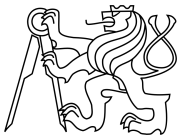
ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ



ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	Editor XML konfigurací modulárních systémů
Student:	Richard Strnad
Vedoucí:	Ing. Tomáš Tisančín
Studijní program:	Informatika
Studijní obor:	Softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce letního semestru 2017/18

Pokyny pro vypracování

Cílem práce je navrhnout a implementovat editor XML konfigurací pro modulární systémy. To jsou systémy sestávající se z malých nezávislých programových modulů, které jsou propojené skrze definované rozhraní. Každý modul má konfiguraci definující jeho vlastnosti, možná propojení a chování.

Předběžné funkční a nefunkční požadavky:
- editor bude realizován jako webová aplikace,
- umožní tvorbu nových i editaci existujících konfigurací,
- umožní práci v textovém a grafickém režimu,
- bude mít k dispozici seznam použitelných modulů včetně detailů o konfiguraci a možnostech propojení.

Postupujte v těchto krocích:
1. Analyzujte současná řešení.
2. Analyzujte konkrétní uživatelské požadavky.
3. Na základě analýzy proveďte návrh a zvolte vhodné technologie.
4. Návrh implementujte, zdokumentujte a vhodným způsobem otestujte.

Seznam odborné literatury

Dodá vedoucí práce.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

prof. Ing. Pavel Tvrdík, CSc.
děkan

V Praze dne 23. ledna 2017

Prohlášení autora práce

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací. Nemám závažný důvod proti užití tohoto školního díla ve smyslu §60 Zákona 4. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon).

V Praze 16. května 2017 .....

Podpis autora práce

Poděkování

Rád bych tímto poděkoval Ing. Tomáši Tisančínovi za jeho ochotu a odbornou pomoc při tvorbě této práce. Také bych rád poděkoval rodině a přátelům za to, že mě při tvorbě práce podporovali.

Abstrakt

Modulární systémy se rychle staly nejběžnějším způsobem, jak budovat rozsáhlejší informační systémy. Ačkoliv je v dnešní době těchto systémů nepřeberné množství, neexistuje žádný dostatečně intuitivní nástroj, kterým lze jednoduše editovat jejich konfigurace. V této práci provedu analýzu toho, co by měl takový editor splňovat a provedu implementaci vybraných metod v rámci praktické části práce.

Klíčová slova

Editor XML konfigurací, implementace, modulární systémy, uživatelské rozhranní, uživatelská přívětivost, Javascript, mxGraph

Abstract

Modular systems have quickly became the most common way to build large-scale information systems. Although there are large amount of these systems in these days, there still isn't any intuitive tool to easily edit their configurations. In this work I will analyze how should intuitive editor look and implement selected methods within the practical part of the thesis.

Key words

XML configuration editor, implementation, modular systems, user interface, user friendliness, Javascript, mxGraph

Obsah

Seznam obrázků

Seznam tabulek

Kapitola 1

Úvod

V dnešní době se na vývoj software klade velké množství požadavků. Jedním z nejžádanějších je znovupoužitelnost informačního systému. Stále populárnějšími se proto stávají takzvané modulární systémy, tedy systémy, které se sestávají z malých, nezávislých modulů.
I přestože jsou tyto systémy populární, většinou zde chybí uživatelsky přívětivá možnost, jak tyto systémy konfigurovat.

Výsledek práce bude prospěšný zejména firmě IndSoft s.r.o., pro účely jejíhož modulárního systému bude editor primárně uzpůsoben. Při vývoji editoru bylo dbáno na to, aby nebylo problematické editor uzpůsobit pro účely jiného systému.
Téma jsem si zvolil, jelikož osobně spoluvyvíjím zmíněný modulární systém a pocítil tak absenci rozumného a přívětivého editoru konfigurací. V této práci se zabývám analýzou dosavadních řešení, návrhem a implementací zmíněného editoru.

1.1 Struktura práce

Práce dále pokračuje v následující struktuře: Prvně se budu zabývat problematikou modulárních systémů a tím, v čem jsou nám tyto systémy prospěšné. Porovnám různé typy programovacích metodik a řeknu něco k historii webového výoje. Zde taktéž vysvětlím související pojmy. V druhé části přejdu na analýzu současných řešení a stanovím zde jejich konkrétní výhody a nevýhody a provedu výběr vhodných technologií pro účely zmiňovaného editoru.
Následně ve třetí části rozeberu návrh architektury editoru za použití zvolených technologií.

1.2 Cíl práce

Cíl práce můžeme rozdělit na cíl rešeršní části práce a cíl praktické části práce.
Cílem rešerše v rámci této práce bylo seznámení se s metodikou vývoje webových aplikací v jazyce Javascript, analýza specifických potřeb pro dostatečnou uživatelskou přívětivost editoru, analýza dostupných knihoven pro práci s grafy a následné studium konkrétní vybrané knihovny.
Cílem praktické části práce bylo vybraní způsobů, které by ulehčily editaci komplexních XML konfigurací a jejich implementace do zmiňovaného editoru.

Kapitola 2

Pozadí a pohled do historie

V této kapitole se budu zaobírat několika základními pojmy týkajících se modulárních systémů a předvedu některá ze současných řešení. Také zde proberu různé metodiky vývoje softwaru a rozeberu zde také krátce něco z historie vývoje webových aplikací.

2.1 Definice pojmů

2.1.1 Modulární systémy

Modulární systémy jsou systémy sestávající se z modulů. Při tvorbě těchto systémů se klade důraz na vysokou míru abstrakce.
Tato vysoká míra abstrakce je příčinou potřeby souboru s upřesňujícími informacemi pro daný projekt.

2.1.2 Konfigurace modulárního systému

Jedná se o soubor obsahující informace o použitých modulech, jejich spojeních a možností jejich konfigurace.

2.1.3 Port

Textový řetězec, který společně s datovým rozhraním tvoří u jednoho modulu jednoznačný identifikátor spojení.

2.1.4 Providing connection (Poskytované spojení)

Obsahuje datové rozhraní, na které se mohou ostatní moduly připojit a modul, kterému rozhraní náleží, skrze něj ovládat nebo od něj získat data.
V případě, že jeden modul poskytuje dvě rozhraní stejného typu, používá se pro odlišení dodatečná informace v podobě portu.

2.1.5 Consuming connection (Požadované spojení)

Obsahuje datové rozhraní, které modul vyžaduje pro svou funkčnost.
V případě, že jeden modul poskytuje dvě rozhraní stejného typu, používá se pro odlišení dodatečná informace v podobě portu. Rozlišujeme dva typy a to spojení povinné a nepovinné.

2.1.6 Generated connection (Generované spojení)

Generované spojení je zvláštním typem buď požadovaného, nebo poskytovaného spojení.
Jedná se o spojení, které není dáno pevně a počet jeho výskytů v modulu se může měnit na základě aktuální konfigurace modulu.

2.1.7 Propojení modulů

Dva moduly mohou být propojeny za předpokladu, že jeden modul nabízí spojení stejného typu, který vyžaduje modul druhý.
Poskytovaná a požadovaná spojení jsou ve vztahu 1:N.
Pokud nastane situace, ve které existuje požadované spojení, které bylo definováno jako povinné a nebylo k němu přiřazeno žádné poskytované spojení, není konfigurace modulárního systému validní.

2.1.8 Modul

Modul je základním stavebním prvkem celé konfigurace.
Každý modul má vlastní definici rozhraní, které poskytuje (Providing connection) a které potřebuje pro svou funkčnost (Consuming connection).
Poskytovaná spojení modulu nemusí poskytovat pouze datové rozhraní modulu, kterému spojení náleží. Může se taktéž jednat o pouhé přeposkytnutí datového rozhraní modulu

jiného, na který je daný modul napojen. Modul sám o sobě musí mít určený datový typ a unikátní jméno.

2.1.9 Deskriptor

Jelikož editor není tvořen na míru jednomu určitému projektu, nastává problém s udržováním informací o použitelných modulech, jejich spojeních a možných konfiguracích.
Editor nemůže obsahovat znalost všech použitelných modulů ve všech projektech. Zaprvé by to omezilo jeho genericitu a možnosti použití, zadruhé by obsahoval spoustu informací, které k editaci konfigurace daného projektu nepotřebuje.
Nejrozumnějším řešením je vytvořit soubor, který by obsahoval všechny informace, které editor potřebuje o daném projektu vědět, aby byl schopen vytvořit validní konfiguraci a poskytnul uživateli všechny moduly a možnosti jejich spojení.
Deskriptor obsahuje informace unikátní vždy pro daný projekt.
Jedná se o XML soubor, který musí být validní dle aktuálního XSD dodávaného s projektem.
Deskriptor musí obsahovat informace o všech použitelných modulech a jejich poskytovaných a požadovaných spojeních a to i v případě, že jsou generovány na základě konfigurace modulu.

2.2 Metody návrhu softwaru

Většina prvních programů, které lidé vytvoří jsou jednoduchými sekvencemi příkazů, které nějakým způsobem pracují s daty. Tato data bývají obvykle globální napříč celou aplikací.
Tento přístup má mnoho nevýhod a s postupným narůstáním velikosti aplikace se její další vývoj stává neudržitelným. Například pokud programátor potřebuje provést stejnou sekvenci příkazů na dvou odlišných místech aplikace, musí danou sekvenci zkopírovat a vložit na obě místa. Metody návrhu softwaru jsou takovým uceleným seznamem postupů a pravidel, které zpřehledňují zdrojový kód a umožňují jeho jednodušší rozšířitelnost. [1]

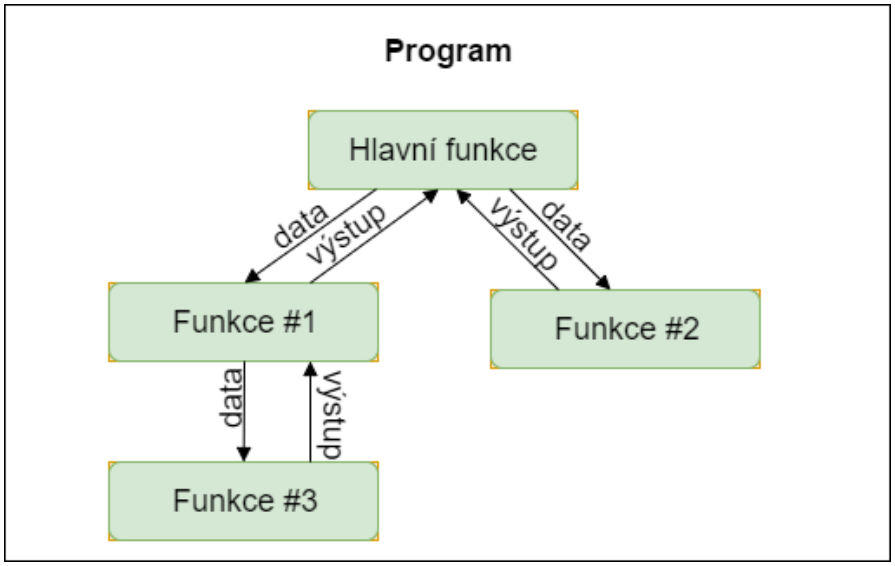
2.2.1 Strukturované programování

Prvním rozšířeným způsobem, jak se zbavit zbytečného kopírování kódu po celé aplikaci je takzvané strukturované programování. Při využití této metody dochází k rozdělení zdrojového kódu do funkcí (procedur).

Historicky se za počátek strukturovaného programování považuje článek Edsgera W. Dijkstry [2], ve kterém Dijkstra žádá o odstranění příkazu skoku z vyšších programovacích jazyků. Ačkoliv je možné, že došlo k prosazení strukturovaného programování na základě tohoto článku, myšlenka strukturovaného programování se více přisuzuje článku Böhma a Jacopiniho z roku 1966 [3], které Dijkstra ve své práci sám citoval. Hlavním účelem strukturovaného programování bylo omezit v programech používání příkazu skoku. Strukturované programování staví na myšlence, že jakýkoliv program lze sestrojit pouze za pomoci tří struktur:

- Selekce – Zvolení následujících příkazů na základě aktuálního stavu programu. Je tvořena podmínkou a jednou, dvěma nebo více výběrovými složkami.
- Sekvence – je tvořena jedním nebo několika kroky, které se provedou právě jednou v daném pořadí.
- Iterace – jedná se o opakování určité sekvence příkazů dokud platí nějaká podmínka.

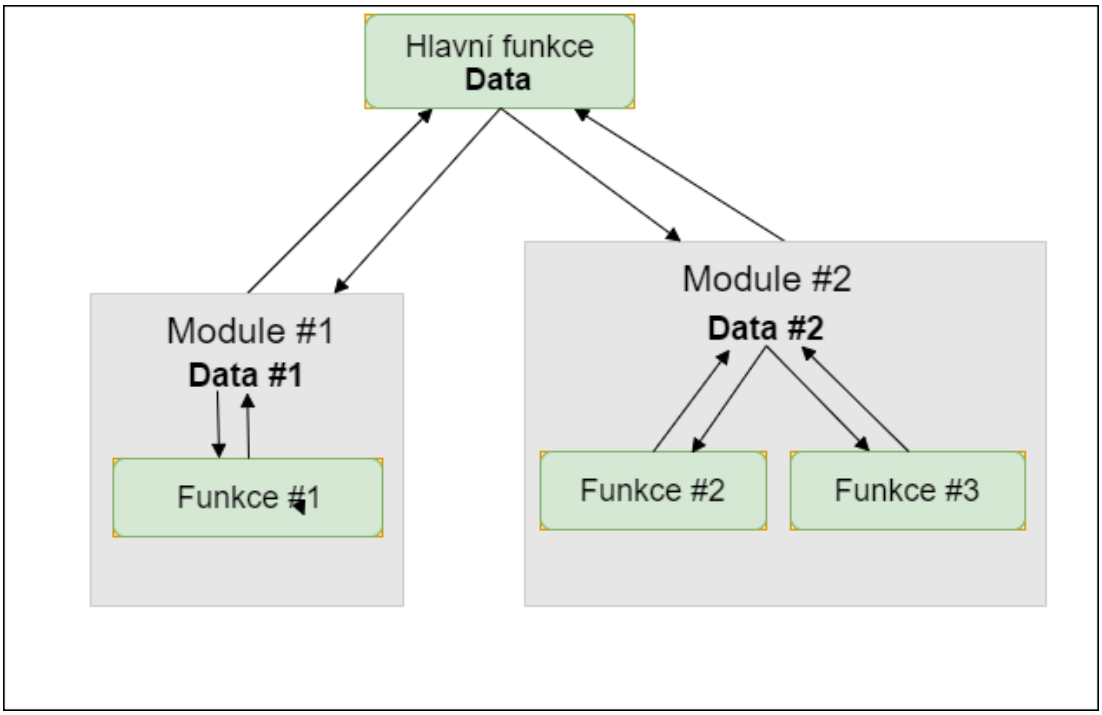
Při dodržení těchto struktur lze na program dále nahlížet pouze jako na sekvenci volání funkcí (procedur). Za tato volání je zodpovědná hlavní funkce programu. Za správu dat v programu zodpovídá hlavní funkce, která je dle potřeby poskytuje ostatním funkcím.



Obrázek 2.1: Zjednodušený náčrt fungování strukturovaného programování

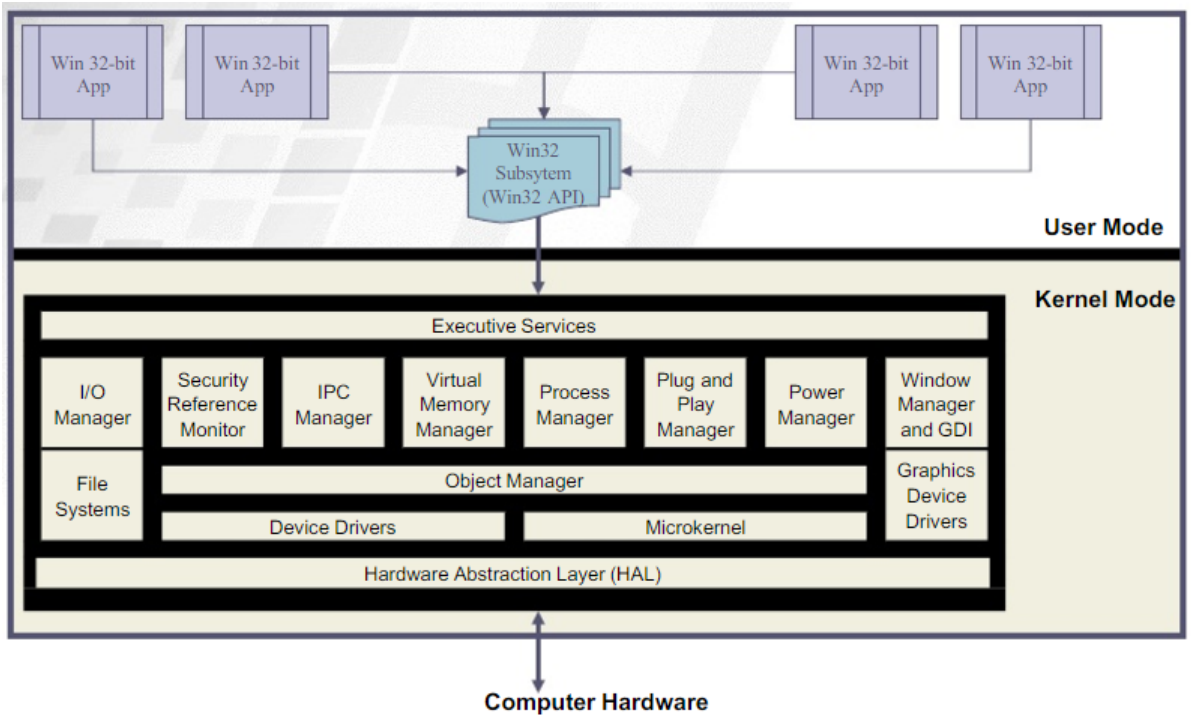
2.2.2 Modulární programování

Modulární programování je metoda vytváření programů, která je založena na myšlence rozložit řešený problém na pokud možno izolované podproblémy, jejichž jednotlivé řešení je podstatně jednodušší než souhrnné řešení celého problému.
O řešení jednotlivých podproblémů se postarají izolované prvky, takzvané moduly. [4]
Cílem modulárního programování je tedy rozdělení programu na různé části tak, aby se z hlediska funkcionality co nejméně překrývaly a byly tudíž zaměnitelné. Tento jev je v programování znám také pod zkratkou SoC (Separation of Concerns, česky oddělení zodpovědnosti).
V dnešní době se jedná o nejpoužívanější metodu programování pro jakékoliv větší



Obrázek 2.2: Zjednodušený náčrt fungování modulárního programování

systémy. Mezi nejznámější příklady oddělení zodpovědnosti do izolovaných modulů jmenujme například referenční model ISO/OSI, technologie používané na internetu (HTML/CSS/JS), nebo například architektura operačního systému Windows XP (na obrázku 2.3). Hlavním rozdílem oproti strukturovanému programování je fakt, že se nyní o správu dat nestará pouze hlavní část programu, ale každý modul může mít svůj vlastní interní stav (data), ke kterému nemá přístup nikdo jiný.



Obrázek 2.3: Architektura operačního systému Windows XP [20]

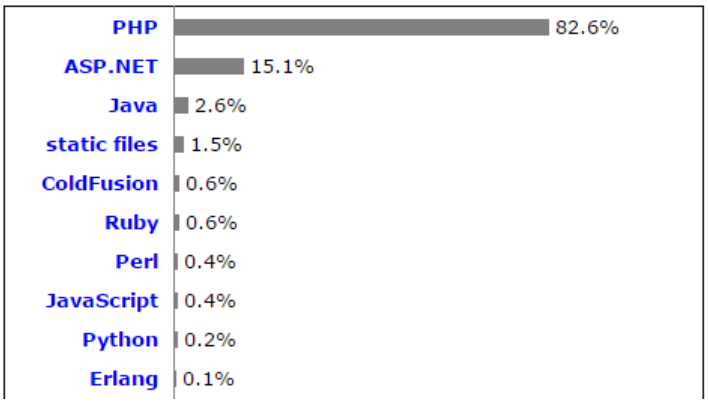
2.3 Historie webového vývoje

2.3.1 Statické webové stránky

Počátek webového vývoje začíná kolem roku 1991, kdy Sir Tim Berners-Lee dokončuje návrh a představuje protokol HTTP a první webový prohlížeč. [5] Jazykem pro tvorbu webových stránek se stal jazyk HTML, ačkoliv pro něj v té době ještě neexistovala schválená specifikace. Stránky pocházející z dané doby byly pouze statické, to znamená, že se všem uživatelům zobrazily vždy v podobě, v jaké byly uloženy na serveru. V roce 1995 poté došlo k první kompletní specifikaci jazyka HTML konsorciem W3C. [6]
K jazyku HTML se později v roce 1996 přidaly kaskádové styly (CSS), které společně s jazykem HTML a jazykem Javascript tvoří základní kameny všech moderních webových stránek.

2.3.2 Dynamické stránky

Příchod prvních dynamických webových stránek jako první umožnila technologie CGI (Common Gateway Interface) v roce 1993, díky které mohl být zobrazen výstup skriptu spuštěného na serveru (v té době se jednalo většinou o jazyk Perl).
V roce 1995 byl představen jazyk PHP (Personal Home Page Tools). Tento jazyk rychle získal na popularitě a v roce 1997 vyšla druhá verze tohoto jazyka obsahující podporu pro práci s databázemi nebo cookies. Zájem o dynamické stránky neustále stoupal a o rok později došlo k vydání další verze PHP (PHP3), v tuto chvíli se do vývojářského týmu přidávají další dva programátoři. V roce 2000 došlo k vydání čtvrté verze tohoto jazyka, již o rok později se PHP podle společnosti Netcraft [7] používalo na více než 1.8 miliónu stránkách. Ačkoliv jazyk ASP.Net existuje na trhu přibližně stejnou dobu (vydání v roce 1996 jako jazyk ASP, na ASP.NET přejmenováno v roce 2001) nikdy se mu nepodařilo PHP v rámci oblíbenosti dostihnout. Jak můžeme vidět na obrázku 2.4 je jazyk PHP k aktuálnímu datu (květen 2017) stále pětkrát používanější než jazyk ASP.NET.



Obrázek 2.4: Procentuální zastoupení programovacích jazyků na webových stránkách [8]

2.3.3 Aktivní prvky webových stránek

2.3.3.1 Javascript v letech 1995-2000

V roce 1995 představila společnost NetScape interpretovaný jazyk JavaScript. Kód tohoto jazyka byl vykonávan přímo na straně prohlížeče, což umožnilo přidat do webových stránek aktivní prvky.
K standardizaci Javascriptu pod názvem ECMAscript (název Javascript nemohl být z právních důvodů při standardizaci použit) došlo následně v roce 1997, v té době se již Javascript používal na více než 300 000 stránkách [9].

Později byla vydána specifikace ECMAScript 2, aby srovnala rozdíly mezi specifikacemi ISO a ECMA.
V prosinci roku 1999 se Javascript dočkal třetí standardizované specifikace ve formě ECMAScriptu 3, jež se na dlouhou dobu stala poslední standardizovanou verzí.

2.3.3.2 ECMAScript 3.1 & ECMAScript 4

S rostoucí popularitou Javascriptu rostly i nároky kladené na tento jazyk a proto se v roce 2000 začalo pracovat na čtvrté specifikaci ECMAScriptu. Dlouhodobé neshody ohledně budoucnosti jazyka však v roce 2003 vyústily v přerušení práce.
Díky rozmachu technologií jako AJAX nebo XMLHttpRequest byla práce na specifikaci ECMAScript 4 v roce 2005 obnovena. Do výboru pracujícím na nejnovější specifikaci patřily společnosti jako Adobe, Mozilla, Opera nebo Microsoft. [9]
Pracovní verze této specifikace se v průběhu dalších let rozrostla o mnoho změn a nových funkcí oproti původní specifikaci ECMAScript 3, na kterých se však výbor nedokázal shodnout. Práce na specifikaci byla po osmi letech finálně ukončena v roce 2008. Celá specifikace se zredukovala pouze na praktická zlepšení a byla vydána pod názvem ECMAScript 3.1.

2.3.3.3 Moderní Javascript

V roce 2009 došlo k přejmenování specifikace ECMAScript 3.1. na ECMAScript 5, jejíž podporu v průběhu dalších několika let přidaly všechny moderní prohlížeče.
Po uvedení této specifikace se v Javascriptu začaly vyvíjet celé aplikace, což vedlo k vývoji frameworků napsaných v tomto jazyce. Mezi nejpopulárnější z nich patří AngularJS (2010), Backbone.js (2010), Ember.js (2011), React (2013) a Vue.js (2014).

Javascript v dnešní době podporují všechny prohlížeče bez potřeby externích programů nebo pluginů a společně s HTML a CSS se stal základním stavebním kamenem moderních webových aplikací. [10]

Kapitola 3

Analýza

V následující kapitole analyzuji funkční a nefunkční požadavky na aplikaci, taktéž se budu zabývat analýzou současných řešení editoru a provedu analýzu způsobu grafického zobrazení editoru.

3.1 Funkční požadavky

- Vytvořit novou konfiguraci
- Načíst konfiguraci stávající
- Přidat do konfigurace nový modul
- Odebrat z konfigurace modul
- Změnit jméno modulu (Zajistit unikátnost jména v editované sestavě)
- Umožnit propojení 2 modulů, u kterých se shodují typy požadovaného a poskytovaného spojení
- Upravit konfiguraci modulu v textovém režimu
- Upravit konfiguraci modulu v grafickém režimu
- Na základě konfigurace se musí vygenerovat správný počet spojení
- Zobrazit vytvořenou konfiguraci
- Stáhnout vytvořenou konfiguraci
- Požadovaná a poskytovaná spojení musí být v kardinálním poměru 1 : N

Prohlížeč	**Procentuální zastoupení**
Google Chrome 1+	75.7 %
Firefox 4+	13.6 %
Internet Explorer 11+	4.4 %
Safari 3+	3.7 %
Opera 9+	1.1 %
Ostatní	1.5 %

Tabulka 3.1: Zastoupení prohlížečů na trhu [15], [16]

- K propojení poskytovaných a požadovaných spojení nesmí dojít v rámci jednoho modulu
- Při propojování modulů musí být zřejmé, které moduly lze propojit.

Nefunkční požadavky

- Webová aplikace
- Podpora všech moderních prohlížečů (IE 11+, Firefox 4+, Chrome 1+, Opera 9+, Safari 3+)

Z dat v tabulce 3.1 vyplývá, že moderní prohlížeče (jejichž podpora je zadána nefunkčním požadavkem) v aktuální době používá 98.5 % uživatelů internetu.

3.2 Analýza současných řešení

V současnosti neexistuje žádný univerzální editor konfigurací modulárních systémů, který by dokázal skloubit požadovanou genericitu s dostatečnou uživatelskou přívětivostí. Pro většinu modulárních systémů tedy vznikají editory přímo určené k editacím daného systému.
V této části se budu tedy zabývat analýzou dostatečně generických řešení vyhovujících stanoveným požadavkům.

3.2.1 Jednoduchý textový editor

Nejjednoduším způsobem, jak editovat XML konfigurace je jakýkoliv jednoduchý textový editor.
Mezi hlavní výhody patří jednoduchost, minimalistické uživatelské rozhranní a s tím

spojená svižnost aplikace.
Použití jednoduchého textového editoru však s sebou nese větší množství nevýhod. Kompletně chybí jakékoliv pomocné mechanismy, které by uživateli naznačovaly, zda neudělal chybu a zda bude výsledný XML dokument validní.
Dále toto řešení vyžaduje znalost názvů konfiguračních tagů a atributů, nebo souběžně otevřenou projektovou dokumentaci.

3.2.2 Textový editor podporující XML syntax highlighting

Editory podporující XML syntax highlighting (česky zvýraznění syntaxe) ulehčují práci s XML konfiguracemi hlavně validací psané konfigurace oproti platné syntaxi jazyka XML.
Případné chyby jsou uživateli poté barevně zvýrazněny. Další výhodou zvýraznění syntaxe je zdánlivě jednodušší orientace v psané konfiguraci.
Většina editorů podporujících XML syntax highlighting taktéž umožňují automatické odsazení psaného textu, což ještě více zpříjemňuje psaní dané konfigurace.
Ačkoliv tyto editory kloubí jednoduchost textových editorů s jistou uživatelskou přívětivostí, stále vyžadují znalost názvů konfiguračních tagů a atributů.

3.2.3 Textový editor umožňující souběžnou kontrolu s definičním schématem (XSD)

Editory podporující souběžnou kontrolu konfigurace s definičním schématem částečně řeší problém s potřebou znalosti názvu konfiguračních tagů a atributů. Většinou se jedná o rozšíření funkce zvýrazňování syntaxe o grafické zvýraznění chyb ve validaci s definičním schématem.
V lepším případě poskytuje editor funkci tzv. našeptávání, kdy uživateli dává na výběr z možných tagů a atributů, které se na daném místě mohou objevit. Tyto funkce dohromady přináší dobrou uživatelskou přívětivost a eliminují většinu možných chyb při psaní konfigurace.
Toto řešení je vhodné pro konfigurace menších rozměrů. Bohužel s narůstající komplexností konfigurace je téměř nemožné se v její textové interpretaci orientovat. Hlavní nevýhodou těchto editorů tedy je obtížná orientace v komplexních konfiguracích.

3.3 Obsah deskriptoru

V sekci 2.1.9 jsem zmínil, co deskriptor je a jaké informace musí obsahovat. Je taktéž jasné, že bude deskriptor ve formě XML. V této kapitole rozeberu jednotlivé prvky, které se mohou v deskriptoru vyskytnout.

3.3.1 Descriptor

Povolené atributy:

- ProjectName – Povinný atribut. Celé jméno projektu, ke kterému deskriptor náleží
- RootElementName – Povinný atribut. Jméno elementu, do kterého se mají vkládat veškeré informace o projektu

Povolené elementy:

- Modules – Povinný element. Musí obsahovat právě jednou.

Jedná se o kořenový element daného souboru. Musí se v konfiguraci vyskytovat právě jednou.

3.3.2 Modules

Povolené atributy: *Neobsahuje žádné atributy.*
Povolené elementy:

- Module – Povinný element. Musí obsahovat alespoň jednou.

Element obsahující informace o všech modulech, které se v konfiguraci nachází.

3.3.3 Module

Povolené atributy:

- Name – Povinný atribut. Jméno modulu. V dané konfiguraci musí být unikátní.

Povolené elementy:

- ConsumingConnections – Nepovinný element. Může obsahovat maximálně jednou.
- ProvidingConnections – Nepovinný element. Může obsahovat maximálně jednou.

- Parameters – Nepovinný element. Může obsahovat maximálně jednou.

Obsahuje veškeré informace o jednom určitém modulu.

3.3.4 ConsumingConnections

Povolené atributy: *Neobsahuje žádné atributy.*
Povolené elementy:

- ConsumingConnection – Nepovinný element. Počet není omezen.
- GeneratedConnection – Nepovinný element. Počet není omezen.

Obsahuje seznam všech vyžadovaných spojení. Tato spojení mohou být buď pevná (ConsumingConnection), nebo generovaná (GeneratedConnection).

3.3.5 ProvidingConnections

Povolené atributy: *Neobsahuje žádné atributy.*
Povolené elementy:

- ProvidingConnection – Nepovinný element. Počet není omezen.
- GeneratedConnection – Nepovinný element. Počet není omezen.

Obsahuje seznam všech poskytovaných spojení. Tato spojení mohou být buď pevná (ProvidingConnection), nebo generovaná (GeneratedConnection).

3.3.6 ConsumingConnection

Povolené atributy:

- Type – Povinný atribut. Typ požadovaného spojení.
- Port – Nepovinný atribut. Port požadovaného spojení.
- Optional – Nepovinný atribut (Výchozí hodnota: Ano). Přepínač určující, zda je dané spojení povinné nebo nepovinné.

Povolené elementy: *Neobsahuje žádné elementy.*
Obsahuje informace o jednom požadovaném spojení.
Požadovaná spojení se v rámci jednoho modulu rozlišují pouze na základě typu.
V případě existence více spojení stejného typu je nutné pro rozlišení spojení vyplnit i port spojení.

3.3.7 ProvidingConnection

Povolené atributy:

- Type – Povinný atribut. Typ poskytovaného spojení.
- Port – Nepovinný atribut. Port poskytovaného spojení.

Povolené elementy: *Neobsahuje žádné elementy.*
Obsahuje informace o jednom poskytovaném spojení.
Poskytovaná spojení se v rámci jednoho modulu rozlišují pouze na základě typu.
V případě existence více spojení stejného typu je nutné pro rozlišení spojení vyplnit i port spojení.

3.3.8 GeneratedConnection

Povolené atributy:

- Type – Povinný atribut. Typ generovaného spojení.
- SyncedProperty – Povinný atribut. Cesta v konfiguraci modulu, na základě které má k generování spojení dojít.
- MaxCount – Nepovinný atribut (Výchozí hodnota: Neomezeno). Maximální počet spojení, které se mohou tímto vztahem vygenerovat.

Povolené elementy: *Neobsahuje žádné elementy.*
Obsahuje informace o jednom generovaném spojení.
Počet vytvořených spojení tohoto typu se určí na základě konfigurace modulu. Z tohoto důvodu se v *SyncedProperty* očekává platná cesta v konfiguraci modulu ve tvaru */CESTA_V_XML/JMÉNO_ELEMENTU@SYNC_JMÉNO:SYNC_PORT*.
CESTA_V_XML zastupuje posloupnost elementů oddělenou lomítky skrze které je třeba se zanořit k získání seznamu elementů, na základě kterých má dojít k generování.
JMÉNO_ELEMENTU zastupuje jméno elementu na základě kterého budou spojení

generována.
SYNC_JMÉNO zastupuje jméno atributu daného elementu, jehož hodnota má být použita jako jméno generovaného spojení.
SYNC_PORT zastupuje jméno atributu daného elementu, jehož hodnota má být použita jako port generovaného spojení.

3.3.9 Parameters

Povolené atributy: *Neobsahuje žádné atributy.*
Povolené elementy:

- Parameter – Povinný element. Musí obsahovat právě jednou.

Obsahuje odkaz na kořenový prvek interní konfigurace modulu.

3.3.10 Parameter

Povolené atributy:

- Name – Povinný atribut. Jméno parametru.
- Path – Nepovinný atribut. Cesta zanoření parametru.
- MaxCount – Nepovinný atribut (Výchozí hodnota: 1). Udává maximální počet výskytů v rámci aktuální cesty v konfiguraci.
- PropertyType – Nepovinný atribut. Pokud je potomkem elementu hodnota, udává typ této hodnoty.

Povolené elementy:

- Parameter – Nepovinný element. Počet není omezen.
- Property – Nepovinný element. Počet není omezen.

Obsahuje informace o jednom prvku v interní konfiguraci modulu.
Může obsahovat cestu zanoření elementu. Tento atribut může být využit v případě, že se cestou k aktuálnímu elementu nachází pouze elementy bez vlastních atributů, mající za potomka vždy pouze následující element v cestě. Pokud je vyplněn *PropertyType*, jedná se o koncový typ elementu a již nesmí obsahovat žádné další vnořené elementy typu *Parameter*.

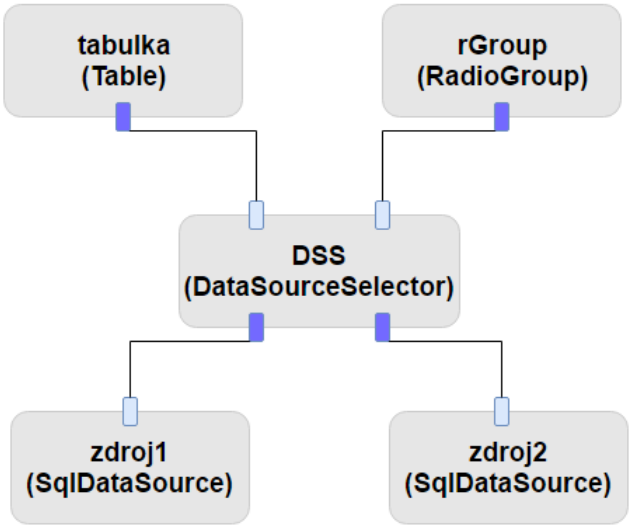
3.3.11 Property

Povolené atributy:

- Name – Povinný atribut. Jméno atributu.
- PropertyType – Povinný atribut. Typ atributu.

Povolené elementy: *Neobsahuje žádné elementy.*
Obsahuje informace o jednom atributu náležicím elementu definovaným rodičem tohoto prvku.

3.4 Způsob zobrazení



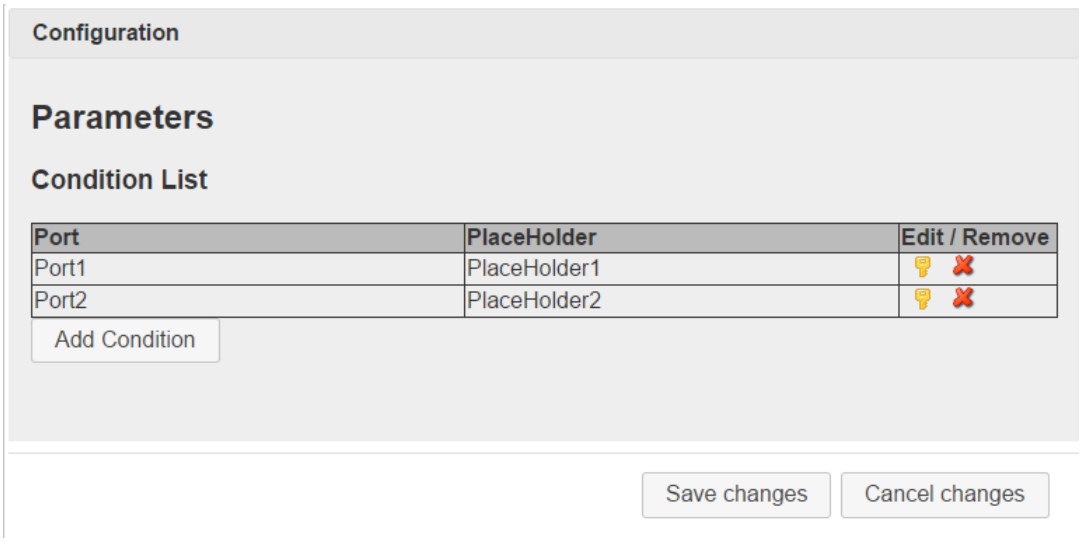
Obrázek 3.1: Návrh zobrazení modulů a příslušných spojení

3.4.1 Zobrazení modulů a spojení

Jedním z požadavků na editor bylo, že má nabídnout přívětivé grafické uživatelské rozhranní. Bylo potřeba však zvolit formát, ve kterém se uživateli zobrazí. Jelikož je konfigurace souborem modulů, které jsou vzájemně propojeny, je logické zobrazit konfiguraci jako graf.

Prvním návrhem bylo zobrazit všechny moduly jako uzly a spojení mezi moduly jako hrany. Bylo však potřeba zvolit způsob, jak rozpoznávat jednotlivé hrany od sebe. Ideálním stavem by bylo, kdyby se dal typ spojení určit již v době vytváření daného spojení. Na tuto část se ukázalo jako nejvhodnější vytvořit pro každé spojení další uzel grafu. Vzájemné propojení by uživatel tedy neprováděl přímo mezi uzly dvou modulů, ale přímo mezi uzly daných spojení. Každý uzel modulu musí uživateli viditelně zobrazit typ a jméno interpretovaného modulu. K tomuto účelu je nejvhodnější vložit text přímo do uzlu patřícího k danému modulu. Návrh způsobu, jakým by mohly být moduly zobrazeny je zobrazen na obrázku 3.1.

3.4.2 Zobrazení grafického editoru konfigurace modulů



Obrázek 3.2: Návrh zobrazení modálního okna na editaci konfigurace modulů

Z funkčních požadavků vyplývá, že pro editaci konfigurací modulů budeme potřebovat jak okno podporující textovou editaci, tak editaci skrze grafické prostředí. Jako nejvhodnější řešení se jeví modální okno obsahující textová pole na editaci příslušných atributů, s případnými tlačítky na přidávání/odebírání/úpravu potomků. Návrh zobrazení tohoto dialogového okna je zobrazen na obrázku 3.2.

Kapitola 4

Výběr technologií

Tato kapitola se věnuje technologiím použitým v aplikaci. Taktéž zde budu u řady technologií analyzovat jejich výhody a nevýhody a určím důvody, proč si myslím, že se jedná o správnou volbu pro tuto aplikaci.

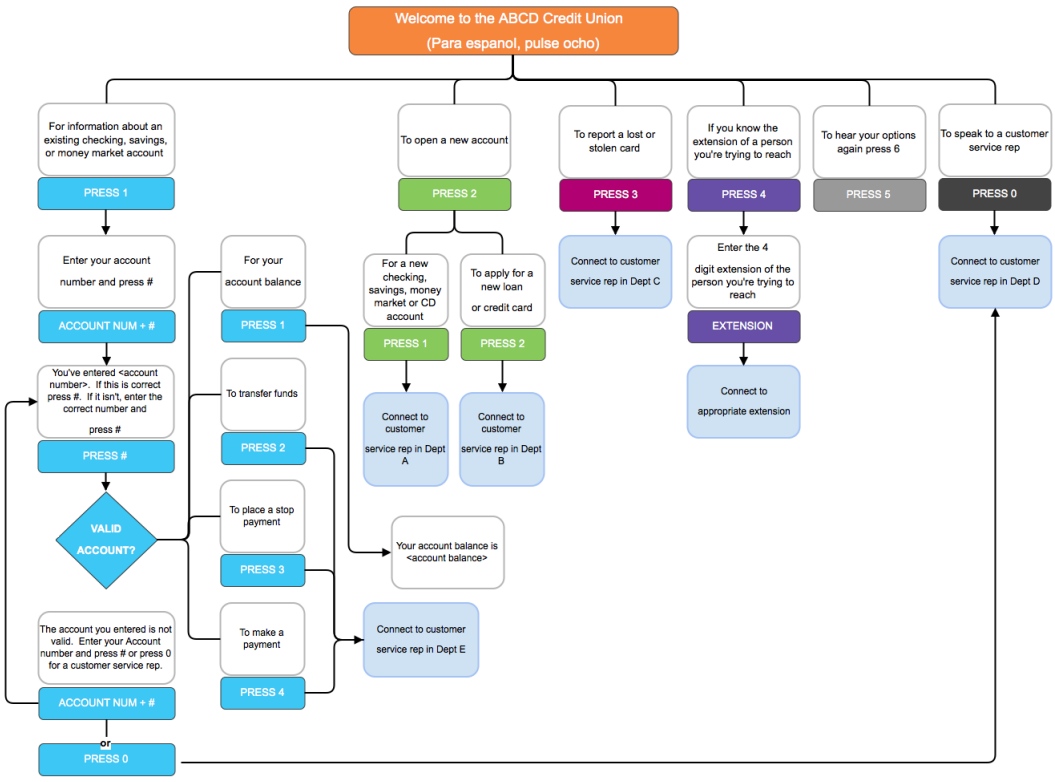
4.1 Jazyk pro tvorbu klientské aplikace

Nejprve bylo potřeba pro vývoj zvolit správný jazyk. Jazyk musel splňovat nefunkční požadavek běhu v každém moderním webovém prohlížeči. Prvním jasným kandidátem je jazyk Javascript, který se (společně s jeho dalšími implementacemi) v dnešní době používá pro vývoj téměř všech aplikací běžících ve webovém prohlížeči.
Mezi jeho hlavní výhody patří, že je součástí každého moderního prohlížeče, pokud není explicitně vypnutý uživatelem.
Dalšími jazyky nebo frameworky umožňujícími spouštění aktivních skriptů ve webovém prohlížeči jsou Silverlight nebo Macromedia Flash. Ani jeden z těchto jazyků však nemá přímou podporu ve webovém prohlížeči a jejich funkčnost závisí na externích programech napojených na prohlížeče přes Netscape Plugin Application Programming Interface (NPAPI).
K dnešnímu dni již NPAPI nepodporuje ani jedna z nejnovějších verzí prohlížečů uvedených v nefunkčních požadavcích. [11] [12] [13]
Z tohoto důvodu nebudu tyto frameworky při výběru ani zohledňovat, jelikož nesplňují požadavky zadavatele. Pro vývoj aplikace jsem tedy zvolil jazyk Javascript.

4.2 Výběr knihovny pro práci s grafy

Z analýzy vyplývá, že pro dostatečnou uživatelskou přívětivost by aplikace měla umožňovat zobrazení konfigurace v grafickém rozhraní. Jako nejrozumnější řešení se zdá zobrazení konfigurace jakožto grafu, kde je každý modul reprezentován jako uzel grafu a propojení mezi moduly jakožto hrana. Jelikož by byla tvorba celého vlastního frameworku pro efektivní zobrazení a práci s grafy nad rámec této práce, bylo potřeba zvolit nějaké již hotové řešení.

4.2.1 mxGraph



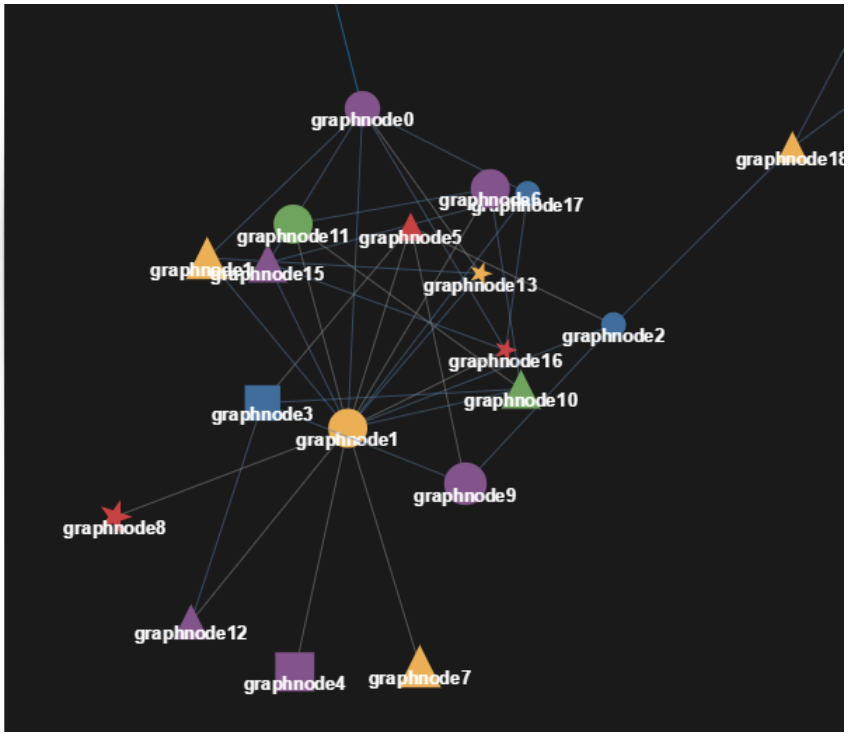
Obrázek 4.1: Ukázka grafu vytvořeného za pomocí knihovny mxGraph [17]

- Dostupné na Github od: 2012
- Počet přispěvatelů za posledních 12 měsíců: 5
- Počet commitů za posledních 12 měsíců: 31
- Počet stars na GitHub: 703

- Počet otevřených / uzavřených Issues: 10/48

Knihovna, která byla v letech 2005-2016 populárním komerčním řešením pro větší projekty pracujícími s grafy.
Její hlavní výhodou v té době byla podpora prohlížečů, které neměly podporu pro SVG. V roce 2016 byla uvolněna jako opensource.
Mezi její hlavní výhody patří jednoduchost rozšíření již existujících funkčních celků, ze kterých se knihovna skládá.
V její prospěch mluví i velké množství ukázek práce s knihovnou poskytnutých přímo tvůrci knihovny a dostatečně velká komunita zajišťující neustálý vývoj.
Jako hlavní nevýhodou se již na první pohled zdá být nekvalitní programátorská dokumentace, která značně znepříjemňuje případnou práci s knihovnou.

4.2.2 JavaScript InfoVis Toolkit



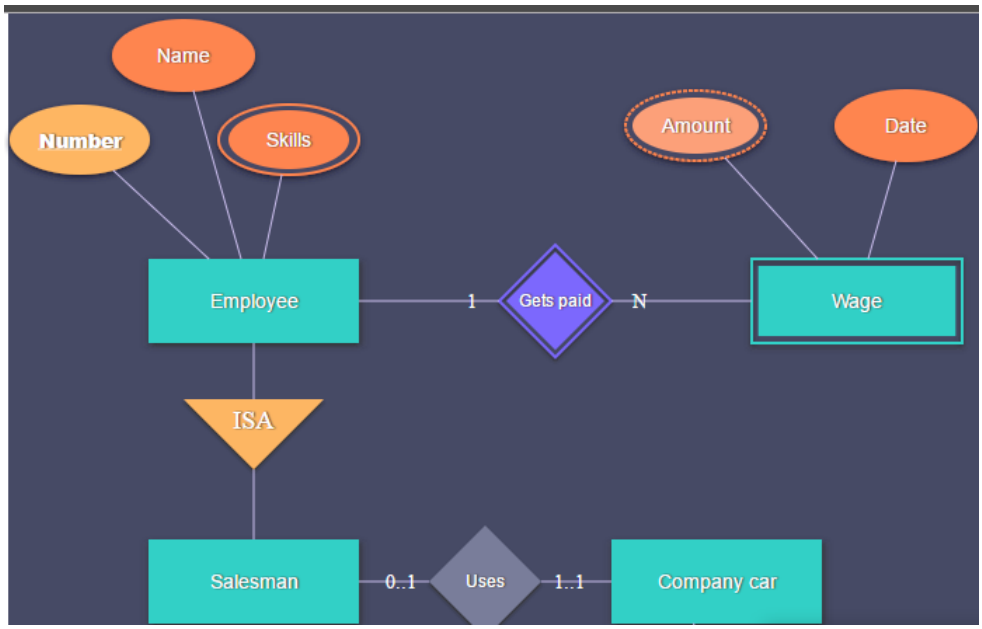
Obrázek 4.2: Ukázka grafu vytvořeného za pomocí knihovny JavaScript InfoVis Toolkit [18]

- Dostupné na Github od: 2009
- Počet přispěvatelů za posledních 12 měsíců: 0

- Počet commitů za posledních 12 měsíců: 0
- Počet stars na GitHub: 1380
- Počet otevřených / uzavřených Issues: 82/59

Na první pohled se jedná o knihovnu, která slouží zejména pro tvorbu běžných diagramů neobsahujících uzly a hrany.
Mezi její hlavní výhody patří velké množství již zabudovaných moderních stylů.
Pro výběr knihovny taktéž hovoří oblíbenost knihovny mezi programátory (1380 stars na GitHub). Avšak pro účely vytvářeného editoru není knihovna vyhovující a vyžadovala by mnoho úprav. Nevýhodou této knihovny je taktéž zastavený vývoj. (Poslední commit 28.9.2014) [14]

4.2.3 JointJS



Obrázek 4.3: Ukázka grafu vytvořeného za pomocí knihovny JointJS [19]

- Dostupné na Github od: 2009
- Počet přispěvatelů za posledních 12 měsíců: 14
- Počet commitů za posledních 12 měsíců: 163
- Počet stars na GitHub: 2 132

- Počet otevřených / uzavřených Issues: 68/215

Knihovna, která je jádrem populárního komerčního HTML5 frameworku pro tvorbu diagramů *Rappid*. JointJS nabízí jednoduchý způsob, jak vytvořit vzhledné diagramy jakéhokoliv typu. Ve svém základě obsahuje velké množství základní prvků, které lze nadále rozšiřovat.
Společně se zdrojovým kódem nabízí knihovna pár ukázkových prací. Na stránkách tvůrce můžeme taktéž nalézt několik návodů na práci s knihovnou.
Jednou z hlavních výhod knihovny je aktivní vývoj (163 commitů za posledních 12 měsíců).
Jako jedinou nevýhodu vidím nedostatečné množství ukázkových prací s knihovou.

4.2.4 Výběr

Ze tří zvolených knihoven se knihovna JavaScript InfoVis Toolkit jevila na první pohled jako nejslabší. Jelikož z velké části není uzpůsobena pro účely vytvářeného editoru. Zbylé dvě analyzované knihovny si jsou velice podobné. Obě knihovny mají horší programovou dokumentaci, jsou aktivně vyvíjeny a pro účely vytvářeného editoru jsou obě vyhovující. Ačkoliv je JointJS oblíbenější mezi uživateli, rozhodl jsem se použít knihovnu mxGraph, hlavně kvůli velkému množství ukázek práce s knihovnou.
Zvolena tedy byla knihovna mxGraph.

4.3 Výběr knihovny pro textovou editaci konfigurace

Další důležitou funkcí editoru je možnost přepnutí editace do textového režimu, kde bude možné konfigurace upravovat v jazyce XML. I zde existuje několik možných řešení.

4.3.1 Textarea element

Jakožto jedno z nejjednoduších řešení je vytvoření HTML textarea elementu a umožnění úprav přímo v něm. Toto řešení bohužel postrádá jakékoliv prvky uživatelské přívětivosti. Chybí automatické doplňování tagů, zvýraznění syntaxe nebo automatické odsazování.

4.3.2 Ace Editor

- Dostupné na Github od: 2010

- Počet přispěvatelů za posledních 12 měsíců: 9
- Počet commitů za posledních 12 měsíců: 81
- Počet stars na GitHub: 14 789
- Počet otevřených / uzavřených Issues: 529/1353

Editor vyvíjený společnostmi Cloud9 IDE a Mozilla. Jedná se o nástupce projektu Mozilla Skywriter. V současnosti jej v jistých podobách využívá velké množství populárních projektů jako třeba Github, Khan Academy, Codeacademy nebo Wikipedia. Mezi jeho hlavní přednosti patří podpora zvýrazňování syntaxe pro více než 110 známých jazyků, velké množství témat, automatické odsazování nebo podpora regulárních výrazů. Pro JavaScript, CoffeeScript, CSS, XQuery nabízí automatickou kontrolu syntaxe. Díky aktivní podpoře od dvou velkých společností se nepochybně jedná o jeden z nejaktivněji vyvíjených opensource webových editorů. Jako hlavní nevýhodu editoru vidím chybějící podporu pro zásah do automatického doplňování (toto napovídání funguje v editoru na základě doplňování slov, které již uživatel napsal v okolí dané části kódu) a přehnanou komplexnost editoru pro účely naší aplikace.

4.3.3 Eclipse Orion

- Dostupné na Github od: 2011
- Počet přispěvatelů za posledních 12 měsíců: 19
- Počet commitů za posledních 12 měsíců: 1210
- Počet stars na GitHub: 15
- Počet otevřených / uzavřených Issues: 0/0

Project Eclipse Orion vznikl s myšlenkou přesunout celou infrastrukturu vývoje pro web, na web. Na vývoji editoru se podílí zejména firma IBM.
Editor je součástí celéhou client-server řešení pro cloudové IDE. Jako svou hlavní výhodu uvádí Eclipse nezávislost na záložce, ve které je editor spuštěn. To znamená, že oproti ostatním editorům umožňuje funkcionalitu jako `Otevřít soubor na nové kartěı. Zvýrazňování syntaxe umí editor ve zvolených asi 10 jazycích běžně užívaných na webu, mezi nimiž se však bohužel XML nenachází. Chybí zde i podpora pro párování XML tagů.

4.3.4 Code Mirror

- Dostupné na Github od: 2011
- Počet přispěvatelů za posledních 12 měsíců: 24
- Počet commitů za posledních 12 měsíců: 456
- Počet stars na GitHub: 11 628
- Počet otevřených / uzavřených Issues: 178/2520

Opensource editor aktivně udržován hlavně jeho tvůrcem. Podobně jako Ace Editor, i Code Mirror se používá ve velkém množství populárních projektů jako jsou třeba Bitbucket, Chrome DevTools nebo JSFiddle. Zvýrazňování syntaxe zvládá taktéž ve více než 100 jazycích, obsahuje velké množství témat, i automatické odsazování a konfigurovatelné automatické doplňování kódu. Jako jednu z hlavních nevýhod tohoto editoru vidím jistotu přetrvávající podpory. Za správou editoru nestojí žádná společnost a i přes velké množství přispěvatelů je zde pouze jeden správce. Ačkoliv se projektu věnuje již 6 let stabilním tempem, mohla by budoucnost projektu ohrozit například nehoda.

4.3.5 Výběr

Code Mirror i Ace Editor jsou hojně používané, dlouhou dobu vyvíjené a v rámci poskytované funkcionality víceméně podobné. Eclipse Orion se i přes usilovnou práci všech zúčastněných bohužel takové popularity nedočkal. Pro účely zvolené aplikace potřebujeme zejména podporu pro jazyk XML, které Code Mirror i Ace Editor obsahují.
Již tímto se Eclipse Origin jeví jako nevhodný kandidát pro účely vytvářeného editoru a dále už nebude posuzován.
Pro jazyk XML oba editory poskytují zvýrazňování syntaxe nebo automatické odsazení dle zanoření elementu. Výhodou CodeMirror se pro účely vytvářeného editoru zdá býti možnost konfigurovatelného automatického doplňování (ačkoliv není za pomoci XSD, ale specifických JSON objektů). U Ace Editoru by se dalo jako výhodu určit stabilní zázemí editoru v podobě dvou velkých společností stojících za projektem.
Při porovnání dat z GitHubu se jako vhodnější jeví Code Mirror. Na CodeMirror se pracuje aktivněji než na Ace Editoru, což se odráží i v počtech vyřešených/nevyřešených Issues. Poměr 529/1353 u Ace Editoru evokuje pocit, že se vývojáři dostatečřně nevěnují zpětné vazbě od uživatelů editoru, což vzbuzuje obavy, že by nalezená chyba nemusela být řádně adresována a řešena dostatečně rychle.
Pro volbu CodeMirror dále mluví i automatické doplňování kódu dle XSD (nebo dle formátu, který lze z XSD generovat), jelikož je tato funkčnost pro nás podstatným

prvkem našeho uživatelského rozhraní. Ačkoliv by tuto funkčnost bylo možné pro Ace Editor doprogramovat, ani tento editor se nejeví jako nejvhodnější pro účely naší aplikace.
Jako výsledný editor byl tedy zvolen CodeMirrror.

Kapitola 5

Návrh architektury

Před samotnou implementací bylo potřeba rozhodnout, jak bude celý program fungovat. Důležitým faktorem bylo, aby byl editor co nejjednodušeji rozšířitelný do budoucna. I na to bylo potřeba při tvorbě architektury přihlížet. Právě z toho důvodu jsem se při tvorbě architektury soustředil na vysokou modularitu celého systému.

5.1 Adaptér

Jak je z analýzy zřejmé, pro každý projekt je potřeba získat deskriptor, tedy soubor obsahující informace o všech použitelných modulech v daném projektu, možnostech jejich vzájemného propojení a konfigurace.
Tento soubor bude poskytovat externí služba, o které bude editor vědět. Jelikož jedním z funkčních požadavků je zobrazení a stažení výsledné konfigurace, je třeba, aby tato externí služba zvládala i převod z interní interprece konfigurace na projektovou.
Opačnou konverzi potřebujeme též a to pro splnění funkčního požadavku načítání a následné úpravy již existující konfigurace. Tato externí služba (Dále jen Adaptér) bude jakousi abstraktní vrstvou, která tyto 3 funkce bude vystavovat na nějaké URL a program se na ně bude dotazovat pomocí GET/POST požadavků.
Je možné, že se Adaptéry pro rozdílné projekty budou nacházet na rozdílných URL, proto nesmí být adresa na kterou se bude program dotazovat pevně daná.

5.2 Rozdělení zodpovědností v klientu

Pro urychlení vývoje a zjednodušení lazení kódu je vhodné kód rozdělit do funkčních celků, které spolu budou vzájemně komunikovat.

První je dobré oddělit z kódu helpery (česky pomocné třídy).
V kódu naší aplikace se budou nacházet následující pomocné třídy:

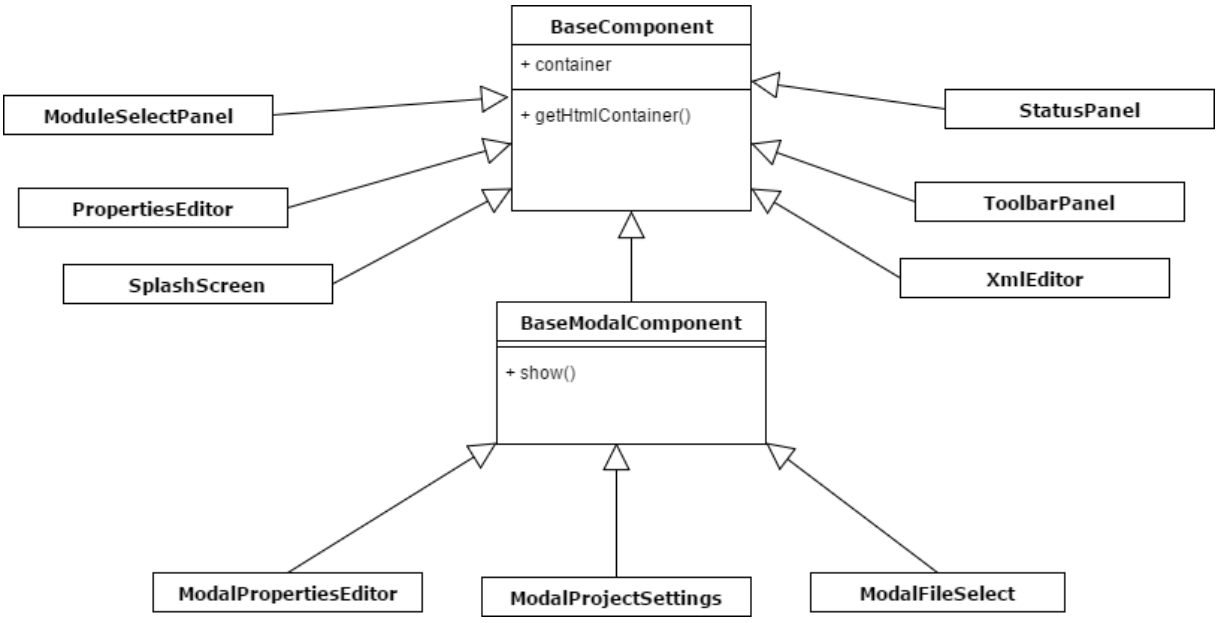
- Editor Control Helper – Třída umožňující přístup k funkcím mxEditoru, umožňuje přidávaní nových akcí do editoru nebo získání editovaného grafu.
- Graph Control Helper – Třída umožňující přístup k funkcím editovaného grafu. Obsahuje například funkce na přidávání a odebírání uzlů, různé pomocné funkce nebo přístup k modelu grafu.
- Module Models Helper – Třída umožňující přístup k seznamu modelů a jejich možnému propojení, dále se třída stará o správné vyčtení dat z poskytnutého deskriptoru.
- Mx Control Helper – Třída umožňující přístup k položkám knihovny mxGraph.
- UI Helper – Třída umožňující generování UI elementů pro použití v různých částech aplikace.
- Utils – Třída obsahující soubor obecných pomocných metod.
- XML Hint Filter – Třída nahrazující původní třídu pro napovídání od CodeMirror.

Každá pomocná třída dědí od třídy *BaseHelper*. Od této třídy získávají pomocné třídy zástupnou implementaci inicializační metody v případě, že neimplementují svou vlastní. Krom pomocných tříd se v rámci aplikace budou vyskytovat i služby

- Module Data Provider – Třída udržující informace o všech modulech v aplikaci.
- XHR Service – Třída umožňující odeslání různých XHR požadavků.
- XML Composing Service – Třída zajišťující správný přepis upravovaných informacích o modulech do podoby XML dokumentu.

Každá služba dědí od třídy *BaseService*. Podobně jako v případě pomocných tříd získávájí služby od této třídy zástupnou implementaci inicializační metody v případě, že neimplementují svou vlastní.

Krom služeb a pomocných tříd bude editor rozdělen do jednotlivých komponent. Tyto komponenty se budou dělit do dvou částí. Klasické komponenty a modální okna. Klasické komponenty dědí od třídy *BaseComponent*, tato třída obsahuje jedinou třídu a tou je *getHtmlContainer*, která očekává existenci kontejneru v rámci dané třídy, který může zavoláním zpřístupnit. Modální okna dědí od třídy *BaseModalComponent*, která je rozšířením *BaseComponent* o metodu umožňující zobrazení dialogového okna.



Obrázek 5.1: Graf znázorňující vztahy mezi jednotlivými komponentami

5.3 Proof of concept

Před tvorbou samotného editoru bylo potřeba vytvořit takzvaný Proof of concept, neboli prototyp editoru, aby bylo jisté, že ve zvolených technologiích lze projekt uskutečnit. Funkční požadavky, kterých musel prototyp dosáhnout byly následující:

- Přidání modulu
- Odebrání modulu
- Změna jména modulu (zajištění unikátnosti jména v editované sestavě)
- Zobrazení XML výstupu
- Umožnit propojení 2 modulů, u kterých sedí typy požadovaného a poskytovaného spojení
- Providing a Consuming connections musí být v kardinálním poměru 1 : N
- K propojení poskytovaných a požadovaných spojení nesmí dojít v rámci jednoho modulu
- Každý modul může, ale nemusí, obsahovat konfiguraci v podobě XML tagu.

Při implementaci těchto funkčních požadavků prototypu jsem narazil s knihovnou pouze na dva problémy, které popíšu níže.

5.3.1 Zajištění unikátnosti jména

Pro udržení validity konfigurace bylo potřeba zajistit unikátnost jména modulu v dané konfiguraci. Na tento problém editor naráží ve dvou případech:

- Při přidání nového modulu do konfigurace
- Při změně jména modulu uživatelem

Prvním možným řešením by bylo generovat jméno nového modulu náhodně. Toto řešení by však nesplnilo požadavek zaručené unikátnosti a ani tento způsob nejde použít při přejmenování modulů uživatelem.
Jako nejrozumnější řešení se jevilo vytvoření služby, která by u sebe udržovala informace o všech modulech, které se aktuálně nachází v aplikaci. Díky tomu může tato služba poskytovat informaci o dostupnosti zvoleného jména.
Při vytváření nového modulu je jako jeho prvotní jméno zvolena kombinace typu daného modulu a pořadového čísla modulu s daným typem v dané konfiguraci.

5.3.2 Propojení dvou modulů

Důležitým problémem, s kterým bylo potřeba se vypořádat, bylo propojení dvou modulů. Moduly se dají vzájemně spojit pouze v případě, že jeden modul poskytuje typ spojení, které druhý vyžaduje. Navíc je zde potřeba počítat se vztahem 1:N, tedy že k jednomu požadovanému spojení může být přiřazeno maximálně jedno spojení poskytované.
Zvolená knihovna sama o sobě v době psaní této práce neposkytuje žádnou funkčnost, která by umožňovala modulu (interpretovanému jako uzel) přiřadit přesně dané porty, skrze které by šlo propojování provádět. Toto lze však jednoduše obejít přidáním portů jakožto několika dalších uzlů, které jsou relativně pozicované oproti modulu, ke kterému náleží. Jediné, co zbývalo zajistit bylo ujištění se, že půjdou porty propojit pouze s opačným druhem spojení stejného typu.
Jednoduchou implementaci této funkčnosti již knihovna mxGraph obsahovala a to mxMultiplicity. Tato třída dokáže přidat podmínky na počet a typ uzlů připojených k aktuálnímu uzlu. Díky tomu se třída stala ideálním kandidátem na poskytnutí funkčnosti, kterou my od editoru požadujeme. Bohužel však nebylo s touto třídou možné vyplnění třetí podmínky na vzájemná spojení modulů, a to, že k vzájemnému propojení požadovaného a poskytovaného spojení nesmí dojít v rámci jendoho modulu.
Jako řešení jsem nakonec zvolil překrytí interní funkce knihovny mxGraph getEdgeValidationError. Tato funkce má za úkol zajistit, aby nešla vytvořit neplatná hrana mezi dvěma uzly, což je přesně chování, kterého bylo potřeba dosáhnout. Stačilo tedy prototyp této funkce rozšířit o následující 3 kontroly :

- Typy obou spojení se rovnají.
- Pokud je jeden z uzlů je označen jako poskytované spojení, druhý nikoliv.
- Oba dva uzly jsou relativně pozicované proti jinému modulu.

5.3.3 Závěr

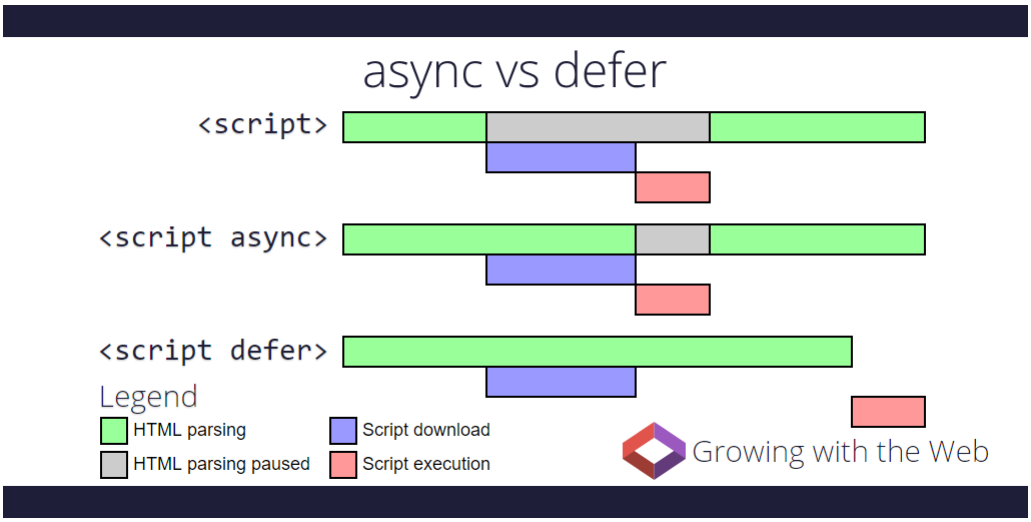
Krom již zmíněných dvou problémů nebyla implementace prototypu nijak problematická a s knihovnou mxGraph se podařilo vyhovět všem funkčním požadavkům kladených na Proof of Concept zmíněného editoru.
Jelikož se knihovna ověřila jako vhodná pro tvorbu editoru, bylo možné, na základě prototypu, začít budovat editor.

5.4 Implementace

Po dokončení prototypu editoru bylo možné začít s implementací editoru.

5.4.1 Inicializace editoru

Pro správné fungování systému bylo potřeba, aby se editor správně inicializoval.
K této inicializaci musí dojít poté, co víme, že byly staženy všechny potřebné skripty a již došlo k vykreslení DOM.
K tomu je ideální použít atribut *onload* u elementu *body*, společně s atributem *defer* u elementu *script*. Atribut *defer* zajistí, že ke spuštění skriptu dojde až na úplném konci, tedy poté, co se načte a dokončí parsování zbytku stránky. (Fungování atributu defer blíže přibližuje obrázek 5.4).
Atribut *onload* u elementu *body* poté zajistí spuštění příslušné inicializační metody po dokončení načítání skriptu.
První dochází k inicializaci služeb. Jelikož každá služba dědí od třídy *BaseService*, víme, že obsahuje metodu *init*, kterou můžeme bezpečně zavolat. Tím dojde k potřebné inicializaci služeb.
Po inicializaci služeb najde editor ve druhém kroku příslušné části dokumentu, do kterých má editační rozhraní vykreslit. Dojde k inicializaci jednotlivých komponent a počátečnímu vykreslení grafického rozhraní editoru.
V následujícím kroku dochází ke konfiguraci knihovny pro práci s grafy, čímž dojde i k přípravě na její prvotní načtení.
V posledním kroku načte editor seznam všech adaptérů z přiloženého souboru a vyzve uživatele k výběru adaptéru, ke kterému se chce připojit.



Obrázek 5.2: Vliv atributů async a defer na zpracovávání stránky [21]

Po výběru adaptéru dochází k připojení k adaptéru a stažení deskriptoru projektu. Deskriptor je následně předán příslušné pomocné třídě, která se nadále postará o jeho zpracování. Následně již dochází k načtení modulů a otevření editačního okna.

5.4.2 Generovaná spojení

Jedním z hlavních problémů, s kterými se musí editor potýkat jsou generovaná spojení. Jak již bylo zmíněno výše, spojení může být buď pevné nebo generované. V případě, že je spojení generované, je počet spojení daného typu závislý na aktuální konfiguraci modulu.
K tomu, abychom věděli, kolik generovaných spojení máme vytvořit a s jakými parametry se mají propsat do výsledného XML, potřebujeme dodatečné informace.
Oproti pevným spojením obsahují generovaná spojení, kromě názvu a typu spojení, ještě informaci o umístění v konfiguraci (zanoření elementů oddělené lomítky), názvu elementu, z kterého se mají informace brát, názvu atributu, který bude použit jako port a názvu atributu, který bude použit jako jméno.
Všechny tyto informace se budou nacházet v deskriptoru u daného generovaného spojení.

Při úpravě konfigurace uživatelem je potřeba vždy provést kontrolu, zda se nezměnil počet nebo nastavení generovaných spojení.
Na základě uloženého modelu, ve kterém jsou uloženy informace o všech možných typech generovaných spojení a aktuální konfiguraci, zjistí program všechna generovaná spojení, která mají být vytvořena a ke všem přidá informace o portu a jméně spojení. Jelikož lze kombinaci jména a portu použít jako unikátní identifikátor spojení, použil

jsem tuto dvojici na porovnávání s již vytvořenými spojeními.
Proto můžeme iterací projít každé spojení, které modul již obsahuje a zkontrolovat, zda v případě, že je generované, se taktéž nachází v novém seznamu.
Pokud ano, stačí port případně repozicovat v případě, že se počet celkových spojení změnil a v našem seznamu se označí jakožto přidaný. Pokud v novém seznamu není, je z modulu odebrán.
Poté již stačí jen projít seznam všech generovaných spojení, která mají být přidána a přidat k modulu ta, která tak dosud nebyla označena.

5.4.3 Zamezení neplatného propojení modulů

Důležité je zamezit situaci, kdy by existovalo spojení mezi moduly, které by bylo neplatné.
První možností je kontrola spojení poté, co se spojení vytvoří s následnou vizuální signalizací, která spojení jsou neplatná.
Toto řešení se ukázalo jako nevhodné, jelikož dovoluje vznik spojení, která nejsou platná. Navíc uživatel chybu zjistí až po vytvoření daného spojení.
Jako nejlepší způsob, jak chybnému spojení zamezit, je takové situaci předejít. Tedy uživateli předem vizuálně naznačit, která spojení lze provést a která nikoliv.
Pro tyto účely se nejvíce osvědčilo nastavení speciálního stylu (průhlednosti) všem spojením, která nemohou být propojena s vybraným spojením.
K dosažení kýženého výsledku nebylo potřeba zasahovat do interních funkcí knihovny, jelikož poskytuje možnost přidání posluchače na akce vyvolané myší.
Jelikož jediný způsob, jak může uživatel vytvořit neplatné spojení je manuální propojení dvou modulů skrze tažení myši, je pro nás tato funkčnost více než vhodná.
Přidáme tedy kontrolu po stisknutí tlačítka myši, zda byl cílem nějaký port existujícího modulu (tedy, že se uživatel bude pokoušet o propojení dvou modulů). V tento moment se provede iterace skrze všechny ostatní porty a na neplatné se nastaví zvolený styl.
Je ovšem potřeba přidat i kontrolu po puštění tlačítka myši, což je vhodný moment, kdy vrátit styly do původního stavu.
I přes to, že je uživateli jasně signalizováno, jaká spojení může provést, stále zde musí existovat kontrola, která zamezí vytvoření neplatného spojení pro případ, kdy by uživatel signalizaci nepochopil, nebo se z jiného důvodu pokusil vytvořit neplatné propojení dvou modulů.
Řešením této situace se již zabýval prototyp editoru a dané řešení bude použito i zde.

5.4.4 Textová editace konfigurace

Jedním ze způsobů, jak může uživatel editovat konfiguraci, je textový mód editace.
Z analýzy vyplynulo, že nejlepším řešením XML Editoru je pro naše účely CodeMirror.
Editor sám o sobě obsahuje velké množství prvků zvyšujících uživatelskou přívětivost

jako třeba zvýrazňování syntaxe, automatické odsazování vnořených tagů nebo vlastní implementace automatického doplňování.
Bohužel na rozdíl od očekávání, že bude automatické doplňování fungovat na základě definičního schématu (XSD), má CodeMirror vlastní JSON formát, do kterého je třeba informace převést.
Pro zprovoznění automatického doplňování bylo tedy potřeba vytvořit pomocné funkce, které informace o tom, jak má vypadat validní konfigurace, převedly do formátu pro CodeMirror.
Po zprovoznění automatického doplňování se vyskytly 2 problémy s knihovnou CodeMirror, které ubíraly na uživatelské přívětivosti:

- Po automatickém doplnění názvu tagu nedojde k automatickému odsazení (indentaci) řádku.
- Automatické doplňování neřeší, zda je u jednoho tagu nastaven jeden atribut více než jednou.

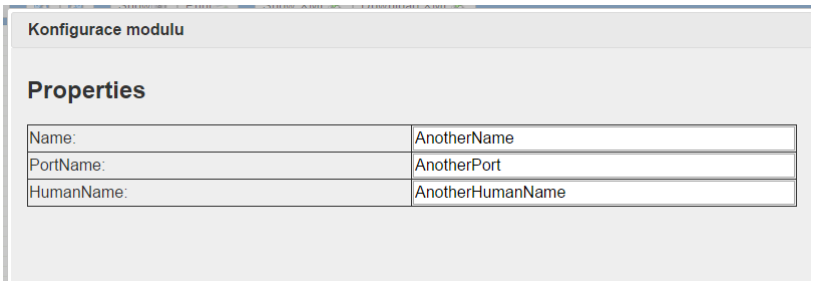
První problém jsem vyřešil skrze přidání posluchače na stisknutí klávesy enter. Po stisku této klávesy dojde nyní automaticky k přepočítání odsazení aktuálního řádku.
CodeMirror neumožňuje žádný způsob, kterým by se dalo zvenčí zasáhnout do výsledků napovídání, proto bylo k účelu vyřešení druhého problému potřeba vytvořit speciální třídu, která přepsala výchozí chování napovídání CodeMirror.
Základem pro tuto třídu se staly zdrojové soubory CodeMirror. Bylo potřeba pouze upravit část rozhodující o přidání parametru do našeptávání a přidat zde kontrolu, zda se parametr v daném elementu již nevyskytuje.

5.4.5 Grafická editace konfigurace

Pro grafickou editaci jsem zvolil formát modálního okna. Modální okno editoru předpokládá, že obdrží model upravovaného elementu a element, do kterého se má výsledek uložit.
Informace, které se v modelu nachází se dají rozdělit na dvě části: parametry (Parameters) a vlastnosti (Properties). Z tohoto důvodu i modální okno rozdělíme na dva celky.

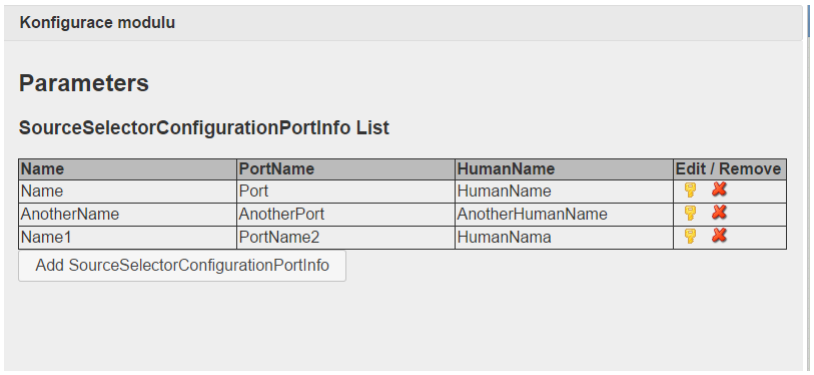
5.4.5.1 Úprava vlastností (Properties)

Jelikož se počet vlastností u modulu nemůže měnit, jako nejvhodnější zobrazení se jeví jednoduchá tabulka s textovými poli, do kterých bude uživateli umožněno vepsat hodnotu dané vlastnosti.



Obrázek 5.3: Ukázka editace vlastností v grafickém režimu

5.4.5.2 Úprava parametrů (Parameters)



Obrázek 5.4: Ukázka editace parametrů v grafickém režimu

Parametry se ukládají jako potomci aktuálního XML elementu. Pokud to model dovoluje, může jich být tedy neomezené množství a jejich počet se může libovolně měnit. Pro zobrazení jsem tedy zvolil tabulku, která zobrazuje všechny vlastnosti jednotlivých potomků. U každého potomka jsou následně přidána tlačítka umožňující jeho úpravu a odstranění.

V případě, že na základě dat v modelu zjistíme, že můžeme ještě přidat další element, je pod tabulkou zobrazeno tlačítko na přidání.

5.4.6 Třídění modulů

Pro zlepšení uživatelské přívětivosti jsem chtěl do programu přidat možnost setřídění modulů přidaných do grafu. Tento problém se původně nejevil jako složitý, jelikož knihovna mxGraph má několik metod, které umožňují uzly v grafu třídit.
Bohužel ani jedna z nich není pro účely vytvářeného editoru nebyla použitelná. Problémem je, že všechny tyto metody spoléhají na to, že mohou pohnout se všemi uzly grafu, nebo alespoň s těmi, ke kterým vedou jednotlivá spojení.

V návrhu vytvářeného editoru jsou však uzly, mezi kterými spojení vedou, pevně ukotvené k modulům.
Z tohoto důvodu by bylo potřeba napsat vlastní metodu, která by se o třídění starala, tato metoda by musela nějakým způsobem počítat se všemi těmito pevnými body modulů.

5.5 Testování

Aby byla zajištěna správná funkčnost programu před jeho nasazením do produkčního prostředí, je první potřeba provést testování.
Testování obvykle probíhá podle předem připravených scénářů.

5.5.1 Základní testování programátorem

Základní testování aplikace programátorem spočívá v přímém testování aplikace jako celku.
Během tohoto testování bylo odhaleno mnoho chyb, zejména v oblasti úpravy konfigurace jednotlivých modulů, jak v grafickém, tak textovém režimu.

Všechny chyby nalezené při tomto testování se podařilo rychle opravit již v průběhu implementace.
Testování bylo prováděno v prohlížeči Google Chrome verze 58.

5.5.2 Scénáře testů

1. Přidat do konfigurace dva moduly stejného typu. Jejich jména se musí lišit.
    - Z levého panelu přetáhnout modul do pole určeného pro editaci. Stejný postup ještě jednou opakovat. Zobrazená jména u modulů se musí i přes stejný typ lišit.
2. Změnit jméno modulu skrze pravý panel.
    - Do konfigurace přidat modul. Vybrat jej a v pravém panelu vepsat do *Name* nové jméno modulu. Potvrdit klávesou enter.
    - Do konfigurace přidat druhý modul. Vybrat jej a v pravém panelu vepsat do *Name* stejné jméno modulu, které má první modul. Potvrdit klávesou enter. Jméno se nesmí změnit.
3. Vzájemně propojit dva moduly

- Do konfigurace přidat dva moduly, které mohou být vzájemně propojeny. Nakliknout modul, který společný typ spojení požaduje (konzumuje). V příslušném boxu vybereme jméno druhého modulu.
- Do konfigurace přidat dva moduly, které mohou být vzájemně propojeny. Myší přejet na spojení, které chceme propojit u prvního modulu. Tažením myší propojit s příslušným spojením u druhého modulu.
- Do konfigurace přidat dva moduly, které mohou být vzájemně propojeny. Myší přejet na spojení, které chceme propojit u druhého modulu. Tažením myší propojit s příslušným spojením u prvního modulu.

4. Zrušit vzájemné propojení dvou modulů

   - Vzájemně propojit dva moduly. Nakliknout modul, který společný typ spojení požaduje (konzumuje). V příslušném boxu vybrat hodnotu -.
   - Vzájemně propojit dva moduly. Nakliknout spojení mezi moduly. Stisknout tlačítko Delete v levém horním rohu.

5. Odstranit modul z konfigurace

   - Přidat do konfigurace modul. Nakliknout přidaný modul. Stisknout tlačítko Delete v levém horním rohu.
   - Přidat do konfigurace dva moduly a vzájemně je propojit. Nakliknout jeden z modulů. Stisknout tlačítko Delete v levém horním rohu.

6. Stáhnout a nahrát konfiguraci

   - Vytvořit novou konfiguraci. Přidat a propojit dva moduly. Stisknout tlačítko *Download XML*.
   - Znovu načíst editor. Při výběru možnosti vytvoření nové konfigurace vybrat *Open existing*. Nahrát stažený soubor.

7. Upravit konfiguraci skrze grafické rozhraní.

   - Přidat do konfigurace modul, který lze konfigurovat. Nakliknout přidaný modul. V pravém okně vybrat možnost *Generate Configuration* a následně *Configure*. Pozměnit konfiguraci a stisknout *Save Changes*. Následně znovu *Configure*, pozměnit konfiguraci a zrušit úpravy skrze *Cancel Changes*.

8. Upravit konfiguraci skrze textové editor.

   - Přidat do konfigurace modul, který lze konfigurovat. Nakliknout přidaný modul. V pravém okně vybrat možnost *Generate Configuration* a následně *Edit XML*. Pozměnit konfiguraci v rámci nabízených změn a stisknout *Save Changes*. Následně znovu *Edit XML*, pozměnit konfiguraci a zrušit úpravy skrze *Cancel*.

- Přidat do konfigurace modul, který lze konfigurovat. Na horním panelu kliknout na *Show XML*. Pozměnit konfiguraci a stistknout *Save Changes*. Následně znovu *Show XML*, pozměnit konfiguraci a zrušit úpravy skrze *Cancel*.

5.5.3 Vyhodnocení testů

Testování proběhlo dle předem připravených scénářů na třech různých internetových prohlížečích (IE11, Chrome 58, Firefox 53).
U každého scénáře došlo k testování přesně dle kroků u něj uvedených.
Všechny testy proběhly úspěšně.

Kapitola 6

Závěr

Cílem této práce bylo provést analýzu požadavků od zadavatele, analyzovat dostupná řešení a analyzovat specifické potřeby pro dostatečnou uživatelskou přívětivost aplikace.
Na základě analýzy bylo cílem navrhnout a implementovat editor splňující zadané požadavky a vhodným způsobem editor otestovat.
Pro vývoj aplikace byl dle očekávání zvolen jazyk Javascript. Jako doplňující open-source řešení jsem zvolil knihovnu mxGraph a CodeMirror.
Volbu editoru CodeMirror považuji i přes několik problémů, na které jsem při implementaci narazil, za správnou.
Kdybych měl znovu volit knihovnu pro práci s grafy, velice pravděpodobně bych znovu využil knihovnu mxGraph, ačkoliv bylo potřeba provést několik úprav a několikrát bylo potřeba přepsat prototypy základních metod, shledal jsem knihovnu pro tuto práci jako vhodnou.
Jako nejobtížnější část celé práce mi přišla úprava dat pro našeptávání editoru CodeMirror a grafická editace konfigurace.
Výsledná aplikace splňuje všechny funkční požadavky zadavatele.

6.1 Rozšíření práce

Ačkoliv editor splňuje všechny funkční požadavky, k dokonalosti má nepochybně ještě daleko. V této kapitole nastíním pár možností, jak by šel editor do budoucna rozšířit.

6.1.1 Vedení spojení

O tom, kudy povede spojení mezi dvěma moduly nyní rozhoduje knihovna mxGraph. Bohužel knihovna nebere v potaz moduly, které jsou již v editačním poli rozložené a v některých případech vede spojení skrz ně.
Bylo by do budoucna dobré upravit metodu starající se o vedení spojení v grafu, aby tyto věci brala v úvahu.

6.1.2 Uživatelské rozhraní

Při tvorbě uživatelského rozhraní jsem dbal hlavně na jednoduchost a přehlednost. Uživatelské rozhraní by bylo vhodné upravit a zlepšit tak uživatelskou přívětivost editoru.

6.1.3 Přidání třídění modulů

Jak jsem se již zmiňoval v implementaci, knihovna sice poskytuje možnost jak provést třídění v grafu, bohužel nejsou její třídící metody pro účely vytvářeného editoru použitelné. Bude potřeba napsat vlastní metodu určující pozici jednotlivých modulů.

Příloha A

Zkratky

- HTML - HyperText Markup Language
- HTTP - HyperText Transfer Protocol
- CSS - Cascading StyleSheets
- XML - Extensible Markup Language
- XSD - XML Schema
- JS - JavaScript
- SW - Software
- SoC - Separation of Concerns
- ISO - International Organization for Standardization
- OSI - Open Systems Interconnection
- W3C - World Wide Web Consortium
- PHP - Personal Home Page Tools
- CGI - Common Gateway Interface
- AJAX - Asynchronous JavaScript and XML
- JSON - JavaScript Object Notation
- IE - Internet Explorer
- NPAPI - Netscape Plugin Application Programming Interface
- SVG - Scalable Vector Graphics

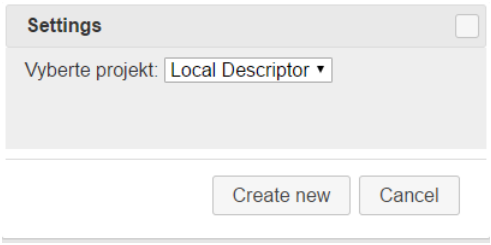
- IDE - Integrated Development Environment
- IBM - International Business Machines Corporation
- URL - Uniform Resource Locator
- UI - User Interface
- XHR - XMLHttpRequest
- DOM - Document Object Model

Příloha B

Manuál k programu

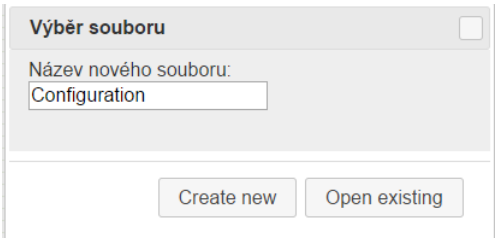
Pro správný běh aplikace je potřeba, aby byl její kód poskytován skrze webový server. Pro jednoduché spuštění webového serveru slouží přiložený dávkový soubor *run-server.bat*. Při úspěšném spuštění bude aplikace k nalezení na adrese `http://localhost:8082`. Tento soubor předpokládá existenci package manageru od Node.js.
V případě, že se na počítači NPM (Node.js Package Manager) nenachází, je třeba jej doinstalovat ze stránky `https://nodejs.org/en/download/`.

Po otevření se jako první zobrazí nabídka s výběrem Adaptéru, ke kterému se má program připojit, jak je vidět na obrázku B.1.
Jelikož adaptér není součástí dodávaného řešení, do programu jsem pro účely ukázky doprogramoval možnost načítání deskriptoru z lokálního disku. Tuto možnost lze nalézt pod názvem *Local Descriptor*.



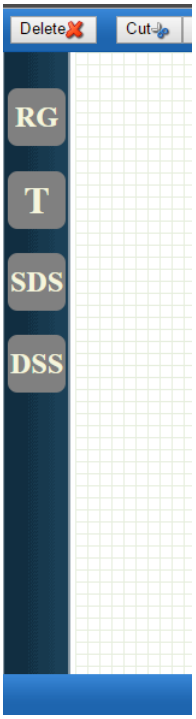
Obrázek B.1: Ukázka - výběr adaptéru

Po výběru adaptéru je možné načíst stávající konfiguraci, nebo vytvořit novou. Při vytváření nové je uživatel taktéž vyzván k zadání jména (můžeme vidět na obrázku B.2). Po výběru již má uživatel plnou kontrolu nad upravovanou sestavou.



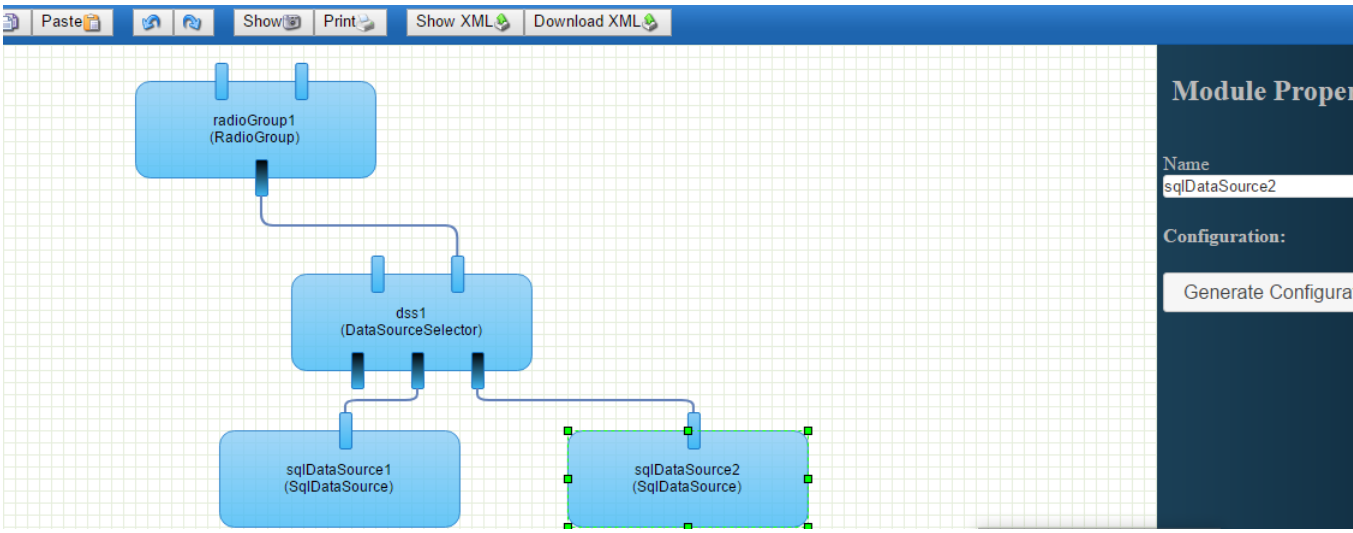
Obrázek B.2: Ukázka - výběr souboru

V levé části nalezneme všechny moduly, které lze tažením myši přidat do sestavy. Levý panel se na obrázku B.3.
V pravé části se nachází panel pro editaci parametrů modulů. Nalezneme zde možnost pro editaci konfigurace nebo propojování modulů. Část editačního okna společně s pravým panelem je zachycena na obrázku B.4)



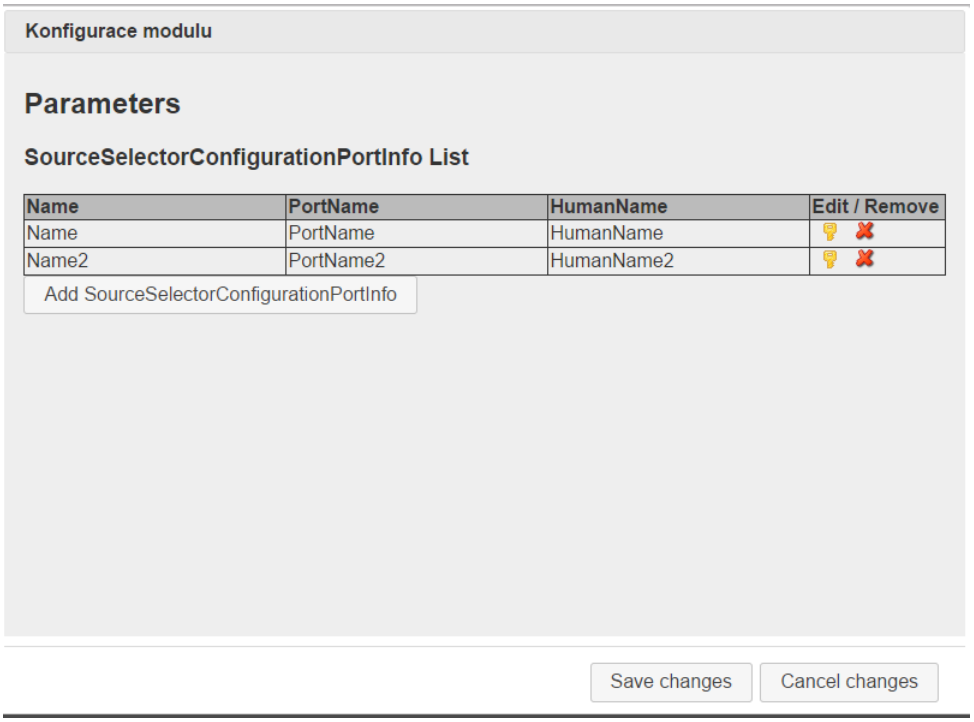
Obrázek B.3: Ukázka - levý panel

V levém panelu si můžeme všimnou možností *Configure* a *Edit XML*. Obě spustí konfiguraci daného modulu, buď v textovém (na obrázku B.5), nebo grafickém režimu (na obrázku B.6. Ukázky náleží k modulu, který jsme měli označený na obrázku B.4. Jelikož tento modul dokáže generovat spojení na základě konfigurace, je k ukázce vhodný.



Obrázek B.4: Ukázka - Editace a pravý panel

Obrázek B.5: Ukázka - textová editace konfigurace



Obrázek B.6: Ukázka - grafická editace konfigurace

Příloha C

Obsah přiloženého paměťového nosiče

Cesta	**Popis**
src/dev	Cesta k programu
src	Zdrojové kódy programu
lib	Využité knihovny
thesis/strnaric.pdf	Cesta k tomuto souboru
thesis/src	Zdrojové soubory k PDF

Literatura

[1] SLAVKO Murko, *A Survey of Programming Techniques* [online] [cit. 9.5.2017] Dostupné z: http://scp.s-scptuj.mb.edus.si/~murkos/Teorija%20in%20vaje/ROM/Interaktivni%20ra%E8unalni%9Aki%20te%E8aji/computing/cpp/cpp1/node4.html

[2] DIJKSTRA, W. Edsger. Letters to the editor: go to statement considered harmful. *Communications of the ACM.* March 1968, vol. 11, issue 3, s. 147-148. ISSN 0001-0782. Dostupné z doi: 10.1145/362929.362947

[3] BOHM, Corrado a Giuseppe JACOPINI. Flow Diagrams, Turing Machines and Languages with Only Two Formation Rules. *Communications of the ACM.* May 1966, vol. 9, issue 5, s. 366-371. ISSN 0001-0782. Dostupné z doi: 10.1145/355592.365646

[4] KRČEK, Břetislav. Metody vytváření algoritmu. *ALGORITMIZACE* [online] [cit. 11.5.2017] Dostupné z: http://homen.vsb.cz/~kol70/algoritmy/cze/kap1/kap_1_5.html

[5] BERNERS-LEE, Tim. A Brief History of the Web. *World Wide Web Consortium.* 1993. [online] [cit 12.5.2017] Dostupné z: https://www.w3.org/DesignIssues/TimBook-old/History.html

[6] BERNERS-LEE, Tim a Daniel Connelly. RFC 1866 – Hypertext Markup Language – 2.0. *Internet Engineering Task Force.* 1995. [online] [cit 12.5.2017] Dostupné z: http://tools.ietf.org/html/rfc1866

[7] IDE, Andy. PHP just grows & grows. *Netcraft - Around the Net.* 2013. [online] [cit 14.5.2017] Dostupné z: https://news.netcraft.com/archives/2013/01/31/php-just-grows-grows.html

[8] W3Techs. *Usage of server-side programming languages for websites.* [online] [cit 9.5.2017] Dostupné z: https://w3techs.com/technologies/overview/programming_language/all

[9] PEYROTT, Sebastián. A Brief History of JavaScript. *Auth0 blog.* [online] [cit 9.5.2017] Dostupné z: https://auth0.com/blog/a-brief-history-of-javascript/

[10] FLANAGAN, David. *JavaScript: the definitive guide*. 6th ed. Sebastopol, CA: O'Reilly, 2011. ISBN 978-0-596-80552-4.

[11] Chromium Blog, *The Final Countdown for NPAPI*. [online] [cit. 9.5.2017] Dostupné z: `https://blog.chromium.org/2014/11/the-final-countdown-for-npapi.html`.

[12] Mozilla Web, *Release Notes for Firefox 52.0*. [online] [cit. 9.5.2017]. Dostupné z: `https://www.mozilla.org/en-US/firefox/52.0/releasenotes/#changed`.

[13] Microsoft Support Website, *Netscape-Style Plug-ins Do Not Work After Upgrading Internet Explorer*. [online] [cit. 9.5.2017] Dostupné z: `https://support.microsoft.com/en-us/help/303401/netscape-style-plug-ins-do-not-work-after-upgrading-internet-explorer`.

[14] Github, *JavaScript InfoVis Toolkit Contributors Graph* [online] [cit. 23.04.2017] Dostupné z: `https://github.com/philogb/jit/graphs/contributors`

[15] W3Schools, *Browser Statistics* [online] [cit. 10.5.2017] Dostupné z: `https://www.w3schools.com/browsers`

[16] StatCounter, *Browser Statistics* [online] [cit. 10.5.2017] Dostupné z: `http://gs.statcounter.com/`

[17] Draw.io Support, *draw.io for Confluence Roadmap as of Q4 2014* [online] [cit. 8.5.2017] Dostupné z: `https://support.draw.io/display/DFCS/2014/10/22/draw.io+for+Confluence+Roadmap+as+of+Q4+2014`

[18] Github, *Force Directed Static Graph Example* [online] [cit. 8.5.2017] Dostupné z: `https://philogb.github.io/jit/static/v20/Jit/Examples/ForceDirected/example1.html`

[19] JointJS, *JointJS Demos ER DIAGRAMS* [online] [cit. 8.5.2017] Dostupné z: `http://resources.jointjs.com/demos/erd`

[20] Scribd, *Introduction to the Windows XP Architecture* [online] [cit. 9.5.2017] Dostupné z: `https://www.scribd.com/doc/18648734/Introduction-to-the-Windows-XP-Architecture`

[21] Growing with the Web, *async vs defer attributes* [online] [cit. 11.5.2017] Dostupné z: `http://www.growingwiththeweb.com/2014/02/async-vs-defer-attributes.html`