

ŽILINSKÁ UNIVERZITA V ŽILINE
FAKULTA RIADENIA A INFORMATIKY

BAKALÁRSKA PRÁCA

Študijný odbor: **Informatika**

Oľga Chovancová

**Vizualizácia dát získaných pomocou SCADA systémov
s využitím HTML 5 štandardov**

Vedúci: **Ing. Juraj Veverka**

Tútor **Ing. Patrik Hrkút, PhD.**

Reg.č. 5/2014

Máj 2015

ZADANIE TÉMY BAKALÁRSKEJ PRÁCE.

Študijný odbor : Informatika

Meno a priezvisko

Olga Chovancová

Osobné číslo

553726

Názov práce v slovenskom aj anglickom jazyku

Vizualizácia dát získaných pomocou SCADA systémov s využitím HTML 5 štandardov

Data visualization acquired by SCADA systems using HTML 5 standards

Zadanie úlohy, ciele, pokyny pre vypracovanie

(Ak je málo miesta, použite opačnú stranu)

Cieľ bakalárskej práce:

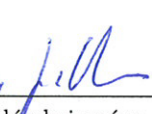
Cieľom práce je vytvorenie vzorovej sady grafických komponentov na vizualizáciu technologických procesov s využitím HTML 5 štandardov. Jedná sa o grafické komponenty, ktoré nie sú bežne dostupné na tvorbu interaktívnych webových aplikácií ako napríklad vizualizácie mechanických súčasti hydraulických systémov, technologických liniek, silových a výkonových častí automatizačných sústav. Cieľová platforma pre výslednú webovú aplikáciu bude kompatibilná s rodinou štandardov HTML 5 pre každý webový prehliadač. Riešenie bude využívať výhradne open-source knižnice s licenciami typu MIT, GNU, GPL, BSD.


Obsah:


- Analyzujte požiadavky, vykonajte prieskum možnosti využitia editorov na tvorbu grafických komponent s možnosťou exportu do formátov SVG, JSON, XML, alebo JavaScript.
- Vyberte vhodné open-source knižnice na tvorbu grafických komponentov kompatibilných s HTML 5.
- Navrhňte REST API na prepojenie grafických komponent so SCADA serverom.
- Analyzujte možnosti automatického mapovania API grafických prvkov pomocou metadát na existujúce API dostupné pre SCADA server D2000.
- Implementujte vzorovú sadu grafických komponentov.
- Zhodnoťte výkonnosť riešenia a definujte výkonnostné obmedzenia.

Meno a pracovisko vedúceho BP: Ing. Juraj Veverka

Meno a pracovisko tutora BP: Ing. Patrik Hrkút, PhD.

2.10.2014 
vedúci bakalárskej práce
(dátum a podpis)

2.10.2014 
tutor
(dátum a podpis)

2.10.2014 
vedúci katedry
(dátum a podpis)

Zadanie zaregistrované dňa 3.10.2014 pod číslom 5/2014 podpis 

Čestné prehlásenie

Prehlasujem, že som bakalársku prácu *Vizualizácia dát získaných zo SCADA systémov použitím HTML5 štandardov* vypracovala samostatne pod vedením Ing. Juraja Veverku, a uviedla v nej všetky použité literárne a iné odborné zdroje v súlade s právnymi predpismi, vnútornými predpismi Žilinskej univerzity a vnútornými aktmi riadenia Žilinskej univerzity a Fakulty riadenia a informatiky.

V Žiline, dňa 15.05.2015

Oľga Chovancová

Podakovanie

Na tomto mieste by som chcela poďakovať vedúcemu bakalárske práce Ing. Jurajovi Veverkovi za cenné pripomienky a odborné rady, ktorými prispel k vypracovaniu tejto bakalárskej práce. Taktiež ďakujem môjmu tútorovi Ing. Patrikovi Hrkútovi, PhD. Zároveň ďakujem mojej rodine a priateľom za ich nekonečnú podporu a trpezlivosť.

V Žiline, dňa 15.05.2015

Oľga Chovancová

Abstrakt

CHOVANCOVÁ OĽGA: *Vizualizácia dát získaných pomocou SCADA systémov s využitím HTML 5 štandardov* [Bakalárska práca]

Žilinská Univerzita v Žiline, Fakulta riadenia a informatiky, Katedra softvérových technológií.

Vedúci: Ing. Juraj Veverka

Tútor: Ing. Patrik Hrkút, PhD.

Stupeň odbornej kvalifikácie: Bakalár Informatiky

Téma práce je vizualizácia dát získaných pomocou SCADA systémov. Cieľ práce je nájsť postup tvorby a vizualizácie grafických komponentov. Produktom bakalárskej práce je grafický komponent na vizualizáciu technologických procesov s využitím HTML 5 štandardov. V práci je popísaný detailný postup vizualizácie prečerpávacej stanice. Bol navrhnutý jednoduchý interface, pomocou ktorého komponenty komunikujú so serverovou časťou SCADA systému. Cieľová platforma pre výslednú webovú aplikáciu bude kompatibilná s rodinou štandardov HTML 5 pre každý webový prehliadač.

Kľúčové slová: vizualizácia, HTML5, SVG, JavaScript, Snap.svg, SCADA, REST API

Abstract

CHOVANCOVÁ OĽGA: *Data visualization acquired by SCADA systems using HTML5 standards* [Bachelor thesis]

University of Žilina, Faculty of Management Science and Informatics, Department of Software Technologies

Tutor: Ing. Juraĵ Veverka

Tutor: Ing. Patrik Hrkút ĩ, PhD.

Qualification level: Bachelor of Informatics

The main aim of this thesis was to visualize data obtained by SCADA systems. Aim is to provide a process of creation and visualization of graphical components. Product of the thesis is a graphical components for visualization of technological processes using HTML 5 standards. The paper describes the detailed process visualization of pumping station. It was designed simple interface through which components communicate with the server part of the SCADA system. The target platform for the resulting web application will be compatible with your family the HTML 5 for each Web browser.

Key words: visualization, HTML5, JavaScript, Snap.svg, SCADA, REST API

Obsah

Úvod	11
1 Analýza požiadaviek	13
1.1 Systémy SCADA	13
1.2 HTML5 štandard	14
1.3 Čo je SVG?	14
1.3.1 Podpora vo webovom prehliadači	15
1.3.2 Rozdiely medzi SVG a Canvas	16
1.4 Nástroje na tvorbu grafických komponentov	17
1.5 JavaScriptové knižnice pre grafické komponenty	17
1.5.1 D3.js	18
1.5.2 jQuery.js	18
1.5.3 Veloncity.js	18
1.5.4 SVG.js	18
1.5.5 Raphaël.js	19
1.5.6 Snap.svg.js	19
1.6 Zhodnotenie požiadaviek	20
2 Postup vytvorenia komponentov	21
2.1 Použitie SVG v HTML dokumente	21
2.1.1 Vytvorenie SVG	21
2.2 UML diagramy	23

2.2.1	Usecase diagram použitia SCADA systémov	23
2.2.2	Postup načítania SVG súboru v HTML	24
2.2.3	Postup zistenia id elementu	25
3	Využitie knižnice Snap.svg	26
3.1	Inicializácia plátna na kreslenie - Snap()	27
3.1.1	Súradnice plátna	27
3.1.2	DOM element	28
3.2	Kreslenie základných tvarov cez knižnicu Snap.svg	29
3.2.1	Popis atribútov tvarov	30
3.3	Vykreslenie obrázku	31
3.4	Atribúty elementu	31
3.4.1	Výplň elementu - fill	32
3.4.2	Nastavenie okraja elementu - stroke	32
3.5	Zoskupovanie elementov - Paper.group()	32
3.5.1	Maskovanie - Paper.mask()	33
3.6	Gradient	34
3.7	Práca s textom - Paper.text(x, y, text)	35
3.8	Transformácie - Element.transform(...)	36
3.9	Animácie	37
3.9.1	Animácie transformácie	37
3.10	Ďalšie metódy v knižnici Snap.svg	38
4	Vzorová sada	40
4.1	Diagram tried vzorovej sady	40
5	Integrácia grafického komponentu	44
5.1	Vytvorenie prečerpávacej stanice v Inkscape	44
5.2	Definovanie id v SVG	44
5.2.1	Object Properties	45

5.2.2	XML Editor	45
5.3	Integrácia prečerpávacej stanice pre dynamické ovládanie SVG objektu	46
5.3.1	HTML súbor	46
5.3.2	Kód	47
5.3.3	Vysvetlenie kódu	47
5.4	PumpingStation.js	48
5.4.1	onPageLoad()	48
5.4.2	PumpingStation(par1, par2)	48
5.4.3	animateTank(percento)	50
5.4.4	openValve1(isOpen)	51
5.4.5	rotateEngine(isRotating)	51
5.5	updateSchema()	53
6	REST API	56
6.1	REST API pre grafické komponenty	56
6.2	Data binding pre system D2000	57
7	Analýza výkonnosti a obmedzení	58
7.1	Test výkonnosti knižníc Snap.svg.js a Raphael.js - animate()	58
7.2	Raphael.js vs Snap.SVG.js vs SVG.js - attr()	62
7.3	SVG vs Canvas	66

Zoznam obrázkov

1.1	HTML 5 API	14
1.2	Podpora SVG vo webových prehliadačoch	15
2.1	Diagram prípadov použitia vizualizácie v SCADA systémoch	23
2.2	Postup načítania SVG súboru v HTML dokumente	24
2.3	Postup zistenia id elementu a jeho aktualizovanie	25
3.1	Súradnicový systém plátna s bodom (300, 200) v Snap.svg knižnici	27
3.2	Príklad zoskupenia elementov a následná zmena atribútov	33
4.1	Diagram tried vzorovej sady grafických komponentov	41
4.2	Prečerpávací stanica	42
4.3	Prepravný pás	42
4.4	Trojcestný ventil	42
4.5	Mapa Slovenska	43
4.6	Termometer	43
5.1	Grafické prostredie programu Inkscape s nakreslenou prečerpávacou stanicou	44
5.2	Object Properties	45
5.3	Xml Editor v Inkscape	46
5.4	Prečerpávací stanica po vykonaní príkazu <code>updateSchema(updateSchema);</code>	55

7.1	Po spustení testovania výkonnosti knižníc Raphael.js vs Snap.svg.js - zobrazenie spustených animácií	60
7.2	Po spustení testovania výkonnosti knižníc Raphael.js vs Snap.svg.js - výsledky pre konkrétny prehliadač	60
7.3	Graf výsledkov pre webové prehliadače	62
7.4	Test pre funkciu attr() pre webový prehliadač Chrome 42	64
7.5	Graf výsledkov pre webové prehliadače	65
7.6	Porovnanie výkonnosti Canvas vs. SVG	66

Zoznam tabuliek

1.1	Podpora HTML <code><svg></code> elementu v webových prehliadačoch	15
1.2	Porovnanie Canvas a SVG	16
3.1	Spôsoby zobrazenia SVG elementu v HTML dokumente	28
3.2	Zoznam tvarov, ktoré podporuje SVG a Snap API	29
3.3	Názvy atribútov a ich význam	30
3.4	Príkazy na tvorbu Path elementu	31
3.5	Výber možných stroke atribútov	32
3.6	Atribúty na zmenu vlastností elementu text	35
3.7	Syntax transformačného stringu pre Snap API	36
3.8	Typy transformácií vo vnútri SVG tagu <code>animateTransform</code>	36
3.9	Parametre pre metódu <code>animate</code>	37
7.1	Výsledky testu pre mobilné zariadenia	61
7.2	Výsledky testu pre deskopové zariadenia	61
7.3	Výsledky testu pre viaceré webové prehliadače (ops/sec)	65

Úvod

Téma práce je vizualizácia dát získaných pomocou SCADA systémov. Cieľ práce je nájsť postup tvorby a vizualizácie grafických komponentov. Produktom bakalárskej práce je grafický komponent na vizualizáciu technologických procesov s využitím HTML 5 štandardov. V práci je popísaný detailný postup vizualizácie prečerpávacej stanice. Bol navrhnutý jednoduchý interface, pomocou ktorého komponenty komunikujú so serverovou časťou SCADA systému.

Súčasná situácia

V súčasnosti je v IPESOFT s.r.o. software, ktorý dokáže vizualizovať dáta z technológií pomocou "hrubých klientov", čo sú natívne (.exe) Windows aplikácie. Aktuálna webová prezentácia takýchto dát nespĺňa súčasné štandardy pre moderné webové aplikácie a preto bolo potrebné nájsť nový spôsob vizualizácie na webe, ktorý bude v budúcnosti použiteľný na rôznych platformách, nielen na PC.

Výsledná webová aplikácia je kompatibilná s štandardmi HTML 5. Riešenie využíva výhradne open-source knižnice s licenciami typu MIT, GNU GPL, BSD, Apache 2.

Zdrojové kódy práce sú udržiavané v Git repository.[2]

Postup práce

1. Analýza požiadaviek, prieskum možnosti využitia WYSIWYG editorov na tvorbu grafických komponent s možnosťou exportu do formátov SVG, JSON, XML, alebo JavaScript.
2. Výber vhodných open-source knižníc na tvorbu grafických komponent kompatibilných s HTML 5.
3. Popis postupu vizualizácie grafického komponentu.
4. Implementácia vzorovej sady grafických komponent.
5. Návrh REST API na prepojenie grafických komponent so SCADA serverom.
6. Analýza možnosti automatického mapovania API grafických prvkov pomocou metadát na existujúce API dostupné pre SCADA server D2000.
7. Analýza výkonnosti a výkonnostné obmedzenia.

Kapitola 1

Analýza požiadaviek

Kapitola popisuje základné pojmy, výber z dostupných nástrojov a knižníc.

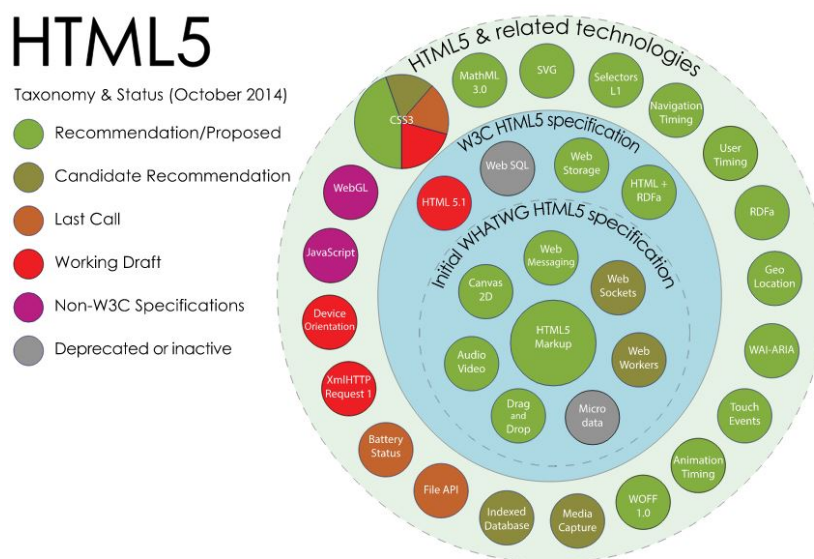
1.1 Systémy SCADA

IPESOFTE D2000® je softvérová technológia reálneho času vyvinutá spoločnosťou IPE-SOFT. Využíva sa pre tvorbu aplikačných riešení pre oblasť výrobných, energetických a obchodných systémov. Táto platforma je vhodná pre aplikácie, kde je potrebné zabezpečiť zber a vizualizáciu dát z priemyselných automatov, riadenie technologických procesov, tvorbu bilančných nástrojov a prehľadov, integráciu rôznych podnikových systémov.[3]

IPESOFTE D2000® je objektovo orientovaný SCADA (Supervisory Control And Data Acquisition) systém, ako aj platforma pre tvorbu komplexných MES (Manufacturing Execution System) aplikácií. V súhrne svojich vlastností predstavuje optimalizovaný nástroj triedy RAD (Rapid Application Development) pre informačné systémy pracujúce súčasne s údajmi technického charakteru v reálnom čase, technickými a obchodnými údajmi vo forme časových radov a obchodnými údajmi vo forme databázových tabuliek. [3]

1.2 HTML5 štandard

World Wide Web Consortium (W3C) vydalo štandard HTML5 dňa 28. októbra 2014. HTML5 je podporovaný vo všetkých moderných webových prehliadačoch. Na obrázku 1.1 [1] je HTML5 API a súvisiace taxonómia technológií a ich status.



Obr. 1.1: HTML 5 API

HTML5 Graphics definuje dva spôsoby vykreslenia využívajúc:

- `<canvas>` - JavaScript
- `<svg>` - SVG

1.3 Čo je SVG?

Scalable Vector Graphics (SVG) je štandardný formát pre vektorovú grafiku. Vektorová grafika je definovaná cez body, priamky, mnohoúhelníky, elipsy, krivky alebo iné geometrické tvary.

SVG je jazyk na opísanie dvojrozmiernej grafiky v EXtensible Markup Language (XML). Vďaka tomu, umožňuje reprezentáciu grafických informácií v kompaktnom a prenositeľnom tvare.

SVG povoľuje tieto tri typy grafických objektov: vektorové grafické tvary, obrázky a text. Grafické objekty môžu byť zoskupené, štylizované, zmenené a kombinované do predošlých vrstiev objektov.

SVG obrázky môžu byť dynamické a interaktívne.

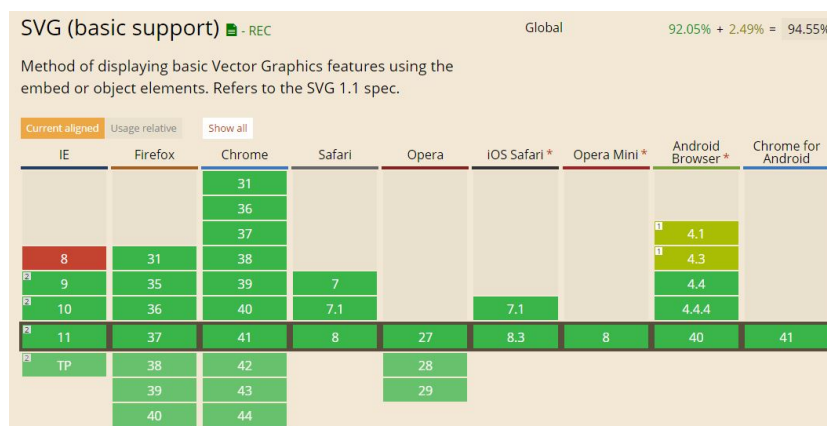
Prispôsobiteľnosť SVG umožňuje zmeniť veľkosť grafického komponentu bez straty kvality vzhľadu, čo umožňuje zobraziť responzívne na viacerých možných zariadení. SVG sa bude zobrazovať rovnako na rôznych platformách. Je kompatibilná s štandardmi HTML5, ktoré navrhla W3C.

1.3.1 Podpora vo webovom prehliadači

Súčasný prehliadač plne podporujú <svg> elementy. Čísla v tabuľke 1.1 špecifikujú prvé verzie webových prehliadačov, ktoré sú schopné zobraziť <svg> element.[21]

Element	Chrome	Internet Explorer	Firefox	Safari	Opera
< svg >	4.0	9.0	3.0	3.2	10.1

Tabuľka 1.1: Podpora HTML < svg > elementu v webových prehliadačoch



Obr. 1.2: Podpora SVG vo webových prehliadačoch

1.3.2 Rozdiely medzi SVG a Canvas

SVG patrí do vektorovej grafiky a Canvas zase do raster bitmap grafiky. SVG je jazyk na opísanie dvojrozmernej grafiky v XML. Canvas kreslí dvojrozmernú grafiku za behu programu cez JavaScript. SVG je XML založený, čo znamená, že každý element je dostupný cez SVG DOM. JavaScript umožňuje ovládanie udalostí elementov. V SVG je každý tvar zapamätaný ako objekt. V prípade zmeny <svg> elementu sa automaticky prekreslí.

Canvas je prekresľovaný pixel za pixelom. Bitmapová grafika je zložitejšia pre dynamické prekresľovanie a má menšie pamäťové nároky a je rýchlejšia.

Moderné zariadenia, ako napríklad smartfóny, majú veľmi vysokú hustotu pixelov.

Zariadenia ako moderné smartfóny majú veľmi vysokú hustotu pixelov. Niektoré potláčajú 300 Pixels Per Inch (PPI) s tým, že sa spoliehajú na obmedzenosť ľudských očí rozoznávať jemné detaily. Pixel nemá v reálnom živote equivalent vo veľkosti, až pokiaľ je na obrazovke s fixovaným rozmerom a rozlíšením. Text s veľkosťou 16 pixelov bude veľmi malý pre oko. Pre tento dôvod zariadenia jednoducho nezobrazujú 1 CSS pixelovú jednotku na 1 pixel zariadenia. Namiesto toho zdvoja svoju veľkosť. [5]

Tabuľka 1.2 zobrazuje niekoľko dôležitých odlišností medzi Canvas a SVG.[28]

Canvas	SVG
Závislé na rozlíšení a DPI	Nezávislé na rozlíšení a DPI
Založený na pixeloch	Založený na tvaroch
Vhodné pre komplexné scény, real-time matematické animácie	Vhodné pre statické obrázky.
Nepodporuje dynamické zmeny	Podporuje dynamické zmeny
Jednoduchý HTML element	Zložený z grafických elementov, ktoré sa stanú časťou DOM
Modifikovateľný len cez script	Modifikovateľné cez script a CSS
Výkonnosť je lepšia s menšou plochou, a väčším množstvom objektov (>10k)	Výkonnosť je lepšia s menším množstvom objektov (<10k) a väčšou plochou

Tabuľka 1.2: Porovnanie Canvas a SVG

1.4 Nástroje na tvorbu grafických komponentov

WYSIWYG editory, ktoré umožňujú tvorbu grafických komponentov sú:

- Inkscape,
- CorelDraw,
- Adobe Illustrator,
- Sketch,
- Libre Office Draw .

Voľne dostupné WYSIWYG online SVG editory:

- svg-edit - Rýchly, webový SVG editor založený na JavaScriptovej technológii, ktorá funguje v akékoľvek modernom webovom prehliadači. [7]
- animatron - online editor, umožňuje vytvoriť HTML5 animácie, a následne vy-exportovať do SVG SMIL animácie.[6]

1.5 JavaScriptové knižnice pre grafické komponenty

Na internete sa nachádzajú tieto OpenSource JavaScriptové knižnice na tvorbu grafických komponentov:

- D3.js,
- Raphael.js,
- Snap.svg.js,
- Svg.js,
- jQuery.js,
- Velocity.js.

Popis jednotlivých JavaScriptových knižníc.

1.5.1 D3.js

D3.js je JavaScriptová knižnica určená na manipuláciu dokumentov založených na dátach. Pomocou HTML, SVG a CSS umožňuje vizualizáciu dát. Je vhodná na vytváranie interaktívnych SVG grafov s hladkými prechodmi a interakciami.

D3 rieši efektívnu manipuláciu dokumentov zakladajúcich si na dátach. Využíva webové štandardy ako HTML, SVG a CSS3. [25] Má licenciu BSD.

1.5.2 jQuery.js

jQuery je knižnica s otvoreným zdrojovým kódom, ktorá poskytuje funkcionálny programacie rozhranie k JavaScriptu. Jedná sa o kompletnú knižnicu, ktorej jadro je postavené pomocou selektorov jazyka CSS pracujúcimi s elementami modelu DOM. Knižnicu jQuery napísal a spravuje John Resig. Má licenciu MIT alebo GPL. [13]

V jQuery API metóda `animate()` umožňuje vytvoriť efekt animácie, ktoré ovplyvňujú CSS vlastnosti. Požadovaný parameter je objekt s CSS vlastnosťami. [8]

1.5.3 Velocity.js

Velocity je nástroj na animáciu s rovnakým API ako jQuery `$.animate()`. Funguje aj bez jQuery. Je rýchly, a podporuje animácie farby, transformácie, opakovania, zjemňovania, SVG podpora a rolovanie. [9] Má licenciu MIT.

1.5.4 SVG.js

SVG.JS je ďalšia knižnica umožňujúca manipulovať a animovať SVG. Medzi hlavné výhody knižnice patrí to, že má ľahko čitateľnú syntax. Umožňuje animovanie veľkosti, pozície, transformácie, farby. Má modulárnu štruktúru, čo umožňuje používanie rôznych rozšírení. Existuje množstvo užitočných pluginov dostupných na internete. [26] Má licenciu MIT.

1.5.5 Raphaël.js

Raphaël je malá JavaScriptová knižnica, ktorá umožňuje jednoducho pracovať s vektorovou grafikou na webe. Umožňuje pomocou jednoduchých príkazov vytvárať špecifické grafy, obrázky.

Raphaël využíva SVG W3C odporúčania a VML na tvorbu grafických komponentov. Z toho vyplýva to, že každý vytvorený grafický objekt je zároveň aj DOM objekt. To umožňuje cez JavaScript pridávať manipuláciu udalostí alebo ich upravovať neskôr. Momentálne podporuje Firefox 3.0+, Safari 3.0+, Chrome 5.0+, Opera 9.5+ and Internet Explorer 6.0+.[24] Autor knižnice je Dmitry Baranovskiy. Raphael API má široké spektrum používateľov. Knižnica neumožňuje load SVG do dokumentu zo súboru.

Má licenciu MIT.

1.5.6 Snap.svg.js

Snap.svg.js je JavaScriptová knižnica na prácu s SVG. Poskytuje pre webových developerov API, ktoré umožňuje animáciu a manipulovanie s buď existujúcim SVG alebo programátorsky vytvorene cez Snap API.

Tvorca Snap knižnice je rovnaký ako pri Raphael knižnici. Bola navrhnutá špeciálne pre moderné prehliadače (IE9 a vyššie, Safari, Chrome, Firefox a Opera). Z toho vyplýva, že umožňuje podporu maskovania, strihania, vzorov, plných gradientov, skupín.

Snap API je schopné pracovať s existujúcim SVG súborom. To znamená, že SVG obsah sa nemusí generovať cez Snap API, aby sa mohol oddelene používať. Obrázok vytvorený v nástroji Inkscape sa dá animovať alebo inak manipulovať cez Snap API. Súbor načítaný cez Ajax sa dá vykresliť, bez toho, aby boli renderované.

Knižnica Snap.svg podporuje animácie. Poskytuje jednoduché a intuitívne JavaScript API pre vizualizáciu grafických komponentov. Snap.svg umožňuje urobiť SVG obsah viac interaktívnejší a záživnejší. [23]

Má licenciu Apache 2, tkré

1.6 Zhodnotenie požiadaviek

Animovanie cez SVG SMIL poskytuje časovo orientované animácie, ktoré sa spúšťajú v špecifickom intervale a dĺžky. Môžu byť zobrazované opakovaných sekvenciách. SCADA aplikácie požadujú okamžitú animáciu na zmenu v okamihu, keď asociované dáta sú aktualizované. Z toho robí samotné použitie SVG SMIL animácií nehodiace sa pre vizualizáciu SCADA systémov. Ďalšia nevýhoda SVG SMIL je, že Internet Explorer ju nepodporuje.

Použitím JavaScriptových knižníc zameraných na animovanie a manipulovanie s SVG súbormi odstráni problém s SVG SMIL animovaním. Umožňujú používať podmienky, stavy a premenné.

Každá zmena objektu v SCADA systéme sa vykreslí v animácii okamžite bez toho, aby webový prehliadač znovu načítal celú schému.

Grafické komponenty sa budú vytvárať v programe Inkscape. Následne budú použité v HTML dokumente. Ovládanie a animovanie bude realizované prostredníctvom knižnice Snap.svg.js.

Zo spomínaných knižníc najviac vyhovuje práve Snap.svg.js pre splnenie cieľov práce. Ďalší dôvod, prečo som sa rozhodla pre túto knižnicu bol, že dokáže načítavať SVG súbor a následné s ním manipulovať.

Spĺňa požiadavku kompatibility pre moderné webové prehliadače. Je to open-source knižnica a má licenciu Apache 2.

Kapitola 2

Postup vytvorenia komponentov

Spôsob vytvorenia grafických komponentov je nasledovný. Najprv používateľ vytvorí SVG súbor, následne ho načíta a vytvorí funkcie v JavaScripte na ovládanie atribútov SVG elementu. Alternatívna možnosť je vytvoriť SVG elementy prostredníctvom JavaScriptovej knižnice a nenačítavať súbor.

2.1 Použitie SVG v HTML dokumente

SVG sa dá použiť a vytvoriť viacerými spôsobmi:

- priamo v HTML dokumente - inline,
- načítanie z oddeleného SVG súboru,
- načítanie pomocou JavaScriptovej knižnice.

2.1.1 Vytvorenie SVG

Načítanie JavaScript knižnice

Postup (načítanie súboru v tele JavaScriptovej metóde):

1. Načítať knižnicu Snap.svg.js do HTML súboru.
2. Pridať atribút onPageLoad(); do definície body.

3. Pridať HTML tag `<svg>` do tela HTML a nastaviť v ňom požadovanú veľkosť cez `viewBox`.
4. Vytvoriť JavaScriptový súbor, alebo tag `<script>`, ktorý bude obsahovať funkcie.
5. Vytvorenie funkcie na načítanie Snap API a `.SVG` súboru.
6. V tele funkcie inicializácia Snap Canvasu. To znamená, kde konkrétne v HTML stránke sa zobrazí.
 - `s = Snap()` - najbližšie voľné miesto
 - `s = Snap(šírka, výška)`
 - `s = Snap(HTMLtag)` - id tagu `<svg>`, ktoré sa pridalo v bode č. 3
7. Načítanie `.SVG` súboru cez funkciu `Snap.load()`, s parametrami: názov súboru a funkcie s parametrom `f`.
8. Zobrazenie súboru cez príkaz `s.append(f);`, ekvivalentné zápisy: `s.appendAll(f);`, `s.add(f);`.
9. V HTML stránke sa zobrazí daný `.SVG` súbor.

Postup ovládania SVG elementu:

1. Vytvorenie novej funkcie.
 - anonymná funkcia
 - pomenovaná funkcia
 - objekt, v ktorom bude zadaná funkcia.
2. Nová premenná `var`, ktorá obsahuje id SVG elementu, ktorý sa ide ovládať. (Na zistenie id SVG viď Postup krokov na zistenie id.)
3. Vytvorenie funkcie, cez ktorú sa bude pristupovať k API Snap knižnice.
4. `s.select(id SVG)`

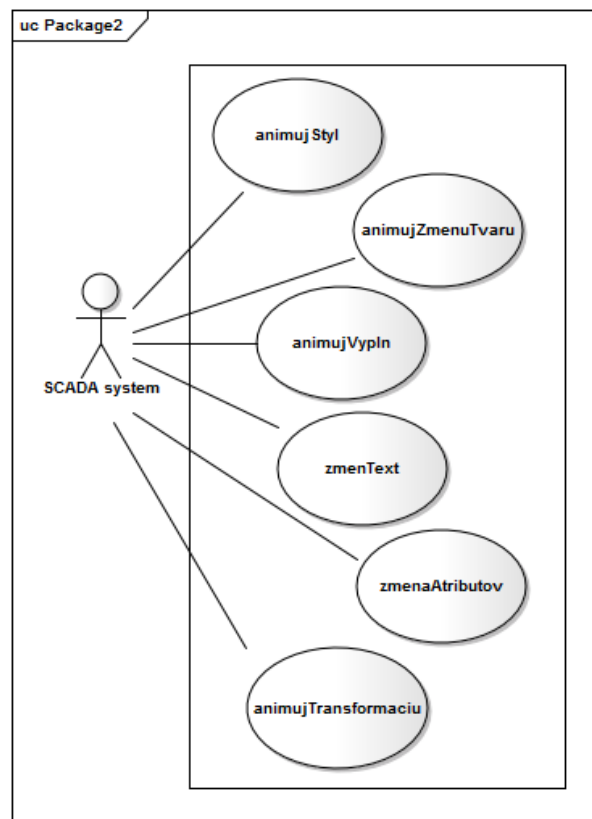
5. V tejto chvíli je možné volať funkcie z Snap.svg API príkazom: `element.funkciaAPISnap..`

- `.animate()` - animácia
- `.attr()` - nastavenie atribútu
- `.add()`, a iné.

2.2 UML diagramy

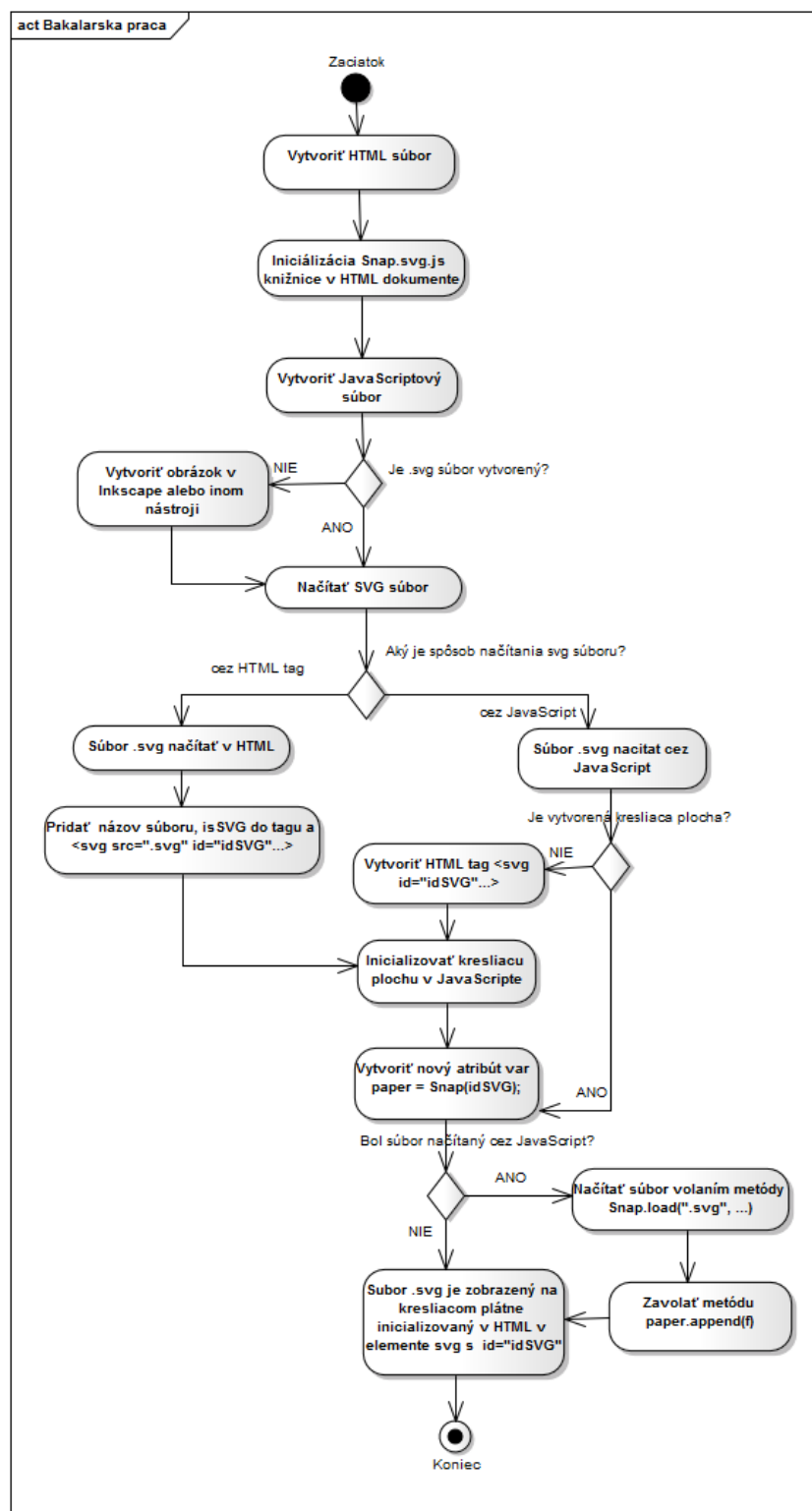
V nasledujúcej časti sú zobrazené UML diagramy, ktoré zobrazujú postup načítania, vytvorenia, ovládania grafických komponentov.

2.2.1 Usecase diagram použitia SCADA systémov



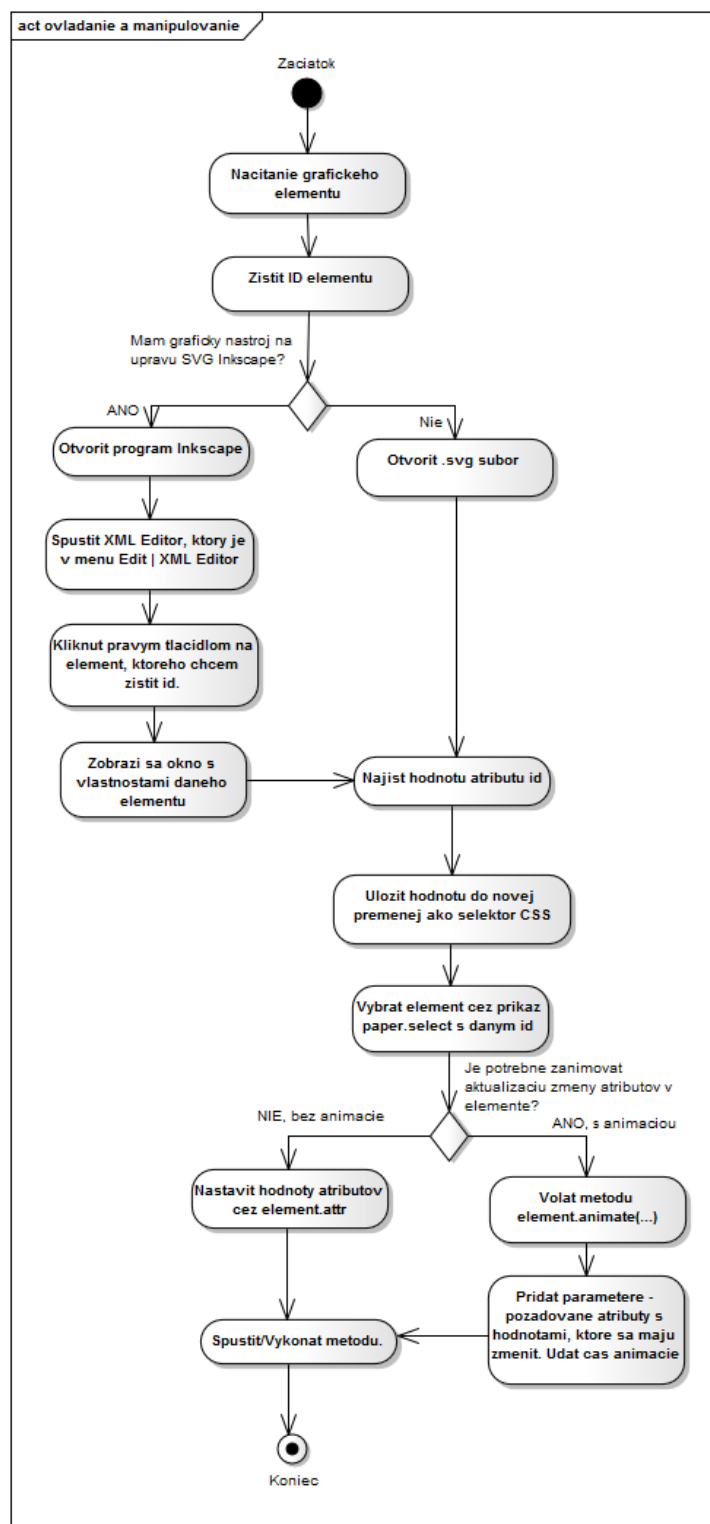
Obr. 2.1: Diagram prípadov použitia vizualizácie v SCADA systémoch

2.2.2 Postup načítania SVG súboru v HTML



Obr. 2.2: Postup načítania SVG súboru v HTML dokumente

2.2.3 Postup zistenia id elementu



Obr. 2.3: Postup zistenia id elementu a jeho aktualizovanie

Kapitola 3

Využitie knižnice Snap.svg

Animovanie vektorov je jednoduchšie cez knižnicu Snap.svg ako cez SVG SMIL.

V nasledujúcom príklade je vykreslený a animovaný obdĺžnik cez SVG SMIL. Animácia spočíva v zmene šírky obdĺžnika z 50 na 100 pixlov. SMIL:

```
<svg><rect x="10" y="10" width="50" height="30">
  <animate attributeName="width"
    to="100" fill="freeze" dur="10s"/> </rect></svg>
```

Nakreslí sa obdĺžnik na súradniciach (10, 10) s šírkou 50, a výškou 30 použitím elementu `<rect>`. Zoskupený element `<animate>` definuje animáciu zmeny šírky obdĺžnika na šírku 100 px, ktorá trvá desať sekúnd. Kde `fill="freeze"` je použité na zachovanie stavu obdĺžnika po ukončení animácie. Inak by bola nastavená znova na 50 px. [10, p. 9]

Ekvivalent k animácii cez Snap.svg API v nasledujúcom príklade:

```
paper = Snap();
var rect = paper.rect(10, 10, 50, 30);
rect.animate({
width: 100
}, 10000);
```

Syntax metód `animate` a `rect` je výstižnejšia a lepšia na pochopenie. Snap sa tiež dobre integruje s inými knižnicami. V práci som využívala iba Snap.svg knižnicu.

3.1 Inicializácia plátna na kreslenie - Snap()

Na to, aby sme boli schopní kresliť grafické komponenty, tak potrebujeme definovať miesto, kde budú vykreslené. Viditeľná oblasť okna prehliadača alebo viewport, definuje oblasť, v ktorej sa vykreslí komponent na plátno. SVG špecifikácia referuje ako miesto vykreslenia seba ako viewport. Inak povedané viewport je akákoľvek obdĺžniková oblasť. Okno prehliadača je referencia na viewport a kresliaca oblasť je plátno. [10]

Vytvorenie plátna cez Snap konštruktor sa dá urobiť viacerými spôsobmi.

3.1.1 Súradnice plátna

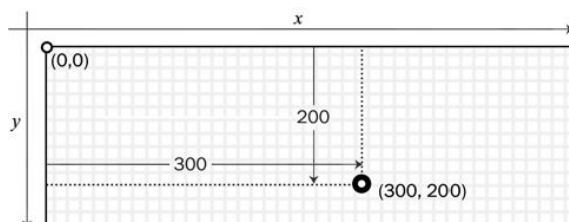
Na to, aby bolo plátno zobraziteľné vo webovom prehliadači na rôznych zariadeniach musia byť nastavené nasledujúce atribúty:

- definovaný viewBox,
- výška a šírka plátna musí byť v relatívnych rozmeroch, najlepšie nastavená na 100%

Nasledujúci príkaz zdefinuje plátno s rozmermi šírka je 300 a výška 200.

```
var paper = Snap(300, 200);
```

Na obrázku 3.1 je znázornená východzí súradnicový systém plátna vytvoreného cez Snap konštruktor. Začiatok súradníc na osi x, y je rovné nule. Bod na plátne so súradnicami $x = 300$, $y = 200$ alebo $(300, 200)$ vo vektorovom zápise je bod 300px vpravo od začiatku x-ovej osi a 200px dole od počiatku y-ovej osi.



Obr. 3.1: Súradnicový systém plátna s bodom $(300, 200)$ v Snap.svg knižnici

3.1.2 DOM element

Sú situácie, kde je potrebné použiť existujúci DOM element ako kontajner pre plátno než viewport. Ako element môžeme použiť napríklad:

```
<div id="mojePlatno"></div>
```

Nasledujúcim kódom vytvorím 500px široké a 300px vysoké plátno.

```
var paper = Snap("mojePlatno", 500, 300);
```

Keď využívam túto formu konštruktora, tak prvý parameter v metode je ID elementu. V prípade, že nie je udaný vykreslí sa na najbližšie voľné miesto. Alternatívne sa dá prvý parameter DOM element napísať nasledovným spôsobom:

```
Snap(document.getElementById('mojePlatno'), 500, 300);
```

SVG v HTML dokumente

Grafický komponent vo formáte .svg môže byť zobrazený buď ako inline v HTML dokumente, alebo ako vložený samostatného .svg súboru. V tabuľke 3.1 sú vymenované HTML tagy na zobrazenie SVG.

Technika (tag)	Popis
<embed>	Načíta vytvorený SVG súbor.
<object>	Vytvorí objekt SVG
<iframe>	Zobrazí SVG v rámci
Inline <svg>	Vytvorí SVG

Tabuľka 3.1: Spôsoby zobrazenia SVG elementu v HTML dokumente

Príklady načítania SVG v HTML dokumente pre tag Image:

```

```

3.2 Kreslenie základných tvarov cez knižnicu Snap.svg

Snap.svg API poskytuje metódy na kreslenie jednoduchých tvarov. V tabuľke 3.2 sú základné tvary, ich equivalentné vyjadrenie v SVG ako elementy, a príkazy na vytvorenie nových elementov cez Snap.svg. Keďže sa jedná o vektorovú grafiku, tak pre elementy vytvorené cez SVG alebo Snap.svg platia rovnaké atribúty. Líši sa iba spôsob zápisu.

Tvar	SVG element	Snap.svg API	Atribúty
Obdlžnik	<rect>	.rect()	x, y, šírka, výška, rx, ry
Kruh	<circle>	.circle()	r, x, y, cx, cy, rx, ry
Elipsa	<ellipse >	.ellipse()	x, y, cx, cy, rx, ry
Čiara	<line>	.line()	x1, y1, x2, y2
Polyline	<polyline>	.polyline()	pole x, y súradnic bodov
Polygon	<polygon>	.polygone()	pole x, y súradnic bodov
Path	<path>	.path()	vid tabuľka 3.4

Tabuľka 3.2: Zoznam tvarov, ktoré podporuje SVG a Snap API

Tvar, ktorý je vykreslený cez Snap API má nasledovnú syntax:

```
var paper = Snap (...);  
var tvar = paper.NazovSnapMetody({  
    nazovAtributu: "hodnotaAtributu", ...  
});
```

Tvar, ktorý je vykreslený priamo na HTML webovej stránke má vo vnútri elementu <svg> definované atribúty nasledujúcim spôsobom:

```
<ElementTvar nazovAtributu = "hodnotaAtributu" ... />
```

3.2.1 Popis atribútov tvarov

Názvy atribútov a ich význam pre obdĺžnik, kruh, elipsu sú vyjadrené v tabuľke 3.3

Parameter	Poznámka
x, y	súradnica x-osi, y-osi
cx	x-os súradnica centra kruhu, alebo elipsy
cy	y-os súradnica centra kruhu, alebo elipsy
r	polomer kruhu, elipsy alebo okruhlých rohov na obdĺžniku
rx	horizontálny polomer elipsy
ry	vertikálny polomer elipsy
x1, y1	začiatkové x, y súradnice
x2, y2	konečné x, y súradnice

Tabuľka 3.3: Názvy atribútov a ich význam

Pre útvary polyline, polygon sú atribúty dvojice súradníc osi x, y, ktoré určujú body, ktoré sa spoja.

Path tvar

V Snap API je to metóda `Paper.path([pathString])`, ktorá vytvorí `<path>` element podľa daného reťazca. Parameter `pathString` pozostáva reťazca skladajúceho sa z jedno písmenkových príkazov, nasledovaných bodkami a oddelovaný argumentami a číslami. Príkazy sú uvedené v tabuľke 3.4.

Napríklad: "M10,20L30,40" obsahuje príkazy: M s argumentami (10, 20) a L (30, 40). Rozdiel vo veľkosti písma v príkaze vyjadruje to, či ide o absolútnu, alebo relatívnu cestu. Ak sú malé znaky jedná sa o relatívne, v prípade veľkých znakov absolútna cesta.

Príkaz	Názov	Parametre
M	moveto	(x y)+
Z	closepath	(none)
L	lineto	(x y)+
H	horizontal lineto	x+
V	vertical lineto	y+
C	curveto	(x1 y1 x2 y2 x y)+
S	smooth curveto	(x2 y2 x y)+
Q	quadratic Bézier curveto	(x1 y1 x y)+
T	smooth quadratic Bézier curveto	(x y)+

Tabuľka 3.4: Príkazy na tvorbu Path elementu

3.3 Vykreslenie obrázku

Snap.svg API povoľuje vloženie bitmapových obrázkov (.jpg alebo .png) na plátno. Používa na to metódu `image` z `Paper` objektu. Parametre metódy `image` sú: zdroj, x, y, šírka, výška. Príklad kódu, ktorý vkladá .jpg obrázok do plátna:

```
var paper = Snap("mojePlatno", 300, 200);
paper.image("obrazok.jpg", 15, 15, 100, 150);
```

3.4 Atribúty elementu

Tvary, ktoré sa dajú nakresliť sa môžu vyfarbiť, orámoviť alebo mnoho iných atribútov sa dá nastaviť. Keď sa vytvorí tvar, tak sa vráti `Element` objekt. Tento objekt má `attr` metódu, ktorá akceptuje key-value pár atribútov. V tomto odseku sa pozrieme na rôzne atribúty, ktoré môžu byť aplikované na naše grafické komponenty používajúc túto metódu.

3.4.1 Výplň elementu - fill

Pozadie pre element nastavím cez metódu `attr` použitím `fill` atribútu ako parameter. Pre jednofarebné výplne je formát vyjadrený cez CSS špecifikáciu. CSS špecifikácia farieb je nasledovná: `#rrggbb` alebo skrátené `#rgb`, `rgb(r, g, b)` reťazec alebo slovné. Napríklad:

```
var kruh = paper.circle(50, 50, 40).attr("fill", "red");
```

Ďalšie spôsoby výplne elementu sú obrázkom, gradientom, alebo vzorom. Pre nastavenie neprehľadnosti nastavíme atribút `opacity` na hodnotou čísla v rozsahu od 0-1. Element bude neviditeľný pri hodnote `opacity`: 0.

3.4.2 Nastavenie okraja elementu - stroke

Elementy môžu mať niekoľko rôznych druhov okrajových atribútov. Prehľad tých najznámejších je v tabuľke 3.5.[20]

Atribút pre <code>attr()</code>	CSS atribút	Poznámka
<code>stroke</code>	<code>stroke</code>	farba výplne okraja
<code>strokeWidth</code>	<code>stroke-width</code>	šírka okraja v px, default je 1
<code>strokeOpacity</code>	<code>stroke-opacity</code>	neprehľadnosť, 0-1
<code>strokeLinecap</code>	<code>stroke-linecap</code>	[<code>"butt"</code> , <code>"square"</code> , <code>"round"</code>], tvar - okraj konca
<code>strokeLinejoin</code>	<code>stroke-linejoin</code>	[<code>"bevel"</code> , <code>"round"</code> , <code>"miter"</code>], tvar - okraj rohu
<code>strokeDasharray</code>	<code>stroke-dasharray</code>	pole čiarok, bodiek., napr.5,3,2

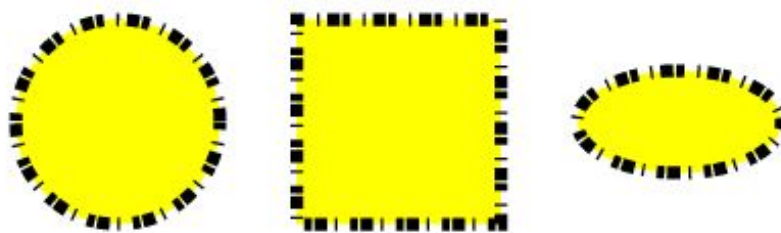
Tabuľka 3.5: Výber možných stroke atribútov

3.5 Zoskupovanie elementov - `Paper.group()`

Niekedy je potrebné použiť rovnaké atribúty, transformácie, alebo animácie pre viacero elementov. V Snap.svg API je možné využiť metódu `Paper.group` alebo `Paper.g`. Group zoskupí viacero elementov do množiny. Príkazom `add` sa dajú pridať ďalšie prvky. V množine sa dajú meniť atribúty pre viacero prvkov naraz volaním metódy `attr`.

Príklad zoskupenia elementov. Výsledné zoskupenie je zobrazené na obrázku 3.2.

```
var paper = Snap();  
var kruh = paper.circle(50, 50, 40);  
var obdlznik = paper.rect(120, 10, 80, 80);  
var elipsa = paper.ellipse(270, 50, 40, 20);  
  
var skupina = paper.g(kruh, obdlznik);  
skupina.add(elipsa);  
  
skupina.attr({  
  fill: 'yellow',  
  stroke: '#000',  
  strokeWidth: 5,  
  strokeDasharray: [3, 5, 1]  
});
```



Obr. 3.2: Príklad zoskupenia elementov a následná zmena atribútov

3.5.1 Maskovanie - `Paper.mask()`

Ekvivalentné správanie ako `Paper.g` až na to, že to je maska. Vráti objekt mask element.

3.6 Gradient

Snap.svg podporuje aplikovanie gradientov cez atribút `fill` s parametrom stringu gradientu. Cez metódu `Paper.gradient(gradient)` sa vytvorí gradientový element, ktorý sa dá nasledovne použiť ako hodnota pre atribút `fill` pre metódy `Element.animate()`, resp. `Element.attr()`. Gradientový string vyrezá nasledovne: `<typ>(<suradnice>)<farby>`.

- Typ môže byť buď lineárny alebo radiálny. Veľkosť písma určuje či ide o absolútne, alebo relatívne súradnice.
- Súradnice špecifikujú lineárny gradient ako vektor `x1, y1, x2, y2` alebo radiálny gradient ako `cx, cy, r`. Nepovinné sú `fx, fy`, ktoré špecifikujú ohniskový bod od centra kruhu.
- Farby je list pomlčkou oddelených hodnôt CSS farieb. Každá farba môže byť nasledovaná vlastným offsetovou hodnotou, oddelenou dvojbodkovým znakom.

Lineárny gradient, vzťažný na horný ľavý roh do pravého dolného rohu. Farby prechádzajú z čiernej cez červenú do bielej.

```
var g = paper.gradient("l(0,0,1,1)#000-#f00-#fff");
```

Lineárny gradient, absolútne z (0,0) do (100,100) z čiernej cez červenú na 25 percentách do bielej.

```
var g = paper.gradient("L(0,0,100,100)#000-#f00:25-#fff");
```

Radiálny gradient, relatívny zo stredu elementu s polomerom polovice šírky, z čiernej do bielej.

```
var g = paper.gradient("r(0.5,0.5,0.5)#000-#fff");
```

Aplikovanie gradientu:

```
paper.circle.attr({fill: g});
```

3.7 Práca s textom - Paper.text(x, y, text)

Vykreslenie textu na plátne namiesto HTML markup s CSS štýlovaným umožňuje animovať a transformovať v rovnakým spôsobom ako iné tvary. Text vytvorení cez metódu `text`. Parametre metódy `text` sú súradnice `x`, `y` a `text`, ktorý sa vykreslí. Vlastnosti textu sa dajú zmeniť volaním metódy `attr`. V tabuľke sú atribúty, ktoré sa dajú zmeniť prostredníctvom metódy `attr`.

Snap atribút	CSS atribút	Poznámka
<code>font</code>	<code>font</code>	napr. "30px Helvetica, sans-serif",
<code>textAnchor</code>	<code>text-anchor</code>	pozícia textu, napr. "middle"
<code>fill</code>	<code>fill</code>	výplň textu farbou, gradientom, vzorom
<code>fontSize</code>	<code>font-size</code>	veľkosť textu
<code>fontFamily</code>	<code>font-family</code>	napr. "monospace"
<code>fontStyle</code>	<code>font-style</code>	štýl písma, napr. kurzíva
<code>fontVariant</code>	<code>font-variant</code>	napr. "small-caps"
<code>fontWeight</code>	<code>font-weight</code>	hrúbka písma, napr. normal, bold, bolder, lighter, 100-900

Tabuľka 3.6: Atribúty na zmenu vlastností elementu text

Príklad zmeny farby:

```
var paper = Snap();  
var text = paper.text(30, 100, "Namestovo");  
  
text.attr({  
  textAnchor: "middle",    fill: "#00b",    fontSize: '16px',  
  fontFamily: "Veranda",  fontStyle: "italic",  fontVariant: "  
  small-caps",  fontWeight: 800,  
});
```

3.8 Transformácie - Element.transform(...)

Transformácia je realizovaná cez metódu `Element.transform()`, ktorá má v parametri transformačný string. Pri transformácii sa dá využiť klonovanie elementov cez metódu `Element.clone()`. Syntax transformačného stringu je vyjadrené z príkazov, ktoré sú v tabuľke 3.8.

Názov	Príkaz	Parameter	Príklad
Posunutie	T, t	x, y	t50,100
Rotácia	R, r	uhol, (bod rotacie x, y)	r45,0,0
Škála	S, s	scale x, y, (scale bod x, y)	S 2,4.5,75,125

Tabuľka 3.7: Syntax transformačného stringu pre Snap API

Transformačný string využíva malé a veľké varianty príkazov. Variant s veľkými písmenami znamená to, že sa transformuje, bez ohľadu na predchádzajúcu transformáciu. Opačne to je pri variante s malými písmenami, ktoré berú na vedomie predošlú transformáciu.[10, p. 52]

V SVG SMIL sa vytvorí k elementu nový tag `animateTransform`, kde sa nastaví `attributeName` na `transform`, a typ a hodnoty sú uvedené v tabuľke 3.8.

Transformation	Popis
<code>translate(x,y)</code>	Posunie súradnicový systém na dané x, y.
<code>scale(xFactor, yFactor)</code>	zmena mierky,
<code>rotate(uhol)</code>	Zrotuje súradnice o daný uhol.
<code>rotate(uhol, centerX, centerY)</code>	Zrotuje súradnice o daný uhol, v daných bodoch.
<code>skewX(uhol)</code>	Skosenie pozdĺž osi X.
<code>skewY(uhol)</code>	Skosenie pozdĺž osi Y.

Tabuľka 3.8: Typy transformácií vo vnútri SVG tagu `animateTransform`

3.9 Animácie

Snap.svg umožňuje animovať SVG grafické komponenty priamo manipulovaním jeho atribútov v JavaScripte.

`Element.animate(attr, ms, easing, callback)`. Parametre sú v tabuľke 3.9

Parameter	Popis
attr	Atribúty požadovaného elementu sú zadané v pároch - typ atribútu a jeho hodnota
duration	Trvanie animácie v milisekundách
easing	zjemnenie animácie, mina objekt
callback	Funkcia, ktorá sa spustí po skončení animácie

Tabuľka 3.9: Parametre pre metódu `animate`

Animácia metód a jednoduchých atribútov animácie

Jednoduché atribúty elementov, ktorých prechod sa zanimuje sú v časti o atribútoch elementov. Ako atribút sa dá použiť aj CSS3 atribút, ale musí byť v úvodzovkách, ak neexistuje v knižnici Snap.svg jeho ekvivalentný názov.

Animovanie path

Pre animovanie zmeny tvaru jedného grafického komponentu na druhý sa využíva atribút `path`. Grafický element sa plynulo zanimuje, pretvaruje na tvar, ktorý je udaný.

3.9.1 Animácie transformácie

Animovanie transformácie atribútu elementu využíva transformačný string. Syntax:

```
Element.animate({ transform: [transformacny string] });
```

3.10 Ďalšie metódy v knižnici Snap.svg

Snap.load(url, callback, [scope])

Načíta externý .svg súbor ako fragment.

Element.append(...), Element.add(...),

Slúži na pridanie, zobrazenie elementu na danom plátne.

Element.stop()

Zastaví všetky animácie pre súčasný element. Vracia súčasný element.

Element.remove()

Odstráni element z DOM.

Element.removeData([key])

Odstráni hodnotu asociovanú s elementom s daným kľúčom. Ak kľúč nie je zadany, odstráni všetky dáta z elementu.

Snap.select(...)

Slúži na wrapovanie DOM elementu špecifikovaný CSS selektorom.

Element.getBBox()

Vráti bounding box opisovač pre daný element. Bounding box vyjadruje hranice elementu cez obdĺžnik.

Vráti objekt s nasledujúcimi atribútmi:

- cx, cy - x, y stredu elementu,
- h, height - výška elementu,

- path - príkaz path pre daný box
- r0 - polomer kruhu, ktorý plne uzatvára box,
- r1 - polomer najmenšieho kruhu, ktorý môže byť opísaný ku box,
- r2 - polomer najväčšieho kruhu, ktorý môže byť opísaný ku box,
- vb - string -box ako viewBox príkaz
- w, width - šírka elementu
- x2 - x súradnica pravej strany,
- x - x súradnica ľavej strany,
- y2 - y súradnica spodnej hrany,
- y - y hornej hrany.

Kapitola 4

Vzorová sada

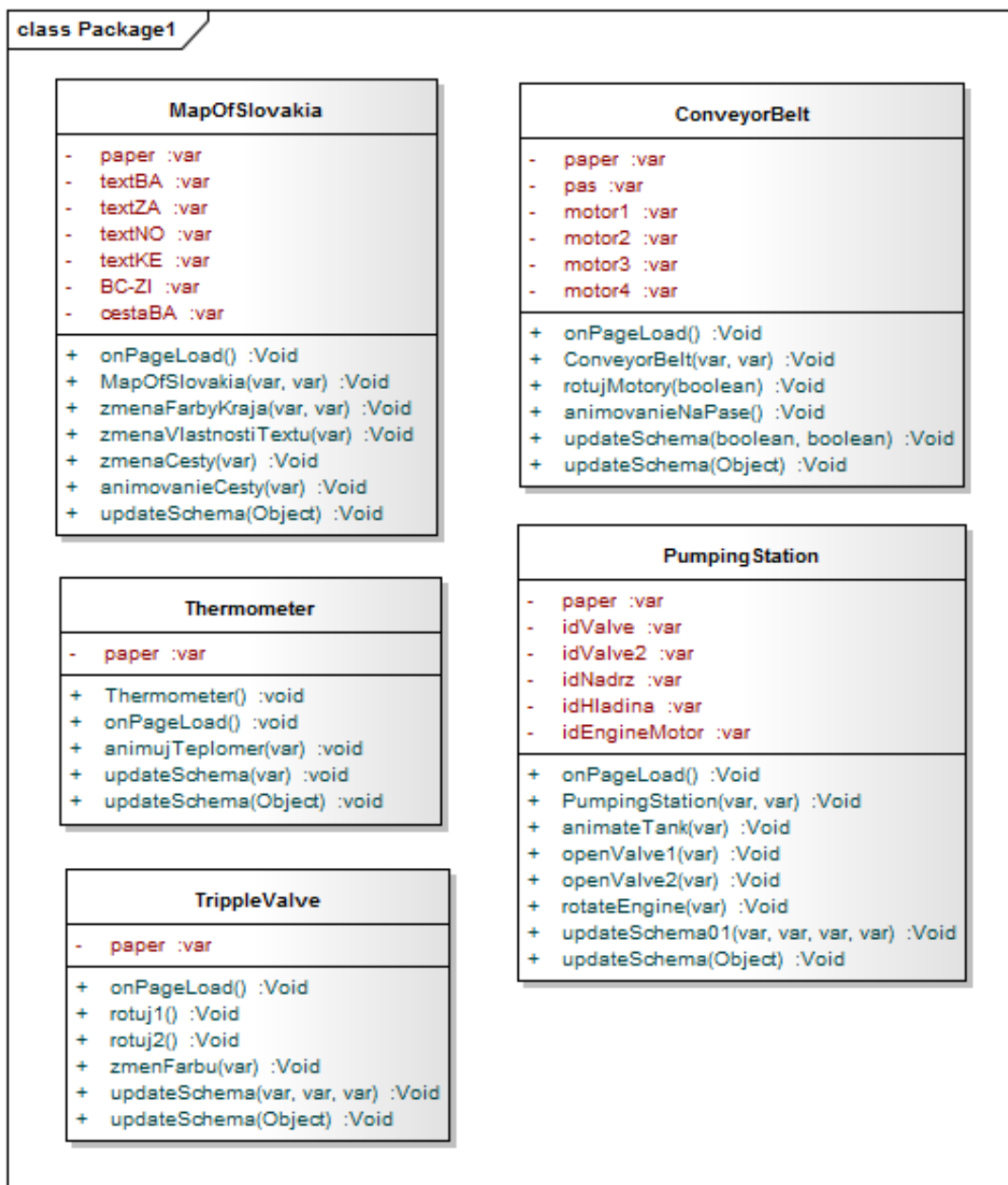
Vzorová sada grafických komponentov SCADA systémov obsahuje:

- Prečerpávacía stanica - obrázok 4.2
- Prepravný pás - obrázok 4.3
- Trojcestný ventil - obrázok 4.4
- Mapa Slovenska - obrázok 4.5
- Thermometer - obrázok 4.6

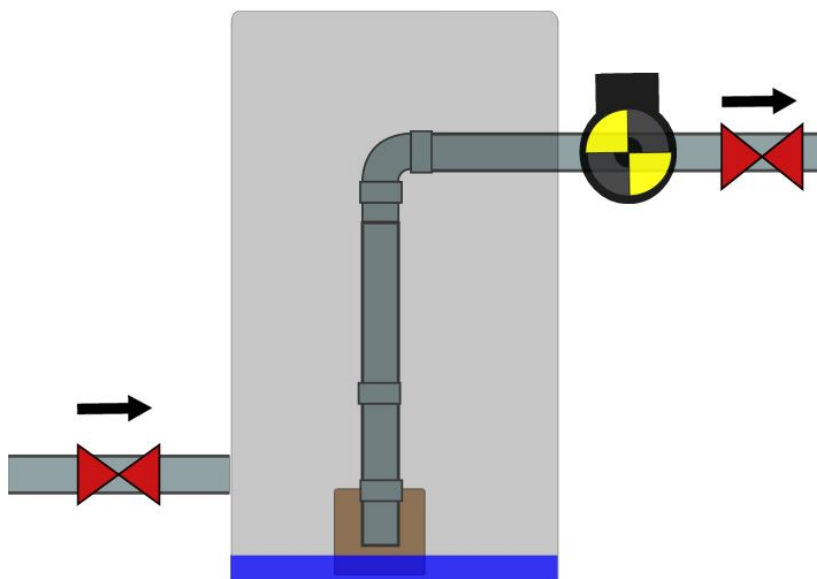
Diagramy tried vzorových grafických komponentov sú na obrázku 4.1

4.1 Diagram tried vzorovej sady

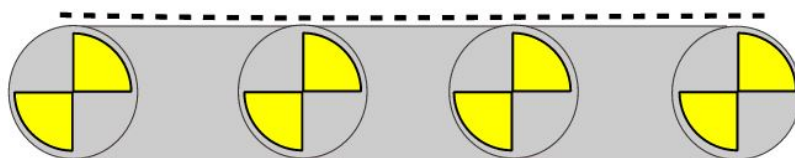
Diagram tried vzorovej sady je na obrázku 4.1 .



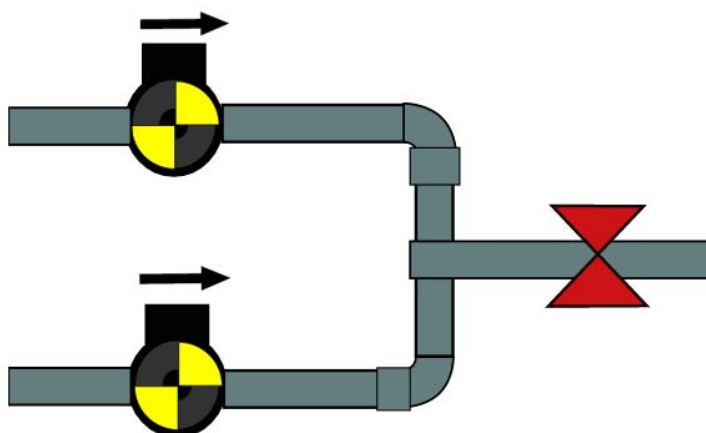
Obr. 4.1: Diagram tried vzorovej sady grafických komponentov



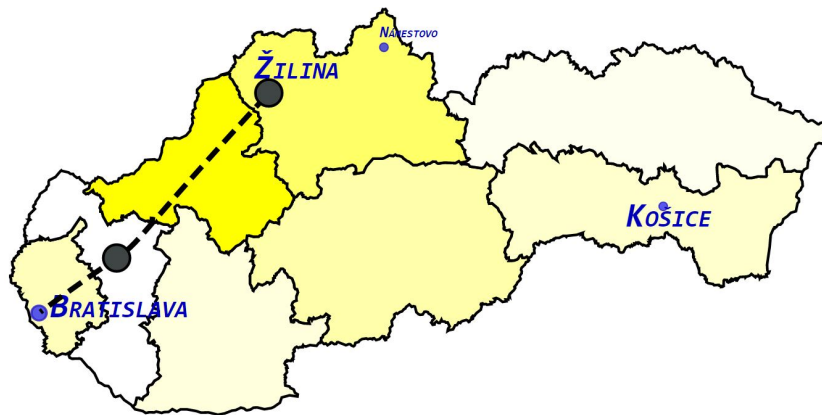
Obr. 4.2: Prečerpávacia stanica



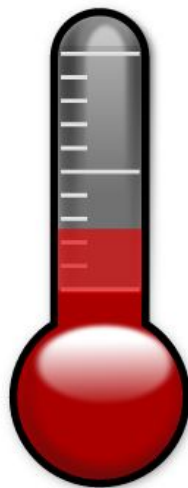
Obr. 4.3: Prepravný pás



Obr. 4.4: Trojcestný ventil



Obr. 4.5: Mapa Slovenska



Obr. 4.6: Termometer

Kapitola 5

Integrácia grafického komponentu

5.1 Vytvorenie prečerpávacej stanice v Inkscape

Nakreslenie jednotlivých častí komponentov prečerpávacej stanice v Inkscape bolo realizované pomocou ľavého bočného panela. Prečerpávacia stanica sa skladá z potrubí, indikátora úrovne hladiny vody, motora, a dvoch symbolov prítoku hladiny. Ako je možné vidieť na obrázku 5.1.

Obr. 5.1: Grafické prostredie programu Inkscape s nakreslenou prečerpávacou stanicou

5.2 Definovanie id v SVG

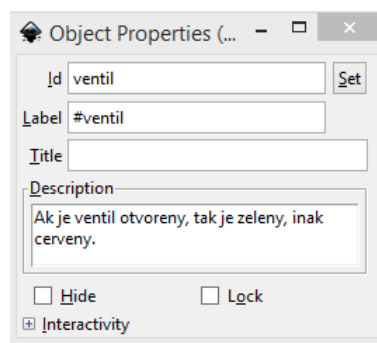
Pre ovládanie JavaScriptom je nutné si pozrieť jednotlivé jedinečné identifikačné názvy. V .svg súboroch sú označované ako id. V zdrojovom kóde .svg súboru je to označené `id="nazovElementu"`. Na ovládanie časti svg elementu v JavaScripte bude realizované cez CSS selektor, kde pristúpim k id svg cez značku `#`. Napríklad: `paper.select("#ventil");`

.

5.2.1 Object Properties

Zistenie id je pomerne jednoduché v Inkscape. Klikneme pravým tlačidlom na daný komponent, ktorého id chceme vedieť, a potom na Objekt Properties.

Po kliknutí sa nám zobrazí okno s názvom Object Properties.



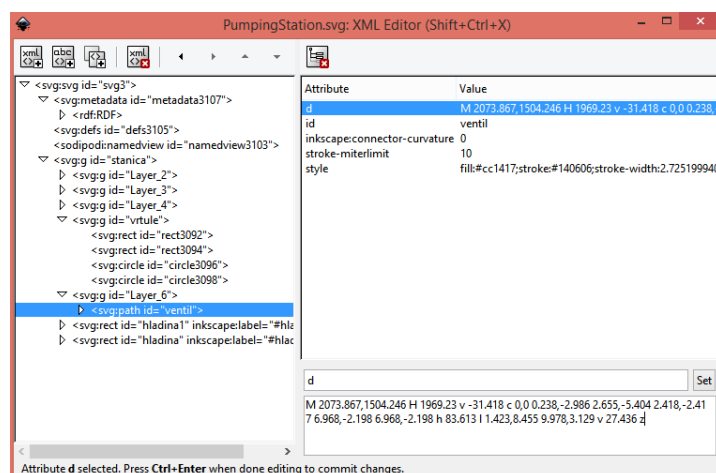
Obr. 5.2: Object Properties

Z obrázka č.5.2 možno vyčítať aké je ID, predvolené sú tam napríklad desc3072. Hodnoty je možné prepísať a zmeniť stlačením tlačidla Set. Pre nás je dôležitá hodnota v kolónke id - ventil.

V okne Object Properties je možné nastaviť script na animovanie. Po kliknutí na Interactivity sa zobrazia ďalšie kolónky, kde je možné zadať akciu, ktorá má nastať.

5.2.2 XML Editor

Ďalší spôsob získania informácií o svg cez Inkscape je cez zabudovaný XML Editor. Stlačením klávesovej skratky SHIFT + CTRL + X alebo v hornej lište v menu vybrať ponuku Edit a na spodku je XML Editor. Následne sa zobrazí okno, ktoré je na obrázku 5.3. XML Editor umožní zistiť ID jednotlivých komponentov, ale i hodnoty atribútov.



Obr. 5.3: Xml Editor v Inkscape

5.3 Integrácia prečerpávacej stanice pre dynamické ovládanie SVG objektu

Súborová štruktúra prečerpávacej stanice:

- index.html
- PumpingStation.js
- PumpingStation.svg
- TestPumpingStation.js

5.3.1 HTML súbor

Do HTML súboru index.html pridáme párový tag `<svg>`. Na toto miesto sa neskôr vykreslí SVG načítané zo súboru cez JavaScript. Môže sa tu uviesť i celý kód SVG obrázka. V prípade, že nebude v dokumente dané, kde presne sa nachádza SVG, tag tak sa pridá na najbližšie voľné miesto.

5.3.2 Kód

```
<svg  
  id="svgStanica"  
  viewBox="0_0_750_600"  
  width="100%"  
  height="100%">  
</svg>
```

5.3.3 Vysvetlenie kódu

Atribúty v tagu sú prispôsobené na to, aby sa grafický element vykreslil responzívne na obrazovku.

- **id** - jedinečný identifikátor.
- **viewBox** - je virtuálne okno, ktorým sa užívateľ uvidí svg obrázok. Je atribút, ktorý povoľuje špecifikovať danú množinu grafických komponentov, aby sa zobrazili v daných súradniciach x, y a šírke, výške. Hodnoty atribútov v viewBox sú štyri čísla - min-x, min-y, width a height.
- **width** a **height** je šírka a výška. Hodnoty atribútov je možné uviesť relatívne v percentách alebo absolútne v pixloch.

Do HTML dokumentu sa pridávajú jednotlivé JavaScriptové knižnice s ktoré bude daný grafický komponent používať.

```
<script src=".. / js / snap.svg-min.js "></script>  
<script src="PumpingStation.js "></script>
```

Musíme sa uistiť, aby sa načítali všetky JavaScriptové knižnice, pred spustením funkcií. To zabezpečíme pridaním onload do tagu <body>.

```
<body onload="onPageLoad();">
```

5.4 PumpingStation.js

V súbore PumpingStation.js sú metódy na vizualizáciu grafického komponentu. V tejto časti je popis jednotlivých funkcií. Interface prečerpávacej stanice:

- onPageLoad();
- PumpingStation(nazovFileSVG, idDOMsvgElement);
- openValve1(isOpenValve1);
- animateTank(intPercent);
- rotateEngine(isRotating);
- openValve2(isOpenValve2);
- updateSchema01(isOpenValve1, intPercent, isRotating, isOpenValve2);
- updateSchema(updateData);

5.4.1 onPageLoad()

onPageLoad() sa spustí pri načítaní tela HTML súboru index.html. Následne spustí PumpingStation(par1, par2). Prvý parameter je udaný svg súbor vytvorení programom Inkscape. Druhý parameter je id tagu svg, kde sa vykreslí prečerpávacia stanica.

```
function onPageLoad() {  
    PumpingStation("PumpingStation.svg", "#svgStanica");  
}
```

5.4.2 PumpingStation(par1, par2)

Parametre pre PumpingStation sú názov svg súboru, a id, ktoré sa nachádza v tagu <svg> html súbore.

Vo vnútri PumpingStation sa nachádza inicializácia globálnych atribútov.

```

var paper;

var idValve, idValve1 idNadrz, idHladina, idEngineMotor;

function PumpingStation(nazovFileSVG, idDOMsvgElement) {
    paper = Snap(idDOMsvgElement);
    Snap.load(nazovFileSVG, function (f) {
        idHladina = "#hladina";
        idNadrz = "#nadrz";
        idValve = "#ventil";
        idValve2 = "#ventil2";
        idEngineMotor = "#engineMotor";
        paper.append(f);
    });
}

```

Atribút `paper` je globálna premenná cez ktorú sa bude pristupovať k metódam JavaScriptovej knižnice `Snap.svg`. Jej parameter je referencia na plochu, kde bude vykreslené SVG elementy.

Vo vnútri funkcie sa volá z `Snap.svg` API funkcia `load()`. Má parametre názov súboru, a funkciu, ktorú spustí následne po načítaní. Vo vnútri funkcie `load()` sa inicializujú globálne premenné. Premenné obsahujú CSS selektor id získaný z programu Inkscape. Premenné budú slúžiť ako parameter pri volaní funkcie `paper.select()`. Pre prípadné zmeny v id, sa dané id zmení na jednom mieste a nemusí sa prepisovať všade.

Význam premenných je nasledový:

- `idHladina` - indikátor prítoku hladiny vody,
- `idNadrz` - je miesto, kde bude prichádzať prítok vody,
- `idValve1`, `idValve2` - je ventil,
- `idEngineMotor` - je symbol rotora motora.

Načítaný `.svg` súbor sa zobrazí volaním metódy `append`.

5.4.3 animateTank(percento)

V tejto metóde je zobrazená vizualizácia prítoku hladiny do nádrže. Ako parameter slúži percento vyplnenia.

```
function animateTank(percento){
    var height = paper.select(idNadrz).getBBox().height;
    var y = paper.select(idNadrz).getBBox().y;
    var newHeight = height * (percento/100);
    var newY = y + height - newHeight;

    paper.select(idHladina).animate({
        y: newY,
        height: newHeight
    }, 1000);
}
```

Animovanie je realizované cez metódu animate zo Snap.svg API. Význam lokálnych premenných:

- height - výška nádrže do, ktorej bude pritekať voda, získaná volaním metódy getBBox(),
- y - súradnica y - navýšením, alebo znížením hladiny sa mení súradnica y, a x zostáva nezmenená,
- newHeight - vypočítaná nová výška hladiny podľa daného percenta,
- newY - je vypočítaná súradnica - osi y.

Volaním metódy animate, sa zanimuje daný element, v tomto prípade to bude hladina nádrže. Metóda má nasledovné parametre:

- atribúty - v pároch udané atribúty, ktoré sa zmenia. Zmení sa iba výška a os y, ostatné atribúty zostávajú nezmenené.
- rýchlosť animácie udaná v milisekundách.

5.4.4 openValve1(isOpen)

Indikátor prítoku hladiny do prečerpávacej stanice je ventil, ktorý mení farbu. Ak je nastavený parameter isOpen na true, tak farba ventila bude zelená, v inom prípade červená. Zmena farby je realizovaná volaním metódy attr zo Snap.svg API, ktorá nastavila atribút fill daného elementu na danú farbu. Farba zmeny je zapísaná termálnym operátorom. Pre zmenu farby v druhom elemente je zápis identický až na to, že je zmena pri volaní select - na idValve2.

```
function openValve1(isOpen){
    paper.select(idValve).attr({
        fill: ((isOpen === "true") ? "green" : "red")
    });
}
```

5.4.5 rotateEngine(isRotating)

Táto metóda bude rotovať až dovtedy pokiaľ sa nezmení parameter isPaused. Rotácia motora je zabezpečená vnorenou metódou rotateLeft. Tá sa volá v toggleRotation(), kde v má parameter element vrtuliek čerpadla.

Popis metódy rotateLeft():

- animationRunning - nastavené na true, znamená to, že animácia ešte beží
- stringT0 - je transformačný string - potrebný na rotáciu vrtuliek čerpadla. Uhol rotácie je nulový, a súradnice stredovej osi x, y sú získané volaním metódy elementu getBBox().cx a getBBox().cy.
- transform() - je metóda z Snap.svg knižnice, ktorá umožní nastaviť transformáciu rotácie, kde parametrom je transformančný string.
- if(!isPaused).. - je podmienka, ktorá zaručí opakované rotovanie elementu.
- stringT360 - je transformačný string, ktorý zrotuje element o uhol 360 stupňov zo stredom súradnicovej osi elementu.

- `animate()` - je metóda, kde animujem transformáciu z uhlu 0 stupňov na 360 stupňov. Ako parametre sú:
 - párový atribút `transform` s hodnotou transformačného stringu,
 - rýchlosť animácie v milisekundách - 2000 ms,
 - mina objekt, ktorý zaručí lineárny plynulý pohyb,
 - callback funkcia, ktorá sa spustí po dokončení animácie. Toto zabezpečí opakované spustenie animácie rotácie elementu.
- `else` - ak skončí animácia - tak sa nastaví `animationRunning` na hodnotu `false`.

Kód metódy

```
var isPaused = true;
var animationRunning = false;
function rotateEngine(isRotating){
    isPaused = isRotating;

    function toggleRotation() {
        if (!animationRunning && isPaused) {
            isPaused = false;
            rotateLeft(paper.select(idEngineMotor));
        } else {
            isPaused = true;}
    }
    toggleRotation();

    function rotateLeft(element) {
        animationRunning = true;
        var stringT0 = "R0,"+ element.getBBox().cx  + ","+
        element.getBBox().cy;
```

```

    element.transform(stringT0);

    if (!(isPaused)) {
        var stringT360 = "R360,"+ element.getBBox().cx + ","+
        element.getBBox().cy;
        element.animate({
            transform: stringT360 },
            2000,
            mina.linear,
            rotateLeft.bind(null, element));
    } else { animationRunning = false;}
}
}

```

5.5 updateSchema()

Metóda updateSchema01 zabezpečí aktualizáciu grafického komponentu. Parametre má nasledovné:

- isOpenValve - boolean hodnota, ktorá nastaví indikátor prítoku vody, v prípade true - na zelený, v opačnom na červený.
- intPercento - je integer hodnota v rozsahu od 0-100. Vyjadruje percentuálne naplnenie nádrže.
- isRotating - boolean hodnota, ktorá nastaví otáčanie motora, až pokým sa znova nezavolá.
- isOpenValve - boolean hodnota druhého ventila.

```

function updateSchema01(isOpenValve1, intPercent, isRotating,
isOpenValve2) {
    openValve1(isOpenValve1) ;

```

```

    animateTank(intPercent);
    rotateEngine(isRotating);
    openValve2(isOpenValve2);
}

```

Dáta vyjadrené vo formáte JavaScript Object Notation (JSON). Zároveň je to interfejsná metóda k grafickému komponentu Pumping Station. Toto je príklad objektu update dátami na nastavenie hladiny nádrže na 20 percent a s ventilami, a rotorom zapnutým.

```

var updateData = {
    "valve1": "true",
    "tank": "20",
    "engineRotation": "true",
    "valve2": "true"
};

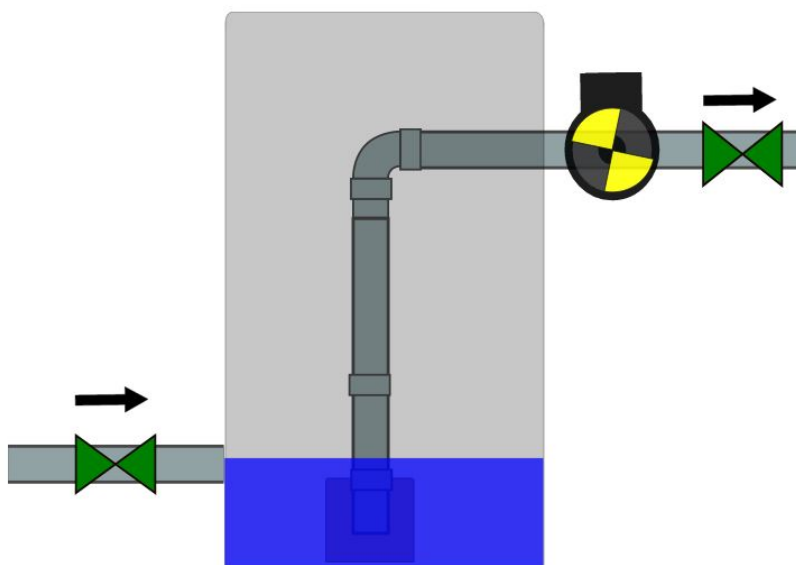
```

Metóda updateSchema s parametrom objekt updateData, aktualizuje grafický komponent na dané hodnoty. Zároveň je to interfejsná metóda k REST API prečerpávacej stanice. Na obrázku 5.4 je zobrazený animovaný komponent s vopred pripravenými dátami.

```

function updateSchema(updateData){
    updateSchema01(updateData.valve1, updateData.tank, updateData
        .engineRotation, updateData.valve2);
}

```

Obr. 5.4: Prečerpávacia stanica po vykonaní príkazu `updateSchema(updateSchema);`

Kapitola 6

REST API

6.1 REST API pre grafické komponenty

Grafické komponenty pre vizualizáciu dát zo systému D2000 budú umiestňované na HTML stránkach použitých ako súčasť web rozhrania frameworku D2000 WebSuite.

Tento framework je založený na technológii Java Enterprise Edition a Java Server Faces.

Životný cyklus webovej stránky - načítanie stránky s technologickou schémou, pre prvé zobrazenie kompletnej schémy je potrebný plný dáta set.

Príklad REST URL:

`http://localhost:8080/scada-demo/rest/pumpingstation/gatfulldataset`

Príklad JSON dát z grafického komponentu prečerpávacia stanica:

```
var updateData = {  
    "valve1": "false",  
    "tank": "0",  
    "engineRotation": "false",  
    "valve2": "false"  
};
```

Zobrazenie stránky v rámci jednej http session, typicky vo web aplikáciách kde sa užívateľ prihlási pomocou mena a hesla, trvanie jeho session je obmedzené na predom

stanovený čas, napríklad jednu hodinu.

Pri zobrazení zložitejšej technologickej schémy, je potrebné optimalizovať množstvo prenesených dát a interakciu z DOM stránky. Preto je počas zobrazenia schémy výhodne implementovať čiastočne aktualizácie, ktoré menia len dotknuté časti schémy a nie celú schému ako je tomu pri načítaní stránky.

Príklad REST URL:

`http://localhost:8080/scada-demo/rest/pumpingstation/getvalvestatus` príklad
JSON dát:

```
var updateDataTank = { "tank": "86" };
```

príklad REST URL

`http://localhost:8080/scada-demo/rest/pumpingstation/getrotorstatus` príklad
JSON dát: príklad REST URL

`http://localhost:8080/scada-demo/rest/pumpingstation/getwaterlevel` príklad
JSON dát:

6.2 Data binding pre system D2000

Priama väzba v prípade jednoduchých schém. Jednoduchá schéma predstavuje vizualizáciu meraného alebo počítaného bodu v systéme D2000. Takáto vizualizácia je realizovaná pomocou widgetu jedna sa o mapovanie 1:1. Zložitejšie schémy pozostávajúce z vizualizácie výrobného procesu alebo komplexnej technológie sú mapované vo vzťahu n:1. Jedna schéma vizualizuje dáta z n meraných alebo počítaných bodov, prípadne získava dáta cez asynchrónne volania Remote Procedure Call (RPC) systému D2000.

Ak to vyžaduje logika aplikácie, server implementuje dátový zdroj (service), ktorý agreguje dáta a systémové udalosti. Pre dosiahnutie real-time odozvy web rozhrania výhodne použiť obojstrannú komunikáciu medzi web browserom a serverom - technológiu web sockets.

Kapitola 7

Analýza výkonnosti a obmedzení

7.1 Test výkonnosti knižníc Snap.svg.js a Raphael.js - animate()

Vytvorila som test prostredníctvom webovej stránky, ktorá umožňuje robiť spustiteľné testy na rôznych platformách. Test sledoval počet operácií za sekundu. Čím viac operácií sa vykonalo, tým lepšie. Test sa spustil 100 krát za sebou najprv pre Raphael.js a potom pre Snap.svg.js.

TODO Odkaz testu: <http://jsperf.com/snap-svg-raphael-js-animate>

Test spočíva vo vytvorení kruhu na plátne na náhodných súradniciach a následne zanimovanie. Boli spustené animácie na kruh:

- náhodnej polohy x, y,
- zmeny farby výplne kruhu na náhodnú farbu,
- zmeny farby okraja na náhodnú farbu,
- zmeny šírky okraja na náhodnú hodnotu v rozsahu od 0 po 2.

Testoval sa nasledovný kód: Pred spustením testu som si zadefinovala:

```
<script src="https://raw.githubusercontent.com/adobe-webplatform/Snap.svg/master/dist/snap.svg.js"></script>
```

```
<script src="https://raw.githubusercontent.com/DmitryBaranovskiy/raphael/master/raphael.js"></script>
```

```
<div id="raphael"></div>
```

```
<svg id="snap"></svg>
```

```
<script>
```

```
var s = Snap('#snap');
```

```
var r = Raphael('raphael', 1000, 1000);
```

```
s.width=1000;
```

```
s.height=1000;
```

```
</script>
```

Testovací kód pre knižnicu Raphael.js:

```
var raphaelCircle = r.circle(Math.random() * 1000, Math.random()  
() * 1000, Math.random() * 10);
```

```
raphaelCircle.animate({  
  x: Math.random() * 1000,  
  y: Math.random() * 1000,  
  fill: '#' + (Math.random() * 0xFFFFFFFF << 0).toString(16),  
  stroke: '#' + (Math.random() * 0xFFFFFFFF << 0).toString(16),  
  "stroke-width": Math.random() * 2,  
}, 10);
```

Testovací kód pre knižnicu Snap.svg.js:

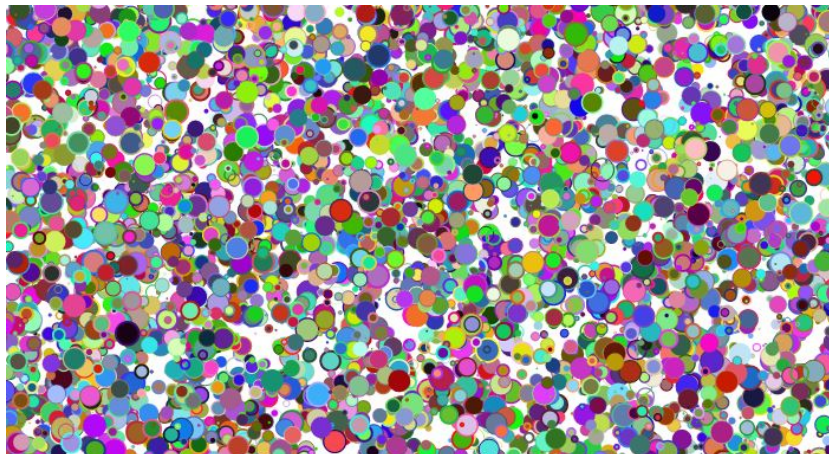
```
var raphaelCircle = r.circle(Math.random() * 1000, Math.random()  
() * 1000, Math.random() * 10);
```

```
raphaelCircle.animate({  
  x: Math.random() * 1000,  
  y: Math.random() * 1000,  
  fill: '#' + (Math.random() * 0xFFFFFFFF << 0).toString(16),  
  stroke: '#' + (Math.random() * 0xFFFFFFFF << 0).toString(16),
```

```

    "stroke-width": Math.random() * 2,
  }, 10);

```



Obr. 7.1: Po spustení testovania výkonnosti knižníc Raphael.js vs Snap.svg.js - zobrazenie spustených animácií

Na obrázku 7.1 je výsledok animácie vykonaného testu. Test som spustila na viacerých platformách a webových prehliadačoch. Príklad výsledku pre konkrétny webový prehliadač je na obrázku 7.2

Testing in Mozilla/5.0 (Windows NT 6.3; WOW64; Trident/7.0; .NET4.0E; .NET CLR 3.5.30729; .NET CLR 2.0.50727; .NET CLR 3.0.30729; InfoPath.3; rv:11.0) like Gecko		
Test		Ops/sec
Raphael	<pre> var raphaelCircle = r.circle(Math.random() * 1000, Math.random() * 1000, Math.random() * 10); raphaelCircle.animate({ x: Math.random() * 1000, y: Math.random() * 1000, fill: '#' + (Math.random() * 0xFFFFFFFF << 0).toString(16), stroke: '#' + (Math.random() * 0xFFFFFFFF << 0).toString(16), "stroke-width": Math.random() * 2, }, 10); </pre>	3,601 ±4.76% 11% slower
Snap.svg.js	<pre> var raphaelCircle = r.circle(Math.random() * 1000, Math.random() * 1000, Math.random() * 10); raphaelCircle.animate({ x: Math.random() * 1000, y: Math.random() * 1000, fill: '#' + (Math.random() * 0xFFFFFFFF << 0).toString(16), stroke: '#' + (Math.random() * 0xFFFFFFFF << 0).toString(16), "stroke-width": Math.random() * 2, }, 10); </pre>	4,078 ±5.55% fastest

Obr. 7.2: Po spustení testovania výkonnosti knižníc Raphael.js vs Snap.svg.js - výsledky pre konkrétny prehliadač

Výsledky testu sú nasledovné:

- pre mobilné zariadenia a ich príslušné webové prehliadače sú v tabuľke 7.1
- pre súčasné a staršie verzie webových prehliadačov, ktoré sú spustiteľné na počítačoch sú v tabuľke 7.2
- pre staršie verzie webových prehliadačov sú v tabuľke

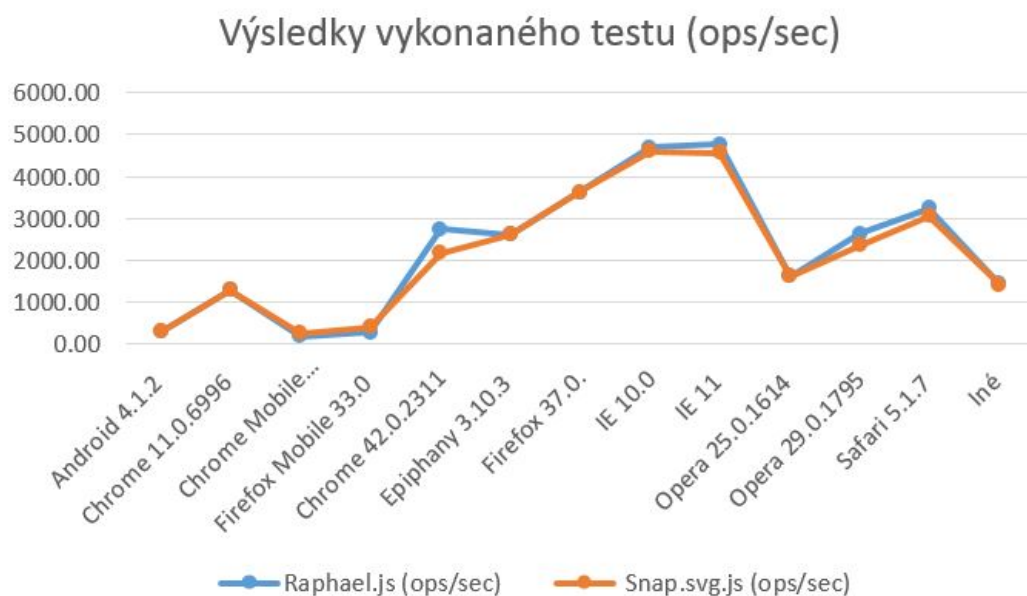
Webový prehliadač	Raphael.js (ops/sec)	Snap.svg.js (ops/sec)
Android 4.1.2	292	292
Chrome 11.0.6996	1 283	1 283
Chrome Mobile 42.0.2311	181	266
Firefox Mobile 33.0	285	414

Tabuľka 7.1: Výsledky testu pre mobilné zariadenia

Webový prehliadač	Raphael.js (ops/sec)	Snap.svg.js (ops/sec)
Chrome 42.0.2311	2 753	2 171
Epiphany 3.10.3	2 609	2 609
Firefox 37.0.	3 626	3 626
IE 10.0	4 688	4 593
IE 11	4 771	4 565
Opera 25.0.1614	1 627	1 627
Opera 29.0.1795	2 626	2 352
Safari 5.1.7	3 249	3 050
Iné	1 432	1 427

Tabuľka 7.2: Výsledky testu pre deskopové zariadenia

Súhrnný graf výsledkov webových prehliadačov je na obrázku 7.3. Aritmetický priemer pre knižnicu Raphael.js je 2263.230769 a pre Snap.svg.js je 2175. Najlepšie výsledky dosiahol Internet Explorer verzia 11.



Obr. 7.3: Graf výsledkov pre webové prehliadače

7.2 Raphael.js vs Snap.SVG.js vs SVG.js - attr()

Tento test bol zameraný na porovnanie výkonosti troch knižníc, ktoré umožňujú animáciu SVG. V teste sa vytváralo hromadne kruhy s náhodnými hodnotami a následne po vykonaní iterácie sa vymazali. Iterácií prebehlo 100.

multiple circles moved to the back

Kód testu:

```
<script src="jquery.min.js"></script>
<script src="snap.svg.js"></script>
<script src="raphael.js"></script>
<script src="svg.js"></script>
```

```
<div id="raphael"></div>
<svg id="snap"></svg>
<svg id="svgjs"></svg>
```



```

<script>
var ncircles = 100;
var circles;

removecircles = function () {
    for (var i=0; i<circles.length; i++) {
        circles[i].remove();
    }
    circles=[];};

var r = Raphael('raphael', 800, 600);
rapcirc = function () {
    var raphaelCircle = r.circle(400, 300, 150);
    raphaelCircle.attr({
        cx: raphaelCircle.attr('cx') + Math.random() * 50 - 25,
        cy: raphaelCircle.attr('cy') + Math.random() * 50 - 25
    }).toBack();
    circles.push(raphaelCircle);
    if (circles.length>=ncircles) removecircles();
};

var s = Snap('#snap');
s.width=800;
s.height=600;
snapcirc = function () {
    var snapCircle = s.circle(400, 300, 150);
    snapCircle.attr({
        cx: snapCircle.attr('cx') + Math.random() * 50 - 25,
        cy: snapCircle.attr('cy') + Math.random() * 50 - 25
    }).prependTo(s);
    circles.push(snapCircle);
};

```

```

    if (circs.length>=ncircs) removecircs();
};

var draw = SVG('svgjs').size(800, 600);
svgcirc = function () {
    var svgjsCircle = draw.circle(300).center(400, 300);
    svgjsCircle.attr({
        cx: svgjsCircle.attr('cx') + Math.random() * 50 - 25,
        cy: svgjsCircle.attr('cy') + Math.random() * 50 - 25
    }).back();
    circs.push(svgjsCircle);
    if (circs.length>=ncircs) removecircs();
};
</script>

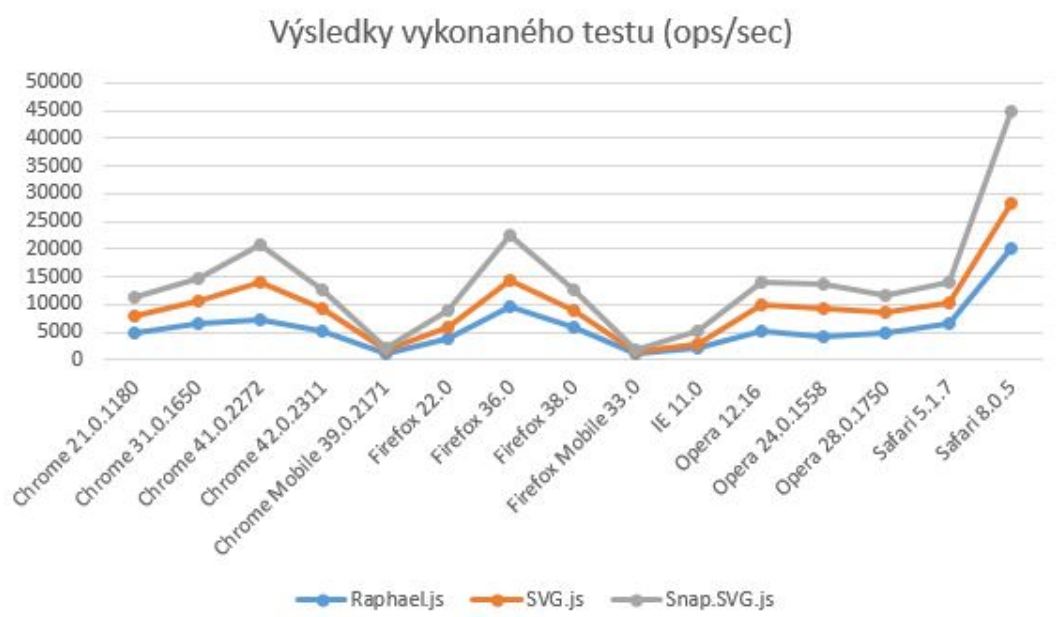
```

Výsledok testu pre webový prehliadač je na obrázku 7.4

Testing in Chrome 42.0.2311.135 32-bit on Windows NT 6.3 64-bit		
	Test	Ops/sec
Raphael	rapcirc();	5,497 ±4.47% fastest
SnapSVG	snapcirc();	3,445 ±3.64% 37% slower
SVG.js	svgcirc();	3,582 ±7.22% 37% slower

Obr. 7.4: Test pre funkciu attr() pre webový prehliadač Chrome 42

Výsledky vykonaných testov vo viacerých webových prehliadačoch sú na obrázku 7.5.



Obr. 7.5: Graf výsledkov pre webové prehliadače

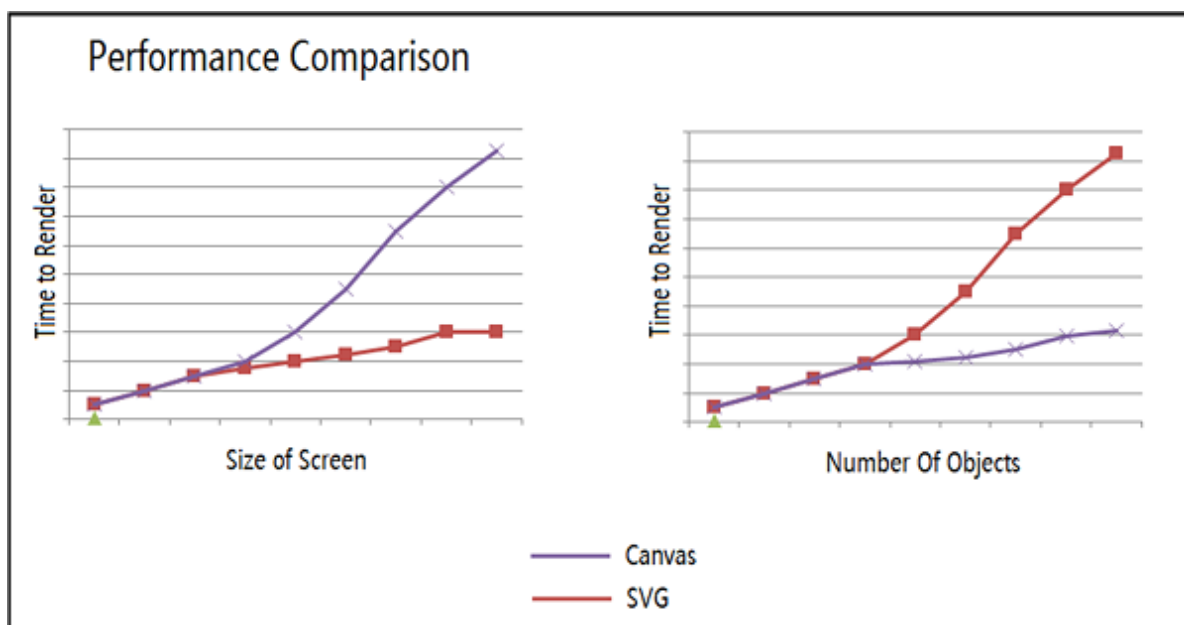
Webový prehliadač	Raphael.js	SVG.js	Snap.SVG.js
Chrome 21.0.1180	4875	3147	3164
Chrome 41.0.2272	7268	6866	6589
Chrome 42.0.2311	5261	3976	3427
Chrome Mobile 39.0.2171	1073	746	398
Firefox 22.0	3946	1856	3158
Firefox 38.0	5835	2965	3783
Firefox Mobile 33.0	1156	309	426
IE 11.0	2272	724	2208
Opera 12.16	5239	4645	4092
Opera 24.0.1558	4150	5134	4461
Opera 28.0.1750	4830	3827	3154
Safari 5.1.7	6709	3475	3919
Safari 8.0.5	20226	8084	16438

Tabuľka 7.3: Výsledky testu pre viaceré webové prehliadače (ops/sec)

7.3 SVG vs Canvas

Na obrázku 7.6 sú zobrazené dva grafy, ktoré porovnávajú vlastnosti Canvas a SVG.

Všeobecne platí, ak sa veľkosť obrazovky zvýši, plátno sa začne zmenšovať až na toľko pixelov, koľko je potrebné vykresliť. Pri SVG je to tak, ak sa počet objektov na obrazovke zvýši, tak začne neustále dané objekty pridávať do DOM. Tieto merania nie sú nevyhnutné presné a môžu sa zmeniť v závislosti od implementácie a platformy, hardvérovej akcelerácie grafiky. [28]



Obr. 7.6: Porovnanie výkonnosti Canvas vs. SVG

Záver

Cieľom práce bolo navrhnúť riešenie pre vizualizáciu SCADA komponentov vo webovom prehliadači. Mojou úlohou v záverečnej práci bolo vytvoriť univerzálny postup pre animáciu komponentov.

Doterajšie riešenie bolo nepraktické, nebolo modulárne a vyžadovalo zásah do riadiacej časti animácie. Navrhnuté riešenie použitím kombinácie SVG a JavaScriptovej knižnice je vhodné a užitočné pre vizualizáciu SCADA systémov.

Úspešne som vytvorila vzorovú sadu grafických komponentov, kde som si vyskúšala vizualizovanie. Grafický komponent som v .svg formáte nakreslila cez Inkscape. Pomocou knižnice Snap.svg som pridávala funkcie na animovanie a manipulovanie grafického komponentu.

Výsledná sada je vhodná pre platformy ako napríklad tablety, mobilné telefóny a iné. Všetky zdrojové kódy práce sú v Git repository na GitHubu[2]. Moje riešenie používa knižnicu a softvér, ktorý sú open-source.

Literatúra

- [1] Mavrody S., *Sergey's HTML5 & CSS3 Quick Reference: HTML5, CSS3 and APIs*, 3rd edition, Belisso, 2012, ISBN 978-0-98338-674-2
- [2] <https://github.com/chovancova/project>
- [3] <https://www.d2000.eu/sk/lang/o-aplikacii/popis/softverova-technologiea-realneho-casu>
- [4] MICROSTEP-LM s.r.o. : *Informatizácia výroby - SCADA a MES systémy v priemyselnej praxi*, 1.10.2009, Dostupné z http://www.microstep-lm.sk/_uvod/VIS.pdf, (30-4-2015)
- [5] <http://www.smashingmagazine.com/2012/01/16/resolution-independence-with-svg/>
- [6] <https://www.animatron.com/>
- [7] <https://code.google.com/p/svg-edit/>
- [8] <http://api.jquery.com/animate/>
- [9] <http://julian.com/research/velocity/>
- [10] Dawber D., *Learning Raphael JS Vector Graphics*, Packt Publishing, 2013, ISBN 978-1-78216-916-1
- [11] Wilson CH., *RaphaelJs Graphic and visualization on the web* , O'Reilly Media, 2013, ISBN 978-1-449-36536-3

- [12] Haverbeke M., *Eloquent Javascript* 2. vyd. No Starch Press, 2014, ISBN 978-1-59327-584-6
- [13] Zakas N. Z., *JavaScript pro webové vývojáře Programujeme profesionálně*, 1. vyd. Brno:Computer Press, a.s., 2009, ISBN 978-80-251-2509-0
- [14] Suehring S., *JavaScript krok za krokem*, 1. vyd. Brno:Computer Press, 2008, ISBN 978-80-251-2241-9
- [15] Zakas N. C., McPeak J., Fawcett J., *Profesionálně Ajax*, Zoner Press, 2007, ISBN 978-80-86815-77-0
- [16] Eisenberg D. J., *SVG Essentials*, O'Reilly Media 2002, ISBN 978-0-596-00223-7, dostupné na http://commons.oreilly.com/wiki/index.php/SVG_Essentials
- [17] Richardson L., Amundsen M., *RESTful Web APIs* 1. vyd. O'Reilly Media, 2013, ISBN 978-1-449-35806-8
- [18] Allamaraju S., *RESTful Web Services Cookbook* 1. vyd. O'Reilly Media, 2010, ISBN 978-0-596-80168-7
- [19] <http://www.w3.org/TR/html/>
- [20] <http://www.w3.org/TR/SVG/painting.html#StrokeProperties>
- [21] http://www.w3schools.com/html/html5_svg.asp
- [22] www.w3schools.com/svg/svg_inhtml.asp
- [23] The JavaScript SVG library for the modern web, <http://snapsvg.io/>.
- [24] <http://raphaeljs.com/>
- [25] <http://d3js.org/>
- [26] <http://www.svgjs.com/>
- [27] Inkscape is a professional vector graphics editor for Windows, Mac OS X and Linux. It's free and open source. <http://www.inkscape.org/en/about/features/>

[28] [https://msdn.microsoft.com/library/gg193983\(v=vs.85\).aspx](https://msdn.microsoft.com/library/gg193983(v=vs.85).aspx)

Zoznam skratiek

API Application Programming Interface

CSS Cascading Style Sheets

D3 Data Driven Document

DOM Document Object Model

DPI Dots Per Inch

GIF Graphics Interchange Format

HTML Hyper Text Markup Language

JPEG Join Photographic Experts Group

JSON JavaScript Object Notation

PPI Pixels Per Inch

REST Representational State Transfer

RGB Red Green Blue

RPC Remote Procedure Call

SCADA Supervisory Control and Data Acquisition

SMIL Synchronized Multimedia Integration Language

SVG Scalable Vector Graphics

VML Vector Markup Language

W3C World Wide Web Consortium

WYSIWYG What You See Is What You Get

XML EXtensible Markup Language