VILNIAUS GEDIMINO TECHNIKOS UNIVERSITETAS

FUNDAMENTINIŲ MOKSLŲ FAKULTETAS

GRAFINIŲ SISTEMŲ KATEDRA

Imantas Vasilavičius

Internetinių svetainių kūrimo technologijos

TECHNOLOGIES FOR WEB SITE DEVELOPMENT

Baigiamasis magistro darbas

Informacinių technologijų studijų programa, valstybinis kodas 62407T104

Erdvinių informacinių sistemų specializacija

Informatikos inžinerijos kryptis

Vilnius, 2009

VILNIAUS GEDIMINO TECHNIKOS UNIVERSITETAS

FUNDAMENTINIŲ MOKSLŲ FAKULTETAS

GRAFINIŲ SISTEMŲ KATEDRA

TVIRTINU
Katedros
vedėjas

_____
(parašas)
Romualdas
Baušys
(Vardas,
pavardė)

_____
(Data)

Imantas Vasilavičius

Internetinių svetainių kūrimo technologijos

TECHNOLOGIES FOR WEB SITE DEVELOPMENT

Baigiamasis magistro darbas

Informacinių technologijų studijų programa, valstybinis kodas 62407T104
Erdvinių informacinių sistemų specializacija
Informatikos inžinerijos kryptis

Vadovas prof. habil. dr. Romualdas Baušys _____ _____
(Moksl. laipsnis, vardas, pavardė) (Parašas) (Data)

Konsultantas doc. dr. Angelė Kaulakiene _____ _____
(Moksl. laipsnis, vardas, pavardė) (Parašas) (Data)

Vilnius, 2009

Užduotis

1. Išanalizuoti internetinių svetainių kūrimo technologijas.
2. Nustatyti perspektyviausią technologiją.
3. Pasiūlyti geriausią architektūros stilių, nustatyti jo savybes bei apribojimus ir palyginti su kitais stiliais.
4. Patobulinti pasirinktą architektūros stilių.
5. Sukurti internetinę svetainę, išryškinančią darbe analizuotas savybes ir praktiškai parodančią AJAX galimybes.

Vilniaus Gedimino technikos universitetas
Fundamentiniu mokslu fakultetas
Grafinių sistemų katedra

Informacinių technologijų studijų programos baigiamasis magistro darbas

Internetinių svetainių kūrimo technologijos

Autorius: Imantas Vasilavičius
Vadovas: Romualdas Baušys
Kalba - lietuviu

Anotacija

Baigiamajame magistro darbe nagrinėjama AJAX technologija. Ši technologija pasitelkiama būsenos neturinčių svetainių prastam interaktyvumui spręsti. Šio naujo metodo esmė yra vieno puslapio sąveikos modelis, kuriuo galima lengviau didinti interaktyvumą. Pristatytas ir patobulintas SPIAR architektūros stilius. Aprašyti naujos programinės įrangos inžineriniai principai ir parinkti apribojimai, suteikiantys pageidaujamas savybes. Stilius apibrėžia naudotojo sąsajos komponentų kūrimą ir tarpinius delta ryšius tarp kliento/serverio komponentų, taip pat pagerina naudotojo interaktyvumą ir palengvina kūrimą. Sukurta internetinė svetainė, išryškinanti geriausias AJAX savybes. Darbo apimtis – 62 puslapiai teksto be priedų, 17 paveikslų, 5 lentelės, 30 bibliografinių šaltinių.

Raktiniai žodžiai: AJAX, SPIAR, COMET, internetinė svetainė, architektūros stilius, interaktyvumas.

Vilnius Gediminas Technical University
Faculty of Fundamental Sciences
Department of Graphical Systems

The final work of Information Technologies MSc. program

Technologies for web site development

Author: Imanas Vasilavičius
Academic supervisor: Romualdas Baušys
Thesis language - Lithuanian

Annotation

In this master thesis is investigating AJAX technology. This technology is emerging in response to a limited degree of interactivity in stateless Web interactions. At the heart of this new approach lies a single page interaction model that facilitates rich interactivity. Was introduced and improved the SPIAR architectural style. Guiding software engineering principles was described and the constraints chosen to induce the desired properties. The style emphasizes user interface component development, and intermediary delta-communication between client/server components, to improve user interactivity and ease of development. A web site was created to emphasis some of the best AJAX properties. Work length – 62 pages without enclosures, 17 images, 5 table, 30 bibliographic sources.

Keywords: AJAX, SPIAR, COMET, web site, architectural style, interactivity.

Turinys

Paveikslų sąrašas

Lentelių sąrašas

Santrumpos

AJAX – asinchroninis *JavaScript* ir XML
XML – išplėstinė žymėjimo kalba
DOM – dokumento objekto modelis
HTML – hiperteksto žymėjimo kalba
CSS – kaskadinės stilių lentelės
UI – naudotojo sąsaja
GWT – *Google* kompanijos karkasas
REST – atvaizdavimo būsenos persiuntimas
HTTP – giperteksto persiuntimo protokolas
SPIAR – architektūros stilius
RPC – nuotolinis procedūros iškvietimas
URI – vienodų išteklių identifikatorius
JSP – java serverio puslapio technologija
ID – identifikatorius
BSD – berklio universiteto programine įranga
IO – įvestis/išvestis
API – programų kūrimo sąsaja
GUI – grafinė naudotojo sąsaja

ĮVADAS

Daug metų tinklo svetainių kūrėjai susidurdavo su viso puslapio persiuntimo problema kiekvieną kartą, kai jie norėdavo gauti naujų duomenų iš serverio arba nusiųsti į serverį. Modernios svetainės turi būti dinamiškos ir interaktyvios. Visas šias problemas išsprendžia AJAX technologija. AJAX arba asinchroninis *JavaScript* ir XML yra svetainių programavimo technologija, naudojanti šias priemones didžiausiam interaktyvumui pasiekti: XHTML, stilių lenteles, dokumento objektinį modelį (toliau – DOM), *JavaScript* kalbą dinamiškam vaizdavimui bei interaktyvumui, XML ir *XMLHttpRequest* objektą asinchroniniams duomenų mainams su serveriu.

Visoms geriausioms AJAX technologijos savybėms pasiekti reikalingos tam tikros taisyklės arba architektūros stilius. Jo nebuvimas yra viena iš priežasčių, dėl kurių atsirado daugiau nei 100 skirtingų AJAX karkasų, o tai savo ruožtu daro technologiją neaiškesnę ir nepalengvina kūrėjų darbo.

Pagrindinė problema yra aiškių taisyklių geriausioms AJAX savybėms pasiekti nebuvimas. Ji išspręsta sukūrus SPIAR architektūros stilių. Nieko panašaus šiuo metu rinkoje nėra.

Architektūros stilius turi būti visų internetinių svetainių ir tinklo programų kūrimo pagrindas, nes tik teisinga architektūra lemia daugumą svarbiausių pageidaujamų savybių, taip pat ji suteikia ir būtinus apribojimus toms savybėms pasiekti. Svetainių kūrimui palengvinti naudojami įvairūs karkasai, tačiau norint išsirinkti geriausią iš jų, pagrindinis kriterijus yra architektūra, nes nuo jos priklausys visų su tuo karkasu sukurtų tinklo programų ir svetainių savybės. Šiame darbe pristatyta ir patobulinta architektūra sprendžiamos šios problemos ir sudaromas ateities pagrindas, kuris įgalina tinklo programas ir svetaines naudotis visomis geriausiomis AJAX savybėmis bei realiuoju laiku siunčiamais atnaujintais duomenimis, gautais integravus į architektūrą COMET technologiją.

Tema aktuali ir nauja, nes liečia naujausias internetinių svetainių kūrimo technologijas. Technologijoms analizuoti buvo naudojamas palyginimo metodas.

Pasiekti architektūros tobulinimo rezultatai yra unikalūs ir niekur kitur nenagrinėti.

Ginamieji teiginiai:

1. Patobulinta COMET karkasuose naudojama brukimu pagrista SPIAR architektūros veikimo schema.

2. Sukurta viską apimanti brukimu pagrista SPIAR architektūros veikimo schema.

Darbo tikslas

Darbo tikslas – išanalizuoti internetinių svetainių kūrimo technologijas, pasiūlyti geriausią architektūros stilių, jį patobulinti ir palyginti su kitais stiliais.

Tikslui pasiekti keliami tokie uždaviniai:

- Nustatyti perspektyviausią technologiją.
- Patobulinti pasirinktą architektūros stilių ir nustatyti jo savybes bei apribojimus.
- Sukurti internetinę svetainę, išryškinančią darbe analizuotas savybes ir praktiškai parodančią AJAX galimybes.

1. Taikomųjų technologijų apžvalga

Modernioms interneto programoms (angl. *Rich Internet Applications* – RIA) kurti naudojamos technologijos, iš kurių pagrindinės yra šios:

- AJAX
- *Microsoft Silverlight*
- *Adobe Flex/Flash*
- *Sun's JavaFX*

Sudėtinga kurti AJAX programas dėl skirtingų naršyklių nesuderinamumo, kuris kyla dėl to, jog ne visi naršyklių kūrėjai laikosi bendrų standartų. *JavaScript* kūrėjai turi daug laiko skirti šioms problemoms spręsti (arba naudoti karkasą *Prototype* arba *jQuery*).

Su *Silverlight/Flex/JavaFX* dirbama naudojant virtualią mašiną (toliau – VM), todėl aplinka yra visur vienoda nepriklausomai nuo to, kokiame naudotojo kompiuteryje kodas yra naudojamas, taip palengvinamas kūrėjų darbas. Dalis kūrėjų mano, kad ateitis yra tokia: visi klientai turės įdiegtą mažą virtualią mašiną. Kad technologiją priimtų naudotojai, reikia turėti technologijos naudojimo galimybes, dėl to *Adobe* turi privalumą, nes jos *Flash Player 9* yra įdiegtas daugiau nei 83% pasaulio kompiuterių (2007 m. duomenimis). Trūkumas tas, jog atnaujinus virtualią mašiną nauja versija, naudotojui ją reikia diegti iš naujo. Internetinių svetainių kūrimo technologijų palyginimas parodytas 1 lentelėje.

Technologijų savybės:

AJAX:

Pliusai:

- Nereikia iš naujo įdiegti.
- Nemokama, atvirasis kodas.

Minusai:

- Priklauso nuo interneto naršyklės.
- 100+ karkasų (sunku išsirinkti).
- Ilgas kūrimo ciklas.

JavaFX:

- Scenarijai ir mobilumas.
- Naudoja *Java Swing* 2D API.
- Yra *Netbeans* ir *Eclipse* plėtiniai.
- Naujesnė *Java 6*. Maža VM įdiegiama į naršyklę.

Silverlight:

- Palaiko iki 720p-HD vaizdą naršyklėje.

- *Silverlight 2.0* programa užima 4.3 MB, galima greitai įdiegti.

Adobe Flex /Flash:

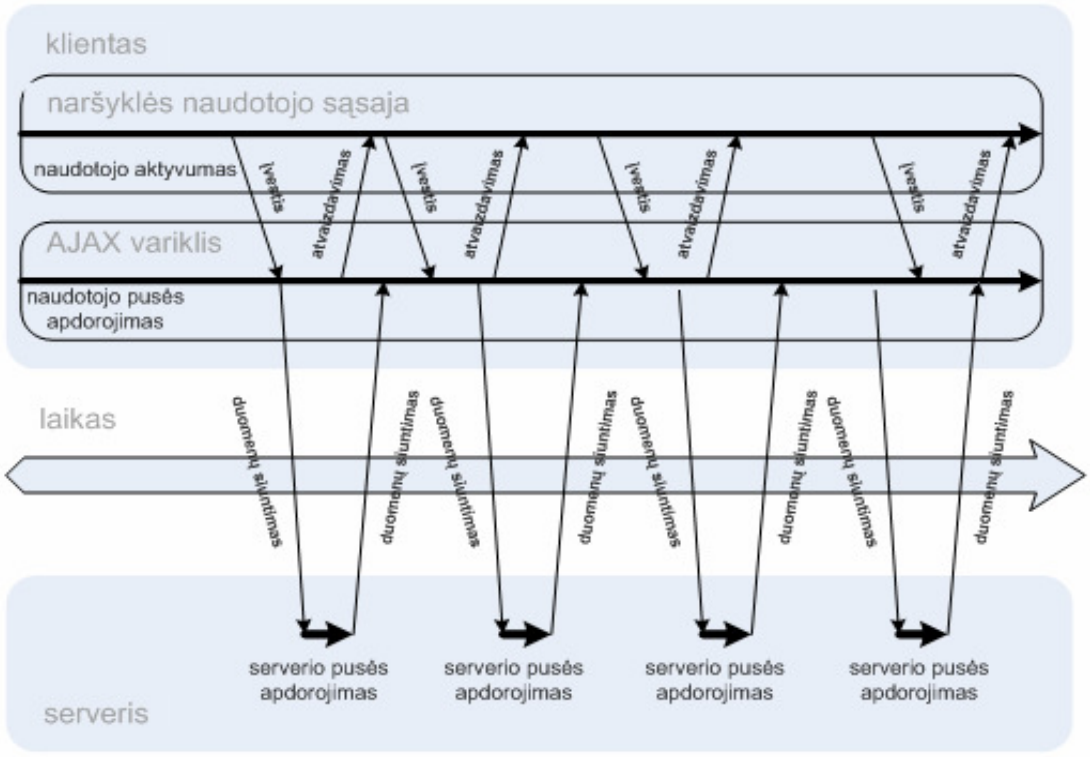
- MXML: XML paremta deklaratyvi programavimo kalba, skirta kurti GUI.
- *ActionScript 3.0*: į objektus orientuota kalba panaši į *Java*.
- *Flash Player 9*: VM kuri paleidžia swf failus.
- *WebOrb*: leidžia naudoti C# objektus siunčiant į *Flex* UI (galima naudoti Flex su .NET).

1 lentelė. Internetinių svetainių kūrimo technologijų palyginimas

Technologijos	Atvirieji standartai	Naudotojui nereikia papildomai diegti programos ir jos naujų dalykų	Dauguma kūrėjų išmano technologijoje vartojamą kalbą	Lankstumas manipuliuojant HTML puslapiu	Populiarumas ir kaina
AJAX	+	+	+	+	+
JavaFX	-	-	-	-	-
Silverlight	-	-	-	-	-
Adobe Flex/Fash	-	-	-	-	+/-

Iš 1 lentelės matyti, jog geriausia technologija yra AJAX, kuri turi daugiausia pliusų, nes palaiko atviruosius standartus bei nevargina naudotojų papildomų modulių siuntimu ir diegimu. Kitos technologijos neturi atitinkamų savybių visai arba neprilygsta AJAX galimybėms.

Modernaus *XMLHttpRequest* objekto pasirodymas ir jo lengvas asinchroninis naudojimas ganėtinai greitai padėjo sukurti naują tinklo programų rūšį. Nebeliko daugiau sinchroninio ir uždaro užklausos bei atsakymo ciklo. Dabar galima nusiųsti užklausą fone, kurios atsakymas sukelia dalinį pokytį (*JavaScript* geba valdyti puslapį per DOM). Galima turėti iš dalies atnaujintus puslapius, foninis serverio bendravimas ir užklausos atliekamos pasitelkus programavimo logiką ir neveikia naudotojo sąveikos. Tai AJAX programos modelis, kuris parodytas 1 pav. (Baila, Beltran 2007).



1 pav. AJAX HTTP modelis

Modelis rodo, kad naudotojui nereikia laukti atsakymo iš serverio, jį dažnai gali atlikti AJAX variklis ir atsakas gali būti dalinis.

AJAX akronimas reiškia asinchroninį *JavaScript* ir XML. Buvo sumanyta, kad į originalią sąvoką bus įtraukta XML kalba, kuri leistų teikti atsakymą tol, kol *JavaScript* galės gramatiškai jį išnagrinėti ir atitinkamai pakeisti puslapio būseną bei turinį naudojant DOM sąsają. Kai kurios programos naudoja tokį modelį, bet dažniausiai kūrėjai naudoja supaprastintą ir patogesnį modelį, kur atsakymas yra parašytas tiesiog HTML kalba (dalinis puslapis). Tad norint viską atlikti tereikia pakeisti atitinkamą puslapio dalį. Taip pat dažnai įdedamas *JavaScript* kodas į serveriui skirtą atsakymą, tokiu būdu serveris gali iškviesti bet kokį puslapio įvykį.

2. AJAX ir jos karkasų analizė

AJAX yra vardas, suteiktas šiuolaikinių tinklo programų kūrimo technologijų komplektui, anksčiau žinomų kaip dinamiška HTML kalba (DHTML) ir nuotolinis scenarijų naudojimas, skirtas suteikti interaktyvią tinklu pagrįstą naudotojo sąsają.

AJAX suvienija: standartais pagrįstą atvaizdavimą naudojant XHTML ir CSS, dinamišką parodymą ir sąveiką naudojant Dokumento Objekto Modelį, duomenų mainus ir manipuliavimą,

asinchronišką duomenų paiešką naudojant *XMLHttpRequest* ir visko sujungimą naudojant *JavaScript*. Šis apibrėžimas susitelkia tiktai ties tinklo programa, esančia kliento pusėje.

Šitų technologijų kombinacija daro AJAX unikalią ir galingą tinkle. Jos galia darėsi akivaizdi tokiose tinklo programose kaip *Google Suggest* ir *Google Map*. Kiti gerai žinomi pavyzdžiai yra *Flickr*, *Gmail* ir nauja *Yahoo* pašto versija.

2.1. Karkasai

Tinklo programų kūrėjai nuolat turėjo problemų su HTML kalbos apribojimais, susijusiais su puslapių seka ir sunkiu kliento pusės *JavaScript* programavimu, kai reikėdavo padidinti tam tikrą dinamiškumo laipsnį naudotojo sąsajai. Problemos dėl naršyklių suderinamumo yra žinomos kiekvienam, kas kūrė tikrą tinklo programą. Moderni naudotojo sąsaja (UI), kurią žada AJAX, turės susidurti su visomis tokiomis problemomis. Kūrėjai, turi turėti gerų įgūdžių iš įvairių tinklo technologijų, kad galėtų kurti sunkiai kuriamas AJAX programas. Taip pat daug pastangų turi būti įdėta šių programų testavimui prieš atiduodant į gamybą. Čia į pagalbą ateina karkasai. Bent jau daugelis jų į tai pretenduoja.

Per tą laikotarpį, kai buvo naudojama AJAX, pasirodė didelis skaičius karkasų. Svarbu, kad šiame konkuruojančiame chaotiškame pasaulyje būtų tvarka, nes prie žinomų karkasų sąrašo kiekvieną dieną prisideda dar po vieną naują karkasą.

Buvo analizuota ir eksperimentuota su keliais AJAX karkasais, bandyta suprasti jų architektūros ypatybes. Šiame skyriuje aptarti trys karkasai. Buvo pasirinktas plačiai naudotas atvirojo kodo karkasas, pavadintas *Echo2*, tinklo karkasas, pasiūlytas *Google* ir pavadintas GWT, ir komercinis paketas, pateiktas *Backbase*. Visi trys karkasai yra pagrindiniai žaidėjai AJAX rinkoje ir jų pagrindinės technologijos skiriasi iš esmės.

2.1.1. *Echo2*

Echo2 yra atvirojo kodo AJAX karkasas, kuriuo programuotojas gali kurti tinklo programas, naudodamas objektinę, naudotojo sąsajos komponentais pagrįstą ir įvykiais valdomą kūrimo tinkle paradigmą. Jo *Java* programos karkasas aprūpina programine sąsaja (UI komponentai, savybių objektai ir įvykiai/klausimo elementai), kad ji atstovautų ir valdytų programos būseną ir jos naudotojo sąsają.

Visas funkcionalumas skirtas komponentų vaizdavimui ar susisiekti su kliento naršykle yra specialiai sutelktas atskirame modulyje, pavadintame tinklo perteikimo varikliu. Variklis susideda iš serverio pusės dalies (parašytos *Java*/J2EE) ir kliento pusės dalies (*JavaScript*).

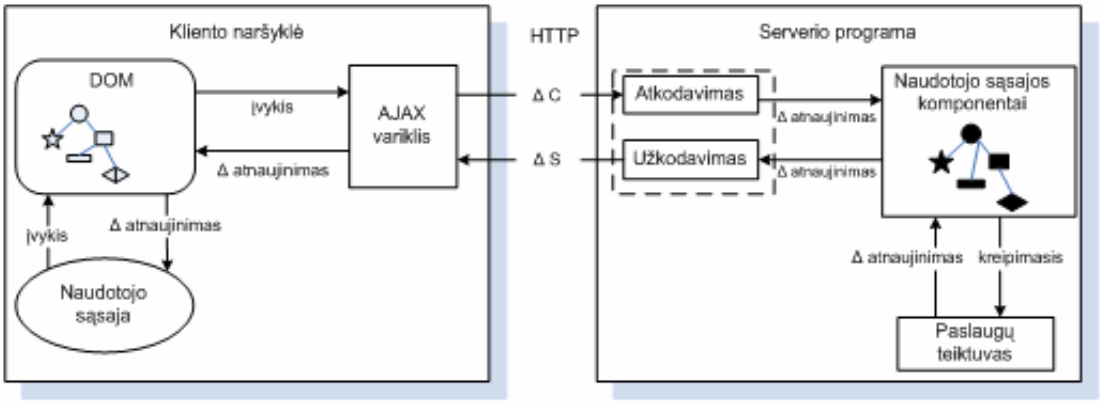
Kliento/serverio sąveikos protokolas yra paslėptas už šio modulio ir yra visiškai atskirtas nuo kitų modulių. *Echo2* turi atnaujintų duomenų valdiklį, kuris yra atsakingas už sekimą atnaujintų duomenų, einančių į naudotojo sąsajos komponentų modelį, kad apdorotų įvestį, gautą iš perteikimo elemento, ir praneštų komponentams.

Echo2 kliento variklis veikia kliento naršyklėje ir suteikia tolimą naudotojo sąsają ir serverio pusės programą. Jo svarbiausia veikla yra sinchronizuoti kliento/serverio būsenas, kai naudotojų operacijos atsiranda sąsajoje.

Kliento žinutė XML formatu yra naudojama, kad persiųstų kliento būsenos pokytį serveriui, aiškiai išdėstydama pakeitimo priežastį ir atitinkamo komponento identifikavimo numerį, ties kuriuo pakeitimas įvyko. Serveris apdoroja kliento žinutę, atnaujindamas komponentinį modelį, kad jis atspindėtų naudotojo veiksmus. Įvykiai, paleisti sudominusių klausimo elementų pusėje, galimai sukelia tolesnius serverio pusės programos būsenos pokyčius. Serveris atsako, teikdamas serverio žinutę, kuri yra vėl XML tipo žinutė, turinti nurodymus, skirtus daliniams DOM vaizdo atnaujinimams kliento pusėje įvykdyti.

2.1.1.1. *Echo2* architektūros veikimas

2 paveiksle pavaizduotas SPIAR pagrįstos architektūros apdorojimo vaizdas. Veikimas atspindi vaizdą įgyvendintą *Echo2* karkase. Vaizdas rodo skirtingų komponentų darbą praėjus šiek tiek laiko nuo inicijuoto puslapio užklausimo (variklis veikia kliento pusėje).



2 pav. SPIAR pagrįstos architektūros veikimo schema

Modelis rodo, kad naudotojui nereikia laukti atsakymo iš serverio, jį dažnai gali atlikti AJAX variklis ir atsakas gali būti dalinis.

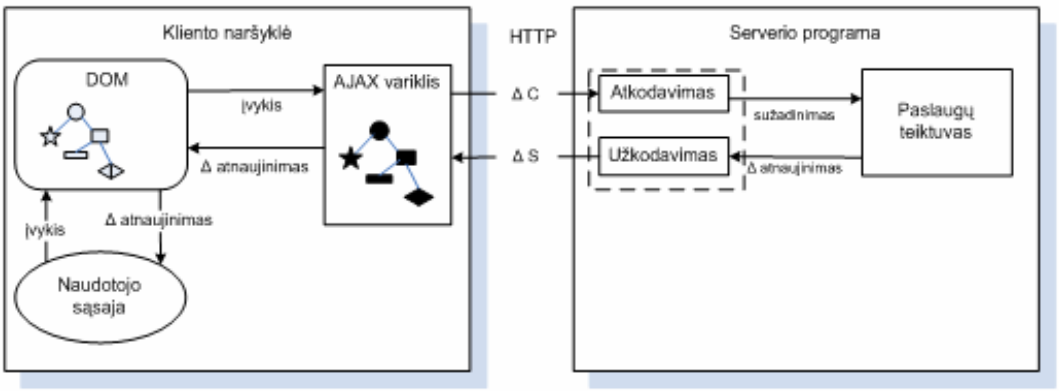
2.1.2. GWT

Google kompanija turi naują AJAX karkaso įgyvendinimo metodą, *Google* tinklo karkasą (GWT). Taip pat kaip *Echo2*, GWT lengvina naudotojo sąsajos kūrimą, panašiai kaip AWT ar *Swing*, ir turi priedų biblioteką, kurią galima naudoti. Unikali GWT savybė yra ta, kaip jis teikia kliento pusės naudotojo sąsają. Užuot laikęs naudotojo sąsajos komponentus serverio pusėje ir pranešinėjęs būsenos pokyčius, GWT surenka visus *Java* UI komponentus į *JavaScript* kodą (prisideda surinkimo laikas). Komponentų vidaus kūrėjui leidžiama panaudoti *Java* 1.4 programavimo sąsają ir taip įgyvendinti būtiną funkcionalumą.

GWT naudoja mažą bendrą kliento variklį ir kompiliatorių. Taip visas naudotojo sąsajos funkcionalumas tampa pasiekiamas naudotojui kliento pusėje. Šis metodas drastiškai sumažina keliones į serverį pirmyn ir atgal. Su serveriu tiktai konsultuojamasi, jei nauji duomenys yra reikalingi, norint papildyti kliento pusės UI komponentus. Tai atliekama priskiriant serverio užklausas apibrėžtoms tarnyboms. Tarnybos (kurios nėra tas pat kas yra tinklo tarnybos) yra įgyvendintos Java kalba, ir duomenys perduodami tinklu į abi puses, naudojant serijinius metodus.

2.1.2.1. GWT architektūros veikimas

3 paveiksle pavaizduotas apdorojimo esančio GWT karkase vaizdas.



3 pav. RPC pagrįstos GWT architektūros veikimo vaizdas

Iš pavyzdžio galima pamatyti, kad GWT nepalaiko serverio pusės komponentų medžio. Vietoj to, serverio pusės naudotojo sąsajos komponentai kompiliavimo metu buvo transformuoti į kliento pusės komponentus. Kliento variklis žino visų prieinamų komponentų rinkinį ir jų buvimo vietą programos veikimo metu. RPC pagrįsta sąveika su serveriu vis dėlto tvarkoma

delta komunikacijos stiliumi. Čia koduotuvas ir dekoderis kalba tiesiogiai su paslaugų tiektuvu neinant pro serverio pusės komponentų modelį.

2.1.3. *Backbase*

Backbase yra Amsterdamo kompanija, kuri pagamino vieną iš pirmų komercinių AJAX karkasų. Karkasas yra vis dar tobulinamas, ir gausiai klientų naudojamas visame pasaulyje.

Pagrindinis *Backbase* karkaso elementas yra *Backbase* pristatymo klientas. Tai standartais pagrįstas variklis, parašytas *Javascript*, kuris veikia tinklo naršyklėje. Jis gali būti užprogramuojamas vartojant deklaratyvią naudotojo sąsajos kalbą, pavadintą BXML (Crane, Pascarello ir kt. 2006). BXML siūlo biblioteką su naudotojo sąsajos valdikliais, mechanizmu skirtu prijungti prie jų veiksmams, priemonėms, kad būtų galima jungtis prie serverio asinchroniškai.

Backbase karkase serverio pusė yra suformuota pasitelkiant BJS. BJS – *Backbase Java* serveris. Jis yra ant *Java* serverio komponentų (JSF) viršaus, t.y. nauja J2EE pristatymo architektūra. JSF apsirūpina, naudotojo sąsajos komponentais pagrįstu karkasu, kuris remiasi modelis-vaizdas-valdiklis šablonu. Sąveika JSF yra pagrįsta klasikiniu puslapių sekos modeliu, todėl nereikia visko integruoti į vieną puslapį.

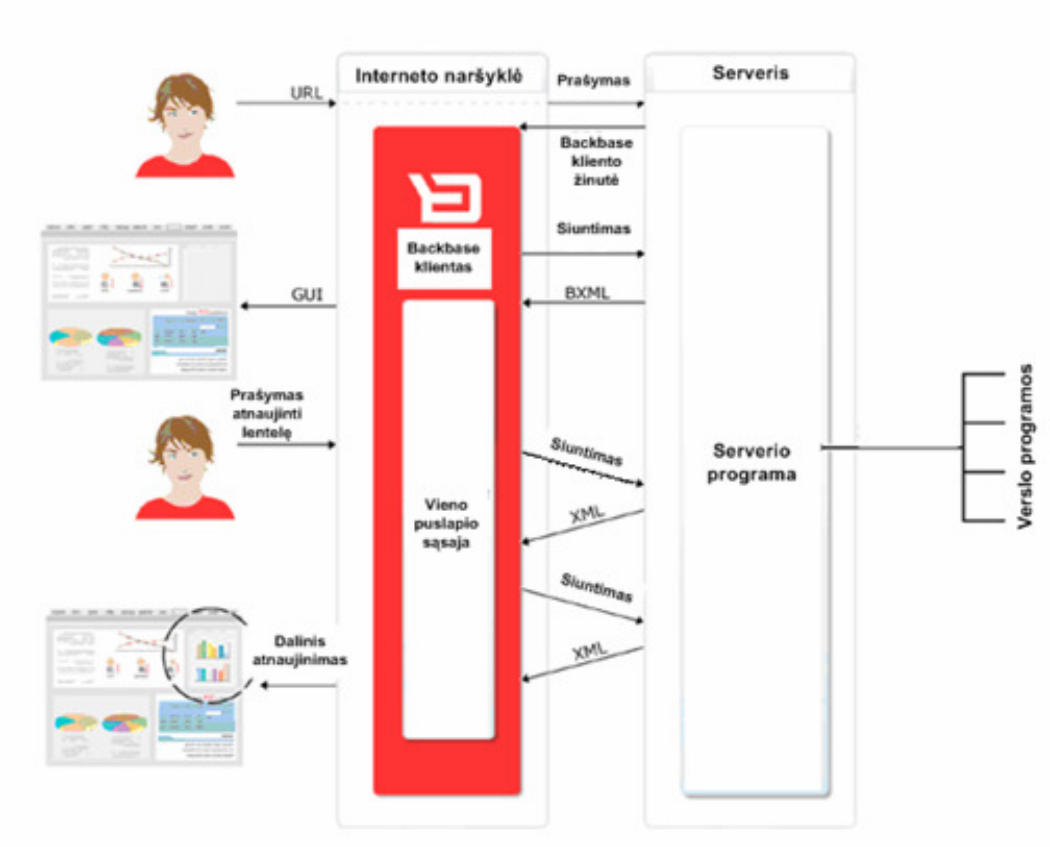
Backbase Java serveris aprūpina savo paties naudotojo sąsajos komponentų komplektu ir plečia JSF karkasą, kad būtų įgyvendinta vieno puslapio sąsaja. Kūrėjai gali naudoti komponentus deklaratyviai (per tinklo scenarijus) ir kurti AJAX programas.

Karkasas perduoda kiekvieną deklaruotą serverio pusės UI komponentą atitinkamam kliento pusės (BXML) UI komponentui ir seka pokyčius abiejuose komponentų medžiuose, kad išliktų sinchroniškumas.

Būsenos pokyčiai kliento pusėje nusiunčiami į serverį, atsiradus tam tikriems apibrėžtiems įvykiams. Tai gali būti veiksmo įvykiai kaip mygtuko spragtelėjimas, ar reikšmės pakeitimo įvykiai kaip radijo mygtuko pažymėjimas. Serveris išverčia šių būsenų pokyčius ir identifikuoja atitinkamą komponentą ar komponentus serverio komponentų medyje. Po reikalingo veiksmo serveris pateikia pokyčius, kurie bus atsakymas vėl BXML formato varikliui.

2.1.3.1. *Backbase* architektūros veikimas

Reikia pastebėti, jog kiekvienas iš karkasų naudoja skirtingą SPIAR architektūrinių elementų rinkinį tam, kad pristatytų esamą architektūros darbo vaizdą. 4 paveiksle parodyta *Backbase* veikimo schema.



4 pav. *Backbase* veikimo schema

Ši schema yra pateikiama *Backbase* kompanijos ir palyginus su kitomis nagrinėjamomis architektūromis nėra tokia informatyvi.

2.2. Karkasų savybės

Nors daugeliu atvejų šie karkasai skirtingi, tačiau turi bendrų architektūrinių savybių. Apskritai šitų karkasų tikslai gali būti tokie:

- Paslėpti sudėtingus dalykus kuriamų AJAX programų, nes šių programų kūrimas yra nuobodus, sudėtingas ir linkęs į klaidas.
- Paslėpti nesuderinamumus tarp skirtingų tinklo naršyklių ir platformų.
- Paslėpti kliento/serverio komunikacijos sudėtingus dalykus.
- Pasiekti didelį interaktyvumą ir mobilumą galutiniams naudotojams ir palengvinti kūrėjų darbą.

Šitų tikslų galima pasiekti aprūpinant naudotojo sąsają komponentų ir kūrimo aplinkos biblioteka, kuri sukurtų daugkartinio naudojimo padarytus pagal užsakymą komponentus. Architektūra turi gerai apibrėžtą protokolą mažoms sąveikoms tarp žinomų kliento/serverio komponentų. Duomenų apimtis, kuri turi būti persiųsta tinku, yra žymiai sumažinama. Todėl gali

būti greičiau persiunčiami atsakymo duomenys. Jų architektūra pasinaudoja kliento pusės apdorojimu, ir taip pagerina naudotojų interaktyvumą, mažesnį skaičių siuntų pirmyn ir atgal, ir sumažina tinklo serverio apkrovą.

3. Architektūros stiliai ir terminologija

Programinės įrangos architektūra yra apibrėžiama kaip konfigūracija architektūrinių elementų, apdorojimo, jungtimis ir ryšiais susietų duomenų, kad galima būtų pasiekti pageidaujamą architektūrinių ypatybių komplektą.

Architektūrinis stilius savo ruožtu yra koordinuotas architektūrinių apribojimų komplektas, kuris suvaržo architektūrinių elementų vaidmenis ir leidžia, bet kurios architektūros santykius tarp vidaus elementų, kuri atitinka tą stilių. Architektūrinis stilius varžo ir dizaino elementus ir santykius tarp jų, tokiu būdu padaro programinės įrangos sistemas su tam tikromis pageidaujamomis ypatybėmis.

Architektūrinė sistema gali būti sudaryta iš daugelio stilių, ir stiliai gali būti kitų stilių hibridais. Stiliai gali būti matomi kaip skirtingų sistemų architektūros vidaus daugkartinio naudojimo bendri architektūriniai šablonai.

3.1. Egzistuojantys stiliai

Naudotojo sąsajos programos apskritai naudoja tokius populiarius stilius kaip Modulis/Vaizdas/Valdiklis, kad apibūdintų didelio masto architektūrą ir specifiškesniais atvejais, stilius kaip C2 naudojančius asinchroninius pranešimus apie būsenų pokyčius ir užklausų žinutes tarp nepriklausomų komponentų.

Daug skirtingų, tinklu pagrįstų, architektūrinių stilių, tokių kaip klientas/serveris, n-pakopa ir kodas pagal pareikalavimą egzistuoja, tačiau, labiausiai užbaigtas ir tinkamas stilius tinklui iki šiol yra REST (klasikinis atvaizdavimo būsenos persiuntimas).

REST apibrėžia duomenų abstrakciją ir paslaugas kaip išteklius, kurių gali prašyti klientai, vartodami išteklių pavadinimą ir adresą, kurie apibrėžti kaip vienodų išteklių paieška (URL). Stilius paveldi savybes iš daugelio kitų stilių, tokių kaip klientas/serveris, vamzdis-ir-filtras bei paskirstyti objektai.

Stilius yra svarbiausių tinklo architektūros ypatybių apibūdinimas architektūriniais apribojimais, kurie žymiai prisidėjo prie tinklo sėkmės.

Kalba eina apie penkis fundamentalius elementus: ištekliai, kurie gali būti bet kuo, kas turi tapatumą, pavyzdžiui, dokumentas ar paveiksliukas, išteklių atvaizdavimas, kurio forma yra

multimedijos tipo, sinchroninė prašymo atsakymo sąveika per HTTP, kad gautų ar pakeistų vaizdą, internetinis puslapis kaip programos būsenos pavyzdys, ir varikliai (pavyzdžiui, naršyklė), kad judėtų nuo vienos būsenos į kitą.

REST apibrėžia kliento/be būsenos/serverio architektūrą, kurioje serijos tarpinių talpyklų ir filtrų gali būti panaudota, ir kiekvienas prašymas yra nepriklausomas nuo ankstesnių. Tai taip pat pabrėžia vienodos sąsajos tarp komponentų, varžančių informaciją, būtinumą persiųsti standartizuota forma (Mesbah, Deursen 2008).

4. Sprendimai ir kompromisai kuriant AJAX karkasus ir programas

Naudojant SPIAR architektūros stilių bus išnagrinėti įvairūs sprendimai ir kompromisai, kuriuos reikia atlikti kuriant AJAX karkasus ir programas.

4.1. Ištekliai ir komponentai

Žiniatinklio architektūra yra pagrįsta ištekliais, kurie yra identifikuojami pasitelkiant vienodų išteklių identifikatorių (URI) ir protokolais, kurie palaiko sąveiką tarp agentų ir išteklių. Naudojant bendrą sąsają ir aprūpinant identifikavimu, kuris yra būdingas tinklo ištekliams buvo vienas iš pagrindinių tinklo sėkmės veiksnių.

Tinklo architektūros prigimtis, kuri susijusi su tinklapiais kaip ištekliais, sukelia nereikalingų duomenų siuntimą. Delta komunikacijos bendravimo būdas SPIAR stiliuje remiasi komponentų lygiu ir nesilaiko išteklių/URI tinklo architektūros apribojimų. Klausimas yra, ar šis pasirinkimas yra pateisinamas. Norint atsakyti į šį klausimą, reikia pažiūrėti į vieno puslapio programų vidinės sąveikos prigimtį: saugios sąveikos ir pavojingos.

4.2. Saugios ir nesaugios sąveikos

Apskritai, kliento/serverio sąveikos tinklo programoje gali būti padalytos į dvi kategorijas *saugias* ir *pavojingas* sąveikas. Saugi sąveika yra tokia, kur naudotojas neturi būti laikomas atsakingas už sąveikos rezultatą, pavyzdžiui, paprastos užklausos. Pavojinga sąveika yra tokia, kur naudotojo užklausa turi galimybę pakeisti išteklių būseną.

Vieno puslapio interneto programose, kur sąveika tampa vis labiau panaši į darbalaukio, kur galų gale Atšaukti/Perdaryti, komandas keičia Atgal/Pirmyn, saugios sąveikos lieka naudoti URI, kai pavojingos gali būti saugiai atliktos fone naudojant delta komunikaciją, kurioje nei

perduodami duomenys, nei duomenys, gauti atsakyme nebūtinai atitinka kokius nors išteklius, identifikuotus URI.

Variklis turi suteikti jungimo priemonę su saugiomis operacijomis taip pat, kaip tai atlieka nuorodos būdu sujungti tinklalapiai. Šiuo tikslu gali būti panaudotas URI fragmento identifikatorius. Fragmento identifikatoriaus interpretacija yra įvykdyta tada, kai variklis iš naujo nurodo URI, kad jis identifikuotų ir atitiktų programos būseną.

4.3. Kliento ar serverio pusės apdorojimas

Dabartiniuose karkasuose kūrėjams nėra galimybės išsirinkti ar tam tikras funkcionalumas turi būti atliekamas kliento pusėje ar serverio pusėje. Tobulinant tinklo programas gali būti svarbiu veiksniu tai, kaip skaičiavimas yra paskirstytas. AJAX karkasų architektūros turi duoti priemones kūrėjui, kad jis galėtų suprasti ir apsispręsti, kokius išplėstinius skaičiavimus atlikti kliento pusėje ir ar iš vis juos ten atlikti.

4.4. Asinchroninės sąveikos sinchronizacija

Asinchroninė sąveika AJAX programose gali sukelti lenktynių sąlygas, jei bus viskas įgyvendinama neatsargiai. naudotojas gali nusiųsti prašymą į serverį anksčiau, negu ankstesnis prašymas buvo atsakytas. Serverio procesoriuje, kuris apdoroja užklausas lygiagrečiai, antras prašymas gali potencialiai būti apdorotas prieš pirmą. Šis elgesys gali turėti drastiškus padarinius sinchroniškumui ir visos programos būsenai. Galimas sprendimas dirbti su kiekvieno kliento iššauktais įvykių prašymais iš eilės, tai darant serverio našumo sąskaita.

4.5. Komunikacijos protokolas

Šiuo metu kiekvienas AJAX karkasas turi įgyvendinęs savo komunikacijos protokolą. Tai daro labai prastą kliento/serverio sąveikų matomumą, kadangi kiekvienas turi žinoti, tiksliai tą protokolą, kad sugebėtų suprasti delta žinutes. Todėl šias programas sunku išplėsti. Klientui, norinčiam susisiekti su AJAX serveriu, vėl reikia žinoti to serverio programos protokolą. Šitos dvi ypatybės gali būti pagerintos, apibrėžiant standartinį protokolo aprašymą komunikacijai ir skirtą AJAX bendruomenei.

4.6. Dizaino modeliai

2 iliustracija rodo AJAX tinklo programos *meta* modelį. Naudotojo sąsaja yra sudaryta iš UI komponentų priedelių (angl. *widgets*). Kliento pirmasis puslapis yra sudarytas iš serverio pusės priedėlių. Delta pokyčiai taip pat kaip vaizdo pokyčiai įvyksta priedėlio lygmenyje. Vaizdo pakeitimas, gali būti matomas keliaujant po pasiekiamus priedėlius. AJAX karkasai kūrėjams turi suteikti aiškius navigacinius modelius. Tyrimas yra būtinas, kad galima būtų pasiūlyti dizaino modelius AJAX kūrėjams, pavyzdžiui, išplečiant UML kalbą, norint modeliuoti naudotojų sąveiką, navigaciją per komponentus, asinchroninius/sinchroninius veiksmus ir daugiau apdorojimo kliento pusėje, o ne serverio pusėje.

4.7. Fragmentais pagrįstas metodas

Tinklo puslapio sekos modelyje sudėtinga tinkamai elgtis su atskiromis tinklalapių (fragmentų) dalimis nepriklausomai. Fragmentais pagrįsti tyrinėjimai nusitaiko į aprūpinimo mechanizmą, kad jis efektyviai surinktų tinklo puslapį iš skirtingų dalių, bei sugebėtų fragmentus kaupti tarpinėje atmintyje. Neseniai pasiūlyti metodai apima kelias serverio puses ir tarpinės atminties mechanizmus. Serverio pusės technikos nusitaiko į serverio apkrovimo mažinimą, leidžiant anksčiau sukurto turinio pakartotinį panaudojimą aptarnaujant naudotojų prašymus. Tarpinės atminties pusės technikos bando sumažinti vėlinimą, perkeldama tam tikrą funkcionalumą į tinklo kraštą. Šios fragmentais pagrįstos technikos gali pagerinti tinko ir serverio našumą, bei sumažinti naudotojui suprantamą vėlinimą, leidžiant siųsti tiktai pakeistus ar naujus fragmentus.

Nors fragmentai gali būti gauti nepriklausomai, šiai technikai trūksta naudotojo sąsajos komponentų interaktyvumo, reikalingo interaktyviose programose. UI komponentais pagrįstas modelis priklausantis SPIAR stiliui, kartu su jo delta komunikacija, aprūpina kliento/serverio sąveikos priemonėmis pagrįstais būsenų pakeitimais, kurie nesinaudoja laikinu kaupimu tarpinėje atmintyje.

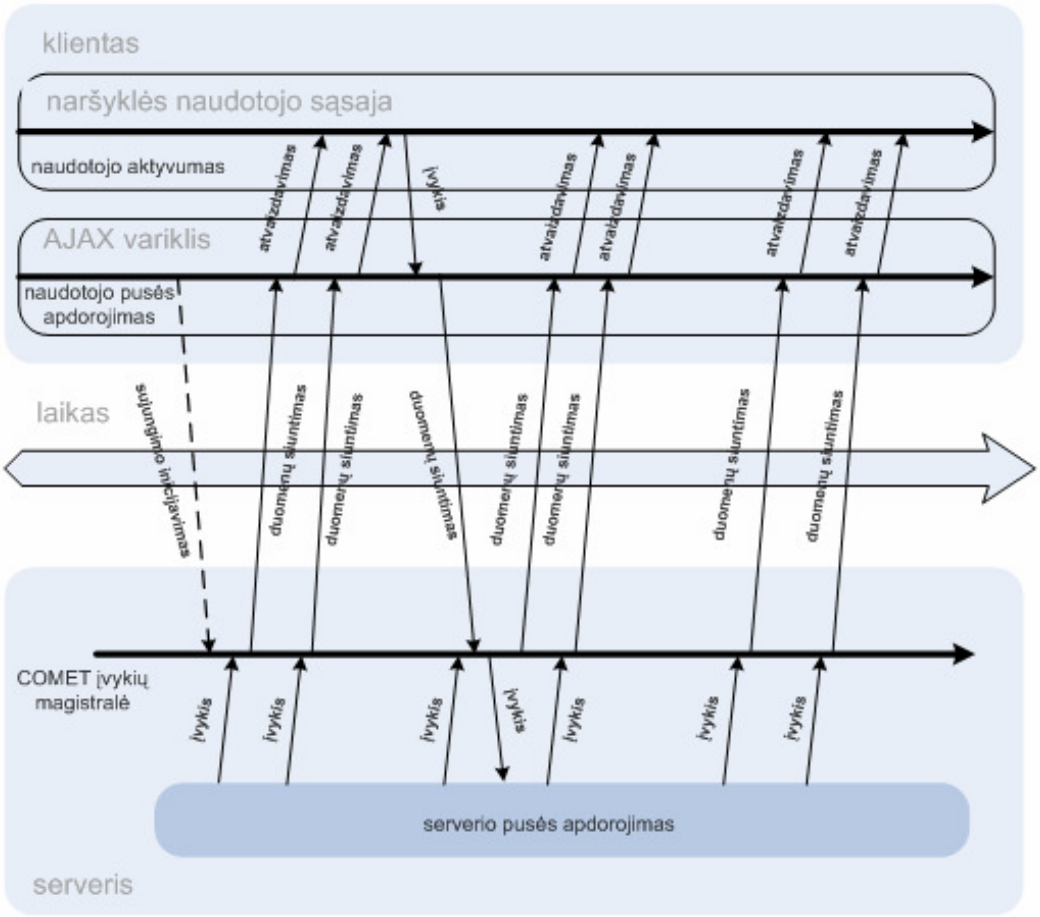
5. COMET

Kaip J.J. Garret paminėjo žodį AJAX (Garrett, 2005), taip pat ten buvo tekstas nuo Alex Russell, kur jis bandė sekti tuo pačiu keliu, paminėdamas COMET terminą, susijusį su serverio galimybe siųsti įvykius klientui, nelaukiant kol iš naršyklės atvyks prašymas. Taip pat kaip ir su AJAX terminu, buvo ankstesnių darbų, kurie bandė išspręsti problemą, dėl tinklo programos negalėjimo gauti asinchroniškų įvykių iš serverio. Tai pažymima, norint pabrėžti minties brandumą.

Šio reikalingumo priežastis buvo AJAX architektūros pasirodymas. Daug naujų labai funkcionalių tinklo programų buvo ištobulinta pasitelkus AJAX, ir abu – kūrėjai ir naudotojai – norėjo patobulinti technologijas dar labiau. Kad ir kokia interaktyvi buvo AJAX programa, ji stokojo pagrindinio mechanizmo, skiriančio ją nuo darbastalio programos: atnaujinimas realiuoju laiku. Kūrėjai pastebėjo, kad reikia skleisti įvykius iš serverio klientui, kad gautųsi tikra įvykiais valdoma tinklo programa.

Pagrindinė problema yra HTTP protokolas. Tai kliento serverio protokolas be galimybės serveriui susisiekti su klientu. Atsirado didelė tinklų bei jungčių tarp naršyklių įvairovė, todėl pasidarė neįmanoma sukurti tokį mechanizmą, kuris galėtų atidaryti jungtį su klientu.

Taigi yra įmanomas tik vienas sprendimas (be įvairių pačio protokolo pokyčių). Turėti atvirąją jungtį laukimo būsenoje laukiančią įvykio, kuris atsiras serveryje (bet kuris COMET klientas tokią turėtų turėti). Taigi kai serveris norės siųsti įvykį vienam ar visiems klientams, jis pasinaudos ta atvirąja jungtimi (tai yra standartinis HTTP sujungimas). Galima pamatyti 5 paveiksle COMET modelio diagramą su HTTP srautinio jungimosi technika. Šio sprendimo grožis yra tas, jog tai veikia. Ir veikia be papildomo klientų, serverių, protokolų ir kt., modifikavimo.



5 pav. HTTP srautų COMET modelis

Deja atsirado kitos prigimties problema. Serveriai su šia architektūra nukenčia dėl keičiamo mastelio problemos.

5.1. COMET architektūra

Tradiciniai tinklo serverių realizavimai, ypatingai tokio serverio, kaip *JavaEE* serveris, naudoja gijų sankaupą, kad galėtų valdyti ateinančius sujungimus. Kai naujas sujungimas atsiranda, jis yra įvykdomas ir gija iš sankaupos yra priskiriama tam prisijungimui. Užklausa yra perskaitoma, kodas yra įvykdomas ir atsakymas yra sugeneruojamas bei išsiunčiamas. Tada gija gražinama į gijų sankaupą. Taip padaryta turint galvoje, kad užklausos vyksta trumpai, bet intensyviai ir priklauso nuo skaičiavimo išteklių. Tačiau COMET sujungimas yra atliekamas su mintim, kad jis bus ilgas pagal laiko tarpą (gali būti kelios minutės) ir labai mažo CPU (CPU – centrinis procesorinis įrenginys) ar atminties intensyvumo. Sujungimas reikalingas tik tam, kad galima būtų išsiųsti klientui įvykius ir tiesiog palaikyti sujungimą atidarytą.

Klasikinė tinklo programa gali lengvai susidoroti su dešimtimis tūkstančių naudotojų prisijungusių vienu metu, nes naudotojai nesiunčia užklausų visą laiką. Taigi aktyvių sujungimų skaičius serveryje visada atspindi programos naudotojų dalį kiekvienu laiko momentu. Su COMET architektūra kiekvienas naudotojas sistemoje yra atvirasis sujungimas serveryje ir gija (su klasikiniu modeliu), kuri yra prirakinta prie atvirojo sujungimo. Net jei nieko nedaroma, bet koks serveris vis tiek turi problemų valdant dešimtis tūkstančių gijų.

Serverius reikia pritaikyti sprendžiant šią problemą. Sprendimas ateina iš žinomo algoritmo, asinchroninio I/O, kuris egzistuoja moderniose operacinėse sistemose labai ilgai. C programuotojai žino tai kaip *select()* arba *pool()* sisteminį iškvietimą. *Java*, pavyzdžiui, palaiko jį nuo 1.4 versijos pristatytos su *java.nio* paketais.

Naujas dizainas atskiria vieno su vienu tarpusavio ryšius tarp sujungimo ir gijos. Taip pat yra gijų sankaupa, tačiau gijos yra naudojamos tik aktyviai naudojamiems sujungimams, o ne tiems sujungimams, kurie tuo metu su duomenimis nedirba. Aišku, serverio pusės komponentų kūrėjams reikia atlikti keletą modifikacijų, tačiau jos reikalingos tik COMET tvarkytuvams.

Be mastelio keitimo serveryje, COMET architektūra sukuria dar vieną problemą kliento pusėje. HTTP protokolas riboja vienu metu galimus prisijungimus iki 2. Naudojant nors vieną sujungimą COMET sujungimui, visas tinklapis lieka tik su vienu sujungimu. Kadangi tinklapis greičiausiai naudoja AJAX modelį, akivaizdu, kad sudėtingas ar paprasčiausiai lėtas atsakas užblokuos visas kitas jungtis ir paliks internetinį puslapį be galimybės išsiųsti kitas užklausas, nes dvi jungtys yra užimtos: viena užimta COMET sujungimo ir kita laukia lėto atsakymo į AJAX užklausą. Taigi darosi beveik neįmanoma viso to apeiti, tačiau galima problemą sumažinti. Vienas reikalingas žingsnis yra perkelti vienu srautu visą COMET bendravimą vienam sujungimui, nes joks puslapis negali sau leisti turėti skirtinguose komponentuose esančius užimtus 2 sujungimus.

COMET architektūra sukuria daug iššūkių ir serverio, ir kliento pusėse. Toliau bus pristatomas tokio modelio labui atlikti darbai ir vidinės detalės.

5.2. *Bayeux* protokolas

Kaip standartizuota architektūra COMET susiduria su ryškiomis suderinamumo problemomis. Tai susiję ir su protokolais ir su įgyvendinimu. Kompanijos su garsiais vardais palaikančios konkrečias bibliotekas ir serverius nori standartinio protokolo skirto komunikacijai tarp COMET kliento (*JavaScript* biblioteka) ir COMET serverio komponentų. Viso to rezultatas yra *Bayeux* protokolas su esančiu už jo *Dojo* karkasu. Taip pat yra atliekamas darbas

autoritetingų W3C žmonių, bandančių tobulinti patį protokolą su nauja versija HTML5, kurioje jau yra numatyti serverio siunčiami įvykiai. Tačiau kol kas šis standartas dar nėra užbaigtas.

Pagrindinis žmogus, stovintis už *Bayeux* yra Alex Russell iš *Dojo*, tai garantuoja, kad šis protokolas tam tikru lygiu bus žinomas. Savo kalboje apie *Bayeux* jis teigė: viena iš didžiausių problemų, prisitaikant COMET yra sunkus jos įgyvendinimas. Tai lėmė, kad buvo pradėtas darbas su *Bayeux* protokolu (Baila, Beltran 2007).

Šio protokolo apibrėžimas skamba taip: „*Bayeux* – tai protokolas, skirtas transportuoti žinutes asinchroniškai per HTTP“. Žinutės yra nukreipiamos per pavadintus kanalus ir gali būti išsiunčiamos: iš serverio į klientą, iš kliento į serverį ir iš kliento į klientą (per serverį viduryje)“. Protokolo specifikacija yra pradinėje stadijoje, tačiau matyti palaikymas bendruomenėje, tai yra geras būdas tobulinti technologijos palaikymą ir kūrimą.

Protokolas bando išspręsti pagrindines problemas, susijusias su COMET architektūra. Jis naudoja JSON (*JavaScript* objektų aprašymas), kaip duomenų mainų formatą aprašyti žinutėms. Šios žinutės aiškiai aprašomos savybių dokumente ir yra paslepiamos visos žemo lygio detalės, reikalingos tokiems dalykams kaip: rankų paspaudimas, sujungimo perdavimas, kanalo patvirtinimas, prisijungimas iš naujo ir kt. Žinučių standartizavimas leidžia kurti suderintas klientines bibliotekas ir serverio komponentus. Papildomai tai užtikrina, kad pagrindiniai dalykai, kaip perdavimas ir prisijungimas iš naujo, yra nepamirštami net ir paprastuose kūrimo darbuose. Protokolas taip pat pristato versijų sistemą, kuria galima perduoti duomenis tarp kliento ir serverio pageidaujamu protokolo lygmeniu, kaip tai daroma HTTP perdavime.

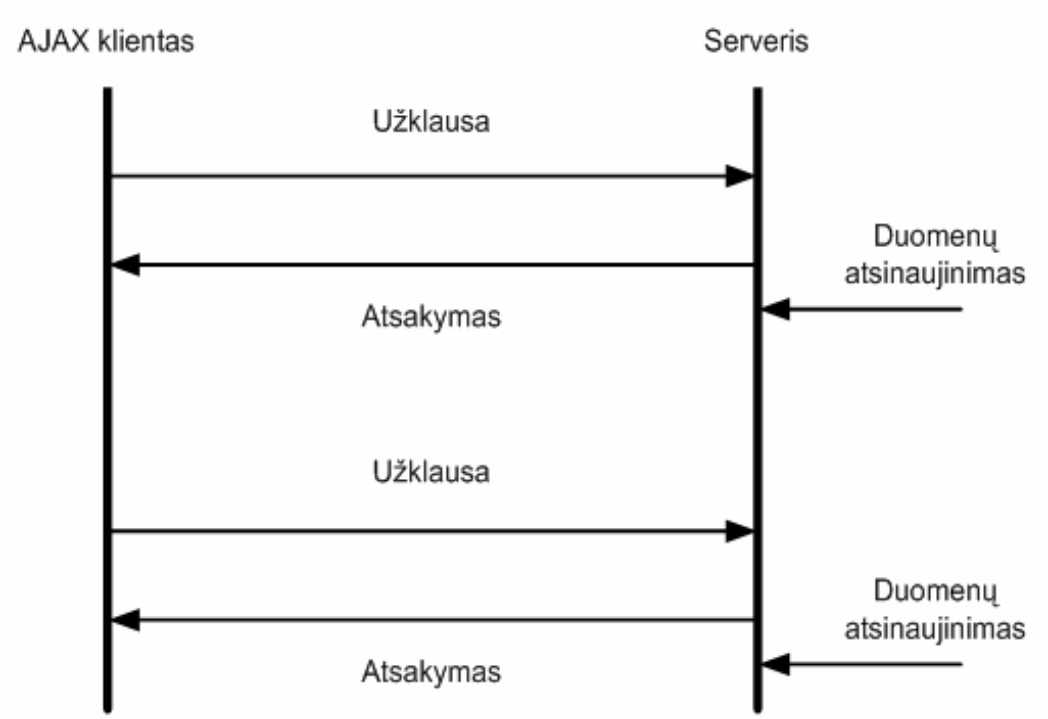
Pagrindinė protokolo koncepcija yra skirtingų galinių taškų sutankinimas, skirtas COMET komponentams, naudojant kanalų mechanizmą. Kiekvienas išsiųstas pranešimas naudojant protokolą žino galutinį kanalą, tai sumažina dviejų sujungimų apribojimo HTTP problemą. Taigi, turint skirtingus serverio komponentus, prieinamus per COMET, nebereikia eikvoti kelių jungčių, o tik vieną. Tai taip pat padeda serverio komponentams likti švariais ir atskirti skirtingus dalykus. Dar vienas naudingas naujumas yra kiekvieno kliento identifikavimas naudojant automatiškai generuojamą identifikacijos numerį, panašų į HTTP sesijos numerius.

Kitas standartinio protokolo pranašumas yra skirtingų sujungimo modelių palaikymas ir perdavimo protokolas. COMET architektūra yra būdas, apeinantis HTTP protokolo apribojimus, taigi skirtingi sujungimo metodai yra ne tik reikalingi, bet net pageidaujami, nes taip galima būtų palaikyti kuo daugiau klientų. Toliau bus analizuojami skirtingi COMET jungimo modeliai.

5.3. COMET sujungimo modeliai

Aišku, jog COMET architektūrai reikia kaip nors turėti nuolatinį sujungimą su serveriu, kad būtų įmanoma gauti serverio generuojamus pranešimus. Bet šio sujungimo valdymas gali būti skirtingas, pagal tai kaip valdoma ne tik kliento pusėje, bet ir serverio pusėje. Pasirinkti geriausią nėra lengva.

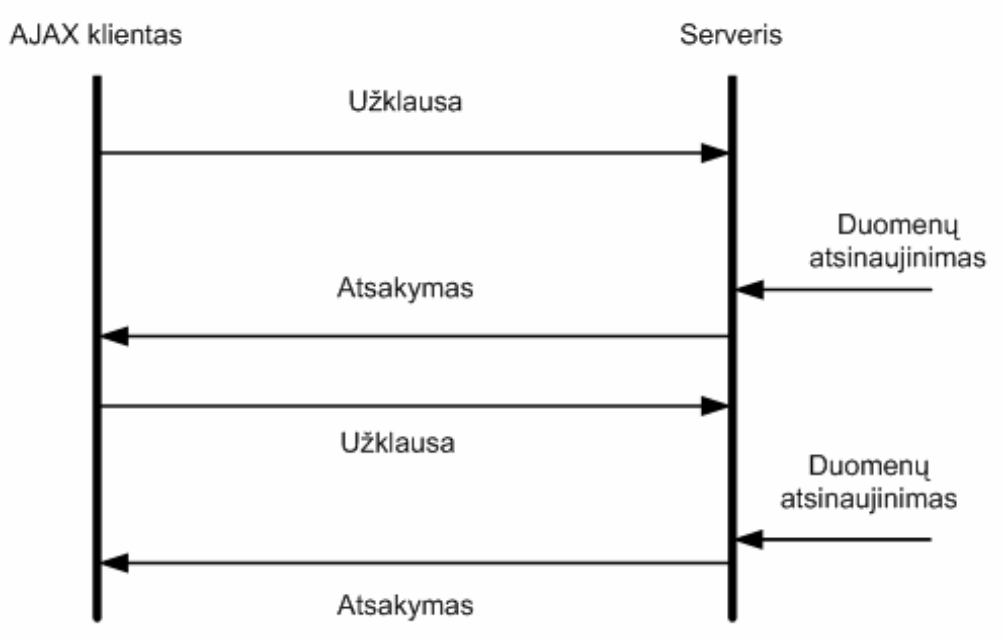
Pirmasis, bei pirmasis naudotas prieš COMET atėjimą ar net prieš AJAX, yra balsavimo (angl. *polling*) sujungimo modelis, parodytas 6 paveiksle. Jis negali būti vadinamas kaip tikras COMET modelis, nes jis negauna įvykio, kai jis įvyksta, tačiau jis rodo, kaip buvo praeityje ir jis rodo kur COMET gali tikrai padėti.



6 pav. Balsavimo sujungimo modelis

Aišku, kad šis modelis turi mastelio keitimo problemą. Nors jis neturi problemų laikant laukiantį sujungimą, tačiau gautų balsavimo užklausų serveryje gali būti labai daug ir didelio dažnio reikšmėmis, tai yra pageidaujama norint, kad programa būtų greitai reaguojanti. Šis modelis gali dirbti su nedideliu naudotojų skaičiumi, net esant 2 ar 3 sekundžių duomenų atnaujinimu. Šis modelis taip pat turi kitų minusų. Yra naujų užklausų ir atsakymų viršijimas. Taip pat, pagrindinė problema, priklausomai nuo programos naudojimo yra ta, kad keletas arba daug jungčių gali būti tuščios, klientas tada prašo serverio įvykių ir gauna neigiamą atsakymą. Tai be reikalo perkrauna serverį ir tinklą.

Ilgas balsavimas yra šios technikos evoliucija, kuri išsprendžia tuščių užklausų problemą, nes serveris atsako tik tada, kai yra duomenų. Tuo laiku jungtis tiesiog laukia. 7 paveiksle galima matyti, jog modelis siunčia užklausą prašydamas duomenų serveriui, kuris taip pat laukia, kol atsiras įvykis. Taigi užklausos tuščios niekada negrįžta, tačiau internetinėje svetainėje galima turėti vidutinį skaičių prisijungusių klientų.



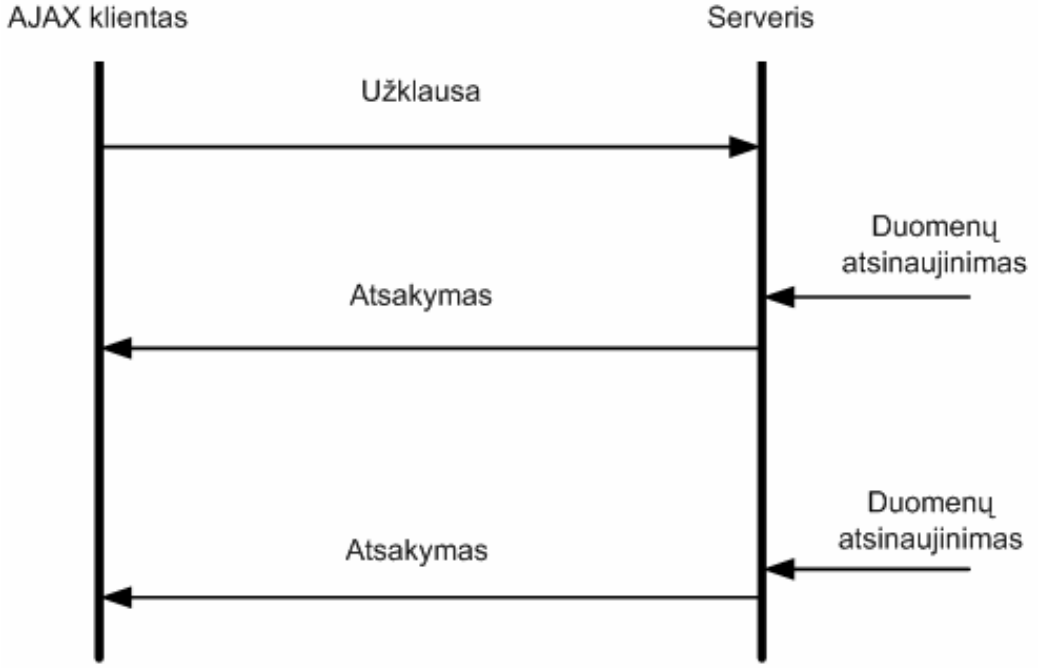
7 pav. Ilgo balsavimo sujungimo modelis

Kadangi šis modelis išsprendžia tuščio atsakymo problemą, gali atsirasti plečiamumo problema tuose serveriuose, kurie blokuoja užklausą ir turi klasikinį 1:1 žymėjimą, esantį tarp gijų ir sujungimų. Taip yra todėl, kad norint turėti programą, kuri apdoroja dešimtis tūkstančių arba daugiau vienu metu prisijungusių klientų, tinklo programoje reikia turėti tiek pat sujungimų. Taigi dėl ilgo balsavimo algoritmo, pavyzdžiui, turint 10000 naudotojų programoje reiškia, reikia taip pat turėti 10000 AJAX sujungimų. Tai klasikiniame serveryje virs į 10000 gijų, kurios tiesiog lauks (miegos) ant kiekvieno sujungimo nieko nedarydamos, tik švaistydamos išteklius. Net jei operacinė sistema ir TCP/IP talpa palaikys tai, mažai tikėtina, kad serverio programa taip pat tai galės. Visa laimė, kad nauji serveriai (*Grizzly*) ir pataisymai senų (*Jetty*, *Tomcat*, *Apache*) įgyvendina tai, tai vadinama asinchroniniu užklausos apdorojimu, kuris yra pagrįstas I/O neblokavimu. Toks mechanizmas randamas daugumoje OS ir bibliotekose.

Tai nėra tobulas sprendimas. Jei serveris stumia įvykius pakankamai greitai, galima atsidurti toje pačioje situacijoje, kaip su balsavimo modeliu, kuriame yra keletas jungčių ir didelė

užklausos/atsakymo ciklo perkrova. Kadangi abu modeliai kenčia nuo tinklo vėlinimo, ypač, kai jiems reikia siųsti naują užklausą kiekvienam atsakymui (įvykiui), kuris yra gautas.

Tai leidžia atsirasti trečiam modeliui, vadinamam HTTP *srautai*. Modelis panašus į ilgo balsavimo, bet neuždarinėja sujungimo, net tada, kai gauna atsakymą. Modelis rodomas 8 paveiksle. Šis triukas čia padaromas naudojant siuntimo mechanizmą, esantį HTTP ir vadinamą didės duomenų apimties persiuntimo užkodavimu, kuris leidžia siųsti atsakymą, sudarytą iš duomenų blokų, nežinant iš anksto duomenų kiekio ir kiekvieno bloko ilgio. Tai tiksliai tinka serverio įvykių serijoms, kurios turi būti nusiųstos klientui iš anksto nežinant įvykių skaičiaus, ilgio ar pačio svarbiausio dalyko, kada tie įvykiai įvyks. Šis modelis stipriai padeda suvaldyti tinklo naudojimą, nes yra eliminuojamas daugelio užklausų perteklius ir sumažinamas vėlinimas, nes atsakymas gali būti išsiųstas nelaukiant, kada pasibaigs užklausa.



8 pav. HTTP srautų sujungimo modelis

Net HTTP srautai nėra atleisti nuo neatidumo. Ne tik serveris turi palaikyti tūkstančius sujungimų su ribotu gijų skaičiumi fonde, didelio mastelio scenarijuje gali atsirasti per daug įvykių, kurie negalės būti teisingai perduoti klientam dėl tinklo perpildymo. Taigi turi būti apgalvotas tam tikras įvykių ribotuvas.

Žinoma yra taip pat keletas iššūkių, kurie turi būti aptarti. Pavyzdžiui, net su HTTP srautų sujungimo modeliu nėra galimybės klientui nusiųsti žinutę serveriui per atvirąjį kanalą. Šiek tiek ironiška, kad standartinis būdas bendrauti neveikia. Tačiau šiuo atveju tai yra daugiau *XMLHttpRequest JavaScript* kaltė, nes jos konstrukcijai reikia pilnos užklausos prieš pradedant persiuntimą.

5.4. Plečiamumo problemos

Nėra jokio platesnio darbo apie AJAX ir COMET įtaką našumui tinklo aplinkose, o egzistuojantis darbas turi tik išankstinius rezultatus. Net be plačių eksperimentų įrodymų vienas su vienu žymėjimas tarp sujungimų ir gijų neatrodo geriausia idėja. Ir ne tik COMET HTTP srautų ar ilgo balsavimo sujungimo modeliai pelnosi iš to. *J.F Arcand*, vienas iš inžinierių, dirbantis prie *Sun* kompanijos *Grizzly* serverio, atliko keletą sintetinių testų ir keletą realių testų naudojant *Grizzly* asinchroninių užklausų apdorojimo modelį, kuris remiasi *Java* neblokuojančia biblioteka *java.nio*. Statinių failų, JSP pralaidumas yra labai panašus, tačiau klasikiniam sujungimo elementui (*Tomcat Catalina*) reikia 500 gijų fondo, lyginant su 10 gijų fondu prie ARP sujungimo elemento. Didžiausio naudotojų skaičiaus, kurį serveris gali atlaikyti, su didžiausiu atsakymo laiku lygiu 2 sekundėms prie 90% užklausų ir vidutinišku galvojimo laiku lygiu 8 sekundėms, testavimas parodė, kad aiškus laimėtojas yra neblokuojantis modelis, nes buvo mažiau konteksto perjungimų ir daugiau prieinamos atminties su daug mažesniais gijų skaičiais, esančiais antrame modelyje.

5.5. COMET karkasai

Kadangi AJAX ir COMET technologijos tobulėja ir populiarėja, atitinkamai atsiranda didelis naujų karkasų skaičius. Iš tikrųjų AJAX karkasų skaičius didėja labai greitai turbūt todėl, kad tai yra nauja technologija ir rinka dar neatsirinko, kurie karkasai yra geriausi. Kaip bebūtų, labai mažas karkasų skaičius palaiko COMET, todėl bus atkreipiamas dėmesys į pačius populiariausius. Pirmiausia bus apžvelgtos kelios kūrimo bibliotekos, skirtos įgyvendinti COMET sprendimus. *Pushlets*, tai yra kombinuota biblioteka, kuri naudoja kliente *JavaScript* komponentus ir *Java Servlet* (technologija skirta dinaminiam turiniui generuoti) serverio pusėje. Pagrindinė sąvoka šioje technologijoje – *servlet*. Tai serveryje esantis komponentas, gaunantis iš kliento (pvz., naršyklės) užklausą (angl. *request*) bei generuojantis atsaką (angl. *response*). Kaip kitas pavyzdys, *Dojo* karkasas yra paprastesnis parašytas *JavaScript* kalba be serverio pusės komponentų. COMET dalis yra gaunama įgyvendinus *Bayeux* protokolą.

Grizzly, priklausantis *Sun* kompanijai yra tinklo konteineris, skirtas *JavaEE GlassFish* serveriui ir yra sukurtas remiantis asinchroniniu užklausų apdorojimu. Jetty yra labai populiarus *servlet* ir JSP konteineris, kuris buvo vienas iš pirmųjų (jei ne pats pirmas), įgyvendinęs sprendimą su savo pratęsimo (angl. continuation) mechanizmu. Naujausia *Apache Tomcat* 6 versija turi 2 sekundžių atsako laiką ir ARP sujungimo mechanizmą.

5.6. *Dojo* ir *Cometd* karkasai

Kita svarstomų karkasų kombinacija yra kliento pusės *Dojo* ir serverio pusės *Cometd* karkasai, kurie kartu palaiko brukimu pagrįstą kliento ir serverio bendravimą. Karkasas yra pagrįsta *Bayeux* protokolu, kurį *Cometd* grupė neseniai išleido, kaip atsakymą į atsiradusį komunikacijos standartų trūkumą.

BAYEUX žinutės formatas yra aprašytas JSON (*JavaScript* objektų notacija), kuris yra duomenų mainų formatas pagrįstas *JavaScript* programavimo kalbos rinkiniu. Protokolas buvo neseniai sukurtas ir įdėtas į daug tinklo serverių, taip pat ir *Jetty* ir IBM *Websphere*.

Karkasai, kurie aprūpina BAYEUX, suteikia sujungimo tipą, kuris vadinamas ilgas balsavimas (angl. *Long Polling*). Serveris užlaiko kliento prašymą, kol duomenys netampa pasiekiami. Jei įvykis atsiranda, serveris siunčia duomenis klientui ir klientas turi jungtis iš naujo, kitaip serveris užlaiko ryšį baigtiniam laikotarpiui, po kurio jis paprašo, kad klientas jungtųsi iš naujo. Jei duomenų publikavimo intervalas yra mažas, sistema veiks kaip grynas *traukimas*, nes klientai turės dažnai jungtis (padaryti prašymą) iš naujo. Jei duomenų publikavimo intervalas yra didelis, tai sistema veiks kaip grynas *brukimo* mechanizmas.

BAYEUX apibrėžia tokius etapus, kurie yra reikalingi ryšiui pagal COMET technologiją nustatyti. Klientas:

- Atlieka rankų paspaudimą su serveriu ir gauna kliento ID (identifikatorius).
- Siunčia sujungimo prašymą su jo ID,
- Užsiprenumeruoja kanalą ir gauna atnaujintus duomenis.

BAYEUX palaikomas kliento pusės AJAX karkaso vadinamo *Dojo*. Jis pilnai parašytas *JavaScript* kalba ir yra planų netolimoje ateityje pritaikyti jam ir žymėjimo (angl. *markup*) kalbą. *Dojo* suteikia daug naudojimui paruoštų naudotojo sąsajos priemonių, kurios yra nesupakuoti *JavaScript* kodo komponentai, taip pat suteikia ir abstraktų sujungimą (angl. *wraper*) (*dojo.io.bind*), tinkantį daugeliui naršyklėse XMLHttpRequest objekto įgyvendinimų, skirtų bendrauti su serveriu. *Dojo* supaprastinta *dojo.io.cometd* biblioteka, skirta atlikti rankų paspaudimą jungiantis ir atitinkamam kanalui užsakyti.

Serverio pusėje BAYEUX yra palaikomas *Cometd*. Tai yra HTTP pagrįstas įvykių kaupimo karkasas, kuris naudoja COMET bendravimo stilių. Šiuo metu jis įgyvendintas kaip *Jetty* modulis.

5.7. Kliento bibliotekos

Pushlets yra serverio programomis pagrįstas (*angl. servlet-based*) mechanizmas, kai duomenys yra tiesiogiai stumiami iš serverio pusės *Java* objektų į (dinaminius) HTML puslapius, esančius kliento naršyklėje nenaudojant *Java applets* arba *plug-ins*. Tai leidžia serveriui periodiškai atnaujinti internetinį puslapį. Šis naršyklės klientas naudoja *JavaSripti*/dinaminio HTML savybes, prieinamas 4+ versijos naršyklėse, tokiose kaip NS (*Netscape*) ir MSIE (*Internet Explorer*). Pagrindinis mechanizmas naudoja *servlet* HTTP sujungimą, per kurį *JavaSripti* kodas yra stumiamas į naršyklę. Per vieną bendrą s*ervlet* (*pushlet*) naršyklės klientai gali užsisakyti, iš kur jie nori gauti įvykius. Kai serveris stumia įvykį, klientai, prašę atitinkamo šaltinio, yra informuojami. Įvykių objektai gali būti siunčiami kaip *JavaScript* (DHTML klientai), serijiniai *Java* objektai (*Java* klientai) arba XML (DHTML arba *Java* klientai).

Dojo priemonių rinkinys modulinis atvirojo kodo *JavaScript* priemonių rinkinys (ar biblioteka), skirtas palengvinti greitą *JavaScript* ar AJAX pagrįstų programų ir internetinių svetainių kūrimą. Tai pradėjo Alex Russell 2004 metais su dviguba licencija, iš jų viena yra BSD licencija, antra – akademinė nemokama licencija. BSD licencija – viena iš licencijų, skirtų atvirojo kodo programinei įrangai. Pradžioje ji buvo naudojama Berklio programinės įrangos *Unix* tipo operacinei sistemai. Iš to kilo licencijos pavadinimas. Pirmoji versija buvo pakoreguota ir taip atsirado šiuo metu plačiai naudojama BSD licencijos versija. *Dojo Faundation* yra ne pelno organizacija sukurta palengvinti priemonių rinkinio prisitaikymą. Alex Rusell pirmas panaudojo COMET terminą ir yra vienas iš žmonių, dirbančių su *Bayeux* protokolu.

5.8. Serverio sprendiniai

Grizzly yra HTTP serverio komponentas, skirtas naujam *Sun JavaEE Glassfish* serveriui. *Grizzly* apibūdinimas, pagal vieną iš jo kūrėjų, skamba taip: *Grizzly* buvo sukurtas, kad dirbtų ant *Apache Tomcat Coyote* HTTP sujungimo viršaus. *Coyote* sujungimas yra naudojamas *Tomcat* 3/4/5 versijose ir įrodė, kad yra labai našus HTTP sujungimas, kai matuojamas grynas pralaidumas. Tačiau kaip kiti *Java* pagrįsti HTTP sujungimai, plečiamumas yra apribotas esamų gijų skaičiumi ir kai reikia laikyti giją gyvą, jis nukenčia nuo vienos gijos sujungimui paradigmos. Dėl to plečiamumas daugiausia laiko yra ribojamas didžiausio platformos gijų skaičiaus. Norint išspręsti šią problemą, žmonės dažnai pastato *Apache* pirmiau nei *Java* arba naudoja grupę užklausų, paskirstytų tarp kelių *Java* serverių. *Grizzly* skiriasi nuo *Coyote*

dviejose srityse. Pirmiausia, *Grizzly* leidžia bet kokios rūšies gijų telkinio prijungimą (šiuo metu darbo erdvėje galimos trys). Antra, Grizzly palaiko du režimus: tradicinį IO ir neblokuojantį IO (Baila, Beltran 2007).

Jetty yra 100% tik *Java* paremtas HTTP serveris ir *Servlet* konteineris. Jetty yra sukurtas kaip atvirojo kodo projektas su *Apache* 2.0 licencija. *Jetty* yra naudojamas kelių kitų populiarių projektų, tokių kaip *JBoss* ir *Geronimo* programinis serveris. Šis serveris tikriausiai buvo pirmasis, sulaužęs vienos gijos per užklausą žymėjimą su savo *Continuations* technologija ir tiekė įvairius COMET serverio karkasus net tada, kai pati koncepcija dar nebuvo aiški.

Apache Tomcat yra tinklo konteineris, sukurtas *Apache Software Foundation* (ASF) kompanijos. *Tomcat* įgyvendina *servlet* ir *JavaServer* puslapių (JSP) iš *Sun Microsystems* specifikacijas, ir tiekia aplinką *Java* kodui, kad jis galėtų veikti kartu su interneto naršykle. Jis suteikia priemones, skirtas derinti ir valdyti, bet taip gali būti pritaikytas keisti derinimo failams, kurie dažniausiai yra suformatuoti XML formatu. *Tomcat* turi savo vidinį HTTP serverį. Nuo 6 versijos, *Tomcat* palaiko NIO HTTP sujungimą ir turi prigimtinį COMET palaikymą.

6. SPIAR architektūros stiliaus kūrimas

AJAX programos gali būti matomos kaip darbalaukio ir tinklo programų hibridas, paveldėjęs savybes iš abiejų pasaulių. 2 lentelėje susumuojami skirtumai tarp to, ką suteikia REST architektūra ir ko pageidauja šiuolaikinės AJAX (su COMET) programos (Mesbah, Deursen 2008). AJAX karkasai aprūpina klientą, viename gale esančiomis paslaugomis, per naudotojo sąsajos komponentus, įvykiais valdomu arba brukimo stiliumi. Tokia architektūra nėra taip lengvai gaunama REST architektūroje dėl šių skirtumų: kai REST tinka didelės duomenų apimties siuntimui dėl jo vieningos sąsajos apribojimų, tai nėra optimalu siunčiant mažos apimties duomenis, reikalingus AJAX programose. REST sutelkta ties susietais ištekliais pagrįstu bendravimu, kai klientas prašo specifinių išteklių. Priešingai AJAX programose, naudotojas bendrauja su sistema panašiai kaip darbalaukio programose, prašydamas atsakymo į specifinį veiksmą. Visa sąveika REST architektūroje, skirta išteklių atvaizdavimui gauti, yra įvykdoma per sinchroninę prašymo – atsakymo porą. Tačiau AJAX programos reikalauja asinchroninio komunikacijos modelio. REST aiškiai trukdo serveriui turėti būseną, t.y. kiekvienas kliento prašymas turi turėti visą informaciją, būtiną serveriui, kad jis suprastų prašymą. Kai šis apribojimas gali pagerinti prieinamumą, noras atsižvelgiant į tinklo našumą ir naudotojų interaktyvumą projektuojant AJAX architektūrą turi didesnę reikšmę (Mesbah, Deursen 2008). REST yra pagrįsta tarpine atmintimi, o AJAX atlieka duomenų paiešką realiuoju laiku. Kiekvienas prašymas turi būti inicijuotas kliento, ir kiekvienas atsakymas turi būti generuojamas nedelsiant, kiekvienas prašymas gali sukurti tiktai vieną atsakymą. COMET reikalauja modelio, kuris įgalintų brukti duomenis klientui iš serverio.

2 lentelė. REST paslaugos ir AJAX reikalaujamos paslaugos

REST suteikia	AJAX reikalauja
Didelės duomenų apimties siuntimas	Mažos duomenų apimties siuntimas
Pagrįsta ištekliais	Komponentais pagrįsta naudotojo sąsaja
Susiejimas nuorodomis	Pagrįsta veiksmo-įvykio modeliu
Sinchroniškas prašymas-atsakymas	Asinchroninis bendravimas
Būsenos nebuvimas	Būsenos
Tarpinis kaupimas	Duomenų gavimas realiuoju laiku
Traukimo metodas	Traukimo ir brukimo metodai

Šie reikalavimai rodo, kad reikia naujo architektūros stiliaus, gebančio suteikti pageidaujamas ypatybes.

Norint sukurti architektūros stilių, reikia išanalizuoti pageidaujamas savybes ir nustatyti būtinus architektūrinius apribojimus.

6.1. Teorinis modelis

6.1.1. Architektūros savybės

Toliau aptariamos architektūros savybės, kurios yra susijusios AJAX esme. Į kitas ypatybes, – plečiamumas ar saugumas, – kurios gali būti pageidautinos bet kokiai sistemai, bet yra mažiau tiesiogiai veikiamos AJAX, neatsižvelgiama. Kai kurios iš aptartų ypatybių yra susietos viena su kita: pavyzdžiui, naudotojų interaktyvumas priklauso nuo naudotojui suprantamo vėlinimo, o vėlinimas priklauso nuo tinklo našumo.

6.1.1.1. Naudotojo interaktyvumas

Žmogaus ir kompiuterio sąveika literatūroje apibrėžia interaktyvumą kaip laipsnį, iki kurio komunikacijos proceso dalyviai turi jos valdymą ir gali keistis vaidmenimis bendraudami. Naudotojų interaktyvumas yra artimai susietas su naudojimo patogumu. Šis terminas vartojamas literatūroje, kurioje kalbama apie programinės įrangos architektūrą. Didesnis interaktyvumo lygmuo turi teigiamą įtaką naudotojo suprantamam pasitenkinimo, efektyvumo, naudingumo, vertės ir visa apimančio požiūrio į internetinę svetainę rezultatams. Šių savybių tinklo gerinimas buvo svarbiausias akstinas, skatinantis AJAX.

6.1.1.2. Naudotojui suprantamas vėlinimas

Naudotojui suprantamas vėlinimas yra apibrėžiamas kaip periodas tarp momento, kai naudotojas atlieka prašymą ir pirmo atsakymo iš sistemos požymio. Apskritai yra du pagrindiniai būdai naudotojui suprantamam našumui pagerinti. Pirma, mažinant kelionės pirmyn ir atgal laiką ir antra, leidžiant naudotojui asinchroniškai dirbti su sistema. Tai yra svarbi ypatybė visose paskirstytose programose, kuriose naudotojas pirmas atlieka veiksmą.

6.1.1.3. Tinklo našumas

Tinklo našumas yra lemiamas pralaidumo, kuris yra duomenų perdavimo tinklo ir juostos pločio santykis, t.y. didžiausias pasiekiamo pralaidumo matas. Tinklo našumas gali būti pagerintas mažinant visų duomenų apimtį ir perduodamų duomenų dalių apimtį.

6.1.1.4. Paprastumas

Paprastumo ar kūrimo pastangos yra apibrėžtos kaip pastangos, kurios yra būtinos suprasti, suprojektuoti, įgyvendinti ir iš naujo suprojektuoti tinklo programą. Tai yra svarbus naudojimo ir bet kokio naujo metodo priėmimo veiksnys.

6.1.1.5. Mastelio keičiamumas

Paskirstytose aplinkose mastelio keičiamumas yra apibrėžtas sistemų gebėjimo laipsniu prisitaikyti prie didėjančio komponentų skaičiaus. Projektuojant tinklą, sistemos mastelio keičiamumas yra nustatytas laipsniu, pagal kurį klientas gali būti aptarnautas skirtingų serverių, nepaveikiant rezultatų. Plečiama tinklo architektūra gali būti lengvai pritaikyta didėjančiam klientų prašymų skaičiui aptarnauti.

6.1.1.6. Mobilumas

Programinė įranga, kuri gali būti naudojama skirtingose aplinkose, yra mobili. Tinklo gebėjimas panaudoti tinklo naršyklę be jokių papildomų veiksmų, reikalingų iš naudotojo pusės, pavyzdžiui, papildinių parsiuntimas, reiškia mobilumo ypatybę.

6.1.1.7. Matomumas

Matomumas yra nustatomas laipsniu, pagal kurį išorinis tarpininkas sugeba suprasti sąveiką tarp dviejų komponentų. Kuo lengviau tarpininkui suprasti tą sąveiką, tuo matomesnė yra sąveika tarp tų dviejų komponentų. Vertinant dabartinių AJAX karkasų įgyvendinimą, matomumas kliento/serverio sąveikoje yra blogas, kadangi jie yra pagrįsti privačiais protokolais.

6.1.1.8. Duomenų koherentiškumas

Svarbus informavimo apie realaus laiko įvykius aspektas yra tinklo duomenų, kurie turi būti prieinami ir perduoti naudotojui iškart, kai jie įvyksta, pavyzdžiui, prekių kainos, koherentiškumo palaikymas. Duomenų dalis yra laikoma koherentiška, jei serverio ir kliento duomenys yra sinchronizuoti. Tinklo programose, naudojančiose HTTP protokolą, klientams dažnai reikia traukti duomenis, pagrįstus iš anksto nustatytu intervalu. Priešingai serveriai, kurie pritaiko brukimo galimybę, palaiko būsenos informaciją, priklausančią klientams, ir siunčia

srautu pokyčius naudotojams, kai tik atsiranda įvykis. Šios dvi technikos turi skirtingų savybių atsižvelgiant į pasiekiamą duomenų koherentiškumą.

6.1.1.9. Pritaikomumas

Pritaikomumas (angl. *adaptability*) yra apibrėžiamas tuo, kaip lengvai sistemas ar sistemų dalis galima pritaikyti prie besikeičiančios aplinkos. Tinklo programų architektūra, kuria galima pokyčius serveryje perduoti klientams, vadinama prisitaikančia architektūra. Kodo mobilumo terminas vartojamas siekiant palyginti skirtingų AJAX architektūrų dinamines būsenas pagal keičiamumo ir pritaikomumo galimybes. Mobilusis kodas iš esmės yra programinės įrangos kodas, gautas iš tolimų serverių, persiųstas tinklu ir tada atsiųstas ir paleistas kliento be jokio išorinio diegimo ar gavėjo paleidimo.

6.1.1.10. Patikimumas

Patikimumas yra apibrėžiamas, kaip tęstinis teisingas paslaugos tiekimas. Bet kokios programinės įrangos sėkmė stipriai priklauso nuo jos patikimumo. Internete, tinklo programos, priklausančios nuo nepatikimos programinės įrangos ir gerai neveikiančios, praras pirkėjus. Išteklių testavimas (testų automatizavimas, visumos ir regresijos testavimas) gali padidinti programos patikimumo lygį. Tačiau tinklo programos žinomos kaip sunkiai testuojamos lyginant su tradicinėmis darbalaukio programomis. Papildomai, dėl trumpo praeinančio laiko tarpo, kol paslauga pasiekia rinką ir dėl to atsirandančio spaudimo, daugelio interneto puslapių bendravimo stilius daro testavimą sudėtingu. Prisitaikant vieno puslapio komponentais pagrįstą tinklo programų kūrimo stilių galima pagerinti sistemos testavimą ir jos patikimumą.

6.1.2. Architektūriniai apribojimai

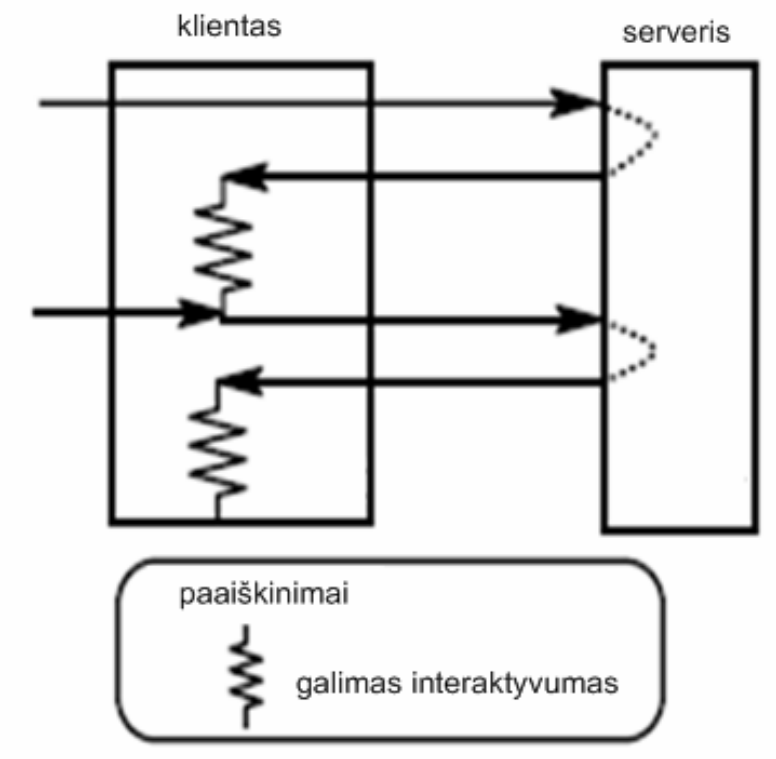
Architektūriniai apribojimai gali būti panaudoti kaip architektūrinių elementų vaidmenų apribojimai, norint suteikti pageidaujamas sistemos architektūrines ypatybes. 3 lentelėje rodomas apribojimų ir reikalingų ypatybių bendras vaizdas. “+” žymi tiesioginį teigiamą rezultatą, o “–” rodo tiesioginį neigiamą efektą. SPIAR priklauso nuo tam tikrų apribojimų, turinčių išsaugoti naudingas pageidaujamas savybes.

3 lentelė. Architektūriniai apribojimai ir reikalingos ypatybės

	Naudotojo interaktyvumas	Naudotojui matomas vėlinimas	Tinklo našumas	Paprastumas	Mastelio keičiamumas	Mobilumas	Matomumas	Duomenų koherentiškumas	Patikimumas	Pritaikomumas
Vieno puslapio sąsaja	+									
Asinchroninis bendravimas	+	+								
Delta komunikacija	+	+	+		–		–	+		
Kliento pusės apdorojimas	+	+	+							
Komponentais pagrįsta naudotojo sąsaja	+			+					+	+
Pagrįsta tinklo standartais				+		+			+	
Būsenos	+	+	+		–		–			
Brukimu pagrįstas publikavimas / prenumeravimas		+	+		–		–	+		+

6.1.2.1 Vieno puslapio sąsaja

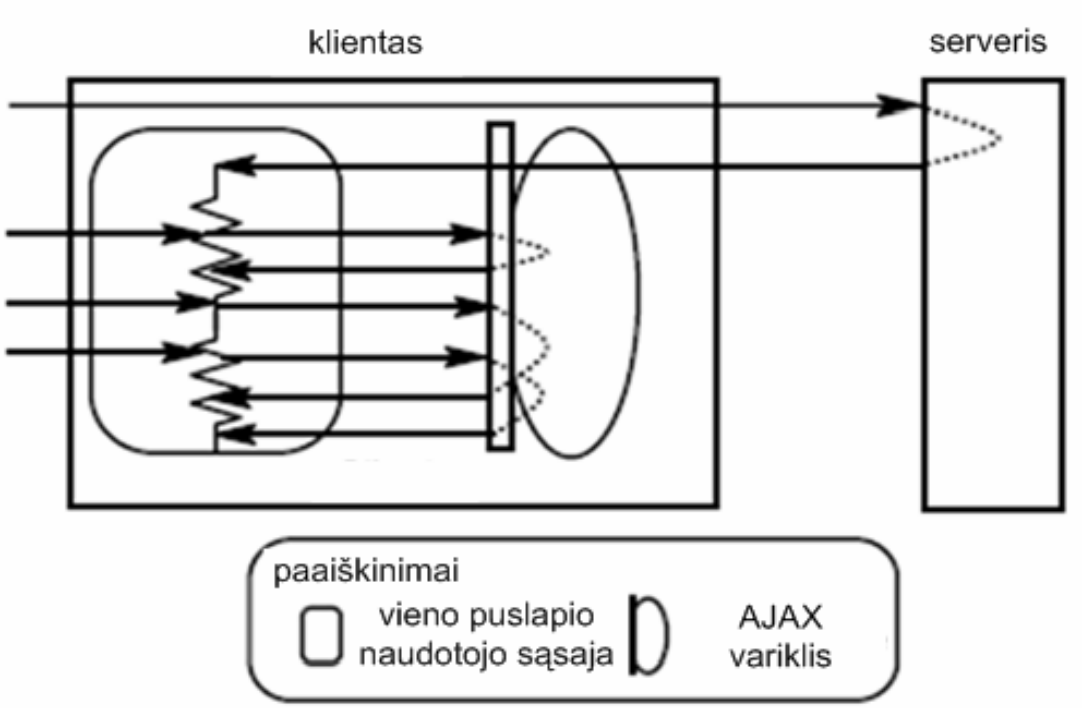
SPIAR pagrįstas kliento-serverio stiliumi, kuris yra geriausiai žinoma architektūra skirta paskirstytoms programoms, kurios naudojasi tinklo aplinkoje naudojamu atskirtų rūpesčių principu. Pagrindinis apribojimas, kuris atskiria šį stilių nuo tradicinės interneto tinklo architektūros, yra jo tikslas naudoti vieno puslapio modelį vietoj puslapių sekos modelio. Šis apribojimas sukuria naudotojo interaktyvumo savybę. Naudotojo interaktyvumas pasidaro daug geresnis, nes bendravimas vyksta komponentų lygyje ir naudotojas neturi laukti, kol visas internetinės svetainės puslapis bus iš naujo sugeneruotas, kaip atsakas į kiekvieną naudotojo veiksmą. 9 ir 10 paveikslai rodo bendravimo stilių, koks yra tradiciniame tinkle ir atitinkamai koks yra vieno sukoncentruoto į klientą puslapio AJAX programoje.



9 pav. Tradicinis kelių puslapių bendravimas

6.1.2.2. Interaktyvumas ir sinchroniškumas

Kliento ir serverio sąveika gali būti įgyvendinta abiem brukimo arba traukimo būdais paremtu stiliumi. Brukimu pagrįstame stiliuje, serveris transliuoja būsenos pokyčius klientams, asinchroniškai kiekvieną kartą, kai jo būsena pasikeičia. Įvykiais pagrįsta integracija ir asinchroninis REST yra įvykiais pagrįsti stiliai, leidžiantys asinchroniškai pranešti apie serverio būsenos pokyčius. Šis sąveikos būdas yra daugiausia palaikomas taškas į tašką architektūros aplinkose. Traukimu pagrįstame stiliuje kliento komponentai patys aktyviai prašo būsenos pasikeitimų.



10 pav. AJAX bendravimas centruojantis į klientą

Įvykiais valdomos architektūros yra sutinkamos paskirstytose programose, kurios reikalauja asinchroninės komunikacijos, pavyzdžiui, darbalaukio programos, kur naudotojo pradėtos UI įvestys tarnauja kaip įvykiai, kurie suaktyvina procesą. AJAX programos yra suprojektuotos, kad palaikytų gerą naudotojų interaktyvumą ir blogą naudotojui suprantamą vėlinimą. Asinchroninė sąveika leidžia naudotojui bet kuriuo metu inicijuoti prašymą serveriui ir tučtuojau gauti valdymą atgal iš kliento. Prašymai yra tvarkomi kliento puseje, fone, ir sąsaja yra atnaujinama pagal serverio atsakymus. Šis sąveikos modelis yra iš esmės kitoks negu klasikinis sinchroninis prašyk, lauk atsakymo ir tęsk modelis.

6.1.2.3. Delta komunikacija

Nereikalingas duomenų persiuntimas, kuris yra daugiausiai priskiriamas prie nepakitusių puslapių persiuntimo iš naujo yra vienas iš klasikinių tinklo programų apribojimų. Daugybė technikų tokių kaip tarpinis kaupimas, įgalioti serveriai ir fragmentais pagrįstų išteklių pasikeitimų įvertinimas ir sumažinimas, buvo pritaikyti, siekiant sumažinti nereikalingų duomenų kiekį. Delta kodavimas naudoja tarpinio kaupimo techniką, kad sumažintų tinklo eismą, tačiau, tai nemažina skaičiavimo naštos, kadangi serveris kuria iš naujo visą puslapį kiekvienam prašymui.

SPIAR žengia vienu žingsniu toliau, ir sąveikai naudoja delta komunikacijos stilių. Čia paprasčiausiai būsenų pokyčiais mainosi klientas ir serveris priešingai, nei viso puslapio persiuntimo metode klasikinėse tinklo programose. Delta komunikacija yra pagrįsta delta

kodavimo architektūriniais principais, bet yra skirtinga: delta komunikacija nesiremia tarpiniu kaupimu, todėl klientui užtenka tik gaminti delta žinutes. Visi AJAX karkasai slepia delta komunikacijos detales nuo svetainės kūrėjų. Šis apribojimas tiesiogiai suteikia papildomas tinklo našumo ypatybes ir kaip padarinys yra naudotojo jaučiamas mažesnis vėlinimas ir geresnis interaktyvumas. Tinklo našumas yra pagerinamas todėl, kad yra transportuojama mažiau nereikalingų duomenų (paprasčiausiai delta).

6.1.2.4. Komponentais pagrįsta naudotojo sąsaja

SPIAR remiasi vieno puslapio naudotojo sąsaja su komponentais panašiais į esančius darbalaukio programose, pavyzdžiui, AWT naudotojo sąsajos komponentinis modelis. Šis modelis apibrėžia naudotojo sąsajos komponentų būseną ir elgesį, bei kaip jie gali kartu veikti. Naudotojo sąsajos komponentų programavimas daro kūrimą paprastesniu, nes svetainių kūrėjai gali panaudoti daugkartinio naudojimo komponentus, kad pagamintų internetinę svetainę deklaratyviai ar programiškai. Naudotojų interaktyvumas yra pagerintas, nes naudotojas gali bendrauti su programa komponentų lygmenyje, panašiai kaip ir darbalaukio programose.

6.1.2.5. Standartais pagrįstas tinklas

Tinklo elementų apribojimas iki standartizuoto formatų komplekto yra būdas pasiekti tinklo mobilumui. Šie apribojimai neapima tokių metodų, kuriems reikia papildomo funkcionalumo (pavyzdžiui papildiniai, virtuali mašina), kad veiktų tinklo naršyklėje, tokie kaip *Flash* ir *Java* priedai, kurie daro skirtingas klientų naršykles suderinamomis. Šie apribojimai riboja duomenų elementų kiekį iki tų, kurie yra palaikomi tinklo naršyklių.

6.1.2.6. Apdorojimas kliento pusėje

Apdorojimas kliento pusėje pagerina naudotojų interaktyvumą ir sumažina naudotojui suprantamą vėlinimą per duomenų kelionės pirmyn ir atgal sutrumpinimą. Pavyzdžiui, kliento pusėje atliekamas formos patvirtinimas mažina nereikalingus serverio pusės klaidos pranešimus ir iš naujo įvesti duomenis prašančias žinutes. Papildomai, kai kuris serverio pusės apdorojimas (pavyzdžiui, elementų rūšiavimas) gali būti užkrautas klientams, naudojant mobilų kodą, kuris pagerins serverio našumą ir padidins prieinamumą papildomiems susijungimams. Kaip problema, gali tapti kliento našumas, jei daug funkcijų reikalaus išteklių kliento pusėje. GWT karkasas pilnai pasinaudoja kliento pusės apdorojimo galia, generuodamas visą naudotojo sąsajos kodą priklausantį kliento pusei kaip *JavaScript* ir paleisdamas jį kliento kompiuteryje.

6.1.2.7. Būsenos

Neturintis būsenos serveris yra toks, kuris elgiasi su kiekvienu prašymu kaip su nepriklausomu siuntimu, nesusijusiu su bet kokiu ankstesniu prašymu t.y., kiekvienas prašymas turi turėti visą informaciją, būtiną jam suprasti ir negali pasinaudoti jokiu iš anksto serveryje išsaugoto konteksto privalumu. Net nepaisant to, kad tinklo architektūra ir HTTP yra suprojektuoti taip, kad neturėtų būsenų, sudėtinga galvoti apie tinklo programas be būsenų. Tinklo programos vidaus, sąveikų tvarka yra susijusi. Sąveikos priklauso viena nuo kitos, tai reikalauja visapusiško komponentų topologijos supratimo. Įgyvendintos būsenos yra HTTP, kliento pusės *sausainių* ir serverio pusės sesijos valdymo kombinacijų mėgdžiojimas.

Skirtingai nuo REST, SPIAR aiškiai nevaržo būsenos prigimties. Vis dėlto, metodas be būsenos gali sumažinti tinklo našumą (didindamas pasikartojančių duomenų kieki) ir dėl komponentais pagrįstos naudotojų sąveikų prigimties, būsenų įgyvendinimo sprendimas galėtų tapti pageidaujamas dėl keičiamumo ir matomumo privalumų.

6.1.2.8. Brukimu pagrįstas publikavimas/prenumeravimas

Kliento-serverio bendravimas gali būti realizuotas abiem – brukimu ar traukimu (angl. *pull*) pagrįstais stiliais. Brukimu pagrįsto stliaus atveju serveris transliuoja būsenos pokyčius klientams asinchroniškai, kai tik esama būsena pasikeičia. Įvykiais pagrįsta integracija ir asinchroninis REST yra įvykiais pagrįsti stiliai, leidžiantys serveriui asinchroniškai informuoti būsenos pokyčius. Šis interaktyvumo stilius iš esmės yra taškas į tašką (angl. *peer-to-peer*) architektūros aplinkose.

Traukimu pagrįstame stiliuje kliento komponentai aktyviai prašo būsenos pokyčių. Įvykiais valdomos architektūros yra randamos paskirstytose programose, kurios reikalauja asinchroninio bendravimo, pavyzdžiui darbalaukio programos, kur naudotojo inicijuotos įvestys naudotojo sąsajoje tarnauja kaip įvykiai aktyvuojantys procesus.

COMET leidžia mums mėgdžioti brukimu pagrįstą publikavimo/prenumeravimo stilių, skirtą bendrauti tinkle. Ši savybė pagerina tinklo našumą, nes yra išvengiama nereikalingų balsavimo (angl. *poll*) užklausų. Naudotojui suprantamas vėlinimas ir pritaikomumas yra taip pat gerinami leidžiant realaus laiko įvykių apie būsenos pokyčius informavimą klientams. Duomenų koherentiškumas yra taip pat žymiai pagerinamas dėl šio apribojimo, tačiau tuo pačiu metu serverio programos našumas ir patikimumas gali pablogėti, todėl neigiamai lemiamas plečiamumas (angl. *scalability*).

6.2. Architektūros stiliaus įgyvendinimas

Įgyvendintas SPIAR architektūros stilius bus tada, kai bus nubraižyta jo schema ir aprašyti jos elementai bei veikimas.

6.2.1. SPIAR veikimo principai

Atsižvelgiant į apdorojimą, duomenis ir sujungtus elementus, galima panaudoti skirtingus architektūrinius vaizdus, kad apibūdinti, kaip kartu dirba elementai formuodami architektūrą. Čia naudojamas apdorojimo vaizdas, kuris koncentruojasi ties duomenų srautu ir tam tikrais ryšių aspektais tarp apdorojimo elementų, atsižvelgiant į duomenis. Šis vaizdas tinka komponentų ir sujungimų vaizdo tipui. 2 iliustracija vaizduoja SPIAR – pagrįstos architektūros veikimo vaizdą, kuri remiasi paleidžiamaisiais komponentais, vaizduojamais taip kaip, pavyzdžiui, *Echo2* karkase. Vaizde matyti skirtingų komponentų sąveika tuo laiku, kai atliktas pradinio puslapio užkrovimo prašymas (variklis veikia kliento pusėje). Naudotojų veikla naudotojų sąsajoje sukelia įvykį, kad jis parodytų tam tikrą komponentais apibrėžtą veiksmą, kuris yra deleguotas AJAX varikliui. Jei klausimo elementas serverio komponento pusėje užregistravo save su įvykiu, variklis padarys DELTA-KLIENTO esamos būsenos pokyčių žinutę su atitinkamais įvykiais ir nusiųs jį į serverį. Serverio pusėje dekoderis atkoduos žinutę, identifikuos ir nustatys tinkamus komponentus komponentų medyje. Pakeisti komponentai galutinai sužadins paslaugos serviso įvykių klausimo elementus. Paslaugų teikėjas, po veiksmų apdorojimo, atnaujins atitinkamus komponentus nauja būsena, kuri bus sugeneruota enkoderiu. Sugeneruotą DELTA-SERVERIO žinutė bus nusiunčiama atgal į variklį, kuris bus panaudotas, kad atnaujintų atvaizdavimo modelį ir galų gale sąsają. Variklis taip pat turi gebėjimą atnaujinti atvaizdavimo modelį iškart po įvykio, jei nėra reikalinga papildoma kelionė pirmyn ir atgal į serverį (Mesbah, Deursen 2006).

6.2.2. SPIAR architektūriniai elementai

SPIAR architektūros stilius, tai naujas šablonas, apimantis reikalingiausias savybes, kurias AJAX architektūros karkasai turi turėti ir nurodo apribojimus, prie kurių tokie karkasai turi prisitaikyti. Stilius gali būti panaudotas tinklo pogramose, kai yra pageidaujamas geras interaktyvumas ir atsakomumas.

Pagrindiniai architektūriniai SPIAR elementai yra padalyti į tris kategorijas: apdorojimą, duomenis ir jungiamuosius elementus.

6.2.2.1. Apdorojimo elementai

Apdorojimo elementai yra apibrėžti kaip tie komponentai, kurie transformuoja duomenų elementus.

Kliento naršyklė siūlo palaikymą tokių standartų rinkiniui, kaip: HTTP, HTML, kaskadinių stilių lentelės, *JavaScript*, ir dokumento objekto modelis. Ji apdoroja atvaizduojamąjį tinklalapio modelį, kad sukurtų naudotojo sąsają. Naudotojo sąveika gali būti grindžiama vienu naudotojo puslapio sąsajos modeliu. Visas vaizdas ir efektai yra pristatomi naudotojui per šią sąsają. Taip pat kaip darbalaukio programa, ji susideda iš vieno svarbiausio puslapio su priedėlių (angl. *widget*) komplektu. Priedėlių ypatybės gali būti valdomos individualiai tuo pačiu metu, kai daromi pakeitimai, ir nereikalauja puslapio perkrovimo.

AJAX *variklis* yra klientinė programa, kuri kraunasi ir veikia kliento naršyklėje ir nereikalauja jokio papildinio, kad tinklo programa galėtų normaliai funkcionuoti. Tačiau, variklio duomenų perkėlimas tikrai sukuria pradinį duomenų vėlavimą naudotojui, kuris gali būti kompensuotas mažos duomenų apimties siuntimais, kai variklis jau yra savo vietoje.

Variklis yra atsakingas už pristatomojo modelio inicijavimą ir manipuliavimą. Jis tvarko naudotojo pradėtus įvykius, bendrauja su serveriu ir gali atlikti apdorojimą kliento pusėje.

Serverio programa yra serveryje ir veikia, priimdama HTTP pagrįstus prašymus iš tinklo, bei aprūpina užsakovus atsakymais. Visas serverio pusės funkcionalumas yra serverio programos apdorojimo elementuose.

Paslaugų teiktuvas atstovauja serverio logikos variklį ir apdoroja būsenų pokyčius bei naudotojo prašymo veiksmus. Jis geba gauti prieigą prie bet kokių išteklių (pavyzdžiui, duomenų bazė, žiniatinklio paslaugos) reikalingų atlikti tiems veiksmams. Paslaugos tiektuvo funkcionalumas yra įgyvendintas per įvykių klausimo elementus, prijungtus prie komponentų, kurie yra inicijuoti įeinančių prašymų.

Delta koduotuvas/dekoderis apdoroja išeinančias/įeinančias delta žinutes. Šiame punkte, komunikacijos protokolas tarp kliento ir serverio yra apibrėžtas ir paslėptas už sąsajos. Šis elementas palaiko delta komunikaciją tarp kliento ir serverio, tai sumažina naudotojui suprantamą vėlinimą ir pagerina tinklo našumą.

UI komponentai susideda iš serverio pusės UI (naudotojo sąsajos) komponentų komplekto. Komponentinis serverio modelis geba sugeneruoti kliento atstovaujamąjį modelį. Kiekvienas serverio pusės komponentas turi duomenis ir elgesį tos dalies atitinkamo kliento priedėlio, kuri atlieka tuos būsenos pokyčius. Yra skirtingi būdai kaip sugeneruoti kliento pusės naudotojo sąsajos kodą. GWT, pavyzdžiui, generuoja (angl. *renders*) visą kliento pusės UI kodo surinkimo laiką, paimdamas viską iš serverio pusės *Java* komponentų. *Echo2*, iš kitos pusės, generuoja

komponentus realiuoju laiku ir palaiko komponentų medžius kliento ir serverio pusėse. Šitie UI komponentai turi įvykių klausimo elementus, kurie gali būti prijungti prie kliento pusės naudotojo inicijuotų įvykių tokių kaip mygtuko paspaudimas. Šis elementas didina paprastumą ir aprūpina surinktais komponentais, kurie reikalingi tinklo programoms kurti.

6.2.2.2. Duomenų elementai

Duomenų elementai turi informaciją, kuri yra naudojama ir transformuojama apdorojimo elementų.

Pristatomasis elementas susideda iš bet kokios terpės tipo, taip pat, kaip REST. HTML, CSS ir paveiksliukai visi yra šio duomenų elemento nariai.

Pristatymo modelis yra veikimo laiko abstrakcija, rodanti kaip kliento naršyklėje yra atstovaujama naudotojo sąsaja. Dokumento objektinis modelis naršyklėje įgijo labai svarbų vaidmenį AJAX programose. Būtent per dinamišką manipuliavimą šis pristatymo modelis padarė įvairiausius efektus įmanomais. Kai kurie karkasai, kaip: *Backbase*, naudoja sričiai specifinę kalbą, kad deklaratyviai apibrėžtų pristatomojo modelio struktūrą ir elgesį. Kiti kaip GWT naudoja tiesioginį metodą, panaudodami *JavaScript*.

Delta komunikacinės žinutės formuoja delta komunikacijos protokolo reikšmes tarp kliento ir serverio. SPIAR atskiria kliento delta duomenis (DELTA-KLIENTAS) nuo serverio *delta* duomenų (DELTA-SERVERIS). Formavimo komponentas yra sukurtas klientui, kad atstovautų kliento pusės būsenos pokyčius ir atitinkamus veiksmus, sukeliančius tuos pokyčius, kai pastarasis yra serverio atsakymas, jis yra rezultatas tų veiksmų atliktų su serverio komponentais. Delta bendravimo duomenys gali būti įvairaus formato, tokiuose karkasuose kaip, pavyzdžiui, XML, BXML, *JavaScript* objekto žymėjimas (JSON), *JavaScript*. Kliento *delta* žinutės turi būtiną informaciją serveriui, kad jis žinotų, pavyzdžiui, koks veiksmas, su kuriuo komponentu turi būti atliktas. GWT paslaugų iškvietimui naudoja RPC stilių, o *Backbase* ir *Echo2* komponentais pagrįstas metodas yra įgyvendintas taip, kad naudotų įvykių klausimo elementus.

6.2.2.3. Jungiamieji elementai

Jungiamieji elementai tarnauja kaip klijai, kurie laiko komponentus kartu ir įgalina juos susisiekti tarpusavyje.

Įvykiai formuoja SPIAR modelio sąveikos esmę. Įvykis yra pradedamas po kiekvieno naudotojo sąsajoje atlikto veiksmo, kuris perduodamas į variklį.

Priklausomai nuo įvykio tipo, gali būti reikalingas prašymo siuntimas į serverį ar dalinai atnaujinta sąsaja. Jei pageidaujamas įvykis gali būti apdorojamas asinchroniškai, tada valdymas būna nedelsiant grąžintas naudotojui.

Serveryje prašymas, kuris buvo pradėtas įvykio, *iškviečia* paslaugą. Paslauga gali būti ar iškviesta tiesiogiai ar per atitinkamo UI komponentų įvykių klausimo elementus.

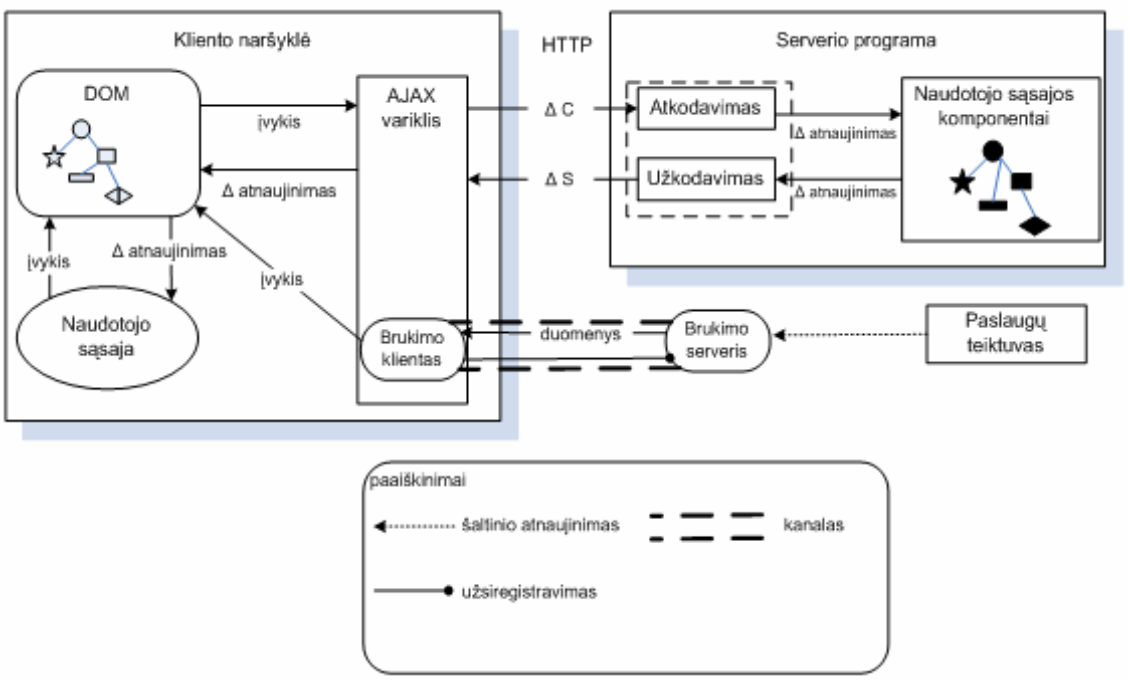
Delta sujungimai yra lengvo svorio komunikacinė terpė sujungianti variklį ir serverį naudojant užklausos/atsakymo mechanizmą, veikiantį per HTTP.

Atnaujintos delta žinutės yra naudojamos, kad atnaujintų atstovaujamąjį modelį kliento pusėje ir komponentų modelį serverio pusėje, kad atspindėtų būsenos pokyčius. Kai atstovaujamojo modelio delta žinutė yra tiesioginio matomo rezultato priežastis, naudotojo sąsajoje, atnaujintas komponentų modelis iškviečia atitinkamus klausimo elementus. Šis modelio atnaujinimas yra paprastai atliekamas naudojant procedūrinius metodų iškvietimus.

6.2.3. SPIAR schemos brėžimas

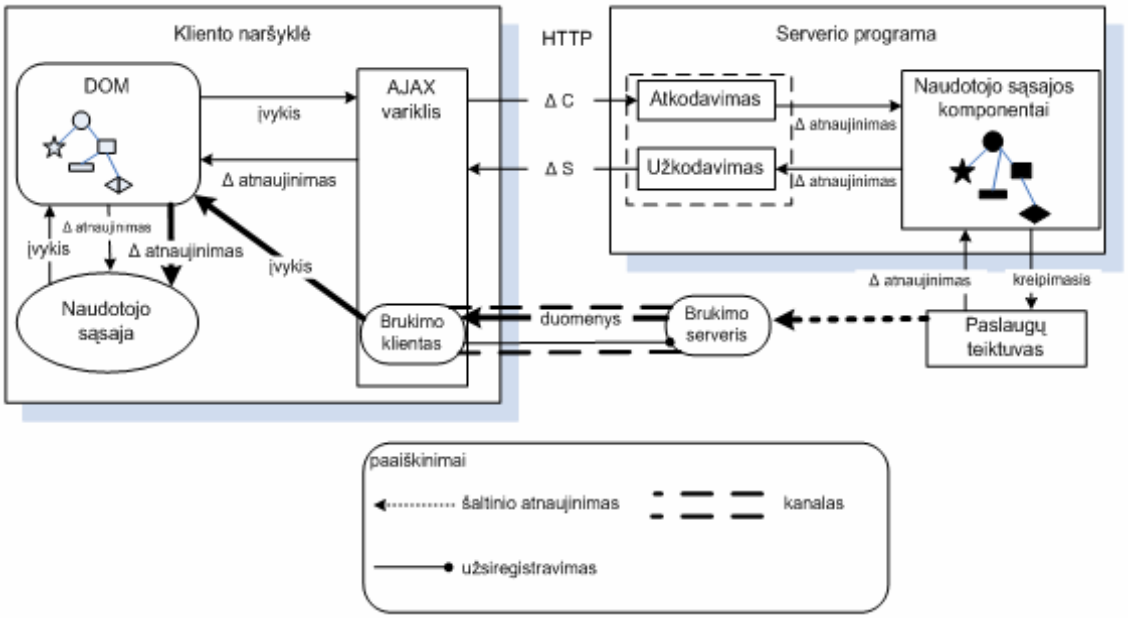
Turint apdorojimo, duomenų ir jungiamuosius elementus, galima naudoti skirtingus architektūrinius vaizdus, kad aprašyti, kaip tie elementai kartu dirba formuodami architektūrą. Toliau bus naudojami du architektūriniai vaizdai, kurie koncentruosis į duomenų srautus ir kelis sujungimų tarp apdorojimo elementų aspektus atsižvelgiant į duomenis. Tokie vaizdai atitinka komponentų ir sujungimo elementų vaizdo tipą.

Daugumos esamų COMET karkasų duomenys yra tiesiogiai stumiami klientui kaip parodyta 11 paveiksle. Šis tiesioginis metodas yra tinkamas įgyvendinimams, kurie nėra pagrįsti komponentais. Vis dėlto, naudotojo sąsajos komponentais pagrįstuose karkasuose, jei stumiami duomenys yra tiesiogiai siunčiami klientui, klientas turi tvarkyti šiuos duomenis pats ir atnaujinti savo komponentus lokaliai. Kad praneštų naudotojo sąsajos komponentams serveryje, klientas turi siųsti klientinį delta pranešimą atgal į serverį. Tai nėra efektyvu, kai brukimo serveris ir programos serveris yra tame pačiame kompiuteryje arba tinkle. SPIAR architektūros stilius atskleidžia įtampą tarp komponentais pagrįstos naudotojo sąsajos ir brukimu pagrįstų apribojimų.



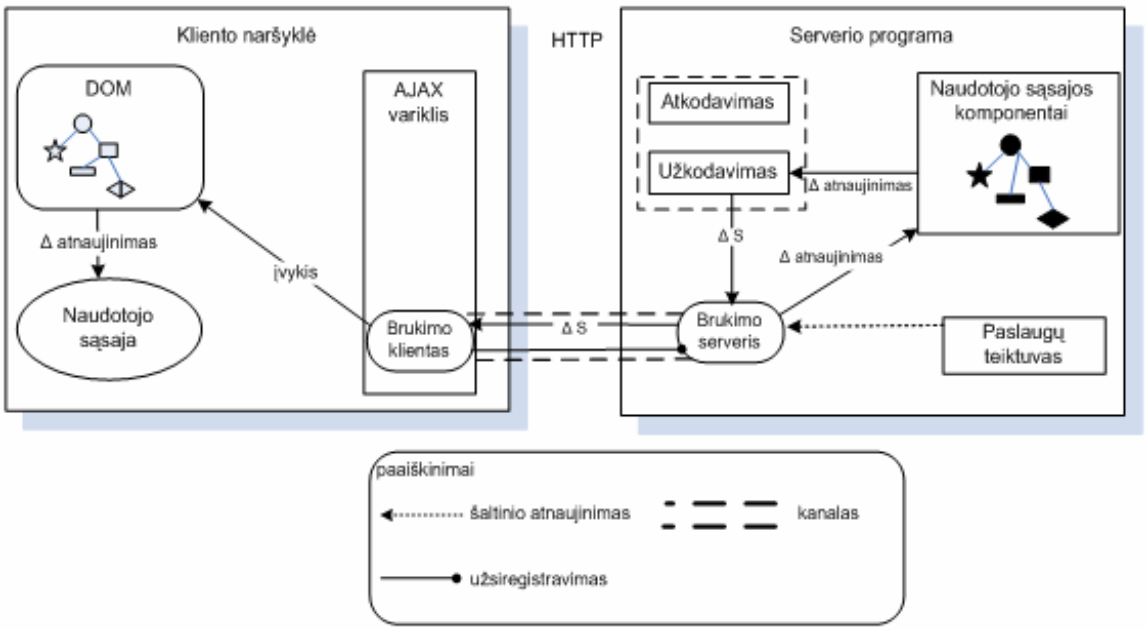
11 pav. COMET karkasuose naudojamu brukimu pagrįstas SPIAR architektūros veikimo vaizdas

Iš 11 paveikslo matyti, kad schema (Mesbah, Deursen 2008) yra neteisingai arba nepilnai nubrėžta, nes naudotojo sąsajos komponentai turi būti atnaujinti paslaugų teiktuvo, o tada perduoti užkodavimo elementui. 11 pav. esanti schema yra netiksli, todėl schemą reikia tobulinti. Patobulintas COMET karkasuose naudojamu brukimu pagrįstas SPIAR architektūros veikimo vaizdas rodomas 12 paveiksle. Brukami duomenys vaizduojami storomis linijomis.



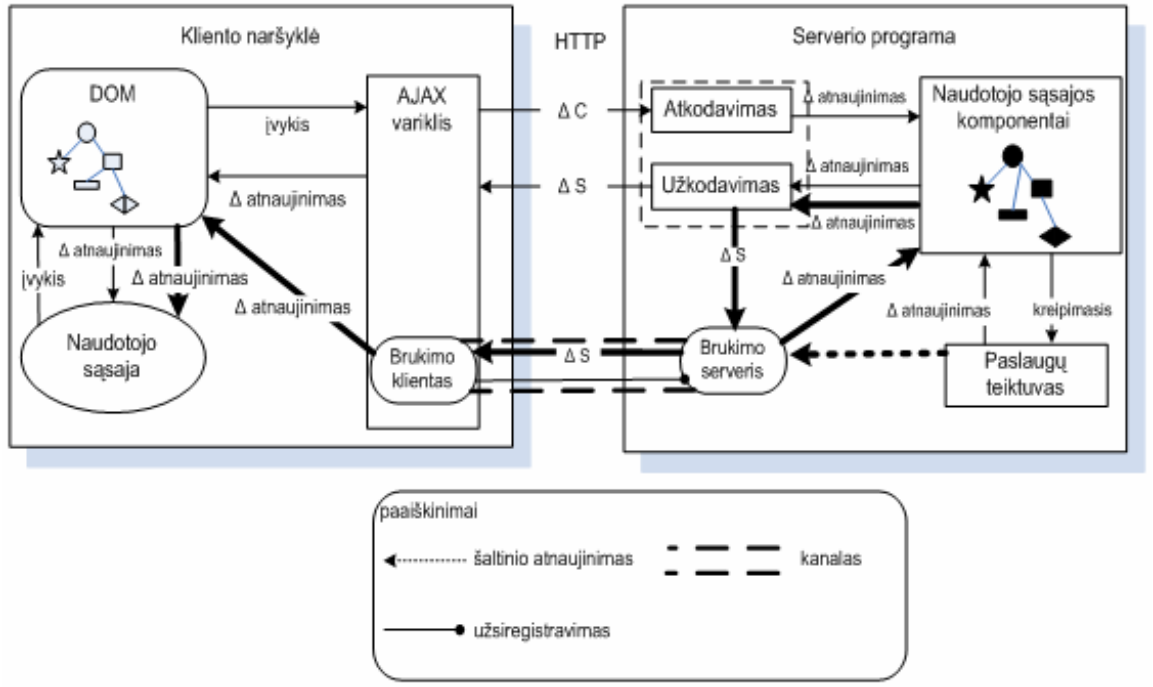
12 pav. Patobulintas COMET karkasuose naudojamu brukimu pagrįstas SPIAR architektūros veikimo vaizdas

Galimas sprendimas gali būti atliktas sutrumpinant sinchronizacijos procesą. Kai įvykis atvyksta iš paslaugos teiktuvo (būsenos pokytis, nauji duomenys), vietoj to, kad nauji duomenys būtų siunčiami tiesiai klientui, pirmiau apie būsenos pokyčius yra pranešama visų užsiregistravusių klientų naudotojo sąsajos (UI) komponentams. Tada pokyčiai yra siunčiami pro užkodavimo įtaisą į brukimo serverį ir tada perduodami, taip kaip siunčia brukimo delta serveris brukimo klientui. Šis metodas įsitikina, kad serverio būsena yra sinchronizuota su kliento būsena kiekvienam persiuntimui. 13 paveikslas rodo geresnės brukimu pagrįstos SPIAR architektūros vaizdą (Mesbah, Deursen 2008). Brukimo klientas užsiregistruoja atitinkamam kanalui per brukimo serverį ir pokyčiai yra perduodami per komponentinį modelį, kaip stumiančio serverio delta pranešimai ir viskas perduodama klientui realiuoju laiku.



13 pav. Brukimu pagrįstas SPIAR architektūros veikimo vaizdas

Iš 13 paveikslo matyti, kad *Mesbah* ir *Deursen* straipsnyje (Mesbah, Deursen 2008) pateiktai SPIAR schemai trūksta rodyklių, dėl to negalima susidaryti pilno architektūros stiliaus vaizdo. Tokios schemos negali iš karto naudoti svetainių kūrėjai, todėl turi būti sukurta nauja, patobulinta schema. Patobulintas brukimu pagrįstas SPIAR architektūros veikimo vaizdas rodomas 14 paveiksle.



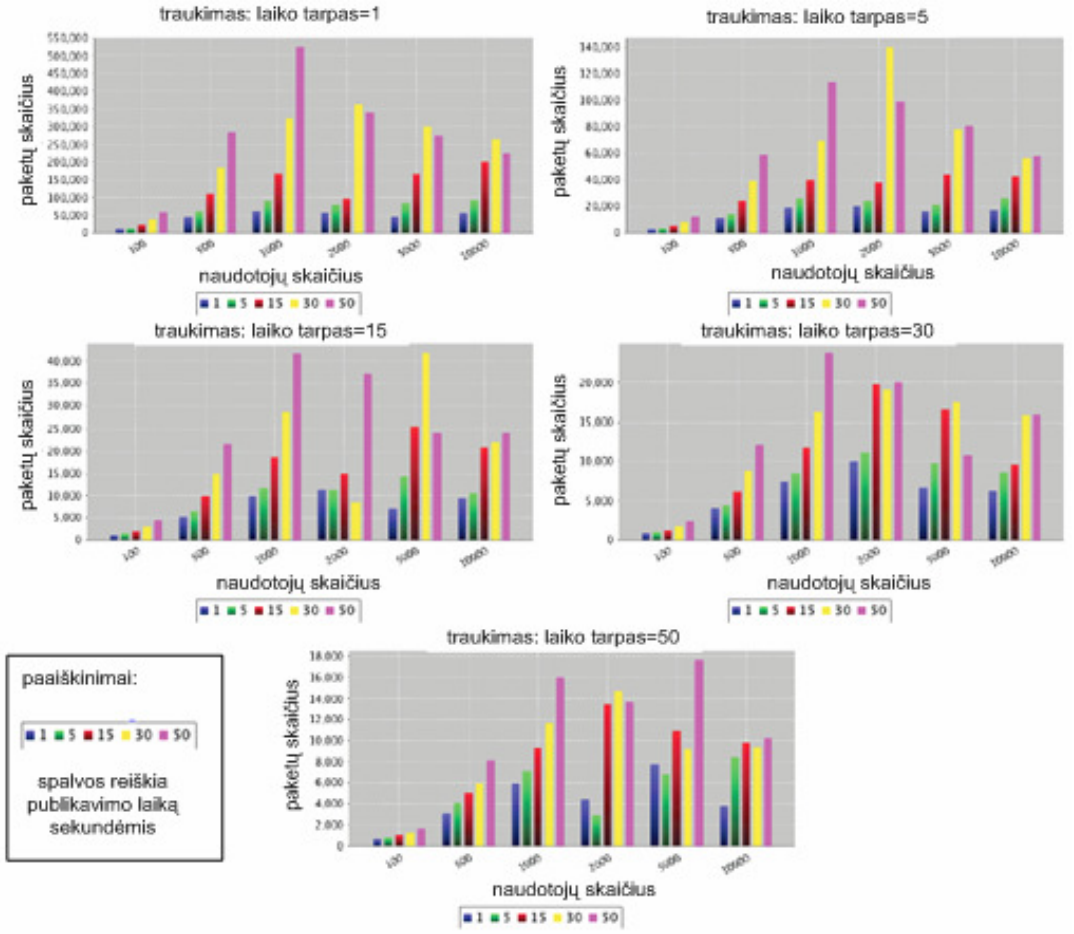
14 pav. Patobulintas brukimu pagrįstas SPIAR architektūros veikimo vaizdas

Iš 14 paveikslo matyti, kad paprastas kliento/serverio bendravimas, tęsiasi nevaržomai (plonos linijos). Tai reiškia, jog šis apibendrintas brukimo modelis ankstesnių teigiamų savybių neprarado. Šis sprendimas turi du privalumus. Pirmiausia jis leidžia ΔS siųsti tiesiogiai naudotojui vienu žingsneliu.

Antras privalumas yra tas, jog palengvinamas programos kūrėjo darbas. Be šio sprendimo programuotojas turi rašyti išorines *JavaSript* funkcijas norėdamas apdoroti ateinančius brukimo duomenis. Pasiūlytame sprendime tokių funkcijų nebereikia, nes atsakymas bus laukiamu ΔS formatu, kuris bus apdorotas AJAX variklio automatiškai.

6.3. Testavimas

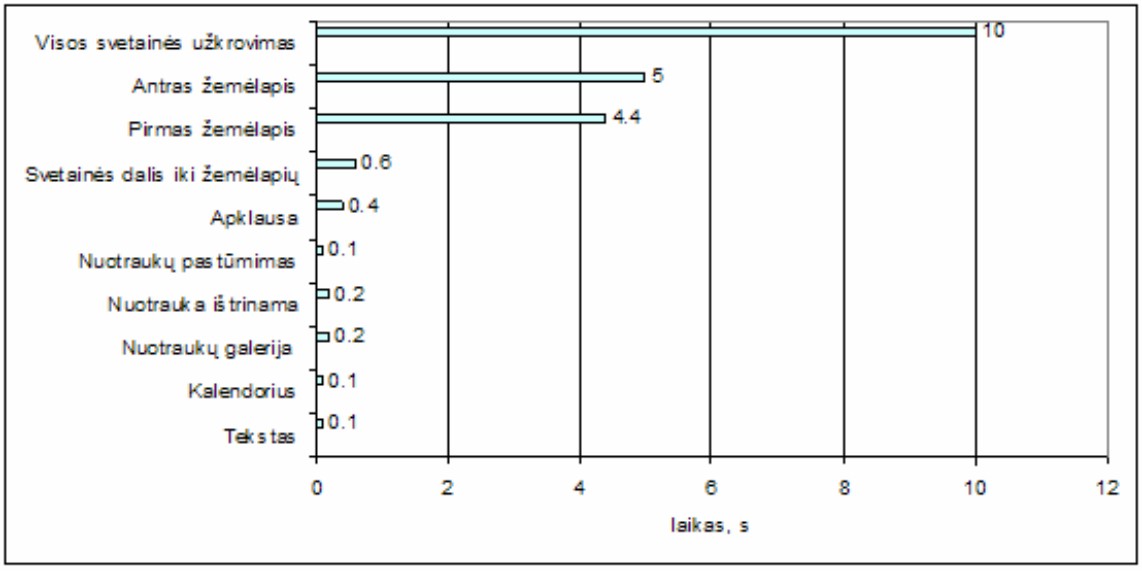
COMET technologija, naudojama SPIAR architektūros stiliuje, veikia dviem būdais: ilgo traukimo metodu ir srautų (brukimo) metodu. Kuris metodas geresnis parodo testai 15 paveiksle. Kiekvieno testo metu yra siunčiama 10 publikavimo žinučių.



15 pav. Testų rezultatai rodantys tinklo našumą

Testų analizė leidžia daryti išvadą, kad brukimo metodas geresnis, kai reikalingas geras duomenų koherentiškumas ir tinklo našumas. Traukimo metodas negali pasiekti tokio pačio duomenų koherentiškumo, net su mažu traukimo laiko tarpu. Brukimo metodas leidžia turėti daug naudotojų. Kai naudotojų skaičius labai padidėja, tikimybė, kad siunčiama žinutė pasieks savo tikslą mažėja (Bozdag, Mesbah 2008). Naudojant traukimo metodą sunku pasiekti gerą duomenų koherentiškumą. Jei traukimo laiko tarpas bus didesnis, nei publikavimo laiko tarpas, tai bus prarandami duomenys. Jei bus priešingai, tada traukimo metodo naudojimas apkraus tinklą septynis kartus labiau, nei brukimo.

Sukurtos svetainės testų rezultatai parodyti 16 paveiksle. Buvo testuojama kaip greitai užkraunama visa svetainė bei jos atskiros dalys. Remiantis tuo galima nustatyti ar naudotojui yra patogu ja naudotis dėl vėlinimo savybės. Svetainė užima 2 MB duomenų. Testavimui naudota *Firebug1.3.3* programa.



16 pav. Sukurtos svetainės ir jos dalių krovimosi laiko testų rezultatai

Iš 16 paveikslo matyti, jog svetainės dalis iki žemėlapių užkraunama per 0.6 sekundės.Toks greitis yra labai geras, nes naudotojai nejaučia vėlinimo. Žemėlapiai kraunasi lėčiau, nes reikia persiųsti daugiau duomenų, tačiau jie yra apatinėje svetainės dalyje ir tos dalies nesimato, kai atidaroma svetainė. Žemėlapių krovimo greitis nedaug skiriasi nuo originalios *Google* svetainės greičio. Dėl visų šių priežasčių galima teigti, jog svetainė labai greitai reaguoja į naudotojo veiksmus, todėl yra interaktyvesnė. Sukurtos internetinės svetainės vaizdai parodyti 17 paveiksle.

17 pav. Sukurtos internetinės svetainės vaizdai

7. SPIAR analizė

Šiame skyriuje bus analizuojamas SPIAR architektūros stilius, tiriant kaip gerai egzistuojantys AJAX karkasai ir tipinės AJAX architektūros yra pritaikomos šiam stiliui Bus aptarti įvairūs sprendimai ir jų daroma įtaka visam AJAX programų dizainui, sprendžiant apie viską pagal architektūros savybes.

SPIAR architektūros stiliaus savybių palyginimas su AJAX savybėmis parodytas 4 lentelėje.

4 lentelė. AJAX ir SPIAR palyginimas

	Naudotojo interaktyvumas	Naudotojui matomas vėlinimas	Tinklo našumas	Paprastumas	Mastelio keičiamumas	Mobilumas	Matomumas	Duomenų koherentiškumas	Patikimumas	Pritaikomumas
AJAX	+ –	+ –	+ –	+ –	+ –	+	–	–	+ –	+ –
SPIAR pagrįsta AJAX	+	+	+	+	+ –	+	–	+	+	+

Iš 4 lentelės matyti, jog AJAX technologijos naudojimas negarantuoja naudotojui ir kūrėjui reikalingų teigiamų savybių gavimo, tik suteikia galimybes joms atsirasti. Reikalingoms savybėms pasiekti reikia aiškaus architektūros stiliaus su iš anksto nustatytomis pageidaujamomis savybėmis ir reikalingais architektūriniais apribojimais. Toks stilius yra SPIAR. Juo remiantis sukurtos tinklo programos ir internetinės svetainės turės visas šiam stiliui būdingas savybes.

7.1. Karkasų modifikavimas pagal SPIAR

Kiekvienas iš karkasų (*Echo2*, GWT, *Backbase*) gali būti SPIAR architektūros pavyzdys, net jei visiškai neatitinka visų SPIAR architektūros apribojimų. Echo2 geriausiai atstovauja SPIAR, nes turi pilnai įvykiais valdomą ir komponentais pagrįstą architektūrą. JSF (*Java* pagrįstas tinklo programų karkasas skirtas supaprastinti naudotojo sąsajų kūrimą *Java EE* programose) pagrįsta *Backbase* architektūra taip pat gerai atspindi SPIAR, net jei JSF ir nėra tikras įvykiais pagrįstas metodas. Iš kitos pusės, GWT yra įdomi architektūra. Nors architektūra naudoja naudotojo sąsajos komponentus kūrimo fazėje, šie komponentai yra įdėti į kliento pusės kodą. GWT nėra tikra serverio pusės komponentais pagrįsta architektūra, todėl galima daryti išvadą, jog ji pilnai neatitinka SPIAR. Nei vienas iš šių trijų karkasų neturi brukimu pagrįstų elementų. Kol brukimu

pagrįsti apribojimai yra gerai pristatyti *Dojo* ir *Cometd* karkasuose, ten visai nėra naudojami komponentai.

SPIAR apibendrina ir sujungia komponentais ir brukimu pagrįstus stilius iš minėtų karkasų ir sukuria naują stilių.

7.2. Tipinės AJAX konfigūracijos

Tinkle daug pramoninių karkasų palaiko Ajax sąveikos stilių. Tačiau, dėl daugybės šių sistemų yra sudėtinga nustatyti jų bendrus bruožus ir nupiešti aiškias linijas tarp sistemų svarbiausių skirtumų. Naudojant SPIAR kaip atspirties tašką, bendrumai ir skirtumai gali būti identifikuoti daug lengviau.

Iš 5 lentelės matyti daug Ajax konfigūracijų kartu su sukurtomis architektūrinėmis ypatybėmis. Pirma eilėje yra REST pagrįsta klasikinė tinklo konfigūracija. Kai ji yra paprasta ir gerai plečiama pagal projektą, ji turi labai blogą atsakomumą, didelį naudotojo suprantamą vėlinimą ir labai didelį tinklu persiunčiamą nereikalingų duomenų kiekį.

Antra konfigūracija yra besiorientuojantis į klientą AJAX. Dauguma AJAX karkasų, pradedami kurti, susitelkiant ties kliento ypatybėmis. Tokie karkasai kaip *Dojo*, *Ext*, ir *jQuery,* visi aprūpina papildančiais naudotojo sąsajos priedais kliento pusėje, palengvindami besiorientuojančio į klientą stiliaus sąveiką, kurioje didžioji dalis funkcionalumo suteikiama naršyklei. Manoma, kad sąveika tarp komponentų, kurie turi tą pačią vietą, turi visai nežymią naudą, lyginant su sąveika esančia tinkle. Šis variantas suteikia gerą naudotojų interaktyvumą ir labai mažą naudotojui suprantamą vėlinimą. Taip pat nėra jokio prisitaikymo palaikymo, kadangi visas kodas yra užkrautas klientui ir tai daro šį variantą statišką išreiškiant kodo pokyčiais atėjusius iš serverio pusės.

Tokie karkasai kaip GWT, DWR, ir JSON-RPC-Java palaiko asinchroninį nuotolinį procedūros iškvietimo (angl. *Asynchronous Remote Procedure Call* (ARPC)) sąveikos stilių. Šioje konfigūracijoje, visas pristatomasis ir funkcinis kodas yra priskirtas naršyklei ir su serveriu asinchroniškai susisiekia tiktai tuo atveju, kai reikia naujų tekstinių duomenų pakeitimo. Mažas naudotojo suprantamas vėlinimas, geras naudotojų interaktyvumas ir sumažintas duomenų iš serverio kelionių skaičius, pirmyn ir atgal, yra šios konfigūracijos savybės. Nepaisant to, kad tekstiniai duomenys gali būti dinamiškai kviečiami iš serverio, pristatomasis ir funkcinis kodas yra riboto pritaikomumo.

5 lentelė. AJAX konfigūracijos ir savybės

	Naudotojo interaktyvumas	Naudotojui matomas vėlinimas	Tinklo našumas	Paprastumas	Mastelio keičiamumas	Mobilumas	Matomumas	Duomenų koherentiškumas	Patikimumas	Pritaikomumas
REST pagrįstas klasikinis tinklas	–	–	–	+	+	+	+	–	+ –	–
Į klientą besiorientuojanti AJAX	+	+		–		+				–
ARPC AJAX	+	+	+	+			–			–
Brukimu pagrįsta AJAX		+	+		–		–	+		+
SPIAR pagrįsta AJAX	+	+	+	+	+ –	+	–	+	+	+

Ketvirta konfigūracija yra tikra brukimu pagrįsta sąveika, kurioje būsenos pokyčiai yra vienu srautu perduodami klientui (išlaikant jungtį) be jokio išorinio kliento prašymo. Geras duomenų koherentiškumo ir pagerintas tinklo našumo lygis, palyginus su tradiciniu traukimo stiliumi, yra svarbiausios teigiamos šio varianto ypatybės. Didelis serverio apkrovimas ir plėtimo problemos atsiranda daugiausiai dėl to, kad serveris turi nuolatos laikyti klientų būsenų informaciją ir atitinkamas jungtis.

Palyginimui, paskutinė konfigūracija 5 lentelėje rodo patį komponentais ir brukimu pagrįstą SPIAR stilių.

7.3. Su AJAX brukimo stiliumi susiję klausimai

Plečiamumas yra svarbiausias svarstomas klausimas brukimo modelyje kartu su tradiciniu serverio modeliu. COMET technologija naudoja nuolatinius sujungimus, tokiu būdu TCP ryšys tarp serverio ir kliento yra išlaikomas gyvas iki išorinio atsijungimo, pertraukos ar tinklo klaidos. Tokiu būdu serveris turi susidoroti su daugeliu sujungimų, jeigu įvykis atsiranda retai, nes serveriui reikia turėti vieną ar daugiau gijų kiekvienam klientui. Tai sukurs problemas matuojamas tūkstančiais vienu metu prisijungusių naudotojų. Todėl yra poreikis geresnių, įvykiais pagrįstų priemonių, esančių serverio pusėje. Paskutiniais tyrimais nustatyta, kad brukimo metodas gali susitvarkyti su didesniu klientų skaičiumi, jei nauja technika, tokia kaip tęsimo (angl. *continuations*) mechanizmas, bus priimta naudoti serverio programose. Tačiau, kai naudotojų skaičius padidėja, žinučių gavimo patikimumas mažėja.

Empirinio studijavimo rezultatai rodo, kad brukimas suteikia gerą duomenų koherentiškumą ir gerą tinklo veikimą, bet tuo pačiu metu COMET serverio programa naudoja daugiau centrinio procesoriaus ciklų, kaip ir traukimo metode.

Būsenas naudojantis serveris yra atsparesnis nesėkmėms, todėl, kad serveris gali išsaugoti būseną bet kuriuo laiku ir atgaivinti ją, kai tik klientas sugrįžta. Brukimo modelis, dėl jo prenumeratorių sąrašo yra mažiau atsparus nesėkmėms. Serveris turi išlaikyti tą pačią būseną tol, kol būsena pasikeis, tik tada bus transliuojami būtini atnaujinti duomenys. Populiariems duomenų elementams naudoti turi būti palaikoma daug būsenų. Šios papildomos sąnaudos, palaikant būseną ir prenumeratorių sąrašą, turės neigiamą efektą plečiamumui (mastelio didinimui).

Plečiamumo problemas taip pat paveldi ir SPIAR technologija, kaip parodyta 4 lentelėje.

7.4. SPIAR apimtis

Būtini reikalavimai AJAX programai yra vykdymo greitis ir pagerinta naudotojų patirtis, nedidelis kliento/serverio duomenų dydis ir labai specifinis sąveikos elgesys. SPIAR yra koordinuotas architektūrinių apribojimų komplektas, kuris bando sumažinti naudotojui suprantamą vėlinimą ir tinklo naudojimą bei pagerinti duomenų koherentiškumą ir naudotojų patirtį. Dėl šių ypatybių AJAX karkasų komponentai yra standžiai sujungti, o laisvas sujungimas nėra viena iš savybių, įtrauktų į SPIAR. Šis būdingas standus sujungimas apima plečiamumo problemas.

SPIAR stilius susitelkia ties naujos rūšies tinklo programų priekine dalimi, t.y. *paslaugų teiktuvas* yra abstraktus komponentas, kuris gali būti sudarytas iš vidurinės arba galinės dalies programinės įrangos. Į paslaugą orientuotos architektūros sprendimas gali lengvai būti sujungtas su SPIAR. Pavyzdžiui, pakeičiant *paslaugų teiktuvą* į SOAP (paprastas objektų pasiekimo protokolas) serverį. SPIAR detalizuoja tiktai tas architektūros dalis, kurios būtinos Ajax interaktyvumui.

Išvados

1. Palyginus internetinių svetainių kūrimo technologijas, buvo nustatyta, jog perspektyviausia technologija dėl atvirų standartų ir patogaus naudojimo yra AJAX. Ištyrus AJAX technologiją paaiškėjo, kad pagrindinės jos problemos yra: aiškių kūrimo taisyklių reikalingoms savybėms pasiekti nebuvimas ir HTML4 protokolo apribojimas, neleidžiantis internetinėms svetainėms prilygti savo galimybėmis darbalaukio programai.
2. Šio apribojimo esmė yra ta, jog serveris negali siųsti naudotojui įvykių ar kitų duomenų jam nepaprašius. Šios problemos išspręstos sukūrus ir patobulinus SPIAR architektūros stilių. Šis stilius, dėl atrinktų apribojimų leidžia svetainių kūrėjams kurti naudojantis aiškiomis taisyklėmis, kurios garantuoja geriausias pageidaujamas savybes jų sukurtose svetainėse. Į SPIAR architektūros stilių buvo integruota COMET technologija, kuri išsprendė antrą pagrindinę AJAX problemą. Atsirado galimybė realiuoju laiku serveriui siųsti įvykius klientui. Netolimoje ateityje COMET technologijos nereikės, nes HTML5 protokolas bus tobulesnis.
3. Šiame darbe buvo praplėsta teorinė medžiaga, susijusi su SPIAR architektūros stiliumi ir AJAX technologija. Jos tapo išsamiau paaiškintos, atliktas SPIAR ir AJAX technologijų palyginimas, įrodantis SPIAR poreikį.
4. Patobulintos SPIAR veikimo schemos ir palygintos su realiųjų karkasų architektūrų schemomis. Architektūros stilius tapo mažiausiai du kartus aiškesnis ir išsamesnis svetainių kūrėjui. Tai ateityje leis šiai technologijai greičiau populiarėti tarp kūrėjų, nes ji pasidarė suprantamesnė ir greičiau įgyvendinama. AJAX technologijos sėkmė, naudojant SPIAR architektūros stilių, yra neabejotina, nes šis stilius sukurtoms internetinėms svetainėms suteikia visas svarbiausias savybes reikalingas ne tik naudotojams, bet ir kompanijoms palaikančioms tas svetaines. Visų savybių naudojimas yra aiškiai ir išsamiai pagrįstas darbe.
5. Sukurta internetinė svetainė praktiškai išryškina dalį pagrindinių savybių. Tai suteikia rimtą pagrindą teigti, kad tos savybės SPIAR architektūros stiliuje pasiektos ne veltui. Šiandien rinkos sąlygomis tik naujausiomis ir geriausiomis technologijomis sukurtos svetainės ir tinklo programos gali padėti laimėti konkurencinę kovą tarp kompanijų ir pelnyti viso pasaulio žmonių pripažinimą, suteikiant naudotojui malonesnį, greitesnį ir patogesnį priėjimą prie interneto paslaugų ir informacijos. Tokia technologija yra AJAX su SPIAR architektūros stiliumi. AJAX su patobulinta SPIAR suteikia kūrėjams naujų galimybių kurti svetaines, todėl galima daryti išvadą, jog ateityje šios technologijos pakeis internetą technologiškai ir vizualiai į gerąją pusę bei leis sukurti daug naujų nematytų interneto paslaugų.

Informacijos šaltiniai

1. Babin, L. 2007. Beginning Ajax with PHP: From Novice to Professional, Apress : 18-139.
2. Baila, S.; Beltran, V. 2007. Web Push, Execution Environments for Distributed Computation Issues 4. [žiūrėta 2009 gegužės 10 d.]. Prieiga per internetą: <http://gsi.ac.upc.edu/reports/2007/71/Book.pdf>: 105-124.
3. Bouras, C.; Konidaris, A. 2005. Estimating and eliminating redundant data transfers over the Web: a fragment based approach: Research articles. Int. J. Commun. Syst., 18(2): 119-142.
4. Bozdag, E.; Mesbah, A.; Deursen, A. 2008. Performance testing of data delivery techniques for AJAX applications [žiūrėta 2009 kovo 10 d.]. Prieiga per internetą: <http://swerl.tudelft.nl/bin/view/Main/TechnicalReports>.
5. Bozdag E.; Mesbah A.; Deursen A. 2007. A Comparison of Push and Pull Techniques for AJAX [žiūrėta 2009 kovo 10 d.]. Prieiga per internetą: <http://swerl.tudelft.nl/bin/view/Main/TechnicalReports>.
6. Brodie, D.; Gupta, A.; Shi, W. 2008. Accelerating dynamic web content delivery using keyword-based fragment detection. J. Web Eng., 4(1): 079-099, 2005.
7. Crane, D.; Pascarello, E.; James, D. 2006. AJAX in action, Manning Publications Co.: 112-113.
8. Challenger, J.; Dantzig, P.; Iyengar, A.; Witting. K. 2005. A Fragment based approach for efficiently creating dynamic Web content, ACM Trans. Inter. Tech., 5(2): 359-389.
9. Garrett, J. 2005. Ajax: A new approach to web applications. [žiūrėta 2009 kovo 10 d.]. Prieiga per internetą: <http://www.adaptivepath.com/publications/essays/archives/000385.php>.
10. *Google* žemėlapių API svetainė. 2009. [žiūrėta 2009 gegužės 9 d.]. Prieiga per internetą: <http://code.google.com/apis/maps/documentation/index.html>.
11. HTML5 protokolo internetinė svetainė. 2009. Server sent events specification. [žiūrėta 2009 kovo 10 d.]. Prieiga per internetą: <http://www.whatwg.org/specs/web-apps/current-work/#server-sent-events>.
12. HTTP srautų šablono svetainė. 2009. [žiūrėta 2009 kovo 10 d.]. Prieiga per internetą: <http://ajaxpatterns.org/HTTP_Streaming>.
13. Hadlock, K. 2006. Ajax for Web Application Developers, Sams: 2-3.

14. Khare, R. 2005. Beyond Ajax: Accelerating web applications with Real-Time event notification. [žiūrėta 2009 gegužės 25 d.]. Prieiga per internetą: <http://www.webbuyersguide.com/resource/white-paper/1672/Beyond-AJAX-Accelerating-Web-Applications-with-Real-Time-Event-Notification>.
15. Khare, R.; Taylor, R. 2004. Extending the representational state transfer (REST) architectural style for decentralized systems, ICSE '04: Proceedings of the 26th International Conference on Software Engineering: 428-437.
16. Mesbah, A.; Deursen, A. 2008. A component- and push-based architectural style for AJAX applications, Journal of Systems and Software 81(12): 2194-2209.
17. Mesbah, A.; Deursen, A. 2006. An architectural style for AJAX, Proceedings of the Sixth Working IEEE/IFIP Conference on Software Architecture: 9-10.
18. Mesbah, A.; Deursen, A. 2007. Migrating multi-page web applications to single-page Ajax interfaces, 11th European Conference on Software Maintenance and Reengineering (CSMR'07): 181-190.
19. Mahemoff, M. 2006. Ajax Design Patterns, O'Reilly : 15-16.
20. Naaman, M.; Garcia-Molina, H.; Paepcke. A. 2004. Evaluation of ESI and class-based delta encoding, 8th International Workshop Web content caching and distribution: 323-343.
21. Offutt, J. 2002. Quality attributes of web software applications, IEEE Software, 19(2): 25-32.
22. Parsons, D. 2007. Evolving architectural patterns for web applications, 11th Pacific Asia Conference on Information Systems (PACIS): 120-126.
23. Russell, A. 2006. AJAX: Low Latency Data to and From the Browser. [žiūrėta 2009 gegužės 25 d.]. Prieiga per internetą: <http://www.avantbard.com/docs/etech2006-/after_ajax_low_latency_data_to_and_from_the_browser.txt>.
24. Russell, A.; Wilkins, G.; Davis, D. 2007. Bayeux – a JSON protocol for publish/subscribe event delivery protocol 0.1draft1. [žiūrėta 2009 gegužės 24 d.]. Prieiga per internetą: < http://svn.cometd.org/trunk/bayeux/bayeux.html>.
25. Suryanarayana, G.; Erenkrantz, J.; Hendrickson, S.; Taylor, R. 2004. PACE: An architectural style for trust management in decentralized applications, 4th Working IEEE/IFIP Conference on Software Architecture (WICSA'04): 221.
26. Stamey, J.; Richardson, T. 2006. Middleware development with ajax, J. Comput. Small Coll., 22(2): 281-287.

27. W3C techninės architektūros grupės svetainė. 2004. Architecture of the World Wide Web, (1). [žiūrėta 2009 gegužės 24 d.]. Prieiga per internetą: <http://www.w3.org/TR/webarch/>.
28. *Wikipedia* puslapis apie COMET. [žiūrėta 2009 gegužės 24 d.]. Prieiga per internetą: < http://en.wikipedia.org/wiki/Comet_(programming)>.
29. Weiss, A. 2005. Webos: say goodbye to desktop applications, netWorker, 9(4): 18-26.
30. Woychowsky, E. 2006. Ajax: Creating Web Pages with Asynchronous JavaScript and XML, Prentice Hall: 7-8.