VILNIAUS UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS
PROGRAMŲ SISTEMŲ KATEDRA

Vektorinė grafika žiniatinklyje

Vector Graphics in the World Wide Web

Magistro baigiamasis darbas

Atliko:	Stanislav Gerasim	(parašas)
Darbo vadovas:	dr. Andrius Kurtinaitis	(parašas)
Recenzentas:	Simas Šaltenis	(parašas)

Vilnius – 2009

Santrauka

Šiame darbe nagrinėjamos vieno vektorinės grafikos formato – SVG – taikymo galimybės kuriant žiniatinklio aplikacijas, svetaines. Darbe pateikti keturi SVG taikymo variantai – „SVG panaudojimas su JavaScript ir CSS“, „SVG paslėpimas po JavaScript objektais“, „SVG paslėpimas po XBL elementais“, „SVG ir elementai symbol/use“. Kiekvienu atveju nurodoma, kaip galima realizuoti naudotojo sąsajos elementus, išvardinti metodo privalumai, trūkumai. Taip pat visais keturiais metodais realizuotas veikiantis pavyzdys – vektorinės grafikos redaktorius, kuris parodo, kad jau šiandien SVG galima taikyti savo projektuose.

Darbe aprašomi bendri SVG integravimo į žiniatinklio turinį principai, trumpai palyginama SVG ir Flash technologija, susipažinama su minėtais realizavimo variantais, naudotojo sąsajos elementais, pagal tam tikrus kriterijus pateikiamos rekomendacijos, kada koks SVG taikymo variantas yra parankesnis.

Darbo raktiniai žodžiai: SVG, naudotojo sąsajos elementai, taikymas, integravimas, metodai.

Summary

This work covers applications of one of the vector graphics format – SVG in internet applications, sites. The work presents four SVG usage variants – “SVG with JavaScript and CSS”, “SVG behind JavaScript objects”, “SVG behind XBL elements”, “SVG and symbol/use elements”. In every case examples of implementing user interface elements are given. Also advantages and disadvantages are listed for mentioned methods. As an example vector graphics redactor was implemented in four different ways, which shows that SVG can be used for today’s applications.

The work describes common SVG integration principles in the web, briefly compares SVG and Flash, introduces four SVG usage variants, user graphics interface elements, gives recommendations which method is better according to chosen criteria.

Key words: SVG, user interface elements, application, integration, methods.

Turinys

Įvadas

Pastaruoju metu žiniatinklis tapo neatsiejama verslo dalimi. Vis daugiau ir daugiau įvairių, ne tik informacinių technologijų paslaugas teikiančių mažų ir vidutinio dydžio įmonių, siekdamos išlikti rinkoje, pranešti apie save, pritraukti naujus klientus ir tuo aplenkti savo konkurentus, atkreipia dėmesį į žiniatinklio panaudojimo galimybes. Sukurdama internetinę svetainę arba, jei leidžia galimybės, portalą, įmonė gali ne tik reklamuoti savo vardą, bet ir pateikti potencialiems klientams daug naudingos ir aktualios informacijos apie save, savo produktus, savo veiklą, siūlomas paslaugas ir t.t. Internetinė svetainė gali būti ne tik reklamavimosi instrumentas, bet ir verslo dalis. Norint sudominti klientą, pritraukti jo dėmesį, išsiskirti iš kitų analogiškų produktų, internetinė svetainė turi gražiai atrodyti, privalo būti labai atsakingai apgalvotas vartotojo valdymas. Vienas būdas pasiekti šį tikslą yra statinės ir dinaminės grafikos panaudojimas.

Iš esmės grafiką galima suskaidyti į du tipus: rastrinė grafika ir vektorinė grafika.

Rastrinė grafika aprašo vaizdą naudodama spalvotus taškus, išdėstytus tinkle. Pagrindiniai rastrinės grafikos ypatumai yra šie:

- vienas taškas nepriklauso nuo kitų taškų;
- pakankamai lengvai galima konvertuoti iš vieno duomenų formato į kitą (tokiam darbui atlikti yra gausybė standartinių, komercinių ir nekomercinių įrankių);
- galimybė pavaizduoti ir piešinius, ir realistiškus paveikslėlius.

Kitaip negu rastrinė grafika, vektorinė grafika vaizdams konstruoti naudoja matematinių objektų – taškų, linijų, kreivių aprašymus. Esminis vektorinės grafikos požymis yra tas, kad objektui pavaizduoti ji naudoja kompiuterinių komandų ir matematinių formulių junginį. Tai leidžia kompiuteriui apskaičiuoti ir nupiešti atitinkamose vietose realius to objekto taškus. Pagrindiniai vektorinės grafikos ypatumai yra šie:

- visas skiriamosios gebos pranašumas naudojamas bet kokiam išvedimo prietaisui. Tai leidžia keisti paveikslėlio matmenis neprarandant kokybės. Vektorinės komandos tiesiog nurodo išvedimo prietaisui, kad reikia nupiešti tam tikro dydžio objektą, naudojant tiek taškų, kiek įmanoma. Kitaip tariant, kuo daugiau taškų gali pavaizduoti išvedimo įrenginys, tuo gražesnis bus piešinys.
- elementai yra nedalomi ir nepriklausomi vienas nuo kito, t.y. egzistuoja galimybė dirbti su atskirais vektorinės grafikos objektais niekaip neveikiant kitų.

Iki šiol žiniatinklyje daugiausia buvo naudojama tik rastrinė grafika. Tačiau, tam tikrose situacijose, tam tikrose dalykinėse srityse vektorinė grafika yra žymiai pranašesnė ir patogesnė negu rastrinė. Pastaruoju metu į tai vis dažniau atkreipiamas dėmesys ir vektorinė grafika tampa vis labiau populiaresnė.

Vektorinė grafika žiniatinklyje atsirado paplitus Flash technologijai. Taip pat vektorinę grafiką galima pavaizduoti ir panaudojant taip vadinamus įskiepius (Internet Explorer), o, pavyzdžiui, Firefox (nuo 1.5 versijos), Opera (nuo 8 versijos), Safari ir kitos naršyklės palaiko vektorinę grafiką (vektorinės grafikos formatą SVG) net pagal nutylėjimą [cod09].

Visa tai reiškia, kad vektorinė grafika, nors ir yra pakankamai naujas reiškinys žiniatinklyje, darosi vis populiaresnė. O tai savo ruožtu yra įdomi alternatyva pritraukti potencialių klientų dėmesį ir plėtotis kitokiems informacijos vaizdavimo būdams. Be to, vektorinė grafika suteikia žymiai daugiau galimybių, negu rastrinė grafika. Ji įneša interaktyvumo, o tai yra didelis žingsnis į priekį.

Kitas labai svarbus vektorinės grafikos panaudojimo aspektas yra tas, jog be vaizdo, kuris yra sudarytas iš objektų, vektorinė grafika turi savyje ir semantinės informacijos. O tai savo ruožtu kartu su kitomis vektorinės grafikos ypatybėmis atveria daug naudingų panaudojimo galimybių, kaip, pavyzdžiui, paieška pagal nurodytus kriterijus, maksimalios raiškos vaizdo spausdinimas (kiek maksimaliai leidžia naudojama aparatūra), vaizdo didinimas ir mažinimas neprarandant kokybės ir t.t.

Darbo tyrimo objektas

Šio darbo tyrimo objektas yra vektorinė grafika žiniatinklyje, jos pagrindinių panaudojimo galimybių tyrimas.

Vektorinė grafika, kaip jau buvo paminėta, nors ir nėra taip plačiai taikoma žiniatinklyje kaip rastrinė, tačiau tam tikrose dalykinėse srityse, pavyzdžiui, rodant brėžinius, diagramas, natas, reklamas, žaidimus ir t.t., yra žymiai pranašesnė už rastrinę grafiką. Be to, vektorinė grafika su savimi atneša ne tik vaizdą, kaip rastrinė grafika, bet ir atributinę informaciją. Kaip vieną iš vektorinės grafikos panaudojimo būdų žiniatinklyje galima paminėti SVG formatą, kuris ir buvo pasirinktas šio magistrinio darbo tyrimo objektu. SVG formatas buvo pasirinktas dėl kelių priežasčių:

- tai yra atviras formatas;
- SVG yra vienas iš vektorinės grafikos standartų, kuris yra viešai prieinamas;

- SVG formatą palaiko pagal nutylėjimą daug populiarių naršyklių (Firefox, Opera, Safari), o su papildomais įskiepiais – Internet Explorer;
- SVG galima integruoti į HTML, pridėti papildomos semantinės informacijos;
- SVG formatas nėra taip gerai „praktiškai“ ištirtas kaip, pavyzdžiui, Flash technologijos.

Darbo tikslai ir uždaviniai

Pagrindinis darbo tikslas yra ištirti vektorinės grafikos panaudojimo galimybes žiniatinklyje ir aprašyti bei palyginti jos taikymo metodus. Konkrečiu atveju – ištirti SVG vektorinės grafikos formato panaudojimo galimybes ir metodus kuriant žiniatinklio aplikacijas. Šiam tikslui įgyvendinti buvo suformuluoti šie uždaviniai:

- Ištirti SVG statinius integravimo į žiniatinklio turinį aspektus:
  - kokiais būdais galima integruoti SVG į HTML (XHTML);
  - kokie egzistuoja būdai kartu panaudoti SVG ir CSS;
  - kaip galima integruoti kliento pusėje vykdomus scenarijus (JavaScript) į SVG.
- Ištirti SVG dinaminius integravimo į žiniatinklio turinį aspektus:
  - kaip yra suderinami SMIL ir SVG;
  - ką SVG standartas numato dinamiškumui realizuoti;
  - kaip pasireiškia vektorinės grafikos dinamika, apjungus SVG su JavaScript.
- Išnagrinėti naudotojo sąsajos grafinius elementus
  - išrinkti naudotojo sąsajos grafinius elementus;
  - ištirti/parodyti, kaip tuos elementus galima realizuoti su SVG;
  - panagrinėti realizavimo variantų privalumus ir trūkumus, pateikti rekomendacijas, kada kokį variantą geriau pasirinkti.
- Sukonstruoti veikiantį SVG technologijų panaudojimo pavyzdį:
  - darbo pavyzdys yra paprastas vektorinės grafikos redaktorius, turintis tokias galimybes: nupiešti objektą (tašką, liniją ir t.t.), pasirinkti spalvą, priskirti papildomą semantinę informaciją ir pan.;
  - parodyti, kaip galima panaudoti AJAX galimybes su SVG.

Atlikus darbą, tikimasi gauti šiuos rezultatus:

- SVG ir kitokio turinio, standarto arba technologijos panaudojimo galimybių sąrašas, t.y. kokiais atvejais ir kokius uždavinius sprendžiant prasminga panaudoti vieną arba kitą SVG taikymo variantą;
- SVG ir kitų technologijų arba standartų integravimo schemos ir/arba rekomendacijos;
- demonstracija, kaip, taikant išvardintas schemas ir rekomendacijas, galima integruoti SVG su kitais standartais ir kitomis technologijomis.

1 Literatūros apžvalga

1.1 Statiniai SVG integravimo į žiniatinklio turinį būdai

Norint pasinaudoti vektorinės grafikos galimybėmis žiniatinklyje, visų pirma, reikia integruoti vektorinį vaizdą į žiniatinklio dokumentą arba jų visumą. Yra keletas būdų arba sprendimų kaip tai padaryti. Kiekvienas iš jų atskiru atveju gali būti priimtinesnis už kitus.

1.1.1 Naršyklių palaikymas

Turbūt vienas iš paprasčiausių būdų yra tiesiog neintegruoti SVG turinio į HTML arba XHTML dokumentą. Šio būdo privalumas žiniatinklio aplikacijų kūrėjams yra aiškus – nereikia galvoti, ar vartotojo naršyklė palaiko SVG, ar ne. Galima tiesiog palikti nuorodą į SVG failą, o vartotojas jau pats pasirinks įrankį, kuriuo galės peržiūrėti turinį [Pe00].

Tačiau toks būdas turi nemažai trūkumų. Visų pirma, vartotojui reikia pačiam rūpintis SVG atvaizdavimo įrankiu, antra, vaizdas iškrenta iš bendro puslapio konteksto. Be to, dabar jau yra pakankamai daug naršyklių, daugiau ar mažiau palaikančių SVG formatą pagal nutylėjimą [cod09]. O naršyklėms, kurios nepalaiko SVG atvaizdavimo pagal nutylėjimą (pvz., Internet Explorer), galima įdiegti papildomą įskiepį, kuris tokią galimybę suteiktų [LV08].

Todėl šis būdas, nors ir yra pats lengviausias, dabar jau nėra priimtinas. Dauguma naršyklių ir įskiepių, nors ne visiškai, bet palaiko SVG atvaizdavimą. Interneto puslapių kūrėjai gali išbandyti įvairių naršyklių galimybes tinkamai atvaizduoti SVG turinį pagal paruoštus testus [W3C08]. Taip pat naršyklių svetainėse galima rasti naujausios informacijos apie SVG specifikacijos palaikymą (Firefox [MSS08], Opera 9 [SSO08]).

1.1.2 Integracija su HTML

Norint publikuoti visiems prieinamą informaciją, reikalinga speciali kalba, kurią bet koks kompiuteris galėtų suprasti ir pateikti vartotojui publikuotą informaciją žiniatinklyje. Tokia kalba naudojama žiniatinklyje yra HTML [HS08]. HTML kūrėjui pateikia tokias sąvokas ir galimybes kaip [IH08]:

- publikuoti žiniatinklyje dokumentus su antraštėmis, tekstu, lentelėmis, sąrašais, paveikslėliais ir t.t.;
- gauti žiniatinklio informaciją per nuorodas, mygtukų paspaudimus;

- kurti įvairiausias formas, kurių pagalba galima vykdyti transakcijas su nutolusiais servisais informacijos paieškai, prekių rezervavimui, užsakymui ir t.t.;
- įtraukti skaičiuokles, vaizdo bei garso klipus ir kitokio tipo turinį į žiniatinklio dokumentus.

1.1.2.1 Vardų erdvė (angl. namespace)

Vaizduojant vektorinę grafiką žiniatinklyje, visų pirma tenka nuspręsti, kaip pateikti paruoštą vaizdą SVG formatu naršyklės lange kartu su kitu turiniu. Vienas iš būdų – integruoti SVG failą į interneto puslapį tokiu būdu, kad naršyklė galėtų jį atvaizduoti [MH01].

SVG integravimas į puslapį negali būti atliktas tiktai nukopijavus į atitinkamą vietą SVG failo turinį, nes, atlikus tik šiuos veiksmus, naršyklė nesupras pateikto dokumento ir nežinos, kaip jį reikia pavaizduoti. Be to toks dokumentas nebus validus. Panaši problema gali kilti ne tik įterpiant skirtingų XML aplikacijų dokumentus į vieną bendrą dokumentą, kaip pvz., integruojant į XHTML matematines formules, užrašytas MathML pagalba, arba minėtus paveikslus, kaip SVG formatas ir t.t. Problema slypi tame, kad skirtingos XML aplikacijos, prisilaikančios savo specifikacijų, gali naudoti vienodus žymių vardus, ir todėl dokumento tikrintojas (validatorius) negali žinoti, ar bendras dokumentas yra suformuotas teisingai, ar ne. Pavyzdys galėtų būti toks: viename XML dokumente yra išsaugotas paveikslas SVG formatu ir matematinė formulė MathML formatu. Tiek SVG, tiek MathML sekcijose gali būti panaudota žymė *set*, kuri SVG reiškia abstraktų konteinerį, o MathML – matematinę aibę. Tokiu būdu, vienas vardas žymi visiškai skirtingus dalykus. Taigi, pasirinkus elementą *set*, reikia tiksliai žinoti, su kokiu objektu yra dirbama, nes kitaip dokumento validacija ar atvaizdavimas, ar indeksavimas ar kt. negali būti tinkamai atliktas, ir dokumentas negalės atlikti savo funkcijų. Tokiems nesusipratimams išvengti ir buvo sugalvotas vardų erdvės (angl. namespace) mechanizmas, kurio pagrindinės funkcijos yra:

- sugrupuoti visus susijusius elementus ir atributus iš bendro XML dokumento į grupes, kad būtų įmanoma tiksliai atpažinti, koks elementas kokia prasme naudojamas;
- leisti skirtingoms XML aplikacijoms naudoti tuos pačius elemento vardus, užtikrinant, kad neatsiras vardų konfliktų.

1.1.2.2 Vardų erdvės sudarymo mechanizmas

Namespace padeda išspręsti vienodų vardų skirtinguose kontekstuose panaudojimą, priskiriant elementus ir atributus prie URI. Bendrai, visi elementai iš vienos XML aplikacijos priskiriami prie vieno URI, ir visi elementai ir atributai iš skirtingų XML aplikacijų priskiriami prie skirtingu URI. Elementai, kurių vardai sutampa, tačiau jie yra priskirti prie skirtingu URI yra skirtingi, ir tik tie elementai, kurių ir vardai, ir URI sutampa, yra vienodi. Tokiu principu namespace mechanizmas padeda išspręsti vienodų vardų panaudojimo konfliktus.

1.1.2.2.1 Prefiksai

Kadangi URI dažnai naudoja tokius simbolius kaip „/“, „%“ ir „~“, kuriuos draudžiama naudoti sudarant XML vardus, įvedami trumpi prefiksai, pavyzdžiui, *xsl, fo*, *svg* ir t.t., kurie atstovauja URI ir gali būti naudojami sudarant elementų bei atributų vardus. Kiekvienas prefiksas asocijuojamas tik su vienu URI, todėl vardai, kuriems panaudotas vienodas prefiksas, yra vienoje vardų erdvėje, o vardai su skirtingais prefiksais yra skirtingose vardų erdvėse. Sintaksiškai elementas su jam nurodytu prefiksu atrodo taip:

```
prefix:element_name
```

Viskas, kas yra prieš dvitaškį, vadinama prefiksu (*prefix*), o viskas, kas eina po dvitaškio, vadinama lokaliąja dalimi (*element_name*). Prefiksas parodo, kokiai vardų erdvei priklauso elementas. Anksčiau minėtame pavyzdyje tam, kad būtų galima atskirti elementą *set* pagal kontekstą, elementui *set*, kuris reiškia SVG abstraktų konteinerį, galima priskirti prefiksą *svg*, o elementui *set*, kuris reiškia MathML matematinę aibę – prefiksą *mathml*. Tokiu būdu, elementai *svg:set* ir *mathml:set* jau priklausys skirtingoms vardų erdvėms ir bus skirtingi.

1.1.2.2.2 Prefiksų priskyrimas URI

Kiekvienas panaudotas prefiksas turi būti susietas su URI. Prefiksai priskiriami URI vardų erdvei pridedant atributą *xmlns:prefix* prie reikiamo elemento arba jo protėvio. Pavyzdžiui, elemento *parent* atributas *xmlns:svg* priskiria prefiksą *svg* prie URI *http://www.w3.org/2000/svg*:

```
<?xml version="1.0" standalone="yes"?>
<parent xmlns="http://example.org"
       xmlns:svg="http://www.w3.org/2000/svg">
```

```
   <!-- parent contents here -->
   <svg:svg width="4cm" height="8cm" version="1.1">
      <svg:ellipse cx="2cm" cy="4cm" rx="2cm" ry="1cm" />
   </svg:svg>
   <!-- ... -->
</parent>
```

Pav. 1 Prefikso priskyrimas prie URI

Prefikso priskyrimo galiojimo sritis susideda iš paties elemento ir jo vaikinių elementų. Prefiksas gali būti apibrėžtas pačiame viršutiniame elemente, kuris naudoja tą prefiksą arba bet kuriame iš jo vaikų. Tai gali būti dokumento šakninis elementas arba žemesnio lygio elementas. Kai dokumente naudojama daug vardų erdvių, prefikso aprašymas dokumento šakniniame elemente yra patogesnis, negu išbarstyti aprašymai po visą dokumentą. Jeigu būtina, vienas elementas gali turėti kelių skirtingų vardų erdvių aprašymus.

Vardų erdvės URI nebūtinai turi rodyti į egzistuojančius ir veikiančius puslapius. Dar daugiau, nereikalaujama, kad būtų naudojamas tik *http* protokolas. Tai galėtų būti, pavyzdžiui, *mailto* protokolas. URI vardų erdvės apibrėžime naudojamas tik kaip unikalus identifikatorius. Bendru atveju, tai nėra nuoroda į kokį nors puslapį, tačiau, pavyzdžiui, tuo adresu patalpinti vardų erdvės dokumentaciją yra visai nebloga idėja.

Tikrintojai paprastai sulygina skirtingas vardų erdves URI paraidžiui. Jeigu URI skiriasi bent vienu simboliu, jie sudaro skirtingas vardų erdves. Pavyzdžiui, *http://www.w3.org/1999/XSL/Transform*, *http://www.W3.ORG/1999/XSL/Transform*, *http://www.w3.org/1999/XSL/Transform/* ir *http://www.w3.org/1999/XSL/Transform/index.html* gali rodyti į vieną puslapį, tačiau sudaro 4 skirtingas vardų erdves.

Dažnai būna taip, kad visas dokumento turinys priklauso vienai XML aplikacijai. Pavyzdžiui, faile apibrėžtas tik vektorinės grafikos paveiksliukas. Šiuo atveju galima nurodyti, kad elementai be išreikštinai nenurodytų prefiksų priklauso tam tikrai vardų erdvei, t.y. galima apibrėžti vardų erdvę pagal nutylėjimą. Tai yra daroma priskiriant elementui *xmlns* atributą be prefikso (žr. 2 pav.). Tačiau taip galima priskirti vardų erdvę pagal nutylėjimą tik elementams, o ne jų atributams, nes pagal standartą vardų erdvė yra tuščia (arba, kitaip tariant, neegzistuoja) tiems atributams, kurie yra be prefikso [NX08]. Šiame pavyzdyje atributai *width*, *height*, *cx*, *cy*, *r* nėra priskirti jokiai vardų erdvei.

```
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>SVG embedded inline in XHTML</title>
  </head>
  <body>
    <h1>SVG embedded inline in XHTML</h1>

    <svg xmlns="http://www.w3.org/2000/svg" width="300" height="200">
```

```
      <circle cx="150" cy="100" r="50" />
    </svg>

  </body>
</html>
```

Pav. 2 SVG įterpimas į žiniatinklio puslapį

1.1.2.3 Kaip objektas

Be namespace mechanizmo pagalbos yra dar vienas būdas integruoti SVG į interneto puslapį. Vietoj tiesioginio SVG turinio įterpimo į HTML dokumentą, galima SVG vaizdą įtraukti per *object* žymę [Ga09], kuri yra standartinė HTML formato žymė [HS08]. Šio būdo privalumas yra tas, kad, jeigu vartotojo naršyklė nemoka atvaizduoti SVG vaizdo, tai galima kaip alternatyvą pavaizduoti paprastą rastrinį vaizdą:

```
<object data="image-svg.svg" type="image/svg+xml" height="48"
width="48">
  <img src="image-png.png" height="48" width="48" alt="this is a PNG"
/>
</object>
```

Pav. 3 SVG integravimas per object žymę

Object žymė nurodo naršyklei, kad reikia įtraukti į dokumentą SVG tipo objektą, kuris pasiekiamas nurodytu adresu. *Object* žymės turinys yra alternatyvus turinys, t.y. jis bus parodytas tuo atveju, jeigu naršyklė nežinos, kaip pavaizduoti patį objektą.

Tačiau šio būdo privalumas kartu yra ir jo trūkumas. Taip įtrauktas į interneto puslapį SVG praranda ryšį su kitais puslapio elementais, t.y. SVG vaizdas dokumento atžvilgiu yra juodoji dėžė. Atskirais atvejais, kai SVG turi bendrauti su kitais dokumento elementais, tai gali sukelti didelių problemų.

1.1.3 Integracija su CSS

Cascading Style Sheets (CSS) – tai paprastas mechanizmas pridėti stilių (pvz., nustatyti šriftą, spalvą, tarpus ir kt.) žiniatinklio dokumentams [CSS08]. SVG visiškai atitinka CSS2 rekomendacijas [SC08]. SVG naudoja CSS parametrus tam, kad apibrėžtų daugumą savo dokumento parametrų. Ypatingai SVG naudoja CSS parametrus apibrėžti tokius parametrus, kurie:

- yra išreikštinai naudojami vizualizavimui aprašyti ir todėl gali būti savarankiškai priskirti prie stilių parametrų. Tokių parametrų pavyzdžiai yra visi parametrai, kurie pasako, kaip objektai yra piešiami: *fill color*, *stroke color*, *linewidth*, *dash style* ir t.t.;
- nusako, kaip turi būti vaizduojamas tekstas, pavyzdžiui, *font-family*, *font-size* ir t.t.;
- reguliuoja būdus, kaip bus vaizduojami grafiniai elementai, pavyzdžiui, *clipping paths*, *masks*, *arrowheads*, *markers and filter effects* ir t.t.

SVG taip pat naudoja tokias CSS2 galimybes kaip:

- CSS2 sintaksės taisyklės kartu su leistinais duomenų tipais;
- pakopinių stilių deklaracijos su selektoriais;
- SVG palaiko ir išorinius stilius (nusakytus atskirame faile), ir vidinius (nusakytus per elemento atributą *style*);
- CSS2 taisyklės, nusakančios reikšmių priskyrimą savybėms, paveldėjimą, pakopinimą;
- *@font-face*, *@media*, *@import* ir *@charset* taisyklės.

Žemiau yra pateiktas pavyzdys kaip galima integruoti CSS į SVG [SSC08]:

```
<svg:svg width="12cm" height="12cm">
    <svg:g style="fill-opacity:0.7; stroke:black; stroke-
width:0.1cm;">
      <svg:circle cx="6cm" cy="2cm" r="100" style="fill:red;"
transform="translate(0,50)" />
      <svg:circle cx="6cm" cy="2cm" r="100" style="fill:blue;"
transform="translate(70,150)" />
      <svg:circle cx="6cm" cy="2cm" r="100" style="fill:green;"
transform="translate(-70,150)"/>
      <svg:rect x="0" y="0" width="12cm" height="12cm"
style="fill:none;stroke:black; stroke-width:1;"/>
    </svg:g>
  </svg:svg>
```

Pav. 4 SVG ir CSS integracijos pavyzdys

1.1.3.1 Išorinis CSS

Kaip jau buvo paminėta anksčiau, SVG palaiko integraciją su CSS, kai visi naudojami stiliai yra apibrėžti atskirame faile. Išorinis CSS susiejamas su SVG panaudojant standartinį susiejimo mechanizmą [SC08].

Pakopiniai stiliai (CSS) gali būti surišti su XML dokumentu panaudojus parengiamosios doroklės (angl. *preprocessor*) instrukciją, kurios pavadinimas yra *xml-stylesheet* [ASS08]. Šita

instrukcija realizuoja HTML elemento <*LINK REL="stylesheet"*> veikimą (t.y. žemiau pateiktos eilutės yra ekvivalenčios):

```
<LINK href="mystyle.css" rel="style sheet" type="text/css">
<?xml-stylesheet href="mystyle.css" type="text/css"?>
```

Pav. 5 Išorinio CSS susiejimas su SVG

1.1.3.2 Vidinis CSS

Kitas būdas panaudoti CSS stilius SVG turinyje yra vidinis CSS. Vidinis CSS gali būti apibrėžtas per elementą <*style*> ir per atributą *stile.*

Elementas <*style*> leidžia autoriui įterpti į SVG dokumentą CSS taisykles. Elementai <*style*> gali būti apibrėžti tik kaip SVG elemento <*defs*> vaikai. Kai kurios pakopinių stilių realizacijos leidžia panaudoti didesnę aibę CSS taisyklių <style> elemente negu *style* atribute, kuris yra prieinamas konteinerio elementams ir grafikos elementams. Pavyzdžiui, su CSS2 taisyklės gali būti nusakytos elemente <*style*>, bet ne atribute *style*. Žemiau paretiktas pavyzdys rodo, kaip galima nusakyti teksto stilių:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG August 1999//EN"
  "http://www.w3.org/Graphics/SVG/SVG-19990812.dtd">
<svg width="4in" height="3in">
  <defs>
    <style><![CDATA[
      .TitleText { font-size: 16; font-family: Helvetica } ]]>
    </style>
  </defs>
  <text class="TitleText">Here is my title</text>
</svg>
```

Pav. 6 Pakopiniai stiliai, nusakyti <style> elemente

Panašiai kaip ir HTML, elemento stilių galima nurodyti panaudojus elemento atributą *style*. Atributas *style* leidžia tiesiai įterpti stilių aprašą į konkretų elementą (žr. 4 pav.). Jeigu tas pats stilius naudojamas daug kartų skirtinguose arba vienodo tipo elementuose, patartina tą stilių apibrėžti elemente <*style*>.

Dar vienas būdas nusakyti elemento stilių yra *class* atributo panaudojimas [Sty08]. Atributas *class* priskiria elementui vieną arba daugiau klasių vardų. Galima sakyti, kad elementas priklauso šioms klasėms. Klasės vardas gali būti panaudotas tarp kelių elementų. Atributas *class* vaidina keletą rolių:

- jis yra pakopinių stilių selektorius (kai autorius nori priskirti kokį nors stilių elementų grupei);
- Bendram naršyklių naudojimui.

1.1.3.3 CSS apimtis

Išskiriami tokie CSS matomumo variantai [SC08]:

1. savarankiškas SVG piešinys;
2. savarankiškas SVG piešinys, įterptas į HTML dokumentą kaip <*img*> arba <*object*> elementas;
3. savarankiškas SVG piešinys, įterptas į XML dokumentą kaip paprastas tekstas.

Pirmas variantas. Egzistuoja vienas nagrinėjimo medis, ir visi elementai yra SVG vardų srityje. CSS stilius, apibrėžtas bet kurioje SVG piešinio vietoje (panaudojus <*style*> elementą arba *style* atributą, arba surišus su išoriniu CSS), pritaikomas visam SVG piešiniui.

Antras variantas. Egzistuoja du visiškai nepriklausomi nagrinėjimo medžiai: vienas – HTML dokumentui, kitas – SVG grafikai. CSS stilius, apibrėžtas bet kurioje vietoje HTML dokumente (panaudojus <*style*> elementą arba *style* atributą, arba surišus su išoriniu CSS), pritaikomas visame HTML dokumente. Kadangi paveldėjimas galioja tik nagrinėjant medį gilyn, tai stilius neliečia SVG grafikos. CSS stilius, apibrėžtas bet kurioje SVG dokumento vietoje, pritaikomas tik SVG piešiniui. SVG stilius niekaip neįtakoja HTML dokumento. Norint, kad stilius būtų toks pat tiek HTML dokumente, tiek SVG piešinyje, reikia HTML ir SVG dokumentus surišti su tuo pačių stiliumi.

Trečias variantas. Egzistuoja vienas nagrinėjimo medis, kuris naudoja skirtingas vardų sritis. Vienas arba daugiau pomedžių yra SVG vardų srityje. CSS stilius, apibrėžtas bet kurioje vietoje XML dokumente, pritaikomas visam dokumentui, tuo pačiu ir SVG elementams. Norint turėti skirtingus CSS stilius SVG dokumento daliai, reikia panaudoti <*style*> elementą <*svg*> elemento viduje. Arba kitas variantas – priskirti <*svg*> elemente reikalingam vidiniam elementui ID, ir naudoti CSS kontekstinius selektorius.

1.1.4 Integracija su JavaScript

Kliento pusėje vykdomi scenarijai – tai tokios programos, kurios gali būti susietos su konkrečiu HTML puslapiu ar netgi būti to puslapio turinio dalimi [Scr08]. Programa arba

scenarijus yra vykdomas kliento kompiuteryje, kai dokumentas yra pakrautas, arba kitu nustatytu momentu, pavyzdžiui, kai yra aktyvuojama nuoroda. Scenarijai leidžia dokumento autoriui išplėsti kuriamo dokumento galimybes aktyvumo bei interaktyvumo prasme:

- scenarijai gali būti vykdomi, kai dokumentas pasikraus dinamiškai keisti pakrauto dokumento turinį;
- scenarijai gali praversti tikrinant į formų laukus įvestas reikšmes. Kūrėjai gali dinamiškai užpildyti dalį įvedimo laukų priklausomai nuo jau įvestų duomenų. Taip pat scenarijai leidžia patikrinti, ar vartotojas įvedė teisingus duomenis į įvedimo laukus ir, jeigu reikia, perspėti vartotoją apie pastebėtas klaidas arba netgi automatiniu būdu jas ištaisyti.
- scenarijai gali būti kviečiami atsitikus tam tikriems įvykiams, tokiems kaip užkrovimas, fokuso gavimas, pelės judėjimas ir pan;
- scenarijai gali būti surišti su valdymo elementais (pvz., su mygtukais), kad generuotų grafinę vartotojo sąsają.

Galima išskirti du scenarijų tipus, kurie galimi žiniatinklio dokumentuose:

1. scenarijai, kurie vykdomi naršyklėje kai dokumentas yra pakrautas;
2. scenarijai, kurie yra vykdomi kiekvieną kartą, kai atsitinka tam tikras įvykis (tokie scenarijai gali būti priskirti dokumento elementams per atitinkamus atributus).

1.1.4.1 Integracijos būdai

Vienas iš paprasčiausių būdų integruoti scenarijų į SVG turinį yra <*script*> elemento panaudojimas. Jo veikimo principas yra analogiškas HTML <*script*> elemento veikimo principui. Bet kokios funkcijos, apibrėžtos <*script*> elemente, yra globalios tam konkrečiam dokumentui.

Žemiau yra pateiktas pavyzdys, kuris parodo kaip galima integruoti ECMAScript (JavaScript) į SVG turinį:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
  "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="6cm" height="5cm" viewBox="0 0 600 500"
     xmlns="http://www.w3.org/2000/svg" version="1.1">
  <desc>Example script01 - invoke an ECMAScript function from an
onclick event
  </desc>
  <!-- ECMAScript to change the radius with each click -->
  <script type="text/ecmascript"> <![CDATA[
```

```
    function circle_click(evt) {
      var circle = evt.target;
      var currentRadius = circle.getAttribute("r");
      if (currentRadius == 100)
        circle.setAttribute("r", currentRadius*2);
      else
        circle.setAttribute("r", currentRadius*0.5);
    }
  ]]> </script>
  <!-- Outline the drawing area with a blue line -->
  <rect x="1" y="1" width="598" height="498" fill="none"
stroke="blue"/>
  <!-- Act on each click event -->
  <circle onclick="circle_click(evt)" cx="300" cy="225" r="100"
          fill="red"/>
  <text x="300" y="480"
        font-family="Verdana" font-size="35" text-anchor="middle">
    Click on circle to change its size
  </text>
</svg>
```

Pav. 7 JavaScript integracija į SVG

Pavyzdyje rodomas raudonas skritulys, kurį galima paspausti pele. Kai įvyksta pelės spragtelėjimas ant skritulio, iškviečiama funkcija *circle_click*. Šita funkcija daro vienintelį veiksmą: pakeičia skritulio dydį (jeigu skritulys buvo mažas, padaro jį dideliu, jeigu didelis - mažu).

1.2 Dinaminiai SVG integracijos aspektai

Vienas iš patraukliausių SVG panaudojimo žiniatinklyje (ir ne tik) aspektų yra galimybė sukurti dinamiškus SVG arba, kitaip tariant, galimybė pridėti animaciją [Ani08a]. Dauguma SVG paveikslo dalių gali būti animuotos – pozicija ekrane, plotis, aukštis, spalvos, permatomumas, tarpinės spalvos gradientė, SVG filtrų atributai.

SVG vaizdai gali būti animuoti dviem metodais: panaudojus deklaratyvius SVG elementus (kurių dauguma yra pasiskolinti iš SMIL standarto) arba taikyti ECMAScipt (JavaScript) arba kitokią scenarijaus kalbą Document Object Model (DOM) manipuliavimui. Bendrai tiek vienas, tiek kitas metodas gali būti panaudotas tame pačiame piešinyje.

1.2.1 SMIL

Teigiama, kad SMIL yra XML pagrindo standartas, skirtas interaktyvioms prezentacijoms su įvairialypiu turiniu aprašyti [Ei08]. SMIL dokumentas pasako, kas turi būti interaktyviojoje prezentacijoje, t.y. deklaratyviai aprašo įvairialypio turinio apdorojimą. Būdas,

kaip konkrečiai animacija yra vykdoma pagal deklaratyvų aprašą, turi būti realizuotas specialiuose grotuvuose pagal W3C pateiktas rekomendacijas. SMIL 2.0 yra suskaidytas į 10 skirtingų modulių. Kiekvienas modulis aprašo savo XML žymes ir atributus. Pavyzdžiui, laiko modulis padengia visas žymes, susijusias su laiku, sinchronizavimu, bendradarbiavimu; turinio modulis aprašo visas žymes, susijusias su personalizacija ir t.t. Moduliai gali būti kombinuojami į profilius. SMIL standartas pateikia du standartinius profilius: pagrindinis profilis (praktiškai visos žymės ir visi elementai, kurie yra prieinami SMIL 1.0 standarte) ir kalbos profilis (visos SMIL 2.0 standarto galimybės, išskyrus laiko modulį). Taip pat, esant reikalui, gali būti sukurti papildomi profiliai (ir realizuoti grotuvuose). Gerai žinomi pavyzdžiai yra XHTML + SMIL profilis, realizuotas Microsoft Internet Explorer, ir 3GPP SML profilis, kuris naudojamas mobiliuose telefonuose (angl. *Multimedia Messaging System* arba MMS).

Patys reikšmingiausi SMIL moduliai yra struktūros modulis, įvairialypių objektų modulis, išdėstimo modulis ir laiko modulis. Išvardinti moduliai reikalingi kiekvienoje prezentacijoje. Struktūros modulis apibrėžia SMIL dokumento struktūrą, nusakydamas dokumento tipą, vardų erdvę, žymes dokumento aprašui ir kūnui (panašiai kaip XHTML žymės *head* ir *body*). Įvairialypių objektų modulis aprašo pagrindinius tipus: *audio*, *video*, *animation*, *img* (bet koks paveikslo formatas), *text* (XHTML arba paprastas tekstas), *textstream* (Real tekstas kartu su laiko formatu) ir *ref* (kiti objektai, nusakyti per MIME tipo atributą). Išdėstymo modulio elementai naudojami objektų daliniam komponavimui apibrėžti. Iš esmės, perklojimo regionai su aukščiu, pločiu ir z-indeksu (gyliu) gali būti nusakyti ir užkrauti įvairialypio tipo objektais. Būdas, kaip objektas (pvz., paveikslas), vaizduojamas jam priklausiančioje srityje (*region*), ir yra aprašomas specialiu atributu, nusakančiu elgseną (pvz., išlaikyti proporcijas, sumažinti pagal dydį ir t.t.).

Pagrindinis SMIL privalumas slypi laiko modulyje. Modulis yra atsakingas už bendrą laiko valdymą, o taip pat įvairialypių objektų sinchronizavimą ir bendradarbiavimą. SMIL palaiko mišinį iš dviejų fundamentalių laiko sampratų: laiko juostos ir laiko įvykių. SMIL kiekvienas objektas ir laiko konteineris turi tiesinę laiko juostą, kurią gali būti pertrauktą laiko įvykiu kai kuriam laikui. Paprastiems sinchronizavimo veiksmams SMIL siūlo 3 laiko konteinerius: *seq* (nuosekli įvairialypių objektų arba kitokių laiko konteinerių tvarka), *par* (lygiagretus vykdymas) ir *excl* (lygiagretus vykdymas, kur tik vienas objektas arba laiko konteineris gali būti visada aktyvus). Taip pat galima nusakyti sinchronizacijos šaltinius, užtrukimus ir serviso kokybės parametrus. Bendradarbiavimui arba sąveikai SMIL naudoja DOM. Kiekvienas SMIL grotuvas, kuris realizuoja bent pagrindinį profilį, turi turėti galimybę gaudyti DOM įvykius (pvz., *click*, *mouseover*). Įvykiai gali būti panaudojami tam, kad paleistų, pertrauktų arba sustabdytų įvairialypių objektų ir laiko konteinerių vykdymą (prezentaciją).

Be anksčiau aprašytų galimybių, SMIL 2.0 siūlo modulius turinio valdymui (personalizacija ir objektų pradinis nustatymas), metaduomenų apibrėžimui (kaip paprastų porų – raktas/reikšmė arba kaip Resource Description Framework aprašus), susiejimui (panašiai, kaip XHTML), animacijai ir perėjimui tarp *media* srautų. Paprastai animacijos žymės ir atributai gali būti naudojami bet kokiai objekto arba srities savybei (tarp jų spalvai, pozicijai, dydžiui) animuoti. SMIL naudoja *key-frame* (raktinių kadrų) animaciją. Kadrai tarp dviejų raktinių kadrų vaizduojami tiesiškai interpoliuojant raktinius kadrus arba interpoliuojant raktinius kadrus pagal užduotą kelią (algoritmą). Pagaliau, perėjimo modulis nusako objektų pradingimą (*fade*) ir SMPTE (angl. *Society of Motion Picture and Television Engineers*) išnykimą. Kadangi kiekvienas objektas turi laikinus išmatavimus (tarp jų tekstas ir paveikslai), perėjimai gali būti pritaikyti bet kokiam objekto tipui (žinoma, garso tipui gali būti pritaikytas tik slopinimas).

Vis dėlto kol kas SMIL turi vieną praktinį trūkumą – jo nepalaiko pagal nutylėjimą dauguma populiarių ir plačiai naudojamų naršyklių. Nors ir yra specialiai sukurti SMIL grotuvai (taip pat ir kaip įskiepiai), tačiau jie vis tiek nerealizuoja visų SMIL galimybių. Dauguma iš jų nepalaiko kai kurių profilių arba turi klaidų (laiko modulyje). Daugelyje egzistuojančių SMIL grotuvų ypač sunkiai realizuojama DOM sąveika.

Žemiau yra pateiktas paprastas SMIL animacijos pavyzdys, rodantis kaip galima sukurti prezentaciją [LSS08]:

```
<smil>
  <head>
   <layout>
     <root-layout height="140" width="210" background-color="#ffffff"
title="Elements"/>
     <region id="text" width="210" height="140" top="0" left="0" z-
index="2"/>
     <region id="textbox2" width="210" height="140" top="0" left="0"
z-index="3" />
   </layout>
  </head>
 <body>
  <par>
   <text src="media/hi.txt" region="text" begin="1.00s" dur="3.00s" />
   <audio src="media/drum4.auz" begin="4.00s" dur="7.00s" system-
bitrate="14000" />
    <seq>
     < img src="media/h.gif" region="textbox2" begin="4.00s"
dur="0.75s" />
     < img src="media/e.gif" region="textbox2" dur="0.24s" />
     < img src="media/l.gif" region="textbox2" dur="0.86s" />
     < img src="media/lb.gif" region="textbox2" dur="0.74s" />
     < img src="media/o.gif" region="textbox2" dur="0.26s" />
     < img src="media/w.gif" region="textbox2" dur="0.24s" />
     < img src="media/o.gif" region="textbox2" dur="0.99s" />
     < img src="media/r.gif" region="textbox2" dur="0.73s" />
     < img src="media/l.gif" region="textbox2" dur="0.26s" />
     < img src="media/d.gif" region="textbox2" dur="0.76s" />
     < img src="media/hello.gif" region="textbox2" dur="0.76s" />
```

```
     < img src="media/world.gif" region="textbox2" dur="0.24s" />
     < text src="media/theend.txt" begin="2.00s" region="textbox2" />
    </seq>
   </par>
  </body>
 </smil>
```

Pav. 8 SMIL prezentacijos pavyzdys

Šiame pavyzdyje kartu vaizduojami tekstas, garso takelis ir vaizdai (nusakoma per žymę *par* – lygiagretusis vykdymas). Savo ruožtu, paveikslai vaizduojami vienas paskui kitą po tam tikro laiko intervalo – tai aprašyta panaudojus žymę *seg* ir atributą *dur* prie kiekvieno paveikslo. Objektų išdėstymas aprašomas dokumento antraštėje su žyme *layout*.

1.2.2 SVG dinaminiai elementai

Animacija – tai paveikslo charakteristikų kitimas laikui bėgant. SVG animacija aprašoma deklaratyviai. SVG animacijos metu SVG atvaizdavimo variklis naudoja dvi animuojamos reikšmės kopijas. Viena yra originali reikšmė (kuri naudojama DOM), kita reikšmė yra darbinė, t.y. tokia reikšmė, kuri kinta animacijos metu ir atvaizduoja animuojamos charakteristikos momentinę reikšmę [Ani08a].

Animacijai nusakyti, visų pirma, reikia žinoti, ką reikia animuoti, kada animacija turi prasidėti, kokia yra pradinė ir galutinė animuojamos charakteristikos reikšmė. Norint animuoti pasirinkto elemento atributą, jis turi būti nurodytas kaip atributo *atributeName* reikšmė. Animacijos pradžia yra nurodoma atribute *begin*, o animacijos trukmė gali būti nurodyta atribute *dur*. Animuojamos charakteristikos pradinei ir galutinei reikšmei apibrėžti atitinkamai naudojami atributai *from* ir *to*, arba kitas būdas – nurodyti pradinę reikšmę (atributas *from*) ir žingsnį (atributas *by*). Jeigu reikia „perbėgti“ per konkrečias reikšmes, jas galima nurodyti atribute *values*. Visi aukščiau aprašyti atributai yra specialaus elemento <*animate*> atributai, kuris pasako, kokius parametrus reikia animuoti ir yra aprašomas kaip animuojamo elemento (konkrečiau vieno iš animuojamo elemento atributo) vaikas. Žemiau yra pateiktas pavyzdys, rodantis kaip galima animuoti stačiakampį (keičiamas stačiakampio aukštis ir plotis):

```
<?xml version='1.0'?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.0//EN"
"http://www.w3.org/TR/2001/PR-SVG-20010719/DTD/svg10.dtd">
<svg width="300" height="250">
  <rect x="100" y="100" width="10px" height="100px" style="stroke:red;
fill:rgb(0,0,0)">
    <animate attributeName="width"  from="10px" to="100px" begin="0s"
dur="10s" repeatCount="1" fill="freeze"/>
```

```
    <animate attributeName="height" from="100px" to="10px" begin="0s"
dur="10s" repeatCount="1" fill="freeze"/>
  </rect>
</svg>
```

Pav. 9 Stačiakampio animacijos pavyzdys

Kiti pavyzdyje neaptarti animacijos atributai: *repeatCount* – nurodo kiek kartų kartoti animaciją, *fill=“freeze“* – pasako, kad animacijai pasibaigus reikia palikti stačiakampio aukščio ir pločio atributus galutinėse reikšmėse (čia nereikia maišyti animacijos atributo *fill*, kuris pasako, kokia turi būti objekto būsena animacijai pasibaigus, su stačiakampio atributu *fill*, kuris pasako, kokios spalvos turi būti stačiakampis).

Be elemento <*animate*>, kuris skirtas animuoti vieną pasirinktą atributą, SVG standartas numato ir kitus animacijos elementus, tokius kaip <*set*>, <*animateMotion*>, <*animateColor*>, <*animateTransform*>, kurie skirti sudėtingesnei animacijai realizuoti [Ani08b].

1.2.3 JavaScript

Kaip jau buvo minėta aukščiau, į SVG galima integruoti scenarijų (vykdomą kliento pusėje), kurio pagalba galima realizuoti ne tik atsaką į vartotojo siunčiamus pranešimus, reikalingą logiką (pavyzdžiui, elementų reikšmių patikrinimas, įvairūs skaičiavimai ir pan.), bet ir animaciją. Kadangi su JavaScript galima manipuliuoti DOM, o SVG yra XML dokumentas, tai JavaScript duoda SVG turinio kūrėjui puikią galimybę manipuliuoti SVG elementais pagal nustatytą algoritmą.

Skirtingai negu anksčiau aprašytuose būduose (SMIL animacija ir SVG animacijos galimybės), JavaScript animacija nusakoma procedūriškai, kai pastarieji metodai pateikia deklaratyvią animaciją. Šiuo požiūriu, tai gali būti JavaScript animacijos realizacijos privalumas. Be to, šis animacijos būdas yra daugiausiai palaikomas praktiškai, t.y. naršyklės (su arba be papildomų įskiepių), SVG vaizdavimo programos ir t.t. greičiau palaiko JavaScript negu deklaratyvios animacijos aprašą.

Žemiau yra pateiktas tokio tipo animacijos pavyzdys [Ani08b]:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20000802//EN"
  "http://www.w3.org/TR/2000/CR-SVG-20000802/DTD/svg-20000802.dtd">
<svg width="12cm" height="2cm" viewBox="0 0 1200 200"
     onload="StartAnimation(evt)" >

  <script type="text/ecmascript"><![CDATA[
    var timevalue = 0;
    var timer_increment = 50;
```

```
    var max_time = 5000;
    var text_element;
    function StartAnimation(evt) {
      text_element =
evt.target.ownerDocument.getElementById("TextElement");
      ShowAndGrowElement();
    }
    function ShowAndGrowElement() {
      timevalue = timevalue + timer_increment;
      if (timevalue > max_time)
        return;

      // Scale the text string gradually until it is 20 times larger
      scalefactor = (timevalue * 20.) / max_time;
      text_element.setAttribute("transform", "scale(" + scalefactor +
")");
      // Make the string more opaque
      opacityfactor = timevalue / max_time;
      text_element.setAttribute("style", "opacity:" + opacityfactor);

      // Call ShowAndGrowElement again <timer_increment> milliseconds
later.
      setTimeout("ShowAndGrowElement()", timer_increment)
    }
    window.ShowAndGrowElement = ShowAndGrowElement
  ]]></script>

  <g transform="translate(50,150)" style="fill:red; font-size:7">
    <text id="TextElement">SVG</text>
  </g>
</svg>
```

Pav. 10 Animacijos realizacija panaudojus JavaScript

Aukščiau pateiktas SVG failas turi tik vieną grafinį elementą – tekstą „SVG“. Animacijos trukmė yra 5 sek. Animacijos pradžioje tekstas „SVG“ yra mažas ir permatomas, o laikui bėgant jis darosi didesnis ir nepermatomas (išskaičiavus teksto dydžio ir permatomumo laipsnio reikšmes, jos yra nustatomos teksto elementui).

1.3 SVG ir Flash

Dabartiniu metu Flash technologija yra labai paplitusi internete. Flash pagalba paprastai piešiama vektorinė grafika, daromi prezentaciniai filmukai, reklama ir t.t. Tačiau pažiūrėjus iš arčiau, visa tai pajėgus padaryti ir SVG, be to kai kuriais atvejais su juo galima padaryti tai, ko negalima padaryti su Flash, pavyzdžiui, daryti paiešką pagal tekstą (tekstas SVG formate yra tiesiog tekstas, palyginus su Flash, kur tekstas yra arba paveikslėlis, arba kitaip užkoduotas dvejetainių pavidalu) [Ka09], [De09]:

Savybė	SWF (Flash)	SVG
Atviras standartas, specifikacija	Ne	Taip
Duomenų tipas	Binarinis	Tekstinis
MIME tipas	application/x-shockwave-flash	image/svg+xml
Interaktyvumas	Yra	Yra
Susiglaudimas	Yra, zlib	Yra, gzip
Failų dydis	Apytiksliai vienodas	
Naršyklių palaikymas	Dauguma naršyklių (tarp kurių Internet Explorer) turi integruota Flash Player	Ne visos. Internet Explorer reikalauja papildomo įskiepio
Indeksavimas paieškos sistemomis	Yra (su pagalbiniais metodais)	Yra. Aprašymą, raktinius žodžius galima integruoti į patį dokumentą
Grafinių filtrų palaikymas	Yra	Yra
Kodo bibliotekų palaikymas (PHP, XSLT, JSP)	Yra	Yra
CSS, XSL, XPath palaikymas	Įgyvendinama sudėtingai	Yra
Integracija su aplinka, tinklu	Gera	Silpna
Daugiaterpės informacijos palaikymas (audio, video)	Geras	Silpnas

Iškyla natūralus klausimas – jeigu SVG nėra prastesnis už Flash, ir atvirkščiai su SVG kartais galima padaryti daugiau negu su Flash, tai kodėl visgi Flash yra populiaresnis už SVG, kodėl Flash naudojamas praktiškai visur, o tuo tarpu SVG tik kur ne kur? Turbūt napavyks rasti vienareikšmio atsakymo į šį klausimą, tačiau egzistuoja keletas aplinkybių, kurios gali paaiškinti tokią situaciją.

Visų pirma, Flash gimimo data laikoma 1996 metai [AF09], o SVG pradėtas kurti 1999 metais, standartu tapo tik 2001 metais [SVG09b]. Taigi Flash turėjo laiko tobulėti. Antra, Flash technologija – tai komercinis produktas, kuris buvo aktyviai reklamuojamas ir diegiamas „Abobe“ kompanijos [Ado09]. Trečia, tais metais, kai atsirado SVG ir Flash, populiariausia naudotojų naršyklė buvo (ir yra iki šiol) Internet Explorer [BS09], kuri nepalaiko SVG, tačiau moka rodyti Flash. Turbūt yra ir kitų priežasčių, tačiau šios yra pagrindinės, paaiškinančios, kodėl Flash yra populiaresnis už SVG.

1.4 SVG taikymai

Kaip jau buvo pasakyta aukščiau, SVG kol kas yra mažai kur naudojamas, palyginus su Flash. Tačiau tai nereiškia, kad SVG yra tik teorinis dalykas. SVG yra naudojamas žiniatinklyje jau dabar, tik jo paplitimas, palyginus su Flash, yra labai ribotas. SVG naudojamas tokiose srityse kaip:

- GIS [GS09], [YHM09]. GIS projektuose SVG labai tinka pateikti žemėlapį, jo sluoksnius. Dažnai būna taip, kad net naudotojo sąsajos elementai taip pat būna sukurti panaudojus SVG, kadangi GIS numato daug specifinių sąsajos elementų, kurių nėra HTML standarte (liniuotė, mastelio keitimas ir pan.);
- paveikslų galerijos. SVG pagalba paveikslai dažnai pateikiami su įvairiais efektais (rėmai, paveikslų parodymai ir pan.);
- e-mokymai [PIE09]; paprastai SVG naudojamas įvairiems (interaktyviems) brėžiniams pateikti, kur pagrindinis SVG privalumas yra jo galimybė keisti mastelį neprarandant vaizdo kokybės, galimybė panaudoti duomenis skirtingose aplikacijose, įtraukti metaduomenis;
- žaidimai [SVG09c]; su SVG ir JavaScript galima sukurti ir paprastą žaidimą;
- kartu su HTML, papildant pastarąjį įvairiais efektais [OCK09]; kadangi SVG galima giliau integruoti į HTML negu, pavyzdžiui, Flash, ir SVG moka transformuoti vaizdą (pritaikyti įvairius efektus), šis būdas naudojamas kuriant nestandartinius svetainių apipavidalinimo variantus;
- mobiliuosiuose įrenginiuose [Em09]; dauguma šiuolaikinių mobiliųjų telefonų palaiko SVG Tiny standartą [MSP09] ir nepalaiko Flash, taigi SVG yra vienintelis būdas pateikti vektorinę grafiką naudotojui.

Taigi, jau dabar yra keletas sričių, kur SVG yra realiai naudojamas.

1.5 Apibendrinimas

Kaip parodė nagrinėjamos temos literatūros apžvalga, vektorinė grafika, o jeigu konkrečiau, nagrinėtas vektorinės grafikos formatas SVG, nėra tik teorinis formatas. Jis taikomas konkrečiuose žiniatinklio projektuose, naudojamas kaip geras pagalbininkas pateikti vektorinį paveikslą. Nors kol kas SVG formato pilnai nepalaikomas nei viena naršyklė, nei jam skirti specialūs grotuvai (kaip atskiros programos ir kaip įskiepiai naršyklėms), tačiau jau ir dabar galima naudotis jo galimybėmis, papildant žiniatinklio turinį.

Pagal savo pobūdį SVG galima suskaidyti į statinį ir dinaminį SVG. Vienu ar kitu atveju egzistuoja įvairūs būdai integruoti SVG į kitą turinį arba, kitaip tariant, panaudoti SVG su kitu turiniu.

Statinį SVG galima integruoti į HTML pasinaudojus vardų erdvės mechanizmu, arba integruoti kaip atskirą objektą. Pastarasis būdas yra geras tuo, kad jo pagalba galima padaryti universalų kodą, kuris tinka tiek toms naršyklėms, kurios palaiko SVG, tiek jo nepalaikančioms. Tačiau ir pirmas būdas turi savo privalumų, pavyzdžiui, tokiu atveju dokumentas su SVG turiniu yra vieningas, jame galimos įvairios manipuliacijos (tiek su paties dokumento elementais, tiek su SVG elementais) ir t.t. Taip pat SVG yra puikiai suderinamas su CSS. Kadangi CSS yra skirtas struktūriniams dokumentams, tai jis gali būti pritaikytas ir SVG turiniui. Iš praktinės pusės ši integracija yra labai panaši į CSS integraciją į HTML. Integracijai su kliento pusės scenarijais (pvz., JavaScript) SVG formatas numato atskirą žymę. Šiuo požiūriu neitin komplikuotiems atvejams scenarijų integracija į SVG nėra sudėtingas uždavinys ir vėlgi panašus į integraciją JavaScript su HTML.

Panaudojus SVG piešinyje dinaminius aspektus, galima žymiai praplėsti žiniatinklio turinio interaktyvumą ir vaizdingumą. Tačiau animacijos galimybės šiuo metu nėra taip plačiai realizuotos praktikoje kaip statinio SVG galimybės, t.y. egzistuoja labai mažai įrankių, kurie galėtų pasinaudoti SVG standarte apibrėžtomis animacijos galimybėmis, o jeigu ir palaiko, tai ko gero nepilnai. Vis dėlto ateityje šis trūkumas turėtų pamažu mažėti ir tada bus galima pasinaudoti ko gero vaizdingiausiomis SVG standarto galimybėmis.

Visgi šiek tiek dinamiškumo galima pasiekti ir su esamais įrankiais, pavyzdžiui, panaudojus JavaScript integraciją į SVG. Su JavaScript galima procedūriškai aprašyti SVG interaktyvumą ir animaciją, kaip tai galima padaryti, pavyzdžiui, su HTML.

Pats SVG standartas numato deklaratyvią animaciją, tačiau, kaip jau buvo pasakyta, tokios animacijos praktinė realizacija dabartiniu metu yra gana ribota. Egzistuoja dar vienas būdas deklaratyviai aprašyti SVG animaciją: galima tiesiog integruoti SVG į SMIL, kuris kaip tik ir skirtas interaktyvioms prezentacijoms su įvairialypiu turiniu aprašyti.

2 Vektorinė grafika ir jos panaudojimas

Norint tyrinėti vektorinę grafiką ir galimybes panaudoti ją savo tikslams, parankiausia tai padaryti konkretaus pavyzdžio pagalba. Norint pasiūlyti ir palyginti kuo daugiau vektorinės grafikos technikų panaudojimo galimybių, pavyzdys turėtų realizuoti:

- statinę vektorinę grafiką (vaizduojami grafiniai objektai nesikeičia laikui bėgant, yra statiški);
- dinaminę vektorinę grafiką (vaizduojami grafiniai objektai keičiasi laikui bėgant pagal užduotą algoritmą);
- sąsają su naudotoju (vaizduojami grafiniai objektai keičiasi laike priklausomai nuo naudotojo veiksmų).

Vienas tokių pavyzdžių galėtų būti vektorinės grafikos redaktorius, kuris ir buvo pasirinktas kaip vektorinės grafikos taikymo metodų stendas.

2.1 Naudotojo sąsajos grafiniai elementai

Kiekvienai programai (tarp jų ir grafiniam redaktoriui) pirmiausia reikalingi standartiniai naudotojo sąsajos arba valdymo elementai. Kadangi bus kuriama internetinė aplikacija, reikia parinkti internetinei sąsajai būdingus naudotojo valdymo elementus. Iš kitos pusės, grafinis redaktorius yra daugiau paprasta programa (arba programų sistema) negu žiniatinklio puslapis. Todėl verta panagrinėti, kokių valdymo elementų gali prireikti redaktoriui, kurie paprastai nenaudojami kuriant internetines svetaines ir kurie yra grafiniuose redaktoriuose. Šiuo metu tokius naudotojo valdymo elementų pavyzdžius galima paimti išnagrinėjus šiuos šaltinius:

- HTML valdymo elementus [XW09]. Kadangi tai yra internetinė aplikacija, todėl pirmiausia prasminga pažiūrėti, kokius elementus galima realizuoti su SVG, kurie atitinka įprastus žiniatinklio valdymo elementus. Šie elementai turi būti realizuojami pirmiausia, nes tai yra tokie elementai, be kurių apsieiti praktiškai nebeįmanoma.
- Flash valdymo elementus [BCS09][CA09]. Flash buvo parinktas todėl, kad jis dabar yra labai plačiai naudojamas žiniatinklio puslapiuose: tai ir atskiri valdymo elementai, ir su valdymu nesusiję objektai. Yra netgi nemažai svetainių, kurios sukurtos tik su Flash, t.y. ir navigacija, ir kiti valdymo elementai, ir turinys – viskas padaryta su Flash. O tai reiškia, kad prasminga pažiūrėti, kokius valdymo

elementus siūlo Flash, kurių nėra HTML ir kurie būtų naudingi nagrinėjimui iš SVG realizavimo pusės.

- Windows Forms valdymo elementus [MS09]. Windows yra populiariausia platforma, o tai reiškia, kad negalima apeiti ir šios dalies. Nors Windows valdymo elementai yra skirti ne internetinėms aplikacijoms, o paprastoms programoms, bet ir čia galima rasti elementų, kuriuos būtų prasminga panagrinėti iš SVG pusės, tuo labiau, kad jų gali prireikti kuriant aplikacijas, kurias teikia paprastoms programoms būdingus servisus.

Išnagrinėjus aukščiau aprašytus šaltinius buvo pasirinkti tokie elementai, kurie teiks reikalingas paslaugas kuriant įvairaus pobūdžio internetines aplikacijas. Galimų valdymo elementų sąrašas pateikiamas sugrupavus elementus į grupes pagal jų teikiamas paslaugas. Taip pat verta paminėti, kad kai kuriuos elementus galima priskirti prie kelių grupių, tačiau šiame sąraše pasikartojantys elementai praleidžiami. Elementų išvaizda (jei ji egzistuoja) pateikta priede A.

- Informacijos pateikimo elementai:
  - Label / Text. Tai yra paprastas statinis tekstas, kuris teikia tam tikrą informaciją. Ribos tarp „Label“ ir „Text“ yra gana miglotos. Paprastai šie elementai skiriami pagal pateikiamos informacijos dydį: jei informacijos nedaug (antraštė, užrašas ir t.t), paprastai turima omeny „Label“, kai informacijos yra daug (tekstas, aprašymas ir t.t.), sakoma, kad tai yra „Text“.
  - Image. Tai yra paprastas neinteraktyvus paveikslas, kuris gali būti ir statinis, pavyzdžiui, jpg, png, ir dinaminis, pavyzdžiui, animuotas gif.
  - Chart. Tai, galima sakyti, yra atskiras „Image“ elemento atvejis. Paprastai „Chart“ skiriasi nuo „Image“ savo turiniu. Chart paprastai naudojamas įvairioms diagramoms, grafikams vaizduoti, tuo tarpu Image turinys iš esmės neribojamas ir gali būti bet koks.
  - Tooltip. Tai nėra savarankiškas elementas. Jis būtinai turi būti susietas su kitais elementais. Šio elemento paskirtis pateikti trumpą informaciją apie kitą, susietą elementą. Paprastai informacija pateikiama atskirame mažame lange šalia žymeklio, užvedus pastarąjį ant susieto elemento.

- Progressbar. Tai yra elementas, skirtas parodyti einamosios, paprastai ilgai vykdomos, užduoties atlikimo eigą, t.y. parodyti, kiek užduoties yra įvykdyta ir kiek dar liko įvykdyti.
- List. Tai yra tam tikros informacijos pateikimas sąrašo pavidalu.
- TreeView. Tai yra informacijos pateikimo medžio pavidalu elementas. Medžio šakas galima išskleisti ir sutraukti.

- Valdymo elementai:
  - Link. Tai yra nuoroda, kurią spragtelėjus galima pereiti į kitą dokumentą arba į kitą dokumento vietą.
  - Input. Tai yra informacijos įvedimo laukas. Paprastai įvedamas nedidelis informacijos kiekis viena eilute.
  - Combobox. Tai yra pasirinkimo elementas. Šis elementas leidžia pasirinkti vieną reikšmę iš galimų reikšmių sąrašo.
  - Button. Tai yra mygtukas. Mygtuko paspaudimas iškviečia jam priskirtą veiksmą.
  - Alert. Tai yra paprastas pranešimo langas. Dažniausiai jame pateikiama aktualiausia informacija, perspėjimas, pranešimai apie klaidas ir pan.
  - Checkbox. Tai yra binarinės informacijos įvedimo elementas. Šis elementas leidžia pasirinkti/įvesti „Taip“ / „Ne“ tipo informaciją. Paprastai „Checkbox“ elementas naudojamas kartu su „Label“ elementu, kuris paaiškina įvedamosios reikšmės prasmę.
  - RadioButton. Tai yra pasirinkimo elementas. Dažniausiai jis naudojamas ne pavieniui, o grupėse. „RadioButton“ elementų grupė leidžia pasirinkti tik vieną elementą iš grupės.
  - IconButton. Šis elementas yra panašus į „Button“ elementą, tik vietoj mygtuko pavadinimo naudojamas paveiksliukas (piktograma), nusakantis to mygtuko paskirtį.
  - Menu. Tai yra sugrupuotų ir struktūrizuotų pagal kokius nors kriterijus veiksmų sąrašas. Dažnai turi daugiau nei vieną lygį.
  - Calendar. Tai yra datos ir laiko pasirinkimo elementas. Data pateikiama kalendoriaus pavidalu.
  - Textarea. Tai yra teksto įvedimo elementas. Šis elementas skirtas didesniems negu elemento „Input“ teksto kiekiams įvesti.

- RichEdit. Tai yra teksto su formatavimu įvedimo elementas. Panašus į „Textarea“, tik su papildomomis funkcijomis, pavyzdžiui, leidžiantis nurodyti spalvą, dydį, šriftą ir t.t.
- Uploader / loader. Tai yra įvedimo elementas, skirtas nusiųsti objektą (pavyzdžiui, bylą) į serverį.
- Colorpicker. Tai yra spalvos pasirinkimo elementas.
- Site Map. Šis elementas parodo svetainės struktūrą.
- Dialog. Tai yra reikalingos informacijos užklausimo elementas. Priklausomai nuo informacijos, „Dialog“ elementas gali būti skirtingo pavidalo ir sudėtingumo.
- Slider. Tai yra skaitinės informacijos įvedimo elementas. Naudotojas, judindamas specialią žymę į kairę / dešinę arba aukštyn / žemyn, pasirenka skaitinė reikšmę.
- Player. Tai yra filmų vaizdavimo elementas.
- Player control. Tai yra „Player“ elemento valdymo elementų grupė. Paprastai valdymo elementai turi tokias funkcijas, kaip paleisti filmą, pristabdyti filmą, sustabdyti filmą, persukti į pradžią arba į pabaigą, sulėtinti vaizdavimą.

- Išdėstymo elementai (konteineriai):
  - Grid. Šis elementas leidžia patalpinti kitus elementus savyje išdėstant tinklo pavidalu.
  - Spliter. Šis elementas skirtas atskirti vienus elementus nuo kitų arba vieną elementų grupę nuo kitos.
  - Tabstrip. Šis elementas leidžia apjungti kelis puslapius į vieną ir persijungti tarp jų su auselėmis.
  - Toolbar. Tai įrankių juostos elementas. Paprastai jis naudojamas saugoti kitus elementus, sugrupuotus pagal paskirtį.
  - Multipage. Tai yra bendresnis „Tabstrip“ variantas. Perjungimo elementai gali būti ne tik auselės, bet ir kitokie.
  - Rotator. Šis elementas leidžia vaizduoti patalpintus jame objektus ciklišk ai. Vaizduojant paskutinį elementą ir nurodžius komandą „vaizduoti kitą“, bus vaizduojamas pirmas elementas ir atvirkščiai, vaizduojant pirmą elementą ir nurodžius komandą „vaizduoti ankstesnį“, bus rodomas paskutinis elementas.

2.2 Elementų realizacijos pavyzdžiai

Toliau panagrinėsime, kaip keletą aukščiau išvardintų elementų galima realizuoti su SVG. Jei SVG standartinių priemonių realizacijai neužteks papildomai panaudosime JavaScript.

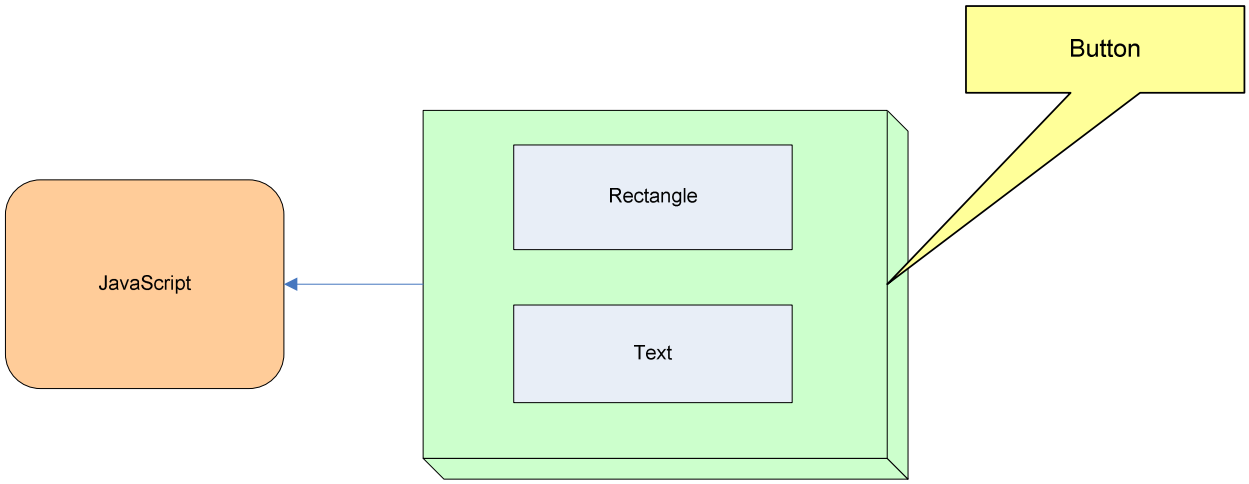
2.2.1 Mygtukas

Button.

```
<?xml version="1.0" encoding="utf-8"?>
<svg xmlns="http://www.w3.org/2000/svg" width="150" height="50" >
  <g onclick="window.open('http://www.google.com','_self')">
    <rect x="0" y="0" width="150" height="30" rx="10" ry="10" fill="red"/>
    <text x="75" y="22" font-size="20" text-anchor="middle" fill="yellow"
font-weight="900">Click Here</text>
  </g>
</svg>
```

Pav. 11 Button elementas

„Button“ elemento realizacijos komentaras. Elementas *g* naudojamas sugrupuoti jame apibrėžtus elementus į vieną objektą. Atributas *onclick* atsakingas už veiksmą, kurį reikia atlikti, kai naudotojas spragtels objektą, sugrupuotą iš kelių elementų. Elementas *rect* piešia stačiakampį, kuris atitinka mygtuką, o elementas *text* nurodo mygtuko tekstą. Stačiakampio ir teksto parametrai nusakomi per atitinkamus tų elementų atributus.



Pav. 12 Button struktūra

2.2.2 Antraštė / tekstas

Label / Text.

```
<?xml version="1.0" encoding="utf-8"?>
<svg xmlns="http://www.w3.org/2000/svg" width="150" height="50" >
  <text x="0" y="20" font-size="20" fill="black">Label</text>
</svg>
```

Pav. 13 Label / Text elementas

„Label / Text“ elemento realizacijos komentaras. Tai, galima sakyti, yra vienas iš paprasčiausių elementų. SVG elementas *text* naudojamas išvesti nurodytą tekstą atitinkamoje pozicijoje ir su nurodytais parametrais.

Pav. 14 Label / Text struktūra

2.2.3 Žymimasis langelis

Checkbox.

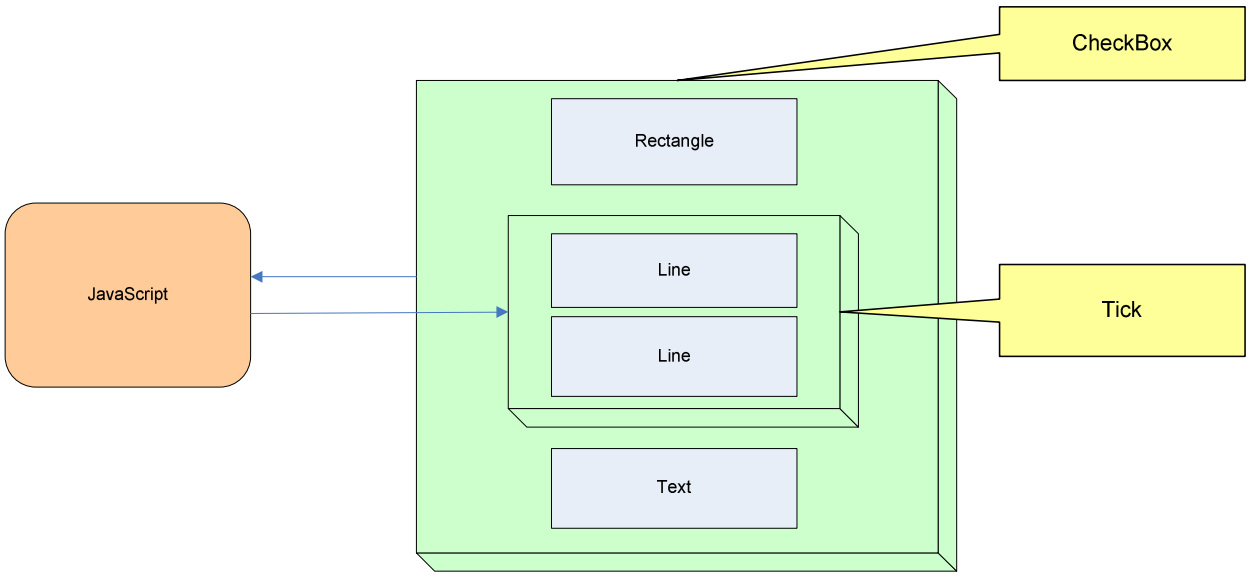
```
<?xml version="1.0" encoding="utf-8"?>
<svg xmlns="http://www.w3.org/2000/svg" width="150" height="50">
  <script>
    <![CDATA[
        function toggle()
        {
          var tick=document.getElementById('tick');
          var newValue = tick.getAttributeNS(null, 'visibility')
          if ('hidden' != newValue)
          {
             newValue = 'hidden';
          }
          else
          {
             newValue = 'visible';
          }
          tick.setAttributeNS(null, 'visibility', newValue);
        }
      ]]>
  </script>
  <g onclick="toggle()">
    <rect x="0" y="0" width="20" height="20" fill="white" stroke="black"/>
    <g id="tick">
      <line x1="0" y1="0" x2="20" y2="20" stroke="green" />
      <line x1="0" y1="20" x2="20" y2="0" stroke="green" />
    </g>
```

```
    <text x="30" y="20" font-size="20" fill="black">Checkbox</text>
  </g>
</svg>
```

Pav. 15 “Checkbox” elementas

„Checkbox“ elemento realizacijos komentaras. Šis elementas priklauso valdymo grupės elementams, todėl be papildomos veikimo logikos realizavimo jis nėra įgyvendinamas. Elemento veikimo logika aprašoma JavaScript pagalba. JavaScript kodas nurodomas elemente *script*: apibrėžiama viena funkcija *toggle()*, kuri iš DOM paima elementą (objektą) su *id „tick“* ir nustato atributo *„visibility“* reikšmę. Ši reikšmė parodo, ar objektas vaizduojamas, ar ne. Toliau, jei objektas buvo vaizduojamas, jam keičiama ši reikšmė, kad jis taptų paslėptas ir atvirkščiai, jei objektas buvo paslėptas, jam keičiama ši reikšmė, kad jis taptų vaizduojamas.

Pirmas (išorinis) elementas *g* sugrupuoja visus viduje esančius elementus į vieną objektą – „Checkbox“ ir su atributu *onclick* aprašo veiksmą, kurį reikia atlikti, kai naudotojas spragtels pele (šiuo atveju bus iškviečiama funkcija *toggle()*). Elementas *rect* naudojamas nupiešti stačiakampį. Antras (vidinis) elementas *g* sugrupuoja dvi linijas, nupieštas su elementu *line*, į vieną objektą ir priskiria tam objektui *id „tick“*. Kaip jau buvo paminėta aukščiau, paprastai elementas „Checkbox“ naudojamas kartu su elementu „Label / Text“, kuris paaiškina įvedamos informacijos prasmę. Elementas „Label / Text“ realizuojamas su žyme *text*.



Pav. 16 „CheckBox“ struktūra

2.2.4 Akutė

RadioButton.

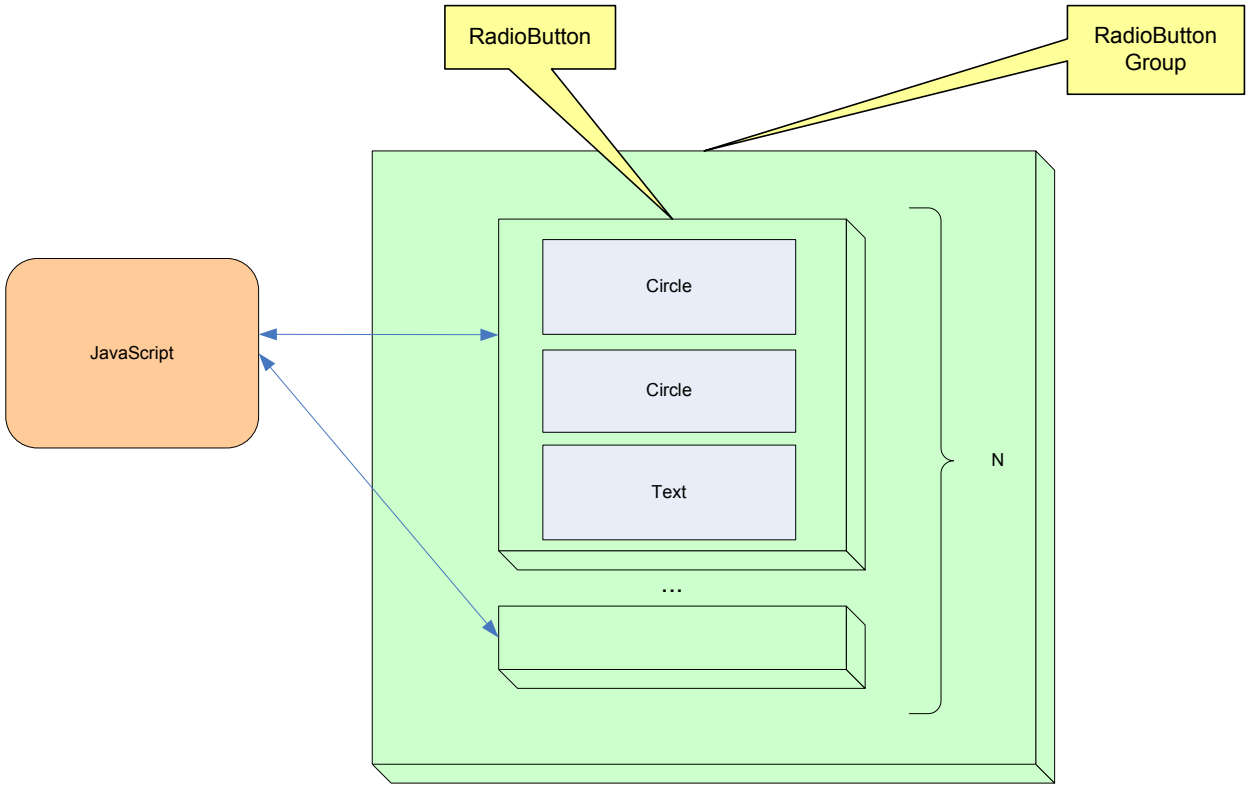
```
<?xml version="1.0" encoding="utf-8"?>
<svg xmlns="http://www.w3.org/2000/svg" width="150" height="150">
  <script>
    <![CDATA[
        function toggle(id, max)
        {
          var i=1;
          for(i=1;i<=max;i++)
          {
            var tick=document.getElementById('rb'+i);
            if(i == id)
            {
              tick.setAttributeNS(null, 'visibility', 'visible');
            }else
            {
              tick.setAttributeNS(null, 'visibility', 'hidden');
            }
          }
        }
      ]]>
  </script>
  <g onclick="toggle(1, 3)" transform="translate(0,0)">
    <circle cx="10" cy="10" r="10" fill="white" stroke="red"/>
    <circle id="rb1" cx="10" cy="10" r="5" fill="black"
visibility="visible"/>
    <text x="30" y="15" font-size="20" fill="black">RadioButton 1</text>
  </g>
  <g onclick="toggle(2, 3)" transform="translate(0,50)">
    <circle cx="10" cy="10" r="10" fill="white" stroke="red"/>
    <circle id="rb2" cx="10" cy="10" r="5" fill="black" visibility="hidden"/>
    <text x="30" y="15" font-size="20" fill="black">RadioButton 2</text>
  </g>
  <g onclick="toggle(3, 3)" transform="translate(0,100)">
    <circle cx="10" cy="10" r="10" fill="white" stroke="red"/>
    <circle id="rb3" cx="10" cy="10" r="5" fill="black" visibility="hidden"/>
    <text x="30" y="15" font-size="20" fill="black">RadioButton 3</text>
  </g>
</svg>
```

Pav. 17 RadioButton Group elementas

„RadioButton“ elemento realizacijos komentaras. Šis elementas, kaip ir „Checkbox“, priklauso įvedimo elementų grupei ir turi savo veikimo logiką, kuri realizuojama su JavaScript. Kaip jau buvo minėta, paprastai šis elementas nenaudojamas atskirai, o grupuojamas po kelis elementus. Funkcija *toggle(id, max)*, realizuojanti elemento logiką, apibrėžiama su žyme *script*. Funkcijos parametras *id* nusako, koks elementas buvo paspaustas (iš „RadioButton“ elementų grupės), o parametras *max* nusako, kiek iš viso yra elementų. Toliau iteruojama per visus elementus ir tikrinama, ar tas elementas sutampa su paspaustu elementu. Jei taip, tai tam elementui vaizduojamas taškas, kitu atveju – elemento taškas yra paslepiamas. Tai daroma atitinkamai nustačius atributą *„visibility“*.

Toliau piešiami trys „RadioButton“ elementai. Panagrinėsime tik pirmą, nes kiti yra analogiški. Elementas *g* sugrupuoja visus vaizdavime naudojamus elementus į vieną vienetą. Taip pat per atributą *onclick* nustatomas veiksmas, kuris bus atliktas, kai naudotojas paspaus tą

bendrą elementą. Šiuo atveju bus iškviesta funkcija *toggle*. Taip pat per atributą *transform* nustatoma elemento padėtis. Pirmasis elementas *circle* piešia išorinį skritulį su atitinkamais parametrais, nusakytais per elemento atributus. Antrasis elementas *circle* piešia vidinį skritulį, kuris vizualiai parodo, koks iš „RadioButton“ elementų pasirinktas šiuo metu. Be standartinių šiam elementui atributų dar naudojami ir kiti: *id* – nusako „RadioButton“ elemento identifikacinį numerį (kuris naudojamas funkcijoje *toggle*) ir *visibility* – pasako, ar vaizduoti elementą, ar ne.



Pav. 18 „RadioButton Group“ struktūra

Pateiktoji realizacija parodo tik esminius principus, kaip galima sukonstruoti vieną ar kitą naudotojo valdymo elementą su SVG. Tačiau kuriant grafinį redaktorių, toks būdas nėra labai geras, nes norint papildyti kokių nors elementų sąsają, reikės kopijuoti visą šablono kodą į redaktoriaus kodą. Geresnis sprendimas būtų apjungti tuos elementus į vieną biblioteką ir, panaudojus iš anksto apibrėžtą sąsają, įterpti elementus į reikalingą vietą.

2.3 Vektorinės grafikos redaktorius

2.3.1 Redaktoriaus reikalavimai

Kadangi vienas iš darbo tikslo uždavinių yra sukonstruoti paprastą vektorinės grafikos redaktorių, tai pirmiausiai reikia apibrėžti tam redaktoriui funkcinius (funkcionalumo arba

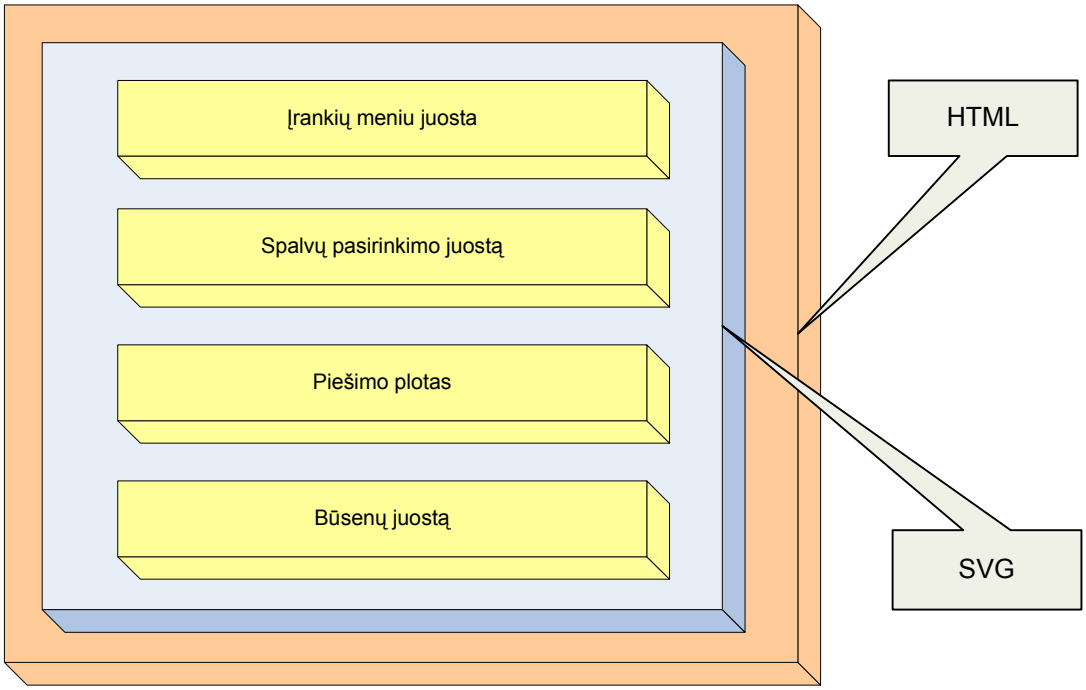
teikiamų paslaugų) ir nefunkcinius (sistemos savybių ir apribojimų) reikalavimus. Apibrėžus šiuos reikalavimus bus galima daugiau dėmesio skirti tiems elementams ir komponentams, kurie reikalingi šiems reikalavimams įgyvendinti. Toliau pateikiamas reikalavimų sąrašas:

- Funkciniai redaktoriaus reikalavimai:
  - turi gebėti nupiešti:
    - tašką;
    - liniją;
    - stačiakampį;
    - ovalą;
    - tekstą.
  - pasirinkti elemento spalvą;
  - išsaugoti nupieštą paveikslą serveryje arba galimybė atsiųsti nupieštą paveikslą į savo kompiuterį;
  - užkrauti išsaugotą paveikslą iš serverio arba iš kompiuterio.
- Nefunkciniai reikalavimai:
  - redaktorius turi veikti naršyklėse (kurios palaiko SVG pagal nutylėjimą) be papildomų įskiepių;
  - turi atitikti SVG 1.1 standartą;
  - reikalauja JavaScript 1.5 versijos.

2.3.2 Redaktoriaus realizavimo pavyzdys

Kaip vienas iš darbo uždavinių buvo sukurtas veikiantis vektorinės grafikos redaktoriaus pavyzdys, kuris atitinka aukščiau aprašytus reikalavimus. Grafinis redaktorius buvo pritaikytas Mozilla Firefox naršyklei, kuri palaiko SVG pagal nutylėjimą (buvo testuojama su Firefox 3.0.x). O tam, kad būtų galima realizuoti vieną iš redaktoriaus reikalavimų (išsaugoti piešinį), buvo sukurta serverinė dalis.

Redaktoriaus pagrindinių elementų išdėstymo diagrama:



Pav. 19 SVG redaktoriaus elementų išdėstymo diagrama

Toliau aptarsime esminius redaktoriaus elementus ir kaip tie elementai atlieka savo funkcijas. Visas redaktoriaus kodas pateiktas priede B.

Žemiau pateiktas mygtuko „Nupiešti liniją“ („Draw Line“) – vieno iš įrankių meniu juostoje – paaiškinimas.

```
<g onclick="selectShape(1)" transform="translate(10,10)"
onmousemove="setStatus('Draw Line')">
  <rect x="0" y="0" width="50" height="20" fill="yellow" stroke="black"
id="b1"/>
  <line x1="5" y1="10" x2="45" y2="10" stroke="black"/>
</g>
```

Pav. 20 Mygtukas „Nupiešti liniją“

Mygtuką sudarantys elementai grupuojami su žyme *g*. Elementas *rect* nupiešia stačiakampio formos mygtuką, o elementas *line* parodo mygtuko piktogramą, kuri atitinka atliekamą veiksmą (šiuo atveju nupiešti liniją). Užvedus pelę ant mygtuko, iškviečiamas įvykis *onmouseover*, kuris savo ruožtu kviečia JavaScript funkciją *setStatus* – parodo pateiktą pranešimą būsenos juostoje. Paspaudus pelę ant mygtuko iškviečiama funkcija *selectShape*, kuri nustato, koks elementas/įrankis (elemento/įrankio ID) pasirinktas. Elemento *g* atributas *transform* nustato mygtuko poziciją. Analogiškai realizuoti ir likusieji piešimo įrankiai. Skirtumas tik tas, kokia piktograma nupiešta ir koks įrankio ID yra nustatomas.

Toliau aprašysime funkcijas, panaudotas aukščiau paminėtam mygtukui JavaScript:

```
function selectShape(id)
```

```
{
    shape=id;
    var el;
    var i;
    for(i=1; i<6; i++)
    {
      el=document.getElementById("b"+i);
      if(i==id)
      {
        el.setAttribute("fill", "yellow");
      }else
      {
        el.setAttribute("fill", "green");
      }
    }
}
```

Pav. 21 Funkcija „selectShape“

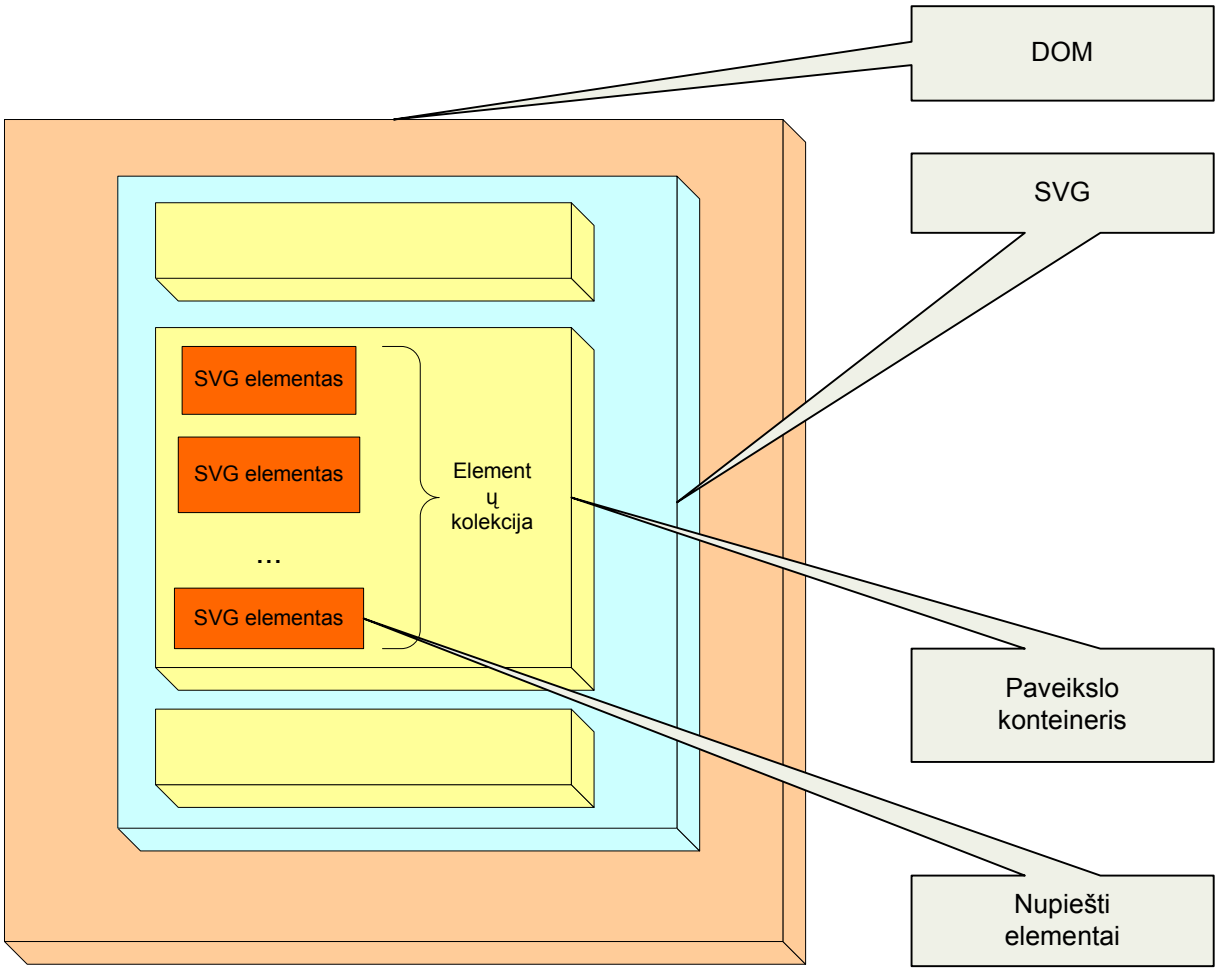
Funkcija *selectShape* pirmiausiai nustato, koks objektas pasirinktas (kintamasis *shape*), o paskui pakeičia visų mygtukų spalvą priklausomai nuo to, ar mygtukas yra pasirinktas (geltona spalva), ar nepasirinktas (žalia spalva).

Funkcija *setStatus* paima elementą, kurio ID yra „StatusText“ ir priskiria jam paduotą tekstinę žymę.

```
function setStatus(text)
{
    document.getElementById("StatusText").firstChild.nodeValue = text;
}
```

Pav. 22 Funkcija „setStatus“

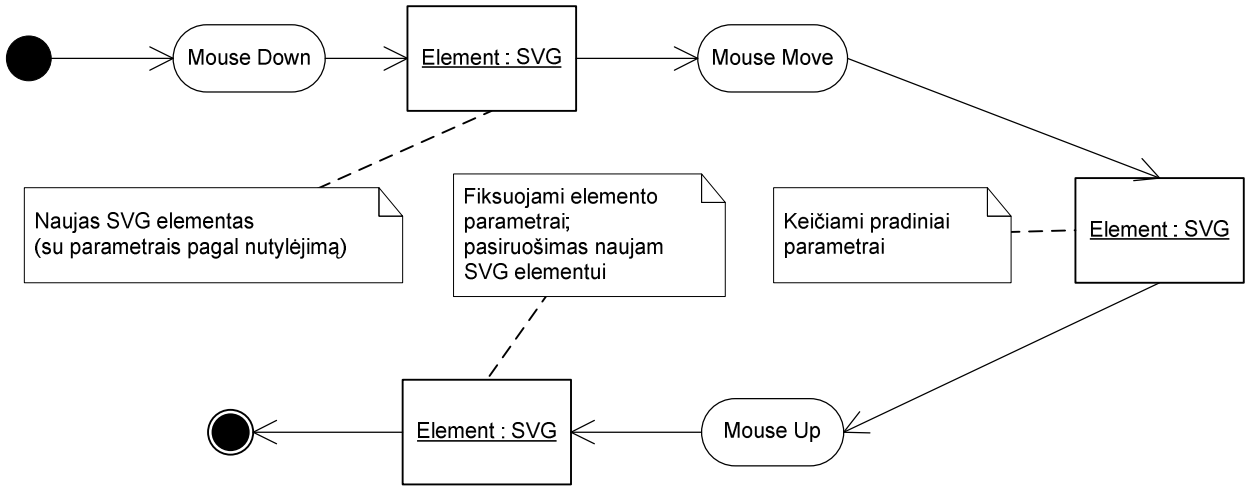
Kitaip realizuotas mygtuko „Save“, kurio tikslas išsaugoti nupieštą paveikslą, funkcionalumas. Piešiamas mygtukas analogiškai, kaip ir aukščiau aprašyti mygtukai, tačiau vietoje JavaScript funkcijos *selectShape* naudojama kita funkcija – *save*. Prieš aprašant funkcijos *save* realizaciją verta pasakyti, kaip konstruojamas SVG piešinys:



Pav. 23 Redaktoriaus DOM struktūra

Išsirinkus piešimo įrankį (taškas, linija, stačiakampis ir t.t.) ir paspaudus pele piešimo sritį, sukuriamas pasirinktas *svg* objektas ir įtraukiamas į DOM medį. Naujai sukurtam objektui priskiriami pradiniai jam reikalingi parametrai (pavyzdžiui, pradinės koordinatės) ir taip pat tam objektui sukurtas ID, pagal kurį bus galima pasiekti sukurtą objektą, kai to prireiks.

Toliau naudotojas stumia pelę (neatleisdamas paspausto mygtuko) ir tokiu būdu keičia objekto parametrus, pavyzdžiui, galutinį linijos tašką (pradinis taškas buvo nustatytas mygtuko paspaudimo vietoje). Visą tą laiką naudotojas gali vizualiai stebėti, kaip keičiasi objekto parametrai (kaip objektas keičia savo formą). Kai galutinė padėtis nustatoma, naudotojas atleidžia pelės mygtuką ir objektas užsifiksuoja, atliekami paruošimo veiksmai piešti kitą objektą. Šį principą iliustruoja diagrama:



Pav. 24 Darbo eiga

Žinant esminius principus, pagal kuriuos piešinys konstruojamas ir įtraukiamas į DOM, galima grįžti prie funkcijos *save* realizacijos.

```
function save()
{
    var i;
    var s="<?xml version='1.0' encoding='UTF-8'?><svg
xmlns='http://www.w3.org/2000/svg'>";
    for (i=0; i<lastId; i++)
    {
        var id = "el_"+i;
        var el = document.getElementById(id);

        var name = el.nodeName;
        var elem="";
        switch(name)
        {
            case "line":
            elem='<line id="'+id+'" x1="'+el.getAttribute("x1")+'"
y1="'+el.getAttribute("y1")+'" x2="'+el.getAttribute("x2")+'"
y2="'+el.getAttribute("y2")+'" stroke="'+el.getAttribute("stroke")+'" />';
            break;
            case "rect":
            elem='<rect id="'+id+'" x="'+el.getAttribute("x")+'"
y="'+el.getAttribute("y")+'" width="'+el.getAttribute("width")+'"
height="'+el.getAttribute("height")+'" stroke="'+el.getAttribute("stroke")+'"
fill="'+el.getAttribute("fill")+'" />';
            break;
            case "ellipse":
            elem='<ellipse id="'+id+'" cx="'+el.getAttribute("cx")+'"
cy="'+el.getAttribute("cy")+'" rx="'+el.getAttribute("rx")+'"
ry="'+el.getAttribute("ry")+'" stroke="'+el.getAttribute("stroke")+'"
fill="'+el.getAttribute("fill")+'" />';
            break;
            case "text":
            elem='<text id="'+id+'" x="'+el.getAttribute("x")+'"
y="'+el.getAttribute("y")+'"
fill="'+el.getAttribute("fill")+'">'+el.firstChild.nodeValue+'</text>';
            break;
            case "circle":
            elem='<circle id="'+id+'" cx="'+el.getAttribute("cx")+'"
cy="'+el.getAttribute("cy")+'" r="'+el.getAttribute("r")+'"
```

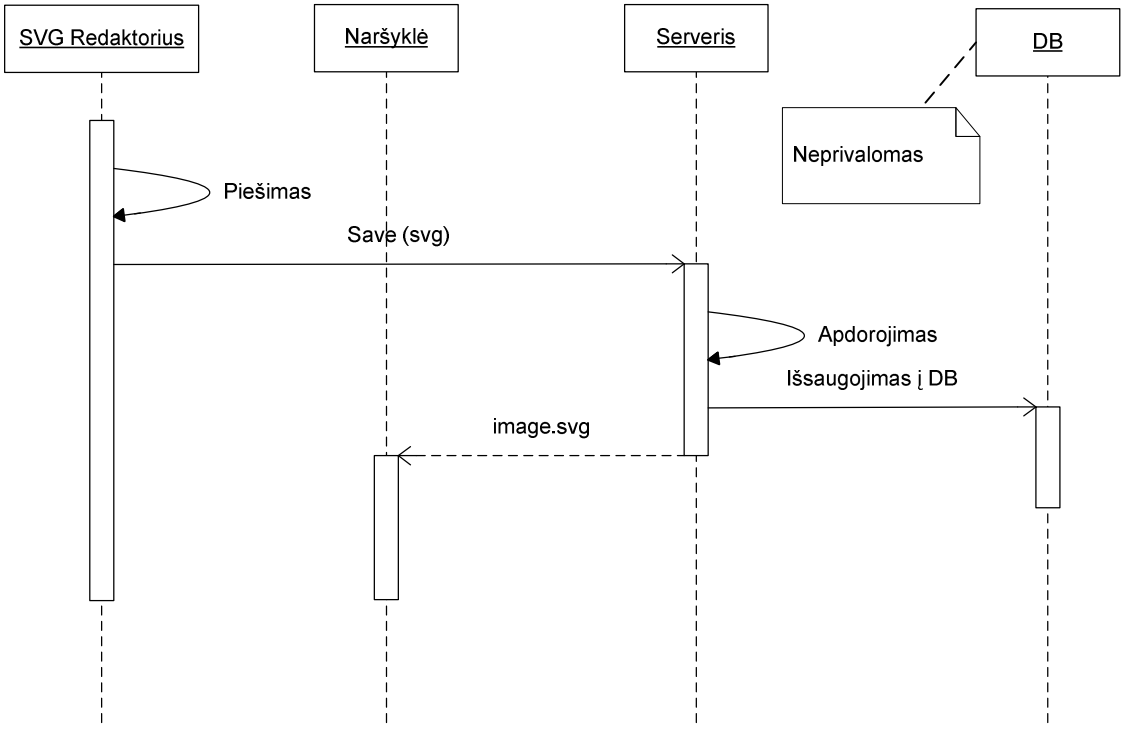
```
stroke="'+el.getAttribute("stroke")+'" fill="'+el.getAttribute("fill")+'"
/>';
            break;
        }
        s=s+elem;
    }
    s=s+"</svg>";
    document.forms['main'].svg.value=s;
    document.forms['main'].submit();
}
```

Pav. 25 Funkcija „save“

Funkcijos save algoritmas yra gana paprastas:

1. serializuoti piešinį;
2. nusiųsti gautą tekstinę eilutę į serverį;
3. serveris apdoroja gautus duomenis, jei to reikia (paprasčiausiu atveju apdorojimas nereikalingas), ir grąžina rezultatą (arba atlieka kitus reikalingus veiksmus, pavyzdžiui, išsaugo piešinį duomenų bazėje).

Aprašytą algoritmą grafiškai galima pavaizduoti tokia diagrama:



Pav. 26 Komandos "Save" vykdymo seka

Dabar kiekvieną žingsnį aptarsime detaliau.

Pirmas žingsnis – serializuoti piešinį. Serializacijos (arba kitais žodžiais objekto konvertavimas į tekstinį pavidalą) rezultatas saugomas kintamajame *s*. Kadangi mūsų rezultatas yra XML failas, tai pradžiai pridedama standartinė XML antraštė ir SVG žymė. Toliau vykdomas ciklas, kurio metu iteruojami visi nupiešti elementai (elementai atpažįstami pagal jiems priskirtus *id*) ir, priklausomai nuo gauto elemento tipo (elemento žymės pavadinimo, pavyzdžiui, *line*, *circle* ir t.t.), iš to elemento ištraukiami jo atributai (konstrukcijoje *switch* .. *case*). Ciklo pabaigoje uždaromas elementas *svg* ir kintamasis, kuris dabar saugo nupieštą piešinį tekstiniame pavidale, priskiriamas formos paslėptam (*hidden*) laukui.

Antras žingsnis – paruošta forma siunčiama į serverį (vykdomas veiksmas *submit*).

Trečias žingsnis – serveris apdoroja gautus duomenis ir grąžina rezultatą. Šio eksperimento serverinėje dalyje buvo naudojama .NET platforma, tačiau lygiai taip pat galima naudoti ir kitas platformas.

```
protected void Page_Load(object sender, EventArgs e)
    {
        XmlDocument xml = new XmlDocument();
        try
        {
            if (Request["svg"] != null)
            {
                //save
                xml.LoadXml(Request["svg"]);
                xml.Save(Server.MapPath("~\\data\\image.svg"));
                Response.ClearContent();
                Response.AddHeader("Content-Disposition", "attachment;
filename=image.svg");
                Response.AddHeader("Content-Length",
xml.OuterXml.Length.ToString());
                Response.ContentType = "image/svg+xml";
                Response.Write(xml.OuterXml);
            }
            else
            {
                //open
                xml.Load(Server.MapPath("~\\data\\image.svg"));
                Response.ContentType = "image/svg+xml";
                Response.Write(xml.DocumentElement.OuterXml);
            }
        }
        catch (Exception ex)
        {
            Response.End();
        }
    }
```

Pav. 27 Duomenų apdorojimo metodo realizacija C# kalba

Serveris atlieka labai paprastą darbą: užkrauna gautą iš formos tekstą į *XmlDocument* tipo kintamąjį, nustato atitinkamas atsakymo antraštes (nustato failo vardą – „image.svg“, failo ilgį ir tipą) ir grąžiną *xml* turinį. Tokiu būdu naršyklė interpretuoja serverio atsakymą kaip failą, kurį

reikia išsaugoti. Jeigu parametras *svg* nebuvo pateiktas, šis veiksmas traktuojamas kaip noras užkrauti anksčiau išsaugotą paveikslėlį.

Dar viena svarbi redaktoriaus funkcija – atidaryti (arba užkrauti) anksčiau išsaugotą paveikslėlį. Šis redaktoriaus funkcionalumas atliekamas su mygtuku „Open Last“. SVG požiūriu tai yra mygtukas, analogiškas aukščiau aprašytam mygtukui „Save“. Skiriasi tik pavadinimas ir kitos JavaScript funkcijos iškvietimas. Svarbiausia šiame veiksme yra tai, kad paveikslėlio užkrovimas vykdomas asinchroniškai (su AJAX), paveikslui užkrauti, nereikia perkrovinėti viso puslapio turinio, keičiasi tik turinio dalis. Žemiau pateikiama paveikslėlio užkrovimo funkcija:

```
function openPic()
{
    var loading = document.getElementById("loading");
    loading.setAttributeNS(null, 'visibility', 'visible');
    var xmlHttp;
    xmlHttp=new XMLHttpRequest();

    xmlHttp.onreadystatechange=function()
    {
        if(xmlHttp.readyState==4)
        {
            // delete all previous content
            var container = document.getElementById("d");
            for (i=0; i<lastId; i++)
            {
                var id = "el_"+i;
                var el = document.getElementById(id);
                container.removeChild(el);
            }
            lastId = 0;

            var xmldoc = xmlHttp.responseXML;
            var root = xmldoc.getElementById('svg');
            lastId = 1 * root.getAttribute("lastId");

            for (var iNode = 0; iNode < root.childNodes.length; iNode++)
            {
               var node = root.childNodes.item(iNode);
               var el=document.createElementNS(svgNS, node.nodeName);
               for(var iattr = 0; iattr < node.attributes.length; iattr++)
               {
                    el.setAttribute(node.attributes[iattr].nodeName,
node.getAttribute(node.attributes[iattr].nodeName));
                    if(node.nodeName == "text")
                    {
                        var el2 =
document.createTextNode(node.firstChild.nodeValue);
                        el.appendChild(el2);
                    }
               }
               container.appendChild(el);
            }
            loading.setAttributeNS(null, 'visibility', 'hidden');
        }
    }

    xmlHttp.open("GET","http://localhost/vge/default.aspx",true);
```

```
    xmlHttp.send(null);
}
```

Pav. 28 Paveikslėlio užkrovimo funkcija „openPic“

Dabar detaliau aptarsime šią funkciją. Pirmiausiai parodomas paveikslėlis (animuoto gif formatu), kuris informuoja naudotoją, kad vyksta paveikslėlio krovimas (to gali prireikti, kai, pavyzdžiui, paveikslėlis yra didelis arba interneto greitis yra mažas). Tai daroma nustačius elemento *image* atributą *„visibility“* (nustatoma reikšmė yra *„visible“*). Po to sukuriamas *XMLHttpRequest* objektas, kuris leidžia kurti HTTP užklausas serveriui neperkrovinėjant viso puslapio. Tai yra svarbiausioji AJAX technologijos dalis. Tada apibrėžiama funkcija, kuri bus kviečiama, kai iš serverio grįš atsakymas (*readyState=4*). Funkcijos pabaigoje vykdoma užklausa į serverį.

Kai iš serverio grįžta atsakymas (XML failas), esamas piešinys ištrinamas iš DOM. Tai daroma vykdant ciklą, kuriame iteruojami visi nupiešti elementai (pagal jų *id*) ir *removeChild* metodu išmetami iš DOM. Po to, kai piešimo plotas pasidarys tuščias, užkraunamas gautas iš serverio XML failas ir įtraukiamas jo turinys (elementai su jų atributais) į DOM panašiai, kaip tai buvo daroma piešiant elementus rankiniu būdu.

Kai visas XML iš serverio jau yra įtrauktas į DOM, paslepiamas naudotoją informuojantis apie vykdomą operaciją paveiksliukas. Tai daroma nustačius elemento *image* atributą *„visibility“* (nustatoma reikšmė yra *„hidden“*).

2.3.3 Realizuoto pavyzdžio aptarimas

Vektorinės grafikos redaktoriaus pavyzdžio realizavimas parodė, kad:

- įmanoma nupiešti statinį SVG vaizdą naršyklės lange;
- galima dinaminius elementus realizuoti įvairiais būdais;
- galima konstruoti nesudėtingas interaktyvias aplikacijas.

Tačiau, kaip galima matyti iš paties pavyzdžio, toks priėjimo būdas nėra itin priimtinas, nes:

- Pavyzdys buvo konstruojamas iš labai žemo lygio primityvų. Programavimo kalba būtų galima sakyti, kad redaktorius buvo kuriamas C arba net Assembler kalba.
- Nebuvo panaudoti jokie metodai ar principai, palengvinantys programinės įrangos kūrimą, pavyzdžiui:

    - pakartotinis kodo panaudojimas;
    - aukšto abstrakcijos lygio konstrukcijos;
    - ir kt.
- Praktiškai beveik neįmanoma sukurti didelės aplikacijos, t.y. toks priėjimas gali tikti tik labai mažoms aplikacijoms.

Naudoti žemo lygio primityvus kuriant aplikacijas šiais laikais nėra geras kūrimo metodas. Visada reikia stengtis operuoti aukštesnio abstrakcijos lygio komponentais. Tai, pirmiausia, gali ženkliai sumažinti klaidų skaičių, sumažina kūrimo laiką. Tokį kodą lengva skaityti, suprasti, pavyzdžiui, kilus būtinybei greitai ką nors pakeisti, apmokyti kitą žmogų ir panašiai. Žinoma, kai kuriais atvejais negalima apsieiti ir be žemo lygio konstrukcijų. Tai dažnai atsitinka, pavyzdžiui, kuriant tokius produktus, kaip operacines sistemas, realaus laiko aplikacijas, žaidimus, kitas programas, kur labai svarbūs kriterijai yra atsakos laikas, sistemos lankstumas ir pan. Nors internetinėse aplikacijose tokio tipo kriterijai yra taip pat svarbūs, tačiau jie neturi lemiamos reikšmės. Taigi internetines aplikacijas vertėtų iškelti į aukštesnį abstrakcijos lygį ir tik kai kuriais atvejais naudoti žemo lygio elementus, kai tai yra tikrai reikalinga ir be to negalima apsieiti.

Kitas paminėto pavyzdžio realizavimo trūkumas yra tas, kad, kuriant pavyzdį, nebuvo panaudoti jokie kūrimo palengvinimo metodai, tokie kaip, pavyzdžiui, pakartotinis kodo panaudojimas. Tai taip pat nėra gera programavimo praktika. Netaikant papildomo kodo panaudojimo metodų pirmiausia pailgėja aplikacijos kūrimo laikas, padidėja tikimybė padaryti daugiau klaidų ir t.t. Be to, jeigu prireiks kurti panašų projektą, tai negalėsime panaudoti mums reikalingų komponentų iš jau sukurto projekto, o tiesiog būsime priversti kopijuoti kodą ir taip padarysime klaidų. Tačiau jei ir pavyks tokiu būdu išplatinti kokį nors funkcionalumą per kelis projektus, atsiras dar viena sunkiai išsprendžiama problema – versijos. Jeigu vėliau tą išplatintą funkcionalumą modifikuosime (pavyzdžiui, pataisysime pastebėtą klaidą, optimizuosime greitį ir pan.), tai vietoje vieno atskiro modulio pakeitimo (pavyzdžiui, failo pakeitimo) teks kopijuoti tą kodą ir įterpti į visus projektus, kur tas funkcionalumas jau buvo panaudotas. Pagal kaštus toks būdas prilygsta funkcionalumo kūrimui iš naujo kiekvienam projektui.

Dar vienas paminėtas tokio kūrimo metodo trūkumas yra tas, kad taip praktiškai neįmanoma sukurti didelės aplikacijos. Toks metodas, galima sakyti, gali būti panaudotas (nors ir nerekomenduotinas dėl aukščiau išvardintų priežasčių) tik labai mažoms aplikacijoms. Tai atsitinka dėl to, kad metode nenumatytas kuriamos sistemos suskaidymas į modulius. Dirbant tokiu būdu nepavyksta arba yra per sudėtinga atskirti duomenų, verslo bei prezentacijos lygius į skirtingus modulius su aiškiomis sąsajomis. Dar prisideda ir labai ilgas kūrimo laikas, ir didelė

tikimybė padaryti klaidų, ir faktas, kad daug labai panašių dalykų teks kurti iš naujo užuot panaudojus tai, kas buvo sukurta seniau (galbūt tik pakeitus kai kuriuos parametrus) ir t.t. Kadangi toks darbo metodas nenumato atskirų modulių kūrimo (nėra pakartotinio kodo panaudojimo), tai neaišku, kaip tokį projektą gali kurti kelių žmonių komanda – jeigu vienas žmogus bus atsakingas už sąsajos elementų kūrimą, o kitas už pačią sąsają (plačiąja prasme), tai visai neaišku, kaip tai galima įgyvendinti. Tai yra tik vienas iš pavyzdžių, kuris rodo, kad toks sukurto pavyzdžio kūrimo metodas nėra geras ir turi daugybę trūkumų, ypač dideliems projektams realizuoti.

Panagrinėjus trūkumus gali iškilti natūralus klausimas – kokio tipo aplikacijoms paminėtas kūrimo metodas gali būti taikomas ir kokiomis aplinkybėmis jis gali turėti pranašumų prieš kitus metodus. Žemiau pateikiamas sąrašas kriterijų, pagal kuriuos būtų galima spręsti, kad toks metodas turi prasmę ir jį verta nagrinėti:

- Aplikacijos apimtis yra maža. Naudotojo sąsajoje naudojama tik keletas elementų, kuriuos realizuoti yra gana paprasta, jų veikimo logika yra nesudėtinga.
- Reikalingas didelis aplikacijos palaikomumas. Kadangi metodas numato grafinių objektų kūrimui naudoti tik SVG, JavaScript ir CSS, tai galima sakyti, kad taip nupieštą vaizdą parodys daugiausiai SVG žiūryklių.
- Aplikacijos kodas turi būti pilnai valdomas. Aplikacijai sukurti gali prireikti nestandartinių elementų arba nebus galima realizuoti standartinių elementų veikimo logiką, išvaizdą per tų elementų parametrus, atributus

Aišku, kito tipo projektams taikyti tokį metodą nerekomenduojama, tačiau, atsižvelgus į vektorinės grafikos redaktorių, buvo pasirinktas būtent toks kūrimo metodas.

2.3.4 Realizuoto pavyzdžio išvados

Realizavus vektorinės grafikos redaktorių, pirmiausia buvo parodyta, kaip praktiškai dirbti su SVG ir kaip galima taikyti SVG savo projektuose. Bandant realizuoti šį pavyzdį prieita nuomonės, kad, jeigu SVG aplikacija yra nesudėtinga (pavyzdžiui, analogiško sudėtingumo kaip ir SVG redaktorius), tai iš esmės galima taikyti tokį aplikacijos kūrimo metodą, kuris buvo panaudotas kuriant šį pavyzdį. Tokio darbo metodo pliusai yra šie:

- tai pakankamai greitas metodas (aplikacijos kūrimo atžvilgiu);

- kadangi naudojami paprasti primityvai, yra didelė tikimybė, kad toks metodas bus palaikomas daugumoje SVG mokančių parodyti aplikacijų (naršyklėse, SVG grotuvuose ir t.t.);
- produktas, sukurtas naudojant šį metodą, atitinka standartus (W3C).

Tačiau šis konstravimo būdas turi savų trūkumų:

- keblu kurti sudėtingas aplikacijas (kur yra daug valdymo elementų, valdymo elementai naudoja sudėtingą logiką ir pan.);
- tokias aplikacijas sunku derinti;
- sunku pakartotinai panaudoti kodą.

3 Alternatyvių taikymo variantų nagrinėjimas

O kaip kitaip būtų galima realizuoti vektorinės grafikos redaktoriaus pavyzdį, kad neliktų aukščiau išvardintų trūkumų arba tie trūkumai būtų minimizuoti? Pirmiausiai reikia pereiti nuo žemo abstrakcijos lygio prie aukštesnio. Perėjimas prie aukštesnio programavimo lygio automatiškai verčia pakartotinai naudoti kodą, o tai savo ruožtu greitina sistemos kūrimą, mažina potencialių klaidų skaičių, leidžia, kad projektą kurtų programuotojų komanda, o ne vienas žmogus ir t.t.

Toks perėjimas primena taip vadinamąsias turtingas interneto aplikacijas (angl. *Rich Internet Applications*, RIA). Tai tokio tipo aplikacijos, kurios realizuoja tradicinių aplikacijų funkcionalumą naršyklės lange. Paprastai RAI aplikacijos:

- perduoda web-klientui reikalingą vartotojo sąsają, palikdamos didžiausią duomenų (programos resursų, duomenų) dalį serveryje;
- paleidžiamos naršyklėje ir nereikalauja papildomos programinės įrangos diegimo;
- lokaliai paleidžiamos saugos srityje, dar vadinamoje smėlio dėže (angl. *sandbox*).

Yra keletas RIA platformų gamintojų, tokių kaip Microsoft (Silverlight), Sun (JavaFX), Adobe (Flex), kurie siūlo vienokį ar kitokį funkcionalumą, kūrimo principus ir panašiai.

Taigi yra daug įvairių būdų ir strategijų, kaip galima pereiti prie aukštesnio abstrakcijos lygio. Šiame darbe užtikrinti aukštesnį abstrakcijos lygį buvo pasirinkti šie vektorinės grafikos redaktoriaus realizavimo variantai:

- JavaScript. SVG elementų patalpinimas į modulius arba bibliotekas, parašytas su JavaScript;
- XBL kalbos panaudojimas Mozillos produktuose;

- SVG priemonės.

Žemiau bus plačiau aptarti išvardinti variantai.

3.1 SVG paslėpimas po JavaScript objektais

Pagrindinis šiuo varianto [SVG09a] principas yra paslėpti SVG primityvus už JavaScript objektų. Kitaip tariant, sukurti JavaScript biblioteką, kuri realizuotų naudotojo sąsajos elementus.

Kiekvieną iš grafinių elementų galima traktuoti kaip JavaScript objektą, kuris realizuos to elemento veikimo logiką, turės tam objektui būdingas savybes ir panašiai. Tokiu būdu galima paslėpti ne tik elementarius SVG elementus, bet ir jų derinius, t.y. ir stambesnius elementus, kurie neturi tiesioginio atitikmens SVG standarte, pavyzdžiui, meniu, medis, lentelė ir t.t. Vėliau, apjungus taip sukurtus sąsajos elementus į vieną biblioteką, pastarąją bus labai lengva panaudoti kituose projektuose.

3.1.1 Realizavimo pavyzdys

Žemiau pateikiamas pavyzdys, kaip būtų galima aprašytu būdu realizuoti vieną iš sąsajos elementų – mygtuką. Pradžiai pateikiama JavaScript biblioteka, realizuojanti mygtuko funkcionalumą:

```
svgNS = "http://www.w3.org/2000/svg";

function svgButton(containerId, id, x, y, text, action)
{
	this.id = id;
	this.x = x;
	this.y = y;
	this.text = text;
	this.action = action;
	this.containerId = containerId;
	this.width = 100;
	this.height = 30;
	this.Update = function()
	{
		var container = document.getElementById(this.containerId);
		var thisElem = document.getElementById(this.id);
		if (thisElem != null)
		{
			container.removeChild(thisElem);
		}

		var g = document.createElementNS(svgNS, "g")
```

```
            g.setAttribute("id", this.id);
            g.setAttribute("onclick", this.action);
            g.setAttribute("transform", "translate(" + this.x + "," + this.y +
")");

            var r=document.createElementNS(svgNS, "rect");
            r.setAttribute("x", 0);
            r.setAttribute("y", 0);
            r.setAttribute("width", this.width);
            r.setAttribute("height", this.height);
            r.setAttribute("stroke", "Red");
            r.setAttribute("fill", "Blue");

            var t=document.createElementNS(svgNS, "text");
            t.setAttribute("x", this.width/2);
            t.setAttribute("y", 20);
            t.setAttribute("text-anchor", "middle");
            t.setAttribute("fill", "Yellow");
            t.appendChild(document.createTextNode(this.text));

            g.appendChild(r);
            g.appendChild(t);

            document.getElementById(this.containerId).appendChild(g);
      }
}
```

Pav. 29 JavaScript bibliotekos pavyzdys

Bibliotekoje aprašomas objektas *svgButton*, nustatomos jo savybės (pozicija, tekstas ir t.t.), apibrėžiamas metodas *„Update“*, kuris atsakingas už mygtuko atvaizdavimą. Metodo pradžioje patikrinama, ar jau egzistuoja toks objektas (pagal objekto *id*). Jei taip, tai jis pašalinamas iš DOM. Toliau pagal esamas objekto savybes konstruojamas naujas SVG elementas ir metodo pabaigoje įterpiamas į DOM medį.

Toliau parodoma, kaip galima naudoti šią biblioteką kuriant savo SVG sąsają:

```
<svg xmlns="http://www.w3.org/2000/svg"
xmlns:xlink="http://www.w3.org/1999/xlink" width="600" height="500">
      <script xlink:href="svgUI.js" />
      <g width="600" heigth="500" id="containerId">

      </g>
      <script>
            var b = new svgButton("containerId", 1, 0, 0, "Button1",
"window.open('http://www.google.com')");
            b.x = 100;
            b.y = 10;
            b.Update();

            var b2 = new svgButton("containerId", 2, 0, 0, "Button2");
            b2.x = 200;
            b2.y = 100;
            b2.action = "b.text = 'Google'; b.Update()";
            b2.Update();

            var b3 = new svgButton("containerId", 3);
            b3.x = 50;
```

```
            b3.y = 150;
            b3.text = "Test";
            b3.action = "test()";
            b3.Update();

            function test()
            {
                  alert("Test ok");
            }
      </script>
</svg>
```

Pav. 30 JavaScript bibliotekos panaudojimas

Pavyzdyje sukuriami trys mygtukai: *b1*, *b2* ir *b3*. Paspaudus mygtuką *b1* atsidarys puslapis *www.google.com*, paspaudus mygtuką *b2* – mygtuko *b1* tekstas „Button1“ pasikeis į „Google“, o paspaudus mygtuką *b3* bus iškviečiama JavaScript funkcija *„test()“*.

Analogiškai galima realizuoti ir kitus elementus, kurie atitinka SVG elementus, bei sudėtingesnius elementus, sukomponavus paprastesnius į vieną bendrą elementą.

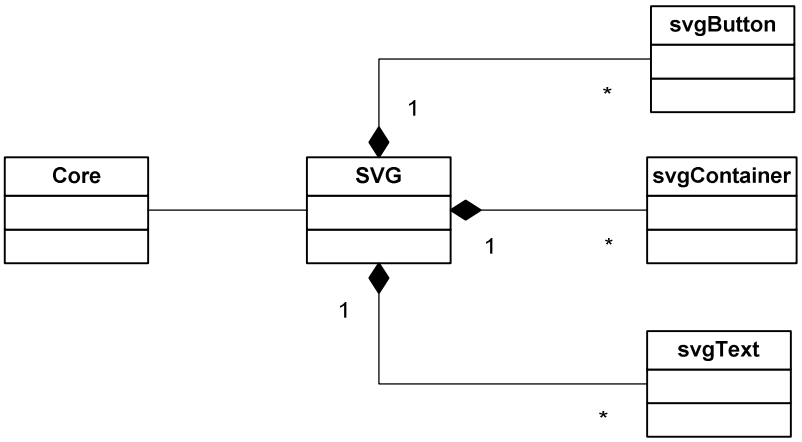
3.1.2 Metodo pritaikymas grafiniam redaktoriui

Pagrindinai redaktoriaus sąsajos elementai yra mygtukas ir panelė (konteineris), kurioje gali būti patalpinti kiti elementai. Grafiniam redaktoriui aukščiau parodytas mygtuko realizavimo pavyzdys buvo praplėstas, taip pat sukurti kiti grafiniai elementai:

- JavaScript mygtukas buvo papildytas naujais atributais:
  - *width* – mygtuko plotis;
  - *height* – mygtuko aukštis;
  - *fillColor* – mygtuko spalva;
  - *strokeColor* – mygtuko rėmelio spalva;
  - *mouseMove* – veiksmas, įvykdomas užvedant pelės žymeklį ant mygtuko.
- sukurtas naujas grafinis elementas – panelė (*svgContainer*), kuri atstovauja elementų konteinerio vaidmenį;
- sukurtas naujas grafinis elementas – tekstas (*svgText*), kuris moka parodyti jam priskirtą tekstą;

Nauji elementai – *svgContainer*, *svgText* – buvo sukurti analogiškai kaip ir detaliau aprašytas elementas *svgButton*. Taip pat galima pasakyti, kad grafinio redaktoriaus branduolys, atsakingas už naudotojo veiksmų apdorojimą, buvo pakartotinai panaudotas.

Pritaikius šį metodą vektorinės grafikos redaktoriaus komponentai ir jų sąsaja tarpusavyje atrodo taip:



Pav. 31 Grafinio redaktoriaus struktūra

3.1.3 Metodo aptarimas

Kaip galima matyti iš pavyzdžio, metodo realizavimui reikalingos geros JavaScript žinios. Galima sakyti, kad toks darbo metodas yra geras, kai kuriamos aplikacijos nėra itin didėlės, tačiau vieną kartą sukūrus gerą elementų biblioteką, ją bus galima panaudoti kituose projektuose. Taip pat kadangi naudojami tik JavaScript ir SVG (netiesiogiai), tai šis metodas turi būti palaikomas daugumos SVG mokančių rodyti įrankių.

Kaip ir visi kiti metodai, šis metodas turi savo privalumų ir trūkumų. Metodo privalumai:

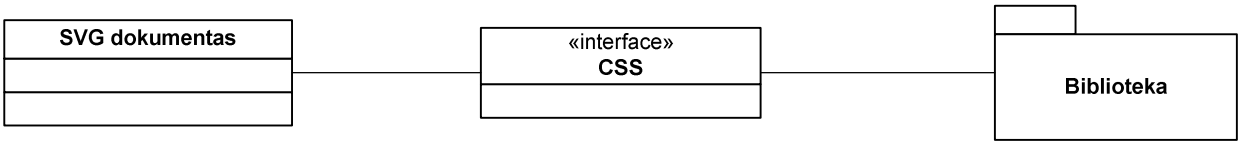
- pakartotinis kodo panaudojimas;
- didelė tikimybė, kad šis metodas bus palaikomas daugumos SVG mokančių parodyti įrankių (naršyklės, įskiepiai ir pan.);
- galima realizuoti ne tik elementų išvaizdą, bet ir elementų veikimo logiką, valdymą.

Tačiau yra ir trūkumų:

- elementai kuriami dinamiškai, dėl to nesimato dokumento struktūros išeities tekstuose, o tai apsunkina programavimą, derinimą, kodas tampa blogai skaitomas;
- kadangi beveik visas SVG dokumentas kuriamas su JavaScript, tai gali neigiamai atsiliepti našumui;
- nelengva sukurti elementų biblioteką.

3.2 SVG paslėpimas po XBL elementais

Kitas sąsajos elementų kūrimo variantas reikalauja panaudoti XBL kalbą. Iš plačiai naudojamų SVG grotuvų XBL kalbą palaiko Mozillos produktai. Kaip ir ankstesniame variante, galima sudaryti sąsajos elementų biblioteką, tačiau, skirtingai negu su JavaScript, elementai aprašomi XML failuose. Tada per CSS failą atskirai aprašytus vartotojo elementus galima susieti su kitu XML failu:



Pav. 32 XBL elementų susiejimas

Aprašant savo elementus galima naudoti ne tik SVG kalbą, bet ir JavaScript elemento veikimo logikai realizuoti. Tokiu būdu galima aprašyti pilnaverčius elementus ir naudoti savo reikmėms.

3.2.1 Realizavimo pavyzdys

Žemiau pateikiamas pavyzdys, kaip tokį sąsajos kūrimo metodą galima realizuoti praktiškai:

Failas „testXBL.xml“:

```
<?xml version="1.0"?>

<?xml-stylesheet href="testXBL-bindings.css" type="text/css"?>
<svg xmlns="http://www.w3.org/2000/svg"
     xmlns:svg="http://www.w3.org/2000/svg" id="canvas">
  <myButton transform="translate(100,100)" style="fill: blue;"
buttonText="button1" myAction="alert('alert button1');" />
  <myButton transform="translate(300,100)" style="fill: red;"
buttonText="button2"  myAction="alert('alert button2');" />
  <myButton transform="translate(100,300)" style="fill: green;"
buttonText="button3" myAction="alert('alert button3');" />
  <myButton transform="translate(300,300)" style="fill: yellow;"
buttonText="Google" myAction="window.open('http://www.google.com')" />
</svg>
```

Pav. 33 „testXBL.xml“

Failas „testXBL-bindings.css“:

```
myButton
{
      -moz-binding: url("library.xml#myButton");
      stroke-width: 0.5;
      stroke: black;
}

myButton > *:hover
{
      stroke-width: 2;
}
```

Pav. 34 „testXBL-bindings.css“

Failas „library.xml“:

```
<?xml version="1.0"?>
<?xml-stylesheet href="testXBL-bindings.css" type="text/css"?>

<bindings xmlns="http://www.mozilla.org/xbl"
          xmlns:xbl="http://www.mozilla.org/xbl"
          xmlns:svg="http://www.w3.org/2000/svg">
  <binding id="myButton" extends="svg:generic">
    <content>
      <svg:g xbl:inherits="onclick=myAction, transform">
        <svg:rect x="0" y="0" width="100" height="50" xbl:inherits="style"/>
        <svg:text x="50" y="25" xbl:inherits="xbl:text=buttonText"
style="stroke-width: 0;" text-anchor="middle"/>
      </svg:g>
    </content>
    <implementation>
      <method name="mDown">
        <body>
          <![CDATA[

document.getAnonymousNodes(this)[0].childNodes[1].setAttribute("rx", "5");

document.getAnonymousNodes(this)[0].childNodes[1].setAttribute("ry", "5");
          ]]>
        </body>
      </method>
      <method name="mUp">
        <body>
          <![CDATA[

document.getAnonymousNodes(this)[0].childNodes[1].setAttribute("rx", "0");

document.getAnonymousNodes(this)[0].childNodes[1].setAttribute("ry", "0");
          ]]>
        </body>
      </method>
    </implementation>
    <handlers>
      <handler event="mousedown">
        <![CDATA[
           this.mDown();
        ]]>
      </handler>
      <handler event="mouseup">
        <![CDATA[
           this.mUp();
```

```
        ]]>
      </handler>
    </handlers>
  </binding>
</bindings>
```

Pav. 35 „library.xml“

Pavyzdyje parodomi keturi mygtukai („*myButton*“). Kiekvienas iš mygtukų turi parametrus, kurie nurodomi per atributus:

- *transform* – mygtuko padėtis;
- *style* – mygtukui priskirtas stilius (kiekvienam mygtukui galima nurodyti savo stilių);
- *buttonText* – mygtuko užrašas;
- *myAction* – mygtuko atliekamas veiksmas (JavaScript kodas).

Naudotojo sukurtas mygtuko tipas siejasi su mygtuko realizacija per CSS failą („testXBL-binding.css“). CSS faile dar nurodyti papildomi mygtuko stiliai („*stroke-width*“, „*stroke*“), kurie yra bendri visiems mygtuko objektams.

Mygtuko realizacija yra faile „library.xml“. Kiekvienas naudotojo sukuriamas elemento tipas aprašomas elementu „*binding*“. Elemento atributas „*id*“ nusako naujo tipo pavadinimą, kuris naudojamas kuriant dokumentą *(„html“*, „*svg*“ ir t.t.). Elementas „*binding*“ gali turėti vaikinius elementus, kurie aprašo atitinkamas kuriamo objekto dalis.

Sekcijoje „*content*“ aprašoma elemento išvaizda. Su atributu „*xbl:inherits*“ galima prijungti kuriamo elemento atributus prie atitinkamų elementų-konstruktorių atributų. Pavyzdžiui, prie elemento „*svg:g*“ prijungtas iš elemento „*myButton*“ atributas *transform* (nekeičiant pavadinimo) ir atributas „*myAction*“ (pakeitus pavadinamą iš „*myAction*“ į „*onclick*“).

Sekcijoje „*implementation*“ realizuojama elemento logika. Pavyzdyje aprašyti du metodai – „*mDown*“ (iškviečiamas, kai elementas paspaudžiamas pelės mygtuku) ir „*mUp*“ (iškviečiamas, kai atleidžiamas pelės mygtukas). Metodų darbas – kai pelės mygtukas yra paspaustas pakeisti stačius kampus į apvalius, o kai pelės mygtukas atleidžiamas – atstatyti buvusią būseną. Tai daroma atitinkamai keičiant „*svg:rect*“ elemento atributus „*rx*“ ir „*ry*“.

Sekcijoje „*handlers*“ aprašomi įvykiai. Kai pelės mygtukas paspaudžiamas, ateina įvykis „*mousedown*“, kuris iškviečia metodą (JavaScript funkciją) „*mDown*“. Kai pelės mygtukas atleidžiamas, ateina įvykis, „*mouseup*“, kuris iškviečia metodą (JavaScript funkciją) „*mUp*“.

3.2.2 Metodo pritaikymas grafiniam redaktoriui

Kadangi šis metodas ženkliai skiriasi nuo visų kitų darbe aprašytų metodų, šiam metodui nepavyko pakartotinai panaudoti redaktoriaus branduolį be pakeitimų. Visi branduolyje esantys pakeitimai buvo susiję su techniniu realizavimu, o veikimo principai ir algoritmai liko nepakitę.

Kuriant mygtuko elementą, visi jam reikalingi parametrai buvo perduodami atitinkamiems SVG elementams per paveldėjimo mechanizmą.

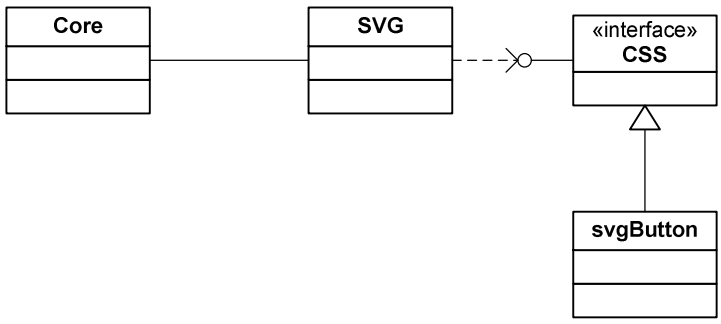
Vienas iš pagrindinių pakeitimų, padarytų branduolyje, buvo tas, kad vietoj jau neveikiančios funkcijos *document.getElementById* teko panaudoti kitą funkciją – *getButton*, kuri atlieka tą patį funkcionalumą. Tai susiję su tuo, kad XBL yra paprastas *xml* failas ir atributas *id* paprastame *xml* faile yra tiesiog eilinis atributas, o ne identifikatorius. Todėl, norint patikrinti elementą pagal jo *id*, tenka naudoti kitus metodus. Vienas jų parodytas žemiau:

```
function getButton(id)
{
    var el=document.getElementsByTagName("svgButton");
    for(i=0; i<el.length; i++)
    {
        if(el[i].getAttribute("id")==id)
        {
            return el[i];
        }
    }
}
```

Pav. 36 Funkcijos "document.getElementById" analogas

Kitas svarbus branduolio pakeitimas susijęs su paveikslėlio atidarymo funkcija. Deja, dabar jau negalime panaudoti *innerHTML* funkcionalumo – tenka kiekvieną elementą apdoroti atskirai ir „piešti“ analogiškai, kaip tai darytų grafinio redaktoriaus naudotojas.

Pritaikius šį metodą, vektorinės grafikos redaktoriaus komponentai ir jų sąsaja tarpusavyje atrodo taip:



Pav. 37 Grafinio redaktoriaus struktūra

3.2.3 Metodo aptarimas

Metodo realizavimui reikalingos ne tik geros SVG, JavaScript bei CSS žinios, bet ir kitų sričių išmanymas. Toks aplikacijų taikymas numato, kad aplikacija yra labai didelė, sudėtinga, sudaryta iš įvairių komponentų, kuriuos galima sudėti į modulius, panaudoti kituose projektuose. Kaip jau buvo minėta, šį metodą kol kas palaiko tik vienas gamintojas – Mozilla.

Šis metodas, kaip ir aprašytas anksčiau, turi gerų ir blogų savybių, ir kurias reikia įvertinti sprendžiant, kokį metodą pasirinkti savo projektui realizuoti. Pagrindiniai metodo privalumai:

- pakartotinis kodo panaudojimas;
- be elementų išvaizdos galima realizuoti ir jų veikimo logiką;
- elementų kūrimo darbas yra įprastas (elementai aprašomi žymėmis, o ne kodu), palyginus su JavaScript variantu.

Metodo trūkumai:

- palaikomas tik Mozillos produktais;
- nėra įrankių, palengvinančių elementų kūrimą.

3.3 SVG ir elementai symbol/use

Šis variantas yra labai panašus į jau aprašytą vektorinės grafikos redaktoriaus kūrimo variantą. Pagrindinis skirtumas yra tas, kad papildomai naudojamos SVG žymės *„symbol“* ir *„use“*, su kuriomis galima sudaryti statinės grafikos šabloną ir vėliau pakartotiniai panaudoti tą šabloną tiek kartų, kiek reikia. Kuriant šabloną su žyme *„symbol“*, sudaromas statinės grafikos paveikslėlis, kuris kol kas niekur neatvaizduojamas. Tam, kad būtų galima pavaizduoti sukurtą šabloną reikalingoje vietoje, naudojama SVG žymė *„use“*. Atvaizdavimo metu žymė *„use“* pakeičiama žyme *„g“* ir kartu paveldi žymės *„use“* aprašytus atributus, išskyrus specifinius.

3.3.1 Realizavimo pavyzdys

Žemiau parodytas pavyzdys, kaip, panaudojus šį metodą, galima aprašyti savo grafinį elementą – mygtuką.

```
<svg width="450" height="300" xmlns="http://www.w3.org/2000/svg"
xmlns:xlink="http://www.w3.org/1999/xlink" >
  <defs>
    <symbol id="myButtonTemplate" overflow="visible">
      <rect x="3" y="3" rx="5" ry="5" width="60" height="20" fill="grey" />
```

```
      <rect x="0" y="0" rx="5" ry="5" width="60" height="20" stroke="black"
fill="red"/>
    </symbol>
  </defs>
  <g id="staedteRect">
    <g onclick="javascript:alert('Button1');" transform="translate(10, 10)">
      <use xlink:href="#myButtonTemplate" />
      <text x="5" y="15" fill="black">Button1</text>
    </g>
    <g onclick="javascript:alert('Button2');" transform="translate(150,
100)">
      <use xlink:href="#myButtonTemplate" />
      <text x="5" y="15" fill="black">Button2</text>
    </g>
    <g onclick="javascript:alert('Button3');" transform="translate(10, 90)">
      <use xlink:href="#myButtonTemplate" />
      <text x="5" y="15" fill="black">Button3</text>
    </g>
  </g>
</svg>
```

Pav. 38 Mygtuko realizavimas

Apibrėžimo srityje (sekcija <*defs*>) aprašomas naujas statinės grafikos tipas (elementas „*symbol*“) vardu „*myButtonTemplate*“. Tai yra mygtuko išvaizdos dalis, kuri nekinta. Kintama dalis (pvz., mygtuko tekstas) aprašoma žemiau. Mygtuko šablonas sudarytas iš mygtuko šešėlio (pirmas elementas „*rect*“) ir paties mygtuko (antras elementas „*rect*“).

Pavyzdyje sukurti tris mygtukai. Kadangi mygtukas turi kintamų atributų (pavyzdžiui, mygtuko pavadinimas, vieta, atliekama funkcija), tai visus kintamus elementus reikia aprašyti atskirai: elementas „*g*“ sugrupuoja visus mygtuko elementus į vieną grupę, nustato mygtuko padėtį (atributas „*transform*“), aprašo atliekamą veiksmą (atributas „*onclick*“); elementas „*use*“ aprašo mygtuko nekintančią išvaizdą, šabloną; elementas „*text*“ nustato mygtuko tekstą.

3.3.2 Metodo pritaikymas grafiniam redaktoriui

Kaip jau buvo pasakyta anksčiau, šis metodas mažai kuo skiriasi nuo pirmo metodo, išskyrus tai, kad panaudojus šį metodą galime išlošti kurdami statinę grafiką, t.y. galime pakartotinai panaudoti statinę grafiką, apibrėžus pastarąją tik vieną kartą. Tam, kad būtų galima pajusti šio metodo naudą buvo pakeista redaktoriaus mygtukų išvaizda – prisidėjo papildomi vizualizacijos efektai (šešėliai, spalvų gradientai ir pan.):

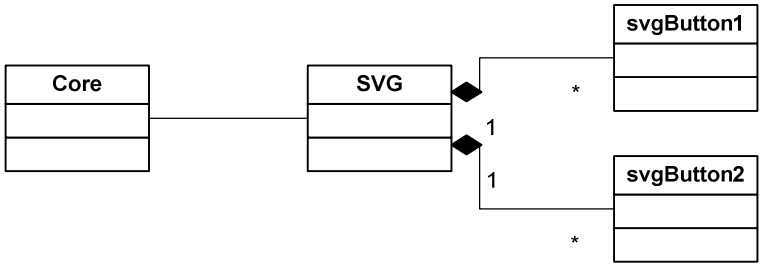
```
<defs>
  <linearGradient id = "g1" x1 = "0%" y1 = "0%" x2 = "100%" y2 = "100%">
    <stop stop-color = "White" offset = "0%" stop-opacity="0"/>
    <stop stop-color = "White" offset = "30%" stop-opacity="0"/>
    <stop stop-color = "Black" offset = "100%" stop-opacity="1"/>
  </linearGradient>
```

```
  <symbol id="svgButton1" overflow="visible">
    <rect x="5" y="5" width="50" height="20" fill="lightgray"
stroke="lightgray"/>
    <rect x="0" y="0" width="50" height="20" stroke="black"/>
    <rect x="0" y="0" width="50" height="20" fill="url(#g1)"/>
  </symbol>
  <symbol id="svgButton2" overflow="visible">
    <rect x="5" y="5" width="40" height="40" fill="lightgray"
stroke="lightgray"/>
    <rect x="0" y="0" width="40" height="40" stroke="black" />
    <rect x="0" y="0" width="40" height="40" fill="url(#g1)"/>
  </symbol>
</defs>
```

Pav. 39 Instrumentų (svgButton1) ir spalvų (svgButton2) mygtukų šablonai

Dabar kiekvieną kartą kuriant mygtuko vaizdą, užtenka vienos eilutės iškviesti mygtuko šabloną, vietoj trijų eilučių, kurie sukuria mygtuko išvaizdą.

Pritaikius šį metodą, vektorinės grafikos redaktoriaus komponentai ir jų sąsaja tarpusavyje atrodo taip:



Pav. 40 Grafinio redaktoriaus struktūra

3.3.3 Metodo aptarimas

Šį metodą realizuoti yra gana paprasta. Esant dideliam statinės grafikos skaičiui jis struktūrizuoja kodą, palengvina ir pagreitina SVG grafikos kūrimą. Kadangi šis metodas mažai skiriasi nuo vektorinės grafikos redaktoriaus kūrimo metodo, galima sakyti, kad tai yra žemo programavimo arba kūrimo lygio metodas.

Kadangi šis metodas yra labai panašus į jau aprašytą metodą, tai metodo stipriosios ir silpnosios vietas yra panašios, tačiau nėra vienodos:

Metodo privalumai:

- pakankamai greitas metodas mažų aplikacijų kūrimo atžvilgiu (nebereikia kurti bibliotekų);
- kadangi naudoja tik standartinius SVG primityvus, yra palaikomas daugumos SVG grafiką mokančių parodyti grotuvų;

- produktas, sukurtas taikant šį metodą, atitiks W3C standartus;
- galima pakartotinai panaudoti kodą paprastiems elementams kurti (kur yra statinė grafika).

Metodo trūkumai:

- tinka tik mažoms aplikacijoms kurti;
- neišeina lengvai pakartotinai panaudoti kodą sudėtingesniems elementams kurti;
- pakankamai sunku derinti aplikaciją.

4 Realizavimo metodų palyginimas

Žemiau pateikiamas aprašytų darbe SVG taikymo žiniatinklyje metodų palyginimas pagal pasirinktus kriterijus:

Kriterijus[1]	**Metodas**			
	SVG panaudojimas su JavaScript ir CSS	**SVG paslėpimas po JavaScript objektais**	**SVG paslėpimas po XBL elementais**	**SVG ir elementai symbol/use**
Realizavimo paprastumas	****	*	**	****
Aplikacijos dydis	*	***	***	**
Pakartotinis kodo panaudojimas	*	**	**	**
Suderinamumas su vaizdavimo įrankiais	**	**	*	**
Atitikimas standartus	W3C	-	Mozilla XBL	W3C
Moduliškumas	*	**	***	*

4.1 Kriterijų paaiškinimai

Realizavimo paprastumas. Parodo, kiek paprasta realizuoti reikalingą grafikos objektą, jo išvaizdą, funkcionalumą. Vertinimo kriterijai:

- **** – reikalingos bendros SVG, JavaScript bei CSS žinios;
- ** – reikalingos ne tik SVG, JavaScript bei CSS žinios, bet ir kitų programavimo sričių išmanymas, pavyzdžiui, XML, XSLT ir pan.;

[1] Kiekvienas kriterijus turi turėti savo koeficientą, kuris nustatomas atsižvelgiant į jo svarbą, poreikus ir t.t.

- * – reikalingos gilios žinios bei praktika.

Aplikacijos dydis. Parodo, kokio dydžio aplikacijoms kurti geriausiai tinka pasirinktas metodas. Vertinimo kriterijai:

- *** – aplikacijoje naudojama virš 50 sąsajos elementų, jų išvaizda bei veikimo logika yra komplikuota;
- ** – aplikacijoje naudojama 10 – 50 sąsajos elementų, jų išvaizda bei veikimo logika yra nesudėtinga;
- * – aplikacija yra maža, naudojama keletas (apie 10) grafinių sąsajos elementų, jų veikimo logika yra paprasta.

Pakartotinis kodo panaudojimas. Metodas leidžia pakartotinai panaudoti kodą, kurti elementų bibliotekas ir panašiai. Vertinimo kriterijai:

- ** – metodas pritaikytas pakartotiniam kodo panaudojimui;
- * – metodas nepritaikytas pakartotiniam kodo panaudojimui.

Suderinamumas su vaizdavimo įrankiais. Parodo, kiek daug vaizdavimo įrankių gali vykdyti aplikaciją, sukurtą pasirinktu metodu. Vertinimo kriterijai:

- ** – dauguma plačiai naudojamų įrankių gali vykdyti aplikaciją;
- * – tik keli vaizdavimo įrankiai gali vykdyti aplikaciją.

Atitikimas standartus. Parodo, kokius (vaizdavimo, grafikos) standartus atitiks pasirinktu metodu sukurta aplikacija. Vertinimo kriterijai:

- standarto pavadinimas.

Moduliškumas. Parodo, ar metodas leidžia atskirti duomenų, verslo, prezentacijos lygius į atskirus modulius su aiškiomis sąsajomis. Vertinimo kriterijai:

- *** – metodas numato tokį skaidymą į modulius pagal savo prigimtį;
- ** – metodą galima naudoti tiek su skaidymu į modulius (iš principo tai galima realizuoti gana nesunkiai), tiek ir be jo;
- * – metodas nepritaikytas skaidymui į modulius, t.y. realizuoti tokį atskyrimą yra labai sunku.

5 Darbo rezultatai

Darbe buvo nagrinėjamas vektorinės grafikos formatas – SVG, jo panaudojimas kuriant žiniatinklio aplikacijas. Darbo pradžioje buvo apžvelgta, kokiais būdais galima integruoti SVG į žiniatinklio turinį, taip pat trumpai apžvelgtos kitokio formato (Flash) galimybės. Buvo pasiūlyti ir išnagrinėti 4 SVG taikymo žiniatinklyje variantai arba metodai:

- SVG panaudojimas su JavaScript ir CSS;
- SVG paslėpimas po JavaScript objektais;
- SVG paslėpimas po XBL elementais;
- SVG ir elementai symbol/use.

Buvo realizuotas pilnai veikiantis SVG taikymo pavyzdys – vektorinės grafikos redaktorius. Grafikos redaktorius realizuotas visais darbe aprašytais metodais. Kiekvienam metodui parodyta, kaip su juo galima realizuoti reikalingą elementą (mygtuką), pateikti metodo privalumai bei trūkumai, detaliau aptartas pirmas metodas (redaktoriaus veikimo principai, algoritmai, kurie visuose nagrinėjamuose metoduose nesikeitė).

Darbo pabaigoje pateiktas metodų palyginimas pagal pasirinktus kriterijus. Iš palyginimo rezultatų galima daryti išvadas, kokį metodą, kada ir kokiomis sąlygomis geriau naudoti norint įgyvendinti savo tikslus.

Išvados

Atlikus darbą, paaiškėjo, kad vieną iš vektorinės grafikos formatų – SVG – iš principo galima taikyti žiniatinklyje. Esant poreikiui, jau dabar galima naudoti SVG žiniatinklio aplikacijoms kurti. SVG panaudojimui galima taikyti skirtingus metodus arba variantus priklausomai nuo uždavinio sudėtingumo, poreikių, resursų ir kitų reikalavimų:

- „SVG panaudojimas su JavaScript ir CSS“ – šis variantas tinka tik labai mažoms aplikacijoms kurti, neturi jokių palengvinančių programavimo technikų (pavyzdžiui, pakartotinio kodo panaudojimo, aukšto lygio konstrukcijų ir pan.);
- „SVG paslėpimas po JavaScript objektais“ – šis metodas paremtas tik JavaScript panaudojimu (vektorinės grafikos elementai konstruojami dinamiškai). Metodas yra geras tuo, kad vieną kartą sukūrus reikalingų elementų biblioteką (šiam etapui paprastai reikia gerų JavaScript programavimo įgūdžių), ją galima naudoti daug kartų skirtinguose projektuose;
- „SVG paslėpimas po XBL elementais“ – šis metodas tinka kurti aplikacijas tik Mozilla produktams (pavyzdžiui, Mozilla Firefox), tačiau duoda aukšto lygio konstrukcijas, moduliškumą ir pan.;
- „SVG ir elementai symbol/use“ – šis metodas, galima sakyti, yra pirmo metodo praplėtimas, kuris kartais (pavyzdžiui, kai naudojama daug statinės vektorinės grafikos) gali būti pranašesnis už pirmą metodą.

Atliekant darbą, o ypač ruošiant pavyzdžius, paaiškėjo, kad SVG yra pakankamai naujas reiškinys žiniatinklyje. Nors yra standartai, demonstraciniai pavyzdžiai, tačiau mažai kas praktiškai naudoja SVG savo projektuose, mažai yra įrankių, palengvinančių kūrimą, mažai yra bibliotekų su naudingu funkcionalumu ir t.t.

Iš kitos pusės, SVG pagal savo galimybes ir funkcionalumą jau dabar gali laisvai konkuruoti su kitomis žiniatinklyje naudojamomis technologijomis. Kartais gali būti net tokių atvejų, kur su SVG galima lengvai padaryti tai, kas neišeina su kitomis technologijomis.

Visa tai reiškia, kad SVG dar yra naujas, plačiai neištirtas dalykas, kuris iš principo gali tapti gera priemone kurti žiniatinklio aplikacijas.

Literatūros sąrašas

[Pe00] Chengyuan Peng. SCALABLE VECTOR GRAPHICS (SVG). Research Seminar on Interactive Digital Media, 2000

[TC03] J. Teague, Marc Campbell. SVG for Web Designers. Wiley; 1st edition, 2003, 312 pages

[PH02] Ellen Pearlman, Lorien House. Developing SVG-based Web Applications, Prentice Hall PTR; 1st edition, 2002, 464 pages

[Cag02] Kurt Cagle. SVG Programming: The Graphical Web, Apress; New edition, 2002, 586 pages

[NW09] Andreas Neumann, Andréas M. Winter. Vector-based Web Cartography: Enabler SVG. URL: http://www.carto.net/papers/svg/index_e.shtml. 14KB, 2009.01.06

[BCS09] Bit Component Set. URL: http://www.flashloaded.com/flashcomponents/bitcomponentset/. 86KB, 2009.01.02

[CA09] ComponentArt. Web UI. URL: http://www.componentart.com/webui/. 228KB, 2009.01.02

[XW09] XHTML2 Working Group Home Page. URL: http://www.w3.org/MarkUp/. 12KB, 2009.01.02

[MS09] MSDN. Windows Forms. URL: http://msdn.microsoft.com/en-us/netframework/aa497342.aspx. 45KB, 2009.01.02

[ASS09] All SVG Samples. URL: http://www.croczilla.com/svg/samples. 81KB, 2009.01.02

[XBL09a] XBL 1.0 Reference. URL: https://developer.mozilla.org/en/XBL/XBL_1.0_Reference. 6KB, 2009.01.02

[XBL09b] XBL 2.0. URL: http://www.mozilla.org/projects/xbl/xbl2.html. 7KB, 2009.01.02

[DS09] Document Structure. URL: http://www.w3.org/TR/SVG/struct.html. 14KB, 2009.01.02

[cod09] codedread. URL: http://www.codedread.com/svg-support.php. 9KB, 2009.01.02

[SVG09a] SVG checkBox and radioButtonGroup Object. URL: http://www.carto.net/papers/svg/gui/checkbox_and_radiobutton/. 6KB, 2009.01.02

[Ga09] Steven Garrity. How to Include Scalable Vector Graphics (SVG) In-line. URL: http://labs.silverorange.com/archives/2006/january/howtoinclude. 21KB, 2009.01.02

[AF09] Adobe Flash. URL: http://en.wikipedia.org/wiki/Adobe_Flash. 36KB, 2009.03.21

[SVG09b] Scalable Vector Graphisc. URL: http://en.wikipedia.org/wiki/Svg. 34KB, 2009.03.21

[Ado09] Adobe. URL: http://www.adobe.com/. 34KB, 2009.03.21

[BS09] Browser Statistics. URL: http://www.w3schools.com/browsers/browsers_stats.asp. 26KB, 2009.03.21

[De09] Steve Dekorte. SVG and Flash.URL: http://patricklogan.blogspot.com/2008/06/svg-and-flash.html. 4KB, 2009.04.06

[Ka09] Ч.А. Кариев. Масштабируемая векторная графика (Scalable Vector Graphics). URL: http://www.intuit.ru/department/graphics/svg/1/2.html. 36KB, 2009.03.21

[Em09] Embeded SVG. URL: http://esvg.ultimodule.com/bin/esvg/templates/splash.asp?NC=6870X. 54KB, 2009.04.06

[SVG09c] SVG Tetris. URL: http://www.croczilla.com/svg/samples/svgtetris. 8KB, 2009.04.06

[OCK09] Robert O'Callahan, Christian, Repatriate Kiwi. Applying SVG Effects To HTML Content. URL: http://weblogs.mozillazine.org/roc/archives/2008/06/applying_svg_ef.html. 11KB, 2009.04.06

[GS09] Gemeinde Schlatterbach. URL: http://www.uismedia.de/mapview/beispiele/schlatterbach/index.html. 42KB, 2009.04.06

[YHM09] Yosemite Hiking Map. URL: http://www.carto.net/williams/yosemite/. 57KB, 2009.04.06

[PIE09] Pilat Informative Educative: SVG - PHP - MySQL – JavaScript. URL: http://pilat.free.fr/english/. 8KB, 2009.04.06

[MSP09] Mobile SVG Profiles: SVG Tiny and SVG Basic. URL: http://www.w3.org/TR/SVGMobile/. 6KB, 2009.04.27

[MH01] W. Scott Means, Elliotte Rusty Harold. XML in a Nutshell. O'Reilly Media, 2001

[NX08] Namespaces in XML 1.0 (Second Edition). URL: http://www.w3.org/TR/REC-xml-names/. 7KB, 2008.04.13

[LV08] Joao Emile Louis, Nandamudi L. Vijaykumar. Visualization of a detailed Rocket Trajectory on a Cartographic maps based on Open Web Standards. URL: http://www.codata.org/04conf/papers/Louis-paper.pdf. 144KB. 2008.04.13

[HS08] HTML 4.01 Specification. URL: http://www.w3.org/TR/html401/. 8KB, 2008.04.13

[W3C08] W3C Scalable Vector Graphics (SVG) Test Suite Overview. URL: http://www.w3.org/Graphics/SVG/Test/. 7KB, 2008.04.13

[MSS08] Mozilla SVG Status. URL: http://www.mozilla.org/projects/svg/status.html. 15KB, 2008.04.13

[SSO08] SVG support in Opera 9. URL: http://www.opera.com/docs/specs/svg/. 16KB, 2008.04.13

[IH08] Introduction to HTML 4. URL: http://www.w3.org/TR/html401/intro/intro.html. 10KB, 2008.04.26

[CSS08] Cascading Style Sheets Home page. URL: http://www.w3.org/Style/CSS/. 43KB, 2008.04.26

[SC08] Styling and CSS. URL: http://www.w3.org/1999/08/WD-SVG-19990812/styling.html. 24KB, 2008.04.26

[SSC08] Styling SVG with CSS. URL: http://www.croczilla.com/~alex/old-site/mini-presentation/styled-svg.xml. 24KB, 2008.04.26

[ASS08] Associating Style Sheets with XML documents. URL: http://www.w3.org/TR/xml-stylesheet/. 22KB, 2008.04.26

[Sty08] Styling. URL: http://www.w3.org/TR/SVG/styling.html. 6KB, 2008.04.26

[Scr08] Scripts. URL: http://www.w3.org/TR/REC-html40/interact/scripts.html. 64KB, 2008.05.04

[Ei08] Horst Eidenberger. SMIL and SVG in teaching. URL: http://www.ims.tuwien.ac.at/media/documents/publications/ei2004-teaching.pdf. 713KB, 2008.05.07

[Ani08a] Animation: SVG and SMIL Animation. URL: http://www2.jasc.com/pub/SVGChpt08.pdf. 826KB, 2008.05.07

[Ani08b] Animation. URL: http://www.w3.org/TR/2000/CR-SVG-20000802/animate.html. 10KB, 2008.05.09

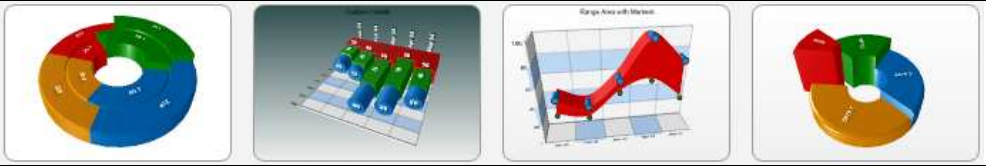
[LSS08] Learning SMIL with a SMIL. URL: http://www.multimedia4everyone.com/elements_sample/elements_sample.html. 39 KB, 2008.05.18

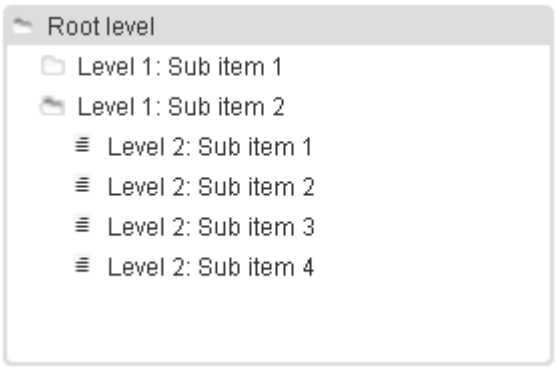
Priedai

Priedas A

Šiame priede pavaizduoti kai kurių naudotojo sąsajos elementų išvaizdos pavyzdžiai.

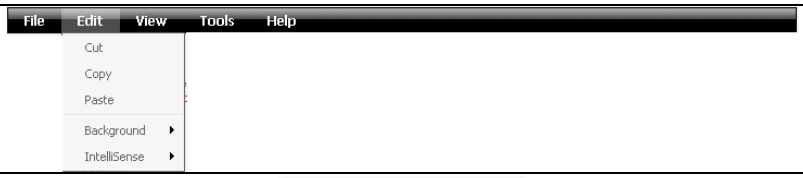
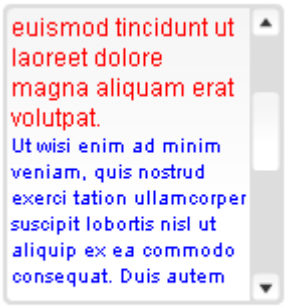
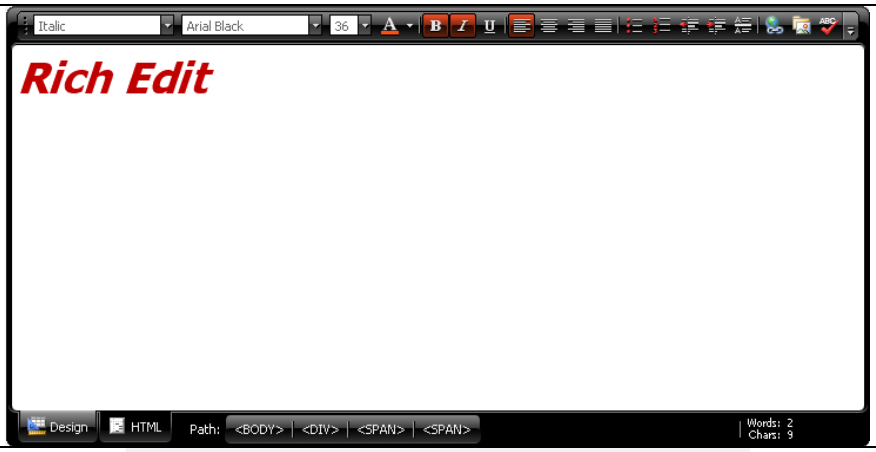
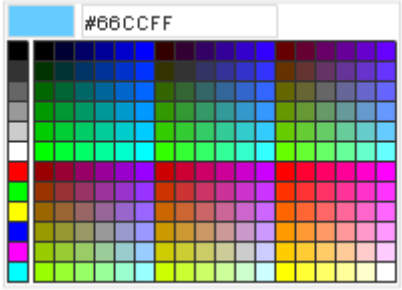
Informacijos pateikimo elementai:

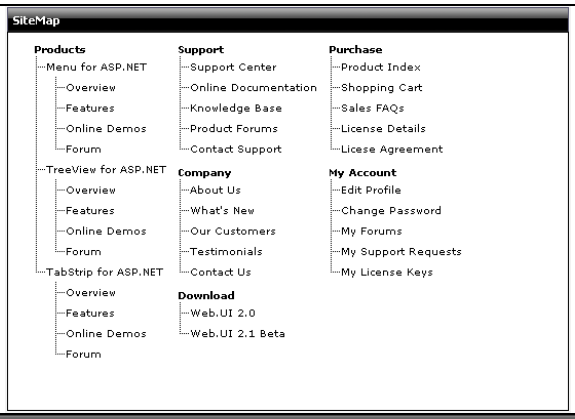
Label 1 Label 2 Label 3 Label 4 Label 5	Label / Text
	Image
	Chart
Another floating tooltip with new style Another floating tooltip with new style	Tooltip
22 %	Progressbar
List item 0 List item 1 List item 2 List item 3 List item 4 List item 5 List item 6 Current: Data of item 3	List

	TreeView

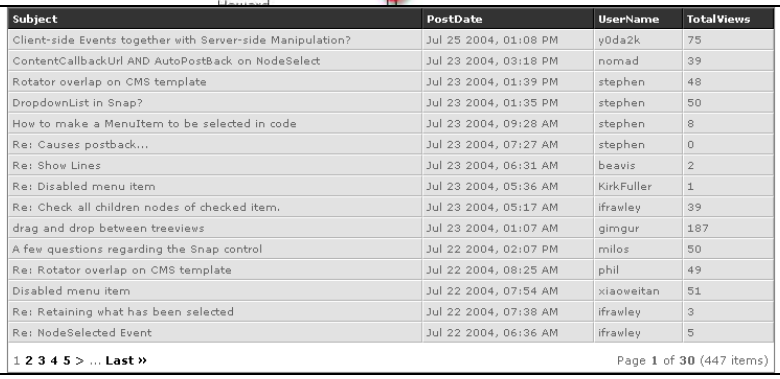
Valdymo elementai:

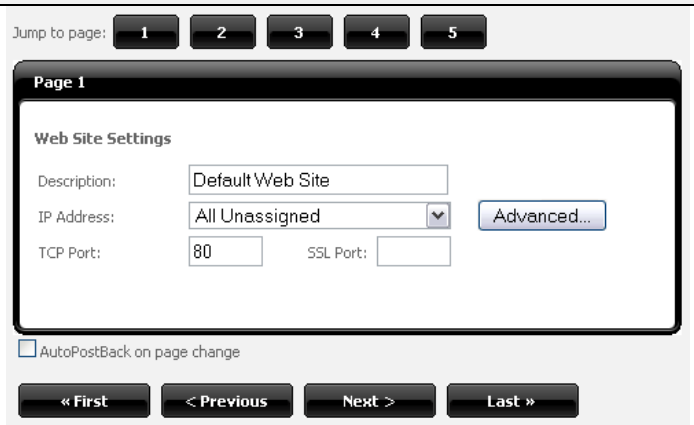
To index	Link
Hello world!	Input
0.036307553 0.036307553 0.021575849 0.921312014 0.332252746 0.627997277	Combobox
Button	Button
Alert Title This is an alert message.	Alert
CheckBox1 CheckBox2 CheckBox3 CheckBox4	Checkbox
RadioButton1 RadioButton2 RadioButton3 RadioButton4	RadioButton
	IconButton

	Menu
	Calendar
 TextArea with CSS styling	Textarea
	RichEdit
	Uploader / loader
	Colorpicker

	Site Map
	Dialog
	Slider
	Player
	Player control

Išdėstymo elementai (konteineriai):

	Spliter
	Grid

	Tabstrip
	Toolbar
	Multipage
	Rotator

Priedas B

Šiame priede pateiktas vektorinės grafikos redaktoriaus kodas (metodas SVG, JavaScript, CSS):
Failas „VGE.xhtml“:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" >
  <head>
    <title>Vector Graphics Editor Demo</title>
  </head>
  <body>
    <h1>Vector Graphics Editor Demo</h1>
    <form name="main" id="main" method="post"
action="http://localhost/vge/default.aspx">
      <input type="hidden" name="svg" value=""/>
      <svg xmlns="http://www.w3.org/2000/svg"
xmlns:xlink="http://www.w3.org/1999/xlink" width="600" height="500"
onload="init()">
        <script xlink:href="VGE.js" />
        <g width="600" heigth="40" name="instruments"
transform="translate(0,0)">
          <rect x="0" y="0" width="600" height="40" fill="white"
stroke="black" onmousemove="setStatus('')" />
          <g onclick="selectShape(1)" transform="translate(10,10)"
onmousemove="setStatus('Draw Line')">
            <rect x="0" y="0" width="50" height="20" fill="yellow"
stroke="black" id="b1"/>
            <line x1="5" y1="10" x2="45" y2="10" stroke="black"/>
```

```
          </g>
          <g onclick="selectShape(2)" transform="translate(70,10)"
onmousemove="setStatus('Draw Ractangle')">
            <rect x="0" y="0" width="50" height="20" fill="yellow"
stroke="black" id="b2"/>
            <rect x="5" y="5" width="40" height="10" fill="blue"
stroke="black"/>
          </g>
          <g onclick="selectShape(3)" transform="translate(130,10)"
onmousemove="setStatus('Draw Ellipse')">
            <rect x="0" y="0" width="50" height="20" fill="yellow"
stroke="black" id="b3"/>
            <ellipse cx="25" cy="10" rx="20" ry="5" fill="blue"
stroke="black"/>
          </g>
          <g onclick="selectShape(4)" transform="translate(190,10)"
onmousemove="setStatus('Draw Text')">
            <rect x="0" y="0" width="50" height="20" fill="yellow"
stroke="black" id="b4"/>
            <text x="20" y="15">A</text>
          </g>
          <g onclick="selectShape(5)" transform="translate(250,10)"
onmousemove="setStatus('Draw Point')">
            <rect x="0" y="0" width="50" height="20" fill="yellow"
stroke="black" id="b5"/>
            <circle cx="25" cy="10" r="1" stroke="black"/>
          </g>
          <g onclick="save()" transform="translate(310,10)"
onmousemove="setStatus('Save Picture')">
            <rect x="0" y="0" width="50" height="20" fill="lightblue"
stroke="black" id="b5"/>
            <text x="10" y="15">Save</text>
          </g>
          <g onclick="openp()" transform="translate(370,10)"
onmousemove="setStatus('Open Last Picture')">
            <rect x="0" y="0" width="50" height="20" fill="lightblue"
stroke="black" id="b5"/>
            <text x="10" y="15">Open</text>
          </g>
        </g>
        <g width="600" height="60" transform="translate(0,40)">
          <rect x="0" y="0" width="600" height="60" fill="white"
stroke="black" onmousemove="setStatus('')"/>
          <rect id="c1" x="0" y="0" width="40" height="40" fill="white"
stroke="black" transform="translate(10,10)" onclick="setColor(this)"
onmousemove="setStatus('White')" />
          <rect id="c2" x="0" y="0" width="40" height="40" fill="black"
stroke="black" transform="translate(60,10)" onclick="setColor(this)"
onmousemove="setStatus('Black')" />
          <rect id="c3" x="0" y="0" width="40" height="40" fill="red"
stroke="black" transform="translate(110,10)" onclick="setColor(this)"
onmousemove="setStatus('Red')" />
          <rect id="c4" x="0" y="0" width="40" height="40" fill="green"
stroke="black" transform="translate(160,10)" onclick="setColor(this)"
onmousemove="setStatus('Green')" />
          <rect id="c5" x="0" y="0" width="40" height="40" fill="blue"
stroke="black" transform="translate(210,10)" onclick="setColor(this)"
onmousemove="setStatus('Blue')" />
          <rect id="c6" x="0" y="0" width="40" height="40" fill="yellow"
stroke="black" transform="translate(260,10)" onclick="setColor(this)"
onmousemove="setStatus('Yellow')" />
          <rect id="c7" x="0" y="0" width="40" height="40" fill="cyan"
stroke="black" transform="translate(310,10)" onclick="setColor(this)"
onmousemove="setStatus('Cyan')" />
```

```
          <rect id="c8" x="0" y="0" width="40" height="40" fill="fuchsia"
stroke="black" transform="translate(360,10)" onclick="setColor(this)"
onmousemove="setStatus('Fuchsia')" />
          <rect id="c9" x="0" y="0" width="40" height="40" fill="gray"
stroke="black" transform="translate(410,10)" onclick="setColor(this)"
onmousemove="setStatus('Gray')" />
        </g>
        <g id="d" width="600" height="300" onmousedown="down()"
onmousemove="move()" onmouseup="up()" transform="translate(0,100)"
cursor="crosshair">
          <rect x="0" y="0" width="600" height="300" fill="white"
stroke="black"/>
        </g>
        <g width="600" heigth="30" name="instruments"
transform="translate(0,400)" onmousemove="setStatus('')">
          <rect x="0" y="0" width="600" height="30" fill="white"
stroke="black"/>
          <text id="StatusText" x="10" y="20">Status</text>
        </g>
      </svg>
      <p id="t"></p>
    </form>
  </body>
</html>
```

Failas „VGE.js“:

```
// JScript source code
document.onmousemove = mmove;
svgNS = "http://www.w3.org/2000/svg";
var mx, my, mxx, myy;
mousest=0;
lastId=0;
shape=0;
color="black";


function mmove(e)
{
    mx=e.pageX-8;
    my=e.pageY-180;

}

function down()
{
    switch(shape)
    {
      case 1: // line
        var el=document.createElementNS(svgNS, "line");
        elid = 'el_'+lastId;
        el.setAttribute('id', elid);
        el.setAttribute('x1', mx);
        el.setAttribute('y1', my);
        el.setAttribute('x2', mx);
        el.setAttribute('y2', my);
        el.setAttribute('stroke', color);
        document.getElementById("d").appendChild(el);
        mousest=1;
      break;
      case 2: // rect
        var el=document.createElementNS(svgNS, "rect");
        elid = 'el_'+lastId;
        el.setAttribute('id', elid);
```

```
        el.setAttribute('x', mx);
        el.setAttribute('y', my);
        el.setAttribute('width', '0');
        el.setAttribute('height', '0');
        el.setAttribute('stroke', color);
        el.setAttribute('fill', color);
        mxx=mx;
        myy=my;
        document.getElementById("d").appendChild(el);
        mousest=1;
      break;
      case 3: // ellipse
        var el=document.createElementNS(svgNS, "ellipse");
        elid = 'el_'+lastId;
        el.setAttribute('id', elid);
        el.setAttribute('cx', mx);
        el.setAttribute('cy', my);
        el.setAttribute('rx', '0');
        el.setAttribute('ry', '0');
        el.setAttribute('stroke', color);
        el.setAttribute('fill', color);
        mxx=mx;
        myy=my;
        document.getElementById("d").appendChild(el);
        mousest=1;
      break;
      case 4: // text
        var text = prompt("Enter text", "");
        if(text != null && text != "")
        {
            var el=document.createElementNS(svgNS, "text");
            elid = 'el_'+lastId;
            el.setAttribute('id', elid);
            el.setAttribute('x', mx);
            el.setAttribute('y', my);
            el.setAttribute('fill', color);
            var el2 = document.createTextNode(text);
            el.appendChild(el2);
            document.getElementById("d").appendChild(el);
            lastId=lastId+1;
        }
      break;
      case 5: // point
        var el=document.createElementNS(svgNS, "circle");
        elid = 'el_'+lastId;
        el.setAttribute('id', elid);
        el.setAttribute('cx', mx);
        el.setAttribute('cy', my);
        el.setAttribute('r', '1');
        el.setAttribute('stroke', color);
        el.setAttribute('fill', color);
        document.getElementById("d").appendChild(el);
        mousest=1;
      break;
    }//*/


}

function move()
{
    if(mousest==1)
    {
      switch(shape)
```

```
      {
        case 1: // line
          var el=document.getElementById('el_'+lastId);
          el.setAttribute('x2', mx);
          el.setAttribute('y2', my);
        break;
        case 2: //rect
          var el=document.getElementById('el_'+lastId);
          el.setAttribute('width', mx-mxx);
          el.setAttribute('height', my-myy);
        break;
        case 3: //ellipse
          var el=document.getElementById('el_'+lastId);
          el.setAttribute('rx', mx-mxx);
          el.setAttribute('ry', my-myy);
        break;
      }
    }
    document.getElementById("StatusText").firstChild.nodeValue = "X=" + mx +
"; Y=" + my;
}

function up()
{
    if(mousest==1)
    {
        lastId=lastId+1;
        mousest=0;
    }
}

function selectShape(id)
{
    shape=id;
    var el;
    var i;
    for(i=1; i<6; i++)
    {
      el=document.getElementById("b"+i);
      if(i==id)
      {
        el.setAttribute("fill", "yellow");
      }else
      {
        el.setAttribute("fill", "green");
      }
    }
}

function save()
{
    var i;
    var s="<?xml version='1.0' encoding='UTF-8'?><svg
xmlns='http://www.w3.org/2000/svg'>";
    for (i=0; i<lastId; i++)
    {
        var id = "el_"+i;
        var el = document.getElementById(id);

        var name = el.nodeName;
        var elem="";
        switch(name)
        {
            case "line":
```

```
            elem='<line id="'+id+'" x1="'+el.getAttribute("x1")+'"
y1="'+el.getAttribute("y1")+'" x2="'+el.getAttribute("x2")+'"
y2="'+el.getAttribute("y2")+'" stroke="'+el.getAttribute("stroke")+'" />';
            break;
            case "rect":
            elem='<rect id="'+id+'" x="'+el.getAttribute("x")+'"
y="'+el.getAttribute("y")+'" width="'+el.getAttribute("width")+'"
height="'+el.getAttribute("height")+'" stroke="'+el.getAttribute("stroke")+'"
fill="'+el.getAttribute("fill")+'" />';
            break;
            case "ellipse":
            elem='<ellipse id="'+id+'" cx="'+el.getAttribute("cx")+'"
cy="'+el.getAttribute("cy")+'" rx="'+el.getAttribute("rx")+'"
ry="'+el.getAttribute("ry")+'" stroke="'+el.getAttribute("stroke")+'"
fill="'+el.getAttribute("fill")+'" />';
            break;
            case "text":
            elem='<text id="'+id+'" x="'+el.getAttribute("x")+'"
y="'+el.getAttribute("y")+'"
fill="'+el.getAttribute("fill")+'">'+el.firstChild.nodeValue+'</text>';
            break;
            case "circle":
            elem='<circle id="'+id+'" cx="'+el.getAttribute("cx")+'"
cy="'+el.getAttribute("cy")+'" r="'+el.getAttribute("r")+'"
stroke="'+el.getAttribute("stroke")+'" fill="'+el.getAttribute("fill")+'"
/>';
            break;
        }
        s=s+elem;
    }
    s=s+"</svg>";
    document.forms['main'].svg.value=s;
    document.forms['main'].submit();
}

function openp()
{
    var xmlHttp;
    xmlHttp=new XMLHttpRequest();


    xmlHttp.onreadystatechange=function()
    {
        if(xmlHttp.readyState==4)
        {
            document.getElementById("d").innerHTML=xmlHttp.responseText;
        }
    }
    xmlHttp.open("GET","http://127.0.0.1/vge/default.aspx",true);
    xmlHttp.send(null);
}

function init()
{
    selectShape(1);
    document.getElementById("c2").setAttribute("stroke-width", "5");
}

function setStatus(text)
{
    document.getElementById("StatusText").firstChild.nodeValue = text;
}

function setColor(c)
```

```
{
    var i;
    col = c.getAttribute("fill");
    color = col;
    for(i=1; i<=9; i++)
    {
      el=document.getElementById("c"+i);
      if(el.getAttribute("fill")==col)
      {
        el.setAttribute("stroke-width", "5");
      }else
      {
        el.setAttribute("stroke-width", "1");
      }
    }
}
```