_acceptAll--23PPA"));

//load some content
for (var i = 0; i<5; i++) {
    driver.sendKey(tag("body"), Keys.PAGE_DOWN);
}
driver.exportPage("output/9gag/5sites.xml");
```